zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

September 4, 2024

Abstract

The universal composability (UC) framework is a "gold standard" for security in cryptography. UC-secure protocols achieve strong security guarantees against powerful adaptive adversaries, and retain these guarantees when used as part of larger protocols. Zero knowledge succinct non-interactive arguments of knowledge (zkSNARKs) are a popular cryptographic primitive that are often used within larger protocols deployed in dynamic environments, and so UC-security is a highly desirable, if not necessary, goal.

In this paper we prove that there exist zkSNARKs in the random oracle model (ROM) that unconditionally achieve UC-security. Here, "unconditionally" means that security holds against adversaries that make a bounded number of queries to the random oracle, but are otherwise computationally unbounded.

Prior work studying UC-security for zkSNARKs obtains transformations that rely on computational assumptions and, in many cases, lose most of the succinctness property of the zkSNARK. Moreover, these transformations make the resulting zkSNARK more expensive and complicated.

In contrast, we prove that widely used zkSNARKs in the ROM are UC-secure without modifications. We prove that the Micali construction, which is the canonical construction of a zkSNARK, is UC-secure. Moreover, we prove that the BCS construction, which many zkSNARKs deployed in practice are based on, is UC-secure. Our results confirm the intuition that these natural zkSNARKs do not need to be augmented to achieve UC-security, and give confidence that their use in larger real-world systems is secure.

Keywords: succinct arguments; random oracle model; universal composability

Contents

1 Introduction **3**
1.1 Our results . . . 4
1.2 Related work . . . 5

2 Techniques **7**
2.1 Unconditional UC-security . . . 7
2.2 UC-friendly properties . . . 8
2.3 The Merkle commitment scheme is UC-friendly . . . 12
2.4 The Micali construction is UC-secure . . . 13
2.5 The BCS construction is UC-secure . . . 15
2.6 Adaptive corruptions and strong UC-friendly properties . . . 16

3 Preliminaries **20**
3.1 Notation . . . 20
3.2 UC-security with unbounded adversaries . . . 21
3.3 Global random oracle . . . 24

4 UC-security for non-interactive arguments in the ROM **26**
4.1 Ideal functionality . . . 26
4.2 Protocol . . . 28

5 UC-friendly security notions for non-interactive arguments **29**
5.1 UC-friendly completeness . . . 29
5.2 UC-friendly zero knowledge . . . 32
5.3 UC-friendly knowledge soundness . . . 36

6 UC-secure zkSNARKs from UC-friendly security notions **40**
6.1 Proof of Theorem 6.1 . . . 41
6.2 Definitions 5.3, 5.10 and 5.16 are necessary . . . 43

7 Merkle commitments and UC-security **46**
7.1 Merkle commitment schemes . . . 46
7.2 UC-friendly completeness . . . 47
7.3 UC-friendly hiding . . . 47
7.4 UC-friendly extraction . . . 53

8 The Micali construction is UC-secure **58**
8.1 Probabilistically checkable proofs . . . 58
8.2 The Micali construction . . . 59
8.3 UC-friendly completeness . . . 60
8.4 UC-friendly zero knowledge . . . 61
8.5 UC-friendly knowledge soundness . . . 63
8.6 UC-secure zkSNARKs from Micali . . . 70

9 The BCS construction is UC-secure **71**
9.1 Interactive oracle proofs . . . 71
9.2 The BCS construction . . . 73
9.3 UC-friendly completeness . . . 73
9.4 UC-friendly zero knowledge . . . 74
9.5 UC-friendly knowledge soundness . . . 76
9.6 UC-secure zkSNARKs from BCS . . . 85

A An analysis of [IW14] **86**

Acknowledgments **90**

References **90**

1 Introduction

The universal composability (UC) framework [Can01] is a "gold standard" for security in cryptography. UC-secure protocols achieve strong security guarantees in the presence of powerful adaptive adversaries, and retain their security when used as part of larger protocols, thereby enabling a modular analysis of these larger protocols. Informally, security in the UC framework is shown by arguing that an adversary (the environment) cannot distinguish between a real execution of the protocol and an "ideal" execution, where the protocol is replaced by an ideal functionality. In a larger protocol then one can argue, via a result known as the composition theorem, that instances of the former protocol can be replaced by this ideal functionality.

Zero knowledge succinct non-interactive arguments of knowledge (zkSNARKs) are a powerful cryptographic primitive that has seen widespread adoption. zkSNARKs are often used within larger protocols deployed in dynamic environments, and so UC-security is a highly desirable (if not necessary) goal.

Achieving UC-security for a zkSNARK is challenging. Security of a zkSNARK is often established via techniques that are problematic, and at times impossible, to use in the UC framework. These techniques include non-black-box extraction and black-box rewinding extraction. In contrast, UC-security prescribes a black-box security proof in a game consisting of polynomially-many interactions with the adversary, and such security proofs are almost exclusively achieved through the use of straightline (non-rewinding) extractors.

UC-security has been studied in the zkSNARK literature, via transformations that "lift" a given zkSNARK into a UC-secure non-interactive argument. In most cases the transformation *increases the argument size to linear in the witness* (of the proved nondeterministic computation) [KZM+15; ARS20; BS21; AGRS24]; the result is a non-interactive argument that is not succinct in the usual desirable sense (the argument size is succinct in the circuit size but not the witness size). One exception is [GKOPTT23], which achieves UC-secure zkSNARKs by combining a simulation-extractable zkSNARK and a straightline-extractable polynomial commitment scheme. A downside is that this transformation incurs computational overheads, and the resulting zkSNARKs do not reflect ones used in practice. We elaborate further on prior work in Section 1.2. Overall, the takeaway is that the desirable goal of UC-secure zkSNARKs has been notably elusive and the known results come with considerable limitations or caveats.

UC-security with random oracles. The focus of this paper is zkSNARKs constructed in the "pure" random oracle model (ROM), where (honest and malicious) parties have query access to a random function and where security holds unconditionally against adversaries that query the random function a bounded number of times.

The ROM is notable for multiple reasons. The elegant Micali construction [Mic00], the "canonical" construction of a zkSNARK, is realized in the ROM. Moreover, many zkSNARKs used in practice follow the BCS construction [BCS16], which is also realized in the ROM.[1] Both constructions are secure in the quantum ROM [CMS19]; in fact, the ROM supports the most efficient post-quantum zkSNARKs to date. Yet, the UC-security of these seminal zkSNARK constructions has, surprisingly, not been investigated so far.

In the context of UC-security, several basic questions arise.

Do zkSNARKs that are (unconditionally) UC-secure in the ROM exist?
Is the Micali construction UC-secure? What about the BCS construction?
More generally, when does a given zkSNARK in the ROM achieve UC-security?

In this paper we investigate these questions. This requires specifying what is meant by "UC-secure in the ROM". Briefly, this involves specifying an ideal functionality GRO that models a **global random oracle model** (GROM). There are several flavors of GROM [CDGLN18]; the most relevant to our setting

[1]In practice the random oracle is heuristically instantiated via a suitable cryptographic hash function. This leads to zkSNARKs that are lightweight (no public-key cryptography is used) and easy to deploy (users only need to agree on which hash function to use).

is the GROM that is *observable* and *(restricted) programmable*. Establishing UC-security then demands arguing, in a hybrid model in which every party has access to GRO, that an adversary cannot distinguish between two cases: (i) a real execution of the given zkSNARK protocol; and (ii) an ideal functionality $\mathcal{F}_{\mathrm{ARG}}$ for zero knowledge non-interactive arguments of knowledge (which equals the ideal functionality in [LR22b], therein called NIZKPoK ideal functionality). Using techniques from UC with Global Subroutines (UCGS) [BCHTZ20] we then lift the hybrid-model analysis to achieve security in the plain UC framework.

1.1 Our results

We prove that there exist zkSNARKs that unconditionally achieve UC-security in the GROM, positively answering a basic question about the feasibility of UC-secure zkSNARKs in the information-theoretic setting of random oracles. In fact, we prove something stronger (and far more useful), namely, we prove that two seminal constructions of zkSNARKs with random oracles are UC-secure: the Micali construction and the BCS construction. (In particular, we do not construct new zkSNARKs or modify existing ones.) This provides formal evidence that supports the intuition that these seminal constructions of zkSNARKs satisfy far stronger security properties than previously shown, and are suitable for secure use within larger protocols.

Definition 1.1 (informal)**.** *Let $\mathcal{F}_{\mathrm{ARG}}$ be the non-interactive argument ideal functionality in [LR22b] (therein called NIZKPoK ideal functionality), and let GRO be the ideal functionality for the (observable and restricted programmable) GROM in [CDGLN18]. A zkSNARK* **unconditionally achieves UC-security in the GROM** *if the zkSNARK unconditionally UC-realizes $\mathcal{F}_{\mathrm{ARG}}$ in the GRO-hybrid model. ("Unconditionally" means that security holds against adversaries that are computationally unbounded and that make a bounded number of queries to the ideal functionality GRO.)*

Theorem 1.2 (informal)**.** *There exists a zkSNARK that unconditionally achieves UC-security in the GROM.*

The above result follows from the following theorem. Recall that the Micali construction compiles a given PCP (probabilistically checkable proof) with suitable properties into a zkSNARK, and the BCS construction compiles a given public-coin IOP (interactive oracle proof) with suitable properties into a zkSNARK.

Theorem 1.3 (informal)**.**

- *The Micali construction unconditionally achieves UC-security in the GROM, provided that the underlying PCP is honest-verifier zero knowledge and knowledge sound.*
- *The BCS construction unconditionally achieves UC-security in the GROM, provided that the underlying IOP is honest-verifier zero knowledge and (state-restoration) knowledge sound with a straightline extractor.*

The properties required of the underlying PCP and IOP for UC-security in Theorem 1.3 are essentially the same as those typically used in the Micali and BCS constructions.[2] We only additionally require the extractor of the IOP to be straightline, a property satisfied by most IOPs in the literature.

As we elaborate further in Section 2, our results are achieved by showing that the given non-interactive argument satisfies certain "UC-friendly" notions of completeness, zero knowledge, and knowledge soundness in the ROM, which in turn we show imply UC-security in the GROM.

Achieving UC-security is a notoriously challenging goal, even for simple cryptographic protocols. As we outline in Section 2, establishing UC-security of the Micali construction is distinctly more involved compared

[2]State-restoration knowledge soundness is a natural strengthening of knowledge soundness that is required for the security of the BCS transformation. See [BCS16; CY24] for more details.

to merely establishing its standalone knowledge soundness or zero knowledge (as done in prior work). Even more involved is establishing the UC-security of the BCS construction, which is used in practice.

Adaptive security. Our results also cover the *adaptive flavor* of UC-security, where the adversary can corrupt parties in the protocol at any time (rather than only at the start of the protocol). This stronger, and more realistic, flavor of UC-security demands additional work both in terms of definitions and analyses.

Concrete security bounds. Throughout our work we provide concrete security bounds, parametrized on security parameters and the capabilities of the adversary (e.g., queries to the global random oracle). This ultimately leads to explicit expressions for the UC-security error of the zkSNARKs that we study. Similarly to the ROM, the GROM can be (heuristically) instantiated via a suitable cryptographic hash function, and these expressions enable practitioners to set parameters for the desired security level for UC-security.

1.2 Related work

We provide references for the model of global random oracle that we use. Then we summarize prior work studying UC-security for non-interactive arguments that are not succinct and for those that are succinct.

Global random oracle. The random oracle model is widely used to analyze the security of cryptographic protocols. The generalized UC (GUC) framework in [CDPW07] extends the basic UC framework in [Can01] to allow for *globally shared* ideal functionalities, such as a global random oracle. Subsequently, [BCHTZ20] identifies a subtle inconsistency in the GUC formulation, and shows a mechanism to model and prove the security of protocols interacting with shared functionalities in the *plain* UC model; this is the framework of UC with Global Subroutines (UCGS) that we use to accommodate for a random oracle functionality. There are multiple flavors of a *global random oracle model* (GROM) in the UC framework: [CJS14] propose a GROM where queries can be observed, but not programmed, by the adversary; and [CDGLN18] introduce a GROM where queries can be observed as well as programmed by the adversary (with some restrictions). We use the latter flavor in this paper (see Section 3.3), since it is usually appropriate for constructions in the "pure" ROM (with no cryptography). For example, the simple commitment scheme $f((m, r))$, where m is a message and r a random salt, can be shown to be UC-secure in the latter GROM flavor, but not in the former.

Non-Succinct zkNARKs. Several works study UC-security for zero knowledge non-interactive arguments of knowledge (zkNARKs) that are *not* succinct (the size of the argument string is at least the size of the witness for the proved nondeterministic computation).

- *From game-based simulation-secure knowledge soundness.* [Gro06] achieves UC-secure zero-knowledge proofs in the CRS model (assuming cryptographic hardness assumptions), using the observation that straightline knowledge extraction that is secure in the presence of a simulation oracle is crucial for UC-security. The proof size in [Gro06] is linear in the circuit size. In this work we also rely on game-based notions of simulation-secure straightline knowledge soundness (in the ROM setting).

- *Encrypt the witness.* A standard approach to achieve UC-security is to have the argument string include an encryption of the witness and a zero knowledge proof that the encrypted message is a valid witness [DDOPS01]. This approach is adopted in various works studying UC-security in the zkSNARK community, including the C∅C∅ framework [KZM+15], LAMASSU [ARS20], TIRAMISU [BS21], and [AGRS24]. All non-interactive arguments following this approach are not succinct since the argument string contains the encryption of a witness. (The argument size can be smaller than the proved circuit but not the witness.)

- *Compile a Σ-protocol.* Other works study UC-security for non-interactive arguments obtained from Σ-protocols: [LR22b] shows that a randomized variant of the Fischlin construction [Fis05; Ks22] applied to a Σ-protocol yields a zkNARK that achieves UC-security in the observable programmable GROM, and with a global reference string the construction can be modified to rely only on an observable GROM; then [LR22a] shows how to extend these results to achieve security against adaptive corruptions, assuming a minor property of the Σ-protocol.

While the constructions studied in [LR22b; LR22a] and in this paper are different (non-interactive arguments obtained from Σ-protocols versus from probabilistic proofs), our work is inspired by the ideas in [LR22b; LR22a]. Specifically, we use "UC-friendly" definitions of completeness, zero knowledge, and knowledge soundness in the ROM that suffice (and are necessary) for UC-security in the GROM, which reduces the goal of UC-security to proving that the relevant zkSNARK constructions satisfy these simpler properties. The definitions that we use (which can be found in Section 5) are variants of those in [LR22b; LR22a], adapted to our pure ROM setting and to facilitate concrete security bounds.

Succinct zkNARKs. [GKOPTT23] construct zkSNARKs that are computationally UC-secure in a model that provides a global reference string and a global random oracle (that is observable but not programmable). Their approach is a compiler that combines any simulation-extractable zkSNARK and a polynomial commitment scheme with certain properties (each comes with its own reference string), leveraging the random oracle to achieve straightline extraction via proof-of-work ideas inspired by [Fis05].[3] Our work is complementary in that we study a setting without any computational assumptions: we achieve unconditional UC-security for well-known zkSNARKs (without modifications) via a global random oracle (that is observable and programmable). Moreover, the zkSNARKs that we consider are not susceptible to quantum attacks whereas the compiler in [GKOPTT23] uses a polynomial commitment scheme that is insecure against quantum attacks (and whether there is a suitable post-quantum replacement is an open question).

[3]Informally, the argument prover, instead of providing an encryption of the witness as in [DDOPS01] (which makes argument strings non-succinct), uses a polynomial commitment scheme to commit to a polynomial whose coefficients are the witness; to achieve straightline extraction, the argument prover also provides a Fischlin-style proof-of-work that requires querying the random oracle on many evaluations of the committed polynomial. The extractor can then use polynomial interpolation to reconstruct the witness from the query-answer trace of a malicious argument prover.

2 Techniques

We outline the main ideas behind our results.

- In Section 2.1 we describe how to adapt the UC-security framework to our setting of unconditional security in the ROM (and with the additional goal of achieving concrete security bounds).
- In Section 2.2 we describe how we reduce UC-security in the GROM to three simpler properties in the ROM: *UC-friendly completeness*; *UC-friendly zero knowledge*; and *UC-friendly knowledge soundness*.
- In Section 2.3 we discuss the Merkle commitment scheme in the ROM (a component of the zkSNARKs that we study), for which we prove several "UC-friendly" properties that we introduce and rely on.
- In Section 2.4 we discuss UC-security of the Micali construction, and then in Section 2.5 we discuss UC-security of the BCS construction. In both cases we do so by showing the above UC-friendly properties.
- In Section 2.6 we discuss how we achieve UC-security against adaptive corruptions.

2.1 Unconditional UC-security

We consider UC-security for protocols in the "pure" ROM, where parties have query access to a random function and where security holds unconditionally against adversaries that query the random function a bounded number of times. This setting is not considered in prior work studying UC-security for zkSNARKs and, more generally, there is no off-the-shelf model of UC-security for this setting. Below we explain how we adapt the UC framework [Can01; Can20] to our needs, and how our goals can be expressed in this adaptation.

UC-security against unbounded adversaries. We consider adversaries that are computationally unbounded, and are limited only in their access to certain resources, such as queries to a random oracle, queries to a prover oracle, and others. As discussed in detail in Section 3.2, we model this setting by modifying the mechanism of import and time budget described in [Can20, Sec 3.2] to work with a generalized notion of budget. We endow the environment (and the protocol) with a budget represented as a numeric vector. Each message sent specifies how much budget is deducted from the sender budget and added to the receiver budget, and the budget can be spent on a certain set of actions. With this, we can define the notion of budget-emulation.

Definition 2.1 (informal). *Let $\mathcal{B}$ be a tuple of non-negative integers. An environment is $\mathcal{B}$-budget if its starting budget is $\mathcal{B}$. A protocol π $\mathcal{B}$-emulates a protocol φ with simulation error σ if π UC-emulates φ with simulation error σ in the presence of any environment that is $\mathcal{B}$-budget.*

GROM and shared functionalities. The analogue of the ROM in our setting is a shared global subroutine: the observable and (restricted) programmable GROM introduced in [CDGLN18]. The GROM interface allows four types of queries: (i) random oracle; (ii) programming; (iii) observation; (iv) and is-programmed. The random oracle query interface is familiar: each query is consistently answered with a random answer. The programming interface enables setting the answer to arbitrary queries, while the is-programmed interface enables parties *in the session* to detect whether a point has been programmed.[4] Finally, the observation interface allows queriers to receive a list of *illegitimate* queries made to the oracle thus far (queries with prefix sid made by the adversary or parties outside the session sid). The programming interface is used to argue zero knowledge, while the observable and is-programmed interfaces are used to argue knowledge soundness.

We use the approach of *UC with Global Subroutines* [BCHTZ20] to argue that UC-security in the presence of a global shared functionality implies standard UC-security. Informally, if the shared functionality and the

[4]Here "in the session" refers to the fact that the environment cannot directly ask is-programmed queries to the GROM, but only through the adversary or a corrupted party. This enables the UC simulator to intercept these queries and choose their answers.

protocols satisfy some mild requirements, then showing UC-emulation in the hybrid model suffices to show (standard) UC-security. See Section 3.2 for more details.

The ARG functionality. We study UC-security for (succinct) non-interactive arguments. The ideal functionality that we use is the *ARG ideal functionality* $\mathcal{F}_{\mathrm{ARG}}$ from [LR22b] (therein called NIZKPoK ideal functionality), given in Section 4.1.[5] Briefly, $\mathcal{F}_{\mathrm{ARG}}$ has a proving interface that produces simulated proofs (to capture zero knowledge) and a verification interface that extracts a witness (to capture knowledge soundness).

Any non-interactive argument ARG in the ROM directly induces a corresponding protocol $\Pi[\mathsf{ARG}]$ in the GROM that matches the proving and verification interface of $\mathcal{F}_{\mathrm{ARG}}$. The protocol $\Pi[\mathsf{ARG}]$, which is described in Section 4.2, consists of two parties, a prover party M_P and a verifier party M_V.

- The prover party M_P, on input an instance-witness pair, runs $\Pi[\mathsf{ARG}]$'s proving interface, which runs ARG's prover using the GROM, and outputs the resulting argument string.
- The verifier party M_V, on input an instance-proof pair, runs $\Pi[\mathsf{ARG}]$'s verification interface, which runs ARG's verifier using the GROM and checks that none of the verifier queries involves programmed points, and outputs the resulting decision bit (or simply rejects if one of the verifier queries was programmed).[6]

We use the generalized budget mechanism to keep track of the resources used by the environment. Since we consider non-interactive arguments in the ROM, security will depend on the number of queries that the environment makes to the GROM; in our setting, these queries include both random oracle queries and programming queries.[7] Moreover, the environment may query the proving and verification interfaces, which can aid an attack; hence we keep track of such queries as well. Overall, a $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$-budget environment is an environment that can make: (1) t_{q} random oracle queries to the GROM; (2) t_{p} programming queries to the GROM; (3) ℓ_{p} prover queries; and (4) ℓ_{v} verifier queries.

The above enables us to state our first result in slightly more detail.

Theorem 2.2 (restatement of Theorem 1.2)**.** *There exists a non-interactive argument* ARG *in the ROM for which the protocol* $\Pi[\mathsf{ARG}]$ $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-emulates the ideal functionality* $\mathcal{F}_{\mathrm{ARG}}$ *with simulation error*

$$\sigma(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) = \frac{\mathsf{poly}(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})}{2^{\lambda}} \quad .$$

We show that natural constructions of zkSNARKs in the ROM suffice for the above theorem: ARG can be the Micali construction or the BCS construction (instantiated over appropriate probabilistic proofs). Moreover, for these constructions we derive explicit expressions for the simulation error $\sigma(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$, which in particular enables setting parameters to achieve concrete UC-security bounds.

Next we describe how we prove such results.

2.2 UC-friendly properties

We informally describe three properties about a non-interactive argument ARG that are sufficient and necessary for (unconditional) UC-security in the GROM:

[5] One could extend the ideal functionality $\mathcal{F}_{\mathrm{ARG}}$ to one that models *preprocessing* non-interactive arguments. Our belief is that all results in this paper straightforwardly extend to this case (we believe that the preprocessing variants of the Micali construction and BCS construction, when based on suitable holographic probabilistic proofs, are unconditionally UC-secure in the GROM).

[6] An honest party does not program the GROM. In contrast, an adversary might instead attempt to produce an argument string accepted by the verification interface by running the zero knowledge simulator of the non-interactive argument (and programming the GROM accordingly). Rejecting argument strings whose verification involves programmed points disallows this.

[7] Observation and is-programmed queries do not affect security bounds. The environment knows its own queries to the random oracle and the points that it has programmed, so it does not need to obtain this information from the GROM. Moreover, observation and is-programmed queries do not change the state of the GROM, and thus do not affect other parties in the execution.

- **UC-friendly completeness** (sketched in Section 2.2.1);
- **UC-friendly zero knowledge** (sketched in Section 2.2.2); and
- **UC-friendly knowledge soundness** (sketched in Section 2.2.3).

These properties are described in detail in Section 5. Intuitively, each property protects against a natural class of attacks against the UC-security of the protocol $\Pi[\mathsf{ARG}]$, which we outline in the corresponding section.

This approach is analogous to the approach taken in [LR22b; LR22a], where the authors rely on somewhat dissimilar security definitions that are sufficient and necessary for UC-security in their setting (NIZKPoKs obtained from Σ-protocols).[8] In particular, the above properties can be viewed as adaptations of their three properties: *overwhelming completeness*; *non-interactive multiple special honest-verifier zero knowledge*; and *non-interactive special simulation soundness*. The main differences in our definitions include: (a) we target unconditional security, while the previous definitions target computational security; and (b) we allow the adversary to additionally program the random oracle (which is necessary in our "pure" ROM setting). The second difference has important ramifications that we discuss further below.

2.2.1 UC-friendly completeness

The ideal functionality $\mathcal{F}_{\mathrm{ARG}}$ that we consider has a verification interface that, to model completeness, accepts any proof that was generated by its proving interface. This might not be the case for the protocol $\Pi[\mathsf{ARG}]$: one attack against UC-security is, for the environment, to invoke the proving interface on inputs that maximize the probability that the resulting proofs are not accepted by the verification interface, which would distinguish the real-world and the ideal-world. UC-friendly completeness bounds the success probability of such an attack.

Definition 2.3 (informal)**.** ARG *has* **UC-friendly completeness with error** ϵ_{ARG} *if every adversary that*

- *queries the random oracle t_{q} times,*
- *programs the random oracle t_{p} times,*
- *requests ℓ_{p} proofs for instances of length at most n, and*
- *requests ℓ_{v} verifications for instance-proof pairs with instances of length at most n*

causes the verification interface to reject a instance-proof pair generated by the honest prover with probability at most $\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$.

One may guess that perfect completeness of the given non-interactive argument ARG implies UC-friendly completeness with zero error. However this is not the case because the verification interface rejects proofs whose verification causes the argument verifier to query points programmed by the adversary. Hence if there are queries by the argument verifier that the adversary can predict (and program in advance) then the adversary can induce a rejection despite the perfect completeness of ARG.

Nevertheless we show that the two natural notions below suffice, together with perfect completeness of the non-interactive argument, to achieve UC-friendly completeness with small error.

Definition 2.4 (informal)**.** ARG *has:*

- **monotone proofs** *if the argument verifier, on input an honestly produced proof, queries the random oracle only at points that have been queried by the honest argument prover that produced that proof; and*
- **unpredictable queries with error** $\epsilon_{\mathbf{P}}$ *if every adversary that queries the random oracle t_{q} times and programs the random oracle t_{p} times cannot produce an instance-witness pair (with instance length at most n) that causes the honest argument prover to query one of the points previously programmed by the adversary with probability more than $\epsilon_{\mathbf{P}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}})$.*

[8] More precisely, [LR22b; LR22a] discuss properties of a compiler for Σ-protocols, but those properties can be straightforwardly defined for the non-interactive argument output by the compiler.

Lemma 2.5 (informal)**.** *A non-interactive argument with perfect completeness, monotone proofs, and unpredictable queries with error $\epsilon_{\mathbf{P}}$ has UC-friendly completeness with error (roughly) $\epsilon_{\mathrm{ARG}} = \ell_{\mathsf{p}} \cdot \epsilon_{\mathbf{P}}$.*

2.2.2 UC-friendly zero knowledge

Definition 2.6 (informal)**.** ARG *has* **UC-friendly zero knowledge with error** ζ_{ARG} *if every adversary that*

- *queries the random oracle t_{q} times,*
- *programs the random oracle t_{p} times, and*
- *requests ℓ_{p} proofs for instances of length at most n*
- *requests ℓ_{v} verifications for instance-proof pairs with instances of length at most n*

cannot distinguish between the game in which the returned proofs are generated by the honest argument prover and the game in which they are generated by the zero knowledge simulator (which can also program the random oracle) with an advantage better than $\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$.

Informally, UC-friendly zero knowledge is a version of adaptive multi-instance zero knowledge wherein the adversary can adaptively program the random oracle.[9] Indeed, every party can program the GROM, so we need a zero knowledge property that accounts for this capability. In the real-world the protocol generates proofs using the honest argument prover and in the ideal-world the ideal functionality generates proofs using a simulator, so UC-friendly zero knowledge bounds the probability that an adversary distinguishes between these two worlds based on this difference.

First, since the adversary can query the random oracle, we show that queries to the verifier do not help the adversary, and thus show that UC-friendly zero knowledge is implied by a simplified notion where this oracle is not present. Next, since the adversary can generate simulated proofs (and thus simulate the proof oracle), we can use a hybrid argument to reduce the case of multiple simulated proofs to the case of a single simulated proof. We rely on these simplifications to more conveniently establish UC-friendly zero knowledge for the Micali construction and the BCS construction.

Lemma 2.7 (informal)**.** *If* ARG *has UC-friendly zero knowledge with error ζ_{ARG} against adversaries that request a single proof and no verifications, then* ARG *has UC-friendly zero knowledge with error (roughly) $\ell_{\mathsf{p}} \cdot \zeta_{\mathrm{ARG}}$ against adversaries that request ℓ_{p} proofs and make ℓ_{v} verifier queries.*

2.2.3 UC-friendly knowledge soundness

Definition 2.8 (informal)**.** ARG *has* **UC-friendly knowledge soundness with error** κ_{ARG} *if there exists a deterministic polynomial-time straightline extractor such that every adversary that*

- *queries the random oracle t_{q} times,*
- *programs the random oracle t_{p} times,*
- *requests ℓ_{p} simulated proofs for instances of length at most n, and*
- *outputs ℓ_{v} instance-proofs pairs with instances of length at most n*

wins with probability at most $\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$. Here "winning" means that one of the instance-proof pairs that the adversary output (a) was for an instance not queried to the simulation oracle, (b) convinces the argument verifier (without querying programmed points), and (c) causes the extractor to fail to extract a valid witness for the instance.

[9]As shown in Section 5.2, UC-friendly zero knowledge is *strictly* stronger: there are non-interactive arguments that are adaptive multi-instance zero knowledge but not UC-friendly zero knowledge.

UC-friendly knowledge soundness can be viewed as a variant of simulation extractability wherein the adversary can adaptively program the random oracle, as allowed by the GROM. Since the difference between the ideal-world verification interface and the real-world counterpart is the additional attempt at extraction on proofs that successfully verify, UC-friendly knowledge soundness upper bounds the probability that an adversary is able to distinguish between the two worlds by outputting proofs on which extraction fails. The protocol (and ideal functionality) rejects proofs whose verification involves points programmed by the environment. This is to disallow the environment from submitting proofs generated using the zero knowledge simulator (and programming accordingly), from which it would be (likely) impossible to extract.

Moreover, while not shown in the above informal definition, UC-friendly knowledge soundness mandates that the extractor be *straightline*: the extractor receives as input the instance, argument string, query-answer trace of the adversary with the oracle (as well as the query-answer trace of the simulator with the oracle),[10] but not the adversary itself; in particular, the extractor cannot rewind the adversary. Straightline extraction is required by the UC-security experiment (in which the ideal functionality also performs straightline extraction).

Similarly to the case of UC-friendly zero knowledge, we generically reduce UC-friendly knowledge soundness to a simpler property, in which the adversary outputs only a single instance-proof pair.

2.2.4 UC-secure zkSNARKs from UC-friendly properties

Lemma 2.9 (informal)**.** *If a non-interactive argument* ARG *satisfies*

- *UC-friendly completeness with error* ϵ_{ARG}*,*
- *UC-friendly zero knowledge with error* ζ_{ARG}*, and*
- *UC-friendly knowledge soundness with error* κ_{ARG}

then the protocol $\Pi[\mathsf{ARG}]$ $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-emulates the ideal functionality* $\mathcal{F}_{\mathrm{ARG}}$ *with simulation error (roughly)*

$$\epsilon_{\mathrm{ARG}} + \zeta_{\mathrm{ARG}} + \kappa_{\mathrm{ARG}} \ .$$

The proof of Lemma 2.9 is given in Section 6, and follows a game-hopping approach in a GRO-hybrid model. We rely on an observation of [CDGLN18] that, in the setting of the restricted programmable GROM, the simulator can program points undetectably. We can then perform three game hops, one for each of our UC-friendly notions. Finally, we lift the result in the GRO-hybrid model to full UC-security by using the UC with Global Subroutines theorem [BCHTZ20].

UC-friendliness is necessary. We show that the UC-friendly properties that we describe are *necessary* for a non-interactive argument ARG in the ROM to unconditionally achieve UC-security. This gives confidence that the UC-friendly properties that we describe are the "right ones" for UC-security in our setting. Moreover, we learn that the upper bound in Lemma 2.9 is almost tight. Specifically, while the upper bound can plausibly be improved in certain cases (e.g., in the Micali and BCS constructions, establishing UC-friendly completeness and UC-friendly zero knowledge involves separately upper bounding overlapping "bad events"), the improvement is limited. Indeed, the necessity of the UC-friendly properties implies that the simulation error of a non-interactive argument ARG is at least $\max\{\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}\} \geq \frac{1}{3} \cdot (\epsilon_{\mathrm{ARG}} + \zeta_{\mathrm{ARG}} + \kappa_{\mathrm{ARG}})$, at most a factor of 3 (i.e., less than 2 bits of security) away from the upper bound in Lemma 2.9.

On tightness. We make an effort, throughout this paper, to obtain concrete security bounds that are relatively tight (e.g., as noted for Lemma 2.9 in the paragraph above). Nevertheless, modest improvements are possible. For example, Lemma 2.7 reduces UC-friendly zero knowledge to a simpler property (where the adversary requests a single proof and no verifications) at a minor but noticeable cost; this cost can be reduced by

[10]More accurately, matching the ideal functionality, the extractor receives a query-answer trace that includes queries performed by the adversary and the simulator but *not* including queries whose answer was previously programmed by the adversary.

directly establishing UC-friendly zero knowledge for the Micali and BCS constructions, avoiding the use of Lemma 2.7. Similarly for UC-friendly knowledge soundness. These choices reflect striking a balance between aiming for good concrete security bounds, and a modular presentation.

2.3 The Merkle commitment scheme is UC-friendly

The Merkle commitment scheme is a key ingredient in the Micali and BCS constructions (the zkSNARKs that we study), where it acts as unconditionally secure vector commitment scheme. In order to show that said constructions satisfy the UC-friendly security notions sketched in Section 2.2, we establish corresponding properties for Merkle commitments. Below we denote by $\mathsf{MT} := \mathsf{MT}[\lambda, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$ the Merkle commitment scheme for messages of length l (a power of 2) with salt size r_{MT}, for a random oracle with output size λ.

2.3.1 Completeness

We formulate notions of monotone proofs and unpredictable queries for vector commitments schemes (in analogy to the notions in Definition 2.4 for ARG), and show that the Merkle commitment scheme satisfies them. This facilitates proving that the Micali and BCS constructions satisfy UC-friendly completeness.

Lemma 2.10. MT *has monotone proofs, and unpredictable queries with error* $\epsilon_{\mathsf{MT}} = t_{\mathsf{p}} \cdot \mathsf{l} \cdot \left(\frac{1}{2^{\mathsf{r}_{\mathsf{MT}}}} + \frac{1}{2^{\lambda}}\right)$.

2.3.2 Hiding

We formulate a notion of UC-friendly hiding for vector commitment schemes, and show that the Merkle commitment scheme satisfies this property. This contributes towards proving UC-friendly zero knowledge for the Micali and BCS constructions.

Definition 2.11 (informal)**.** MT *has* **UC-friendly hiding with error** ζ_{MT} *if every adversary that*
- *queries the random oracle* t_{q} *times,*
- *programs the random oracle* t_{p} *times, and*
- *requests* ℓ_{p} *commitments for messages of size at most* l *and corresponding openings for sets of size at most* q

cannot distinguish between the game in which the returned commitments and openings are real and the game in which they are generated by a simulator (that can also program the random oracle) with an advantage better than $\zeta_{\mathsf{MT}}(\lambda, \mathsf{l}, q, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$.

Lemma 2.12 (informal)**.** MT *has UC-friendly hiding with error (roughly)* $\zeta_{\mathsf{MT}} = \ell_{\mathsf{p}} \cdot q \cdot \mathsf{l} \cdot \frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\mathsf{r}_{\mathsf{MT}}}}$.

The proof of Lemma 2.12 is similar to the hiding proof for the Merkle commitment scheme in the ROM, but adapted to reflect the additional programming capabilities of the adversary.

2.3.3 Extraction

The Merkle commitment scheme in the ROM is known to satisfy strong notions of extraction [BCS16; CY24]. Any adversary that outputs a Merkle commitment and subsequently outputs a valid opening proof must have "known" the opening at commitment time; moreover, this holds even when the adversary outputs multiple commitments and openings at different times. In the definition below we extend extraction to be UC-friendly, considering adversaries that can program the random oracle. We prove that the Merkle commitment scheme satisfies this stronger property.

Definition 2.13 (informal)**.** MT *has* **UC-friendly extraction with error** κ_{MT} *if every adversary that*

- *queries the random oracle t_{q} times,*
- *programs the random oracle t_{p} times,*
- *requests ℓ_{p} simulated commitments for messages of size at most l and corresponding simulated openings for sets of size at most q,*
- *submits n commitments, and*
- *finally outputs k opening proofs for submitted commitments.*

wins with probability at most $\kappa_{\mathsf{MT}}(\lambda, \mathsf{l}, q, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, n, k)$. Here "winning" means to: (i) submit a list of commitments such that the extractor outputs different messages for duplicate elements in the list; or (ii) output opening proofs that verify successfully on whose commitment the extractor outputs inconsistent messages.

Lemma 2.14. MT *has UC-friendly extraction with error (roughly)* $\kappa_{\mathsf{MT}} = \frac{3}{2} \cdot \frac{(t_{\mathsf{q}}+2\ell_{\mathsf{p}}\mathsf{l})^2}{2^\lambda} + \frac{2k(\mathsf{d}+1)\cdot(t_{\mathsf{q}}+2\ell_{\mathsf{p}}\mathsf{l})}{2^\lambda}$.

We do not prove Lemma 2.14; it straightforwardly follows from the extraction property shown in [CY24]. Instead, we prove that the Merkle commitment scheme satisfies an *even stronger* extraction property (i.e., which implies Lemma 2.14) that we use to achieve adaptive security and we discuss later in Section 2.6.3.

Definition 2.13 already incorporates some notions on non-malleability that will be crucial for establishing UC-friendly knowledge soundness of the Micali and BCS constructions. UC-friendly extraction allows the adversary to submit simulated commitments (as those obtained from the simulation oracle), and guarantees that the Merkle commitment scheme extractor outputs consistent messages on those simulated commitments.

2.4 The Micali construction is UC-secure

We show that the Micali construction unconditionally achieves UC-security in the GROM, when instantiated with suitable ingredients. By Lemma 2.9, it suffices to show that the Micali construction satisfies UC-friendly completeness, zero knowledge, and knowledge soundness, which we now discuss in turn. After that, we explain how this leads to a proof of Theorem 1.2.

Review of the Micali construction. A probabilistically checkable proof (PCP) is a proof system in which the prover sends a (long) proof string, which the verifier checks by probabilistically reading a few locations of it. The Micali construction compiles a (suitable) PCP into a zkSNARK, by using the Merkle commitment scheme in the ROM and the Fiat–Shamir transformation with salt size r. We denote this construction as $\mathsf{Micali}[\mathsf{PCP}, \mathsf{r}]$, and sketch it next.

- The argument prover runs the PCP prover, and commits to the resulting PCP string using the Merkle commitment scheme. Then the argument prover queries the random oracle with the instance, the Merkle commitment, and a random r-bit salt, to obtain PCP randomness. Finally, the argument prover emulates the PCP verifier on the obtained PCP randomness, which induces queries to the PCP string. The argument string output by the argument prover consists of the Merkle commitment, the salt, the queries, their answers, and an opening proof for the queries and answers.

- The argument verifier checks the opening proof, derives PCP randomness like the argument prover did, and checks that the PCP verifier accepts when run with that randomness on the given queries and answers.

2.4.1 UC-friendly completeness

We use Lemma 2.10 to show that the Micali construction has monotone proofs and unpredictable queries. Then by Lemma 2.5 we deduce that the Micali construction satisfies UC-friendly completeness.

Lemma 2.15 (informal)**.** $\mathsf{Micali}[\mathsf{PCP},\mathsf{r}]$ *has monotone proofs and unpredictable queries with error* $\epsilon_{\mathsf{MT}} + \frac{t_{\mathsf{p}}}{2^{\mathsf{r}}}$ *(*ϵ_{MT} *is from Lemma 2.10). By Lemma 2.5,* $\mathsf{Micali}[\mathsf{PCP},\mathsf{r}]$ *has UC-friendly completeness with error (roughly)* $\epsilon_{\mathrm{ARG}} = \ell_{\mathsf{p}} \cdot (\epsilon_{\mathsf{MT}} + \frac{t_{\mathsf{p}}}{2^{\mathsf{r}}})$.

2.4.2 UC-friendly zero knowledge

We show that the Micali construction satisfies UC-friendly zero knowledge.

Lemma 2.16 (informal)**.** *Let* PCP *be an honest-verifier zero knowledge PCP with error* ζ_{PCP}*. Let* ζ_{MT} *be the UC-friendly hiding error in Lemma 2.12. Then* $\mathsf{Micali}[\mathsf{PCP},\mathsf{r}]$ *has UC-friendly zero knowledge with error (roughly)* $\zeta_{\mathrm{ARG}} = \ell_{\mathsf{p}} \cdot (\frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\mathsf{r}}} + \zeta_{\mathrm{PCP}} + \zeta_{\mathsf{MT}})$.

The proof of this statement uses Lemma 2.7 to reduce UC-friendly zero knowledge to a game in which the adversary makes only a single query to the prover oracle. Then we use a sequence of game hops, relying among other things on the UC-friendly hiding property of the Merkle commitment scheme (Lemma 2.12).

2.4.3 UC-friendly knowledge soundness

We show that the Micali construction satisfies UC-friendly knowledge soundness.

Lemma 2.17 (informal)**.** *Let* PCP *be a knowledge sound PCP with error* κ_{PCP}*. Let* κ_{MT} *be the UC-friendly extraction error in Lemma 2.14. Then* $\mathsf{Micali}[\mathsf{PCP},\mathsf{r}]$ *has UC-friendly knowledge soundness with error (roughly)* $\kappa_{\mathrm{ARG}} = \ell_{\mathsf{v}} \cdot ((t_{\mathsf{q}}+1) \cdot \kappa_{\mathrm{PCP}} + \kappa_{\mathsf{MT}})$.

Note that Lemma 2.17 imposes no additional requirements on the PCP compared to what is usually required for regular knowledge soundness of $\mathsf{Micali}[\mathsf{PCP},\mathsf{r}]$. Yet we achieve the UC-friendly strengthening.

The proof of Lemma 2.17 informally works as follows. We reduce to the state-restoration knowledge soundness of the PCP (a notion implied by the PCP's knowledge soundness) and to the UC-friendly extraction property of the Merkle commitment scheme. This is similar to prior work [BCS16; CY24] except that in our setting the adversary has access to a simulation oracle, so part of the work in our analysis is showing that simulated proofs do not help the adversary.

In the reduction to the PCP's state-restoration knowledge soundness, the adversary's queries to the Fiat–Shamir oracle are translated to moves in the state-restoration game. The simulator has an advantage over the adversary in its ability to undetectably program the Fiat–Shamir query (the point used to derive the PCP randomness used for PCP verification). In order for the reduction to succeed, we must argue that this additional capability does not help the adversary. This is because points programmed by the simulator are domain-separated by instance, and the adversary wins the UC-friendly knowledge soundness game only by outputting "fresh" instance-proof pairs (the instance was not previously submitted to the simulator oracle). Thus, the instance-proof pair that the adversary outputs must not have been produced by the simulator oracle.

Having made this observation, the state-restoration knowledge soundness adversary runs the UC-friendly knowledge soundness adversary, simulating the simulator oracle and extracting (in a straightline fashion) PCP strings from instance-root-salt triples submitted to the Fiat–Shamir oracle using the Merkle commitment extractor guaranteed by UC-friendly extraction (Definition 2.13). The analysis of the reduction follows then similarly to that of state-restoration knowledge soundness in the ROM.

2.4.4 Conclusion

Lemma 2.15, Lemma 2.16, and Lemma 2.17 together show that the Micali construction satisfies UC-friendly completeness, UC-friendly zero knowledge, and UC-friendly knowledge soundness, provided that the underlying PCP is honest-verifier zero knowledge and knowledge sound. In turn, Lemma 2.9 implies that, under these conditions, the Micali construction is unconditionally UC-secure. Both steps provide concrete security bounds, leading to an overall concrete security bound for the UC-security of the Micali construction.

2.5 The BCS construction is UC-secure

We follow a similar approach to show that the BCS construction is unconditionally UC-secure: we prove that the BCS construction satisfies UC-friendly completeness, zero knowledge, and knowledge soundness. Recall that the BCS construction underlies many zkSNARKs that are concretely efficient (and widely deployed). We achieve concrete UC-security bounds for this notable class of zkSNARKs.

Review of the BCS construction. The BCS construction extends the Micali construction to work with interactive oracle proofs (IOPs), a multi-round generalization of PCPs. It compiles a (suitable) public-coin IOP into a zkSNARK, by using Merkle commitment schemes in the ROM, and the (multi-round) Fiat–Shamir transformation with salt size r. We denote this construction as $\mathsf{BCS}[\mathsf{IOP}, \mathsf{r}]$, and sketch it next.

- The argument prover runs the IOP prover, using the random oracle to simulate an interaction with the (public-coin) IOP verifier. For each round, the argument prover computes the round's IOP string, commits to it using the Merkle commitment scheme, and derives the next IOP verifier message using the random oracle (in a certain way that depends on the Merkle commitment and a salt, and either the instance or the previous Merkle commitment). Once the interaction is complete, the argument prover deduces the queries to the IOP strings and corresponding answers, and outputs an argument string containing the Merkle commitments, the salts, the query-answer pairs, and opening proofs of the commitments for those queries.

- The argument verifier checks the opening proofs, re-derives the IOP verifier randomness, and checks that the IOP verifier accepts when run with that randomness on the given queries and answers.

Remark 2.18 (BCS variant)**.** We consider a minor simplification of the BCS construction where the IOP verifier messages are derived by querying the random oracle at a point consisting of the instance and all Merkle commitment and salts so far. This simplifies the knowledge soundness analysis compared to the more common approach of querying at a point consisting of the last computed IOP verifier message, and the current Merkle commitment and salt. All results that we present directly extend to this more common approach.

2.5.1 UC-friendly completeness

We show that the BCS construction has monotone proofs and unpredictable queries, by building on Lemma 2.10 (which states that the Merkle commitment scheme has monotone proofs and unpredictable queries). Then by Lemma 2.5 we conclude that the BCS construction satisfies UC-friendly completeness.

Lemma 2.19 (informal)**.** $\mathsf{BCS}[\mathsf{IOP}, \mathsf{r}]$ *has monotone proofs and unpredictable queries with error* $\mathsf{k} \cdot (\epsilon_{\mathsf{MT}} + \frac{t_{\mathsf{p}}}{2^{\mathsf{r}}})$ *(*ϵ_{MT} *is from Lemma 2.10). By Lemma 2.5,* $\mathsf{BCS}[\mathsf{IOP}, \mathsf{r}]$ *has UC-friendly completeness with error (roughly)* $\epsilon_{\mathrm{ARG}} = \ell_{\mathsf{p}} \cdot \mathsf{k} \cdot (\epsilon_{\mathsf{MT}} + \frac{t_{\mathsf{p}}}{2^{\mathsf{r}}})$.

2.5.2 UC-friendly zero knowledge

We prove that the BCS construction satisfies UC-friendly zero knowledge, using a strategy similar to the case of the Micali construction (which is captured in Lemma 2.16). The proof of the lemma is similar, with the main difference being that we need the UC-friendly hiding property of the Merkle commitment scheme to hold for k commitment-openings pairs rather than a single one.

Lemma 2.20 (informal)**.** *Let* IOP *be a* k*-round public-coin IOP that has honest-verifier zero knowledge with error* ζ_{IOP}*. Let* ζ_{MT} *be the UC-friendly hiding error in Lemma 2.12. Then* $\mathsf{BCS}[\mathsf{IOP}, \mathsf{r}]$ *has UC-friendly zero knowledge with error (roughly)* $\zeta_{\mathrm{ARG}} := \ell_{\mathsf{p}} \cdot (\frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\mathsf{r}}} + \zeta_{\mathrm{IOP}} + \zeta_{\mathsf{MT}})$.

2.5.3 UC-friendly knowledge soundness

The BCS construction, when instantiated with an IOP that is state-restoration knowledge sound (with a straightline extractor), satisfies straightline knowledge soundness in the ROM [BCS16; CY24]. We prove a much stronger statement: the BCS construction satisfies UC-friendly knowledge soundness.

Lemma 2.21 (informal)**.** *Let* IOP *be an IOP with straightline state-restoration knowledge soundness with error* κ_{sr}*. Let* κ_{MT} *be the UC-friendly extraction error in Lemma 2.14. Then* $\mathsf{BCS}[\mathsf{IOP}, \mathsf{r}]$ *has UC-friendly knowledge soundness with error (roughly)* $\kappa_{\mathrm{ARG}} = \ell_{\mathsf{v}} \cdot (\kappa_{\mathsf{sr}} + \kappa_{\mathsf{MT}})$.

We prove Lemma 2.21 similarly to Lemma 2.17, making use of the fact that in that analysis we can reduce to the state-restoration knowledge soundness of the underlying PCP. In the case of the BCS construction, we reduce to the IOP version of state-restoration knowledge soundness. We again have to ensure that the adversary cannot use the simulation oracle in order to obtain an advantage, and an argument similar to that in Lemma 2.17 readily establishes that.

2.5.4 Conclusion

Lemma 2.19, Lemma 2.20, and Lemma 2.21 together show that the BCS construction satisfies UC-friendly completeness, UC-friendly zero knowledge, and UC-friendly knowledge soundness, provided that the underlying IOP is honest-verifier zero knowledge and (straightline) state-restoration knowledge sound. In turn, Lemma 2.9 implies that, under these conditions, the BCS construction is unconditionally UC-secure. Both steps provide concrete security bounds, leading to an overall concrete security bound for the UC-security of the BCS construction. This directly shows that existing zkSNARKs constructed from (state-restoration) knowledge sound and honest-verifier zero knowledge IOPs (e.g. [BCRSVW19; BBHR19] and similar constructions) are unconditionally UC-secure.

2.6 Adaptive corruptions and strong UC-friendly properties

The previous sections consider UC-security against *non-adaptive* corruptions. Here we outline how we additionally achieve UC-security against *adaptive* corruptions.

In the setting of UC-security against adaptive corruptions, the environment (through the adversary) may corrupt parties at *any time* during the protocol execution. When a party becomes corrupted, it reveals to the environment its private randomness (i.e., its private state). In the real-world the corrupted party directly reveals its own private randomness, while in the ideal-world the UC simulator must somehow sample randomness that "explains" a posteriori the past behavior of the party (possibly up to some error). Specifically, the challenge is that this randomness must be consistent with the input-output behavior of the party until this point of the execution. (The environment can send inputs to any party and receive corresponding outputs.)

Depending on the role of the corrupted party, simulating such randomness presents different challenges. If the corrupted party is the verifier, simulating its private randomness is easy, since it is the same in both the real-world and ideal-world. In contrast, if the corrupted party is the prover party then simulating its private randomness is more challenging. Indeed, the prover party invokes the proving interface, which is different in the two worlds: (i) in the real-world the proving interface runs the honest argument prover; and (ii) in the ideal-world the proving interface forwards its input to the ideal functionality, which in turn runs the zero knowledge simulator. In the ideal-world then, if the prover party is corrupted, the UC simulator must be able to produce, a posteriori, argument prover randomness that is consistent with all argument strings produced by the proving interface so far. More explicitly, the UC simulator must output randomness that the honest argument prover would have used to produce the argument strings that were output by the prover party thus far, *despite those argument strings being sampled by the zero knowledge simulator*. These additional capabilities must be explicitly accounted for in the UC-friendly properties.

Therefore, inspired by [LR22a], we consider "strong" variants of the UC-friendly properties in Section 2.2, which we obtain by adding a *corruption oracle* that returns the (possibly reconstructed) prover randomness used by the proving oracle of the game. Once the corruption oracle has been queried, we forbid further queries to the corruption oracle (and to the proving oracle), modeling how in the UC-security experiment control of a newly corrupted party (in this case the prover party) is relinquished to the environment.

By using these strong properties, Lemma 2.9 can be extended to provide emulation in the setting of adaptive corruptions.

Lemma 2.22 (informal)**.** *If the non-interactive argument* ARG *in Lemma 2.9 satisfies strong UC-friendly completeness, strong UC-friendly zero knowledge, and strong UC-friendly knowledge soundness, the conclusion of Lemma 2.9 holds even in the setting of adaptive corruptions (with the same error bound).*

The challenge is to show that the additional capability conferred to the adversary (by the new corruption oracles) in these strong UC-friendly experiments is not a problem. We focus on the steps required to satisfy these properties for the Micali construction; the strategy for the BCS construction is similar.

2.6.1 Strong UC-friendly completeness

Strong UC-friendly completeness is, conveniently, already implied by the three properties of perfect completeness, monotone proofs, and unpredictable queries, with the same error bounds. In other words, the Micali construction has strong UC-friendly completeness for free.

Lemma 2.23 (informal)**.** $\mathsf{Micali}[\mathsf{PCP},\mathsf{r}]$ *has strong UC-friendly completeness with the same error as in Lemma 2.15.*

2.6.2 Strong UC-friendly zero knowledge

Establishing strong UC-friendly zero knowledge for the Micali construction is more involved. We show that if the PCP underlying the Micali construction satisfies a natural notion that we call strong honest-verifier zero knowledge, the Micali construction satisfies strong UC-friendly zero knowledge.

Lemma 2.24 (informal)**.** *Let* PCP *be a strong honest-verifier zero knowledge PCP with error* ζ_{PCP}*. Then* $\mathsf{Micali}[\mathsf{PCP},\mathsf{r}]$ *has strong UC-friendly zero knowledge with the same error as in Lemma 2.16.*

The strong UC-friendly zero knowledge simulator is required to sample randomness that "explains" a simulated Micali argument string. This randomness has three components: (i) the PCP prover randomness; (ii) the Merkle commitment randomness; and (iii) the Fiat–Shamir randomness.

The strong honest-verifier zero knowledge property of the PCP is used to reconstruct the first piece of randomness. Roughly, strong honest-verifier zero knowledge PCPs are honest-verifier zero knowledge PCPs where the simulator additionally can, a posteriori, sample randomness that "explains" the sampled PCP local view. (Later, in Section 2.6.4, we show PCPs that satisfy this notion.) In order to reconstruct the Merkle commitment randomness, we show that Merkle commitment schemes satisfy a notion of strong UC-friendly hiding (briefly, this property extends Definition 2.11 with a corruption oracle). Finally, the Fiat–Shamir randomness is included in the Micali argument string, and thus the simulator has no need to reconstruct it. The combination of these three observations yields Lemma 2.24.

2.6.3 Strong UC-friendly knowledge soundness

Showing strong UC-friendly knowledge soundness for the Micali construction also requires some additional work. We strengthen the UC-friendly extraction property for the Merkle commitment scheme by adding a corruption oracle, and prove that the Merkle commitment scheme satisfies this stronger property.

Lemma 2.25. MT *has strong UC-friendly extraction with error (roughly)* $\kappa_{\mathsf{MT}} = \frac{3}{2} \cdot \frac{(t_{\mathsf{q}}+2\ell_{\mathsf{p}}\mathsf{l})^2}{2^\lambda} + \frac{2k(\mathsf{d}+1)\cdot(t_{\mathsf{q}}+2\ell_{\mathsf{p}}\mathsf{l})}{2^\lambda}$.

Lemma 2.25 directly implies Lemma 2.14. Our proof of Lemma 2.25 closely follows the proof of multi-extraction for the Merkle commitment scheme in [CY24], adapted to reflect the additional programming capabilities of the adversary and the presence of simulation and corruption oracles.

We adapt the proof of Lemma 2.17 to rely on strong UC-friendly extraction, and directly show that the Micali construction satisfies strong UC-friendly knowledge soundness. (Without any additional requirements on the underlying PCP.)

Lemma 2.26 (informal). *Let* PCP *be a knowledge sound PCP with error* κ_{PCP}. *Then* $\mathsf{Micali}[\mathsf{PCP}, \mathsf{r}]$ *has strong UC-friendly knowledge sound with the same error as in Lemma 2.17.*

2.6.4 Conclusion

UC-secure zkSNARKs from PCPs. The properties required of the underlying PCP are the ones that one would naturally expect to need for the adaptive UC-security of the Micali construction. Yet to our knowledge the PCP literature does not explicitly provide an off-the-shelf PCP with these properties.

We address this gap, by revisiting a transformation in [IW14] that combines a PCP and a zero knowledge PCP of proximity (PCPP) to obtain a zero knowledge PCP. We show that: (a) if the given PCP is knowledge sound then the resulting PCP is also knowledge sound; and (b) if the PCPP is strong honest-verifier zero knowledge then the resulting PCP is also strong honest-verifier zero knowledge. Then we construct a strong honest-verifier zero knowledge PCPP, and apply the transformation to any knowledge sound PCP (e.g., [BFLS91]) and this PCPP, concluding the proof of Theorem 1.2.

UC-secure zkSNARKs from IOPs. As mentioned before, we can prove analogues of Lemmas 2.24 and 2.26 for the BCS construction.

Lemma 2.27 (informal). *Let* IOP *be an IOP.*

- *If* IOP *is strong honest-verifier zero knowledge IOP with error* ζ_{IOP}, *then* $\mathsf{BCS}[\mathsf{IOP}, \mathsf{r}]$ *is strong UC-friendly zero knowledge with the same error as in Lemma 2.20.*

- *If* IOP *is a state-restoration knowledge sound IOP with error* κ_{IOP}*, then* $\mathsf{BCS}[\mathsf{IOP}, \mathsf{r}]$ *is strong UC-friendly knowledge sound with the same error as in Lemma 2.21.*

By inspection, we see that many IOPs used in practice satisfy these properties, and thus lead to UC-secure zkSNARKs. We sketch how the masked univariate sumcheck protocol [BCRSVW19; BCFGRS17], a core building block of many honest-verifier zero knowledge IOPs is strong honest-verifier zero knowledge. Let $\hat{p}$ be a polynomial, which the verifier has oracle access to, and $H \subseteq \mathbb{F}$ be a domain. The unmasked univariate sumcheck protocol allows the verifier to check that $\sum_{h \in H} \hat{p}(h) = \beta$ for some claimed value β. In the masked version, to achieve zero knowledge, the prover sends (as an oracle) a masking polynomial $\hat{q}$ and the value $\beta' = \sum_{h \in H} \hat{q}(h)$, the verifier samples a challenge c and then both parties run a unmasked univariate sumcheck to check the claim $\sum_{h \in H}(c \cdot \hat{p} + \hat{q})(h) = c \cdot \beta + \beta'$, which ultimately requires the verifier to query $\hat{p}, \hat{q}$ at a single location. The strong honest verifier zero knowledge simulator can reconstruct the prover randomness by sampling $\hat{q}$ uniformly at random, conditioned on the sum equaling β' and on the value of the query to $\hat{q}$ as determined during the honest verifier zero knowledge simulation phase. (The conditioning consists of linear constraints on the coefficients, so this sampling can be done efficiently.)

3 Preliminaries

3.1 Notation

List operations. For $i \in [n]$ and a list $x \in \Sigma^n$, we denote by $x[i]$ the i-th entry of x. For a set $S \subseteq [n]$, $x[S] \colon S \to \Sigma$ is the function that maps $i \in S$ to $x[i]$. We write $x \circ y$ for the concatenation of two lists, and (slightly abusing notation) $x \cap y$ for their intersection as sets.

Sampling. We write $x \leftarrow \mathcal{D}$ to denote that x is sampled from the distribution $\mathcal{D}$. For a set S, we write $x \leftarrow S$ to denote that x is sampled from the uniform distribution on S.

Oracles. We denote by $x \leftarrow \mathcal{A}^{f_1,\dots,f_k}$ the execution of an (oracle) algorithm $\mathcal{A}$, with a uniformly sampled random tape, and access to oracles $f_1, \dots, f_k$. We denote by $\mathcal{U}(\lambda)$ the set of functions $f \colon \{0,1\}^* \to \{0,1\}^\lambda$. A function $f \leftarrow \mathcal{U}(\lambda)$ is called a **random oracle**. We can derive from a random oracle $f \leftarrow \mathcal{U}(\lambda)$ another random oracle with smaller output size by truncation. An oracle can be **domain-separated** into *independent* oracles, by prefixing queries to the original oracle with a unique string for each (new) oracle. For $\ell_1, \dots, \ell_k \leq \lambda$, we write $f_1, \dots, f_k \leftarrow \mathcal{U}(\ell_1, \dots, \ell_k)$ for the oracles obtained from $f \leftarrow \mathcal{U}(\lambda)$ by domain separating and modifying the output size so that $f_i : \{0,1\}^* \to \{0,1\}^{\ell_i}$.

Next, we introduce notions and notation for programming random oracles. A **query-answer trace** is a list $\mathrm{tr} = ((\mathsf{qid}_i, x_i, y_i))_{i \in [t]}$, where $\mathsf{qid}_i \in \{\mathsf{query}, \mathsf{prog}\}$ specifies if the query obtains an answer or programs an answer, x_i is the query, y_i is the answer. We say that tr is **invalid** if there exists $i, j \in [t]$ such that $x_i = x_j$ and $y_i \neq y_j$. For a function $f \in \mathcal{U}(\lambda)$, the function $f[\mathrm{tr}]$ is defined as follows:

$$f[\mathrm{tr}](x) \coloneqq \begin{cases} \bot & \text{if tr is invalid} \\ y_i & \text{else if } \exists\, i \text{ s.t. } x_i = x \\ f(x) & \text{otherwise} \end{cases} .$$

For $f \in \mathcal{U}(\lambda)$ and a trace tr' we define the (stateful) programmable oracle $[\![f, \mathrm{tr}']\!]$ as follows.

```
$[\![f, \mathrm{tr}']\!]$:
1. Initialize a list $\mathrm{tr} \coloneqq \mathrm{tr}'$.
2. On a random oracle query $x$, set $y \coloneqq f[\mathrm{tr}](x)$, append $(\mathsf{query}, x, y)$ to $\mathrm{tr}$, and return $y$.
3. On a programming query $\mathsf{trace}_{\mathsf{prog}}$:
    (a) If there exist $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}$ with $x_i = x$, return $0$.
    (b) Else append $((\mathsf{prog}, x, y))_{(x,y) \in \mathsf{trace}_{\mathsf{prog}}}$ to $\mathrm{tr}$ and return $1$.
```

We write $[\![f]\!] \coloneqq [\![f, \emptyset]\!]$, and write $y \xleftarrow{\mathrm{tr}} \mathcal{A}^{[\![f,\mathrm{tr}']\!]}$ for the output y of $\mathcal{A}$ when running with oracle $[\![f, \mathrm{tr}']\!]$ and with final trace tr (note that in this case tr denotes the list maintained by the oracle, so it does not include failed programming queries). If tr' is invalid, so is tr; conversely, if tr' is valid so is tr. We denote by $\mathsf{ro}(\mathrm{tr}) \coloneqq ((\mathsf{qid}, x, y) \in \mathrm{tr} : \mathsf{qid} = \mathsf{query})$ and $\mathsf{prog}(\mathrm{tr}) \coloneqq ((\mathsf{qid}, x, y) \in \mathrm{tr} : \mathsf{qid} = \mathsf{prog})$ the (deduplicated, ordered) lists of query-answer pairs made, respectively, to the random and programming oracle. We also write $y \xleftarrow{\mathrm{tr}} \mathcal{A}^f$ to denote that running $\mathcal{A}$ with the (non-programmable) random oracle f has output y and query-answer trace tr (and, in this case, $\mathsf{qid}_i = \mathsf{query}$ for $i \in [t]$). We naturally extend the notions above for multiple random oracle, in which case the query-answer trace is augmented with an entry oid specifying to which oracle the query in question was made.

An adversary $\mathcal{A}$ that has access to a programmable random oracle is $(t_{\mathsf{q}}, t_{\mathsf{p}})$**-query** if it makes at most t_{q} random oracle queries and t_{p} programming queries (where a programming query with input $\mathsf{trace}_{\mathsf{prog}}$ is counted as $|\mathsf{trace}_{\mathsf{prog}}|$ queries). For any algorithm $\mathbf{A}$,

- $\mathsf{q}_{\mathbf{A}}(x_1, \ldots, x_k)$ is an upper bound on the number of random oracle queries made by $\mathbf{A}(x_1, \ldots, x_k)$;
- $\mathsf{p}_{\mathbf{A}}(x_1, \ldots, x_k)$ is an upper bound on the number of programming queries made by $\mathbf{A}(x_1, \ldots, x_k)$.

We also define $\mathsf{q}_{\mathbf{A}}(n_1, \ldots, n_k) := \max_{|x_i| \leq n_i} \mathsf{q}_{\mathbf{A}}(x_1, \ldots, x_k)$ and $\mathsf{p}_{\mathbf{A}}(n_1, \ldots, n_k) := \max_{|x_i| \leq n_i} \mathsf{p}_{\mathbf{A}}(x_1, \ldots, x_k)$.

Relation. A **relation** R is a set of tuples $(\mathbb{x}, \mathbb{w})$, where $\mathbb{x}$ is an instance and $\mathbb{w}$ is a witness. We associate a language $L(R)$, which is the set of instances $\mathbb{x}$ such that there exists a witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in R$.

Statistical distance. Let G_0, G_1 be two algorithms with outputs in D. The statistical distance (i.e., total variation distance) between G_0, G_1 on input x is defined as

$$\Delta_x(G_0, G_1) := \frac{1}{2} \sum_{\alpha \in D} |\Pr[G_0(x) = \alpha] - \Pr[G_1(x) = \alpha]| \enspace .$$

If $D = \{0, 1\}$ then $\Delta_x(G_0, G_1) = |\Pr[G_0(x) = 1] - \Pr[G_1(x) = 1]|$. We write $G_0 \equiv G_1$ if, for every x, $\Delta_x(G_0, G_1) = 0$.

3.2 UC-security with unbounded adversaries

Universally composable (UC) security [Can01; Can20] provides a general framework for establishing the security of cryptographic protocols. The security guarantees hold under a general composition operation, which enables modular analysis. In this work, we use a global random oracle [CJS14; CDGLN18], which is a shared global entity every party in the security experiment can access. The plain UC model does not provide a composability theorem for protocols interacting with such a shared global functionality, which was later rectified by the generalized universally composable (GUC) framework [CDPW07]. However, [BCHTZ20, Appendix A] noted that the GUC framework is subtly inconsistent, and provide a blueprint for proving security of protocols with global setup in the plain UC model, which we sketch next, and follow in this work.

In this section, we provide an informal description of the model for UC-security, and refer the reader to [Can20] for more details. Furthermore, we describe the (minor) modifications to said model that we undertake in order to capture security against computationally unbounded adversaries.

The model of computation in the UC model is **interactive Turing machines** (ITMs [Can20, Sec 3.1.1, Def 4]), a generalization of Turing machines that can communicate with each other. An ITM is uniquely identified by its identity tape, which contains an identity (consisting of a **party-id** and a **session-id**) and a description of the code of the ITM. This information, together with the content of its tapes, is referred to as an **ITM instance** ([Can20, Sec 3.1.1, Def 5]). Execution of a system of ITMs is defined in [Can20, Sec 3.1.2]. A **system of ITMs** is specified by an initial ITM I and a **control function** C. The execution starts by running the initial ITM, and terminates when that same ITM halts, outputting the content of its tape. ITMs in the system can run **external-write** instructions, which can be used to send messages, spawn new ITMs, and more. Once an external-write instruction is issued, the control function decides whether it is allowed, and possibly modifies the instruction written.[11] A **parametrized system of ITMs** is a list of systems of ITMs $((I_\lambda, C_\lambda))_\lambda$ parametrized by a security parameter $\lambda \in \mathbb{N}$, which, abusing notation, we write (I, C) leaving λ implicit. A **protocol** is an ITM, which in this work we assume to be **subroutine exposing** [Can20, Def. 21].

Definition 3.1 ([Can20, Sec 3.1.2]). *For a system of ITMs (I, C), $\mathsf{UCOut}_{I,C}(z)$ is the random variable denoting the output of the execution under the control function C when the initial ITM I is started with input z, where the randomness is taken over the random tapes of the ITMs in the system. For a parametrized system of ITMs (I, C), we define $\mathsf{UCOut}_{\lambda,I,C}(z) := \mathsf{UCOut}_{I_\lambda,C_\lambda}(z)$.*

[11]More precisely, this is an extended system of ITMs in the terminology of [Can20]; we use the system terminology for simplicity.

The control function is parametrized by an adversary $\mathcal{A}$ and a protocol π, and determines what is allowed for the main security experiment. We use a control function $C_{\mathcal{G}}^{\pi,\mathcal{A}}$ to model UC-security in the presence of a global ITM $\mathcal{G}$. Our control function builds upon the standard UC-security control function, which is formally described in [Can20, Fig 6]. In the control function $C_{\mathcal{G}}^{\pi,\mathcal{A}}$:

- The adversary is not allowed to pass or receive input from ITMs in the executions, it is only allowed to interact with those machines via designated backdoor tapes.
- The environment can communicate with the adversary, and is only allowed to spawn ITM instances of the protocol π with the same session-id.
- Additionally we allow the adversary to pass and receive output to and from a single specified ITM $\mathcal{G}$.

By setting $\mathcal{G}$ to be a "dummy" ITM, we recover the standard control function.

Unconditional security. Unlike previous works, we consider a setting in which the environment is computationally unbounded, and whose capabilities are only limited by the number of times it is allowed to access some shared resources, such as a random oracle. To model this setting, we revisit the mechanism of **import** and **time budget** introduced in [Can20, Sec 3.2], to introduce a generalized **budget**. First, we review import and time budget, as a modeling of efficient computation. [Can20] mandates that each external-write must contain a numeric field called an import. Each ITM has a starting time budget, which is incremented by the import of received messages, and decremented by the import of sent messages. A protocol is T-bounded if, at any point in the execution, the number of steps it took is at most $T(n)$, where n is the current time budget. A protocol is **efficient** is it is p-bounded for some polynomial p.

We extend this mechanism, and we assume that each ITM has a starting **budget vector**, containing a non-negative integer for each resource whose access we wish to limit. We mandate the following requirements.

- Each external-write instruction requires specifying a budget vector.
- At any point in time, the current budget of an ITM is the sum (componentwise) of the starting budget and the budget of all incoming messages, minus (componentwise) the budget of all outgoing messages.
- If at any point in time the budget vector of an ITM has a negative entry, the execution halts.

For the main security experiment, we assume that the environment starts with some budget vector, and the adversary starts with the zero budget vector. The protocol has its own budget (separate from the environment) that it can use, which we leave unspecified (and assume large enough at all times). A protocol is **$\mathcal{B}$-budget** if its starting budget is $\mathcal{B}$. We also still use the original import mechanism to ensure that honest protocols are efficient, and in a parametrized system of ITMs we assume that each protocol does not start execution until it received import at least λ.

With this new budget mechanism, we can define notation for the main security experiment. The output of the main security experiment, when started with (i) protocol π; (ii) environment $\mathcal{E}$; (iii) adversary $\mathcal{A}$; (iv) global functionality $\mathcal{G}$; (v) security parameter λ; and (vi) input z, is the output of the execution of the system of ITMs with parameter λ, initial ITM $\mathcal{E}$, and the control function $C_{\mathcal{G}}^{\pi,\mathcal{A}}$, on the input z.

Definition 3.2. *Let $\pi, \mathcal{E}, \mathcal{A}, \mathcal{G}$ be ITMs. Define* $\mathsf{EXEC}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{E}}(\lambda, z) := \mathsf{UCOut}_{\lambda,\mathcal{E},C_{\mathcal{G}}^{\pi,\mathcal{A}}}(z)$.

Next, our aim is to give a description of the composition theorem [Can20, Thm. 22] that is amenable to our unconditional security setting. We start by defining UC-emulation [Can20, Sec 4.2, Def 9]. Informally, a protocol π UC-emulates a protocol φ if the output of the environment $\mathcal{E}$ in the main security experiment, when run with protocol π and an adversary $\mathcal{A}$, is statistically close to that while running with protocol φ and some simulator $\mathcal{S}$ (which may depend on $\mathcal{A}$ but not on $\mathcal{E}$).

Definition 3.3. *Let $\mathcal{G}, \pi, \varphi$ be protocols. We say that π $\mathcal{B}$-***UC-emulates** *φ* **in the $\mathcal{G}$-hybrid model** *with simulation error σ and simulation overhead $\mathcal{B}'$ if for every $\mathcal{A}$ there exists an efficient $\mathcal{B}'$-budget simulator $\mathcal{S}$ such that for every $\mathcal{B}$-budget $\mathcal{E}$*

$$\Delta_\lambda(\mathsf{EXEC}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{E}}, \mathsf{EXEC}^{\mathcal{G}}_{\varphi,\mathcal{S},\mathcal{E}}) \leq \sigma(\lambda) \ .$$

Remark 3.4. Using [Can20, Sec 4.3.1] we can replace $\mathcal{A}$ with a "dummy adversary" $\mathcal{A}_{\mathrm{D}}$, which yields an equivalent definition that is significantly easier to work with.

An **ideal-functionality** $\mathcal{F}$ is an ITM instance, and induces a protocol $\mathsf{IDEAL}_{\mathcal{F}}$ [Can20, Sec 5.3]. In $\mathsf{IDEAL}_{\mathcal{F}}$ there is a single instance of $\mathcal{F}$, and multiple dummy parties that simply forward their inputs to $\mathcal{F}$ and then return the outputs of $\mathcal{F}$ to their callers.

Definition 3.5. *Let π a protocol, $\mathcal{F}$ an ideal functionality, and $\mathcal{G}$ a global functionality. Let $\mathsf{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \mathsf{EXEC}^{\mathcal{G}}_{\mathsf{IDEAL}_{\mathcal{F}},\mathcal{S},\mathcal{E}}$. We say that π $\mathcal{B}$-***UC-realizes** *$\mathcal{F}$* **in the $\mathcal{G}$-hybrid model** *if π $\mathcal{B}$-UC-emulates $\mathsf{IDEAL}_{\mathcal{F}}$ in the $\mathcal{G}$-hybrid model.*

Specializing Definitions 3.2, 3.3 and 3.5 to the case where $\mathcal{G}$ is a dummy functionality recovers the standard notion of UC emulation and ideal functionalities.

Definition 3.6. *Let $\mathcal{D}$ be a dummy ITM, which passes no output to its caller. Let $\mathcal{G}, \pi, \varphi$ be protocols. We say that π $\mathcal{B}$-***UC-emulates** *φ if π $\mathcal{B}$-UC-emulates φ in the $\mathcal{D}$-hybrid model. We further say that π $\mathcal{B}$-***UC-realizes** *$\mathcal{F}$ if π $\mathcal{B}$-UC-realizes $\mathcal{F}$ in the $\mathcal{D}$-hybrid model.*

For protocols ρ, π, φ, the UC operator $\rho^{\pi\to\varphi} := \mathsf{UC}(\rho, \pi, \varphi)$ is defined in [Can20, Sec 6.1]. Intuitively, it replaces invocations of π in ρ with invocations of φ. The composition theorem formalizes the intuitive notion that if π UC-emulates φ then this transformation yields a protocol that emulates ρ.

Theorem 3.7 ([Can20, Thm. 22])**.** *Let ρ, π, φ be protocols, and let $t_\pi(\rho, \lambda)$ be a bound on the number of instances of π that ρ spawns when started with parameter λ. Suppose that:*
- *ρ is (π, φ)-compliant [Can20, Sec 6.1];*
- *π, φ are subroutine respecting [Can20, Def 19]; and*
- *π $\mathcal{B}$-UC-emulates φ with simulation error σ and simulation overhead $\mathcal{B}'$.*

Then $\rho^{\pi\to\varphi}$ $\mathcal{B}$-UC-emulates ρ with simulation error $t_\pi(\rho, \lambda) \cdot \sigma$ and simulation overhead $t_\pi(\rho, \lambda) \cdot \mathcal{B}'$.

The UC theorem has some technical preconditions. Compliance is a requirement on the calling protocol, and thus it is out of scope for this work. Subroutine respecting protocols are protocols whose subprotocols (and subprotocols of those protocols) communicate only with parties outside their session through the main protocol. This precondition is what prevents the UC theorem from being applied in presence of a global functionality, as said functionality is outside the main session and will be queried by the emulator and the emulated protocol. In order to allow global shared functionalities $\mathcal{G}$, [BCHTZ20, Def 3.2] introduce **$\mathcal{G}$-subroutine respecting protocols**, which informally are subroutine respecting protocols whose subprotocols (including themeselves) are allowed to pass and receive output from $\mathcal{G}$. They also introduce a new "manager" transformation M [BCHTZ20, Appendix B] that can be used to formulate a composition theorem for UC with global subroutines. Roughly, for "nice" protocols π, φ and a global protocol $\mathcal{G}$, the UCGS theorem shows that if $\mathsf{M}[\pi, \mathcal{G}]$ UC-emulates $\mathsf{M}[\varphi, \mathcal{G}]$ then the composition theorem can be applied. For the UC with Global Subroutines theorem, we require $\mathcal{G}$ to be **π-regular**, which disallows $\mathcal{G}$ from spawning new ITMs and from using π as a subroutine.

Theorem 3.8 ([BCHTZ20, Thm. 3.5]). *Let $\rho, \pi, \varphi, \mathcal{G}$ be protocols. Suppose that:*

- *ρ is (π, φ)-compliant and $(\pi, \mathsf{M}[\zeta, \mathcal{G}])$-compliant for $\zeta \in \{\pi, \varphi\}$;*
- *$\mathcal{G}$ is subroutine respecting and π-regular [BCHTZ20, Def 3.3];*
- *π, φ are $\mathcal{G}$-subroutine respecting [BCHTZ20, Def 3.2];*
- *$\mathsf{M}[\pi, \mathcal{G}]$ $\mathcal{B}$-UC-emulates $\mathsf{M}[\varphi, \mathcal{G}]$ with simulation error σ and simulation overhead $\mathcal{B}'$.*

Then $\rho^{\pi \to \varphi}$ $\mathcal{B}$-UC-emulates ρ with simulation error $t_\pi(\rho, \lambda) \cdot \sigma$ and simulation overhead $t_\pi(\rho, \lambda) \cdot \mathcal{B}'$.

Modeling corruptions. Corruptions are not explicitly modeled in the UC framework, but instead are modeled as additional interfaces exposed by protocols. The corruption models that we study in this work are **static corruptions** and **adaptive corruptions**. In the case of static corruptions, the adversary can corrupt a party at the start of the execution, and assumes complete control of it for the rest of the execution. In the case of adaptive corruptions, the adversary can dynamically assume control of a party, and when it does so it forces said party to reveal the randomness used thus far. Our result will hold in both settings, and we will use blue to detail the modifications required for the case of adaptive corruptions. In accordance to the budget mechanism that we introduced, we additionally extend the traditional corruption mechanism to set the budget of corrupted parties to 0. This ensures that the environment/adversary cannot access additional resources using corruptions.

Remark 3.9. The mechanism of budget that we have introduced to model unconditional security is not a standard UC notion, and is not considered in previous works. In principle, it could invalidate some of the results that we later rely on such as Theorem 3.7 and Theorem 3.8. We have verified that the proofs of these results can be adapted, with minor bookkeeping modifications, to hold in our model. We suggest that future work that aims for UC-results in this unconditional setting employs the mechanism we introduced. We also considered alternative mechanisms to give unconditional security bounds, which we briefly mention.

- Modifying the global functionalities to stop answering queries after a certain number of queries have been made. While this is a conceptually simple modification to make, it enables a simple distinguishing attack. Consider for example a global random oracle that only allows t_{q} queries, and suppose that the real and ideal protocol make a *distinct and known* number of queries to the GROM. Then, an environment could run the protocol, and query the GROM until it stops answering to deduce the number of queries the protocol made, and, consequently, deduce if it is run in the real-world or ideal-world. While we could still achieve UC-security in this context with tweaks to the UC-simulator, this adds additional complexity to disallow an attack that anyways does not reflect real-world attacks.

- Giving theorems for *quantified* environments. This would imply giving results of the form "for every environment $\mathcal{E}$ that makes at most t_{q} oracle queries...". In fact, the environment can make queries to restricted functionalities through the adversary and corrupted parties, which would make the quantification even more unwieldy than in this example. We prefer to introduce budgets within the UC-framework, in order to give more compact and precise theorems.

3.3 Global random oracle

Our results hold in the **global restricted programmable observable random oracle** [CDGLN18]. In this model all parties have access to an oracle that can be queried and programmed. Every party can also check whether a point has been programmed. The simulator has an advantage over the environment in that it can program points undetectably. This model was designed to prove the security of particularly efficient protocols, such as the folklore commitment scheme $\mathsf{cm} := f((m, r))$ (where m is a message and r a random salt).

We refer the reader to [CDGLN18] for a discussion of the features of this model, compared to other global random oracle models. Our definition slightly differs from prior ones, as we allow parties to *atomically* program many query-answer pairs at once (if any of the pairs was previously programmed the entire request fails and the oracle's state remains unchanged). An atomic programming request requires the calling party to expend budget equivalent to repeatedly calling the programming functionality for each query-answer pair. In the language of Section 3.2, in this paper we establish that certain GRO-subroutine-respecting protocols UC-realize a desired ideal functionality, where the global functionality GRO is defined next.

Functionality 3.1. The GRO functionality [CDGLN18] is defined as follows:
Parameters: security parameter λ
State: underlying random oracle $f \leftarrow \mathcal{U}(\lambda)$, initially empty lists tr, $\{\mathsf{IllegitimateTrace}_{\mathsf{sid}}\}_{\mathsf{sid}}$
Functionality:

- $\mathsf{GRO.Query}(x)$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$ or the adversary:
  1. Set $y := f[\mathrm{tr}](x)$ and append (query, x, y) to tr.
  2. Parse x as (sid, x') for sid a session ID.
  3. If the query came from the adversary or $\mathsf{sid} \neq \mathsf{sid}_M$, append (x', y) to $\mathsf{IllegitimateTrace}_{\mathsf{sid}}$.
  4. Output $(\mathtt{Query}, y)$ to the caller.
- $\mathsf{GRO.Observe}(\mathsf{sid})$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$ or the adversary:
  1. Output $(\mathtt{Observe}, \mathsf{IllegitimateTrace}_{\mathsf{sid}})$.
- $\mathsf{GRO.Program}(\mathsf{trace}_{\mathsf{prog}})$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$ or the adversary:
  1. If there exist $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ and $(\mathsf{query}, x_i, y_i) \in \mathrm{tr}$ with $x_i = x$, output $(\mathtt{Program}, 0)$.
  2. Else append $((\mathsf{prog}, x, y))_{(x,y)\in\mathsf{trace}_{\mathsf{prog}}}$ to tr.
  3. Output $(\mathtt{Program}, 1)$.
- $\mathsf{GRO.IsProgrammed}(x)$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$ or the adversary:
  1. Parse x as (sid, x') for sid a session ID.
  2. If the query was made by the adversary or $\mathsf{sid} \neq \mathsf{sid}_M$, return $\perp$.
  3. If there exists y such that $(\mathsf{prog}, x, y) \in \mathrm{tr}$, return $(\mathtt{IsProgrammed}, 1)$; else return $(\mathtt{IsProgrammed}, 0)$.

We introduce notation for less verbose queries to the global random oracle.

Definition 3.10. *We write* $\mathsf{GRO}_{\mathsf{sid}}$ *for the domain separated oracle* $\mathsf{GRO}_{\mathsf{sid}}(x) := \mathsf{GRO.Query}((\mathsf{sid}, x))$.

Note that GRO is π-regular for every protocol π, as it does not invoke subprotocols nor passes output to any ITM that did not query it. Moreover, GRO is subroutine respecting. Hence GRO satisfies the preconditions of Theorem 3.8.

4 UC-security for non-interactive arguments in the ROM

We describe the notion of security that we establish for non-interactive arguments in the ROM. First we recall the relevant syntax. Let f be sampled from $\mathcal{U}(\lambda)$. A non-interactive argument in the ROM is a tuple $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ that works as follows.

- The *argument prover* $\mathbf{P}$, given query access to f, receives as input an instance $\mathbb{x}$ and a witness $\mathbb{w}$, and outputs an argument string π.
- The *argument verifier* $\mathbf{V}$, given query access to f, receives as input an instance $\mathbb{x}$ and an argument string π, and outputs a decision bit.

In this work we study UC-security for non-interactive arguments, so we do not state the usual notions of completeness and soundness. Instead, in Section 4.1 we provide an ideal functionality $\mathcal{F}_{\mathrm{aARG}}$ that captures these notions, as well as *zero knowledge* and *knowledge soundness*. Then in Section 4.2 we construct, starting from a non-interactive argument ARG in the ROM, a protocol $\Pi_a[\mathsf{ARG}]$ in the GROM. In later sections we show that if ARG satisfies certain "UC-friendly" properties then $\Pi_a[\mathsf{ARG}]$ UC-emulates $\mathcal{F}_{\mathrm{aARG}}$ in the GRO-hybrid model. (Recall that these UC-friendly properties and the UC-emulation are unconditional.)

4.1 Ideal functionality

In Functionality 4.1 we provide the **ARG ideal functionality** $\mathcal{F}_{\mathrm{aARG}}$ introduced in [LR22b] (called NIZKPoK functionality there), and later extended in [LR22a] to include adaptive corruptions. We outline how $\mathcal{F}_{\mathrm{aARG}}$ captures the usual desiderata of a non-interactive argument.

- **Syntax.** The ideal functionality has a prover interface $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Prove}$ and a verifier interface $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Verify}$, matching the prover and verifier of a non-interactive argument. Additionally, the ideal functionality exposes the interface $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Setup}$ and the interface $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Corrupt}$. The simulator uses $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Setup}$ to pass to the functionality the tuple of algorithms to be used for proving and verification. $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Corrupt}$ is called by the simulator in the event of a corruption, and returns information used to simulate the random tape of the party being corrupted. If the party is the verifier, this is the randomness used thus far in the verification; if the corrupted party is the prover, this information is the randomness simulated in the proving.
- **Non interactivity.** The ideal functionality interacts with the simulator only in $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Setup}$. This implies that only non-interactive argument systems can realize the functionality.
- **Completeness.** $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Verify}$ accepts all argument strings generated by $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Prove}$.
- **Knowledge soundness.** $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Verify}$ attempts to extract a witness for instances not previously queried to the proving oracle accompanied by valid proofs, and outputs an error if extraction fails.
- **Zero knowledge.** $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Prove}$ outputs simulated proofs generated without the witness.

For simplicity, we give the definition of $\mathcal{F}_{\mathrm{aARG}}$ for a specific session id sid.

Functionality 4.1. The $\mathcal{F}_{\mathrm{aARG}}$ functionality for a session sid is defined as follows.
Parameters: A relation R, an instance bound n.
Participants: A (dummy) prover party M_P and a (dummy) verifier party M_V.
State: A tuple of algorithms $\mathsf{algTuple}$, initially equal to $\perp$. Several lists (initially empty):

- $\mathsf{InstanceList}$, list of proved instances;
- Proved, list of proved statements;
- $\mathsf{hProgrammed}$, list of (honestly) programmed points;
- $\mathsf{extTrace}$, list of queries of the adversary and the simulator to the GROM;

```
• $\mathsf{Random}_{\mathbf{P}}$, list of prover randomness strings;
• $\mathsf{Random}_{\mathbf{V}}$, list of verifier randomness strings;
• $\mathsf{Corrupted}$, list of corrupted parties.
Functionality:
• $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Setup}()$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$:
    1. If this interface was previously called, $\mathsf{sid} \neq \mathsf{sid}_M$, or or $M \in \mathsf{Corrupted}$, return $\bot$.
    2. Pass $(\mathtt{Setup}, \mathsf{sid})$ to the simulator $\mathcal{S}$ and receive a tuple of algorithms $(\mathbf{V}, \mathbf{S}, \mathbf{E})$.
    3. Set $\mathsf{algTuple} := (\mathbf{V}, \mathbf{S}, \mathbf{E})$.
• $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Prove}(\mathbb{x}, \mathbb{w})$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$:
    1. If $\mathsf{sid} \neq \mathsf{sid}_M$ or $\mathsf{algTuple} = \bot$ or $|\mathbb{x}| > n$ or $M \in \mathsf{Corrupted}$, return $\bot$.
    2. If $(\mathbb{x}, \mathbb{w}) \notin R$, return $\bot$.
    3. Obtain $\mathsf{IllegitimateTrace}_{\mathsf{sid}}$ from $\mathsf{GRO.Observe}(\mathsf{sid})$.
    4. Append to $\mathsf{extTrace}$ the query-answer pairs in $\mathsf{IllegitimateTrace}_{\mathsf{sid}}$ not already present.
    5. Compute $(\pi, \mathrm{tr}, z_\pi) \xleftarrow{\mathrm{tr}_{\mathbf{S}}} \mathbf{S}^{\mathsf{GRO}_{\mathsf{sid}}}(\mathbb{x})$.
    6. Compute $(\rho_{\mathbf{P}}, \mathrm{tr}') \xleftarrow{\mathrm{tr}'_{\mathbf{S}}} \mathbf{S}^{\mathsf{GRO}_{\mathsf{sid}}}(\mathbb{w}, z_\pi)$.
    7. Set $\mathsf{extTrace} := \mathsf{extTrace} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}}$.
    8. Call $\mathsf{GRO.Program}(((\mathsf{sid}, x), y)_{(x,y) \in \mathrm{tr} \circ \mathrm{tr}'})$, outputting Fail if the call returns $(\mathtt{Program}, 0)$.
    9. Set $\mathsf{hProgrammed} := \mathsf{hProgrammed} \circ \mathrm{tr} \circ \mathrm{tr}'$.
   10. Append $\mathbb{x}$ to $\mathsf{InstanceList}$.
   11. Append $(\mathbb{x}, \pi)$ to $\mathsf{Proved}$.
   12. Append $\rho_{\mathbf{P}}$ to $\mathsf{Random}_{\mathbf{P}}$.
   13. Return $(\mathtt{Proof}, \mathsf{sid}, \mathbb{x}, \pi)$.
• $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Verify}(\mathbb{x}, \pi)$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$:
    1. If $\mathsf{sid} \neq \mathsf{sid}_M$ or $\mathsf{algTuple} = \bot$ or $|\mathbb{x}| > n$ or $M \in \mathsf{Corrupted}$, return $\bot$.
    2. Sample $\rho_{\mathbf{V}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ and append it to $\mathsf{Random}_{\mathbf{V}}$.
    3. Compute $b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{\mathsf{GRO}_{\mathsf{sid}}}(\mathbb{x}, \pi; \rho_{\mathbf{V}})$.
    4. If $(\mathbb{x}, \pi) \in \mathsf{Proved}$, return $(\mathtt{Verification}, \mathsf{sid}, \mathbb{x}, \pi, 1)$.
    5. If $b = 0$ return $(\mathtt{Verification}, \mathsf{sid}, \mathbb{x}, \pi, 0)$.
    6. If there exists $(x, y) \in \mathrm{tr}_{\mathbf{V}} \setminus \mathsf{hProgrammed}$ such that $\mathsf{GRO.IsProgrammed}((\mathsf{sid}, x)) = (\mathtt{IsProgrammed}, 1)$,
       return $(\mathtt{Verification}, \mathsf{sid}, \mathbb{x}, \pi, 0)$.
    7. If $\mathbb{x} \notin \mathsf{InstanceList}$:
       (a) Obtain $\mathsf{IllegitimateTrace}_{\mathsf{sid}}$ from $\mathsf{GRO.Observe}(\mathsf{sid})$.
       (b) Append to $\mathsf{extTrace}$ the query-answer pairs in $\mathsf{IllegitimateTrace}_{\mathsf{sid}}$ not already present.
       (c) Set $\mathsf{extTrace}' := ((x, y) \in \mathsf{extTrace} : \mathsf{GRO.IsProgrammed}((\mathsf{sid}, x)) = (\mathtt{IsProgrammed}, 0))$.
       (d) Compute $\mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace}')$.
       (e) If $(\mathbb{x}, \mathbb{w}) \notin R$, return Fail.
    8. Return $(\mathtt{Verification}, \mathsf{sid}, \mathbb{x}, \pi, 1)$.
• $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Corrupt}(P)$ from $\mathcal{S}$:
    1. Append $P$ to $\mathsf{Corrupted}$.
    2. If $P = M_P$, return $\mathsf{Random}_{\mathbf{P}}$.
    3. If $P = M_V$, return $\mathsf{Random}_{\mathbf{V}}$.
```

The ideal functionality $\mathcal{F}_{\mathrm{aARG}}$ has an instance bound n as one of its parameters, which later on will facilitate giving concrete security bounds. Moreover, $\mathcal{F}_{\mathrm{aARG}}$ is GRO-subroutine respecting, as it only interacts with GRO and with parties in the same session. Finally, in the verification procedure, $\mathcal{F}_{\mathrm{aARG}}$ invokes a straightline extractor $\mathbf{E}$ that receives as input a query-answer trace consisting of the ordered query-answer pairs resulting from queries to the GROM by the environment *and the simulator*, filtered to *exclude* queries whose answers were previously programmed by the environment. (In particular, the extractor $\mathbf{E}$ may receive queries to the random oracle that were previously programmed by the simulator.)

4.2 Protocol

A non-interactive argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM implies a corresponding protocol $\Pi_a[\mathsf{ARG}]$ in the GRO-hybrid UC framework, described below. $\Pi_a[\mathsf{ARG}]$ is a thin wrapper around ARG that uses the global random oracle with domain separation (using the $\mathsf{GRO}_{\mathsf{sid}}$ notation from Definition 3.10) to run the argument prover $\mathbf{P}$ and the argument verifier $\mathbf{V}$ of ARG. To disallow trivial breaks of knowledge soundness (such as those that the simulator for zero knowledge would allow), the verification algorithm checks whether any of the points queried are programmed.

Protocol 4.1. The protocol $\Pi_a[\mathsf{ARG}]$ for a session sid is defined as follows.
Parameters: A non-interactive argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$, an instance bound n.
Participants: A designated prover M_P and a designated verifier M_V.

- $\Pi_a[\mathsf{ARG}].\mathsf{Setup}()$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$: Do nothing.
- $\Pi_a[\mathsf{ARG}].\mathsf{Prove}(\mathbb{x}, \mathbb{w})$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$:
  1. Prover M_P
     (a) If $\mathsf{sid} \neq \mathsf{sid}_M$ or $|\mathbb{x}| > n$, return $\perp$.
     (b) If $(\mathbb{x}, \mathbb{w}) \notin R$, return $\perp$.
     (c) Compute $\pi \leftarrow \mathbf{P}^{\mathsf{GRO}_{\mathsf{sid}}}(\mathbb{x}, \mathbb{w})$.
     (d) Return $(\mathtt{Proof}, \mathsf{sid}, \mathbb{x}, \pi)$.
- $\Pi_a[\mathsf{ARG}].\mathsf{Verify}(\mathbb{x}, \pi)$ from $M = (\mathsf{pid}_M, \mathsf{sid}_M)$:
  1. Verifier M_V
     (a) If $\mathsf{sid} \neq \mathsf{sid}_M$ or $|\mathbb{x}| > n$, return $\perp$.
     (b) Get $b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{\mathsf{GRO}_{\mathsf{sid}}}(\mathbb{x}, \pi)$.
     (c) If for some $(x, y) \in \mathrm{tr}_{\mathbf{V}}$ $\mathsf{GRO}.\mathsf{IsProgrammed}((\mathsf{sid}, x)) = (\mathtt{IsProgrammed}, 1)$, then set $b := 0$.
     (d) Return $(\mathtt{Verification}, \mathsf{sid}, \mathbb{x}, \pi, b)$.
- $\Pi_a[\mathsf{ARG}].\mathsf{Corrupt}(M)$ from the adversary $\mathcal{A}$:
  1. If $M \notin \{ M_P, M_V \}$ return $\perp$.
  2. Return all the randomness of M, and relinquish control to the adversary.

$\Pi_a[\mathsf{ARG}]$ is GRO-subroutine respecting, because it interacts only with protocols in the same session and with GRO.

5 UC-friendly security notions for non-interactive arguments

We describe three security notions for a non-interactive argument $\mathsf{ARG} := (\mathbf{P}, \mathbf{V})$: a "UC-friendly" notion of completeness in Section 5.1; a "UC-friendly" notion of zero knowledge in Section 5.2; and a "UC-friendly" notion of knowledge soundness in Section 5.3. Later on in Section 6 we show that if a non-interactive argument ARG satisfies each of these security notions then $\Pi_a[\mathsf{ARG}]$ (Protocol 4.1) UC-realizes $\mathcal{F}_{\mathrm{aARG}}$ (Functionality 4.1) in the GRO-hybrid model; in fact, we show that these notions are necessary to achieve such goal. The latter two security notions are variants of those in [LR22b; LR22a], adapted to provide concrete security bounds and simplified when allowed by our setting.

Below we consider adversaries that can make multiple types of oracle queries: (1) random oracle queries; (2) programming queries; (3) prover queries; (4) verifier queries; and (5) corruption queries.

Definition 5.1. *An adversary is* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$**-query** *if it makes at most* t_{q} *random oracle queries,* t_{p} *programming queries,* ℓ_{p} *prover queries, a single prover corruption query, and a single verifier corruption query. An adversary is* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$**-query** *if it makes at most* t_{q} *random oracle queries,* t_{p} *programming queries,* ℓ_{p} *prover queries,* ℓ_{v} *verifier queries, a single prover corruption query, and a single verifier corruption query.*

Remark 5.2. As for the GRO, here and throughout the paper we allow the adversary to program the random oracle in "batches". Accordingly, we count a single query with batch tr as $|\mathrm{tr}|$ individual queries.

5.1 UC-friendly completeness

We introduce the notion of UC-friendly completeness. It models the capability of the adversary to induce the proving interface to generate proofs that do not verify successfully.

Definition 5.3. *For* $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$*, we define the* **UC-friendly completeness experiment** *as follows:*

```
$\mathsf{sUCCompleteness}^f(n, \mathcal{A})$:
1. Initialize empty lists $\mathrm{tr}, \mathsf{ProofList}, \mathsf{Random}_{\mathbf{P}}, \mathsf{Random}_{\mathbf{V}}$.
2. Set $\mathsf{advWin} := 0$.
3. Run $\mathcal{A}$ answering its queries as follows:
  – On a random oracle query $x$, set $y := f[\mathrm{tr}](x)$, append $(\mathsf{query}, x, y)$ to $\mathrm{tr}$, and return $y$.
  – On a programming query $\mathsf{trace}_{\mathsf{prog}}$:
    (a) If there exists $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}$ with $x_i = x$, return $0$.
    (b) Else append $((\mathsf{prog}, x, y))_{(x,y)\in\mathsf{trace}_{\mathsf{prog}}}$ to $\mathrm{tr}$ and return $1$.
  – On a prover query $(\mathbb{x}, \mathbb{w}) \in R$ with $|\mathbb{x}| \le n$:
    (a) Sample argument prover randomness $\rho_{\mathbf{P}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{P}}}$ and append it to $\mathsf{Random}_{\mathbf{P}}$.
    (b) Compute the argument string $\pi \xleftarrow{\mathrm{tr}_{\mathbf{P}}} \mathbf{P}^{f[\mathrm{tr}]}(\mathbb{x}, \mathbb{w}; \rho_{\mathbf{P}})$.
    (c) Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{P}}$.
    (d) Append $(\mathbb{x}, \pi)$ to $\mathsf{ProofList}$.
    (e) Return $\pi$.
  – On a verifier query $(\mathbb{x}, \pi)$:
    (a) Sample argument verifier randomness $\rho_{\mathbf{V}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ and append it to $\mathsf{Random}_{\mathbf{V}}$.
    (b) Compute the decision bit $b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi; \rho_{\mathbf{V}})$.
    (c) Set $\tilde{b} := b \wedge (\mathrm{tr}_{\mathbf{V}} \cap \mathsf{prog}(\mathrm{tr}) = \emptyset)$.
    (d) If $(\mathbb{x}, \pi) \in \mathsf{ProofList} \wedge \tilde{b} = 0$, set $\mathsf{advWin} := 1$.
    (e) Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{V}}$.
```

    (f) *Return* $\tilde{b}$.
  – *On a prover corruption query, return* $\mathsf{Random}_{\mathbf{P}}$ *(and do not answer further prover corruption and prover queries).*
  – *On a verifier corruption query, return* $\mathsf{Random}_{\mathbf{V}}$ *(and do not answer further verifier corruption and verifier queries).*
4. *Return* advWin.

ARG *has* **weak (resp. strong) UC-friendly completeness** *with error* ϵ_{ARG} *if, for every* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-query adversary* $\mathcal{A}$*, instance bound* n*, security parameter* λ*,*

$$\Pr\left[\mathsf{advWin} = 1 \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \mathsf{advWin} \leftarrow \mathsf{sUCCompleteness}^{f}(n, \mathcal{A}) \end{array}\right] \leq \epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \enspace .$$

We show that strong UC-friendly completeness is implied by natural notions that are typically satisfied by non-interactive arguments. We begin by recalling the definition of perfect completeness.

Definition 5.4. $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *has* **perfect completeness** *if, for every instance-witness pair* $(\mathbb{x}, \mathbb{w}) \in R$*,*

$$\Pr\left[\mathbf{V}^{f}(\mathbb{x}, \pi) = 1 \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \pi \leftarrow \mathbf{P}^{f}(\mathbb{x}, \mathbb{w}) \end{array}\right] = 1 \enspace .$$

A counterexample shows that perfect completeness is *insufficient* to achieve UC-friendly completeness.

Lemma 5.5. *Let* $n, \lambda \in \mathbb{N}$*. There exists a non-interactive argument* $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *with perfect completeness and UC-friendly completeness error* $\epsilon_{\mathrm{ARG}}(\lambda, n, 0, 1, 1, 1) = 1$.

Proof. Let R be a relation and consider the non-interactive argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ for R defined as follows:

- $\mathbf{P}^{f}(\mathbb{x}, \mathbb{w})$: return 0.
- $\mathbf{V}^{f}(\mathbb{x}, \pi)$: query $f(0)$, return 1.

ARG clearly satisfies perfect completeness. Next, consider the adversary against UC-friendly completeness that requests a proof from the prover oracle, programs the oracle f at 0, and request verification of the received proof. This adversary wins the UC-friendly completeness game with probability 1, using only one query to the programming oracle, one to the proving oracle, and one to the verification oracle. $\square$

The previous counterexample is rather artificial, as typically non-interactive arguments do not have verifiers that perform spurious queries to the random oracle. In fact, non-interactive arguments typically satisfy the property of monotone proofs, which we define next, and which disallows the previous counterexample. Informally, the property states that while verifying a proof the verifier queries the random oracle only at points that were previously queried by the prover.

Definition 5.6. $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *has* **monotone proofs** *if, for every* $(\mathbb{x}, \mathbb{w}) \in R$ *and adversary* $\mathcal{A}$*,*

$$\Pr\left[\mathrm{tr}_{\mathbf{V}} \subseteq \mathrm{tr}_{\mathbf{P}} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \pi \xleftarrow{\mathrm{tr}_{\mathbf{P}}} \mathbf{P}^{f}(\mathbb{x}, \mathbb{w}) \\ \perp \xleftarrow{\mathrm{tr}} \mathcal{A}^{[\![f, \mathrm{tr}_{\mathbf{P}}]\!]} \\ b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi) \end{array}\right] = 1 \enspace ,$$

where the inclusion $\mathrm{tr}_{\mathbf{V}} \subseteq \mathrm{tr}_{\mathbf{P}}$ *interprets the lists as sets.*

Perfect completeness and monotone proofs are still not sufficient, as the following counterexample shows.

Lemma 5.7. *Let $n, \lambda \in \mathbb{N}$. There exists a non-interactive argument* $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *with perfect completeness, monotone proofs, and UC-friendly completeness error* $\epsilon_{\mathrm{ARG}}(\lambda, n, 0, 1, 1, 1) = 1$.

Proof. Let R be a relation and consider the non-interactive argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ for R defined as follows:

- $\mathbf{P}^{f}(\mathbb{x}, \mathbb{w})$: query $f(0)$, return 0.
- $\mathbf{V}^{f}(\mathbb{x}, \pi)$: query $f(0)$, return 1.

ARG clearly satisfies perfect completeness, and has monotone proofs. Next, consider the adversary against UC-friendly completeness that programs the oracle f at 0, requests a proof from the prover oracle, and request verification of the received proof. Again, this adversary wins the UC-friendly completeness game with probability 1, using only one query to the programming oracle, one to the prover, and one to the verifier. $\square$

The above counterexample shows that if the adversary can predict which points the prover will query when generating a proof then there is an attack on UC-friendly completeness. This in particular shows that any (non-trivial) non-interactive argument with a deterministic prover is not UC-friendly complete. However, typical (zero knowledge) non-interactive arguments can be shown to satisfy a property that disallows such attacks. We dub this property unpredictable queries, defined next.

Definition 5.8. $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *has* **unpredictable queries with error** $\epsilon_{\mathbf{P}}$ *if, and every* $(t_{\mathsf{q}}, t_{\mathsf{p}})$*-query adversary* $\mathcal{A}$*, security parameter* λ*, and instance bound* n*:*

$$\Pr\left[\begin{array}{l|l} |\mathbb{x}| \leq n & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\ (\mathbb{x}, \mathbb{w}) \in R & (\mathbb{x}, \mathbb{w}) \xleftarrow{\mathrm{tr}} \mathcal{A}^{[\![f]\!]} \\ \wedge\ \mathsf{prog}(\mathrm{tr}) \cap \mathrm{tr}_{\mathbf{P}} \neq \emptyset & \pi \xleftarrow{\mathrm{tr}_{\mathbf{P}}} \mathbf{P}^{f[\mathrm{tr}]}(\mathbb{x}, \mathbb{w}) \end{array}\right] \leq \epsilon_{\mathbf{P}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}) \ .$$

Perfect completeness, monotone proofs, and unpredictable queries all imply UC-friendly completeness.

Lemma 5.9. *If* $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *has perfect completeness (Definition 5.4), monotone proofs (Definition 5.6), and unpredictable queries with error* $\epsilon_{\mathbf{P}}$ *(Definition 5.8), then* ARG *has strong UC-friendly completeness (Definition 5.3) with error*

$$\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \ell_{\mathsf{p}} \cdot \epsilon_{\mathbf{P}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{p}} \cdot \mathsf{q}_{\mathbf{P}}(n) + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}}) \ .$$

Proof. Let $\mathcal{A}$ be an adversary against the strong UC-friendly completeness security game. We construct an adversary against the unpredictable queries game.

```
$\mathcal{B}^{[\![f]\!]}(\mathcal{A})$:
1. Initialize empty lists advProg, Random$_{\mathbf{P}}$, Random$_{\mathbf{V}}$.
2. Sample $\tilde{i} \leftarrow [\ell_{\mathsf{p}}]$.
3. Run $\mathcal{A}$, answering its queries as follows:
   – Forward random oracle queries to the random oracle.
   – Forward programming queries to the programming oracle, appending the queries to advProg if the
     programming succeeds.
   – On the $i$-th prover query $(\mathbb{x}_i, \mathbb{w}_i) \in R$ with $|\mathbb{x}_i| \leq n$:
      (a) If $i = \tilde{i}$: output $(\mathbb{x}_i, \mathbb{w}_i)$ and terminate.
```

  (b) Sample $\rho_{\mathbf{P}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{P}}}$ and add it to $\mathsf{Random}_{\mathbf{P}}$.
  (c) Compute $\pi_i \leftarrow \mathbf{P}^f(\mathbb{x}_i, \mathbb{w}_i; \rho_{\mathbf{P}})$.
  (d) Return π_i.
- On a verifier query $(\mathbb{x}, \pi)$ with $|\mathbb{x}| \leq n$:
  (a) Sample $\rho_{\mathbf{V}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ and append it to $\mathsf{Random}_{\mathbf{V}}$.
  (b) Run $b \overset{\mathrm{tr}}{\leftarrow} \mathbf{V}^f(\mathbb{x}, \pi; \rho_{\mathbf{V}})$. If any of the points queried by $\mathbf{V}$ are in advProg, return 0 to $\mathcal{A}$, else return b.
- On a prover corruption query, return $\mathsf{Random}_{\mathbf{P}}$ and stop answering further prover or prover corruption queries.
- On a verifier corruption query, return $\mathsf{Random}_{\mathbf{V}}$ and stop answering further verifier or verifier corruption queries.

Whenever $\mathcal{A}$ wins the UC-friendly completeness game, advWin is set. This implies that there is at least a proof $(\mathbb{x}_i, \pi_i) \in$ ProofList did not verify successfully. This can happen if either the argument verifier rejects (which cannot occur by perfect completeness) or if the verification interface queries a point that was previously programmed. Since ARG has monotone proofs, this implies that the proving algorithm must have queried some programmed points. By a standard hybrid argument, we learn that $\epsilon_{\mathrm{ARG}} \leq \ell_{\mathsf{p}} \cdot \epsilon_{\mathbf{P}}$. The adversary $\mathcal{B}$ performs the same number of queries to the random oracle as $\mathcal{A}$, if not for the costs of simulating the proof and verification oracle, which are $\ell_{\mathsf{p}} \cdot \mathsf{q}_{\mathbf{P}}$ and $\ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}$ queries respectively. □

5.2 UC-friendly zero knowledge

We describe a "UC-friendly" notion of zero knowledge for a non-interactive argument. The definition is a natural extension of adaptive zero knowledge in the ROM, in which the adversary can additionally program the oracle. We additionally consider a stronger version, in which the adversary can ask (once only) for the randomness that the argument prover used to construct argument strings so far.

Definition 5.10. *Let* ARG $= (\mathbf{P}, \mathbf{V})$ *be a non-interactive argument, and let* $\mathbf{S}$ *be an (oracle) algorithm. We define two security games* sUCZeroKnowledge$_0$ *and* sUCZeroKnowledge$_1^{\mathbf{S}}$.

```
sUCZeroKnowledge$_0(\lambda, n, \mathcal{A})$:
1. Sample $f \leftarrow \mathcal{U}(\lambda)$.
2. Initialize empty lists $\mathrm{tr}$, ProofList, $\mathsf{Random}_{\mathbf{P}}$, $\mathsf{Random}_{\mathbf{V}}$.
3. Run $\mathcal{A}$, answering each query as follows.
   – On a random oracle query $x$, set $y := f[\mathrm{tr}](x)$, append $(\mathsf{query}, x, y)$ to $\mathrm{tr}$, and return $y$.
   – On a programming query $\mathsf{trace}_{\mathsf{prog}}$:
      (a) If there exists $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}$ with $x_i = x$, return $0$.
      (b) Else append $((\mathsf{prog}, x, y))_{(x,y)\in\mathsf{trace}_{\mathsf{prog}}}$ to $\mathrm{tr}$ and return $1$.
   – On a prover query $(\mathbb{x}, \mathbb{w}) \in R$ with $|\mathbb{x}| \leq n$:
      (a) Sample argument prover randomness $\rho_{\mathbf{P}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{P}}}$ and append it to $\mathsf{Random}_{\mathbf{P}}$.
      (b) Compute the argument string $\pi \overset{\mathrm{tr}_{\mathbf{P}}}{\longleftarrow} \mathbf{P}^{f[\mathrm{tr}]}(\mathbb{x}, \mathbb{w}; \rho_{\mathbf{P}})$.
      (c) Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{P}}$.
      (d) Append $(\mathbb{x}, \pi)$ to ProofList.
      (e) Return $\pi$.
   – On a verifier query $(\mathbb{x}, \pi) \in R$ with $|\mathbb{x}| \leq n$:
      (a) Sample argument verifier randomness $\rho_{\mathbf{V}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ and append it to $\mathsf{Random}_{\mathbf{V}}$.
      (b) Compute the decision bit $b \overset{\mathrm{tr}_{\mathbf{V}}}{\longleftarrow} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi; \rho_{\mathbf{V}})$.
```

```
        (c) Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{V}}$.
        (d) If $(\mathbb{x}, \pi) \in \mathsf{ProofList}$, return $1$.
        (e) Return $b \wedge (\mathrm{tr}_{\mathbf{V}} \cap \mathsf{prog}(\mathrm{tr}) = \emptyset)$.
    – On a prover corruption query, return $\mathsf{Random}_{\mathbf{P}}$. (Refuse further prover or prover corruption queries.)
    – On a verifier corruption query, return $\mathsf{Random}_{\mathbf{V}}$. (Refuse further verifier or verifier corruption
      queries.)
4. Output $\mathcal{A}$'s output.
```

```
$\mathsf{sUCZeroKnowledge}_1^{\mathbf{S}}(\lambda, n, \mathcal{A})$:
1. Sample $f \leftarrow \mathcal{U}(\lambda)$.
2. Initialize empty lists $\mathrm{tr}, \mathsf{advProg}, \mathsf{Random}_{\mathbf{P}}, \mathsf{Random}_{\mathbf{V}}$.
3. Run $\mathcal{A}$, answering each query as follows:
    – On a random oracle query $x$, set $y := f[\mathrm{tr}](x)$, append $(\mathsf{query}, x, y)$ to $\mathrm{tr}$, and return $y$.
    – On a programming query $\mathsf{trace}_{\mathsf{prog}}$:
        (a) If there exists $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}$ with $x_i = x$, return $0$.
        (b) Else append $((\mathsf{prog}, x, y))_{(x,y) \in \mathsf{trace}_{\mathsf{prog}}}$ to $\mathrm{tr}$ and $\mathsf{advProg}$ and return $1$.
    – On a prover query $(\mathbb{x}, \mathbb{w}) \in R$ with $|\mathbb{x}| \le n$:
        (a) Compute $(\pi, \mathrm{tr}', z_\pi) \xleftarrow{\mathrm{tr}_{\mathbf{S}}} \mathbf{S}^{f[\mathrm{tr}]}(\mathbb{x})$.
        (b) Compute $(\rho_{\mathbf{P}}, \mathrm{tr}'') \xleftarrow{\mathrm{tr}'_{\mathbf{S}}} \mathbf{S}^{f[\mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}}]}(\mathbb{w}, z_\pi)$.
        (c) If $\mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}} \circ \mathrm{tr}' \circ \mathrm{tr}''$ is invalid, return $\bot$.
        (d) Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}} \circ \mathrm{tr}' \circ \mathrm{tr}''$.
        (e) Append $\rho_{\mathbf{P}}$ to $\mathsf{Random}_{\mathbf{P}}$.
        (f) Return $\pi$.
    – On a verifier query $(\mathbb{x}, \pi) \in R$ with $|\mathbb{x}| \le n$:
        (a) Sample argument verifier randomness $\rho_{\mathbf{V}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ and append it to $\mathsf{Random}_{\mathbf{V}}$.
        (b) Compute the decision bit $b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi; \rho_{\mathbf{V}})$.
        (c) Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{V}}$.
        (d) If $(\mathbb{x}, \pi) \in \mathsf{ProofList}$, return $1$.
        (e) Return $b \wedge (\mathrm{tr}_{\mathbf{V}} \cap \mathsf{advProg} = \emptyset)$.
    – On a prover corruption query, return $\mathsf{Random}_{\mathbf{P}}$. (Do not answer further prover or prover corruption
      queries.)
    – On a verifier corruption query, return $\mathsf{Random}_{\mathbf{V}}$. (Do not answer further verifier or verifier corruption
      queries.)
4. Output $\mathcal{A}$'s output.
```

ARG *has* **weak (resp. strong) UC-friendly zero knowledge with error** ζ_{ARG} *if there exists a probabilistic polynomial-time algorithm* $\mathbf{S}$ *such that for every security parameter* λ*, instance bound* n*, and every* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-query adversary* $\mathcal{A}$

$$\Delta_{(\lambda,n,\mathcal{A})}\left(\mathsf{sUCZeroKnowledge}_0, \mathsf{sUCZeroKnowledge}_1^{\mathbf{S}}\right) \le \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \ .$$

We define a simplified notion of zero knowledge, which suffices to imply UC-friendly zero knowledge.

Definition 5.11. *Let* $\mathsf{sUCZeroKnowledgeSimple}_0$, $\mathsf{sUCZeroKnowledgeSimple}_1$ *be identical to* $\mathsf{sUCZeroKnowledge}_0$, $\mathsf{sUCZeroKnowledge}_1$*, with the verification and verification corruption oracle removed.* ARG *has* **weak (resp. strong) simplified UC-friendly zero knowledge with error** ζ_{simple} *if there exists a probabilistic polynomial-time algorithm* $\mathbf{S}$ *such that for every security parameter* λ*, instance bound* n *and every* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$*-query* *adversary $\mathcal{A}$*

$$\Delta_{(\lambda,n,\mathcal{A})}(\mathsf{sUCZeroKnowledgeSimple}_0, \mathsf{sUCZeroKnowledgeSimple}_1^{\mathbf{S}}) \leq \zeta_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) \enspace .$$

Lemma 5.12. *Suppose that* ARG *has weak (resp. strong) simplified UC-friendly zero knowledge with error* ζ_{simple}*. Then* ARG *has weak (resp. strong) UC-friendly zero knowledge with error*

$$\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \zeta_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}}, \ell_{\mathsf{p}}) \enspace .$$

Proof. Let $\mathcal{A}$ be an adversary against simple UC-friendly zero knowledge. We construct an adversary against UC-friendly zero-knowledge.

```
B(A):
1. Initialize empty lists advProg, ProofList and Random_V.
2. Run A, answering queries as follows:
   – Forward random oracle and prover corruption queries to the corresponding oracles.
   – Forward prover queries to the corresponding oracle, appending the resulting instance-proof pair to
     ProofList.
   – Forward programming queries to the corresponding oracle, and, if the programming succeeds, add
     the queries to advProg.
   – On a verifier query $(\mathbb{x}, \pi)$ with $|\mathbb{x}| \leq n$:
      (a) Sample $\rho_{\mathbf{V}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ and append it to $\mathsf{Random}_{\mathbf{V}}$.
      (b) Compute $b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}(\mathbb{x}, \pi; \rho_{\mathbf{V}})$.
      (c) If $(\mathbb{x}, \pi) \in \mathsf{ProofList}$, answer 1.
      (d) Return $b \wedge (\mathrm{tr}_{\mathbf{V}} \cap \mathsf{advProg} = \emptyset)$.
3. Output whatever A outputs.
```

Note that $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in the UC-friendly zero knowledge game, and only performs an additional $\ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}$ queries. □

We reduce simple UC-friendly zero knowledge to a definition in which the adversary makes a single prover query.

Lemma 5.13. *Suppose that* ARG *satisfies a version of Definition 5.11 in which the adversary is allowed only a single query to the prover, with error* $\zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$.

Then ARG *satisfies Definition 5.11 against* ℓ_{p} *prover queries, with error*

$$\zeta_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) := \ell_{\mathsf{p}} \cdot \zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}} + \mathsf{so}^{(1)}_{\mathsf{q}}(n, \ell_{\mathsf{p}}), t_{\mathsf{p}} + \mathsf{so}^{(1)}_{\mathsf{p}}(n, \ell_{\mathsf{p}})) \enspace .$$

Above, $\mathsf{so}^{(1)}_{\mathsf{q}}(n, \ell_{\mathsf{p}}) := \ell_{\mathsf{p}} \cdot \max(\mathsf{q}_{\mathbf{P}}(n), 2\mathsf{q}_{\mathbf{S}}(n))$ *and* $\mathsf{so}^{(1)}_{\mathsf{p}}(n, \ell_{\mathsf{p}}) := 2\ell_{\mathsf{p}} \cdot \mathsf{p}_{\mathbf{S}}(n)$.

Proof. Consider a sequence of hybrid games $\mathsf{G}_0, \ldots, \mathsf{G}_{\ell_{\mathsf{p}}}$. In game G_i, the first i oracle calls to the prover are answered with the oracle of $\mathsf{sUCZeroKnowledgeSimple}_1^{\mathbf{S}}$ while the remaining calls are answered with the oracle of $\mathsf{sUCZeroKnowledgeSimple}_0$. Note that $\mathsf{G}_0 \equiv \mathsf{sUCZeroKnowledgeSimple}_0$ and $\mathsf{G}_{\ell_{\mathsf{p}}} \equiv \mathsf{sUCZeroKnowledgeSimple}_1^{\mathbf{S}}$. We show that

$$\Delta_{\mathcal{A}}(\mathsf{G}_i, \mathsf{G}_{i+1}) \leq \zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}} + \mathsf{so}^{(1)}_{\mathsf{q}}(n, \ell_{\mathsf{p}}), t_{\mathsf{p}} + \mathsf{so}^{(1)}_{\mathsf{p}}(n, \ell_{\mathsf{p}})) \enspace .$$

The lemma readily follows since $\Delta_{\mathcal{A}}(\mathsf{G}_0, \mathsf{G}_{\ell_{\mathsf{p}}}) \leq \sum_{i=0}^{\ell_{\mathsf{p}}-1} \Delta_{\mathcal{A}}(\mathsf{G}_i, \mathsf{G}_{i+1})$.

Let $\mathcal{A}$ be an adversary that aims to distinguish between G_i and G_{i+1}.

We construct an adversary $\mathcal{B}$ against the single prover query game as follows.

$\mathcal{B}(\mathcal{A})$:

1. Run the adversary $\mathcal{A}$, answering oracle queries as follows.
   - Forward queries to the random and programming oracles to the corresponding oracles provided by the game.
   - For prover queries:
     * For the first $i-1$ queries, simulate the oracle as in $\mathsf{sUCZeroKnowledgeSimple}_1^{\mathbf{S}}$ using the random and programming oracle of the game.
     * For the i-th query, use the prover oracle of the game.
     * For the remaining queries, simulate the oracle as in $\mathsf{sUCZeroKnowledgeSimple}_0$ using the random oracle of the game.
   - For the corruption oracle query:
     * For the first $i-1$ queries, output the simulated randomness as in $\mathsf{sUCZeroKnowledgeSimple}_1^{\mathbf{S}}$.
     * For the i-th query, use the randomness oracle of the challenger.
     * For the remaining queries, simulate the oracle as in $\mathsf{sUCZeroKnowledgeSimple}_0$ (which just involves revealing the randomness used).
2. Output $\mathcal{A}$'s output.

We tally the simulation costs that $\mathcal{B}$ incurs. Each of $\mathcal{A}$'s queries to the random and programming oracles translates to a single query to the corresponding game oracles, resulting in at most t_{q} random and t_{p} programming queries. In each of the first $(i-1)$ queries of $\mathcal{A}$ to the prover, $\mathcal{B}$ has to simulate the oracle in $\mathsf{sUCZeroKnowledgeSimple}_1^{\mathbf{S}}$, which involves $2\mathsf{q}_{\mathbf{S}}$ random oracle queries and $2\mathsf{p}_{\mathbf{S}}$ queries to the programming oracle. The i-th query is answered using a single query to the prover of the game. Each of the remaining $\ell_{\mathsf{p}}-i+1$ prover queries instead involve simulating the oracle in $\mathsf{sUCZeroKnowledgeSimple}_0$, which requires $\mathsf{q}_{\mathbf{P}}$ random oracle queries. Finally, simulating the corruption oracle requires no further oracle queries.

Therefore $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$, making at most $t_{\mathsf{q}} + 2(i-1)\cdot \mathsf{q}_{\mathbf{S}} + (\ell_{\mathsf{p}}-i+1)\cdot \mathsf{q}_{\mathbf{P}}$ queries to the random oracle, $t_{\mathsf{p}} + 2(i-1)\cdot \mathsf{p}_{\mathbf{S}}$ queries to the programming oracle, 1 query to the prover, and querying only instances of size at most n. Hence,

$$\Delta_{\mathcal{A}}(\mathsf{G}_i, \mathsf{G}_{i+1}) \le \zeta^{(1)}_{\mathsf{simple}} \left(\begin{array}{l} \lambda, n, \\ t_{\mathsf{q}} + 2(i-1)\cdot \mathsf{q}_{\mathbf{S}} + (\ell_{\mathsf{p}}-i+1)\cdot \mathsf{q}_{\mathbf{P}}, \\ t_{\mathsf{p}} + 2(i-1)\cdot \mathsf{p}_{\mathbf{S}} \end{array} \right) .$$

Noting that $2(i-1)\cdot \mathsf{q}_{\mathbf{S}} + (\ell_{\mathsf{p}}-i+1)\cdot \mathsf{q}_{\mathbf{P}} \le \ell_{\mathsf{p}} \cdot \max(\mathsf{q}_{\mathbf{P}}, 2\mathsf{q}_{\mathbf{S}})$ and $2(i-1)\cdot \mathsf{p}_{\mathbf{S}} \le 2\ell_{\mathsf{p}} \cdot \mathsf{p}_{\mathbf{S}}$ concludes the proof. □

Comparison with adaptive ZK. By considering weak UC-friendly zero knowledge, and restricting the adversary to not make any programming queries, we recover the standard notion of multi-instance adaptive zero knowledge in the (explicitly programmable) ROM. Below we show that UC-friendly zero knowledge is, in fact, strictly stronger.

Lemma 5.14. *Let $k \in \mathbb{N}$. There exist a relation R_k and a non-interactive argument for R_k that:*

- *has multi-instance adaptive zero knowledge with error $\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, \ell_{\mathsf{p}}) = \frac{1}{2^\lambda}$.*
- *has UC-friendly zero knowledge error $\zeta_{\mathrm{ARG}}(\lambda, 1, 0, 1, 1, 0) \ge 1 - \frac{1}{2^k}$.*

Proof. Consider the (rather uninteresting) relation

$$R_k := \left\{ (\mathbb{x}, \mathbb{w}) : \begin{array}{l} \mathbb{x} = 0 \\ \mathbb{w} \in \{0,1\}^k \end{array} \right\} .$$

Here is an adaptive zero knowledge proof system for R_k with perfect completeness and perfect soundness:

$\mathbf{P}^f(\mathbb{x}, \mathbb{w})$: if $f(0) = 0^\lambda$, output $\mathbb{w}$; else output 0.
$\mathbf{V}^f(\mathbb{x}, \pi)$: check if $\mathbb{x} = 0$.

Perfect completeness and soundness are clear. It is straightforward to see that $(\mathbf{P}, \mathbf{V})$ is also adaptive zero knowledge: consider the simulator that outputs 0; conditioned on $f(0) \neq 0^\lambda$, this simulator perfectly simulates proofs, thus $(\mathbf{P}, \mathbf{V})$ has adaptive zero knowledge with error $\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, \ell_{\mathsf{p}}) := \frac{1}{2^\lambda}$. Next, consider the following adversary $\mathcal{A}$ against UC-friendly zero knowledge:

```
$\mathcal{A}$:
1. Sample $\mathbb{w} \leftarrow \{0,1\}^k$.
2. Query the programming oracle with $\mathsf{trace}_{\mathsf{prog}} := ((0, 0^\lambda))$.
3. Query the prover with $(0, \mathbb{w})$ to obtain $\pi$.
4. Output 1 if $\mathbb{w} = \pi$, 0 otherwise.
```

For every simulator $\mathbf{S}$,

$$\begin{aligned}
&\Delta_{(\lambda,n,\mathcal{A})}\left(\mathsf{sUCZeroKnowledge}_0, \mathsf{sUCZeroKnowledge}_1^{\mathbf{S}}\right) \\
&= \left|\Pr[\mathsf{sUCZeroKnowledge}_0(\mathcal{A}) = 1] - \Pr[\mathsf{sUCZeroKnowledge}_1^{\mathbf{S}}(\mathcal{A}) = 1]\right| \\
&= 1 - \Pr\left[\pi = \mathbb{w} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \mathbb{w} \leftarrow \{0,1\}^k \\ \mathrm{tr} := (\mathsf{prog}, 0, 0^\lambda) \\ \pi, \mathrm{tr}' \leftarrow \mathbf{S}^{f[\mathrm{tr}]}(0) \end{array}\right] \\
&\geq 1 - \frac{1}{2^k} \ .
\end{aligned}$$

The last line follows from the fact that $\mathbb{w}$ is hidden from $\mathbf{S}$. Thus, for every λ, the UC-friendly zero knowledge error is $\zeta_{\mathrm{ARG}}(\lambda, 1, 0, 1, 1, 0) \geq 1 - \frac{1}{2^k}$. □

Remark 5.15. Lemma 5.14 uses a trivial relation, without relying on any computational assumptions. The ideas in the proof can be modified to show that adaptive zero knowledge is strictly weaker than UC-friendly zero knowledge for any hard relation, yielding a separation for "interesting" relations as well.

5.3 UC-friendly knowledge soundness

We introduce a notion of UC-friendly straightline knowledge soundness, which is a strengthening of simulation knowledge soundness (extraction in the presence of a simulation oracle) where the adversary can additionally program the random oracle.

Definition 5.16. *Let* $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *be a non-interactive argument. We define the* **UC-friendly knowledge soundness game** *with respect to a simulator* $\mathbf{S}$ *and an extractor* $\mathbf{E}$ *as follows.*

```
$\mathsf{sUCKnowledgeSoundness}^f_{\mathbf{S},\mathbf{E}}(n, \mathcal{A})$ :
1. Initialize empty lists $\mathsf{InstanceList}, \mathsf{ProofList}, \mathrm{tr}, \mathsf{extTrace}, \mathsf{advProg}, \mathsf{Random}_{\mathbf{P}}, \mathsf{Random}_{\mathbf{V}}$.
2. Set $\mathsf{advWin} := 0$.
3. Run $\mathcal{A}$, answering its queries as follows:
   – On a random oracle query $x$, set $y := f[\mathrm{tr}](x)$, append $(\mathsf{query}, x, y)$ to $\mathrm{tr}, \mathsf{extTrace}$, and return $y$.
   – On a programming query $\mathsf{trace}_{\mathsf{prog}}$:
      (a) If there exists $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}$ with $x_i = x$, return 0.
```

```
        (b) Else append $((\mathsf{prog}, x, y))_{(x,y)\in\mathsf{trace}_{\mathsf{prog}}}$ to $\mathrm{tr}$ and $\mathsf{advProg}$ and return $1$.
    – On a prover query $(\mathbb{x}, \mathbb{w}) \in R$ with $|\mathbb{x}| \le n$:
        (a) Compute $(\pi, \mathrm{tr}', z_\pi) \xleftarrow{\mathrm{tr}_{\mathbf{S}}} \mathbf{S}^{f[\mathrm{tr}]}(\mathbb{x})$.
        (b) Compute $(\rho_{\mathbf{P}}, \mathrm{tr}'') \xleftarrow{\mathrm{tr}'_{\mathbf{S}}} \mathbf{S}^{f[\mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}}]}(\mathbb{w}, z_\pi)$.
        (c) If $\mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}} \circ \mathrm{tr}' \circ \mathrm{tr}''$ is invalid, return $\bot$.
        (d) Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}} \circ \mathrm{tr}' \circ \mathrm{tr}''$.
        (e) Set $\mathsf{extTrace} := \mathsf{extTrace} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}}$.
        (f) Append $\mathbb{x}$ to $\mathsf{InstanceList}$.
        (g) Append $(\mathbb{x}, \pi)$ to $\mathsf{ProofList}$.
        (h) Append $\rho_{\mathbf{P}}$ to $\mathsf{Random}_{\mathbf{P}}$.
        (i) Return $\pi$.
    – On a verifier query $(\mathbb{x}, \pi)$ with $|\mathbb{x}| \le n$:
        (a) Sample argument verifier randomness $\rho_{\mathbf{V}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ and append it to $\mathsf{Random}_{\mathbf{V}}$.
        (b) Compute the decision bit $b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi; \rho_{\mathbf{V}})$.
        (c) Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{V}}$.
        (d) If $(\mathbb{x}, \pi) \in \mathsf{ProofList}$ return $1$.
        (e) Set $\tilde{b} := b \wedge (\mathrm{tr}_{\mathbf{V}} \cap \mathsf{prog}(\mathsf{advProg}) = \emptyset)$.
        (f) Compute $\mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg})$.
        (g) If $\tilde{b} = 1 \wedge \mathbb{x} \notin \mathsf{InstanceList} \wedge (\mathbb{x}, \mathbb{w}) \notin R$, set $\mathsf{advWin} = 1$.
        (h) Return $\tilde{b}$.
    – On a prover corruption query, return $\mathsf{Random}_{\mathbf{P}}$. (Do not answer further prover or prover corruption queries.)
    – On a verifier corruption query, return $\mathsf{Random}_{\mathbf{V}}$. (Do not answer further verifier or verifier corruption queries.)
4. Return $\mathsf{advWin}$.
```

ARG *has* **weak (resp. strong) UC-friendly knowledge soundness** *with respect to a simulator* $\mathbf{S}$ *with error* κ_{ARG} *if there exists a probabilistic polynomial-time extractor* $\mathbf{E}$ *such that, for every* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-query adversary* $\mathcal{A}$*,*

$$\Pr\left[\mathsf{advWin} = 1 \,\middle|\, \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \mathsf{advWin} \leftarrow \mathsf{sUCKnowledgeSoundness}^{f}_{\mathbf{S},\mathbf{E}}(n, \mathcal{A}) \end{array}\right] \le \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \enspace .$$

We define a single-instance version of the above game, with slightly different notation for convenience. In particular, we allow the adversary a single query to the verification oracle, and additionally refactor the conditions for the adversary's win to be outside of the game's main body.

Definition 5.17. *Let* $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *be a non-interactive argument. We define the* **single-instance UC-friendly knowledge soundness game** *with respect to a simulator* $\mathbf{S}$ *as follows.*

```
$\mathsf{sUCKnowledgeSoundness1}^{f}_{\mathbf{S}}(n, \mathcal{A})$:
1. Initialize empty lists $\mathsf{InstanceList}, \mathrm{tr}, \mathsf{extTrace}, \mathsf{advProg}, \mathsf{Random}_{\mathbf{P}}$.
2. Set $\mathsf{advWin} = 0$.
3. Run $\mathcal{A}$, answering its queries as follows:
    – On a random oracle query $x$, set $y := f[\mathrm{tr}](x)$, append $(\mathsf{query}, x, y)$ to $\mathrm{tr}, \mathsf{extTrace}$, and return $y$.
    – On a programming query $\mathsf{trace}_{\mathsf{prog}}$:
```

(a) *If there exists* $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ *and* $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}$ *with* $x_i = x$*, return* 0.
(b) *Else append* $((\mathsf{prog}, x, y))_{(x,y)\in\mathsf{trace}_{\mathsf{prog}}}$ *to* tr *and* $\mathsf{advProg}$ *and return* 1.

- *On a prover query* $(\mathbb{x}, \mathbb{w}) \in R$ *with* $|\mathbb{x}| \le n$*:*
  (a) *Compute* $(\pi, \mathrm{tr}', z_\pi) \xleftarrow{\mathrm{tr}_{\mathbf{S}}} \mathbf{S}^{f[\mathrm{tr}]}(\mathbb{x})$.
  (b) *Compute* $(\rho_{\mathbf{P}}, \mathrm{tr}'') \xleftarrow{\mathrm{tr}'_{\mathbf{S}}} \mathbf{S}^{f[\mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}}]}(\mathbb{w}, z_\pi)$.
  (c) *If* $\mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}} \circ \mathrm{tr}' \circ \mathrm{tr}''$ *is invalid, return* $\bot$.
  (d) *Set* $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}} \circ \mathrm{tr}' \circ \mathrm{tr}''$.
  (e) *Set* $\mathsf{extTrace} := \mathsf{extTrace} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}}$.
  (f) *Append* $\mathbb{x}$ *to* $\mathsf{InstanceList}$.
  (g) *Append* $\rho_{\mathbf{P}}$ *to* $\mathsf{Random}_{\mathbf{P}}$.
  (h) *Return* π.
- *On a corruption query, return* $\mathsf{Random}_{\mathbf{P}}$. *(Do not answer further prover or corruption queries.)*

4. $\mathcal{A}$ *outputs* $(\mathbb{x}, \pi)$.
5. *Return* $(\mathbb{x}, \pi, \mathsf{InstanceList}, \mathsf{extTrace}, \mathsf{advProg})$.

ARG *has* **weak (resp. strong) single-instance UC-friendly knowledge soundness** *with respect to a simulator* $\mathbf{S}$ *with error* $\kappa^{(1)}_{\mathrm{ARG}}$ *if there exists a probabilistic polynomial-time extractor* $\mathbf{E}$ *such that, for every* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$*-query adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{l|l} (\mathbb{x}, \mathbb{w}) \notin R & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, b = 1 & \begin{pmatrix} \mathbb{x}, \pi, \\ \mathsf{InstanceList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{pmatrix} \xleftarrow{\mathrm{tr}} \mathsf{sUCKnowledgeSoundness1}^{f}_{\mathbf{S}}(n, \mathcal{A}) \\ \wedge\, \mathrm{tr}_{\mathbf{V}} \cap \mathsf{advProg} = \emptyset & b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi) \\ \wedge\, \mathbb{x} \notin \mathsf{InstanceList} & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array}\right] \le \kappa^{(1)}_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) \enspace .$$

A hybrid argument shows that UC-friendly knowledge soundness is implied by this weaker notion with the error growing by a multiplicative factor of ℓ_{v}.

Lemma 5.18. *If* $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ *has weak (resp. strong) single-instance UC-friendly knowledge soundness (Definition 5.17) with error* $\kappa^{(1)}_{\mathrm{ARG}}$*, then* ARG *has weak (resp. strong) UC-friendly knowledge soundness (Definition 5.16) with error*

$$\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \le \ell_{\mathsf{v}} \cdot \kappa^{(1)}_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}}, \ell_{\mathsf{p}}) \enspace .$$

Proof. Let $\mathcal{A}$ be an adversary against "multi" UC-friendly knowledge soundness. We construct a new adversary $\mathcal{B}$ against (single) UC-friendly knowledge soundness.

$\mathcal{B}(\mathcal{A})$:
1. Initialize empty lists ProofList, advProg, $\mathsf{Random}_{\mathbf{V}}$.
2. Sample $\tilde{i} \leftarrow [\ell_{\mathsf{v}}]$.
3. On a random oracle query, use the random oracle of the game.
4. On a programming query, use the programming oracle of the game, appending the query to advProg if it succeeds.
5. On a prover query, use the challenger's prover oracle, appending the resulting instance-proof pair to ProofList.

6. On a verifier query $(\mathbb{x}, \pi)$ with $|\mathbb{x}| \leq n$:
   (a) If this is the $\tilde{i}$-th query to the verification oracle, output $(\mathbb{x}, \pi)$ and terminate.
   (b) Sample $\rho_{\mathbf{V}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ and append it to $\mathsf{Random}_{\mathbf{V}}$.
   (c) Compute $b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi; \rho_{\mathbf{V}})$ (using the random oracle of the game).
   (d) If $(\mathbb{x}, \pi) \in \mathsf{ProofList}$ answer 1.
   (e) Return $b \wedge (\mathrm{tr}_{\mathbf{V}} \cap \mathsf{advProg} = \emptyset)$.
7. On a prover corruption query, use the challenger's corruption oracle.
8. On a verifier corruption query return $\mathsf{Random}_{\mathbf{V}}$. (Do not answer further verifier or verifier corruption queries.)

The new adversary $\mathcal{B}$ makes a single query to the verifier oracle, $t_{\mathsf{q}} + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}$ random oracle queries, t_{p} programming queries, and ℓ_{p} prover queries. The view of $\mathcal{A}$, until the $\tilde{i}$ verifier query is performed, is as in the multi-instance version of the game. To see this, note that the random, programming, prover, prover corruption queries are directly forwarded to the single-instance game oracles, and are identical to the multi-instance game. For the first $\tilde{i}$ verifier queries, the reduction faithfully simulates the verifier and verifier corruption oracle. Furthermore, whenever $\mathcal{A}$ wins, there exists at least one index i where the advWin flag is set. Since $\tilde{i}$ is chosen uniformly at random, the results follows. $\square$

6 UC-secure zkSNARKs from UC-friendly security notions

We prove that if a non-interactive argument ARG satisfies the UC-friendly security notions of Section 5 then the corresponding protocol $\Pi_{a}[\mathsf{ARG}]$ (Protocol 4.1) UC-realizes the ideal functionality $\mathcal{F}_{\mathrm{aARG}}$ (Functionality 4.1) in the GRO-hybrid model.

As discussed in Section 3.2, we use budgets to account for the capabilities of the environment. We keep track of a budget tuple $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ representing respectively:

- t_{q}: query budget that can be spent on GRO.Query queries;
- t_{p}: programming budget that can be spent on GRO.Program queries;
- ℓ_{p}: prover budget that can be spent on $\Pi_{a}[\mathsf{ARG}]$.Prove (resp. $\mathcal{F}_{\mathrm{aARG}}$.Prove) queries;
- ℓ_{v}: verifier budget that can be spent on $\Pi_{a}[\mathsf{ARG}]$.Verify (resp. $\mathcal{F}_{\mathrm{aARG}}$.Verify) queries.

Theorem 6.1. *Let* ARG $= (\mathbf{P}, \mathbf{V})$ *be a non-interactive argument with the following properties:*

- *weak (resp. strong) UC-friendly completeness (Definition 5.3) with error* ϵ_{ARG}*;*
- *weak (resp. strong) UC-friendly zero knowledge (Definition 5.10) with error* ζ_{ARG} *and simulator* $\mathbf{S}$*;*
- *weak (resp. strong) UC-friendly knowledge soundness (Definition 5.16) with respect to* $\mathbf{S}$ *with error* κ_{ARG}.

Then (when all protocols are instantiated with security parameter λ *and instance size* n*)* $\Pi_{a}[\mathsf{ARG}]$ $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-UC-realizes* $\mathcal{F}_{\mathrm{aARG}}$ *in the* GRO*-hybrid model with no simulation overhead and error*

$$z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$$

where

$$z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \begin{array}{l} \quad\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \\ +\ \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \\ +\ \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \end{array} \quad .$$

As mentioned in the relevant sections, GRO is subroutine respecting and $\Pi_{a}[\mathsf{ARG}]$-regular, and $\Pi_{a}[\mathsf{ARG}]$ is GRO-subroutine respecting. Thus, we can apply [BCHTZ20, Prop 3.4] to conclude that the transcript established by the ITM instances in the execution of $\mathsf{M}[\Pi_{a}[\mathsf{ARG}], \mathsf{GRO}]$ is identical to that in an execution in the GRO-hybrid model. Thus, Theorem 6.1 implies that $\mathsf{M}[\Pi_{a}[\mathsf{ARG}], \mathsf{GRO}]$ UC-emulates $\mathsf{M}[\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{aARG}}}, \mathsf{GRO}]$ (with the same simulation error and overhead). Therefore, all preconditions of Theorem 3.8 are satisfied, and Corollary 6.2 readily follows.

Corollary 6.2. *Let:*

- M *be the manager protocol introduced in Theorem 3.8;*
- ARG *be a non-interactive argument as in Theorem 6.1;*
- ρ *be* $(\Pi_{a}[\mathsf{ARG}], P)$*-compliant protocol for* $P \in \{\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{aARG}}}, \mathsf{M}[\Pi_{a}[\mathsf{ARG}], \mathsf{GRO}], \mathsf{M}[\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{aARG}}}, \mathsf{GRO}]\}$*;*
- $\tilde{\rho} := \mathsf{UC}(\rho, \Pi_{a}[\mathsf{ARG}], \mathsf{IDEAL}_{\mathcal{F}_{\mathrm{aARG}}})$ *where* UC *is the UC operator.*

Then, $\tilde{\rho}$ $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-UC-emulates* ρ *with no simulation overhead and simulation error*

$$t_{\pi}(\rho, \lambda) \cdot z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \ .$$

In the above:

- $z_{\mathrm{UC}}, \epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}$ *are defined as in Theorem 6.1; and*
- $t_{\pi}(\rho, \lambda)$ *bounds the number of instances of* $\Pi_{a}[\mathsf{ARG}]$ *that* ρ *spawns when parametrized with security parameter* λ.

6.1 Proof of Theorem 6.1

Let $\mathbf{E}$ be the extractor guaranteed by Definition 5.16, and let M_P, M_V denote, respectively, the prover and verifier party in the UC-security experiment. The UC simulator $\mathcal{S}$ is defined as follows.

$\mathcal{S}$:
1. Initialize an empty list advProg.
2. When $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Setup}$ asks for a tuple of algorithms by sending $(\mathtt{Setup}, \mathsf{sid})$, send $\mathsf{algTuple} := (\mathbf{V}, \mathbf{S}, \mathbf{E})$.
3. When any corrupted party issues a GRO.Program query, forward the query to GRO, and, if successful, append the list of programmed query-answer pairs to advProg.
4. When any corrupted party issues a GRO.IsProgrammed query, if the point is in advProg, answer $(\mathtt{IsProgrammed}, 1)$, otherwise answer with $(\mathtt{IsProgrammed}, 0)$.
5. When the adversary asks to corrupt M_P:
   (a) Call $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Corrupt}(M_P)$ which returns a list of randomness $\mathsf{Random}_{\mathbf{P}}$.
   (b) Return $\mathsf{Random}_{\mathbf{P}}$ to the adversary, and relinquish control of M_P.
6. When the adversary asks to corrupt M_V:
   (a) Call $\mathcal{F}_{\mathrm{aARG}}.\mathsf{Corrupt}(M_V)$ which returns a list of randomnesses $\mathsf{Random}_{\mathbf{V}}$.
   (b) Return $\mathsf{Random}_{\mathbf{V}}$ to the adversary, and relinquish control of M_V.

The simulator $\mathcal{S}$ can be implemented efficiently, and does not use any budget. We show security via a sequence of games (listed below); each game is played against an environment $\mathcal{E}$. We recall that in each game the environment has access to (i) a prover interface that outputs an argument string; (ii) a verifier interface that verifies arguments; (iii) two corruption interfaces (one for the prover party and one for the verifier party); and (iv) the global random oracle .

- $\mathsf{EXPA}(\mathcal{E}) \equiv \mathsf{EXEC}^{\mathsf{GRO}}_{\Pi_a[\mathsf{ARG}],\mathcal{A}_{\mathrm{D}},\mathcal{E}}(\lambda)$: The "real-world" security game in the GRO-hybrid model as in Definition 3.2.
- $\mathsf{EXPB}(\mathcal{E})$: Same as previous but answer false to GRO.IsProgrammed queries on any point not programmed by corrupted parties.
- $\mathsf{EXPC}(\mathcal{E})$: Modify the proving interface to maintain a list Proved of instance-proof pairs that it generated. Modify the verifier interface to accept proofs in that list by default. This is a relaxation of the verifier interface, as in the previous game honestly generated proofs can be rejected.
- $\mathsf{EXPD}(\mathcal{E})$:
  1. Modify the prover interface to match that of the ideal functionality.
     (a) Instead of generating proofs using $\mathbf{P}$, simulate proofs using $\mathbf{S}$, programming the GROM accordingly (outputting Fail if any such programming attempt fails). Further, use $\mathbf{S}$ to reconstruct prover randomness as in the ideal functionality.
     (b) Keep track of the points programmed by $\mathbf{S}$ in hProgrammed.
  2. Relax the check in Item 1c of the verifier interface to match that of the ideal functionality in Item 6 (if a proof verifies successfully and the only programmed points it queries are in hProgrammed, accept).
- $\mathsf{EXPE}(\mathcal{E})$: Modify the verifier interface by appending the extraction procedure of the ideal functionality.
  1. After Item 6, if the check passes, obtain the list of illegitimate queries $\mathsf{IllegitimateTrace}_{\mathsf{sid}}$.
  2. Run $\mathbf{E}$ to obtain a witness $\mathbb{w}$, and output Fail if the witness is not valid for the instance.
- $\mathsf{EXPF}(\mathcal{E}) \equiv \mathsf{IDEAL}^{\mathsf{GRO}}_{\mathcal{F}_{\mathrm{ARG}},\mathcal{S},\mathcal{E}}(\lambda)$: The "ideal-world" security game in the GRO-hybrid model as in Definition 3.5.

We study each game hop separately. In each game hop (apart from the first), we define an adversary $\mathcal{B}(\mathcal{E})$

against some UC-friendly property described in Section 5. The adversary will be the same in each hop, so we describe it here to avoid duplication.

$\mathcal{B}(\mathcal{E})$:

1. Run the environment $\mathcal{E}$, answering its requests as follows.
   - For GRO queries (random oracle or programming) that do not have prefix sid, $\mathcal{B}$ (lazily) simulates a random oracle. In the rest of the description we assume that queries have prefix sid.
   - On a GRO query (sid, x), query x to the random oracle of the game to obtain y, then return $(\mathtt{Query}, y)$ to the environment.
   - On a GRO programming $\mathsf{trace}_{\mathsf{prog}}$, set $\mathsf{trace}'_{\mathsf{prog}} := ((x, y))_{((\mathsf{sid},x),y)\in\mathsf{trace}_{\mathsf{prog}}}$ and query $\mathsf{trace}'_{\mathsf{prog}}$ to the programming oracle of the game to obtain a bit b. Return $(\mathtt{IsProgrammed}, b)$ to the environment.
   - When the environment queries the prover interface with $(\mathbb{x}, \mathbb{w}) \in R$, forward the query to the prover of the game to obtain a proof π or a failure symbol $\bot$. If the result is $\bot$, return Fail, else return $(\mathtt{Proof}, \mathsf{sid}, \mathbb{x}, \pi)$ to the environment.
   - When the environment queries the verifier interface with $(\mathbb{x}, \pi)$, forward the query to the verifier of the game to obtain a bit b. Return $(\mathtt{Verification}, \mathsf{sid}, \mathbb{x}, \pi, b)$ to the environment.
   - When the environment asks to corrupt the prover, query the prover corruption oracle of the game and forward the result to the environment.
   - When the environment asks to corrupt the verifier, query the verifier corruption oracle of the game and forward the result to the environment.
2. Output whatever $\mathcal{E}$ outputs.

Note that $\mathcal{B}$ has the same query complexity of $\mathcal{E}$.

REAL is EXPB. We show that:

$$\mathsf{EXEC}^{\mathsf{GRO}}_{\Pi_a[\mathsf{ARG}],\mathcal{A}_{\mathrm{D}},\mathcal{E}}(\lambda) \equiv \mathsf{EXPA} \equiv \mathsf{EXPB} \ .$$

The argument is as in [CDGLN18]. Only parties in the session can ask $\mathsf{GRO}.\mathsf{IsProgrammed}$ queries, and in the "real-world" experiment no honest party makes programming queries. Thus, in both games, no programming (other than that the corrupted parties engage on) will occur, and all the queries to $\mathsf{GRO}.\mathsf{IsProgrammed}$ on those points would return false. Therefore, modifying the experiment to answer false to $\mathsf{GRO}.\mathsf{IsProgrammed}$ queries on any point not programmed by corrupted parties does not change the view of the environment.

EXPB is close to EXPC. We rely on UC-friendly completeness (Definition 5.3) to argue that:

$$\Delta_{\mathcal{E}}(\mathsf{EXPB}, \mathsf{EXPC}) \leq \epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \ .$$

The two games are identical, if not for the fact that in EXPC all (honestly) generated proofs are accepted, while in EXPB they might not be. $\mathcal{B}$ simulates perfectly the view of $\mathcal{E}$ in EXPB (as long as the advWin flag is not set) and in EXPC. Hence any distinguishing advantage of $\mathcal{E}$ translates directly into $\mathcal{B}$ winning the UC-friendly completeness game.

EXPC is close to EXPD. We rely on UC-friendly zero knowledge (Definition 5.10) to argue that:

$$\Delta_{\mathcal{E}}(\mathsf{EXPC}, \mathsf{EXPD}) \leq \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \ .$$

When $\mathcal{B}$ is in $\mathsf{sUCZeroKnowledge}_0$ the view of $\mathcal{E}$ is as in EXPB. Instead, when $\mathcal{B}$ is in $\mathsf{sUCZeroKnowledge}^{\mathrm{S}}_1$ the view of $\mathcal{E}$ is as in EXPC. Hence whenever $\mathcal{E}$ distinguishes between EXPB and EXPC, $\mathcal{B}$ distinguishes between the real-world and ideal-world in the UC-friendly zero knowledge experiment.

EXPD is close to EXPE. We rely on UC-friendly knowledge soundness (Definition 5.16) to argue that:

$$\Delta_{\mathcal{E}}(\mathsf{EXPD}, \mathsf{EXPE}) \leq \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \ .$$

The only (detectable) difference between the two experiments is that in EXPE the verifier interface can output Fail if extraction fails, while this does not happen in EXPD. This is because in EXPE the verification interface attempts to extract a valid witness, and outputs Fail if this extraction fails, and apart from this difference the two games are identical. In light of the above, the experiments are identical until Fail is output, and since Fail is output exactly when $\mathsf{advWin} = 1$ in the UC-friendly knowledge soundness game, any distinguishing advantage of $\mathcal{E}$ directly translates to $\mathcal{B}$ winning the UC-friendly knowledge soundness game. Note in particular that in both the verification interface of the ideal functionality and the verifier oracle of the UC-friendly knowledge soundness game, the extractor has access to a trace consisting of both the adversary random oracle query and the queries the proving interface made to the random oracle, both filtered to exclude adversarially programmed queries.

EXPE is IDEAL. Since the two games are syntactically equal, we have that:

$$\mathsf{EXPE} \equiv \mathsf{EXPF} \equiv \mathsf{IDEAL}^{\mathsf{GRO}}_{\mathcal{F}_{\mathrm{ARG}},\mathcal{S},\mathcal{E}}(\lambda)$$

6.2 Definitions 5.3, 5.10 and 5.16 are necessary

We show that the UC-friendly security notions in Section 5 are necessary for the UC-security of $\Pi_a[\mathsf{ARG}]$ in the GROM. In Lemmas 6.3 to 6.5 below, we lift an adversary $\mathcal{A}$ against the UC-friendly security notion to an environment $\mathcal{E}(\mathcal{A})$ against the UC-security of $\Pi_a[\mathsf{ARG}]$ in the GROM. The environment for each lemma can be described starting from the same basic template, which we present next.

$\mathcal{E}_0(\mathcal{A})$:
1. Spawn a single instance of the protocol (say with session ID sid).
2. Run $\mathcal{A}$, answering queries as follows.
   - On a random oracle query x, query $\mathsf{GRO.Query}((\mathsf{sid}, x))$ to obtain $(\mathtt{Query}, y)$ and return the answer y to $\mathcal{A}$.
   - On a programming query $\mathsf{trace}_{\mathsf{prog}}$, set $\mathsf{trace}'_{\mathsf{prog}} := (((\mathsf{sid}, x), y))_{(x,y)\in\mathsf{trace}_{\mathsf{prog}}}$, query $\mathsf{GRO.Program}(\mathsf{trace}'_{\mathsf{prog}})$ obtaining $(\mathtt{IsProgrammed}, b)$. Return b to $\mathcal{A}$.
   - On a prover query $(\mathbb{x}, \mathbb{w}) \in R$, make a query to the prover interface of the protocol. If the result is Fail, return $\perp$ to $\mathcal{A}$. If instead it is a message $(\mathtt{Proof}, \mathsf{sid}, \mathbb{x}, \pi)$, return π to $\mathcal{A}$.
   - On a verifier query $(\mathbb{x}, \pi)$, make a query to the verifier interface. If the result is a message $(\mathtt{Verification}, \mathsf{sid}, \mathbb{x}, \pi, b)$, return b to $\mathcal{A}$. If instead it is Fail, return 1 to $\mathcal{A}$.
   - On a prover corruption query, corrupt the prover party in the session, and return the received randomness to $\mathcal{A}$.
   - On a verifier corruption query, corrupt the verifier party in the session, and return the received randomness to $\mathcal{A}$.

Note that the environment $\mathcal{E}_0$, on a verifier query, returns 1 to the adversary if the verifier returns Fail. This is because the only instance in which this occurs is when (in the ideal UC-security experiment) the ideal functionality successfully verifies a proof from which it is unable to extract a valid witness. In both the UC-friendly completeness and UC-friendly zero knowledge game this extraction is not part of the security experiment, while the successful verification is, so returning 1 is the intended behavior.

Further, $\mathcal{E}_0$ inherits the query complexity of $\mathcal{A}$.

Lemma 6.3. *1If* ARG *does not satisfy Definition 5.3 with error* ϵ_{ARG}*, for every simulator* $\mathcal{S}$ *there exists a* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-budget environment* $\mathcal{E}$ *such that*

$$\Delta_\lambda \left(\mathsf{EXEC}^{\mathsf{GRO}}_{\Pi_a[\mathsf{ARG}],\mathcal{A}_{\mathrm{D}},\mathcal{E}}, \mathsf{IDEAL}^{\mathsf{GRO}}_{\mathcal{F}_{\mathrm{a}}\mathrm{ARG},\mathcal{S},\mathcal{E}} \right) > \epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \enspace .$$

Proof. For every adversary $\mathcal{A}$ against the weak (resp. strong) UC-friendly completeness game, we construct an environment $\mathcal{E}$ by modifying the template environment $\mathcal{E}_0$ as follows.

```
E(A):
1. Initialize an empty list Proved.
2. Run $\mathcal{E}_0(\mathcal{A})$, additionally performing the following:
   – On a prover query, append the returned $(\mathbb{x}, \pi)$ pair to Proved.
   – On a verifier query, check if $(\mathbb{x}, \pi) \in \mathsf{Proved}$ and verification does not succeed. In that case, output 0
     and terminate.
3. When $\mathcal{E}_0$ halts, output 1.
```

By definition of the ideal functionality, in the ideal-world proofs that are returned by the prover interface are always accepted, so $\mathcal{E}$ always outputs 1.

In the real-world, $\mathcal{A}$ wins the UC-friendly completeness experiment exactly when it manages to set the advWin flag, which implies that it submitted an instance-proof pair $(\mathbb{x}, \mathbb{w}) \in \mathsf{ProofList}$ to the verification oracle, but verification of said proof did not succeed. When this occurs, $\mathcal{E}$ will output 0.

Thus, if we assume that $\mathcal{A}$ has advantage $> \epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ against the UC-friendly completeness game, the statistical distance of the two games is at least ϵ_{ARG}. □

Lemma 6.4. *If* ARG *does not satisfy Definition 5.10 with error* ζ_{ARG}*, for every simulator* $\mathcal{S}$ *there exists a* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-budget environment* $\mathcal{E}$ *such that*

$$\Delta_\lambda \left(\mathsf{EXEC}^{\mathsf{GRO}}_{\Pi_a[\mathsf{ARG}],\mathcal{A}_{\mathrm{D}},\mathcal{E}}, \mathsf{IDEAL}^{\mathsf{GRO}}_{\mathcal{F}_{\mathrm{a}}\mathrm{ARG},\mathcal{S},\mathcal{E}} \right) > \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \enspace .$$

Proof. For every adversary $\mathcal{A}$ against the weak (resp. strong) UC-friendly zero knowledge game, we construct an environment $\mathcal{E}$ by modifying the template environment $\mathcal{E}_0$ as follows.

$\mathcal{E}(\mathcal{A})$: Simulate $\mathcal{E}_0(\mathcal{A})$ outputting whatever $\mathcal{A}$ outputs when it halts.

Let $\mathcal{S}$ be any simulator for the UC-security experiment, and let $\mathbf{S}$ be the simulator that it passes to $\mathcal{F}_{\mathrm{a}}\mathrm{ARG}.\mathsf{Setup}$. By assumption, for this simulator $\mathbf{S}$, there exist an adversary $\mathcal{A}$ that makes at most t_{q} queries to its random oracle, t_{p} queries to the programming oracle, ℓ_{p} queries to its prover oracle, ℓ_{v} to its verifier oracle, a single query to either corruption oracle, and queries instances of size at most n such that

$$\Delta_{(\lambda,n,\mathcal{A})} \left(\mathsf{sUCZeroKnowledge}_0, \mathsf{sUCZeroKnowledge}_1^{\mathbf{S}} \right) > \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \enspace .$$

In the "real-world" security experiment the view of the $\mathcal{A}$ when simulated by $\mathcal{E}(\mathcal{A})$ is that in $\mathsf{sUCZeroKnowledge}_0$, while in the "ideal-world" game it is as in $\mathsf{sUCZeroKnowledge}_1^{\mathbf{S}}$. The resulting environment inherits the number of queries of the adversary. □

Lemma 6.5. *Let* $\mathbf{S}$ *be an algorithm. If* ARG *does not satisfy Definition 5.16 with respect to* $\mathbf{S}$ *with error* κ_{ARG}*, for every simulator* $\mathcal{S}$ *(that chooses* $\mathbf{S}$ *as simulation algorithm) there exists a* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-budget environment* $\mathcal{E}$ *such that*

$$\Delta_\lambda \left(\mathsf{EXEC}^{\mathsf{GRO}}_{\Pi_a[\mathsf{ARG}],\mathcal{A}_{\mathrm{D}},\mathcal{E}}, \mathsf{IDEAL}^{\mathsf{GRO}}_{\mathcal{F}_{\mathrm{a}}\mathrm{ARG},\mathcal{S},\mathcal{E}} \right) > \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \enspace .$$

Proof. For every adversary $\mathcal{A}$ against the weak (resp. strong) UC-friendly knowledge soundness game, we construct an environment $\mathcal{E}$ by modifying the template environment $\mathcal{E}_0$ as follows.

$\mathcal{E}(\mathcal{A})$:
1. Run $\mathcal{E}_0(\mathcal{A})$, additionally performing the following:
   - On a verifier query $(\mathbb{x}, \pi)$, if the verifier interface returns Fail output 1 and terminate.
2. When $\mathcal{A}$ halts, output 0.

By definition of the protocol, in the real-world proofs Fail is never returned, and so in that experiment $\mathcal{E}$ always outputs 1.

In the ideal-world, $\mathcal{A}$ wins the UC-friendly knowledge soundness experiment exactly when it manages to set the advWin flag, which implies that it submitted an instance-proof pair $(\mathbb{x}, \mathbb{w})$ to the verification oracle on which (i) verification succeeds; (ii) extraction fails; and (iii) which is fresh in the sense that the instance was not previously queried to the proving interface. In this case, the verification interface will return Fail, and $\mathcal{E}$ will output 0.

Thus, if we assume that $\mathcal{A}$ has advantage $> \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ against the UC-friendly knowledge soundness game, the statistical distance of the two games is at least κ_{ARG}. □

7 Merkle commitments and UC-security

The constructions of zkSNARKs that we study in this paper rely on Merkle commitment schemes [Mer89] in the ROM. We describe Merkle commitment schemes in Section 7.1 and then prove several UC-friendly properties that we rely on: in Section 7.2 we prove UC-friendly completeness; in Section 7.3 we prove UC-friendly hiding; and in Section 7.4 we prove UC-friendly extraction.

7.1 Merkle commitment schemes

We introduce some notation for binary trees with l leaves (assumed to be a power of 2).

- The depth of the tree is $\mathsf{d} := \log \mathsf{l}$.
- Vertices are identified with pairs $(j, i) \in [\mathsf{d}] \times [2^j]$. Odd nodes have i odd and even ones have i even.
- The root of the tree is $(0, 1)$.
- The path from a node (d, i) to the root is denoted as $\mathsf{path}(i)$ and we let $\mathsf{p}(j, i) \in \{j\} \times [2^j]$ be the node in the j-th layer of $\mathsf{path}(i)$.
- The copath from a node (d, i) to the root is denoted as $\mathsf{copath}(i)$, and we let $\bar{\mathsf{p}}(j, i) \in \{j\} \times [2^j]$ be the node in the j-th layer of $\mathsf{copath}(i)$.
- The span of a node (j, i) is denoted as $\mathsf{span}(j, i)$ and is the list of leaves at the subtree rooted at (j, i).

The Merkle commitment scheme $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r_{MT}}]$ over an alphabet $\Sigma \subseteq \{0, 1\}^*$ is defined as follows. Let $\mathsf{r_{MT.Commit}} := \mathsf{l} \cdot \mathsf{r_{MT}}$.

```
$\mathsf{MT.Commit}^f(\mathbf{m} \in \Sigma^{\mathsf{l}}; \rho_{\mathsf{MT}} \in \{0,1\}^{\mathsf{r_{MT.Commit}}})$
1. Parse $\rho_{\mathsf{MT}}$ as $(\rho_1, \ldots, \rho_{\mathsf{l}})$ with $\rho_i \in \{0,1\}^{\mathsf{r_{MT}}}$.
2. For $i \in [\mathsf{l}]$, set $c_{(\mathsf{d},i)} := f(m_i, \rho_i)$.
3. For $j = \mathsf{d} - 1, \ldots, 0$ (in this order) and $i \in [2^j]$: set $c_{(j,i)} := f(c_{(j+1,2i-1)}, c_{(j+1,2i)})$.
4. Set $\mathsf{rt} := c_{(0,1)}$.
5. Set $\mathsf{td} := (\mathbf{m}, (\rho_i)_{i \in [\mathsf{l}]}, (c_{(j,i)})_{j \in [0,\mathsf{d}], i \in [2^j]})$.
6. Output $(\mathsf{rt}, \mathsf{td})$.

$\mathsf{MT.Open}(\mathsf{td}, I \subseteq [\mathsf{l}])$
1. For $i \in I$, set $\mathsf{auth}_i := (\rho_i, (c_{\bar{\mathsf{p}}(j,i)})_{j \in [d]})$.
2. Output $\mathsf{pf} := (\mathsf{auth}_i)_{i \in I}$.

$\mathsf{MT.Check}^f(\mathsf{rt}, I \subseteq [\mathsf{l}], \mathbf{a} \in \Sigma^I, \mathsf{pf})$
1. For $i \in I$:
   (a) Set $c_{(\mathsf{d},i)} := f(\mathbf{a}[i], \rho_i)$.
   (b) For $j = \mathsf{d} - 1, \ldots, 0$:
         i. If $\mathsf{p}(j+1, i)$ is odd, set $c_L := \mathsf{p}(j+1, i)$ and $c_R := \bar{\mathsf{p}}(j+1, i)$
        ii. If $\mathsf{p}(j+1, i)$ is even, set $c_R := \mathsf{p}(j+1, i)$ and $c_L := \bar{\mathsf{p}}(j+1, i)$
       iii. Set $c_{\mathsf{p}(j,i)} := f((c_L, c_R))$.
   (c) Check that $c_{(0,1)} = \mathsf{rt}$.
```

We obtain the following query complexity bounds:

- The $\mathsf{MT.Commit}$ algorithm performs $\mathsf{q_{MT.Commit}}(\mathsf{l}) = 2\mathsf{l}$ queries,
- The $\mathsf{MT.Open}$ algorithm performs 0 queries,
- The $\mathsf{MT.Check}$ algorithm performs $\mathsf{q_{MT.Check}}(\mathsf{l}, q) \leq q \cdot \log \mathsf{l}$ queries.

7.2 UC-friendly completeness

We show that the Merkle commitment scheme satisfies notions of completeness that makes it compatible with UC-friendly completeness for non-interactive arguments (Definition 5.3).

First, the Merkle commitment scheme is well known to have perfect completeness.

Lemma 7.1. *Let* $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$. *For every message* $\mathbf{m} \in \Sigma^{\mathsf{l}}$ *and query set* $I \subseteq [\mathsf{l}]$

$$\Pr\left[\mathsf{MT.Check}^{f}(\mathsf{rt}, I, \mathbf{m}[I], \mathsf{pf}) = 1 \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MT.Commit}^{f}(\mathbf{m}) \\ \mathsf{pf} := \mathsf{MT.Open}(\mathsf{td}, I) \end{array} \right] = 1 \enspace .$$

Second, the checking algorithm of the Merkle commitment scheme is compatible with our notion of monotone proofs (Definition 5.6).

Lemma 7.2. *Let* $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$. *For every message* $\mathbf{m} \in \Sigma^{\mathsf{l}}$ *and query set* $I \subseteq [\mathsf{l}]$,

$$\Pr\left[\mathrm{tr}_{\mathsf{check}} \subseteq \mathrm{tr}_{\mathsf{commit}} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{rt}, \mathsf{td}) \xleftarrow{\mathrm{tr}_{\mathsf{commit}}} \mathsf{MT.Commit}^{f}(\mathbf{m}) \\ \mathsf{pf} := \mathsf{MT.Open}(\mathsf{td}, I) \\ \bot \xleftarrow{\mathrm{tr}} \mathcal{A}^{[\![f, \mathrm{tr}_{\mathsf{commit}}]\!]} \\ b \xleftarrow{\mathrm{tr}_{\mathsf{check}}} \mathsf{MT.Check}^{f[\mathrm{tr}]}(\mathsf{rt}, I, \mathbf{m}[I], \mathsf{pf}) \end{array} \right] = 1 \enspace .$$

Finally, the Merkle commitment scheme also satisfies a notion of unpredictable queries, making it compatible with Definition 5.8.

Lemma 7.3. *Let* $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$. *For every* $(t_{\mathsf{q}}, t_{\mathsf{p}})$*-query adversary* $\mathcal{A}$ *and security parameter* λ*:*

$$\Pr\left[\begin{array}{l} \mathbf{m} \in \Sigma^{\mathsf{l}} \\ \wedge\ \mathsf{prog}(\mathrm{tr}) \cap \mathrm{tr}_{\mathsf{commit}} \neq \emptyset \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \mathbf{m} \xleftarrow{\mathrm{tr}} \mathcal{A}^{[\![f]\!]} \\ (\mathsf{rt}, \mathsf{td}) \xleftarrow{\mathrm{tr}_{\mathsf{commit}}} \mathsf{MT.Commit}^{f[\mathrm{tr}]}(\mathbf{m}) \end{array} \right] \leq \epsilon_{\mathsf{MT}}(\lambda, \mathsf{l}, t_{\mathsf{q}}, t_{\mathsf{p}}) \enspace .$$

In the above, $\epsilon_{\mathsf{MT}}(\lambda, \mathsf{l}, t_{\mathsf{q}}, t_{\mathsf{p}}) := \mathsf{l} \cdot (t_{\mathsf{q}} + t_{\mathsf{p}}) \cdot \left(\frac{1}{2^{\mathsf{r}_{\mathsf{MT}}}} + \frac{1}{2^{\lambda}} \right)$.

Sketch. The proof is very similar to that in Lemma 7.6. The adversary wins exactly if it it is able to program a point before it is queried. Since leaf queries contain a uniformly random string sampled from $\{0,1\}^{\mathsf{r}_{\mathsf{MT}}}$, the probability that any of them is predicted is at most $\frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\mathsf{r}_{\mathsf{MT}}}}$. Conditioned on these points not being queried, their answers are strings sampled uniformly at random from $\{0,1\}^{\lambda}$, so each one of them can be predicted with probability at most $\frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\lambda}}$. Continuing layer-by-layer yields the claimed bound. (We remark that the above bound is most likely not tight, and we suspect a tighter bound would not depend on t_{q}. We leave tightening the bound for future work.) □

7.3 UC-friendly hiding

We describe a notion of UC-friendly hiding, and prove that Merkle commitment scheme satisfy it.

Definition 7.4. *Let* $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$. *We define two security experiments* $\mathsf{sUCMerkleHiding}_0$ *and* $\mathsf{sUCMerkleHiding}_1$.

```
$\mathsf{sUCMerkleHiding}_0(\mathcal{A})$:
1. Sample $f \leftarrow \mathcal{U}(\lambda)$.
2. Initialize empty lists $\mathrm{tr}$, $\mathsf{Random}_{\mathsf{MT}}$.
3. Run the adversary $\mathcal{A}$, answering each query as follows:
   (a) On a random oracle query $x$, set $y := f[\mathrm{tr}](x)$, append $(\mathsf{query}, x, y)$ to $\mathrm{tr}$, and return $y$.
   (b) On a programming query $\mathsf{trace}_{\mathsf{prog}}$:
       i. If there exists $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}$ with $x_i = x$, return $0$.
       ii. Else append $((\mathsf{prog}, x, y))_{(x,y)\in\mathsf{trace}_{\mathsf{prog}}}$ to $\mathrm{tr}$ and return $1$.
   (c) On a prover query $(\mathbf{m}, I)$ with $|\mathbf{m}| \leq \mathsf{l}$ and $|I| \leq q$:
       i. Sample $\rho_{\mathsf{MT}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathsf{MT.Commit}}}$.
       ii. Compute $(\mathsf{rt}, \mathsf{td}) \xleftarrow{\mathrm{tr}_{\mathsf{commit}}} \mathsf{MT.Commit}^{f[\mathrm{tr}]}(\mathbf{m}; \rho_{\mathsf{MT}})$.
       iii. Compute $\mathsf{pf} := \mathsf{MT.Open}(\mathsf{td}, I)$.
       iv. Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathsf{commit}}$.
       v. Append $\rho_{\mathsf{MT}}$ to $\mathsf{Random}_{\mathsf{MT}}$.
       vi. Answer $(\mathsf{rt}, \mathsf{pf})$ .
   (d) On a corruption query, return $\mathsf{Random}_{\mathsf{MT}}$. (Refuse further prover or corruption queries.)
4. Output $\mathcal{A}$'s output.
```

```
$\mathsf{sUCMerkleHiding}_1^{\mathsf{MT.Sim}}(\mathcal{A})$:
1. Sample $f \leftarrow \mathcal{U}(\lambda)$.
2. Initialize empty lists $\mathrm{tr}$, $\mathsf{Random}_{\mathsf{MT}}$.
3. Run the adversary $\mathcal{A}$, answering each query as follows:
   (a) On a random oracle query $x$, set $y := f[\mathrm{tr}](x)$, append $(\mathsf{query}, x, y)$ to $\mathrm{tr}$, and return $y$.
   (b) On a programming query $\mathsf{trace}_{\mathsf{prog}}$:
       i. If there exists $(x, y) \in \mathsf{trace}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}$ with $x_i = x$, return $0$.
       ii. Else append $((\mathsf{prog}, x, y))_{(x,y)\in\mathsf{trace}_{\mathsf{prog}}}$ to $\mathrm{tr}$ and return $1$.
   (c) On a prover query $(\mathbf{m}, I)$ with $|\mathbf{m}| \leq \mathsf{l}$ and $|I| \leq q$:
       i. Compute $(\mathsf{rt}, \mathsf{pf}, z_\pi) \xleftarrow{\mathrm{tr}_{\mathsf{sim}}} \mathsf{MT.Sim}^{f[\mathrm{tr}]}(\mathbf{m}[I], I)$.
       ii. Compute $(\rho_{\mathsf{MT}}, \mathrm{tr}') \xleftarrow{\mathrm{tr}'_{\mathsf{sim}}} \mathsf{MT.Sim}^{f[\mathrm{tr}\circ\mathrm{tr}_{\mathsf{sim}}]}(\mathbf{m}, z_\pi)$.
       iii. If $\mathrm{tr} \circ \mathrm{tr}_{\mathsf{sim}} \circ \mathrm{tr}'_{\mathsf{sim}} \circ \mathrm{tr}'$ is invalid, return $\bot$.
       iv. Set $\mathrm{tr} := \mathrm{tr} \circ \mathrm{tr}_{\mathsf{sim}} \circ \mathrm{tr}'_{\mathsf{sim}} \circ \mathrm{tr}'$.
       v. Append $\rho_{\mathsf{MT}}$ to $\mathsf{Random}_{\mathsf{MT}}$.
       vi. Answer with $(\mathsf{rt}, \mathsf{pf})$.
   (d) On a corruption query, return $\mathsf{Random}_{\mathsf{MT}}$. (Refuse further prover or corruption queries.)
4. Output $\mathcal{A}$'s output.
```

MT *has* **weak (resp. strong) UC-friendly hiding with error** ζ_{MT} *if there exists a probabilistic polynomial time (oracle) algorithm* $\mathsf{MT.Sim}$ *such that for every* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$*-adversary* $\mathcal{A}$*, security parameter* λ*, message length bound* l*, opening size bound* q*,*

$$\Delta_{\mathcal{A}}\left(\mathsf{sUCMerkleHiding}_0, \mathsf{sUCMerkleHiding}_1^{\mathsf{MT.Sim}}\right) \leq \zeta_{\mathsf{MT}}(\lambda, \mathsf{l}, q, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) \ .$$

Similarly to UC-friendly zero knowledge for non-interactive arguments in Section 5.2, Definition 7.4 reduces to a simpler definition in which the adversary is only allowed a single prover query.

Lemma 7.5. *Suppose that* $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$ *satisfies a version of Definition 7.4 in which the adversary is allowed only a single query to the prover oracle, with error* $\zeta^{(1)}_{\mathsf{MT}}$*.*

Then MT *satisfies Definition 7.4 against* ℓ_{p} *prover queries, with error*

$$\zeta_{\mathsf{MT}}(\lambda, \mathsf{l}, q, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) = \ell_{\mathsf{p}} \cdot \zeta^{(1)}_{\mathsf{MT}}(\lambda, \mathsf{l}, q, t_{\mathsf{q}} + \mathsf{so}^{(1)}_{\mathsf{q}}(\mathsf{l}, q, \ell_{\mathsf{p}}), t_{\mathsf{p}} + \mathsf{so}^{(1)}_{\mathsf{p}}(\mathsf{l}, q, \ell_{\mathsf{p}})) \ .$$

In the above:

- $\mathsf{so}^{(1)}_{\mathsf{q}}(\mathsf{l}, q, \ell_{\mathsf{p}}) := \ell_{\mathsf{p}} \cdot \max\{\mathsf{q}_{\mathsf{MT.Commit}}(\mathsf{l}), 2\mathsf{q}_{\mathsf{MT.Sim}}(\mathsf{l}, q)\}$,
- $\mathsf{so}^{(1)}_{\mathsf{p}}(\mathsf{l}, q, \ell_{\mathsf{p}}) := \ell_{\mathsf{p}} \cdot \mathsf{p}_{\mathsf{MT.Sim}}(\mathsf{l}, q)$.

Proof. The proof is identical to that of Lemma 5.13, and leads to slightly different costs of simulating the oracles. □

We show that Merkle commitment schemes satisfy this strong one-shot version of Definition 7.4 in the sequel.

Lemma 7.6. $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$ *has (one-shot) strong UC-friendly hiding with error*

$$\zeta^{(1)}_{\mathsf{MT}}(\lambda, \mathsf{l}, q, \mathsf{r}_{\mathsf{MT}}, t_{\mathsf{q}}, t_{\mathsf{p}})$$

where the error bound $\zeta^{(1)}_{\mathsf{MT}}$ *is given in Lemma 7.12. In particular, for the simulator therein* $\mathsf{q}_{\mathsf{MT.Sim}}(\mathsf{l}, q) \leq 2\mathsf{l}$ *and* $\mathsf{p}_{\mathsf{MT.Sim}}(\mathsf{l}, q) \leq 2q \cdot \mathsf{l}$*.*

Proof. Let $\mathcal{A}$ be an arbitrary $(t_{\mathsf{q}}, t_{\mathsf{p}})$-adversary against the strong one-shot version of $\mathsf{sUCMerkleHiding}$. We assume, without loss of generality, that the adversary makes exactly one query to the prover oracle and one to the corruption oracle. Further, again without loss of generality, we assume that the call to the corruption oracle occurs immediately after the call to prover oracle. For a given simulator $\mathsf{MT.Sim}$, the simulation error then corresponds exactly to the statistical distance of following two distributions:

$$D_1(\mathcal{A}) := \left\{ \mathcal{A}^{[\![f, \mathrm{tr}^{(1)} \circ \mathrm{tr}_{\mathsf{commit}}]\!]}(\mathsf{rt}, \mathsf{pf}, \rho_{\mathsf{MT}}) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \left(\mathbf{m} \in \Sigma^{\mathsf{l}}, I \in \binom{[\mathsf{l}]}{q}\right) \xleftarrow{\mathrm{tr}^{(1)}} \mathcal{A}^{[\![f]\!]} \\ \rho_{\mathsf{MT}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathsf{MT.Commit}}} \\ (\mathsf{rt}, \mathsf{td}) \xleftarrow{\mathrm{tr}_{\mathsf{commit}}} \mathsf{MT.Commit}^{f[\mathrm{tr}^{(1)}]}(\mathbf{m}; \rho_{\mathsf{MT}}) \\ \mathsf{pf} := \mathsf{MT.Open}(\mathsf{td}, I) \end{array} \right\}$$

and

$$D_2(\mathcal{A}) := \left\{ \mathcal{A}^{[\![f, \mathrm{tr}^{(1)} \circ \mathrm{tr}_{\mathsf{sim}} \circ \mathrm{tr}'_{\mathsf{sim}} \circ \mathrm{tr}]\!]}(\mathsf{rt}, \mathsf{pf}, \rho_{\mathsf{MT}}) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \left(\mathbf{m} \in \Sigma^{\mathsf{l}}, I \in \binom{[\mathsf{l}]}{q}\right) \xleftarrow{\mathrm{tr}^{(1)}} \mathcal{A}^{[\![f]\!]} \\ (\mathsf{rt}, \mathsf{pf}, z_{\mathsf{MT}}) \xleftarrow{\mathrm{tr}_{\mathsf{sim}}} \mathsf{MT.Sim}^{f[\mathrm{tr}^{(1)}]}(I, \mathbf{m}[I]) \\ (\rho_{\mathsf{MT}}, \mathrm{tr}) \xleftarrow{\mathrm{tr}'_{\mathsf{sim}}} \mathsf{MT.Sim}^{f[\mathrm{tr}^{(1)} \circ \mathrm{tr}_{\mathsf{sim}}]}(\mathbf{m}, z_{\mathsf{MT}}) \end{array} \right\} .$$

In Construction 7.11 we construct a simulator $\mathsf{MT.Sim}$, and in Lemma 7.12 we show that, for that simulator, $\Delta_{\mathcal{A}}(D_1, D_2) \leq \zeta^{(1)}_{\mathsf{MT}}$, which implies the lemma statement. □

To prove Lemma 7.12, we proceed in three steps: (i) in Section 7.3.1 we prove a UC-friendly hiding property of the basic commitment scheme (a building block); (ii) in Section 7.3.2 we prove a UC-friendly hiding property of a Merkle commitment (the root hash); and (iii) in Section 7.3.3 we prove the UC-friendly hiding property of Merkle commitment schemes described above.

7.3.1 UC-friendly hiding of the basic commitment scheme

The *basic commitment scheme* CM is defined as follows.

$\mathsf{CM.Commit}^f(m \in \Sigma; \rho \in \{0,1\}^{\mathsf{r}})$: Output $\mathsf{cm} := f((m, \rho))$.

In Construction 7.7 we give a simulator $\mathsf{CM.Sim}$ for CM and then in Lemma 7.8 we prove that CM satisfies a notion of UC-friendly hiding.

Construction 7.7. Let $\mathsf{CM.Sim}$ be the following (pair of) algorithms.

$\mathsf{CM.Sim}$: Sample and output $\mathsf{cm} \leftarrow \{0,1\}^{\lambda}$.

$\mathsf{CM.Sim}(m, \mathsf{cm})$:
1. Sample $\rho \leftarrow \{0,1\}^{\mathsf{r}}$.
2. Set $\mathrm{tr} := ((\mathsf{prog}, (m, \rho), \mathsf{cm}))$.
3. Output (ρ, tr).

Lemma 7.8. *Consider the two distributions*

$$D_1(\mathcal{A}) := \left\{ \mathcal{A}^{[\![f, \mathrm{tr}^{(1)} \circ \mathrm{tr}_{\mathsf{CM}}]\!]}(\mathsf{cm}, \rho) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ m \xleftarrow{\mathrm{tr}^{(1)}} \mathcal{A}^{[\![f]\!]} \\ \rho \leftarrow \{0,1\}^{\mathsf{r}} \\ \mathsf{cm} \xleftarrow{\mathrm{tr}_{\mathsf{CM}}} \mathsf{CM.Commit}^{f[\mathrm{tr}^{(1)}]}(m; \rho) \end{array} \right\}$$

and

$$D_2(\mathcal{A}) := \left\{ \mathcal{A}^{[\![f, \mathrm{tr}^{(1)} \circ \mathrm{tr}]\!]}(\mathsf{cm}, \rho) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ m \xleftarrow{\mathrm{tr}^{(1)}} \mathcal{A}^{[\![f]\!]} \\ \mathsf{cm} \leftarrow \mathsf{CM.Sim} \\ (\rho, \mathrm{tr}) \leftarrow \mathsf{CM.Sim}^{[\![f, \mathrm{tr}^{(1)}]\!]}(m, \mathsf{cm}) \end{array} \right\} .$$

For every $(t_{\mathsf{q}}, t_{\mathsf{p}})$*-query adversary* $\mathcal{A}$*,*

$$\Delta_{\mathcal{A}}(D_1, D_2) \le \zeta_{\mathsf{CM}}(\lambda, \mathsf{r}, t_{\mathsf{q}}, t_{\mathsf{p}}) := \frac{t_{\mathsf{q}} + t_{\mathsf{p}}}{2^{\mathsf{r}}} .$$

Proof. Define the event E that $(m, \rho) \in \mathrm{tr}^{(1)}$. Since ρ is chosen uniformly at random in $\{0,1\}^{\mathsf{r}}$, and $|\mathrm{tr}^{(1)}| \le t_{\mathsf{q}} + t_{\mathsf{p}}$ we have that $\Pr[E_1] \le \frac{t_{\mathsf{q}} + t_{\mathsf{p}}}{2^{\mathsf{r}}}$. Conditioned on E not occurring, cm is a uniformly random string in $\{0,1\}^{\lambda}$ in both games, ρ is uniformly distributed in both games and f is valid. Thus, the distributions are identical, and we are done. □

7.3.2 UC-friendly hiding of the root of Merkle commitment schemes

We show that a Merkle commitment (the root hash) satisfies a UC-friendly hiding property: in Construction 7.9 we give a simulator $\mathsf{MT.RootSim}$ and then in Lemma 7.10 we prove the property. This builds on the basic commitment scheme CM in Section 7.3.1.

Construction 7.9. Let $\mathsf{MT.RootSim}$ be the following (pair of) algorithms.

$\mathsf{MT.RootSim}^f$:

1. For every $i \in [\mathsf{l}]$, sample $\mathsf{cm}_i \leftarrow \mathsf{CM.Sim}$.
2. Compute rt by constructing the (unsalted) Merkle commitment with leaves $\mathsf{cm}_1, \ldots, \mathsf{cm}_\mathsf{l}$.
3. Set $z_\mathsf{rt} := (\mathsf{cm}_1, \ldots, \mathsf{cm}_\mathsf{l})$.
4. Output $(\mathsf{rt}, z_\mathsf{rt})$.

$\mathsf{MT.RootSim}^f(m, z_\mathsf{rt})$:
1. For every $i \in [\mathsf{l}]$, sample $(\rho_i, \mathrm{tr}_i) \leftarrow \mathsf{CM.Sim}(m_i, \mathsf{cm}_i)$.
2. Set $\rho := (\rho_1, \ldots, \rho_\mathsf{l})$ and $\mathrm{tr} := \circ_i \mathrm{tr}_i$.
3. Return (ρ, tr).

Lemma 7.10. *Consider the two distributions*

$$D_1(\mathcal{A}) := \left\{ \mathcal{A}^{[\![f, \mathrm{tr}^{(1)} \circ \mathrm{tr}_\mathsf{commit}]\!]}(\mathsf{rt}, \rho_\mathsf{MT}) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \mathbf{m} \xleftarrow{\mathrm{tr}^{(1)}} \mathcal{A}^{[\![f]\!]} \\ \rho_\mathsf{MT} \leftarrow \{0,1\}^{\mathsf{r}_\mathsf{MT.Commit}} \\ (\mathsf{rt}, \mathsf{td}) \xleftarrow{\mathrm{tr}_\mathsf{commit}} \mathsf{MT.Commit}^{f[\mathrm{tr}^{(1)}]}(\mathbf{m}; \rho_\mathsf{MT}) \end{array} \right\}$$

and

$$D_2(\mathcal{A}) := \left\{ \mathcal{A}^{[\![f, \mathrm{tr}^{(1)} \circ \mathrm{tr}_\mathsf{sim} \circ \mathrm{tr}'_\mathsf{sim} \circ \mathrm{tr}]\!]}(\mathsf{rt}, \rho_\mathsf{MT}) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \mathbf{m} \xleftarrow{\mathrm{tr}^{(1)}} \mathcal{A}^{[\![f]\!]} \\ (\mathsf{rt}, z_\mathsf{rt}) \xleftarrow{\mathrm{tr}_\mathsf{sim}} \mathsf{MT.RootSim}^{f[\mathrm{tr}^{(1)}]} \\ (\rho_\mathsf{MT}, \mathrm{tr}) \xleftarrow{\mathrm{tr}'_\mathsf{sim}} \mathsf{MT.RootSim}^{f[\mathrm{tr}^{(1)} \circ \mathrm{tr}_\mathsf{sim}]}(\mathbf{m}, z_\mathsf{rt}) \end{array} \right\} .$$

For every $(t_\mathsf{q}, t_\mathsf{p})$-query adversary $\mathcal{A}$,

$$\Delta_\mathcal{A}(D_1, D_2) \le \zeta_\mathsf{rt}(\lambda, \mathsf{l}, \mathsf{r}_\mathsf{MT}, t_\mathsf{q}, t_\mathsf{p}) := \mathsf{l} \cdot \frac{t_\mathsf{q} + t_\mathsf{p} + 2\mathsf{l} - 1}{2^{\mathsf{r}_\mathsf{MT}}} .$$

Proof. We proceed via a sequence of hybrid games. For $i \in [\mathsf{l}]$, in the i-th game the first i leaves are simulated using $\mathsf{CM.Sim}$, while the remaining $\mathsf{l} - i$ leaves are computed using $\mathsf{CM.Commit}$. Let G_i be the i-th such game, so that $D_1 \equiv \mathsf{G}_0$ and $D_2 \equiv \mathsf{G}_\mathsf{l}$. The i-th reduction adversary $\mathcal{B}_i$ that argues closeness between G_i and G_{i-1} makes l oracle queries to compute the Merkle commitment over the leaves, $\mathsf{l} - i$ queries to compute the leaves that are not simulated, and $i - 1$ programming queries to compute the randomness of the simulated leaves. Hence, $\Delta_\mathcal{A}(\mathsf{G}_i, \mathsf{G}_{i+1}) \le \zeta_\mathsf{CM}(\lambda, \mathsf{r}_\mathsf{MT}, t_\mathsf{q} + 2\mathsf{l} - i, t_\mathsf{p} + i - 1)$. We deduce that

$$\begin{aligned} \Delta_\mathcal{A}(D_1, D_2) &\le \sum_{i \in [0, \mathsf{l}-1]} \zeta_\mathsf{CM}(\lambda, \mathsf{r}_\mathsf{MT}, t_\mathsf{q} + 2\mathsf{l} - i, t_\mathsf{p} + i - 1) \\ &= \sum_{i \in [0, \mathsf{l}-1]} \frac{t_\mathsf{q} + t_\mathsf{p} + 2\mathsf{l} - 1}{2^{\mathsf{r}_\mathsf{MT}}} \\ &= \mathsf{l} \cdot \frac{t_\mathsf{q} + t_\mathsf{p} + 2\mathsf{l} - 1}{2^{\mathsf{r}_\mathsf{MT}}} . \end{aligned}$$

□

7.3.3 UC-friendly hiding of Merkle commitment schemes

Finally, we show that authentication paths as well do not leak any information about the (other) leaves of the Merkle commitment scheme.

Construction 7.11. Let $\mathsf{MT.Sim}$ be the following (pair of) algorithms:

```
$\mathsf{MT.Sim}^f(I, (m_i)_{i \in I})$:
1. For $i \in I$, sample a random $\rho_i \leftarrow \{0,1\}^{\mathsf{r_{MT}}}$, set $c_{(d,i)} := f(m_i, \rho_i)$.
2. For $i \notin I$, set $c_{(d,i)} := \bot$.
3. For $j = d-1, \ldots, 0$ and $i \in [2^j]$
    (a) If $c_{(j+1,2i-1)} = c_{(j+1,2i-1)} = \bot$, set $c_{(j,i)} := \bot$.
    (b) Otherwise:
        i. If $c_{(j+1,2i-1)} = \bot$, set $c_{(j+1,2i-1)}, z_{\mathsf{rt}}) := \mathsf{MT.RootSim}^f$.
        ii. If $c_{(j+1,2i)} = \bot$, set $(c_{(j+1,2i)}, z_{\mathsf{rt}}) := \mathsf{MT.RootSim}^f$.
        iii. Set $c_{(j,i)} := f(c_{(j+1,2i-1)}, c_{(j+1,2i)}), z_{\mathsf{rt}}^{(j,i)} := z_{\mathsf{rt}}$.
4. Set $\mathsf{rt} := c_{(0,1)}$.
5. For $i \in I$, set $\mathsf{auth}_i := (\rho_i, (c_{\bar{\mathsf{p}}(i,j)})_{j \in [d]})$ and $\mathsf{pf} := (\mathsf{auth}_i)_{i \in I}$.
6. Set $z_{\mathsf{MT}} := \{I, \mathsf{rt}, \mathsf{pf}, (\rho_i)_{i \in I}, (z_{\mathsf{rt}}^{(j,i)})_{j,i}, \}$
7. Return $(\mathsf{rt}, \mathsf{pf})$.

$\mathsf{MT.Sim}(m, z_{\mathsf{MT}})$:
1. For $i \in I$, set $c_{(d,i)} := \top$.
2. For $i \notin I$, set $c_{(d,i)} := \bot$.
3. For $j = d-1, \ldots, 0$ and $i \in [2^j]$
    (a) If $c_{(j+1,2i-1)} = c_{(j+1,2i-1)} = \bot$, set $c_{(j,i)} := \bot$.
    (b) Otherwise:
        i. If $c_{(j+1,2i-1)} = \bot$, compute $\rho_{\mathsf{span}(j+1,2i-1)}, \mathrm{tr}^{(j,i)} \leftarrow \mathsf{MT.Sim}^f(m[\mathsf{span}(j+1, 2i-1)], z_{\mathsf{rt}}^{(j,i)})$.
        ii. If $c_{(j+1,2i)} = \bot$, compute $\rho_{\mathsf{span}(j+1,2i)}, \mathrm{tr}^{(j,i)} \leftarrow \mathsf{MT.Sim}^f(m[\mathsf{span}(j+1, 2i)], z_{\mathsf{rt}}^{(j,i)})$.
        iii. Set $c_{(j,i)} := \top$.
4. Return $\rho := (\rho_i)_{i \in [\mathsf{l}]}$, $\mathrm{tr} := \circ_{(j,i):c_{(j,i)}=\top} \mathrm{tr}^{(j,i)}$.
```

Lemma 7.12. *Consider the two distributions*

$$D_1(\mathcal{A}) := \left\{ \mathcal{A}^{[\![f, \mathrm{tr}^{(1)} \circ \mathrm{tr}_{\mathsf{commit}}]\!]}(\mathsf{rt}, \mathsf{pf}, \rho_{\mathsf{MT}}) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \left(\mathbf{m} \in \Sigma^{\mathsf{l}}, I \in \binom{[\mathsf{l}]}{q}\right) \xleftarrow{\mathrm{tr}^{(1)}} \mathcal{A}^{[\![f]\!]} \\ \rho_{\mathsf{MT}} \leftarrow \{0,1\}^{\mathsf{r_{MT.Commit}}} \\ (\mathsf{rt}, \mathsf{td}) \xleftarrow{\mathrm{tr}_{\mathsf{commit}}} \mathsf{MT.Commit}^{f[\mathrm{tr}^{(1)}]}(\mathbf{m}; \rho_{\mathsf{MT}}) \\ \mathsf{pf} := \mathsf{MT.Open}(\mathsf{td}, I) \end{array} \right\}$$

and

$$D_2(\mathcal{A}) := \left\{ \mathcal{A}^{[\![f, \mathrm{tr}^{(1)} \circ \mathrm{tr}_{\mathsf{sim}} \circ \mathrm{tr}'_{\mathsf{sim}} \circ \mathrm{tr}]\!]}(\mathsf{rt}, \mathsf{pf}, \rho_{\mathsf{MT}}) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \left(\mathbf{m} \in \Sigma^{\mathsf{l}}, I \in \binom{[\mathsf{l}]}{q}\right) \xleftarrow{\mathrm{tr}^{(1)}} \mathcal{A}^{[\![f]\!]} \\ (\mathsf{rt}, \mathsf{pf}, z_{\mathsf{MT}}) \xleftarrow{\mathrm{tr}_{\mathsf{sim}}} \mathsf{MT.Sim}^{f[\mathrm{tr}^{(1)}]}(I, \mathbf{m}[I]) \\ (\rho_{\mathsf{MT}}, \mathrm{tr}) \xleftarrow{\mathrm{tr}'_{\mathsf{sim}}} \mathsf{MT.Sim}^{f[\mathrm{tr}^{(1)} \circ \mathrm{tr}_{\mathsf{sim}}]}(\mathbf{m}, z_{\mathsf{MT}}) \end{array} \right\} .$$

For every $(t_{\mathsf{q}}, t_{\mathsf{p}})$-query adversary $\mathcal{A}$,

$$\Delta_{\mathcal{A}}(D_1, D_2) \leq \zeta_{\mathsf{MT}}(\lambda, \mathsf{l}, \mathsf{r_{MT}}, t_{\mathsf{q}}, t_{\mathsf{p}}) := q \cdot \sum_{j \in [\mathsf{d}]} \zeta_{\mathsf{rt}}(\lambda, 2^{\mathsf{d}-j}, \mathsf{r_{MT}}, t_{\mathsf{q}} + 2\mathsf{l}, t_{\mathsf{p}} + \mathsf{l}) \ .$$

Proof. We again proceed by a sequence of hybrid games. At each of the d layers, at most q roots will be simulated. At level $j \in [d]$, each of these simulated roots will be of a tree of size 2^{d-j}, thus the overall error will grow as $q \cdot \sum_{j\in[d]} \zeta_{\mathsf{rt}}(\lambda, 2^{d-j}, \mathsf{r}_{\mathsf{MT}}, \dots)$. As before, we will incur in a simulation overhead which will be at most of $2\mathsf{l}$ queries and l programming queries. This leads to the bound we stated. □

Remark 7.13. In Section 7.4, to prove UC-friendly extraction, we require that the simulator $\mathsf{MT.Sim}$ in Construction 7.11 re-queries points that it has programmed, to ensure that the extractor receives as input these query-answer pairs. (Recall that the query-answer trace given as input to the extractor consists only of points queried by the adversary and simulator to the random oracle, and does not include points programmed by the adversary or points programmed by the simulator that were not later queried by either the adversary or the simulator.) With this change, the query complexity of an invocation of $\mathsf{MT.Sim}$ is exactly $2\mathsf{l}$.

7.4 UC-friendly extraction

Merkle commitment schemes in the ROM have strong extraction properties. We show that corresponding properties hold in our model, even in the face of an adversary who can program the random oracle.

We define the notion of UC-friendly extraction for the Merkle commitment scheme.

Definition 7.14. *We define the* **Merkle extraction game** *with respect to a simulator* $\mathsf{MT.Sim}$ *and a stateful extractor* $\mathsf{MT.MultiExtract}$ *as follows.*

```
s UCMerkleExtraction^f(A, l, q, n, k):
1. Initialize empty lists tr, advProg, extTrace, Random_MT.
2. Run A, answering queries as follows:
   – On a random oracle query x, set y := f[tr](x), append (query, x, y) to tr, extTrace, and return y.
   – On a programming query trace_prog:
      (a) If there exist (x, y) ∈ trace_prog and (qid_i, x_i, y_i) ∈ tr with x_i = x, return 0.
      (b) Else append ((prog, x, y))_{(x,y)∈trace_prog} to tr and advProg and return 1.
   – On a simulator query (m ∈ Σ^l, I ⊆ [l]):
      (a) Compute (rt, pf, z_MT) <-trS- MT.Sim^{f[tr]}(I, m[I]).
      (b) Compute (ρ_MT, tr') <-tr'S- MT.Sim^{f[tr]}(m, z_MT).
      (c) If tr ∘ trS ∘ tr'S ∘ tr' is invalid, return ⊥.
      (d) Set tr := tr ∘ trS ∘ tr'S ∘ tr'.
      (e) Set extTrace := extTrace ∘ trS ∘ tr'S.
      (f) Append ρ_MT to Random_MT.
      (g) Return (rt, pf).
   – On the j-th root query rt_j (for j ∈ [n]):
      (a) Let extTrace_j be the query-answer pairs added to extTrace since the last invocation of MT.MultiExtract,
          excluding query-answers pairs in advProg.
      (b) Compute (m_j, td_j) := MT.MultiExtract(rt_j, extTrace_j).
      (c) Return m_j.
   – On a corruption query, return Random_MT (and stop answering further simulator queries).
3. A eventually outputs ((i_j, I_{i_j}, a_{i_j}, pf_{i_j}))_{j∈[k]}.
4. Let extTrace* be the query-answer pairs added to extTrace since the last invocation of MT.MultiExtract,
   excluding query-answer pairs in advProg.
5. For j = 1, ..., k:
```

(a) $b_j \xleftarrow{\mathrm{tr}^j_{\mathsf{check}}} \mathsf{MT.Check}^{f[\mathrm{tr}]}(\mathsf{rt}_{i_j}, I_{i_j}, \mathbf{a}_{i_j}, \mathsf{pf}_{i_j})$.
(b) $\mathsf{pf}'_j := \mathsf{MT.Open}(\mathsf{td}_{i_j}, I_{i_j})$.

6. *Output* $\mathsf{advWin} := 1$ *if any of the following conditions are satisfied.*
   (a) $\exists\, i, i' \in [n] : \mathsf{rt}_i = \mathsf{rt}_{i'} \wedge \mathbf{m}_i \neq \mathbf{m}_{i'}$.
   (b) $\exists\, j \in [k]$*:* $b_j = 1$, $\mathrm{tr}^j_{\mathsf{check}} \cap \mathsf{advProg}_j = \emptyset$*, and* $\mathbf{m}_{i_j}[I_{i_j}] \neq \mathbf{a}_{i_j} \vee \mathsf{pf}_{i_j} \neq \mathsf{pf}'_j$.
7. *Else output* $\mathsf{advWin} := 0$.

MT *has* **weak (resp. strong) UC-friendly extraction** *with respect to* MT.Sim *with error* κ_{MT} *if there exists a (stateful) extractor* MT.MultiExtract *such that for every* $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$*-query adversary* $\mathcal{A}$*:*

$$\Pr\left[\mathsf{advWin} = 1 \,\middle|\, \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \mathsf{advWin} \leftarrow s\mathsf{UCMerkleExtraction}^f(\mathcal{A}, \mathsf{l}, q, n, k) \end{array}\right] \leq \kappa_{\mathsf{MT}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \mathsf{l}, q, n, k) \ .$$

We directly show that the Merkle commitment scheme in the ROM satisfies strong UC-friendly extraction, which also implies that it satisfies weak UC-friendly extraction with the same bound.

Lemma 7.15. *Let* $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$ *and* MT.Sim *be the simulator in Construction 7.11. Then,* MT *has strong UC-friendly extraction with respect to* MT.Sim *with the following error:*

$$\kappa_{\mathsf{MT}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \mathsf{l}, q, n, k) := \frac{3}{2} \cdot \frac{(t_{\mathsf{q}} + 2\ell_{\mathsf{p}}\mathsf{l})^2}{2^\lambda} + 3k(\log \mathsf{l} + 1) \cdot \frac{t_{\mathsf{q}} + 2\ell_{\mathsf{p}}\mathsf{l}}{2^\lambda} \ .$$

Our proof is an extension of the proof of multi-extraction for the Merkle commitment scheme in [CY24, Lemma 18.5.6]. We highlight the parts in which the two proofs differ; throughout, the citation [CY24] is understood to refer to the proof of that lemma. The extractor MT.MultiExtract, whose description we include for completeness below, is identical to the one in [CY24].

Construction 7.16. The multi-extractor MT.MultiExtract is defined based on a single-extraction subroutine MT.Extract. MT.Extract receives as input a Merkle commitment $\mathsf{rt} \in \{0,1\}^\lambda$ and a query-answer trace extTrace, and works as follows.

MT.Extract(rt, extTrace):
1. If rt is not the answer of any query in extTrace, return $(\mathbf{m}, \mathsf{td}) := (\bot, \bot)$.
2. Partition extTrace into three sets:
   - $\mathrm{tr}_{\mathsf{leaf}}$ contains all query-answer pairs (x, y) with $x = (m, \rho) \in \Sigma \times \{0,1\}^{\mathsf{r}_{\mathsf{MT}}}$.
   - $\mathrm{tr}_{\mathsf{inner}}$ contains all query-answer pairs (x, y) with $x \in \{0,1\}^{2\lambda}$.
   - $\mathrm{tr}_{\mathsf{other}}$ contains all other query-answer pairs.
3. Label a binary tree $\mathcal{T}$ of depth d as follows.
   (a) The root of the tree is labeled with rt.
   (b) While there is $(x, y) \in \mathrm{tr}_{\mathsf{inner}}$ where y is a label of a inner node of $\mathcal{T}$, write $x = (x_L, x_R) \in (\{0,1\}^\lambda)^2$, and label the left child of that vertex with x_L, and the right with x_R. Then, remove (x, y) from $\mathrm{tr}_{\mathsf{inner}}$.
   (c) For every $(x, y) = ((m, \rho), y) \in \mathrm{tr}_{\mathsf{leaf}}$, if y is the label of the i-th leaf of $\mathcal{T}$, set $m_i := m$, $\rho_i := \rho$. Remove (x, y) from $\mathrm{tr}_{\mathsf{leaf}}$.
   (d) Label every remaining vertex of $\mathcal{T}$ with $\bot$.
4. For every $i \in [\mathsf{l}]$, if (m_i, ρ_i) are not yet defined, set them to be $(\bot, \bot)$.
5. Let $c_{(j,i)}$ be the label of the i-th inner node in the j-th level of $\mathcal{T}$.
6. Set $\mathbf{m} := (m_i)_{i \in [\mathsf{l}]}$ and $\mathsf{td} := (\mathbf{m}, (\rho_i)_i, (c_{(j,i)})_{j,i})$.

7. Return $(\mathbf{m}, \mathsf{td})$.

The multi-extractor $\mathsf{MT.MultiExtract}$ maintains an internal query-answer trace $\mathsf{extTrace}$. On the i-th invocation, $\mathsf{MT.MultiExtract}$ is defined as follows.

```
$\mathsf{MT.MultiExtract}(\mathsf{rt}, \mathsf{extTrace}_i)$:
1. Set $\mathsf{extTrace} := \mathsf{extTrace} \circ \mathsf{extTrace}_i$.
2. Return $\mathsf{MT.Extract}(\mathsf{rt}, \mathsf{extTrace})$.
```

Proof. [CY24] defines several query-answer traces. We also define several traces accordingly:

- $\mathrm{tr} := \mathsf{extTrace}_1 \circ \cdots \circ \mathsf{extTrace}_n$.
- $\mathrm{tr}' := \mathsf{extTrace}^*$.
- For every $j \in [k]$, $\mathrm{tr}^j_{\mathsf{check}}$, is the trace of the computation $b_j \leftarrow \mathsf{MT.Check}^f(\mathsf{rt}_{i_j}, I_{i_j}, \mathbf{a}_{i_j}, \mathsf{pf}_{i_j})$.

Define $t_1 := |\mathrm{tr}|$ and $t_2 := |\mathrm{tr}'|$, and $t := t_1 + t_2$. In our setting, $t \leq t_{\mathsf{q}} + 2\ell_{\mathsf{p}}\mathsf{l}$ because, unlike [CY24], the query-answer traces also contain queries performed by the simulator to the random oracle. (There is no simulator in [CY24]; it corresponds to the setting of $\ell_{\mathsf{p}} = 0$ calls to the simulator.)

Similarly to [CY24], we define several events:

- E is the event $\mathsf{advWin} = 1$.
- E_{col} is the event that tr contains a collision.
- $E_{\mathsf{tree},1}$ is the event that there exists $j \in [k]$ with $\mathcal{T}_{i_j} \neq \hat{\mathcal{T}}_{i_j}$ where:
  - $\mathcal{T}_{i_j}$ is the binary tree reconstructed during the extraction of $\mathsf{MT.Extract}(\mathsf{rt}_{i_j}, \mathsf{extTrace}_1 \circ \cdots \circ \mathsf{extTrace}_{i_j})$.
  - $\hat{\mathcal{T}}_{i_j}$ is the binary tree reconstructed during the extraction of $\mathsf{MT.Extract}(\mathsf{rt}_{i_j}, \mathrm{tr} \circ \mathrm{tr}')$.
- $E_{\mathsf{tree},2}$ is the event that there exist $i, i' \in [n]$ with $\mathsf{rt}_i = \mathsf{rt}_{i'}$ and $\mathcal{T}_i \neq \mathcal{T}_{i'}$ where:
  - $\mathcal{T}_i$ is the binary tree reconstructed during the extraction of $\mathsf{MT.Extract}(\mathsf{rt}_i, \mathsf{extTrace}_1 \circ \cdots \circ \mathsf{extTrace}_i)$.
  - $\mathcal{T}_{i'}$ is the binary tree reconstructed during the extraction of $\mathsf{MT.Extract}(\mathsf{rt}_{i'}, \mathsf{extTrace}_1 \circ \cdots \circ \mathsf{extTrace}_{i'})$.
- E_{tree} is $E_{\mathsf{tree},1} \vee E_{\mathsf{tree},2}$.
- E_{check} is the event that there exists $j \in [k]$ with $\mathrm{tr}^j_{\mathsf{check}} \not\subseteq \mathrm{tr}$, $b_j = 1$, and $\mathrm{tr}^j_{\mathsf{check}} \cap \mathsf{advProg} = \emptyset$.

The events are as in [CY24], with the only difference being in E_{check}, which adds the condition that $\mathrm{tr}^j_{\mathsf{check}} \cap \mathsf{advProg} = \emptyset$ (in our our setting the adversary can program the random oracle).

The main difference between the two proofs is that the query-answer traces $\mathrm{tr}, \mathrm{tr}'$ in our setting not only contain queries made by the adversary, but also queries made by the simulator (excluding queries programmed by the adversary). The traces may contain points programmed by the simulator (and later queried by either the simulator or adversary). Since the simulator in Construction 7.11 only programs query-answer pairs with answers selected uniformly at random from $\{0,1\}^\lambda$, in both our setting and [CY24] the query-answer trace have answers distributed uniformly at random.

Both proofs proceed by bounding the probability of some combination of the above events.

Bounding E_{col}.

$$\Pr[E_{\mathsf{col}}] \leq \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\lambda} \quad .$$

This bound follows in [CY24] by a standard collision analysis, which only relies on the answers in tr being distributed uniformly at random.

Bounding $E_{\mathsf{tree},1}$ given $\neg E_{\mathsf{col}}$.

$$\Pr[E_{\mathsf{tree},1} | \neg E_{\mathsf{col}}] \leq t_2 \cdot \frac{\min\{3t_1, k \cdot 2\mathsf{l}\}}{2^\lambda} \quad .$$

The analysis is analogous to [CY24]. The only case in which $E_{\mathsf{tree},1}$ holds is if one of the non-dummy labels in $\mathcal{T}_{i_j}$ appears as an answer in the trace used to construct $\hat{\mathcal{T}}_{i_j}$. Since no collision occurs and by Remark 7.13 each point programmed by the simulator is re-queried, the offending query-answer pair must appear in tr'. The number of non-dummy labels is upperbounded by $\min\{3t_1, k \cdot 2\mathsf{l}\}$, and since each answer in $\mathrm{tr}'_{\mathsf{leaf}}$ is a string chosen uniformly at random from $\{0,1\}^\lambda$, the result follows.

Bounding $E_{\mathsf{tree},2}$ given $\neg E_{\mathsf{col}}$.

$$\Pr\left[E_{\mathsf{tree},2} | \neg E_{\mathsf{col}}\right] \le \frac{(t_1 - 1)\cdot t_1}{2^\lambda} \quad .$$

The analysis is analogous to [CY24]. Since no collision occurs, for every $i \in [n]$, each vertex of the tree $\mathcal{T}_i$ is labeled exactly once. Then, if for some $i, i' \in [n]$ with $i < i'$, $\mathsf{rt}_i = \mathsf{rt}_{i'}$ and $\mathcal{T}_i \ne \mathcal{T}_{i'}$, there must be a query in $\mathsf{extTrace}_1 \circ \cdots \circ \mathsf{extTrace}_{i'}$ that is not in $\mathsf{extTrace}_1 \circ \cdots \circ \mathsf{extTrace}_i$ with answer equaling a non-dummy label in $\mathcal{T}_i$. For every $j \in [t_1]$, there are at most $2(j-1)$ non-dummy labels in the binary trees constructed thus far (since any query-answer pairs leads to at most two new labels inside a binary tree). So the probability that the j-th query (whose answer is distributed uniformly at random in $\{0,1\}^\lambda$) matches one of these labels is at most $\frac{2(j-1)}{2^\lambda}$. The bound above then follows via a union bound.

Bounding E_{check} given $\neg E_{\mathsf{col}} \land \neg E_{\mathsf{tree}}$.

$$\Pr\left[E_{\mathsf{check}} | \neg E_{\mathsf{col}} \land \neg E_{\mathsf{tree}}\right] \le k \cdot (d+1) \cdot \frac{\min\{3t_1, k\cdot 2\mathsf{l}\}}{2^\lambda} \quad .$$

The analysis is analogous to [CY24], with the only difference that the condition $\mathrm{tr}^j_{\mathsf{check}} \cap \mathsf{advProg} = \emptyset$ in E_{check} guarantees that the query-answer trace of the execution of $\mathsf{MT.Check}$ only contains uniformly distributed query-answer pairs.

If E_{check} holds for some $j \in [k]$ there exists a query in $\mathrm{tr}^j_{\mathsf{check}}$ that is not in tr with an answer equaling some non-dummy label in $\mathcal{T}_{i_j}$. The query-answer trace $\mathrm{tr}^j_{\mathsf{check}}$ contains queries that were previously in tr, or in tr', or in neither. Queries in tr do not count towards the event, and since $\neg E_{\mathsf{tree}}$ holds (and thus $\neg E_{\mathsf{tree},1}$) we have that $\mathcal{T}_{i_j} = \hat{\mathcal{T}}_{i_j}$ and thus queries in tr' cannot equal non-dummy labels in $\mathcal{T}_{i_j}$. Since $\mathrm{tr}^j_{\mathsf{check}}$ contains no points programmed by the adversary (by the verification check), each remaining query is uniformly distributed, and has thus a probability at most $\min\{3t_1, k \cdot 2\mathsf{l}\}$ of matching a non-dummy label. Taking a union bound over the number of queries in an authentication path and the number of openings the result follows.

Bounding E given $\neg E_{\mathsf{col}} \land \neg E_{\mathsf{tree}} \land \neg E_{\mathsf{check}}$.

$$\Pr\left[E | \neg E_{\mathsf{col}} \land \neg E_{\mathsf{tree}} \land \neg E_{\mathsf{check}}\right] = 0$$

As in [CY24], $\neg E_{\mathsf{tree}}$ implies that, for every $i, i' \in [n]$ with $\mathsf{rt}_i = \mathsf{rt}_{i'}$, $\mathcal{T}_i = \mathcal{T}_{i'}$, so the extracted messages must also be the same.

We are left to show that, for every $j \in [k]$, it cannot simultaneously hold that (i) $b_j = 1$; (ii) $\mathrm{tr}^j_{\mathsf{check}} \cap \mathsf{advProg} = \emptyset$; and (iii) $\mathbf{m}_{i_j}[I_{i_j}] \ne \mathbf{a}_{i_j}$ or $\mathsf{pf}_{i_j} \ne \mathsf{pf}'_{i_j}$. Fix j, and assume the first two conditions hold. Then, by E_{check}, $\mathrm{tr}^j_{\mathsf{check}} \subseteq \mathrm{tr}$. Suppose first that $\mathbf{m}_{i_j}[I_{i_j}] \ne \mathbf{a}_{i_j}$. Then, there is an index $q \in I_{i_j}$ such that $\mathbf{m}_{i_j}[q] \ne \mathbf{a}_{i_j}[q]$. Since the verification check of Merkle commitments verifies each authentication path individually, and the extractor (by virtue of its definition) reconstructs messages and opening that will successfully verifies, this leads to two authentication paths that successfully verify. This must lead to a collision in tr by [CY24, Lemma 18.3.2], a contradiction since $\neg E_{\mathsf{col}}$ holds. Likewise, if the above does not hold and $\mathsf{pf}_{i_j} \ne \mathsf{pf}'_{i_j}$ then there will be two distinct authentication paths for the same opening, and again by [CY24, Lemma 18.3.1] a collision will occur. Again the above argument is as in [CY24], with the only additional condition that $\mathrm{tr}^j_{\mathsf{check}} \cap \mathsf{advProg} = \emptyset$ carried over from the definition of E_{check}, and with the observation that these checks (and the fact the extractor does not receive programmed points) make [CY24, Lemma 18.3.1, Lemma 18.3.2] hold unchanged.

Bounding E. We bound the probability of E based on the bounds discussed above.

$$\begin{aligned}\Pr[E] &\le \Pr[E_{\mathsf{col}}] + \Pr[E_{\mathsf{tree},1}|\neg E_{\mathsf{col}}] + \Pr[E_{\mathsf{tree},2}|\neg E_{\mathsf{col}}] + \Pr[E_{\mathsf{check}}|\neg E_{\mathsf{col}} \wedge \neg E_{\mathsf{tree}}] + \Pr[E|\neg E_{\mathsf{col}} \wedge \neg E_{\mathsf{tree}} \neg E_{\mathsf{check}}] \\ &\le \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\lambda} + t_2 \cdot \frac{\min\{3t_1, k \cdot 2\mathsf{l}\}}{2^\lambda} + \frac{(t_1 - 1) \cdot t_1}{2^\lambda} + k \cdot (d+1) \cdot \frac{\min\{3t_1, k \cdot 2\mathsf{l}\}}{2^\lambda} + 0 \\ &\le \frac{3}{2} \cdot \frac{t_1^2}{2^\lambda} + (t - t_1 + k(d+1)) \cdot \frac{3t_1}{2^\lambda}\end{aligned}$$

The above is maximized when $t_1 = t$, and thus

$$\Pr[E] \le \frac{3}{2} \cdot \frac{(t_{\mathsf{q}} + 2\ell_{\mathsf{p}}\mathsf{l})^2}{2^\lambda} + 3k(\log \mathsf{l} + 1) \cdot \frac{t_{\mathsf{q}} + 2\ell_{\mathsf{p}}\mathsf{l}}{2^\lambda}$$

□

8 The Micali construction is UC-secure

We prove that the Micali construction [Mic00], when instantiated with a suitable PCP, yields a zkSNARK that is UC-secure. In Section 8.1 we recall the definition of a PCP. In Section 8.2 we recall the Micali construction. In Section 8.3 we prove that the Micali construction satisfies UC-friendly completeness. In Section 8.4 we prove that the Micali construction satisfies UC-friendly zero knowledge. In Section 8.5 we prove that the Micali construction satisfies UC-friendly knowledge soundness. Finally, in Section 8.6 we combine these results to deduce UC-security of the Micali construction.

8.1 Probabilistically checkable proofs

A *probabilistically checkable proof* is a tuple $\mathsf{PCP} = (\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}})$ with the following syntax.

- $\mathbf{P}_{\mathrm{PCP}}(\mathbb{x}, \mathbb{w}) \to \Pi$: $\mathbf{P}_{\mathrm{PCP}}$ receives as input an instance-witness pair $(\mathbb{x}, \mathbb{w})$ and outputs a PCP string Π.
- $\mathbf{V}^{\Pi}_{\mathrm{PCP}}(\mathbb{x}) \to b$: receives as input an instance $\mathbb{x}$ and oracle access to a PCP string Π, and outputs a bit.

We consider the following efficiency measures (which can be functions of $|\mathbb{x}|$).
- Σ is the alphabet used to write symbols of the PCP string.
- l is the number of symbols in the PCP string.
- q is the number of queries that $\mathbf{V}_{\mathrm{PCP}}$ makes to the PCP string.
- $\mathsf{r}_{\mathbf{P}}$ is the number of random bits that $\mathbf{P}_{\mathrm{PCP}}$ uses.
- $\mathsf{r}_{\mathbf{V}}$ is the number of random bits that $\mathbf{V}_{\mathrm{PCP}}$ uses.

For $Q \subseteq [\mathsf{l}]$ and $\mathbf{a} \in \Sigma^{Q}$ we denote by $\mathbf{V}^{[Q,\mathbf{a}]}_{\mathrm{PCP}}$ the algorithm that runs $\mathbf{V}_{\mathrm{PCP}}$, answering queries to $i \in Q$ with $\mathbf{a}[i]$ (and immediately rejecting if any query is not in Q or if any query in Q is not made).

The PCPs that we consider satisfy perfect completeness, knowledge soundness, and honest-verifier zero knowledge (defined below). Note that we consider a strengthening of honest-verifier zero knowledge wherein we require the simulator to (a posteriori) reconstruct randomness used to simulate a PCP local view.

Definition 8.1. $\mathsf{PCP} = (\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}})$ *has* **perfect completeness** *for a relation R if, for every $(\mathbb{x}, \mathbb{w}) \in R$,*

$$\Pr\left[\mathbf{V}^{\Pi}_{\mathrm{PCP}}(\mathbb{x}) = 1 \middle| \Pi \leftarrow \mathbf{P}_{\mathrm{PCP}}(\mathbb{x}, \mathbb{w})\right] = 1 \ .$$

Definition 8.2. $\mathsf{PCP} = (\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}})$ *for a relation R has* **knowledge soundness with error** κ_{PCP} *if there exists a polynomial-time algorithm $\mathbf{E}_{\mathrm{PCP}}$ such that, for every instance $\mathbb{x}$ and PCP string $\widetilde{\Pi}$,*

$$\Pr\left[\begin{array}{c} \mathbf{V}^{\widetilde{\Pi}}_{\mathrm{PCP}}(\mathbb{x}) = 1 \\ \wedge\ (\mathbb{x}, \mathbb{w}) \notin R \end{array} \middle| \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{PCP}}(\mathbb{x}, \widetilde{\Pi})\right] \leq \kappa_{\mathrm{PCP}}(|\mathbb{x}|) \ .$$

Definition 8.3. *Let $\mathsf{PCP} = (\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}})$ be a probabilistically checkable proof for R. The* **joint PCP verifier view** *on the instance-witness pair $(\mathbb{x}, \mathbb{w})$, denoted as $\mathsf{jView}_{\mathrm{PCP}}(\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}}, \mathbb{x}, \mathbb{w})$, is the random variable $(\mathbb{x}, \mathbb{w}, \rho_{\mathbf{P}}, \rho, Q, \mathbf{a})$ where:*
- *$\rho_{\mathbf{P}} \in \{0,1\}^{\mathsf{r}_{\mathbf{P}}}$ is a choice of randomness for $\mathbf{P}_{\mathrm{PCP}}$;*
- *$\rho \in \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ is a choice of randomness for $\mathbf{V}_{\mathrm{PCP}}$;*
- *$Q \subseteq [\mathsf{l}]$ and $\mathbf{a} \in \Sigma^{Q}$ are the queries and answers of the verifier when running $\mathbf{V}^{\Pi}_{\mathrm{PCP}}(\mathbb{x}; \rho)$ with $\Pi \leftarrow \mathbf{P}_{\mathrm{PCP}}(\mathbb{x}, \mathbb{w}; \rho_{\mathbf{P}})$.*

The **verifier view** *is similarly denoted as $\mathsf{View}_{\mathrm{PCP}}(\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}}, \mathbb{x}, \mathbb{w})$, and is obtained by dropping $\mathbb{w}$ and $\rho_{\mathbf{P}}$ from $\mathsf{jView}_{\mathrm{PCP}}$.*

PCP *has* **honest-verifier zero knowledge with error** ζ_{PCP} *if there exists a probabilistic polynomial time algorithm* $\mathbf{S}_{\mathrm{PCP}}$ *such that, for every* $(\mathbb{x}, \mathbb{w}) \in R$*,* $\zeta_{\mathrm{PCP}}(|\mathbb{x}|)$ *is an upper bound on the statistical distance of the two random variables*

$$\mathsf{View}_{\mathrm{PCP}}(\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}}, \mathbb{x}, \mathbb{w}) \text{ and } \mathbf{S}_{\mathrm{PCP}}(\mathbb{x}) \enspace .$$

PCP *has* **strong honest-verifier zero knowledge with error** ζ_{PCP} *if there exists a (pair of) polynomial-time probabilistic algorithm* $\mathbf{S}_{\mathrm{PCP}}$ *such that, for every* $(\mathbb{x}, \mathbb{w}) \in R$*,* $\zeta_{\mathrm{PCP}}(|\mathbb{x}|)$ *is an upper bound on the statistical distance of the two random variables*

$$\mathsf{jView}_{\mathrm{PCP}}(\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}}, \mathbb{x}, \mathbb{w}) \text{ and } \left\{ (\mathbb{x}, \mathbb{w}, \rho_{\mathbf{P}}, \rho, Q, \mathbf{a}) \middle| \begin{array}{l} (\rho, Q, \mathbf{a}, z_{\mathsf{SIM}}) \leftarrow \mathbf{S}_{\mathrm{PCP}}(\mathbb{x}) \\ \rho_{\mathbf{P}} \leftarrow \mathbf{S}_{\mathrm{PCP}}(\mathbb{w}, z_{\mathsf{SIM}}) \end{array} \right\} \enspace .$$

Below we recall the notion of state-restoration knowledge soundness, which we use as a technical stepping stone in the proof of UC-friendly knowledge soundness of the Micali construction. Then we recall the fact that knowledge soundness implies state-restoration knowledge soundness with a multiplicative loss.

Definition 8.4. *The* **state-restoration** *game is defined as follows.*

```
$\mathsf{Game}_{\mathsf{sr}}(\mathcal{A}, \mathsf{rnd}, s)$
1. Repeat until $\mathcal{A}$ decides to exit the loop.
    (a) Get $(\mathbb{x}, \Pi, \sigma)$ from $\mathcal{A}$.
    (b) Compute $\rho := \mathsf{rnd}(\mathbb{x}, \Pi, \sigma)$
    (c) Send $\rho$ to $\mathcal{A}$
2. Get $(\mathbb{x}, \Pi, \sigma)$ from $\mathcal{A}$.
3. Compute $\rho := \mathsf{rnd}(\mathbb{x}, \Pi, \sigma)$
4. Output $(\mathbb{x}, \Pi, \sigma, \rho)$.
```

The adversary $\mathcal{A}$ *is* t_{sr}*-move if it enters the loop at most* t_{sr} *times.*

$\mathsf{PCP} = (\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}})$ *for a relation* R *has* **state-restoration knowledge soundness with error** κ_{sr} *if there exists a polynomial-time algorithm* $\mathbf{E}_{\mathrm{PCP}}$ *such that, for every* t_{sr}*-move* $\mathcal{A}$*, instance bound* n*, and salt-size* s*,*

$$\Pr\left[\begin{array}{l} |\mathbb{x}| \le n \\ \wedge\ \mathbf{V}^{\Pi}_{\mathrm{PCP}}(\mathbb{x}; \rho) = 1 \\ \wedge\ (\mathbb{x}, \mathbb{w}) \notin R \end{array} \middle| \begin{array}{l} \mathsf{rnd} \leftarrow \mathcal{U}(\mathsf{r}_{\mathbf{V}}) \\ (\mathbb{x}, \Pi, \sigma, \rho) \leftarrow \mathsf{Game}_{\mathsf{sr}}(\mathcal{A}, \mathsf{rnd}, s) \\ \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{PCP}}(\mathbb{x}, \Pi) \end{array} \right] \le \kappa_{\mathsf{sr}}(n, t_{\mathsf{sr}}, s) \enspace .$$

Claim 8.5. *If* PCP *has knowledge soundness with error* κ_{PCP}*, then* PCP *has state-restoration knowledge soundness with error* κ_{sr} *where* $\kappa_{\mathsf{sr}}(n, t_{\mathsf{sr}}, s) := (t_{\mathsf{sr}} + 1) \cdot \kappa_{\mathrm{PCP}}(n)$*.*

8.2 The Micali construction

We describe the Micali construction of a SNARG, starting from two ingredients: (a) a $\mathsf{PCP} := (\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}})$; and (b) a Merkle commitment scheme $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$ in the ROM. Throughout we assume that $\mathsf{r}_{\mathbf{V}} \le \lambda$.[12] Let $f_{\mathsf{MT}} \colon \{0,1\}^* \to \{0,1\}^{\lambda}$ and $f_{\mathsf{FS}} \colon \{0,1\} \to \{0,1\}^{\mathsf{r}_{\mathbf{V}}}$ be two domain-separated random oracles obtained as detailed in Section 3. We define $\mathsf{Micali} := \mathsf{Micali}[\mathsf{PCP}, \mathsf{r}] := (\mathbf{P}, \mathbf{V})$ to be the non-interactive argument constructed as follows.

- $\mathbf{P}^{f_{\mathsf{MT}}, f_{\mathsf{FS}}}(\mathbb{x}, \mathbb{w})$:

[12]The analysis can be straightforwardly adapted otherwise, with only slightly increased simulation overheads.

1. Compute the PCP: $\Pi \leftarrow \mathbf{P}_{\mathrm{PCP}}(\mathbb{x}, \mathbb{w})$
2. Compute the Merkle commitment: $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MT.Commit}^{f_{\mathsf{MT}}}(\Pi)$.
3. Sample a salt $\sigma \leftarrow \{0,1\}^{\mathsf{r}}$.
4. Compute PCP randomness: $\rho := f_{\mathsf{FS}}((\mathbb{x}, \mathsf{rt}, \sigma))$.
5. Run the PCP verifier $\mathbf{V}_{\mathrm{PCP}}^{\Pi}(\mathbb{x}; \rho)$ to deduce the query set $Q \subseteq [\mathsf{l}]$.
6. Set the PCP answers: $\mathbf{a} := \Pi[Q]$
7. Compute the opening proof: $\mathsf{pf} := \mathsf{MT.Open}(\mathsf{td}, Q)$.
8. Output the argument string $\pi := (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$.

- $\mathbf{V}^{f_{\mathsf{MT}}, f_{\mathsf{FS}}}(\mathbb{x}, \pi)$:

1. Check that $\mathsf{MT.Check}^{f_{\mathsf{MT}}}(\mathsf{rt}, Q, \mathbf{a}, \mathsf{pf}) = 1$.
2. Compute PCP randomness: $\rho := f_{\mathsf{FS}}((\mathbb{x}, \mathsf{rt}, \sigma))$.
3. Check that $\mathbf{V}_{\mathrm{PCP}}^{[Q,\mathbf{a}]}(\mathbb{x}; \rho) = 1$.

The argument prover and argument verifier have the following query complexities:
- $\mathsf{q}_{\mathbf{P}}(n) = \mathsf{q}_{\mathsf{MT.Commit}}(\mathsf{l}(n)), \mathsf{q}(n)) + 1$,
- $\mathsf{q}_{\mathbf{V}}(n) = \mathsf{q}_{\mathsf{MT.Check}}(\mathsf{l}(n), \mathsf{q}(n)) + 1$.

8.3 UC-friendly completeness

We prove that the Micali construction is UC-friendly complete.

Lemma 8.6. Micali *satisfies UC-friendly completeness with error*

$$\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \ell_{\mathsf{p}} \cdot \epsilon_{\mathbf{P}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{p}} \cdot \mathsf{q}_{\mathbf{P}}(n), t_{\mathsf{p}}) \ .$$

In the above, $\epsilon_{\mathbf{P}}$ is defined as in Claim 8.9.

Proof. We argue that the Micali construction satisfies perfect completeness, has monotone proofs, and has unpredictable queries in Claims 8.7 to 8.9. The lemma then directly follows from Lemma 5.9 (which shows that UC-friendly completeness follows from these properties). □

Claim 8.7. Micali *satisfies perfect completeness (Definition 5.4).*

Proof. This follows from the completeness of Merkle commitment schemes (Lemma 7.1) and the completeness of the PCP (Definition 8.1). □

Claim 8.8. Micali *has monotone proofs (Definition 5.6).*

Proof. The argument verifier in the Micali construction queries only one additional point compared to MT.Check, and this point us previously queried when deriving the PCP verifier's randomness. Combining this with the monotonicity of Merkle commitment schemes (Lemma 7.2) concludes the proof. □

Claim 8.9. Micali *has unpredictable queries (Definition 5.8) with error*

$$\epsilon_{\mathbf{P}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}) := \epsilon_{\mathsf{MT}}(\lambda, \mathsf{l}(n), t_{\mathsf{q}}, t_{\mathsf{p}}) + \frac{t_{\mathsf{p}}}{2^{\mathsf{r}}} \ .$$

In the above, ϵ_{MT} is defined as in Lemma 7.3.

Proof. The Merkle commitment scheme has unpredictable queries (Lemma 7.3) and σ is random in $\{0,1\}^{\mathsf{r}}$. □

8.4 UC-friendly zero knowledge

We prove that the Micali construction satisfies UC-friendly zero knowledge.

Lemma 8.10. *Let* PCP *be (resp. strong) honest-verifier zero knowledge (Definition 8.3) with error* ζ_{PCP}*. Then* Micali *satisfies weak (resp. strong) UC-friendly zero knowledge (Definition 5.10) with error*

$$\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) = \ell_{\mathsf{p}} \cdot \zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}} + \mathsf{so}^{(1)}_{\mathsf{q}}(n, \ell_{\mathsf{p}}) + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}} + \mathsf{so}^{(1)}_{\mathsf{p}}(n, \ell_{\mathsf{p}})) \ .$$

Above:

- $\zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) := \frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\mathsf{r}}} + \zeta_{\mathrm{PCP}}(n) + \zeta_{\mathsf{MT}}(\lambda, \mathsf{l}(n), \mathsf{q}(n), t_{\mathsf{q}}, t_{\mathsf{p}}, 1)$*;*
- ζ_{MT} *is as in Lemma 7.12;*
- $\mathsf{so}^{(1)}_{\mathsf{q}}(n, \ell_{\mathsf{p}}) := \ell_{\mathsf{p}} \cdot \max\{\mathsf{q}_{\mathsf{MT.Commit}}(\mathsf{l}(n)) + 1, 2\mathsf{q}_{\mathsf{MT.Sim}}(\mathsf{l}(n), \mathsf{q}(n))\}$*;*
- $\mathsf{so}^{(1)}_{\mathsf{p}}(n, \ell_{\mathsf{p}}) := 2\ell_{\mathsf{p}} \cdot (\mathsf{p}_{\mathsf{MT.Sim}}(\mathsf{l}(n), \mathsf{q}(n)) + 1)$.

Construction 8.11. Let $\mathbf{S}_{\mathrm{PCP}}$ be the simulator for the PCP (Definition 8.3), and $\mathsf{MT.Sim}$ be the simulator for the Merkle commitment scheme (Lemma 7.6). We construct a simulator $\mathbf{S}$ for UC-friendly zero knowledge.

```
$\mathbf{S}^{f_{\mathsf{MT}}}(\mathbb{x})$:
1. Sample a PCP view: $(\rho, Q, \mathbf{a}, z_{\mathsf{SIM}}) \leftarrow \mathbf{S}_{\mathrm{PCP}}(\mathbb{x})$.
2. Sample simulated root and opening proof: $(\mathsf{rt}, \mathsf{pf}, z_\pi) \leftarrow \mathsf{MT.Sim}^{f_{\mathsf{MT}}}(Q, \mathbf{a})$.
3. Sample a salt: $\sigma \leftarrow \{0,1\}^{\mathsf{r}}$.
4. Program the random oracle: $\mathrm{tr}_{\mathsf{FS}} := ((\mathbb{x}, \mathsf{rt}, \sigma), \rho)$.
5. Set $\pi := (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$
6. Return $(\pi, \mathrm{tr}_{\mathsf{FS}}, (\sigma, z_{\mathsf{SIM}}, z_\pi))$.

$\mathbf{S}(\mathbb{w}, (\sigma, z_{\mathsf{SIM}}, z_\pi))$:
1. Reconstruct PCP prover randomness: $\rho_{\mathbf{P}} \leftarrow \mathbf{S}_{\mathrm{PCP}}(\mathbb{w}, z_{\mathsf{SIM}})$.
2. Rederive the PCP string: $\Pi \leftarrow \mathbf{P}_{\mathrm{PCP}}(\mathbb{x}, \mathbb{w}; \rho_{\mathbf{P}})$.
3. Reconstruct Merkle commitment randomness: $(\rho_{\mathsf{MT}}, \mathrm{tr}_{\mathsf{MT}}) \leftarrow \mathsf{MT.Sim}(\Pi, z_\pi)$.
4. Return $((\rho_{\mathbf{P}}, \rho_{\mathsf{MT}}, \sigma), \mathrm{tr}_{\mathsf{MT}})$.
```

The simulator $\mathbf{S}$ makes $\mathsf{q}_{\mathbf{S}}(n) = 2\mathsf{q}_{\mathsf{MT.Sim}}(\mathsf{l}(n), \mathsf{q}(n))$ queries to the random oracle, and programs $\mathsf{p}_{\mathbf{S}}(n) = \mathsf{p}_{\mathsf{MT.Sim}}(\mathsf{l}(n), \mathsf{q}(n)) + 1$ locations.

Proof. We argue that $\mathbf{S}$ yields the simulation error in the lemma statement.

To do so, we show the following claim.

Claim 8.12. Micali *has weak (resp. strong) simplified UC-friendly zero knowledge (Definition 5.11) against adversaries which make a single prover oracle query, with simulator* $\mathbf{S}$ *(Construction 8.11) and error:*

$$\zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}) = \frac{t_{\mathsf{q}} + t_{\mathsf{p}}}{2^{\mathsf{r}}} + \zeta_{\mathsf{MT}}(\lambda, \mathsf{l}, \mathsf{q}, t_{\mathsf{q}}, t_{\mathsf{p}}, 1) + \zeta_{\mathrm{PCP}}(n) \ .$$

The claim suffices to show our main result, as argued next. By applying Lemma 5.13 we obtain then that Micali satisfies Definition 5.11 against adversaries which make ℓ_{p} prover oracle queries, with the same simulator $\mathbf{S}$ and the following error:

$$\zeta_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) = \ell_{\mathsf{p}} \cdot \zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}} + \mathsf{so}^{(1)}_{\mathsf{q}}(n, \ell_{\mathsf{p}}), t_{\mathsf{p}} + \mathsf{so}^{(1)}_{\mathsf{p}}(n, \ell_{\mathsf{p}})) \ ,$$

where $\mathsf{so}_{\mathsf{q}}^{(1)}, \mathsf{so}_{\mathsf{p}}^{(1)}$ are as in the lemma statement. Finally, by applying Lemma 5.12 we obtain the Micali satisfies Definition 5.10, with again the same simulator and error as in the lemma statement, namely:

$$\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) = \zeta_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}}, \ell_{\mathsf{p}}) \ .$$

We are left argue Claim 8.12 holds, which we do via a sequence of games (defined next).

- $\mathsf{sUCZeroKnowledge}_0$: The "real-world" security game in Definition 5.11. (Recall, this is defined as in Definition 5.10 while limiting the adversary to a single prover query and no verifier or verifier corruption queries)
- EXPA: Modify the prover oracle as follows:
  1. Run $\Pi \leftarrow \mathbf{P}_{\mathrm{PCP}}(\mathbb{x}, \mathbb{w})$, then sample ρ and deduce query-answer sets $Q, \mathbf{a}$ from $\mathbf{V}_{\mathrm{PCP}}^{\Pi}(\mathbb{x}; \rho)$.
  2. Compute the root and the opening as before.
  3. Program the random oracle so that $f_{\mathsf{FS}}(\mathbb{x}, \mathsf{rt}, \sigma) = \rho$.
- EXPB
  - Modify the prover oracle as follows:
    1. Use MT.Sim to obtain $(\mathsf{rt}, \mathsf{pf}, z_{\pi})$ from the query-answer pair induced by $\mathbf{V}_{\mathrm{PCP}}$ instead of using MT.Commit and MT.Open.
    2. Additionally use MT.Sim to obtain Merkle randomness ρ_{MT} using Π as the message, programming the random oracle according to the returned programming list tr.
  - Modify the corruption oracle to return the (simulated) Merkle randomness ρ_{MT} used while generating the proof.
- EXPC
  - Modify the prover oracle to, instead of using the $Q, \mathbf{a}$ from the verifier, use ones obtained from $\mathbf{S}_{\mathrm{PCP}}$.
  - Modify the corruption oracle to use the simulator $\mathbf{S}_{\mathrm{PCP}}$ to obtain simulated prover randomness $\rho_{\mathbf{P}}$.
- $\mathsf{sUCZeroKnowledge}_1^{\mathbf{S}}$: The "ideal-world" security game of Definition 5.11. (Recall, this is defined as in Definition 5.10 while limiting the adversary to a single prover query and no verifier or verifier corruption queries).

REAL is close to EXPA.

$$\Delta_{\mathcal{A}}(\mathsf{sUCZeroKnowledge}_0, \mathsf{EXPA}) \le \frac{t_{\mathsf{q}} + t_{\mathsf{p}}}{2^{\mathsf{r}}} \ .$$

The difference between the two games is that the random oracle in EXPA is programmed on $(\mathbb{x}, \mathsf{rt}, \sigma)$. Since the adversary has $t_{\mathsf{q}} + t_{\mathsf{p}}$ possible queries and σ is uniformly distributed over $\{0,1\}^{\mathsf{r}}$, the probability that the adversary queries or programs $(\mathbb{x}, \mathsf{rt}, \sigma)$ before such point is programmed is at most $\frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\mathsf{r}}}$.

EXPA is close to EXPB.

$$\Delta_{\mathcal{A}}(\mathsf{EXPA}, \mathsf{EXPB}) \le \zeta_{\mathsf{MT}}(\lambda, \mathsf{l}, \mathsf{q}, t_{\mathsf{q}}, t_{\mathsf{p}}, 1) \ .$$

Let $\mathcal{A}$ be an adversary that aims to distinguish between the two games. We construct a new adversary $\mathcal{B}$ against the UC-friendly hiding property of the Merkle commitment scheme (Definition 7.4).

1. Answer $\mathcal{A}$'s random oracle queries to f_{MT} with the random oracle of the experiment, and those to f_{FS} via lazy random oracle simulation.
2. Answer $\mathcal{A}$'s programming oracle queries to f_{MT} with the programming oracle of the experiment, and those to f_{FS} via lazy random oracle simulation (with programming).
3. On a prover oracle query $(\mathbb{x}, \mathbb{w}) \in R$:
   (a) Sample prover randomness $\rho_{\mathbf{P}}$
   (b) Compute $\Pi \leftarrow \mathbf{P}_{\mathrm{PCP}}(\mathbb{x}, \mathbb{w}; \rho_{\mathbf{P}})$.

   (c) Sample ρ and run $\mathbf{V}^{\Pi}_{\mathrm{PCP}}(\mathbb{x};\rho)$ to obtain Q.
   (d) Call the prover oracle of the experiment with (Π, Q) to obtain $(\mathsf{rt}, \mathsf{pf})$.
   (e) Program $f_{\mathsf{FS}}((\mathbb{x}, \mathsf{rt}, \sigma)) = \rho$.
   (f) Reply with $\pi := (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$ where $\mathbf{a} := \Pi[Q]$.
4. On a prover corruption query, call the corruption oracle of the experiment to obtain the Merkle randomness ρ_{MT}. Parse the string of Merkle randomness as a list, and concatenate each of these random strings with the sampled PCP prover randomness $\rho_{\mathbf{P}}$.

Note that $\mathcal{B}$ makes the same number of random oracle queries as $\mathcal{A}$ adversary, and a single prover query. If $\mathcal{B}$ is in the "real-world" security experiment, the view of $\mathcal{A}$ is exactly as in EXPA, otherwise it will be as in EXPB. Thus, any advantage in distinguishing between the two experiments is an advantage against the hiding security game.

EXPB is close to EXPC.

$$\Delta_{\mathcal{A}}(\mathsf{EXPB}, \mathsf{EXPC}) \leq \zeta_{\mathrm{PCP}}(n) \ .$$

In this game hop we replace the PCP query/answer sets with those sampled by the simulator. The statistical distance of the two distributions is bound by ζ_{PCP}. Thus, since the view of the adversary is otherwise identical, the statistical distance of its output in the two games can be at most ζ_{PCP}.

EXPC is IDEAL.

$$\mathsf{EXPC} \equiv \mathsf{sUCZeroKnowledge}^{\mathbf{S}}_{1} \ .$$

The two games are syntactically equal.

□

8.5 UC-friendly knowledge soundness

We show that the Micali construction satisfies UC-friendly knowledge soundness with respect to the simulator for UC-friendly zero knowledge described in Section 8.4.

Lemma 8.13. *Suppose that:*
- PCP *satisfies knowledge soundness with error* κ_{PCP} *(Definition 8.2);*
- MT *has weak (resp. strong) UC-friendly extraction with error* κ_{MT} *with respect to* MT.Sim *(Definition 7.14).*

Then Micali *satisfies weak (resp. strong) UC-friendly knowledge soundness (Definition 5.16) with respect to the simulator* $\mathbf{S}$ *in Construction 8.11 with error*

$$\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \leq \ell_{\mathsf{v}} \cdot \kappa^{(1)}_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}}, \ell_{\mathsf{p}}) \ .$$

where $\kappa^{(1)}_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) \leq (t_{\mathsf{q}} + 1) \cdot \kappa_{\mathrm{PCP}}(n) + \kappa_{\mathsf{MT}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \mathsf{l}, \mathsf{q}, t_{\mathsf{q}} + 1, 1)$.

Construction 8.14. Let MT.Extract be the Merkle extractor in Definition 7.14 and $\mathbf{E}_{\mathrm{PCP}}$ be the PCP extractor in Definition 8.2. We describe an extractor $\mathbf{E}$ for Micali.

```
E(x, π, extTrace):
1. Parse $\pi$ as $(\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$.
2. Denote by $\mathsf{extTrace}_{\mathsf{MT}}$ the queries made to $f_{\mathsf{MT}}$ in extTrace.
3. Compute $(\Pi, \mathsf{td}) := \mathsf{MT.Extract}(\mathsf{rt}, \mathsf{extTrace}_{\mathsf{MT}})$.
4. Compute $\mathbb{w} \leftarrow \mathbf{E}_{\mathrm{PCP}}(\mathbb{x}, \Pi)$.
5. Output $\mathbb{w}$.
```

Proof. We show that the extractor $\mathbf{E}$ in Construction 8.14 has the claimed error.

Recall that Lemma 5.18 reduces UC-friendly knowledge soundness (Definition 5.16) to *single-instance* UC-friendly knowledge soundness (Definition 5.17), a simpler property where the adversary makes a single verifier query. Specifically, if the non-interactive argument satisfies single-instance UC-friendly knowledge soundness with error $\kappa^{(1)}_{\mathrm{ARG}}$ then the non-interactive argument satisfies UC-friendly knowledge soundness with error $\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \leq \ell_{\mathsf{v}} \cdot \kappa^{(1)}_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}}, \ell_{\mathsf{p}})$.

We are left to prove the upper bound on $\kappa^{(1)}_{\mathrm{ARG}}$. Let $\mathcal{A}$ be a $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$-query adversary against single-instance UC-friendly knowledge soundness. We upper bound the following probability:

$$\Pr\left[\begin{array}{l|l} (\mathbb{x}, \mathbb{w}) \notin R & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, b = 1 & \begin{pmatrix} \mathbb{x}, \pi, \\ \mathsf{InstanceList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{pmatrix} \xleftarrow{\mathrm{tr}} \mathsf{sUCKnowledgeSoundness1}^{f}_{\mathbf{S}}(n, \mathcal{A}) \\ \wedge\, \mathrm{tr}_{\mathbf{V}} \cap \mathsf{advProg} = \emptyset & b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi) \\ \wedge\, \mathbb{x} \notin \mathsf{InstanceList} & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array}\right].$$

Massaging the probability statement. We introduce some notation to simplify later reduction steps. Let $\mathsf{ProofList} = ((\mathbb{x}^{(s)}_j, \pi^{(s)}_j))_j$ denote the instances queried by $\mathcal{A}$ to the simulator oracle and their corresponding proofs (for those cases when the simulator oracle does not return $\bot$). Define $\mathsf{RootList} := ((\mathbb{x}^{(s)}_j, \mathsf{rt}^{(s)}_j, \sigma^{(s)}_j))_j$, where $\mathsf{rt}^{(s)}_j$ and $\sigma^{(s)}_j$ are the Merkle commitment and salt in $\pi^{(s)}_j$. Letting $(\mathbb{x}, \pi = (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf}))$ denote the final instance-proof pair output by $\mathcal{A}$, note that $(\mathbb{x}, \mathsf{rt}, \sigma) \in \mathsf{RootList}$ implies that $\mathbb{x} \in \mathsf{InstanceList}$. Hence, the previous probability statement is upper bounded by:

$$\Pr\left[\begin{array}{l|l} (\mathbb{x}, \mathbb{w}) \notin R & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, b = 1 & \begin{pmatrix} \mathbb{x}, \\ \pi = (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf}), \\ \mathsf{RootList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{pmatrix} \xleftarrow{\mathrm{tr}} \mathsf{sUCKnowledgeSoundness1}^{f}_{\mathbf{S}}(n, \mathcal{A}) \\ \wedge\, \mathrm{tr}_{\mathbf{V}} \cap \mathsf{advProg} = \emptyset & b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi) \\ \wedge\, (\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{RootList} & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array}\right].$$

To simplify probability statements, we slightly abuse notation in that the game sUCKnowledgeSoundness1 now returns RootList instead of InstanceList.

Next, for notational convenience, we define an algorithm Check that captures the winning conditions other than the PCP verifier accepting.

```
Check^{f_MT}($\mathbb{x}, \pi = (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$, advProg, RootList):
1. Denote by $\mathsf{advProg}_{\mathsf{FS}}, \mathsf{advProg}_{\mathsf{MT}}$ the queries to $f_{\mathsf{FS}}, f_{\mathsf{MT}}$ in advProg.
2. Compute $b_{\mathsf{MT}} \xleftarrow{\mathrm{tr}_{\mathsf{check}}} \mathsf{MT.Check}^{f_{\mathsf{MT}}}(\mathsf{rt}, Q, \mathbf{a}, \mathsf{pf})$.
3. Check that:
   – $b_{\mathsf{MT}} = 1$;
   – $\mathrm{tr}_{\mathsf{check}} \cap \mathsf{advProg}_{\mathsf{MT}} = \emptyset$;
   – $(\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{advProg}_{\mathsf{FS}}$;
   – $(\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{RootList}$.
```

The Micali verifier $\mathbf{V}$ queries the Fiat–Shamir oracle at $(\mathbb{x}, \mathsf{rt}, \sigma)$ and the Merkle commitment oracle, so $\mathrm{tr}_{\mathbf{V}} \cap \mathsf{advProg} = \emptyset$ can be rewritten as $\mathrm{tr}_{\mathsf{check}} \cap \mathsf{advProg}_{\mathsf{MT}} = \emptyset$ and $(\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{advProg}_{\mathsf{FS}}$.

Thus, we can rewrite the previous probability statement as:

$$\Pr\left[\begin{array}{l|l} & f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ (\mathbb{x}, \mathbb{w}) \notin R & \left(\begin{array}{c} \mathbb{x}, \\ \pi := (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf}), \\ \mathsf{RootList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{array}\right) \xleftarrow{\mathrm{tr}_{\mathsf{MT}}, \mathrm{tr}_{\mathsf{FS}}} \mathsf{sUCKnowledgeSoundness}^{f}_{\mathbf{S}}(n, \mathcal{A}) \\ \wedge\, b = 1 & b := \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList}) \\ \wedge\, \mathbf{V}^{[Q,\mathbf{a}]}_{\mathrm{PCP}}(\mathbb{x}; \rho) = 1 & \rho := f_{\mathsf{FS}}[\mathrm{tr}_{\mathsf{FS}}](\mathbb{x}, \mathsf{rt}, \sigma) \\ & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array}\right].$$

Reducing to state-restoration knowledge soundness. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against the state-restoration game of PCP. We denote in red steps not used in the reduction but used in the analysis.

```
B^{f_MT}(A):
1. Initialize empty $\mathrm{tr}_{\mathsf{MT}}, \mathrm{tr}_{\mathsf{FS}}, \mathsf{advProg}, \mathsf{extTrace}, \mathsf{ProofList}$, $\mathsf{Random}_{\mathbf{P}}$.
2. Run $\mathcal{A}$, answering queries as follows:
   – When $\mathcal{A}$ performs a query to $f_{\mathsf{MT}}$, record the query and answer in $\mathrm{tr}_{\mathsf{MT}}, \mathsf{extTrace}$.
   – When $\mathcal{A}$ performs the $j$-th query $q_j$ to $f_{\mathsf{FS}}$:
      (a) If $\exists\, \mathsf{qid}, y$ such that $(\mathsf{qid}, q_j, y) \in \mathrm{tr}_{\mathsf{FS}}$ then return $y$.
      (b) Parse $q_j$ as $(\mathbb{x}_j, \mathsf{rt}_j, \sigma_j)$. (If this fails, answer the query with a lazily sampled random oracle.)
      (c) Denote by $\mathsf{extTrace}_j$ the queries in $\mathsf{extTrace}$ added since the last execution of $\mathsf{MT.MultiExtract}$.
      (d) Compute $(\Pi_j, \mathsf{td}_j) := \mathsf{MT.MultiExtract}(\mathsf{rt}_j, \mathsf{extTrace}_j \setminus \mathsf{advProg})$.
      (e) Submit $(\mathbb{x}_j, \Pi_j, (\mathsf{rt}_j, \sigma_j))$ to the state-restoration game, obtaining randomness $\rho_j$.
      (f) Append $(\mathsf{query}, (\mathbb{x}_j, \mathsf{rt}, \sigma_j), \rho_j)$ to $\mathrm{tr}_{\mathsf{FS}}$.
      (g) Return $\rho_j$.
   – When $\mathcal{A}$ makes a programming query $\mathsf{trace}_{\mathsf{prog}}$ to $f_{\mathsf{FS}}$ or $f_{\mathsf{MT}}$:
      (a) Partition $\mathsf{trace}_{\mathsf{prog}}$ into $\mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$ and $\mathsf{trace}^{\mathsf{FS}}_{\mathsf{prog}}$ depending on the queried oracle.
      (b) If there exist $(x, y) \in \mathsf{trace}^{\mathsf{FS}}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}_{\mathsf{FS}}$ with $x_i = x$, return $0$.
      (c) If there exist $(x, y) \in \mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}_{\mathsf{MT}}$ with $x_i = x$, return $0$.
      (d) Else append $((\mathsf{prog}, x, y))_{(x,y) \in \mathsf{trace}^{\mathsf{FS}}_{\mathsf{prog}}}$ to $\mathrm{tr}_{\mathsf{FS}}$, $((\mathsf{prog}, x, y))_{(x,y) \in \mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}}$ to $\mathrm{tr}_{\mathsf{MT}}$ (and add both
          to $\mathsf{advProg}$) and return $1$.
   – When $\mathcal{A}$ requests the $j$-th proof for $(\mathbb{x}^{(s)}_j, \mathbb{w}^{(s)}_j) \in R$:
      (a) Compute $(\pi^{(s)}_j, \mathrm{tr}, z_\pi) \xleftarrow{\mathrm{tr}_{\mathbf{S}}} \mathbf{S}^{f_{\mathsf{MT}}}(\mathbb{x}_j{}^{(s)})$.
      (b) Compute $(\rho_{\mathbf{P}}, \mathrm{tr}') \xleftarrow{\mathrm{tr}'_{\mathbf{S}}} \mathbf{S}(\mathbb{w}, z_\pi)$.
      (c) Program the oracles $f_{\mathsf{FS}}, f_{\mathsf{MT}}$ according to $\mathrm{tr}, \mathrm{tr}'$, outputting $\bot$ if the programming fails.
      (d) Set $\mathsf{extTrace} := \mathsf{extTrace} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}}$.
      (e) Append $(\mathbb{x}^{(s)}_j, \pi^{(s)}_j)$ to $\mathsf{ProofList}$.
      (f) Append $\rho_{\mathbf{P}}$ to $\mathsf{Random}_{\mathbf{P}}$.
   – When $\mathcal{A}$ makes a corruption query, return $\mathsf{Random}_{\mathbf{P}}$ (and stop answering further simulator or
     corruption queries).
3. The adversary $\mathcal{A}$ produces its final output $(\mathbb{x}, \pi = (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf}))$.
```

4. Denote by $\mathsf{extTrace}'$ the queries in $\mathsf{extTrace}$ added since the last execution of $\mathsf{MT.MultiExtract}$.
5. Compute $(\Pi, \mathsf{td}) := \mathsf{MT.MultiExtract}(\mathsf{rt}, \mathsf{extTrace}' \setminus \mathsf{advProg})$.
6. Derive $\mathsf{RootList}$ from $\mathsf{ProofList}$.
7. Set $b := \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList})$.
8. Return $(\mathbb{x}, \Pi, (\mathsf{rt}, \sigma))$.

We argue that the view of $\mathcal{A}$ when run within $\mathcal{B}$ in the state-restoration game $\mathsf{Game}_{\mathsf{sr}}(\mathcal{B}^{f_{\mathsf{MT}}}(\mathcal{A}), \mathsf{rnd}, s)$ is identical to the view of $\mathcal{A}$ in the single-instance UC-friendly knowledge soundness game $\mathsf{sUCKnowledgeSoundness1}^{f}_{\mathbf{S}}(n, \mathcal{A})$. We consider each type of query that $\mathcal{A}$ makes.

- Queries to f_{MT}. The reduction adversary $\mathcal{B}$ answers queries to f_{MT} faithfully, so the distribution of answers to queries to f_{MT} is the same as in the (single-instance) UC-friendly knowledge soundness game.
- Programming queries to $f_{\mathsf{FS}}, f_{\mathsf{MT}}$. Programming is perfectly simulated by $\mathcal{B}$. In the case of f_{MT}, $\mathcal{B}$ implements the programming logic. In the case of f_{FS}, $\mathcal{B}$ stores the move-answer pairs in the state-restoration game thus far, which also allows it to faithfully answer programming queries.
- Queries to f_{FS}. Queries to f_{FS} that are not successfully parsed are answered by a (lazily sampled) random oracle, so those queries have identically distributed answers in both games. Queries that are successfully parsed (and have not been previously queried) are forwarded to the state-restoration game, which returns uniformly distributed randomness. Duplicate queries are also handled consistently by $\mathcal{B}$. Thus, all these queries are answered as in the single-instance UC-friendly knowledge soundness game.
- Simulator queries and prover corruption queries. These are answered identically in both games, as the reduction adversary faithfully simulates the oracle.

We define the event E_{extr} as follows:

$$\left[\begin{array}{l|l} b = 1 \wedge \Pi[Q] \neq \mathbf{a} & f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ \text{or} & (\mathbb{x}, \pi, \mathsf{RootList}, \mathrm{tr}_{\mathbf{S}}) \leftarrow \mathsf{sUCKnowledgeSoundness}^{f}_{\mathbf{S}}(n, \mathcal{A}) \\ \exists\, j \text{ s.t. } \mathsf{rt} = \mathsf{rt}_j \wedge \Pi \neq \Pi_j & (\mathbb{x}, \Pi, (\mathsf{rt}, \sigma), \rho) \xleftarrow{b, (\mathsf{rt}_j, \Pi_j)_j} \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}^{f_{\mathsf{MT}}}(\mathcal{A}), f_{\mathsf{FS}}, \lambda + \mathsf{r}) \end{array}\right]. \quad (1)$$

We will soon argue that overloaded variables in the above experiment are identical so we do not disambiguate them in the notation.

We argue that the following two distributions are identical:

$$\left\{ \begin{array}{l|l} \dfrac{(\mathbb{x}, \Pi, \rho, b_f, \mathrm{tr}_{\mathsf{MT}})}{\begin{array}{l}\text{conditioned on:} \\ (\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{RootList} \\ \wedge\ (\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{advProg} \\ \wedge\ \neg E_{\mathsf{extr}}\end{array}} & \begin{array}{l} f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ \left(\begin{array}{c} \mathbb{x}, \\ \pi := (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf}), \\ \mathsf{RootList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{array}\right) \xleftarrow{\mathrm{tr}_{\mathsf{MT}}, \mathrm{tr}_{\mathsf{FS}}} \mathsf{sUCKnowledgeSoundness}^{f}_{\mathbf{S}}(n, \mathcal{A}) \\ b := \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList}) \\ \rho := f_{\mathsf{FS}}[\mathrm{tr}_{\mathsf{FS}}](\mathbb{x}, \mathsf{rt}, \sigma) \\ b_f := b \wedge \mathbf{V}^{[Q, \mathbf{a}]}_{\mathrm{PCP}}(\mathbb{x}; \rho) \\ \Pi := \mathsf{MT.Extract}(\mathsf{rt}, \mathsf{extTrace}_{\mathsf{MT}} \setminus \mathsf{advProg}) \end{array} \end{array} \right\}$$

and (using the $\leftarrow$ notation to bring into scope internal variable of $\mathcal{B}$)

$$\left\{ \begin{array}{l|l} \frac{(\mathbb{x}, \Pi, \rho, b_f, \mathrm{tr}_{\mathsf{MT}})}{\begin{array}{l}\text{conditioned on:} \\ (\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{RootList} \\ \wedge\ (\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{advProg} \\ \wedge\ \neg E_{\mathsf{extr}}\end{array}} & \begin{array}{l} f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ (\mathbb{x}, \Pi, (\mathsf{rt}, \sigma), \rho) \xleftarrow{b, \mathrm{tr}_{\mathsf{MT}}, \mathsf{advProg}, \mathsf{RootList}} \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}^{f_{\mathsf{MT}}}(\mathcal{A}), f_{\mathsf{FS}}, \lambda + \mathsf{r}) \\ b_f := b \wedge \mathbf{V}_{\mathrm{PCP}}^{\Pi}(\mathbb{x}; \rho) \end{array} \end{array} \right\} .$$

We discuss each random variable in turn.

- $\mathbb{x}, \mathrm{tr}_{\mathsf{MT}}$. The instance $\mathbb{x}$ and the Merkle trace $\mathrm{tr}_{\mathsf{MT}}$ are determined by the output and trace of the adversary $\mathcal{A}$. We have argued that the view of $\mathcal{A}$ is identical in both experiments, so the distribution of these random variables is also identical.
- Π. In both experiments, $\Pi := \mathsf{MT.Extract}(\mathsf{rt}, \mathsf{advTrace}_{\mathsf{MT}}, \mathsf{simTrace}_{\mathsf{MT}})$. As in the previous point, rt, $\mathsf{advTrace}_{\mathsf{MT}}$ are directly determined by the adversary $\mathcal{A}$ output and trace, and thus are identically distributed in both games. $\mathsf{simTrace}_{\mathsf{MT}}$ is also identically distributed, since, as argued before, $\mathcal{B}$ faithfully simulates the simulator oracle, and $\mathcal{A}$ makes identically distributed queries in both games. Thus, Π has the same distribution in both games.
- ρ. The distribution of $(\mathbb{x}, \mathsf{rt}, \sigma)$ is identical in both experiments. Note that, since $\mathbf{S}$ in Construction 8.11 only programs f_{FS} on $(\mathbb{x}, \mathsf{rt}, \sigma)$, the Fiat–Shamir oracle is only programmed on points in $\mathsf{advProg}$ or in $\mathsf{RootList}$. So, since $(\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{RootList} \cup \mathsf{advProg}$, in the first experiment $\rho = f_{\mathsf{FS}}[\mathrm{tr}_{\mathsf{FS}}](\mathbb{x}, \mathsf{rt}, \sigma) = f_{\mathsf{FS}}(\mathbb{x}, \mathsf{rt}, \sigma)$. We distinguish two cases:
  - $(\mathbb{x}, \mathsf{rt}, \sigma)$ was previously queried to f_{FS} by $\mathcal{A}$. Let $j \in [t_{\mathsf{q}}]$ denote the index of the first such query. In the first experiment, ρ is a uniformly sampled string, consistent to the j-th query to f_{FS}. In the second experiment, ρ is obtained by querying $(\mathbb{x}, \Pi, (\mathsf{rt}, \sigma))$ to f_{FS}. Letting Π_j denote the PCP string extracted in the j-th query, since $\neg E_{\mathsf{extr}}$ holds and $\mathsf{rt}_j = \mathsf{rt}$, we deduce that $\Pi_j = \Pi$. Thus, both queries map to the same state-restoration move $(\mathbb{x}, \Pi, (\mathsf{rt}, \sigma))$. ρ is also uniformly distributed as desired.
  - $(\mathbb{x}, \mathsf{rt}, \sigma)$ was not previously queried to f_{FS} by $\mathcal{A}$. In the first experiment ρ is a string sampled uniformly at random. In the second experiment, ρ is obtained by querying f_{FS} at $(\mathbb{x}, \Pi, (\mathsf{rt}, \sigma))$ which is also a fresh new state-restoration move. Thus, ρ is also a uniformly random string.
- b_f. In both experiments, b is computed by running Check with inputs that are identically distributed, so the distribution of b is identical. If $b = 0$, $b_f = 0$ in both experiments. Consider then the case when $b = 1$. In the bottom experiment, since $b = 1$ and $\neg E_{\mathsf{extr}}$ holds, it must be that $\Pi[Q] = \mathbf{a}$ and so $\mathbf{V}_{\mathrm{PCP}}^{\Pi}(\mathbb{x}; \rho) = \mathbf{V}_{\mathrm{PCP}}^{[Q,\mathbf{a}]}(\mathbb{x}; \rho)$. Since $\mathbb{x}, \Pi, \rho$ are identically distributed in both experiment, b_f has the same distribution in both experiments.

Since the two distributions are identical, we obtain:

$$\Pr \left[\begin{array}{l|l} \begin{array}{l} (\mathbb{x}, \mathbb{w}) \notin R \\ \wedge\ b = 1 \\ \wedge\ \mathbf{V}_{\mathrm{PCP}}^{[Q,\mathbf{a}]}(\mathbb{x}; \rho) = 1 \\ \wedge\ (\mathbb{x}, \mathsf{rt}, \sigma) \notin \mathsf{RootList} \end{array} & \begin{array}{l} f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ \left(\begin{array}{c} \mathbb{x}, \\ \pi := (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf}), \\ \mathsf{RootList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{array} \right) \xleftarrow{\mathrm{tr}_{\mathsf{MT}}, \mathrm{tr}_{\mathsf{FS}}} \mathsf{sUCKnowledgeSoundness}_{\mathbf{S}}^{f}(n, \mathcal{A}) \\ b := \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList}) \\ \rho := f_{\mathsf{FS}}[\mathrm{tr}_{\mathsf{FS}}](\mathbb{x}, \mathsf{rt}, \sigma) \\ \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array} \end{array} \right]$$

$$\leq \Pr\left[\begin{array}{l|c} (\mathbb{x},\mathbb{w}) \notin R & \\ \wedge\ b = 1 & f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ \wedge\ \mathbf{V}^{\Pi}_{\mathrm{PCP}}(\mathbb{x};\rho) = 1 & (\mathbb{x},\Pi,(\mathsf{rt},\sigma),\rho) \xleftarrow{b,\mathsf{RootList}} \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}^{f_{\mathsf{MT}}}(\mathcal{A}), f_{\mathsf{FS}}, \lambda + \mathsf{r}) \\ \wedge\ (\mathbb{x},\mathsf{rt},\sigma) \notin \mathsf{RootList} & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{PCP}}(\mathbb{x},\Pi) \\ \wedge\ \neg E_{\mathsf{extr}} & \end{array}\right] + \Pr\left[E_{\mathsf{extr}}\right]$$

$$\leq \Pr\left[\begin{array}{l|l} (\mathbb{x},\mathbb{w}) \notin R & f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ \wedge\ \mathbf{V}^{\Pi}_{\mathrm{PCP}}(\mathbb{x};\rho) = 1 & (\mathbb{x},\Pi,(\mathsf{rt},\sigma),\rho) \leftarrow \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}^{f_{\mathsf{MT}}}(\mathcal{A}), f_{\mathsf{FS}}, \lambda + \mathsf{r}) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{PCP}}(\mathbb{x},\Pi) \end{array}\right] + \Pr\left[E_{\mathsf{extr}}\right] \ .$$

The first term is bounded above by the PCP state restoration error $\kappa_{\mathsf{sr}}(n, t_{\mathsf{q}}, \lambda + \mathsf{r})$, which, in turn, by Claim 8.5 is at most $(t_{\mathsf{q}} + 1) \cdot \kappa_{\mathrm{PCP}}(n)$.

Bounding Merkle extraction error. We are left to upper bound $\Pr[E_{\mathsf{extr}}]$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}_{\mathsf{MT}}$ against the UC-friendly extraction property of Merkle commitment schemes (Definition 7.14), as follows. We highlight differences from the adversary $\mathcal{B}$ against the PCP state-restoration game in red.

```
$\mathcal{B}^{f_{\mathsf{FS}}}_{\mathsf{MT}}(\mathcal{A})$:
1. Initialize empty $\mathrm{tr}_{\mathsf{FS}}$, $\mathsf{advProg}$, $\mathsf{ProofList}$, , $\mathsf{aux}$.
2. Run $\mathcal{A}$, answering queries as follows:
   – When $\mathcal{A}$ performs a query to $f_{\mathsf{MT}}$, forward the query to the random oracle of the game.
   – When $\mathcal{A}$ performs the $j$-th query $q_j$ to $f_{\mathsf{FS}}$:
      (a) If $\exists\ \mathsf{qid}, y$ such that $(\mathsf{qid}, q_j, y) \in \mathrm{tr}_{\mathsf{FS}}$ then return $y$.
      (b) Parse $q_j$ as $(\mathbb{x}_j, \mathsf{rt}_j, \sigma_j)$. (If this fails, answer the query with a lazily sampled random oracle.)
      (c) Submit $\mathsf{rt}_j$ as a root query to the game, obtaining $\Pi_j$.
      (d) Let $\rho_j := f_{\mathsf{FS}}(\mathbb{x}_j, \Pi_j, (\mathsf{rt}_j, \sigma_j))$.
      (e) Append $(\mathsf{query}, (\mathbb{x}_j, \mathsf{rt}, \sigma_j), \rho_j)$ to $\mathrm{tr}_{\mathsf{FS}}$.
      (f) Return $\rho_j$.
   – When $\mathcal{A}$ makes a programming query $\mathsf{trace}_{\mathsf{prog}}$ to $f_{\mathsf{FS}}$ or $f_{\mathsf{MT}}$:
      (a) Partition $\mathsf{trace}_{\mathsf{prog}}$ into $\mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$ and $\mathsf{trace}^{\mathsf{FS}}_{\mathsf{prog}}$ depending on which oracle was queried.
      (b) If there exist $(x, y) \in \mathsf{trace}^{\mathsf{FS}}_{\mathsf{prog}}$ and $(\mathsf{qid}_i, x_i, y_i) \in \mathrm{tr}_{\mathsf{FS}}$ with $x_i = x$, return $0$.
      (c) Make a programming query to the programming oracle of the game with $\mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$, if this returns
          $0$, return $0$.
      (d) Else append $((\mathsf{prog}, x, y))_{(x,y) \in \mathsf{trace}^{\mathsf{FS}}_{\mathsf{prog}}}$ to $\mathrm{tr}_{\mathsf{FS}}$.
      (e) Append $\mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}, \mathsf{trace}^{\mathsf{FS}}_{\mathsf{prog}}$ to $\mathsf{advProg}$, and return $1$.
   – When $\mathcal{A}$ requests a proof for $(\mathbb{x}, \mathbb{w}) \in R$:
      (a) Sample a PCP view $(\rho, Q, \mathbf{a}, z_{\mathsf{SIM}}) \leftarrow \mathbf{S}_{\mathrm{PCP}}(\mathbb{x})$.
      (b) Set $\Pi$ so that $\Pi[q] = \mathbf{a}[q]$ for every $q \in Q$ and $\Pi[q] = \bot$ otherwise.
      (c) Reconstruct PCP prover randomness $\rho_{\mathbf{P}} \leftarrow \mathbf{S}_{\mathrm{PCP}}(\mathbb{w}, z_{\mathsf{SIM}})$.
      (d) Rederive the PCP string $\Pi \leftarrow \mathbf{P}_{\mathrm{PCP}}(\mathbb{x}, \mathbb{w}; \rho_{\mathbf{P}})$.
      (e) Submit $(\Pi, Q)$ to the simulator oracle of the game, obtaining $(\mathsf{rt}, \mathsf{pf})$. (If instead the oracle
          returns $\bot$, return $\bot$.)
      (f) Sample a salt string $\sigma \leftarrow \{0,1\}^{\mathsf{r}}$.
      (g) If $(\mathbb{x}, \mathsf{rt}, \sigma) \in \mathrm{tr}_{\mathsf{FS}}$, return $\bot$.
      (h) Append $(\mathbb{x}, \mathsf{rt}, \sigma)$ to $\mathrm{tr}_{\mathsf{FS}}$.
      (i) Set $\pi := (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$.
      (j) Append $(\mathbb{x}, \pi)$ to $\mathsf{ProofList}$.
      (k) Append $\rho_{\mathbf{P}}$ to $\mathsf{aux}$.
```

    (l) Return π.
  - When $\mathcal{A}$ makes a corruption query, query the corruption oracle of the game, which returns a list of randomnesses, concatenate them with the PCP prover randomness in aux, return the concatenated list $\mathsf{Random}_{\mathbf{P}}$ (and stop answering further simulator or corruption queries).
3. Derive $\mathsf{RootList}$ from $\mathsf{ProofList}$.
4. Set $b := \mathsf{Check}^{f_{\mathsf{MT}}[\mathsf{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList})$.
5. The adversary $\mathcal{A}$ produces its final output $(\mathbb{x}, \pi = (\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf}))$.
6. Make a root query rt.
7. Return $(i, Q, \mathbf{a}, \mathsf{pf})$, where i is the number of root queries performed during the execution.

We use $\mathcal{B}_{\mathsf{MT}}$ to bound the probability of E_{extr}. We define the event E'_{extr} to be the following one:

$$\left[\begin{array}{l|l} b = 1 \wedge \Pi[Q] \neq \mathbf{a} & f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ \text{or} & (\mathbb{x}, \Pi, (\mathsf{rt}, \sigma), \rho) \leftarrow \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}^{f_{\mathsf{MT}}}(\mathcal{A}), f_{\mathsf{FS}}, \lambda + \mathsf{r}) \\ \exists\, j \text{ s.t. } \mathsf{rt} = \mathsf{rt}_j \wedge \Pi \neq \Pi_j & \mathsf{advWin} \xleftarrow{b, Q, \mathbf{a}, \Pi, (\mathsf{rt}_j, \Pi_j)_j} s\mathsf{UCMerkleExtraction}^{f_{\mathsf{MT}}}(\mathcal{B}_{\mathsf{MT}}^{f_{\mathsf{FS}}}(\mathcal{A}), \mathsf{l}, \mathsf{q}, t_{\mathsf{q}} + 1, 1) \end{array}\right] . \tag{2}$$

As before, we will soon argue that overloaded variables are identical, so we do not disambiguate.

First note that the distribution of the answer of queries of $\mathcal{A}$ in $\mathcal{B}_{\mathsf{MT}}$ is the same as when run within $\mathcal{B}$ in the state-restoration game.

- Queries f_{MT} are forwarded to the random oracle of the game.
- Queries to f_{FS} are answered by extracting a PCP proof (by making a root query, which runs $\mathsf{MT.MultiExtract}$) and then querying the oracle f_{FS}.
- Programming queries are answered by first checking if the Fiat–Shamir programming requests have been previously determined, if not, the Merkle programming requests are forwarded to the programming oracle of the game, and only then the Fiat–Shamir oracle is programmed. This ensure they are handled as in the state-restoration game (maintaining atomicity).
- Simulator queries are identically distributed to those in a execution of $\mathcal{B}$ within the state-restoration game. In the weak UC-friendly extraction game, this is because the reduction faithfully simulates the simulator oracle (replacing the execution of $\mathsf{MT.Sim}$ with a query to the simulator oracle of the game). In the strong UC-friendly extraction game, the order in which the PCP strong simulator and the Merkle simulator are run is changed, but this is immaterial since the PCP strong simulator does not query/program the random oracle and Merkle strong simulator only program f_{MT}.

Write $\mathsf{rt}_1, \dots, \mathsf{rt}_i = \mathsf{rt}$ for the list of roots queried by $\mathcal{B}_{\mathsf{MT}}$, and $\Pi_1, \dots, \Pi_i$ for the corresponding extracted strings (note that $i \in [t_{\mathsf{q}} + 1]$). We argue that, in the experiments of Equations (1) and (2), the distribution of $b, Q, \mathbf{a}, \mathsf{rt}_1, \dots, \mathsf{rt}_i, \Pi_1, \dots, \Pi_i$ are identically distributed. Since $Q, \mathbf{a}, \mathsf{rt}_1, \dots, \mathsf{rt}_i$ are directly determined by the output of $\mathcal{A}$, they are identically distributed in both experiments. Further, in both experiments, the traces $\mathsf{advTrace}, \mathsf{simTrace}$ are identically distributed (and also are their restrictions $\mathsf{advTrace}_j, \mathsf{simTrace}_j$). Thus, since $\Pi_j := \mathsf{MT.MultiExtract}(\mathsf{rt}_j, \mathsf{advTrace}_j, \mathsf{simTrace}_j)$, the PCP strings are also identically distributed. Finally, b in both experiments is a deterministic function of the variables mentioned above, and thus also has the correct distribution in both experiments.

Since the conditions for which $E_{\mathsf{extr}}, E'_{\mathsf{extr}}$ hold are identical, it must be that $\Pr[E_{\mathsf{extr}}] = \Pr[E'_{\mathsf{extr}}]$.

The proof concludes by noting that E'_{extr} is a relaxation of the conditions required for $\mathsf{advWin} = 1$ in the UC-friendly extraction game. Indeed, E'_{extr} holds if: (i) $b = 1$ and $\Pi_i[Q_i] \neq \mathbf{a}_i$; or (ii) $\exists\, j$ such that $\mathsf{rt}_i = \mathsf{rt}_j$ and $\Pi_i \neq \Pi_j$.

If the first item holds, we have that $b_{\mathsf{MT}} = 1$, $\mathsf{advProg} \cap \mathsf{tr}_{\mathsf{check}} = \emptyset$, and $\Pi_i[Q_i] \neq \mathbf{a}_i$, and thus Item 6b in Definition 7.14 holds, and thus $\mathsf{advWin} = 1$. If the second item holds then $\mathsf{advWin} = 1$, because if $\exists\, j$ s.t.

$\mathsf{rt}_i = \mathsf{rt}_j$ such that $\Pi_i \neq \Pi_j$ it must hold that $\exists\, i', j$ s.t. $\mathsf{rt}_{i'} = \mathsf{rt}_j$ and $\Pi_{i'} \neq \Pi_j$ (namely by having $i = i'$).

We conclude that

$$\begin{aligned}
\Pr\left[E_{\mathsf{extr}}\right] &= \Pr\left[E'_{\mathsf{extr}}\right] \\
&\leq \Pr\left[\mathsf{advWin} = 1 \,\middle|\, \begin{array}{l} f = (f_{\mathsf{MT}}, f_{\mathsf{FS}}) \leftarrow \mathcal{U}(\lambda) \\ \mathsf{advWin} \leftarrow s\mathsf{UCMerkleExtraction}^{f_{\mathsf{MT}}}(\mathcal{B}_{\mathsf{MT}}^{f_{\mathsf{FS}}}(\mathcal{A}), \mathsf{l}, \mathsf{q}, t_{\mathsf{q}} + 1, 1) \end{array}\right] \\
&\leq \kappa_{\mathsf{MT}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \mathsf{l}, \mathsf{q}, t_{\mathsf{q}} + 1, 1) \ ,
\end{aligned}$$

where the last inequality follows since $\mathcal{B}_{\mathsf{MT}}$ makes at most t_{q} random oracle queries[13] and t_{p} programming queries to f_{MT}, submits at most $t_{\mathsf{q}} + 1$ roots, makes at most ℓ_{p} simulator queries, and outputs a single opening. By UC-friendly extraction, the probability that the advWin flag (as defined in that game) is set is at most $\kappa_{\mathsf{MT}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \mathsf{l}, \mathsf{q}, t_{\mathsf{q}} + 1, 1)$. □

8.6 UC-secure zkSNARKs from Micali

We combine the results in Sections 8.3 to 8.5 to show that, when instantiated with a suitable PCP, the Micali construction yields a UC-secure zkSNARK.

Theorem 8.15. *Let* PCP *be a probabilistically checkable proof with:*

- *(resp. strong) honest-verifier zero knowledge (Definition 8.3) with error ζ_{PCP}.*
- *knowledge soundness (Definition 8.2) with error κ_{PCP}.*

Set $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$ *and* $\mathsf{ARG} := \mathsf{Micali}[\mathsf{PCP}, \mathsf{r}]$. *Then* $\Pi_a[\mathsf{ARG}]$ $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-UC-realizes* $\mathcal{F}_{a\mathrm{ARG}}$ *in the* GRO*-hybrid model with no simulation overhead and error*

$$z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$$

In the above we let:

- $z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) + \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) + \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Theorem 6.1,*
- $\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.6.*
- $\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.10,*
- $\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.13.*

Proof. By Lemma 8.6 we obtain that the Micali construction has strong UC-completeness. By Lemma 8.10, we show that it is weak (resp. strong) UC-friendly zero knowledge.We apply Claim 8.5 to conclude that PCP is state-restoration knowledge sound, and then Lemma 8.13 to conclude weak (resp. strong) UC-friendly knowledge soundness with respect to the simulator from the UC-friendly zero knowledge proof. Applying then Theorem 6.1 concludes the result. □

[13]Formally, $\mathcal{B}_{\mathsf{MT}}$ makes $t_{\mathsf{q}} + \mathsf{q}_{\mathsf{MT.Check}}$ random oracle queries, as those are required to compute Check. However, since b is only used to define E'_{extr}, we can modify the reduction adversary to avoid performing the extra queries.

9 The BCS construction is UC-secure

We prove that the BCS construction [BCS16], when instantiated with a suitable IOP, yields a zkSNARK that is UC-secure. In Section 9.1 we recall the definition of an IOP. In Section 9.2 we recall the BCS construction. In Section 9.3 we prove that the BCS construction satisfies UC-friendly completeness. In Section 9.4 we prove that the BCS construction satisfies UC-friendly zero knowledge. In Section 9.5 we prove that the BCS construction satisfies UC-friendly knowledge soundness. Finally, in Section 9.6 we combine these results to deduce UC-security of the BCS construction.

9.1 Interactive oracle proofs

An *interactive oracle proof* is a tuple $\mathsf{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ with the following syntax.

- $\mathbf{P}_{\text{IOP}}$ is a next message function. On its first invocation, $\mathbf{P}_{\text{IOP}}(\mathbb{x}, \mathbb{w}) \to (\Pi_1, \mathsf{aux}_1)$ receives as input an instance-witness pair $(\mathbb{x}, \mathbb{w})$ and outputs a proof string Π_1 and auxiliary state aux_1. For $i \in \{2, \dots, \mathsf{k}\}$, on its i-th invocation, $\mathbf{P}_{\text{IOP}}(\mathsf{aux}_{i-1}, \rho_{i-1}) \to (\Pi_i, \mathsf{aux}_i)$ receives as input auxiliary state aux_{i-1} and a verifier message ρ_{i-1} and outputs a proof Π_i and auxiliary state aux_i.

- $\mathbf{V}_{\text{IOP}}$ is a next message function. On its first invocation, $\mathbf{V}_{\text{IOP}}^{\Pi_1}(\mathbb{x}) \to (\rho_1, \mathsf{aux}_1)$ receives as input an instance $\mathbb{x}$ and oracle access to a proof Π_1 and outputs a verifier message ρ_1 and auxiliary state aux_1. On its i-th invocation for $i > 1$, $\mathbf{V}_{\text{IOP}}^{\Pi_1,\dots,\Pi_i}(\mathsf{aux}_{i-1}) \to (\rho_i, \mathsf{aux}_i)$ takes in auxiliary state aux_{i-1} and oracle access to $\Pi_1, \dots, \Pi_i$ and outputs a verifier message ρ_i and auxiliary state aux_i. On its final invocation, the verifier additionally outputs a decision bit.

We write $\langle \mathbf{P}_{\text{IOP}}(\mathbb{x}, \mathbb{w}), \mathbf{V}_{\text{IOP}}(\mathbb{x}) \rangle$ for the decision bit output by $\mathbf{V}_{\text{IOP}}$ after the interaction of $\mathbf{P}_{\text{IOP}}$ on instance-witness pair $(\mathbb{x}, \mathbb{w})$.

We consider the following efficiency measures, which might be (and generally are) functions of $|\mathbb{x}|$.

- k is the number of proof strings sent by $\mathbf{P}_{\text{IOP}}$.
- Σ is the alphabet used to write symbols of the IOP strings.
- l_i is the number of symbols in the i-th IOP string.
- l is the total number of symbols across all IOP strings.
- q_i is the number of queries that $\mathbf{V}_{\text{IOP}}$ makes to the i-th IOP string.
- q is the total number of queries that $\mathbf{V}_{\text{IOP}}$ makes across all IOP strings.
- $\mathsf{r}_{\mathbf{P}}$ is the number of random bits that $\mathbf{P}_{\text{IOP}}$ uses.
- $\mathsf{r}_{\mathbf{V}}$ is the number of random bits that $\mathbf{V}_{\text{IOP}}$ uses.

Definition 9.1. $\mathsf{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ *has* **perfect completeness** *for a relation R if, for every $(\mathbb{x}, \mathbb{w}) \in R$,*

$$\Pr\left[\langle \mathbf{P}_{\text{IOP}}(\mathbb{x}, \mathbb{w}), \mathbf{V}_{\text{IOP}}(\mathbb{x}) \rangle = 1\right] = 1 \ .$$

In this work we consider only public-coin IOPs.

Definition 9.2. $\mathsf{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ *is* **public-coin** *if every message sent by the verifier is a uniformly random string ρ_i of some prescribed length r_i for $i \in [\mathsf{k}]$. In this case, the decision of an IOP is a function of the instance $\mathbb{x}$, the verifier randomness $(\rho_1, \dots, \rho_{\mathsf{k}})$, and answers to queries to the received IOP strings $(\Pi_1, \dots, \Pi_{\mathsf{k}})$, which we denote by writing*

$$\mathbf{V}_{\text{IOP}}^{\Pi_1,\dots,\Pi_{\mathsf{k}}}(\mathbb{x}, \rho_1, \dots, \rho_{\mathsf{k}}) \ .$$

For a public-coin IOP, we denote by $\mathbf{V}_{\mathrm{IOP}}^{[(Q_1,\mathbf{a}_1),\ldots,(Q_\mathsf{k},\mathbf{a}_\mathsf{k})]}(\mathbb{x},\rho_1,\ldots,\rho_\mathsf{k})$ *the algorithm that runs the IOP verifier with randomness* $(\rho_1,\ldots,\rho_\mathsf{k})$ *answering queries* q *to the* i*-th proof string with* $\mathbf{a}_i[q]$ *if* $q \in Q_i$ *and rejecting otherwise.*

A PCP is a 1-round public-coin IOP, thus the results in these sections subsume the ones we saw previously.

Straightline knowledge soundness of the BCS construction (even as a standalone property) requires the IOP to satisfy a strong security notion: straightline state-restoration knowledge soundness, introduced next.

Definition 9.3. *We define the* **state-restoration game** *as follows:*

$\mathsf{Game}_{\mathsf{sr}}(\mathcal{A},\mathsf{rnd}_1,\ldots\mathsf{rnd}_\mathsf{k},s)$*:*

1. *Repeat until* $\mathcal{A}$ *decides to exit the loop.*
   (a) *Compute* $\rho_i := \mathsf{rnd}_i(\mathbb{x},(\Pi_1,\ldots,\Pi_i),(\sigma_1,\ldots,\sigma_i))$.
   (b) *Send* ρ_i *to* $\mathcal{A}$.
2. *Get* $(\mathbb{x},(\Pi_1,\ldots,\Pi_\mathsf{k}),(\sigma_1,\ldots,\sigma_\mathsf{k}))$ *from* $\mathcal{A}$.
3. *Set* $\rho_i := \mathsf{rnd}_i(\mathbb{x},(\Pi_1,\ldots,\Pi_i),(\sigma_1,\ldots,\sigma_i))$ *for* $i \in [\mathsf{k}]$.
4. *Output* $(\mathbb{x},(\Pi_1,\ldots,\Pi_\mathsf{k}),(\sigma_1,\ldots,\sigma_\mathsf{k}),(\rho_1,\ldots,\rho_\mathsf{k}))$.

We say $\mathcal{A}$ *is* t_{sr}*-move if it enters the loop at most* t_{sr} *times.*

$\mathsf{IOP} = (\mathbf{P}_{\mathrm{IOP}},\mathbf{V}_{\mathrm{IOP}})$ *for a relation* R *has* **(straightline) state-restoration knowledge soundness** *with error* κ_{sr} *if there exists an extractor* $\mathbf{E}_{\mathrm{IOP}}$ *such that for every* t_{sr}*-move* $\mathcal{A}$

$$\Pr\left[\begin{array}{l} |\mathbb{x}| \le n \\ \wedge\ (\mathbb{x},\mathbb{w}) \notin R \\ \wedge\ \mathbf{V}_{\mathrm{IOP}}^{\Pi_1,\ldots,\Pi_\mathsf{k}}(\mathbb{x};\rho_1,\ldots,\rho_\mathsf{k}) = 1 \end{array}\ \middle|\ \begin{array}{l} (\mathsf{rnd}_1,\ldots,\mathsf{rnd}_\mathsf{k}) \leftarrow \mathcal{U}(\mathsf{r}_1,\ldots,\mathsf{r}_\mathsf{k}) \\ \ (\mathbb{x},(\Pi_1,\ldots,\Pi_\mathsf{k}),(\sigma_1,\ldots,\sigma_\mathsf{k}),(\rho_1,\ldots,\rho_\mathsf{k})) \\ \ \leftarrow \mathsf{Game}_{\mathsf{sr}}(\mathcal{A},\mathsf{rnd}_1,\ldots,\mathsf{rnd}_\mathsf{k},s) \\ \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{IOP}}(\mathbb{x},\Pi_1,\ldots,\Pi_\mathsf{k}) \end{array}\right] \le \kappa_{\mathsf{sr}}(n,t_{\mathsf{sr}},s)\ .$$

Definition 9.4. *Let* $\mathsf{IOP} = (\mathbf{P}_{\mathrm{IOP}},\mathbf{V}_{\mathrm{IOP}})$ *be an interactive oracle proof for* R*. The* **joint IOP verifier view** *on the instance-witness pair* $(\mathbb{x},\mathbb{w})$*, denoted as* $\mathsf{jView}_{\mathrm{IOP}}(\mathbf{P}_{\mathrm{IOP}},\mathbf{V}_{\mathrm{IOP}},\mathbb{x},\mathbb{w})$*, is the random variable* $(\mathbb{x},\mathbb{w},\rho_{\mathbf{P}},(\rho_1,\ldots,\rho_\mathsf{k}),(Q_1,\ldots,Q_\mathsf{k}),(\mathbf{a}_1,\ldots,\mathbf{a}_\mathsf{k}))$ *where:*

- $\rho_{\mathbf{P}} \in \{0,1\}^{\mathsf{r}_{\mathbf{P}}}$ *is a choice of randomness for* $\mathbf{P}_{\mathrm{IOP}}$*;*
- $\rho_i \in \{0,1\}^{\mathsf{r}_i}$ *is a choice of randomness for* $\mathbf{V}_{\mathrm{IOP}}$*;*
- $Q_i \subseteq [\mathsf{l}_i]$ *and* $\mathbf{a}_i \in \Sigma_i^Q$ *are the queries and answers of the verifier makes to* Π_i *when running* $\mathbf{V}_{\mathrm{IOP}}^{\Pi_1,\ldots,\Pi_\mathsf{k}}(\mathbb{x};\rho_1,\ldots,\rho_\mathsf{k})$ *where* $(\Pi_1,\mathsf{aux}_1) \leftarrow \mathbf{P}_{\mathrm{IOP}}(\mathbb{x},\mathbb{w};\rho_{\mathbf{P}})$ *and* $(\Pi_j,\mathsf{aux}_j) := \mathbf{P}_{\mathrm{IOP}}(\mathsf{aux}_{j-1},\rho_{j-1})$ *for* $j > 2$.

The **verifier view** *is similarly denoted as* $\mathsf{View}_{\mathrm{IOP}}(\mathbf{P}_{\mathrm{IOP}},\mathbf{V}_{\mathrm{IOP}},\mathbb{x},\mathbb{w})$*, and is obtained by dropping* $\mathbb{w}$ *and* $\rho_{\mathbf{P}}$ *from* $\mathsf{jView}_{\mathrm{IOP}}$.

IOP *has* **honest-verifier zero knowledge with error** ζ_{IOP} *if there exists a probabilistic polynomial time algorithm* $\mathbf{S}_{\mathrm{IOP}}$ *such that, for every* $(\mathbb{x},\mathbb{w}) \in R$*,* $\zeta_{\mathrm{IOP}}(|\mathbb{x}|)$ *is an upper bound on the statistical distance of the two random variables*

$$\mathsf{View}_{\mathrm{IOP}}(\mathbf{P}_{\mathrm{IOP}},\mathbf{V}_{\mathrm{IOP}},\mathbb{x},\mathbb{w}) \textit{ and } \mathbf{S}_{\mathrm{IOP}}(\mathbb{x})\ .$$

IOP *has* **strong honest-verifier zero knowledge with error** ζ_{IOP} *if there exists a (pair of) polynomial-time probabilistic algorithm* $\mathbf{S}_{\mathrm{IOP}}$ *such that, for every* $(\mathbb{x},\mathbb{w}) \in R$*,* $\zeta_{\mathrm{IOP}}(|\mathbb{x}|)$ *is an upper bound on the statistical distance of the two random variables* $\mathsf{jView}_{\mathrm{IOP}}(\mathbf{P}_{\mathrm{IOP}},\mathbf{V}_{\mathrm{IOP}},\mathbb{x},\mathbb{w})$ *and*

$$\left\{(\mathbb{x},\mathbb{w},\rho_{\mathbf{P}},(\rho_1,\ldots,\rho_\mathsf{k}),(Q_1,\ldots,Q_\mathsf{k}),(\mathbf{a}_1,\ldots,\mathbf{a}_\mathsf{k}))\ \middle|\ \begin{array}{l} ((\rho_1,\ldots,\rho_\mathsf{k}),(Q_1,\ldots,Q_\mathsf{k}),(\mathbf{a}_1,\ldots,\mathbf{a}_\mathsf{k}),z_{\mathsf{SIM}}) \leftarrow \mathbf{S}_{\mathrm{IOP}}(\mathbb{x}) \\ \rho_{\mathbf{P}} \leftarrow \mathbf{S}_{\mathrm{IOP}}(\mathbb{w},z_{\mathsf{SIM}}) \end{array}\right\}\ .$$

9.2 The BCS construction

We describe the BCS construction of a SNARG, starting from two ingredients: (a) a public-coin IOP $\mathsf{IOP} = (\mathbf{P}_{\mathrm{IOP}}, \mathbf{V}_{\mathrm{IOP}})$; and (b) the Merkle commitment scheme in the ROM. Using the techniques in Section 3, we assume that the prover and verifier have access to domain-separated oracles $f_1, \ldots, f_{\mathsf{k}}, f_{\mathsf{MT}} \leftarrow \mathcal{U}(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}}, \lambda)$. For simplicity of exposition, we assume that $\mathsf{r}_1 = \cdots = \mathsf{r}_{\mathsf{k}} =: \mathsf{r}$ and let $(\mathbf{P}, \mathbf{V}) := \mathsf{BCS}[\mathsf{IOP}, \mathsf{r}]$ be the non-interactive argument constructed as follows:

- $\mathbf{P}^{f_1,\ldots,f_{\mathsf{k}},f_{\mathsf{MT}}}(\mathbb{x}, \mathbb{w})$:
  1. Compute the first IOP string $(\Pi_1, \mathsf{aux}_1) \leftarrow \mathbf{P}_{\mathrm{IOP}}(\mathbb{x}, \mathbb{w})$.
  2. Commit to it $(\mathsf{rt}_1, \mathsf{td}_1) \leftarrow \mathsf{MT.Commit}^{f_{\mathsf{MT}}}(\Pi_1)$.
  3. Sample salt $\sigma_1 \leftarrow \{0,1\}^{\mathsf{r}}$.
  4. Derive IOP randomness $\rho_1 := f_1(\mathbb{x}, \mathsf{rt}_1, \sigma_1)$.
  5. For every $i \in \{2, \ldots, \mathsf{k}\}$:
     (a) Compute the IOP string $(\Pi_i, \mathsf{aux}_i) \leftarrow \mathbf{P}_{\mathrm{IOP}}(\mathsf{aux}_{i-1}, \rho_{i-1})$.
     (b) Commit to it $(\mathsf{rt}_i, \mathsf{td}_i) \leftarrow \mathsf{MT.Commit}^{f_{\mathsf{MT}}}(\Pi_i)$.
     (c) Sample salt $\sigma_i \leftarrow \{0,1\}^{\mathsf{r}}$.
     (d) Derive IOP randomness $\rho_i := f_i(\mathbb{x}, (\mathsf{rt}_1, \ldots, \mathsf{rt}_i), (\sigma_1, \ldots, \sigma_i))$.
  6. Run $\mathbf{V}_{\mathrm{IOP}}^{\Pi_1,\ldots,\Pi_{\mathsf{k}}}(\mathbb{x}, \rho_1, \ldots, \rho_{\mathsf{k}})$ to deduce query and answer lists $(Q_1, \ldots, Q_{\mathsf{k}}), (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}})$.
  7. Compute opening proofs $\mathsf{pf}_i := \mathsf{MT.Open}(\mathsf{td}_i, Q_i)$ for $i \in [\mathsf{k}]$.
  8. Output $\pi := ((\mathsf{rt}_1, \ldots, \mathsf{rt}_{\mathsf{k}}), (\sigma_1, \ldots, \sigma_{\mathsf{k}}), (Q_1, \ldots, Q_{\mathsf{k}}), (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}), (\mathsf{pf}_1, \ldots, \mathsf{pf}_{\mathsf{k}}))$.
- $\mathbf{V}^{f_1,\ldots,f_{\mathsf{k}},f_{\mathsf{MT}}}(\mathbb{x}, \pi)$:
  1. For every $i \in [\mathsf{k}]$:
     (a) Rederive IOP randomness $\rho_i := f_i(\mathbb{x}, (\mathsf{rt}_1, \ldots, \mathsf{rt}_i), (\sigma_1, \ldots, \sigma_i))$.
     (b) Check that $\mathsf{MT.Check}^{f_{\mathsf{MT}}}(\mathsf{rt}_i, Q_i, \mathbf{a}_i, \mathsf{pf}_i) = 1$.
  2. Check $\mathbf{V}_{\mathrm{IOP}}^{[(Q_1,\mathbf{a}_1),\ldots,(Q_{\mathsf{k}},\mathbf{a}_{\mathsf{k}})]}(\mathbb{x}, \rho_1, \ldots, \rho_{\mathsf{k}}) = 1$

The argument prover and argument verifier have the following query complexities:
- $\mathsf{q}_{\mathbf{P}}(n) = \mathsf{k}(n) + \sum_{i\in[\mathsf{k}]} \mathsf{q}_{\mathsf{MT.Commit}}(\mathsf{l}_i(n)), \mathsf{q}_i(n))$,
- $\mathsf{q}_{\mathbf{V}}(n) = \mathsf{k}(n) + \sum_{i\in[\mathsf{k}]} \mathsf{q}_{\mathsf{MT.Check}}(\mathsf{l}_i(n), \mathsf{q}_i(n))$,

Throughout this section, we let $\mathsf{l} := \max_{i\in[\mathsf{k}]} \mathsf{l}_i$ and $\mathsf{q} := \max_{i\in[\mathsf{k}]} \mathsf{q}_i$.

9.3 UC-friendly completeness

We prove that the BCS construction is UC-friendly complete.

Lemma 9.5. BCS *is UC-friendly complete with error*

$$\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \ell_{\mathsf{p}} \cdot \epsilon_{\mathbf{P}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{p}} \cdot \mathsf{q}_{\mathbf{P}}(n), t_{\mathsf{p}}) \ .$$

In the above, $\epsilon_{\mathbf{P}}$ *is defined as in Claim 9.8.*

Proof. We argue that the BCS construction satisfies perfect completeness, monotone proofs and unpredictable queries in Claims 9.6 to 9.8. The statement then follows from Lemma 5.9 which shows UC-friendly completeness follows from those properties. □

Claim 9.6. BCS *has perfect completeness (Definition 5.4).*

Proof. This follows directly from perfect completeness of the Merkle commitment scheme (Lemma 7.1) and perfect completeness of the IOP (Definition 9.1). □

Claim 9.7. BCS *has monotone proofs (Definition 5.6).*

Proof. The verification algorithm of the BCS construction only queries k additional point compared to MT.Check, and those point were previously queried when deriving the IOP verifier's randomness. Since Merkle proofs are monotone (Lemma 7.2) this concludes the proof. □

Claim 9.8. BCS *has unpredictable queries (Definition 5.8) with error*

$$\epsilon_{\mathbf{P}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}) := \mathsf{k} \cdot \left(\epsilon_{\mathsf{MT}}(\lambda, \mathsf{l}(n), t_{\mathsf{q}} + \mathsf{k} \cdot \mathsf{q}_{\mathsf{MT.Commit}}(\mathsf{l}(n)), t_{\mathsf{p}}) + \frac{t_{\mathsf{p}}}{2^{\mathsf{r}}} \right) .$$

In the above, ϵ_{MT} *is defined as in Lemma 7.3.*

Proof. This follows directly from a union bound, relying on the high entropy of the Merkle commitment scheme (Lemma 7.3) and the fact that, for every $i \in [\mathsf{k}]$, the salt σ_i is uniformly distributed over $\{0,1\}^{\mathsf{r}_i}$. □

9.4 UC-friendly zero knowledge

We prove that the BCS construction is UC-friendly zero knowledge.

Lemma 9.9. *Let* IOP *be (resp. strong) UC-friendly zero knowledge with error* ζ_{IOP}. *Then* BCS[IOP, r] *is weak (resp. strong) UC-friendly zero knowledge with error*

$$\zeta_{\text{ARG}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) = \ell_{\mathsf{p}} \cdot \zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}} + \mathsf{so}^{(1)}_{\mathsf{q}}(n, \ell_{\mathsf{p}}), t_{\mathsf{p}} + \mathsf{so}^{(1)}_{\mathsf{p}}(n, \ell_{\mathsf{p}})) .$$

Above:
- $\zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}) := \frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\mathsf{r}}} + \zeta_{\mathsf{MT}}(\lambda, \mathsf{l}(n), \mathsf{q}(n), t_{\mathsf{q}}, t_{\mathsf{p}}, \mathsf{k}(n)) + \zeta_{\text{IOP}}(n)$.
- ζ_{MT} *is as in Lemma 7.12.*
- $\mathsf{so}^{(1)}_{\mathsf{q}}(n, \ell_{\mathsf{p}}) := \ell_{\mathsf{p}} \cdot \max\left(\mathsf{k}(n) + \sum_{i\in[\mathsf{k}(n)]} \mathsf{q}_{\mathsf{MT.Commit}}(\mathsf{l}_i(n)), 2\sum_{i\in[\mathsf{k}(n)]} \mathsf{q}_{\mathsf{MT.Sim}}(\mathsf{l}_i(n), \mathsf{q}_i(n))\right)$.
- $\mathsf{so}^{(1)}_{\mathsf{p}}(n, \ell_{\mathsf{p}}) := 2\ell_{\mathsf{p}} \cdot \left(\mathsf{k}(n) + \sum_{i\in[\mathsf{k}(n)]} \mathsf{p}_{\mathsf{MT.Sim}}(\mathsf{l}_i(n), \mathsf{q}_i(n))\right)$.

Construction 9.10. Let $\mathbf{S}_{\text{IOP}}$ be the simulator for IOP (Definition 9.4) and MT.Sim be the simulator for the Merkle commitment scheme (Lemma 7.12). We construct a simulator $\mathbf{S}$ for UC-friendly zero knowledge as follows.

```
$\mathbf{S}^{f_{\mathsf{MT}}}(\mathbb{x})$:
1. Compute $((\rho_1,\ldots,\rho_{\mathsf{k}}), (Q_1,\ldots,Q_{\mathsf{k}}), (\mathbf{a}_1,\ldots,\mathbf{a}_{\mathsf{k}}), z_{\mathsf{SIM}}) \leftarrow \mathbf{S}_{\text{IOP}}(\mathbb{x})$.
2. Set $(\mathsf{rt}_i, \mathsf{pf}_i, z_i) \leftarrow \mathsf{MT.Sim}^{f_{\mathsf{MT}}}(Q_i, \mathbf{a}_i)$ for $i \in [\mathsf{k}]$.
3. Sample $\sigma_1,\ldots,\sigma_{\mathsf{k}} \leftarrow \{0,1\}^{\mathsf{r}}$.
4. Set tr to program $f_i(\mathbb{x}, (\mathsf{rt}_1,\ldots,\mathsf{rt}_i), (\sigma_1,\ldots,\sigma_i)) = \rho_i$ for $i \in [\mathsf{k}]$.
5. Set $\pi := ((\mathsf{rt}_1,\ldots,\mathsf{rt}_{\mathsf{k}}), (\sigma_1,\ldots,\sigma_{\mathsf{k}}), (Q_1,\ldots,Q_{\mathsf{k}}), (\mathbf{a}_1,\ldots,\mathbf{a}_{\mathsf{k}}), (\mathsf{pf}_1,\ldots,\mathsf{pf}_{\mathsf{k}}))$.
6. Set $z_\pi := ((\rho_1,\ldots,\rho_{\mathsf{k}}), (\sigma_1,\ldots,\sigma_{\mathsf{k}}), (z_1,\ldots,z_{\mathsf{k}}), z_{\mathsf{SIM}})$
7. Return $(\pi, \mathrm{tr}, z_\pi)$.

$\mathbf{S}(\mathbb{w}, z_\pi)$:
1. Let $\rho_{\mathbf{P}} \leftarrow \mathbf{S}_{\text{IOP}}(\mathbb{w}, z_{\mathsf{SIM}})$.
2. Compute $\Pi_1,\ldots,\Pi_{\mathsf{k}}$ by running $\mathbf{P}_{\text{IOP}}$ on $(\mathbb{x}, \mathbb{w})$ with prover randomness $\rho_{\mathbf{P}}$ and verifier randomness
   $\rho_1,\ldots,\rho_{\mathsf{k}}$.
```

3. Compute $(\rho_{i,\mathsf{MT}}, \mathrm{tr}_i) \leftarrow \mathsf{MT.Sim}(\Pi_i, z_i)$ for $i \in [\mathsf{k}]$.
4. Set $\rho_{\mathsf{MT}} = (\rho_{1,\mathsf{MT}}, \sigma_1, \dots, \rho_{\mathsf{k},\mathsf{MT}}, \sigma_{\mathsf{k}})$.
5. Output $((\rho_{\mathbf{P}}, \rho_{\mathsf{MT}}), \mathrm{tr} := \circ_i \mathrm{tr}_i)$.

Proof. We argue that $\mathbf{S}$ yields the simulation error in the lemma statement. To do so, we prove the following claim.

Claim 9.11. BCS *has weak (resp. strong) simplified UC-friendly zero knowledge (Definition 5.11) against adversaries that make a single prover oracle query, with simulator* $\mathbf{S}$ *(Construction 9.10) and error:*

$$\zeta^{(1)}_{\mathsf{simple}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}) = \frac{t_{\mathsf{q}} + t_{\mathsf{p}}}{2^{\mathsf{r}}} + \zeta_{\mathsf{MT}}(\lambda, \mathsf{l}, \mathsf{q}, t_{\mathsf{q}}, t_{\mathsf{p}}, \mathsf{k}) + \zeta_{\mathrm{IOP}}(n) \enspace .$$

Then, exactly as in Lemma 8.10, applying Lemma 5.12 and Lemma 5.13 concludes the proof.
We are left to argue Claim 9.11 via a sequence of games (defined next).

- $\mathsf{sUCZeroKnowledge}_0$:
  - The "real-world" security game of Definition 5.11. (Recall, this is defined as in Definition 5.10 while limiting the adversary to a single prover query and no verifier or verifier corruption queries).
- EXPA:
  - Modify the proof oracle as follows.
    1. Sample $\rho_1, \dots, \rho_{\mathsf{k}}$ at the beginning of the proof oracle execution (instead of obtaining them by querying the random oracle).
    2. Compute the first IOP string $(\Pi_1, \mathsf{aux}_1) \leftarrow \mathbf{P}_{\mathrm{IOP}}(\mathbb{x}, \mathbb{w})$.
    3. For $i \in [\mathsf{k}]$, compute the remaining IOP strings $\Pi_i \leftarrow \mathbf{P}_{\mathrm{IOP}}(\mathsf{aux}_{i-1}, \rho_{i-1})$, by using the randomness ρ_{i-1} previously sampled.
    4. Compute roots and openings as before.
    5. Program the random oracle so that, for $i \in [\mathsf{k}]$, $f_i(\mathbb{x}, (\mathsf{rt}_1, \dots, \mathsf{rt}_i), (\sigma_1, \dots, \sigma_i)) = \rho_i$.
- EXPB:
  - Modify the proof oracle as follows:
    1. Use $\mathsf{MT.Sim}$ to obtain $((\mathsf{rt}_i, \mathsf{pf}_i))_{i \in [\mathsf{k}]}$ for the query-answer pairs induced by $\mathbf{V}_{\mathrm{IOP}}(\mathbb{x}; \rho_1, \dots, \rho_{\mathsf{k}})$instead of using the values generated by $\mathsf{MT.Commit}$ and $\mathsf{MT.Open}$.
    2. Additionally, use $\mathsf{MT.Sim}$ to obtain Merkle commitment randomness, programming the random oracle accordingly.
  - We modify the corruption oracle to return the simulated merkle randomness ρ_{MT} obtained as in the simulator of Construction 9.10 from the simulated Merkle randomness and the sampled salts.
- EXPC:
  - We modify the proof oracle to use query-answer pairs $Q_1, \dots, Q_{\mathsf{k}}, \mathbf{a}_1, \dots, \mathbf{a}_{\mathsf{k}}$ generated by $\mathbf{S}_{\mathrm{IOP}}$ rather than those induced by the verifier.
  - The corruption oracle uses the simulator $\mathbf{S}_{\mathrm{IOP}}$ to obtain the simulated prover randomness.
- $\mathsf{sUCZeroKnowledge}^{\mathbf{S}}_1$:
  - The "ideal-world" security game of Definition 5.11. (Recall, this is defined as in Definition 5.10 while limiting the adversary to a single prover query and no verifier or verifier corruption queries).

REAL is close to EXPA.

$$\Delta_{\mathcal{A}}(\mathsf{sUCZeroKnowledge}_0, \mathsf{EXPA}) \leq \frac{t_{\mathsf{q}} + t_{\mathsf{p}}}{2^{\mathsf{r}}} \enspace .$$

The only difference between the two games is that we have programmed a point in k *domain-separated* random oracles, each of the form $(\mathbb{x}, (\mathsf{rt}_1, \dots, \mathsf{rt}_i), (\sigma_1, \dots, \sigma_i))$ for σ_i a uniformly distributed string in $\{0,1\}^{\mathsf{r}}$. Thus, the probability that any of these k points is queried/programmed before they are programmed by the simulator is bounded above by $\frac{t_{\mathsf{q}}+t_{\mathsf{p}}}{2^{\mathsf{r}}}$.

EXPA is close to EXPB.

$$\Delta_{\mathcal{A}}(\mathsf{EXPA}, \mathsf{EXPB}) \le \zeta_{\mathsf{MT}}(\lambda, \mathsf{l}, \mathsf{q}, t_{\mathsf{q}}, t_{\mathsf{p}}, \mathsf{k}) \quad .$$

We construct an adversary $\mathcal{B}$ that against the UC-friendly hiding game (Lemma 7.6).

$\mathcal{B}(\mathcal{A})$:
1. Run $\mathcal{A}$, answering oracle queries as follows:
   - Answer oracle queries to f_{MT} by querying the challenger's random oracle, and those to $f_1, \dots, f_{\mathsf{k}}$ by lazily simulating that oracle.
   - On the proof-oracle query $(\mathbb{x}, \mathbb{w}) \in R$ run the following procedure:
     (a) Sample $\rho_1, \dots, \rho_{\mathsf{k}}$.
     (b) Compute $\Pi_1, \dots, \Pi_{\mathsf{k}}$ by running $\mathbf{P}_{\mathrm{IOP}}$ on $(\mathbb{x}, \mathbb{w})$ with verifier randomness $\rho_1, \dots, \rho_{\mathsf{k}}$.
     (c) Run $\mathbf{V}_{\mathrm{IOP}}^{\Pi_1, \dots, \Pi_{\mathsf{k}}}(\mathbb{x}, \rho_1, \dots, \rho_{\mathsf{k}})$ to deduce query and answer lists $(Q_1, \dots, Q_{\mathsf{k}}), (\mathbf{a}_1, \dots, \mathbf{a}_{\mathsf{k}})$.
     (d) For $i \in [\mathsf{k}]$:
        i. Call the proof oracle with input (Π_i, Q_i) to obtain $(\mathsf{rt}_i, \mathsf{pf}_i)$.
        ii. Program the random oracle f_i so that $f_i(\mathbb{x}, (\mathsf{rt}_1, \dots, \mathsf{rt}_i), (\sigma_1, \dots, \sigma_i)) = \rho_i$.
   - On a corruption oracle query, call the corruption oracle of the challenger.
2. Return the output of $\mathcal{A}$.

Note that $\mathcal{B}$ makes the same number of random oracle and programming queries as $\mathcal{A}$ and makes at most k queries to the proof oracle. If $\mathcal{B}$ is in the "real-world" of the security experiment, the view of $\mathcal{A}$ is exactly as in EXPA, otherwise it will be as in EXPB. Thus, any distinguishing advantage of $\mathcal{A}$ translates in an advantage against the UC-friendly hiding game.

EXPB is close to EXPC.

$$\Delta_{\mathcal{A}}(\mathsf{EXPB}, \mathsf{EXPC}) \le \zeta_{\mathrm{IOP}}(n) \quad .$$

In this game hop we replace the IOP query/answer sets with those sampled by the simulator. The statistical distance of the two distributions is bound by ζ_{IOP}. Thus, since the view of the adversary is otherwise identical, the statistical distance of its output in the two games can be at most ζ_{IOP}.

EXPC is IDEAL.

$$\mathsf{EXPC} \equiv \mathsf{sUCZeroKnowledge}_1^{\mathbf{S}} \quad .$$

The games are syntactically equal. □

9.5 UC-friendly knowledge soundness

We prove that the BCS construction satisfies UC-friendly knowledge soundness with respect to the simulator described in Construction 9.10.

Lemma 9.12. *Suppose that:*

- IOP *satisfies (straightline) state-restoration knowledge soundness with error* κ_{sr} *(Definition 9.3);*
- MT *has weak (resp. strong) UC-friendly extraction for* $\mathsf{MT.Sim}$ *with error* κ_{MT}.

BCS *satisfies weak (resp. strong) UC-friendly knowledge soundness (Definition 5.16) with respect to* $\mathbf{S}$ *(defined in Construction 9.10) with error*

$$\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \ell_{\mathsf{v}} \cdot \kappa^{(1)}_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}}, \ell_{\mathsf{p}}) \enspace .$$

In the above:

- $\kappa^{(1)}_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) := \kappa_{\mathsf{sr}}(n, t_{\mathsf{q}}, \lambda + \mathsf{r}) + \kappa_{\mathsf{MT}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \mathsf{l}, \mathsf{q}, \mathsf{k} \cdot (t_{\mathsf{q}} + 1), \mathsf{k})$.

Construction 9.13. Let $\mathsf{MT.Extract}$ be the Merkle extractor in Definition 7.14 and $\mathbf{E}_{\mathrm{IOP}}$ be the extractor for the IOP in Definition 9.3. We construct an extractor $\mathbf{E}$ for the BCS construction as follows.

```
$\mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace})$:
1. Parse $\pi$ as $((\mathsf{rt}_1, \ldots, \mathsf{rt}_{\mathsf{k}}), (\sigma_1, \ldots, \sigma_{\mathsf{k}}), (Q_1, \ldots, Q_{\mathsf{k}}), (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}), (\mathsf{pf}_1, \ldots, \mathsf{pf}_{\mathsf{k}}))$.
2. Denote by $\mathsf{extTrace}_{\mathsf{MT}}$ the queries made to $f_{\mathsf{MT}}$ in $\mathsf{extTrace}$.
3. For every $i \in [\mathsf{k}]$, compute $(\Pi_i, \mathsf{td}_i) := \mathsf{MT.Extract}(\mathsf{rt}_i, \mathsf{extTrace}_{\mathsf{MT}})$.
4. Compute $\mathbb{w} \leftarrow \mathbf{E}_{\mathrm{IOP}}(\mathbb{x}, \Pi_1, \ldots, \Pi_{\mathsf{k}})$.
5. Output $\mathbb{w}$.
```

Proof. We show that the extractor $\mathbf{E}$ in Construction 9.13 yields the error in the lemma statement.

Recall Lemma 5.18, which reduces UC-friendly knowledge soundness (Definition 5.16) to *single-instance* UC-friendly knowledge soundness (Definition 5.17), a simpler property in which the adversary is allowed a single verifier query. Specifically, if the non-interactive argument satisfies single-instance UC-friendly knowledge soundness with error $\kappa^{(1)}_{\mathrm{ARG}}$ then the non-interactive argument satisfies UC-friendly knowledge soundness with error $\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) \leq \ell_{\mathsf{v}} \cdot \kappa^{(1)}_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}} + \ell_{\mathsf{v}} \cdot \mathsf{q}_{\mathbf{V}}(n), t_{\mathsf{p}}, \ell_{\mathsf{p}})$. We are left to show that the BCS construction satisfies single-instance UC-friendly knowledge soundness with error at most $\kappa^{(1)}_{\mathrm{ARG}}$.

Let $\mathcal{A}$ be a $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}})$-query adversary against single-instance UC-friendly knowledge soundness. We upper bound the following probability:

$$\Pr\left[\begin{array}{l|l} (\mathbb{x}, \mathbb{w}) \notin R & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\ b = 1 & \begin{pmatrix} \mathbb{x}, \pi, \\ \mathsf{InstanceList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{pmatrix} \xleftarrow{\mathrm{tr}} \mathsf{sUCKnowledgeSoundness1}^{f}_{\mathbf{S}}(n, \mathcal{A}) \\ \wedge\ \mathrm{tr}_{\mathbf{V}} \cap \mathsf{advProg} = \emptyset & b \xleftarrow{\mathrm{tr}_{\mathbf{V}}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x}, \pi) \\ \wedge\ \mathbb{x} \notin \mathsf{InstanceList} & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array}\right]$$

Massaging the probability statement. We introduce some notation to simplify later reduction steps. Let $\mathsf{ProofList} = ((\mathbb{x}^{(s)}_j, \pi^{(s)}_j))_j$ denote the instances queried to the simulator oracle and their corresponding proof (only when the simulator oracle did not return $\perp$). Write $\mathsf{RootList}$ for the list of points programmed by the simulator oracle in $f_1, \ldots, f_{\mathsf{k}}$ when generating $\mathsf{ProofList}$. Letting $(\mathbb{x}, \pi)$ (with $\pi = ((\mathsf{rt}_1, \ldots, \mathsf{rt}_{\mathsf{k}}), (\sigma_1, \ldots, \sigma_{\mathsf{k}}), (Q_1, \ldots, Q_{\mathsf{k}}), (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}), (\mathsf{pf}_1, \ldots, \mathsf{pf}_{\mathsf{k}})))$ denote the final instance-proof pair output by the adversary, we note that if for some i we have that $(\mathbb{x}, (\mathsf{rt}_1, \ldots, \mathsf{rt}_i), (\sigma_1, \ldots, \sigma_i)) \in \mathsf{RootList}$

then $\mathbb{x} \in \mathsf{InstanceList}$. Thus, we have that the previous probability is upper-bounded by:

$$\leq \Pr\left[\begin{array}{l|l} (\mathbb{x},\mathbb{w}) \notin R & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, b = 1 & \left(\begin{array}{c} \mathbb{x}, \\ \pi := \left(\begin{array}{c} (\mathsf{rt}_1,\dots,\mathsf{rt}_\mathsf{k}), \\ (\sigma_1,\dots,\sigma_\mathsf{k}), \\ (Q_1,\dots,Q_\mathsf{k}), \\ (\mathbf{a}_1,\dots,\mathbf{a}_\mathsf{k}), \\ (\mathsf{pf}_1,\dots,\mathsf{pf}_\mathsf{k}) \end{array}\right) \\ \mathsf{RootList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{array}\right) \\ \wedge\, \mathrm{tr}_\mathbf{V} \cap \mathsf{advProg} = \emptyset & \xleftarrow{\mathrm{tr}} \mathsf{sUCKnowledgeSoundness1}^f_\mathbf{S}(n,\mathcal{A}) \\ \wedge\, \forall\, i \left(\begin{array}{c} \mathbb{x}, \\ (\mathsf{rt}_1,\dots,\mathsf{rt}_i), \\ (\sigma_1,\dots,\sigma_i) \end{array}\right) \notin \mathsf{RootList} & b \xleftarrow{\mathrm{tr}_\mathbf{V}} \mathbf{V}^{f[\mathrm{tr}]}(\mathbb{x},\pi) \\ & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x},\pi,\mathsf{extTrace} \setminus \mathsf{advProg}) \end{array}\right].$$

To simplify probability statements, we slightly abuse notation to have the sUCKnowledgeSoundness1 game return RootList instead of InstanceList.

We define an algorithm Check, which captures the winning conditions of the UC-friendly knowledge soundness game (other than the IOP verifier accepting).

```
$\mathsf{Check}^{f_\mathsf{MT}}(\mathbb{x},\pi,\mathsf{advProg},\mathsf{RootList})$:
1. Parse $\pi$ as $((\mathsf{rt}_1,\dots,\mathsf{rt}_\mathsf{k}),(\sigma_1,\dots,\sigma_\mathsf{k}),(Q_1,\dots,Q_\mathsf{k}),(\mathbf{a}_1,\dots,\mathbf{a}_\mathsf{k}),(\mathsf{pf}_1,\dots,\mathsf{pf}_\mathsf{k}))$.
2. Denote by $\mathsf{advProg}_\mathsf{FS}, \mathsf{advProg}_\mathsf{MT}$ the set of queries to $f_\mathsf{FS}, f_\mathsf{MT}$ in advProg.
3. For every $i \in [\mathsf{k}]$, compute $b_i := \mathsf{MT.Check}^{f_\mathsf{MT}}(\mathsf{rt}_i, Q_i, \mathbf{a}_i, \mathsf{pf}_i)$, denoting by $\mathrm{tr}_\mathsf{check}$ the resulting query-answer trace.
4. Check that:
   – $\mathrm{tr}_\mathsf{check} \cap \mathsf{advProg}_\mathsf{MT} = \emptyset$.
   – $\forall\, i \in [\mathsf{k}]$:
      * $b_i = 1$.
      * $(\mathbb{x},(\mathsf{rt}_1,\dots,\mathsf{rt}_i),(\sigma_1,\dots,\sigma_i)) \notin \mathsf{advProg}_\mathsf{FS}$.
      * $(\mathbb{x},(\mathsf{rt}_1,\dots,\mathsf{rt}_i),(\sigma_1,\dots,\sigma_i)) \notin \mathsf{RootList}$.
```

The BCS verifier $\mathbf{V}$, for each round $i \in [\mathsf{k}]$, queries the Fiat–Shamir oracle at $(\mathbb{x},(\mathsf{rt}_1,\dots,\mathsf{rt}_i),(\sigma_1,\dots,\sigma_i))$, and the Merkle commitment oracle, so the check $\mathrm{tr}_\mathbf{V} \cap \mathsf{advProg} = \emptyset$ can be rewritten as $(\mathbb{x},(\mathsf{rt}_1,\dots,\mathsf{rt}_i),(\sigma_1,\dots,\sigma_i)) \notin \mathsf{advProg}_\mathsf{FS}$ and $\mathrm{tr}_\mathsf{check} \cap \mathsf{advProg}_\mathsf{MT} = \emptyset$.

The previous probability statement is then equivalent to:

$$\Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ & \left(\begin{array}{c} \mathbb{x}, \\ \pi := \left(\begin{array}{c} (\mathsf{rt}_1, \ldots, \mathsf{rt}_\mathsf{k}), \\ (\sigma_1, \ldots, \sigma_\mathsf{k}), \\ (Q_1, \ldots, Q_\mathsf{k}), \\ (\mathbf{a}_1, \ldots, \mathbf{a}_\mathsf{k}), \\ (\mathsf{pf}_1, \ldots, \mathsf{pf}_\mathsf{k}) \end{array}\right), \\ \mathsf{RootList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{array}\right) \\ (\mathbb{x}, \mathbb{w}) \notin R & \\ \wedge\, b = 1 & \\ \wedge\, \mathbf{V}_{\mathrm{IOP}}^{[(Q_1,\mathbf{a}_1),\ldots,(Q_\mathsf{k},\mathbf{a}_\mathsf{k})]}(\mathbb{x}, \rho_1, \ldots, \rho_\mathsf{k}) = 1 & \xleftarrow{\mathrm{tr}_{\mathsf{MT}},\mathrm{tr}_1,\ldots,\mathrm{tr}_\mathsf{k}} \mathsf{sUCKnowledgeSoundness}^f_{\mathbf{S}}(n, \mathcal{A}) \\ & b := \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList}) \\ & \forall\, i \in [\mathsf{k}],\ \rho_i := f_i[\mathrm{tr}_i](\mathbb{x}, (\mathsf{rt}_1, \ldots, \mathsf{rt}_i), (\sigma_1, \ldots, \sigma_i)) \\ & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array}\right].$$

Reducing to (straightline) state-restoration knowledge soundness. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against the (straightline) state-restoration game of IOP. We denote in red steps that are not used in the reduction but will be used in the argument.

$\mathcal{B}(\mathcal{A})$:

1. Initialize empty $\mathrm{tr}_{\mathsf{MT}}, \mathrm{tr}_1, \ldots, \mathrm{tr}_\mathsf{k}, \mathsf{advProg}, \mathsf{extTrace}, \mathsf{ProofList}$, $\mathsf{Random}_{\mathbf{P}}$.
2. Run $\mathcal{A}$, answering queries as follows:
   - When $\mathcal{A}$ performs a query to f_{MT}, record the query and answer in $\mathrm{tr}_{\mathsf{MT}}, \mathsf{extTrace}$.
   - When $\mathcal{A}$ performs the j-th query q_j to one of f_i (queries that we count cumulatively across $f_1, \ldots, f_\mathsf{k}$):
     (a) If $\exists\ \mathsf{qid}, y$ such that $(\mathsf{qid}, q_j, y) \in \mathrm{tr}_i$ return y.
     (b) Parse q_j as $(\mathbb{x}_j, (\mathsf{rt}_{j,1}, \ldots, \mathsf{rt}_{j,i}), (\sigma_{j,1}, \ldots, \sigma_{j,i}))$. (If this fails, answer the query with a lazily sampled random oracle.)
     (c) Denote by $\mathsf{extTrace}_j$ the queries in $\mathsf{extTrace}$ added since the last execution of $\mathsf{MT.MultiExtract}$.
     (d) Compute $(\Pi_{j,k}, \mathsf{td}_{j,k}) := \mathsf{MT.MultiExtract}(\mathsf{rt}_{j,k}, \mathsf{extTrace}_j \setminus \mathsf{advProg})$ for every $k \in [i]$.
     (e) Submit $(\mathbb{x}_j, (\Pi_{j,1}, \ldots, \Pi_{j,i}), ((\mathsf{rt}_{j,1}, \sigma_{j,1}), \ldots, (\mathsf{rt}_{j,i}, \sigma_{j,i})))$ to the state-restoration game, obtaining randomness ρ_j.
     (f) Append $(\mathsf{query}, q_j, \rho_j)$ to tr_i.
     (g) Return ρ_j.
   - When $\mathcal{A}$ makes a programming query $\mathsf{trace}_{\mathsf{prog}}$ to $f_1, \ldots, f_\mathsf{k}$ or f_{MT}:
     (a) Partition $\mathsf{trace}_{\mathsf{prog}}$ into $\mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$ and $\mathsf{trace}^{i}_{\mathsf{prog}}$ for every $i \in [\mathsf{k}]$ depending on which oracle was queried.
     (b) For every $i \in [\mathsf{k}]$, if there exist $(x, y) \in \mathsf{trace}^{i}_{\mathsf{prog}}$ and $(\mathsf{qid}_j, x_j, y_j) \in \mathrm{tr}_i$ with $x_j = x$, return 0.
     (c) If there exist $(x, y) \in \mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$ and $(\mathsf{qid}_j, x_j, y_j) \in \mathrm{tr}_{\mathsf{MT}}$ with $x_j = x$, return 0.
     (d) Else, for every $i \in [\mathsf{k}]$ append $((\mathsf{prog}, x, y))_{(x,y) \in \mathsf{trace}^{i}_{\mathsf{prog}}}$ to tr_i, $((\mathsf{prog}, x, y))_{(x,y) \in \mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}}$ to $\mathrm{tr}_{\mathsf{MT}}$ (and add both to $\mathsf{advProg}$) and return 1.
   - When $\mathcal{A}$ requests the j-th proof for $(\mathbb{x}^{(s)}_j, \mathbb{w}^{(s)}_j) \in R$:
     (a) Compute $(\pi^{(s)}_j, \mathrm{tr}, z_\pi) \xleftarrow{\mathrm{tr}_{\mathbf{S}}} \mathbf{S}^{f_{\mathsf{MT}}}(\mathbb{x}_j{}^{(s)})$.
     (b) Compute $(\rho_{\mathbf{P}}, \mathrm{tr}') \xleftarrow{\mathrm{tr}'_{\mathbf{S}}} \mathbf{S}(\mathbb{w}, z_\pi)$.

   (c) Program the oracles $f_1, \ldots, f_{\mathsf{k}}, f_{\mathsf{MT}}$ according to $\mathrm{tr}, \mathrm{tr}'$, outputting $\bot$ if the programming fails.
   (d) Set $\mathsf{extTrace} := \mathsf{extTrace} \circ \mathrm{tr}_{\mathbf{S}} \circ \mathrm{tr}'_{\mathbf{S}}$.
   (e) Append $(\mathbb{x}_j^{(s)}, \pi_j^{(s)})$ to $\mathsf{ProofList}$.
   (f) Append $\rho_{\mathbf{P}}$ to $\mathsf{Random}_{\mathbf{P}}$.
 – When $\mathcal{A}$ asks a corruption query, return $\mathsf{Random}_{\mathbf{P}}$ (and stop answering further simulator or corruption queries).
3. The adversary finally outputs a pair $(\mathbb{x}, \pi)$.
4. Parse $\pi = ((\mathsf{rt}_1, \ldots, \mathsf{rt}_{\mathsf{k}}), (\sigma_1, \ldots, \sigma_{\mathsf{k}}), (Q_1, \ldots, Q_{\mathsf{k}}), (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}), (\mathsf{pf}_1, \ldots, \mathsf{pf}_{\mathsf{k}}))$.
5. Denote by $\mathsf{extTrace}'$ the queries in $\mathsf{extTrace}$ added since the last execution of $\mathsf{MT.MultiExtract}$.
6. For every $i \in [\mathsf{k}]$, compute $(\Pi_i, \mathsf{td}_i) := \mathsf{MT.MultiExtract}(\mathsf{rt}_i, \mathsf{extTrace} \setminus \mathsf{advProg})$.
7. Derive $\mathsf{RootList}$ from $\mathsf{ProofList}$.
8. Set $b = \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList})$.
9. Return $(\mathbb{x}, (\Pi_1, \ldots, \Pi_{\mathsf{k}}), ((\mathsf{rt}_1, \sigma_1), \ldots (\mathsf{rt}_{\mathsf{k}}, \sigma_{\mathsf{k}})))$.

We argue that the view of $\mathcal{A}$ when run within $\mathcal{B}$ during the $\mathsf{Game}_{\mathsf{sr}}$ game is identical to the view of $\mathcal{A}$ in the single-instance UC-friendly knowledge soundness game $\mathsf{sUCKnowledgeSoundness1}^f_{\mathbf{S}}(n, \mathcal{A})$. We consider each type of query that $\mathcal{A}$ makes:

- Queries to f_{MT}. The reduction adversary $\mathcal{B}$ answer queries to f_{MT} faithfully, so the distribution of answers is the same as in the (single-instance) UC-friendly knowledge soundness game.
- Programming queries to $f_1, \ldots, f_{\mathsf{k}}, f_{\mathsf{MT}}$. Programming is perfectly simulated by the $\mathcal{B}$. In the case of the f_{MT}, $\mathcal{B}$ implements the programming logic. For every $i \in [\mathsf{k}]$, in the case of f_i, $\mathcal{B}$ stores the move-answer pairs in the state-restoration performed thus far, which also allows it to faithfully answer programming queries.
- For every $i \in [\mathsf{k}]$, queries to f_i. Queries to f_i that are not successfully parsed are answered by a (lazily sampled) random oracle, and so those queries have identically distributed answers in both games. Queries that are successfully parsed (and have not been previously queried or programmed) are forwarded to the state-restoration game, which returns uniformly distributed randomness. Duplicate queries are also handled consistently by $\mathcal{B}$. Thus, all these queries are answered consistently to the UC-friendly knowledge soundness game.
- Simulator queries and prover corruption queries. These are answered identically in both games, as the reduction adversary faithfully simulates the oracle.

Define the event E_{extr} to hold if: (i) $b = 1$ and for some $i \in [\mathsf{k}]$ we have that $\Pi_i[Q_i] \neq \mathbf{a}_i$; or (ii) $\exists\, j, i \in [\mathsf{k}]$ such that $\mathsf{rt}_i = \mathsf{rt}_{j,i}$ and $\Pi_i \neq \Pi_{j,i}$.

We define the event E_{extr} as follows:

$$\left[\begin{array}{l|l} \left(\begin{array}{l} b = 1 \wedge \\ \exists\, i \in [\mathsf{k}] \text{ s.t. } \Pi_i[Q_i] \neq \mathbf{a}_i \end{array} \right) & f = (f_1, \ldots, f_{\mathsf{k}}, f_{\mathsf{MT}}) \leftarrow \mathcal{U}(\lambda) \\ & (\mathbb{x}, \pi, \mathsf{RootList}, \mathrm{tr}_{\mathbf{S}}) \leftarrow \mathsf{sUCKnowledgeSoundness}^f_{\mathbf{S}}(n, \mathcal{A}) \\ \text{or} & (\mathbb{x}, (\Pi_1, \ldots, \Pi_{\mathsf{k}}), ((\mathsf{rt}_1, \sigma_1), \ldots, (\mathsf{rt}_{\mathsf{k}}, \sigma_{\mathsf{k}})), \rho_1, \ldots, \rho_{\mathsf{k}}) \\ \exists\, j, i \text{ s.t. } \mathsf{rt}_i = \mathsf{rt}_{j,i} \wedge \Pi_i \neq \Pi_{j,i} & \xleftarrow{b, (\mathsf{rt}_{j,i}, \Pi_{j,i})_{j,i}} \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}^{f_{\mathsf{MT}}}(\mathcal{A}), f_1, \ldots, f_{\mathsf{k}}, \lambda + \mathsf{r}) \end{array} \right] . \quad (3)$$

We will soon argue that overloaded variables in the above experiment are identical so we do not disambiguate them in the notation.

We argue the following distributions are identical:

$$\left\{ \begin{array}{l} \underline{(\mathbb{x}, (\Pi_1, \dots, \Pi_{\mathsf{k}}), (\rho_1, \dots, \rho_{\mathsf{k}}), b_f, \mathrm{tr}_{\mathsf{MT}})} \\ \text{conditioned on:} \\ \forall\, i \begin{pmatrix} \mathbb{x}, \\ (\mathsf{rt}_1, \dots, \mathsf{rt}_i), \\ (\sigma_1, \dots, \sigma_i) \end{pmatrix} \notin \mathsf{RootList} \cup \mathsf{advProg} \\ \wedge \neg E_{\mathsf{extr}} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \begin{pmatrix} \pi := \begin{pmatrix} \mathbb{x}, \\ (\mathsf{rt}_1, \dots, \mathsf{rt}_{\mathsf{k}}), \\ (\sigma_1, \dots, \sigma_{\mathsf{k}}), \\ (Q_1, \dots, Q_{\mathsf{k}}), \\ (\mathbf{a}_1, \dots, \mathbf{a}_{\mathsf{k}}), \\ (\mathsf{pf}_1, \dots, \mathsf{pf}_{\mathsf{k}}) \end{pmatrix}, \\ \mathsf{RootList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{pmatrix} \\ \xleftarrow{\mathrm{tr}_{\mathsf{MT}}, \mathrm{tr}_1, \dots, \mathrm{tr}_{\mathsf{k}}} \mathsf{sUCKnowledgeSoundness}^f_{\mathbf{S}}(n, \mathcal{A}) \\ b := \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList}) \\ \forall\, i \in [\mathsf{k}],\ \rho_i := f_i[\mathrm{tr}_i](\mathbb{x}, (\mathsf{rt}_1, \dots, \mathsf{rt}_i), (\sigma_1, \dots, \sigma_i)) \\ b_f := b \wedge \mathbf{V}_{\mathrm{IOP}}^{[(Q_1, \mathbf{a}_1), \dots, (Q_{\mathsf{k}}, \mathbf{a}_{\mathsf{k}})]}(\mathbb{x}, \rho_1, \dots, \rho_{\mathsf{k}}) \\ \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array} \right\}$$

and (using the $\leftarrow$ notation to bring into scope internal variable of $\mathcal{B}$)

$$\left\{ \begin{array}{l} \underline{(\mathbb{x}, (\Pi_1, \dots, \Pi_{\mathsf{k}}), (\rho_1, \dots, \rho_{\mathsf{k}}), b_f, \mathrm{tr}_{\mathsf{MT}})} \\ \text{conditioned on:} \\ \forall\, i \begin{pmatrix} \mathbb{x}, \\ (\mathsf{rt}_1, \dots, \mathsf{rt}_i), \\ (\sigma_1, \dots, \sigma_i) \end{pmatrix} \notin \mathsf{RootList} \cap \mathsf{advProg} \\ \wedge\ \neg E_{\mathsf{extr}} \end{array} \middle| \begin{array}{l} f_1, \dots, f_{\mathsf{k}} \leftarrow \mathcal{U}(\mathsf{r}_1, \dots, \mathsf{r}_{\mathsf{k}}) \\ (\mathbb{x}, (\Pi_1, \dots, \Pi_{\mathsf{k}}), ((\mathsf{rt}_1, \sigma_1), \dots, (\mathsf{rt}_{\mathsf{k}}, \sigma_{\mathsf{k}})), \rho_1, \dots, \rho_{\mathsf{k}}) \\ \xleftarrow{b, \mathrm{tr}_{\mathsf{MT}}, \mathsf{advProg}, \mathsf{RootList}} \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}(\mathcal{A}), f_1, \dots, f_{\mathsf{k}}, \lambda + \mathsf{r}) \\ b_f := b \wedge \mathbf{V}_{\mathrm{IOP}}^{\Pi_1, \dots, \Pi_{\mathsf{k}}}(\mathbb{x}; \rho_1, \dots, \rho_{\mathsf{k}}) \end{array} \right\}$$

We discuss each random variable in turn:

- $\mathbb{x}, \mathrm{tr}_{\mathsf{MT}}$. Both the instance $\mathbb{x}$ and the Merkle trace $\mathrm{tr}_{\mathsf{MT}}$ are determined by the output and trace of the adversary $\mathcal{A}$. We have argued that the view of $\mathcal{A}$ is identical in both experiments, so the distribution of these random variables is also identical.
- $\Pi_1, \dots, \Pi_{\mathsf{k}}$. In both experiments, $\Pi_i := \mathsf{MT.Extract}(\mathsf{rt}_i, \mathsf{extTrace}_{\mathsf{MT}})$. As in the previous point, rt_i, $\mathsf{advTrace}_{\mathsf{MT}}$ are directly determined by the adversary $\mathcal{A}$, and thus are identically distributed in both games. $\mathsf{simTrace}_{\mathsf{MT}}$ is also identically distributed, since, as argued before, $\mathcal{B}$ faithfully simulates the simulator oracle, and $\mathcal{A}$ makes identically distributed queries in both games. Thus, Π_i has the same distribution in both games.
- $\rho_1, \dots, \rho_{\mathsf{k}}$. Let $i \in [\mathsf{k}]$. First, note that the distribution of the query-point $p = (\mathbb{x}, (\mathsf{rt}_1, \dots, \mathsf{rt}_i), (\sigma_1, \dots, \sigma_i))$ is identical in both experiments. Note that, because the simulator only programs the Fiat–Shamir oracle on points in $\mathsf{RootList}$, the Fiat–Shamir oracle is only programmed on points in $\mathsf{advProg}$ or in $\mathsf{RootList}$ So, since $p \notin \mathsf{RootList} \cup \mathsf{advProg}$, in the first experiment $\rho_i = f_i[\mathrm{tr}_i](p) = f_i(p)$. We distinguish two cases:
  - p was previously queried to f_i by $\mathcal{A}$. Let $j \in [t_{\mathsf{q}}]$ denote the index of the first such query. In the first experiment, ρ_i is a uniformly sampled string, consistent to the j-th query to f_i. In the second experiment, ρ_i is obtained by querying $(\mathbb{x}, (\Pi_1, \dots, \Pi_i), (\mathsf{rt}_1, \dots, \mathsf{rt}_i), (\sigma_1, \dots, \sigma_i))$ to f_i. Letting $\Pi_{j,1}, \dots, \Pi_{j,i}$ denote the IOP strings extracted in the j-th query, since $\neg E_{\mathsf{extr}}$ holds and for every $k \in [i]$, $\mathsf{rt}_{j,k} = \mathsf{rt}_k$, $\Pi_{j,k} = \Pi_k$. Thus, both queries map to the same state-restoration move $(\mathbb{x}, (\Pi_1, \dots, \Pi_i), ((\mathsf{rt}_1, \dots, \mathsf{rt}_i), (\sigma_1, \dots, \sigma_i)))$. ρ_i is also uniformly distributed as desired.

  - p was not previously queried to f_i by $\mathcal{A}$. In the first experiment ρ_i is a string sampled uniformly at random. In the second experiment, ρ_i is obtained by querying f_i at $(\mathbb{x}, (\Pi_1, \ldots, \Pi_i), ((\mathsf{rt}_1, \sigma_1), \ldots, (\mathsf{rt}_i, \sigma_i)))$ which is also a fresh new state-restoration move. Thus, ρ_i is also a uniformly random string.

- b_f. In both experiments, b is computed by running Check with inputs that are identically distributed, so the distribution of b is identical. If $b = 0$, $b_f = 0$ in both experiments. Consider then the case when $b = 1$. In the bottom experiment, since $b = 1$ and $\neg E_{\mathsf{extr}}$ holds, it must be that, for every $i \in [\mathsf{k}]$, $\Pi_i[Q_i] = \mathbf{a}_i$ and so $\mathbf{V}_{\mathrm{IOP}}^{\Pi_1,\ldots,\Pi_{\mathsf{k}}}(\mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}) = \mathbf{V}_{\mathrm{IOP}}^{[(Q_1,\mathbf{a}_1),\ldots,(Q_{\mathsf{k}},\mathbf{a}_{\mathsf{k}})]}(\mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}})$. Since $\mathbb{x}$, $(\Pi_1, \ldots, \Pi_{\mathsf{k}})$, $(\rho_1, \ldots, \rho_{\mathsf{k}})$ are identically distributed in both experiment, b_f has the same distribution in both experiments.

Since the two distributions are identical, we get that

$$\Pr\left[\begin{array}{l|l} (\mathbb{x}, \mathbb{w}) \notin R & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, b = 1 & \left(\begin{array}{c} \mathbb{x}, \\ \pi := \left(\begin{array}{c} (\mathsf{rt}_1, \ldots, \mathsf{rt}_{\mathsf{k}}), \\ (\sigma_1, \ldots, \sigma_{\mathsf{k}}), \\ (Q_1, \ldots, Q_{\mathsf{k}}), \\ (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}), \\ (\mathsf{pf}_1, \ldots, \mathsf{pf}_{\mathsf{k}}) \end{array}\right), \\ \mathsf{RootList}, \\ \mathsf{extTrace}, \\ \mathsf{advProg} \end{array}\right) \xleftarrow{\mathrm{tr}_{\mathsf{MT}}, \mathrm{tr}_1, \ldots, \mathrm{tr}_{\mathsf{k}}} \mathsf{sUCKnowledgeSoundness}^f_{\mathbf{S}}(n, \mathcal{A}) \\ \wedge\, \mathbf{V}_{\mathrm{IOP}}^{[(Q_1,\mathbf{a}_1),\ldots,(Q_{\mathsf{k}},\mathbf{a}_{\mathsf{k}})]}(\mathbb{x}, \rho_1, \ldots, \rho_{\mathsf{k}}) = 1 & b := \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList}) \\ & \forall\, i \in [\mathsf{k}],\ \rho_i := f_i[\mathrm{tr}_i](\mathbb{x}, (\mathsf{rt}_1, \ldots, \mathsf{rt}_i), (\sigma_1, \ldots, \sigma_i)) \\ & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{x}, \pi, \mathsf{extTrace} \setminus \mathsf{advProg}) \end{array}\right]$$

$$\le \Pr\left[\begin{array}{l|l} (\mathbb{x}, \mathbb{w}) \notin R & (f_1, \ldots, f_{\mathsf{k}}) \leftarrow \mathcal{U}(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}}) \\ \wedge\, b = 1 & (\mathbb{x}, (\Pi_1, \ldots, \Pi_{\mathsf{k}}), ((\mathsf{rt}_1, \sigma_1), \ldots, (\mathsf{rt}_{\mathsf{k}}, \sigma_{\mathsf{k}})), \rho_1, \ldots, \rho_{\mathsf{k}}) \\ \wedge\, \mathbf{V}_{\mathrm{IOP}}^{\Pi_1,\ldots,\Pi_{\mathsf{k}}}(\mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}) = 1 & \xleftarrow{b, \mathsf{RootList}} \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}(\mathcal{A}), f_1, \ldots, f_{\mathsf{k}}, \lambda + \mathsf{r}) \\ \wedge\, \neg E_{\mathsf{extr}} & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{IOP}}(\mathbb{x}, \Pi_1, \ldots, \Pi_{\mathsf{k}}) \end{array}\right] + \Pr[E_{\mathsf{extr}}]$$

$$\le \Pr\left[\begin{array}{l|l} & (f_1, \ldots, f_{\mathsf{k}}) \leftarrow \mathcal{U}(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}}) \\ (\mathbb{x}, \mathbb{w}) \notin R & (\mathbb{x}, (\Pi_1, \ldots, \Pi_{\mathsf{k}}), ((\mathsf{rt}_1, \sigma_1), \ldots, (\mathsf{rt}_{\mathsf{k}}, \sigma_{\mathsf{k}})), \rho_1, \ldots, \rho_{\mathsf{k}}) \\ \wedge\, \mathbf{V}_{\mathrm{IOP}}^{\Pi_1,\ldots,\Pi_{\mathsf{k}}}(\mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}) = 1 & \xleftarrow{b, \mathsf{RootList}} \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}(\mathcal{A}), f_1, \ldots, f_{\mathsf{k}}, \lambda + \mathsf{r}) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{IOP}}(\mathbb{x}, \Pi_1, \ldots, \Pi_{\mathsf{k}}) \end{array}\right] + \Pr[E_{\mathsf{extr}}] \quad .$$

The first term is bounded above by the IOP state restoration error, so it is bounded above by $\kappa_{\mathsf{sr}}(n, t_{\mathsf{q}}, \lambda + \mathsf{r})$.

Bounding Merkle extraction error. We are left to upper bound $\Pr[E_{\mathsf{extr}}]$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}_{\mathsf{MT}}$ against the UC-friendly extraction property of the Merkle commitment scheme (Definition 7.14) as follows. We highlight differences from the adversary $\mathcal{B}$ against the IOP (straightline) state-restoration game in red.

$\mathcal{B}_{\mathsf{MT}}^{f_1,\ldots,f_{\mathsf{k}}}(\mathcal{A})$:

1. Initialize empty $\mathrm{tr}_1, \ldots, \mathrm{tr}_{\mathsf{k}}$, advProg, ProofList, aux.
2. Run $\mathcal{A}$, answering queries as follows:
   - When $\mathcal{A}$ performs a query to f_{MT}, forward the query to the random oracle of the game.
   - When $\mathcal{A}$ performs the j-th query q_j to one of f_i (queries that we count cumulatively across $f_1, \ldots, f_{\mathsf{k}}$):

    (a) If $\exists$ qid, y such that $(\mathsf{qid}, q_j, y) \in \mathrm{tr}_i$ return y.
    (b) Parse q_j as $(\mathbb{x}_j, (\mathsf{rt}_{j,1}, \ldots, \mathsf{rt}_{j,i}), (\sigma_{j,1}, \ldots, \sigma_{j,i}))$. (If this fails, answer the query with a lazily sampled random oracle.)
    (c) Submit $\mathsf{rt}_{j,1}, \ldots, \mathsf{rt}_{j,i}$ as root queries to the game, obtaining $\Pi_{j,1}, \ldots, \Pi_{j,i}$.
    (d) Set $\rho_j := f_i(\mathbb{x}_j, (\Pi_{j,1}, \ldots, \Pi_{j,i}), ((\mathsf{rt}_{j,1}, \sigma_{j,1}), \ldots, (\mathsf{rt}_{j,i}, \sigma_{j,i})))$.
    (e) Append $(\mathsf{query}, q_j, \rho_j)$ to tr_i.
    (f) Return ρ_j.
  – When $\mathcal{A}$ makes a programming query $\mathsf{trace}_{\mathsf{prog}}$ to $f_1, \ldots, f_{\mathsf{k}}$ or f_{MT}:
    (a) Partition $\mathsf{trace}_{\mathsf{prog}}$ into $\mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$ and $\mathsf{trace}^{i}_{\mathsf{prog}}$ for every $i \in [\mathsf{k}]$ depending on which oracle was queried.
    (b) For every $i \in [\mathsf{k}]$, if there exist $(x, y) \in \mathsf{trace}^{i}_{\mathsf{prog}}$ and $(\mathsf{qid}_j, x_j, y_j) \in \mathrm{tr}_i$ with $x_j = x$, return 0.
    (c) Make a programming query to the programming oracle of the game with $\mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$, if this returns 0, return 0.
    (d) Else, for every $i \in [\mathsf{k}]$ append $((\mathsf{prog}, x, y))_{(x,y)\in \mathsf{trace}^{i}_{\mathsf{prog}}}$ to tr_i.
    (e) Append $\mathsf{trace}^{1}_{\mathsf{prog}}, \ldots, \mathsf{trace}^{\mathsf{k}}_{\mathsf{prog}}, \mathsf{trace}^{\mathsf{MT}}_{\mathsf{prog}}$ to $\mathsf{advProg}$, and return 1.
  – When $\mathcal{A}$ requests the j-th proof for $(\mathbb{x}^{(s)}_j, \mathbb{w}^{(s)}_j) \in R$:
    (a) Sample a IOP view $((\rho_1, \ldots, \rho_{\mathsf{k}}), (Q_1, \ldots, Q_{\mathsf{k}}), (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}), z_{\mathsf{SIM}}) \leftarrow \mathbf{S}_{\mathrm{IOP}}(\mathbb{x})$.
    (b) For every $i \in [\mathsf{k}]$, set Π_i so that $\Pi_i[q] = \mathbf{a}[q]$ for every $q \in Q_i$ and $\Pi_i[q] = \perp$ otherwise.
    (c) Reconstruct IOP prover randomness $\rho_{\mathbf{P}} \leftarrow \mathbf{S}_{\mathrm{IOP}}(\mathbb{w}, z_{\mathsf{SIM}})$.
    (d) Rederive IOP strings $\Pi_1, \ldots, \Pi_{\mathsf{k}}$ by running $\mathbf{P}_{\mathrm{IOP}}$ with prover randomness $\rho_{\mathbf{P}}$ and verifier randomness $\rho_1, \ldots, \rho_{\mathsf{k}}$.
    (e) For every $i \in [\mathsf{k}]$, submit (Π_i, Q_i) to the simulator oracle of the game, obtaining $(\mathsf{rt}_i, \mathsf{pf}_i)$. (If instead the oracle returns $\perp$, return $\perp$.)
    (f) Sample salt strings $\sigma_1, \ldots, \sigma_{\mathsf{k}} \leftarrow \{0, 1\}^{\mathsf{r}}$.
    (g) If for any $i \in [\mathsf{k}]$, $(\mathbb{x}, (\mathsf{rt}_1, \ldots, \mathsf{rt}_i), (\sigma_1, \ldots, \sigma_i)) \in \mathrm{tr}_i$, return $\perp$.
    (h) For every $i \in [\mathsf{k}]$, append $(\mathbb{x}, (\mathsf{rt}_1, \ldots, \mathsf{rt}_i), (\sigma_1, \ldots, \sigma_i))$ to tr_i.
    (i) Set $\pi := ((\mathsf{rt}_1, \ldots, \mathsf{rt}_{\mathsf{k}}), (\sigma_1, \ldots, \sigma_{\mathsf{k}}), (Q_1, \ldots, Q_{\mathsf{k}}), (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}), (\mathsf{pf}_1, \ldots, \mathsf{pf}_{\mathsf{k}}))$.
    (j) Append $(\mathbb{x}, \pi)$ to $\mathsf{ProofList}$.
    (k) Append $\rho_{\mathbf{P}}$ to aux.
    (l) Return π.
  – When $\mathcal{A}$ asks a corruption query, query the corruption oracle of the game, which return a list of randomnesses, concatenate them with the IOP prover randomness in aux, return the concatenated list $\mathsf{Random}_{\mathbf{P}}$ (and stop answering further simulator or corruption queries).
3. The adversary finally outputs a pair $(\mathbb{x}, \pi)$.
4. Parse $\pi = ((\mathsf{rt}_1, \ldots, \mathsf{rt}_{\mathsf{k}}), (\sigma_1, \ldots, \sigma_{\mathsf{k}}), (Q_1, \ldots, Q_{\mathsf{k}}), (\mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}), (\mathsf{pf}_1, \ldots, \mathsf{pf}_{\mathsf{k}}))$.
5. Derive $\mathsf{RootList}$ from $\mathsf{ProofList}$.
6. Set $b = \mathsf{Check}^{f_{\mathsf{MT}}[\mathrm{tr}_{\mathsf{MT}}]}(\mathbb{x}, \pi, \mathsf{advProg}, \mathsf{RootList})$.
7. Let ℓ denote the number of root queries performed so far.
8. Make root queries $\mathsf{rt}_1, \ldots, \mathsf{rt}_{\mathsf{k}}$.
9. Return $((\ell + 1, Q_1, \mathbf{a}_1, \mathsf{pf}_1), \ldots, (\ell + \mathsf{k}, Q_{\mathsf{k}}, \mathbf{a}_{\mathsf{k}}, \mathsf{pf}_{\mathsf{k}}))$.

We use $\mathcal{B}_{\mathsf{MT}}$ to bound the probability of E_{extr}. We define the event E'_{extr} to be the following one:

$$\left[\begin{array}{l|l} \left(\begin{array}{l} b = 1 \wedge \\ \exists i \in [\mathsf{k}] \text{ s.t. } \Pi_i[Q_i] \neq \mathbf{a}_i \end{array}\right) & f = (f_1, \ldots, f_{\mathsf{k}}, f_{\mathsf{MT}}) \leftarrow \mathcal{U}(\lambda) \\ \text{or} & (\mathbb{x}, (\Pi_1, \ldots, \Pi_{\mathsf{k}}), ((\mathsf{rt}_1, \sigma_1), \ldots, (\mathsf{rt}_{\mathsf{k}}, \sigma_{\mathsf{k}})), \rho_1, \ldots, \rho_{\mathsf{k}}) \\ \exists\, j, i \text{ s.t. } \mathsf{rt}_i = \mathsf{rt}_{j,i} \wedge \Pi_i \neq \Pi_{j,i} & \leftarrow \mathsf{Game}_{\mathsf{sr}}(\mathcal{B}(\mathcal{A}), f_1, \ldots, f_{\mathsf{k}}, \lambda + \mathsf{r}) \\ & \mathsf{advWin} \xleftarrow{b, (Q_i)_i, (\mathbf{a}_i)_i, (\Pi_i)_i, (\mathsf{rt}_{j,i}, \Pi_{j,i})_{j,i}} \\ & s\mathsf{UCMerkleExtraction}^{f_{\mathsf{MT}}}(\mathcal{B}^{f_1, \ldots, f_{\mathsf{k}}}\mathsf{MT}(\mathcal{A}), \mathsf{l}, \mathsf{q}, \mathsf{k} \cdot (t_{\mathsf{q}} + 1), \mathsf{k}) \end{array}\right] . \tag{4}$$

As before, we will soon argue that overloaded variables are identical, so we do not disambiguate.

First note that the distribution of the answer of queries of $\mathcal{A}$ in $\mathcal{B}_{\mathsf{MT}}$ is the same as when run within $\mathcal{B}$ in the state-restoration game.

- Queries f_{MT} are forwarded to the random oracle of the game.
- For every $i \in [\mathsf{k}]$, queries to f_i are answered by extracting IOP strings (by making a root query, which runs $\mathsf{MT.MultiExtract}$) and then querying the oracle f_i.
- Programming queries are answered by first checking if the Fiat–Shamir programming requests have been previously determined, if not, the Merkle programming requests are forwarded to the programming oracle of the game, and only then the Fiat–Shamir oracle is programmed. This ensure they are handled as in the state-restoration game (maintaining atomicity).
- Simulator queries are identically distributed to those in a execution of $\mathcal{B}$ within the state-restoration game. In the weak UC-friendly extraction game, this is because the reduction faithfully simulates the simulator oracle (replacing the execution of $\mathsf{MT.Sim}$ with a query to the simulator oracle of the game). In the strong UC-friendly extraction game, the order in which the IOP strong simulator and the Merkle simulator are run is changed, but this is immaterial since the IOP strong simulator does not query/program the random oracle and Merkle strong simulator only program f_{MT}.

Write $\mathsf{rt}_1, \ldots, \mathsf{rt}_\ell, \mathsf{rt}_{\ell+1} = \mathsf{rt}_1, \ldots \mathsf{rt}_{\ell+\mathsf{k}} = \mathsf{rt}_{\mathsf{k}}$ for the list of roots queried by $\mathcal{B}_{\mathsf{MT}}$, and $\Pi_1, \ldots, \Pi_{\ell+\mathsf{k}}$ for the corresponding extracted strings (note that $\ell \in [\mathsf{k} \cdot t_{\mathsf{q}}]$). We argue that, in the experiments of Equations (3) and (4), the distribution of $b, Q_1, \ldots, Q_{\mathsf{k}}, \mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}, \mathsf{rt}_1, \ldots, \mathsf{rt}_{\ell+\mathsf{k}}, \Pi_1, \ldots, \Pi_{\ell+\mathsf{k}}$ are identically distributed. Since $Q_1, \ldots, Q_{\mathsf{k}}, \mathbf{a}_1, \ldots, \mathbf{a}_{\mathsf{k}}, \mathsf{rt}_1, \ldots, \mathsf{rt}_{\ell+\mathsf{k}}$ are directly determined by the output of $\mathcal{A}$, they are identical in both experiments. Further, in both experiments, the traces $\mathsf{advTrace}, \mathsf{simTrace}$ are identical (and also are their restrictions $\mathsf{advTrace}_j, \mathsf{simTrace}_j$). Thus, since $\Pi_{j,i} := \mathsf{MT.MultiExtract}(\mathsf{rt}_{j,i}, \mathsf{advTrace}_j, \mathsf{simTrace}_j)$, the IOP strings are also identically distributed. Finally, b in both experiments is a deterministic function of the variables mentioned above, and thus also has the correct distribution in both experiments.

Since the conditions for which $E_{\mathsf{extr}}, E'_{\mathsf{extr}}$ hold are identical, it must be that $\Pr[E_{\mathsf{extr}}] = \Pr[E'_{\mathsf{extr}}]$.

The proof concludes by noticing that E'_{extr} is a relaxation of the conditions required for the advWin flag to be set in the UC-friendly extraction game. Indeed E_{extr} holds if: (i) $b = 1$ and for some $i \in [\mathsf{k}]$ $\Pi_i[Q_i] \neq \mathbf{a}_i$; or (ii) $\exists\, j, i$ such that $\mathsf{rt}_i = \mathsf{rt}_{j,i}$ and $\Pi_i \neq \Pi_{j,i}$. If the first item holds, then, for some $i \in [\mathsf{k}]$, $b_i = 1$, $\mathsf{advProg} \cap \mathrm{tr}_{\mathsf{check}} = \emptyset$ and $\Pi_i[Q_1, \ldots, Q_{\mathsf{k}i}] \neq \mathbf{a}_i$, and thus Item 6b in Definition 7.14 holds, and thus $\mathsf{advWin} = 1$. If the second item holds, then $\mathsf{advWin} = 1$, because if $\exists\, j, i$ such that $\mathsf{rt}_i = \mathsf{rt}_{j,i}$ and $\Pi_i \neq \Pi_{j,i}$ it must hold that $\exists\, f, g$ such that $\mathsf{rt}_f = \mathsf{rt}_g$ and $\Pi_f \neq \Pi_g$ (namely, set $f = \ell + i$ and $g = \mathsf{k} \cdot j + i$). We conclude that

$$\begin{aligned} \Pr[E_{\mathsf{extr}}] &= \Pr[E'_{\mathsf{extr}}] \\ &\leq \Pr\left[\mathsf{advWin} = 1 \,\middle|\, \begin{array}{l} f = (f_1, \ldots, f_{\mathsf{k}}, f_{\mathsf{MT}}) \leftarrow \mathcal{U}(\lambda) \\ \mathsf{advWin} \leftarrow s\mathsf{UCMerkleExtraction}^{f_{\mathsf{MT}}}(\mathcal{B}_{\mathsf{MT}}^{f_1, \ldots, f_{\mathsf{k}}}(\mathcal{A}), \mathsf{l}, \mathsf{q}, \mathsf{k} \cdot (t_{\mathsf{q}} + 1), \mathsf{k}) \end{array}\right] \\ &\leq \kappa_{\mathsf{MT}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \mathsf{l}, \mathsf{q}, \mathsf{k}(t_{\mathsf{q}} + 1), \mathsf{k}) \ , \end{aligned}$$

where the last inequality follows since $\mathcal{B}_{\mathsf{MT}}$ makes at most t_{q} random oracle[14] and t_{p} programming queries to f_{MT}, submits at most $\mathsf{k} \cdot (t_{\mathsf{q}} + 1)$ roots, makes at most ℓ_{p} simulator queries and outputs at most k openings. By UC-friendly extraction, the probability that the advWin flag (as defined in that game) is set is then at most $\kappa_{\mathsf{MT}}(\lambda, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \mathsf{l}, \mathsf{q}, \mathsf{k} \cdot (t_{\mathsf{q}} + 1), \mathsf{k})$. □

9.6 UC-secure zkSNARKs from BCS

We combine the results in Sections 9.3 to 9.5 to show that, when instantiated with a suitable IOP, the BCS construction yields a UC-secure zkSNARK.

Theorem 9.14. *Let* IOP *be an interactive oracle proof with:*

- *(resp. strong) honest-verifier zero knowledge (Definition 9.4) with error* ζ_{IOP}.
- *(straightline) state-restoration knowledge soundness (Definition 9.3) with error* κ_{IOP}.

Set $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{l}, \mathsf{r}_{\mathsf{MT}}]$ *and* $\mathsf{ARG} := \mathsf{BCS}[\mathsf{IOP}, \mathsf{MT}, \mathsf{r}]$. *Then* $\Pi_{\mathsf{a}}[\mathsf{ARG}]$ $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-UC-realizes* $\mathcal{F}_{\mathrm{aARG}}$ *in the* GRO*-hybrid model with no simulation overhead and error*

$$z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$$

In the above we let

- $z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) + \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) + \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, \ell_{\mathsf{v}})$ *as in Theorem 6.1,*
- $\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 9.5.*
- $\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 9.9,*
- $\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 9.12.*

Proof. By Lemma 9.5 we obtain that the BCS construction has strong UC-completeness. By Lemma 9.9, we show that it is weak (resp. strong) UC-friendly zero knowledge. By Lemma 9.12 we conclude weak (resp. strong) UC-friendly knowledge soundness with respect to the simulator from the UC-friendly zero knowledge proof. Applying then Theorem 6.1 concludes the result. □

[14] Formally, $\mathcal{B}_{\mathsf{MT}}$ makes $t_{\mathsf{q}} + \mathsf{k} \cdot \mathsf{q}_{\mathsf{MT.Check}}$ random oracle queries, as those are required to compute Check. However, since b is only used to define E'_{extr}, we can modify the reduction adversary to avoid performing the spurious extra $\mathsf{k} \cdot \mathsf{q}_{\mathsf{MT.Check}}$ queries.

A An analysis of [IW14]

We define the **Hamming distance** of two strings of length n as $\Delta(f,g) \coloneqq \Pr_{i \leftarrow [n]}[f[i] \neq g[i]]$, and extend the notation to sets to have $\Delta(f,S) \coloneqq \min_{g \in S} \Delta(f,g)$, with the convention that $\Delta(f,\emptyset) \coloneqq 1$. For a relation R, we let $R[\mathbb{x}] \coloneqq \{\mathbb{w} : (\mathbb{x},\mathbb{w}) \in R\}$.

We also recall some notation for non-adaptive PCPs.

Definition A.1. *A probabilistically checkable proof* $\mathsf{PCP} = (\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}})$ *is* **non-adaptive** *if there exist deterministic algorithms* $\mathsf{S}, \mathsf{Q}, \mathsf{D}$ *such that, for every* $\mathbb{x}, \rho, \Pi$

$$\mathbf{V}_{\mathrm{PCP}}^{\Pi}(\mathbb{x};\rho) = \mathsf{D}(\mathsf{S}(\mathbb{x},\rho), \Pi[\mathsf{Q}(\mathbb{x},\rho)]) \enspace .$$

We write thus $\mathbf{V}_{\mathrm{PCP}} = (\mathsf{S},\mathsf{Q},\mathsf{D})$

We recall the definition of probabilistically checkable proof of proximity.

Definition A.2. *A tuple of algorithms* $\mathsf{PCPP} = (\mathbf{P}_{\mathrm{PCPP}}, \mathbf{V}_{\mathrm{PCPP}})$ *is a* **probabilistically checkable proof of proximity** *for a relation R with proximity parameter δ and proximity soundness error ϵ_{PCPP} if it satisfies the following two properties:*

- ***Completeness*** *For every* $(\mathbb{x},\mathbb{w}) \in R$,

$$\Pr\left[\ \mathbf{V}_{\mathrm{PCPP}}^{\mathbb{w},\Pi}(\mathbb{x}) = 1 \ \middle|\Pi \leftarrow \mathbf{P}_{\mathrm{PCPP}}(\mathbb{x},\mathbb{w})\right] = 1 \enspace .$$

- ***Soundness*** *For every* $\mathbb{x}, \mathbb{w}$*, if* $\Delta(\mathbb{w}, R[\mathbb{x}]) \geq \delta$*, for any proof* $\widetilde{\Pi}$*,*

$$\Pr\left[\mathbf{V}_{\mathrm{PCPP}}^{\mathbb{w},\widetilde{\Pi}}(\mathbb{x}) = 1\right] \leq \epsilon_{\mathrm{PCPP}}(|\mathbb{x}|) \enspace .$$

If $\delta = 0$, then we say PCPP *is* **exact**.

The notions of zero knowledge and strong zero knowledge for PCPPs are defined analogously to Definition 8.3 and Definition 8.3 but including queries to the witness in the view as well.

We will need some notation for secret-sharing.

Definition A.3. *Let* $\Pi \in \{0,1\}$ *be a bit. A list of bits* $\Pi^{(1)}, \ldots, \Pi^{(d+1)} \in \{0,1\}$ *is a d-***secret-share** *of* Π *iff* $\oplus_{i \in [d+1]} \Pi^{(i)} = \Pi$*. We also write* $\mathsf{SShare}(\Pi)$ *for the algorithm that samples secret shares of* Π *uniformly at random, and extend both the definition and this notation to strings in the obvious way.*

For a non-adaptive PCP, with $\mathbf{V}_{\mathrm{PCP}} = (\mathsf{S},\mathsf{Q},\mathsf{D})$, we define relation of accepting views:

$$R(\mathbf{V}_{\mathrm{PCP}}) \coloneqq \{(s,\mathbf{a}) : \mathsf{D}(s,\mathbf{a}) = 1\} \enspace .$$

and its d-private equivalent, namely

$$R^{(d)}(\mathbf{V}_{\mathrm{PCP}}) \coloneqq \left\{(s, (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(d+1)})) : (s, \oplus_{i \in [d+1]} \mathbf{a}^{(i)}) \in R(\mathbf{V}_{\mathrm{PCP}})\right\} \enspace .$$

Construction A.4. Let $d \in \mathbb{N}$, and let $\mathsf{PCP}_{\mathrm{out}} = (\mathbf{P}_{\mathrm{out}}, \mathbf{V}_{\mathrm{out}})$ be a non-adaptive PCP for a relation R with $\mathbf{V}_{\mathrm{out}} = (\mathsf{S},\mathsf{Q},\mathsf{D})$. Let $\mathsf{PCPP}_{\mathrm{in}} = (\mathbf{P}_{\mathrm{in}}, \mathbf{V}_{\mathrm{in}})$ be an PCPP for the relation $R^{(d)}(\mathbf{V}_{\mathrm{out}})$. We define a new PCP $\mathsf{IW}[\mathsf{PCP}_{\mathrm{out}}, \mathsf{PCPP}_{\mathrm{in}}, d] = (\mathbf{P}_{\mathrm{PCP}}, \mathbf{V}_{\mathrm{PCP}})$ for R as follows.

```
$\mathbf{P}_{\text{PCP}}(\mathbb{x}, \mathbb{w})$:
1. Compute $\Pi_{\text{out}} \leftarrow \mathbf{P}_{\text{out}}(\mathbb{x}, \mathbb{w})$.
2. Set $\Pi^{(1)}_{\text{out}}, \ldots, \Pi^{(d+1)}_{\text{out}} \leftarrow \mathsf{SShare}(\Pi_{\text{out}})$.
3. For $\rho_{\text{out}} \in \{0,1\}^{\mathsf{r}_{\text{out}}}$:
   (a) Compute $s_{\rho_{\text{out}}} := \mathsf{S}(\mathbb{x}, \rho_{\text{out}}), Q_{\rho_{\text{out}}} := \mathsf{Q}(\mathbb{x}, \rho_{\text{out}})$.
   (b) For $i \in [d+1]$, set $\mathbf{a}^{(i)} := \Pi^{(i)}_{\text{out}}[Q_{\rho_{\text{out}}}]$.
   (c) Compute $\Pi[\rho_{\text{out}}] \leftarrow \mathbf{P}_{\text{in}}(s_{\rho_{\text{out}}}, (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(d+1)}))$.
4. Output $\Pi := ((\Pi^{(i)}_{\text{out}})_{i \in [d+1]}, (\Pi[\rho_{\text{out}}])_{\rho_{\text{out}}})$.

$\mathbf{V}^{\Pi}_{\text{PCP}}(\mathbb{x})$:
1. Sample $\rho_{\text{out}} \leftarrow \{0,1\}^{\mathsf{r}_{\text{out}}}$.
2. Compute $s := \mathsf{S}(\mathbb{x}, \rho_{\text{out}}), Q := \mathsf{Q}(\mathbb{x}, \rho_{\text{out}})$
3. Return $\mathbf{V}_{\text{in}}^{\Pi^{(1)}_{\text{out}}[Q], \ldots, \Pi^{(d+1)}_{\text{out}}[Q], \Pi[\rho_{\text{out}}]}(s)$.
```

We show that, if the outer PCP is knowledge sound, so is the composed one.

Lemma A.5. *Suppose* $\mathsf{PCP}_{\text{out}}$ *is non-adaptive and has knowledge soundness error* κ_{PCP}*, and* $\mathsf{PCPP}_{\text{in}}$ *is exact and has soundness error* ϵ_{PCPP}*. Then* $\mathsf{IW}[\mathsf{PCP}_{\text{out}}, \mathsf{PCPP}_{\text{in}}, d]$ *has knowledge soundness error* $\kappa_{\text{PCP}} + \epsilon_{\text{PCPP}}$*.*

Proof. Letting $\mathbf{E}_{\text{out}}$ be the extractor for $\mathsf{PCP}_{\text{out}}$, the new extractor $\mathbf{E}_{\text{PCP}}$ is defined by $\mathbf{E}_{\text{PCP}}(\mathbb{x}, \Pi) := \mathbf{E}_{\text{out}}(\mathbb{x}, \oplus_{i \in [d+1]} \Pi^{(i)}_{\text{out}})$. First, suppose that $\mathbf{V}_{\text{PCP}}$ is accepting. Thus, unless with probability at most ϵ_{PCPP}, since the inner PCPP is exact, $(\Pi^{(1)}_{\text{out}}[Q], \ldots, \Pi^{(d+1)}_{\text{out}}[Q]) \in R^{(d)}(\mathbf{V}_{\text{out}})[s]$ and thus $\mathbf{a} := \oplus_{i \in [d+1]} \Pi^{(i)}_{\text{out}}[Q]$ must be in $R(\mathbf{V}_{\text{out}})[s]$.

Letting $\Pi := \oplus_{i \in [d+1]} \Pi^{(i)}_{\text{out}}$, we see that whenever ρ_{out} makes $\mathbf{V}_{\text{PCP}}$ accept, then (unless with a probability of at most ϵ_{PCPP}), then $\mathbf{V}_{\text{out}}$ would have accepted as well, and thus extraction succeeds on Π with probability at least $1 - \kappa_{\text{PCP}}$. □

Remark A.6. Note that Lemma A.5 is already sufficient to conclude that knowledge sound honest-verifier zero knowledge PCPs exist (and thus UC-secure zkSNARKs via our main results), the further work that we include simply aims to establish the strong HVZK property.

Next, we show that if the inner PCPP is strong honest-verifier zero knowledge, the resulting PCP also is (as long as the inner PCPP does not make too many queries).

Lemma A.7. *Let* $\mathsf{q} \leq d$*. Suppose that* $\mathsf{PCP}_{\text{out}}$ *is a non-adaptive PCP, and that* $\mathsf{PCPP}_{\text{in}}$ *is strong honest-verifier zero knowledge whose verifier makes at most* q *oracle queries. Then* $\mathsf{IW}[\mathsf{PCP}_{\text{out}}, \mathsf{PCPP}_{\text{in}}, d]$ *is also strong honest-verifier zero knowledge.*

Proof. Let $\mathbf{S}_{\text{in}}$ be the simulator for $\mathsf{PCPP}_{\text{in}}$. We build a new simulator as follows.

```
$\mathbf{S}_{\text{PCP}}(\mathbb{x})$:
1. Sample $\rho_{\text{out}} \leftarrow \{0,1\}^{\mathsf{r}_{\text{out}}}$.
2. Compute $s := \mathsf{S}(\mathbb{x}, \rho_{\text{out}})$.
3. Compute $(\rho_{\text{in}}, Q, \mathbf{a}, z'_{\mathsf{SIM}}) \leftarrow \mathbf{S}_{\text{in}}(s)$ answering witness oracle queries with uniformly random bits.
4. Return $(\rho := (\rho_{\text{out}}, \rho_{\text{in}}), Q, \mathbf{a}, z_{\mathsf{SIM}} := (\mathbb{x}, \rho_{\text{out}}, Q, \mathbf{a}, z'_{\mathsf{SIM}}))$.
$\mathbf{S}_{\text{PCP}}(\mathbb{w}, z_{\mathsf{SIM}})$:
1. Sample $\rho_{\mathbf{P}_{\text{out}}} \leftarrow \{0,1\}^{\mathsf{r}_{\mathbf{P}_{\text{out}}}}$
2. Compute $\Pi_{\text{out}} \leftarrow \mathbf{P}_{\text{out}}(\mathbb{x}, \mathbb{w}; \rho_{\mathbf{P}_{\text{out}}})$.
```

3. Parse $Q \coloneqq (Q^{(1)}, \dots, Q^{(d+1)}, Q_{\text{in}})$ and $\mathbf{a} \coloneqq (\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(d+1)}, \mathbf{a}_{\text{in}})$ dividing the queries-answers by the oracle queried.
4. Sample $\Pi^{(1)}_{\text{out}}, \dots, \Pi^{(d+1)}_{\text{out}}$ uniformly at random conditioned on $\oplus_{i \in [d]} \Pi^{(1)}_{\text{out}} = \Pi_{\text{out}}$ and $\Pi^{(i)}_{\text{out}}[Q^{(i)}] = \mathbf{a}^{(i)}$.
5. Compute $\rho_{\mathbf{P}_{\text{in}}}[\rho_{\text{out}}] \leftarrow \mathbf{S}_{\text{in}}(\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(d+1)}, z'_{\mathsf{SIM}})$
6. For $\rho \in \{0,1\}^{\mathbf{r}_{\mathbf{P}_{\text{out}}}} \setminus \{\rho_{\text{out}}\}$, set $\rho_{\mathbf{P}_{\text{in}}}[\rho] \leftarrow \{0,1\}^{\mathbf{r}_{\mathbf{P}_{\text{in}}}}$
7. Return $\rho_{\mathbf{P}} \coloneqq (\rho_{\mathbf{P}_{\text{out}}}, (\Pi^{(i)}_{\text{out}})_{i \in [d+1]}, (\rho_{\mathbf{P}_{\text{in}}}[\rho])_\rho)$.

The distinguishing advantage on adversary is bound by the statistical distance of the following variables in the real and simulated games.

$$((\rho_{\text{out}}, \rho_{\text{in}}), Q, \mathbf{a}, (\rho_{\mathbf{P}_{\text{out}}}, (\rho_{\mathbf{P}_{\text{in}}}[\rho])_\rho))$$

Note that, by completeness of $\mathsf{PCP}_{\text{out}}$, the answers $\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(d+1)} \in R^{(d)}(\mathbf{V}_{\text{out}})[s]$, and thus by the strong honest-verifier zero knowledge properties the distance of the variables is at most the simulation error (since all the other variables are identically distributed in both games). ☐

We turn our attention to designing the inner PCPP for our construction.

Lemma A.8. *Let* $\mathsf{3COL}$ *denote the graph 3 coloring* NP*-complete problem. There exists an exact proof of proximity for* R *with:*
- *Perfect completeness.*
- *Soundness error* $\epsilon_{\text{PCPP}} = \left(1 - \Omega(\frac{1}{n^2})\right)$
- *Constant queries (in fact at most 3).*
- *Perfectly strong honest-verifier zero knowledge.*

Sketch. The prover first samples $\varphi \leftarrow S_3$. It then computes $\theta \coloneqq \varphi \circ \sigma$, and outputs the proof $\Pi \coloneqq (\varphi, \theta(v)_{v \in V})$. The verifier samples a random coin, and does one of the following things:
1. Samples $e \in E$ and checks $\theta(u) \neq \theta(v)$.
2. Samples $v \in V$ and checks $\varphi(\sigma(v)) = \theta(v)$.

Perfect completeness is easy to see.

For soundness, assume that $\sigma \notin \mathsf{3COL}[G]$, there are two cases to consider. Suppose first that the malicious prover sends $\theta \not\equiv \varphi \circ \sigma$. Then, there exists at least one vertex at which the two disagree, and thus the verifier will reject with probability at least $\geq \frac{1}{2} \cdot \frac{1}{n}$. In the other case, the prover sent a $\theta \equiv \varphi \circ \sigma$, and thus θ cannot be a coloring of G (or else σ would also be one) and thus there must be at least one edge at which $\theta(u) = \theta(v)$, which makes the verifier reject with probability at least $\frac{1}{2} \cdot \frac{1}{|E|} \geq \frac{1}{2\binom{n}{2}}$. Finally, for strong honest-verifier zero knowledge, the simulator (which has oracle access to σ) behaves as following.
1. Sample $b \leftarrow \{0,1\}$.
   (a) If $b = 0$, sample $e = (u, v) \leftarrow E$, and two random distinct colors $c_u \neq c_v$, outputs those.
   (b) If $b = 1$, sample $v \leftarrow V$ and queries $\sigma(v)$. It then samples $\varphi \leftarrow S_3$ and answer the query to φ with φ and that to θ with $\varphi(\sigma(v))$.

When asked to come up with prover randomness the simulator looks at the bit that it previously sampled, and acts as follows:
1. If $b = 0$, query $\sigma(u), \sigma(v)$, and compute a random permutation with $\varphi(\sigma(u)) = c_u$ and $\varphi(\sigma(v)) = c_v$.
2. If $b = 1$, return φ.

Note that the resulting view is identically distributed to that in an honest execution, and thus this simulator gives perfect strong honest-verifier zero knowledge. ☐

This inner PCP of course has very small soundness, but this can be amplified using the same techniques as in [IW14] (sequential repetition) while preserving strong zero knowledge.

Next, we require a straightline extractable outer PCP, and to achieve it we turn to [BFLS91].

Lemma A.9 ([BFLS91])**.** *Let R be an NP-relation. There exists a PCP for R with (i) perfect completeness; (ii) constant knowledge soundness error; (iii) polynomial proof length; and (iv) polylogarithmic query complexity.*

We wrap things here with details of the construction.

Construction A.10. Let $\mathsf{PCP}_{\text{out}} := (\mathbf{P}_{\text{out}}, \mathbf{V}_{\text{out}})$ be the PCP guaranteed by Lemma A.9 for the relation R, with soundness amplified to $\frac{1}{\omega(n^2)}$ by sequential repetition applied logarithmically many times. Instead, for $\mathsf{PCPP}_{\text{in}}$ use the PCPP from Lemma A.8 (adapted to the relation $R^{(3)}(\mathbf{V}_{\text{out}})$). The final PCP PCP is obtained from $\mathsf{IW}[\mathsf{PCPP}_{\text{in}}, \mathsf{PCP}_{\text{out}}, 3]$ by doing sequential repetition logarithmically many times to obtain soundness $2^{-\lambda}$ (which is constant in n). From Lemma A.5 and Lemma A.7 it is easy to see that the resulting PCP is knowledge sound and perfectly strong honest-verifier zero knowledge.

Acknowledgments

We thank Ran Canetti, Megan Chen, Anna Lysyanskaya and Leah Namisa Rosenbloom for insightful discussions on the UCGS framework, the ARG (i.e., NIZKPoK) ideal functionality, and global random oracles. We also thank Francesco Intoci, Giorgio Seguini, Kien Tuong Truong, Eylon Yogev for valuable feedback and suggestions on earlier drafts of this paper. The authors are partially supported by the Ethereum Foundation.

References

[AGRS24] Behzad Abdolmaleki, Noemi Glaeser, Sebastian Ramacher, and Daniel Slamanig. "Circuit-Succinct Universally Composable NIZKs with Updatable CRS". In: *Proceedings of the 37th IEEE Computer Security Foundations Symposium*. CSF '24. 2024.

[ARS20] Behzad Abdolmaleki, Sebastian Ramacher, and Daniel Slamanig. "Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically". In: *Proceedings of the 27th ACM Conference on Computer and Communications Security*. CCS '20. 2020, pp. 1987–2005.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO '19. 2019, pp. 733–764.

[BCFGRS17] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. "Zero Knowledge Protocols from Succinct Constraint Detection". In: *Proceedings of the 15th Theory of Cryptography Conference*. TCC '17. 2017, pp. 172–206.

[BCHTZ20] Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. "Universal Composition with Global Subroutines: Capturing Global Setup Within Plain UC". In: *Proceedings of the 18th Theory of Cryptography Conference*. TCC 20. 2020, pp. 1–30.

[BCRSVW19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '19. 2019, pp. 103–128.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. "Checking computations in polylogarithmic time". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. STOC '91. 1991, pp. 21–32.

[BS21] Karim Baghery and Mahdi Sedaghat. "TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model". In: *Proceedings of the 20th International Conference on Cryptology and Network Security*. CANS '21. 2021, pp. 531–551.

[CDGLN18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. "The Wonderful World of Global Random Oracles". In: *Proceedings of the 37th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '18. 2018, pp. 280–312.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. "Universally Composable Security with Global Setup". In: *Proceedings of the 4th Theory of Cryptography Conference*. TCC '07. 2007, pp. 61–85.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. "Practical UC security with a Global Random Oracle". In: *Proceedings of the 21st ACM Conference on Computer and Communications Security*. CCS '14. 2014, pp. 597–608.

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. "Succinct Arguments in the Quantum Random Oracle Model". In: *Proceedings of the 17th Theory of Cryptography Conference*. TCC '19. 2019, pp. 1–29.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. URL: `https://github.com/hash-based-snargs-book`.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*. FOCS '01. 2001, pp. 136–145.

[Can20] Ran Canetti. "Universally Composable Security". In: *Journal of the ACM* 67 (2020), pp. 1–94.

[DDOPS01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. "Robust Non-interactive Zero Knowledge". In: *Proceedings of the 21st Annual International Cryptology Conference*. CRYPTO '01. 2001, pp. 556–598.

[Fis05] Marc Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *Proceedings of the 25th Annual International Cryptology Conference*. CRYPTO '05. 2005, pp. 152–168.

[GKOPTT23] Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. "Witness-Succinct Universally-Composable SNARKs". In: *Proceedings of the 42nd Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '23. 2023, pp. 315–346.

[Gro06] Jens Groth. "Simulation-sound NIZK proofs for a practical language and constant size group signatures". In: *Proceedings of the 12th International Conference on Theory and Application of Cryptology and Information Security*. ASIACRYPT '06. 2006, pp. 444–459. URL: `http://www0.cs.ucl.ac.uk/staff/J.Groth/NIZKGroupSignFull.pdf`.

[IW14] Yuval Ishai and Mor Weiss. "Probabilistically Checkable Proofs of Proximity with Zero-Knowledge". In: *Proceedings of the 11th Theory of Cryptography Conference*. TCC '14. 2014, pp. 121–145.

[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, Paper 2015/1093. 2015.

[Ks22] Yashvanth Kondi and abhi shelat. "Improved Straight-Line Extraction in the Random Oracle Model with Applications to Signature Aggregation". In: *Proceedings of the 28th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '22. 2022, pp. 279–309.

[LR22a] Anna Lysyanskaya and Leah Namisa Rosenbloom. *Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions*. Cryptology ePrint Archive, Paper 2022/1484. 2022.

[LR22b] Anna Lysyanskaya and Leah Namisa Rosenbloom. "Universally Composable Σ-protocols in the Global Random-Oracle Model". In: *Proceedings of the 20th Theory of Cryptography Conference*. TCC '22. 2022, pp. 203–233.

[Mer89] Ralph C. Merkle. "A certified digital signature". In: *Proceedings of the 9th Annual International Cryptology Conference*. CRYPTO '89. 1989, pp. 218–238.

[Mic00] Silvio Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS '94., pp. 1253–1298.