# Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Brown University, Providence RI 02906, USA
{anna_lysyanskaya,leah_rosenbloom}@brown.edu

**Abstract.** Non-interactive zero-knowledge proofs of knowledge (NIZK-PoK) serve as a key building block in many important cryptographic constructions. Achieving universally composable NIZKPoK secure against adaptive corruptions was a long-standing open problem, recently solved by Canetti, Sarkar, and Wang (Asiacrypt'22). This sole known construction requires heavy cryptographic machinery such as correlation-intractable hash functions, and is not ready for use in practice. In this paper, we give constructions of adaptively secure universally composable NIZKPoK in the global random-oracle model; we consider both the programmable and the non-programmable versions of the model. For many practical NIZK proof systems, our constructions incur only a polylogarithmic slowdown factor compared to stand-alone security.

# Table of Contents

## 1 Introduction

An adaptive adversary attacking a cryptographic protocol can decide on the fly which protocol participant(s) it wants to corrupt. As a result, the adversary gains access to the memory of a device that is currently running, or has previously run, the protocol. In the event that the adversary does not corrupt a party, the protocol must protect this party's security. Achieving security from such adaptive corruptions is a notoriously difficult problem.

*Non-interactive zero-knowledge proofs of knowledge* (NIZKPoK) are an important building block in cryptographic constructions. For example, NIZKPoK allow honest protocol participants to prove they have formatted their protocol messages correctly, and catch any adversary who is trying to deviate maliciously from a protocol specification. This technique transforms protocols secure in the "honest-but-curious" model (in which protocol participants are guaranteed to act according to the protocol) into those secure in the more realistic "malicious" model (in which the adversary can issue adaptive corruptions and execute arbitrary code). NIZKPoK that are safe to deploy in such a complex cryptographic system must have *composable security*: they must maintain security properties when composed concurrently with other protocols.

Thus, constructing a non-interactive zero-knowledge proof system that can withstand adaptive corruptions in the *universal composition* (UC) framework of Canetti [14] is a natural and well-motivated problem. Obtaining *adaptive and composable* NIZKPoK was a long-standing open problem until, in a paper to appear in Asiacrypt'22, Canetti, Sarkar, and Wang [19] developed a compiler that leverages UC non-interactive commitments, Camenisch and Damgård's commitment-based straight-line extractor [5], and a correlation intractable hash function to obtain adaptive UC NIZKPoK from standard assumptions. In particular, Canetti et al. consider $\Sigma$-protocols in the $\mathcal{F}_{\texttt{NICOM}}$ model, which assumes the first message of the $\Sigma$-protocol is a UC non-interactive commitment, instantiated using equivocal commitments and CCA-2 secure public-key encryption with oblivious ciphertext sampling. Camenisch and Damgård's extractor adds an additional $O(\lambda)$ "commit-and-open" operations for security parameter $\lambda$, and correlation intractable hash functions typically rely on heavy-weight primitives like fully homomorphic encryption or indistinguishability obfuscation.

In this paper, we show how to obtain *efficient and adaptive UC NIZKPoK* from any $\Sigma$-protocol with a natural adaptivity property in the global random-oracle model. For most practical $\Sigma$-protocols—i.e. those on the cusp of widespread adoption in practice—our construction does not require any additional cryptographic machinery; rather, it is as efficient as the $\Sigma$-protocol under the randomized Fischlin transform [24,26], which creates a multiplicative overhead for the prover that is only superlogarithmic in $\lambda$. By treating the random oracle (RO) as a global subroutine in the universal composition with global subroutines (UCGS) model [1], our adaptive NIZKPoK retain composability even when the global RO is shared among different sessions and protocols, as is likely in practice.

**Adaptive $\Sigma$-protocols.** A standard $\Sigma$-protocol [23] is a three-move, public coin zero-knowledge proof system over a binary NP relation $R$, where statements $x$ are proven to be in the language $L_R$ using a "witness" $w$ such that $(x, w) \in R$. The 'three-move" form of a $\Sigma$-protocol is defined as follows: a prover $P$ sends a verifier $V$ a first message $\texttt{com}$, $V$ sends $P$ a uniformly random challenge $\texttt{chl}$, $P$ responds with a value $\texttt{res}$, and $V$ decides whether or not to "accept" (output 1) based on the proof transcript $(x, \texttt{com}, \texttt{chl}, \texttt{res})$. The "zero-knowledge proof system" piece of the definition implies three properties: *completeness*, *special honest-verifier zero-knowledge* (SHVZK), and *special soundness* (SS). The completeness property says that if $P$ forms a proof using the three-move form on input $(x, w) \in R$, then $V$ always accepts. The SHVZK property states there must exist a *simulator* algorithm $\texttt{SimProve}$ that, on input the statement *and the challenge in advance*, can produce a proof that looks statistically close to that of a "real" prover without using a witness. Finally, the SS property implies an *extractor* algorithm $\texttt{Extract}$ that can compute a witness $w$ from any two proofs of a statement $x$ with the same first message but different challenges.

$P$ is a *probabilistic* Turing machine, and its random coins $r$ determine the value of the first message $\texttt{com}$. This randomness is the crux of the zero-knowledge property—it is the only information hidden from the adversary during the SHVZK experiment. In the *adaptive* SHVZK experiment, the adversary can corrupt $P$ after $P$ has already issued proofs, revealing the entirety of $P$'s random tape. We therefore consider an "adaptive" $\Sigma$-protocol to be one that has an additional simulator algorithm, $\texttt{SimRand}$, that uses information from $\texttt{SimProve}$ and the witness that was supposedly used to compute the proof to generate convincing-looking coins for $P$'s random tape. Many popular $\Sigma$-protocols are adaptive according to this definition.

**Adaptive Straight-Line Compilers.** A straight-line compiler (SLC) [27] is an algorithm $\texttt{SLC}$ that takes as input any $\Sigma$-protocol $\Sigma_R$ for relation $R$ (as described above) and produces as output a *non-interactive, straight-line extractable* (NISLE) proof system $\Pi_R^{\texttt{SLC}}$ for $R$ in the random-oracle model (ROM). Recall that a proof system is straight-line extractable if the challenger in the security experiment can obtain the two proofs needed to run the $\texttt{Extract}$ algorithm (and thereby compute a witness for the statement $x$) immediately after an adversarial prover issues a single proof, i.e. without any further interaction with the prover. In the ROM, the security experiment has access to the adversary's queries to the random oracle (RO); therefore, as long as an adversarial prover is forced to query the RO on two proof transcripts with the same first message but different challenges, the $\texttt{Extract}$ algorithm can immediately compute a witness for the prover's statement. The randomized Fischlin transform [24,26] sets the proof repetition parameters such that the prover is guaranteed (with probability that is overwhelming in the security parameter) to issue two such transcripts to the RO. Other forms of straight-line extraction, such as the aforementioned commit-and-open construction due to Camenisch and Damgård [5] work in the plain model, but require the prover to couple each repetition with a cryptographic commitment, creating substantial computational overhead.

We define an *adaptive* SLC as an SLC that preserves the adaptive security properties of the underlying $\Sigma$-protocol—that is, the resulting proof system $\Pi_R^{\mathsf{SLC}}$ is also secure against adaptive corruptions. Specifically, we define new adaptive versions of the non-interactive multiple SHVZK and special simulation-soundness (SSS) properties guaranteed by the regular SLC, and show these properties (with standard completeness, Definitions 7, 8, and 9) are both *necessary* and *sufficient* to obtain adaptive UC NIZKPoK in the global ROM.

**Theorem 1 (Informal).** *If a protocol $\Pi$ creates adaptive UC NIZKPoK in the ROM, then it must have the properties from Definitions 7-9.*

**Theorem 2 (Informal).** *Given any adaptive $\Sigma$-protocol $\Sigma_R$ and adaptive straight-line compiler, we construct adaptive UC NIZKPoK in the programmable global ROM (i.e. assuming the global RO can be programmed by the security experiment).*

**Theorem 3 (Informal).** *Given any adaptive $\Sigma$-protocol $\Sigma_R$, any adaptive straight-line compiler, and an adaptive version of Lysyanskaya and Rosenbloom's OR-protocol compiler [27], we construct adaptive UC NIZKPoK in the (non-programmable) global RO-CRS hybrid model.*

We prove that the randomized Fischlin transform meets our definition of an adaptive SLC, and can therefore efficiently transform any adaptive $\Sigma$-protocol into adaptive UC NIZKPoK in the global ROM.

**Theorem 4 (Informal).** *The randomized Fischlin transform [24,26] is an adaptive straight-line compiler; that is, it preserves the security properties of adaptive $\Sigma$-protocols under transformation.*

**Universal Composition with Global Subroutines.** The security experiment in the UC framework [14,1] tests a session of a cryptographic protocol $\Pi$ in the presence of arbitrary concurrent protocols, which are modeled using an adversarial "environment" machine $\mathcal{Z}$. $\mathcal{Z}$ controls the corrupted protocol participants through an adversary machine, and can also send inputs to honest protocol participants and observe their outputs. In the "real-world" half of the security experiment, the honest parties form their outputs according to the protocol $\Pi$, and the adversary machine is the traditional notion of a cryptographic protocol adversary $\mathscr{A}$ (i.e. one that executes arbitrary instructions). In the "ideal-world" experiment, honest parties are just placeholder "dummies" who pass all of their inputs to an ideal functionality $\mathcal{F}$, which acts according to an "ideal" version of the protocol. The adversary $\mathscr{A}$ is replaced with an "ideal adversary" $\mathcal{S}$, who acts like $\mathscr{A}$ when talking to $\mathcal{Z}$, but also communicates with $\mathcal{F}$ and simulates the view of the corrupted parties. When $\mathcal{Z}$ wishes to adaptively corrupt a protocol participant $P$, $\mathcal{F}$ hands over any relevant information about $P$ to $\mathcal{S}$, and all future communications with $P$ are handled through $\mathcal{S}$. If $\mathcal{Z}$ cannot distinguish its interaction with "real" parties running $\Pi$ in the presence of the "real" adversary

$\mathscr{A}$ from its interaction with "ideal" (dummy) parties running $\mathcal{F}$ in the presence of the "ideal" adversary, then $\Pi$ is said to "UC-realize" (or be UC-secure with respect to) $\mathcal{F}$ in the presence of $\mathcal{Z}$.

In the original version of the framework [14], the "test session" of a protocol must be *subroutine respecting*—its subroutines cannot process inputs coming from other sessions or protocols. Since it is reasonable to expect protocols in real-world applications to share a common reference string (CRS) or RO, subsequent versions of the UC framework such as joint-state UC (JUC) [18] and general UC (GUC) [15] tweak the model to allow this feature. The GUC model in particular is designed to incorporate *global functionalities* $\mathcal{G}$ that can be accessed by *any* party in the security experiment. However, Badertscher, Canetti, Hesse, Tackmann, and Zikas [1] observe subtle inconsistencies in the GUC model stemming from the (unconstrained) environment's ability to spawn parties with arbitrary session identifiers (see Appendix A in Badertscher et al. [1]). Rather than relax the constraints of the model, Badertscher et al. leverage the "shell-and-body" construct of the original UC framework [14] to create a UC with global subroutines (UCGS) model. In the UCGS model, the *global subroutine protocol* $\mathcal{G}$ is "wrapped" into a joint body with the test protocol session $\pi$ by a "shell" protocol $\mathtt{M}$. The shell $\mathtt{M}$ processes all communications in and out of $\pi$ and $\mathcal{G}$ and ensures that the combined entity $\mathtt{M}[\pi, \mathcal{G}]$ is subroutine respecting, even though $\pi$ and $\mathcal{G}$ are not.

In this work, the ideal functionality $\mathcal{F}$ is the ideal functionality $\mathcal{F}_{\mathtt{aNIZK}}$ for adaptive NIZKPoK. The global subroutines are the restricted programmable observable global RO $\mathcal{G}_{\mathtt{rpoRO}}$ of Camenisch, Drijvers, Gagliardoni, Lehmann, and Neven [6] and the restricted observable (non-programmable) global RO $\mathcal{G}_{\mathtt{roRO}}$ of Canetti, Jain, and Scafuro [15].

**Applications.** NIZKPoK are widely used in group [12,5], blind [25], threshold [3], aggregate [26], and multi- [3,22] signatures, cryptographic shuffles [34,28] and accumulators [2,10], anonymous networks [20], credentials [9], e-cash [8], e-token [7], and voting [30,28,33], distributed ledgers [30], verifiable secret sharing [33] and encryption schemes [5,11] that are secure against adaptively chosen ciphertext attacks (CCA security). We demonstrate that many of the $\Sigma$-protocols used as core building blocks in these constructions, including proofs of knowledge of discrete logarithm [32] and proofs of knowledge of $n$ representations of discrete logarithm [13,11], qualify as adaptive $\Sigma$-protocols, and can therefore be efficiently transformed into adaptive UC NIZKPoK in the global ROM. The result is an immediate and significant boost in the security potential of all of the above real-world systems.

**Theorems 5-7 (Informal).** *Many common $\Sigma$-protocols are adaptive $\Sigma$-protocols, and can therefore be converted to efficient and adaptive UC NIZPoK in the global ROM using our techniques.*

**Organization.** In Section 2, we introduce the various oracles, models, and security definitions we will use in our constructions. Section 3 contains formal

definitions of adaptive $\Sigma$-protocols and adaptive SLCs. We prove in Section 4 that the security guaranteed by adaptive SLCs is both necessary and sufficient to create adaptive UC NIZKPoK in the programmable global ROM, and sufficient along with an ideal CRS functionality [27] in the (non-programmable) global RO-CRS hybrid model. In Section 5 we demonstrate that the randomized Fischlin transform [24,26] is an adaptive SLC, and can therefore create efficient and adaptive UC NIZKPoK from any adaptive $\Sigma$-protocol. Finally in Section 6, we prove that many common $\Sigma$-protocols are adaptive, concluding that efficient and adaptive UC NIZKPoK are realizable for a variety of real-world systems.

## 2 Preliminaries

In this section, we give preliminary definitions of the global RO (Section 2.1) and RO-CRS hybrid (Section 2.2) models we will use in our constructions, as well as a specification of the adaptive corruption mechanism (Section 2.3), the ideal adaptive NIZKPoK functionality $\mathcal{F}_{\texttt{aNIZK}}$ (Section 2.4), and finally adaptive security in the UCGS model (Section 2.5).

### 2.1 The Global Random Oracle Model(s)

We will demonstrate how to obtain adaptive UC NIZKPoK in two global random oracle models: the restricted *observable* global ROM of Canetti et al. [17] and the restricted *programmable* observable global ROM of Camenisch et al. [6]. Recall from the introduction that NIZKPoK in the ROM traditionally require a proof simulator algorithm `SimProve` that *programs* the outputs of the RO, and an extractor algorithm `Extract` that *observes* the adversary's RO queries. While making the global RO programmable is easier from a construction standpoint, the non-programmable model is closer to the intended vision of a truly "global" RO that cannot be edited or controlled by any one entity. We highlight and discuss the differences between the two models below; the formal versions of the definitions are provided in the supplementary materials (A.2).

The global ROs in both models have a traditional random function interface, `Query`, which takes an any-length string as input and returns a uniformly random $\ell$-bit string as output [4]. The "global" designation refers to the fact that there exists a single instance of the oracle for all sessions of a protocol, and potentially across protocols (as opposed to one functionality per protocol session, as in the standard UC model [14]). In order to be considered a global subroutine in the UCGS model [1], the global RO must be *subroutine respecting* and *regular* with respect to the challenge protocol. Both global ROs are subroutine respecting since the "extended instance" of each includes querying parties in any session, and no oracle subroutines interact directly with the querent. They are also regular with respect to the challenge protocol, since they neither invoke new NIZKPoK protocol participants nor run them as subroutines.

The observable-only RO $\mathcal{G}_{\texttt{roRO}}$ [17,6] records all "illegitimate" queries made for a session $s$ by parties with an $\texttt{sid} \neq s$, which captures all of the environment's direct queries (since the environment is external to all legitimate protocol

sessions by definition). The observability property is captured by the interface `Observe`, which takes a session $s$ as input and produces a list of illegitimate queries $\mathcal{Q}_s$ for $s$ as output. Since the only queries in $\mathcal{Q}_s$ are adversarial, we model $\mathcal{Q}_s$ as completely public—$\mathcal{G}_{\texttt{roRO}}$ can release it to anyone who asks [6].

The *programmable* version $\mathcal{G}_{\texttt{rpoRO}}$ [6] builds on the observable-only functionality with an additional interface, the `IsProgrammed` interface, which reveals the programmed entries for a session $s$ to any parties in the same session. Again since the environment is external to all legitimate protocol sessions by definition, it will not be able to query the `IsProgrammed` interface directly, and must instead ask through the corrupted session participants (in the ideal world, the simulator can always return "false" to corrupted session participants' `IsProgrammed` queries). Unlike the "illegitimate queries" of the observation interface, the list of programmed queries cannot be made public to everyone (or the environment could trivially distinguish the real from ideal experiments). It is unclear to date how this distinction affects the security of protocols under composition in the global ROM, if at all. In the meantime, we recall in the next section the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model, which will allow us to obtain adaptive UC NIZKPoK without programming the global RO.

### 2.2 The $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid Model

In the observable-only $\mathcal{G}_{\texttt{roRO}}$-hybrid model, the `SimProve` algorithm has no additional power over a regular prover, since it cannot program $\mathcal{G}_{\texttt{roRO}}$. Thus, NIZKPoK in the plain $\mathcal{G}_{\texttt{roRO}}$-hybrid model are impossible [17,6,31,16,15]. To work around this impossibility result and construct UC NIZKPoK while avoiding the session-localized `IsProgrammed` interface of the programmable global RO $\mathcal{G}_{\texttt{rpoRO}}$, Lysyanskaya and Rosenbloom introduce the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model [27], where $\mathcal{F}_{\texttt{CRS}}$ is the (local) ideal common reference string (CRS) functionality.

In the real-world execution of the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model, $\mathcal{F}_{\texttt{CRS}}$ has one interface, `Query`, that simply returns a consistent CRS to the participants of a particular session $s$. Protocol participants can generate this secure CRS for a one-time cost at the beginning of a protocol session using Canetti et al.'s UC NISC protocol [17] and only $\mathcal{G}_{\texttt{roRO}}$ [27].

In the ideal-world experiment, the simulator plays the role of $\mathcal{F}_{\texttt{CRS}}$, and can generate $\texttt{CRS}_s$ for each session $s$ with a secret trapdoor $\texttt{trap}_s$. Provers in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model prove via an OR-type $\Sigma$-protocol [21,23] that *either* they know a real witness to a statement, *or* they know the trapdoor $\texttt{trap}_s$ to $\texttt{CRS}_s$. This allows the simulator in the ideal-world experiment to "simulate" proofs of statements without witnesses using its extra power—the CRS trapdoor—which a real-world prover will never have.

In order to make the CRS statement compatible with the definition of $\Sigma$-protocols, it must be drawn from a *samplable-hard relation* $S$—that is, generating the CRS must be efficient, and, given any $\texttt{CRS}_s \in L_S$, it must be overwhelmingly difficult to generate a $\texttt{trap}_s$ such that $S(\texttt{CRS}_s, \texttt{trap}_s) = 1$ [27]. The relation $S$ must additionally be *$\Sigma$-friendly*: it must have a corresponding *efficient* $\Sigma$-protocol $\Sigma_S$. Proofs in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model are therefore well-specified

as OR-protocols $\Sigma_{R \vee S}$ over the relation $R \vee S$, where $R$ is the relation of the original $\Sigma$-protocol $\Sigma_R$, and $S$ is the samplable-hard relation underlying $\Sigma_S$. Formal definitions of the ideal functionality and context-friendly properties of $\mathcal{F}_{\texttt{CRS}}$ are given in the supplementary materials (A.2).

### 2.3 Adaptive Corruptions in the UC Model

Adaptive corruptions allow the (adversarial) environment $\mathcal{Z}$ in the UC model to obtain full internal views of honest parties *after* they have already participated in a cryptographic protocol. Briefly, adaptive corruptions in the UC framework are modeled as follows. When it wants to corrupt an extant party $P$, $\mathcal{Z}$ sends a message $(\textbf{Corrupt}, P)$ to the control function [14].[1]

In the real world, the control function passes the message $(\textbf{Corrupt}, P)$ to the traditional adversary $\mathscr{A}$, who passes the message to $P$. Upon receiving the corruption instruction, $P$ relinquishes all of its hidden internal tapes, including its input, output, work, and random tapes, to $\mathscr{A}$.[2]

In the ideal world, the control function passes the message $(\textbf{Corrupt}, P)$ to the ideal adversary $\mathcal{S}$, who must be able to provide a convincing internal view of $P$ to $\mathcal{Z}$. $\mathcal{S}$ does this by first querying the ideal functionality $\mathcal{F}$ (who has been running computations on "dummy party" $P$'s behalf) for any relevant information about $P$; it then runs some algorithms (in our case `SimRand`) to simulate $P$'s internal tapes. The control function routes all subsequent instructions for $P$ to $\mathscr{A}$ in the real world, and $\mathcal{S}$ in the ideal world. We call any protocol that satisfies the UC security definition (given in the next section) with adaptive corruptions *adaptive* UC, or aUC for short.

### 2.4 The NIZKPoK Ideal Functionality

Recall from the introduction that ideal functionalities $\mathcal{F}$ in the UC model operate on behalf of the honest "dummy parties" in the ideal-world experiment [14]. Upon receiving instructions from the ideal adversary $\mathcal{S}$ and setting up any necessary parameters, $\mathcal{F}_{\texttt{aNIZK}}$ proves statements for honest parties using the `SimSetup` and `SimProve` algorithms (which do not take witnesses as input) and verifies proofs using the `Extract` algorithm. If the algorithms from $\mathcal{S}$ do not function

[1] The control function is the entity in the UC experiment in charge of passing messages back and forth between all protocol participants. It is easiest to think of the control function as a modeling technique that prevents $\mathcal{Z}$ from sending messages that are outside the scope of the desired security experiment. For example, in the passive corruption model, the control function would not allow a message $(\textbf{Corrupt}, P)$ to go through after $P$ was initialized. In the adaptive setting, corruption messages are allowed at any point during the security experiment.

[2] Coins from the random tape in the UC model [14] are defined (without loss of generality) to be read-once and flipped on-the-fly, such that the calling TM can generate as much randomness as it wants (within its polynomial run-time bound). This implies that the corrupted party $P$ will only need to relinquish its random tape *history*, rather than a "full" tape (the length of which is undefined).

as $\mathcal{F}_{\texttt{aNIZK}}$ intends, for instance if `SimProve` produces proofs that do not verify or `Extract` outputs invalid witnesses, $\mathcal{F}_{\texttt{aNIZK}}$ outputs `Fail`. During its processing, $\mathcal{F}_{\texttt{aNIZK}}$ stores information about the proofs it has computed on each dummy party's behalf. $\mathcal{S}$ retrieves this information via $\mathcal{F}_{\texttt{aNIZK}}$'s **Corrupt** interface whenever an honest party is corrupted by the environment.

**Definition 1 (Adaptive NIZKPoK Ideal Functionality).** *The ideal functionality $\mathcal{F}_{\texttt{aNIZK}}$ of an* adaptive non-interactive zero-knowledge proof of knowledge (adaptive NIZKPoK) *for a particular session $s$ is defined as follows.*

**Setup:** *Initialize an empty list $C$ of corrupted parties. Upon receiving a request $(\texttt{Setup}, P)$ from a party $P = (\texttt{pid}, \texttt{sid})$, check whether $\texttt{sid} = s$ and $P \notin C$. If either check fails, output $\perp$. Otherwise, if this is the first time a request $(\texttt{Setup}, P)$ was received from an uncorrupted party with $\texttt{sid} = s$, do as follows: pass $(\texttt{Setup}, s)$ to the ideal adversary $\mathcal{S}$, receive and store $(\texttt{Algorithms}, \texttt{Setup}, \texttt{Prove}, \texttt{Verify}, \texttt{SimSetup}, \texttt{SimProve}, \texttt{Extract})$. Then run `SimSetup` and store $(\texttt{ppm}, z_s)$, where `ppm` are public parameters and $z_s$ is any auxiliary output of `SimSetup`. Otherwise, output $\perp$.*

**Prove:** *Upon receiving a request $(\texttt{Prove}, x, w)$ from a party $P = (\texttt{pid}, \texttt{sid})$, check that $\texttt{sid} = s$, $P \notin C$, and $R(x, w) = 1$. If any check fails, output $\perp$. Otherwise, set $w$ aside, compute $\pi$ according to the `SimProve` algorithm, and check that $\texttt{Verify}(x, \pi) = 1$. If it doesn't, output `Fail`. Otherwise, record the tuple $(\texttt{Proof}, P, x, w, \pi, z_s, z_\pi)$, where $z_\pi$ is any auxiliary output of the `SimProve` algorithm. Output $(\texttt{Proof}, P, x, \pi)$.*

**Verify:** *Upon receiving a request $(\texttt{Verify}, x, \pi)$ from a party $P = (\texttt{pid}, \texttt{sid})$, first check that $\texttt{sid} = s$ and $P \notin C$. If either check fails, output $\perp$. Otherwise if $\texttt{Verify}(x, \pi) = 0$, output $(\texttt{Verification}, P, x, \pi, 0)$. Otherwise if $(\texttt{Proof}, P, x, \pi)$ is already stored, output $(\texttt{Verification}, P, x, \pi, 1)$. Otherwise, compute $w$ according to the `Extract` algorithm. If $R(x, w) = 1$, output $(\texttt{Verification}, P, x, \pi, 1)$ for a successful extraction. Else if $R(x, w) = 0$, output `Fail`.*

**Corrupt:** *Upon receiving a request $(\textbf{Corrupt}, P)$ from $\mathcal{S}$, add $P$ to $C$ and return all of the stored tuples $(\texttt{Proof}, P, *)$, if they exist. Otherwise, output $\perp$.*

### 2.5 UC Security with Adaptive Corruptions

At a high level, a protocol $\Pi$ qualifies as adaptive UC (aUC) with respect to an ideal functionality $\mathcal{F}$ (i.e. $\Pi$ "aUC-emulates" $\mathcal{F}$) in the global ROM if for all efficient players in the security experiment, no adaptively-corrupting environment can distinguish the real-world experiment (with $\Pi$ and $\mathscr{A}$) from the ideal-world experiment (with $\mathcal{F}_{\texttt{aNIZK}}$ and $\mathcal{S}$).

In the UC with global subroutines (UCGS) model, $\Pi$ UC-emulates $\mathcal{F}$ *in the presence of a global subroutine* $\mathcal{G}$ if $\texttt{M}[\Pi, \mathcal{G}]$ UC-emulates $\texttt{M}[\mathcal{F}, \mathcal{G}]$, where `M` is the "shell" wrapper discussed in the introduction. Badertscher et al. show in Proposition 3.4 [1] that as long as the global subroutine is subroutine respecting and regular with respect to the challenge protocol, and that the challenge protocol is

subroutine respecting *except in its interactions with* $\mathcal{G}$ (i.e. it is $\mathcal{G}$-subroutine respecting), then $\mathtt{M}[\Pi, \mathcal{G}]$ (resp. $\mathtt{M}[\mathcal{F}, \mathcal{G}]$) are subroutine respecting *and behave just like* $\Pi$ *and* $\mathcal{G}$ *(resp.* $\mathcal{F}$ *and* $\mathcal{G}$*) would behave as individual entities.* We argued in Section 2.1 that both $\mathcal{G}_{\mathtt{rpoRO}}$ and $\mathcal{G}_{\mathtt{roRO}}$ qualify as global subroutines, and as $\mathcal{G}_{\mathtt{rpoRO}}$ and $\mathcal{G}_{\mathtt{roRO}}$ are the only global subroutines in our experiments, $\Pi$ and $\mathcal{F}_{\mathtt{aNIZK}}$ are $\mathcal{G}_{\mathtt{rpoRO}}$- and $\mathcal{G}_{\mathtt{roRO}}$-subroutine respecting. Therefore, without loss of generality, we consider our UC experiments "in the presence of global subroutines" $\mathcal{G}_{\mathtt{rpoRO}}$ and $\mathcal{G}_{\mathtt{roRO}}$—or in the $\mathcal{G}_{\mathtt{rpoRO}}$- and $\mathcal{G}_{\mathtt{roRO}}$-hybrid models for short—and make no further reference to $\mathtt{M}$ (for details, see Section 3 in Badertscher et al. [1]).

We review the (standard) UC-security definition with respect to a generic global subroutine $\mathcal{G}$, and instantiate the individual versions (i.e. with the generic global RO $\mathcal{G}_{\mathtt{RO}}$, $\mathcal{G}_{\mathtt{rpoRO}}$, $\mathcal{G}_{\mathtt{roRO}}$, and $\mathcal{F}_{\mathtt{CRS}}$) as needed throughout the paper.

**Definition 2 (aUC Protocols in the $\mathcal{G}$-hybrid Model).** *A protocol* $\Pi$ *with security parameter* $\lambda$ aUC-realizes the ideal functionality $\mathcal{F}$ with adaptive corruptions in the $\mathcal{G}$-hybrid model *if for all efficient* $\mathscr{A}$*, there exists an ideal adversary* $\mathcal{S}$ *efficient in expectation such that for all efficient environments* $\mathcal{Z}$ *that can issue adaptive corruptions,*

$$\mathsf{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathtt{aux}) \approx_c \mathsf{REAL}^{\mathcal{G},*}_{\Pi,\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}),$$

*where* $\mathcal{G}$ *is a global subroutine,* $\mathtt{aux}$ *is any auxiliary information provided to the environment, and* $*$ *represents any additional local functionality included in the real-world experiment.*

## 3 Adaptive $\Sigma$-protocols and Straight-line Compilers

In this section, we formalize the notions of adaptive $\Sigma$-protocols (Section 3.1) and adaptive straight-line compilers (Section 3.2).

### 3.1 Defining Adaptive $\Sigma$-protocols

Recall from the introduction that a $\Sigma$-protocol for a binary NP relation $R$ is a three-move public-coin protocol between a prover $P$ and a verifier $V$, after which $V$ is convinced that $P$ has a witness $w$ for some statement $x$ such that $R(x, w) = 1$. $P$ is assumed to be a *probabilistic* (polynomial-time) Turing Machine (TM)—that is, $P$ is assumed to have a random tape, from which it can sample polynomially-many random bits. In the traditional definitions of $\Sigma$-protocols [23,27], the contents of $P$'s random tape are not explicitly captured in the inputs and outputs of the $\Sigma$-protocol algorithms. However, $P$'s randomness is vital to the security experiment—it is the only piece of information hidden from the adversary (recall the adversary in the SHVZK game is allowed to query for proofs of statements using witnesses of its choosing), and in the adaptive corruption setting, the adversary will have access to this randomness after proofs have been generated. Therefore, rather than keeping the randomness necessary to compute proofs implicit in $P$'s random tape, we make it an explicit quantity, denoted $r$.

The basic three-move form of a $\Sigma$-protocol is generally described as an interactive "protocol template" $\tau$ [23,27]. The algorithmic version of the protocol template definition [27] consists of the following algorithms: $\tau$.`Setup`, $\tau$.`Commit`, $\tau$.`Challenge`, $\tau$.`Respond`, and $\tau$.`Decision`. We modify the $\tau$.`Setup` algorithm such that the public parameters contain a *randomness security parameter* $\lambda_r$, derived from the overall security parameter $\lambda$, that specifies the amount of randomness necessary to compute the first message `com`. In particular, we assume the randomness $r$ that is given as input to $\tau$.`Commit` is sampled uniformly at random from $\{0,1\}^{\lambda_r}$, representing a $\lambda_r$-length section of $P$'s random tape. The algorithms $\tau$.`Challenge`, $\tau$.`Respond`, and $\tau$.`Decision` are unchanged. We provide a full version of the definition in the supplementary materials (A.3).

**Definition 3 (Adaptive $\Sigma$-protocol Template).** *The* adaptive $\Sigma$-protocol template *for a relation $R$ modifies the standard $\Sigma$-protocol template (Definition 3 in A.3) as follows.*

- *The public parameters* `ppm` *returned by* $\tau$.`Setup` *include the randomness security parameter $\lambda_r$.*
- $\tau$.`Commit` *takes $r \leftarrow \{0,1\}^{\lambda_r}$ as an additional input.*

In order to maintain correctness and security for the prover and to adequately convince the verifier, $\Sigma$-protocols must have three properties: completeness, special honest-verifier zero-knowledge (SHVZK), and special soundness (SS). Lysyanskaya and Rosenbloom formalized a $\Sigma$-protocol as a tuple of algorithms, $\Sigma = (\texttt{Setup}, \texttt{Prove}, \texttt{Verify}, \texttt{SimSetup}, \texttt{SimProve}, \texttt{Extract})$, that capture the requirements of the three-move form as well as the correctness and security properties. In the adaptive corruption setting, the zero-knowledge simulator must additionally be able to produce a view of the prover's randomness that is consistent with the proofs generated by the `SimProve` algorithm. We therefore introduce a new algorithm—`SimRand`—which, given a statement, a proof, some auxiliary information produced by `SimSetup` and `SimProve`, and *the witness that was supposedly used to compute the proof*, outputs some simulated randomness. This algorithm captures the intuition that an adaptive $\Sigma$-protocol simulator must be able to generate convincing randomness for the prover's random tape *after the proof has already been generated* and the prover is corrupted by the adversary. We will show in Section 6 that several widely-used instantiations of $\Sigma$-protocols are adaptive according to this definition.

Finally, adaptive $\Sigma$-protocols require an adaptive version of SHVZK, in which the adversary should not be able to tell the difference between the outputs of $\Sigma$.`Prove` and $\Sigma$.`SimProve` *or* between the randomness $r$ of a real prover and the output of $\Sigma$.`SimRand`.

**Definition 4 (Adaptive $\Sigma$-protocol).** *An* adaptive $\Sigma$-protocol for a relation *$R$ based on adaptive protocol template $\tau$ (Definition 3) is a tuple of efficient procedures $\Sigma_{R,\tau} = (\texttt{Setup}, \texttt{Prove}, \texttt{Verify}, \texttt{SimSetup}, \texttt{SimProve}, \texttt{SimRand}, \texttt{Extract})$, defined as follows.*

- $\texttt{ppm} \leftarrow \texttt{Setup}(1^\lambda)$*: Given a security parameter* $1^\lambda$*, invoke* $\tau.\texttt{Setup}(1^\lambda)$ *to obtain the public parameters* ppm*.*
- $\pi \leftarrow \texttt{Prove}((\texttt{ppm}, x, w, r), (\texttt{ppm}, x))$*: Let the first (resp. second) argument to* Prove *be the input to* $P$ *(resp.* $V$*), where both parties get* ppm *and the statement* $x$*, but only* $P$ *gets witness* $w$ *and randomness* $r \leftarrow_\$ \{0,1\}^{\lambda_r}$*.* $P$ *and* $V$ *run* $\tau.\texttt{Commit}$*,* $\tau.\texttt{Challenge}$*, and* $\tau.\texttt{Respond}$*. Output* $\pi = (\texttt{com}, \texttt{chl}, \texttt{res})$*.*
- $\{0,1\} \leftarrow \texttt{Verify}(\texttt{ppm}, x, \pi)$*: Given a proof* $\pi$ *for statement* $x$*, parse* $\pi$ *as* $(\texttt{com}, \texttt{chl}, \texttt{res})$ *and output the result of running* $\tau.\texttt{Decision}$ *on input* $(x, \texttt{com}, \texttt{chl}, \texttt{res})$*.* Verify *must satisfy the* completeness property *from Definition 6 in A.5.*
- $(\texttt{ppm}, z_s) \leftarrow \texttt{SimSetup}(1^\lambda)$*: Generate* ppm *and a general simulation trapdoor* $z_s$*. Together,* SimSetup*,* SimProve*, and* SimRand *must satisfy the* adaptive special honest-verifier zero-knowledge property *from Definition 5.*
- $(\pi, z_p i) \leftarrow \texttt{SimProve}(\texttt{ppm}, z_s, x, \texttt{chl}, r)$ : *Given public parameters* ppm*, trapdoor* $z_s$*, statement* $x$*, challenge* chl*, and randomness* $r \leftarrow \{0,1\}^{\lambda_r}$*, produce a proof* $\pi = (\texttt{com}, \texttt{chl}, \texttt{res})$ *and proof trapdoor* $z_\pi$*.*
- $r \leftarrow \texttt{SimRand}(\texttt{ppm}, z_s, z_\pi, x, \pi, w)$ : *Given public parameters* ppm*, general trapdoor* $z_s$*, proof trapdoor* $z_\pi$*, proof* $\pi$ *for statement* $x$*, and a witness* $w$ *such that* $R(x, w) = 1$*, produce randomness (secret state)* $r$*.*
- $w \leftarrow \texttt{Extract}(\texttt{ppm}, x, \pi, \pi')$ : *Given two proofs* $\pi = (\texttt{com}, \texttt{chl}, \texttt{res})$ *and* $\pi' = (\texttt{com}, \texttt{chl}', \texttt{res}')$ *for a statement* $x$*, output a witness* $w$*.* Extract *must satisfy the* special soundness property *from Definition 7 in A.5.*

*For convenience and when the meaning is clear, we use* $\Sigma_R$ *to represent* $\Sigma_{R,\tau}$ *and omit* ppm *from the input of the algorithms.*

**Definition 5 (Adaptive Special Honest-Verifier Zero-Knowledge).** *A* $\Sigma$*-protocol* $\Sigma_R$ *for relation* $R$ *is statistical (resp. computational)* adaptive special honest-verifier zero-knowledge (adaptive SHVZK) *if there exist algorithms* SimSetup*,* SimProve*, and* SimRand *such that for any security parameter* $\lambda$*, any adversary (resp. any* PPT *adversary)* $\mathscr{A}$*, and a bit* $b \leftarrow_\$ \{0,1\}$*, there exists some negligible function* $\mathsf{negl}$ *such that* $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$*, where* $b'$ *is the result of running the game* $\mathsf{adSHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, b)$ *from Figure 1. We say* $\mathscr{A}$ *wins the* $\mathsf{adSHVZK}$ *game if* $\Pr[b' = b] > \frac{1}{2} + \mathsf{negl}(\lambda)$*.*

### 3.2 Defining Adaptive Straight-Line Compilers

Adaptive straight-line compilers are straight-line compilers [27] that preserve the adaptive security properties of the $\Sigma$-protocol being transformed. Recall from the introduction that a regular straight-line compiler SLC takes a $\Sigma$-protocol $\Sigma_R$ for a relation $R$ as input and produces a new proof system $\Pi_R^{\texttt{SLC}}$ that is a tuple of (non-interactive) *algorithms* $(\texttt{Setup}^H, \texttt{Prove}^H, \texttt{Verify}^H, \texttt{SimSetup}, \texttt{SimProve}, \texttt{Extract})$, where $H$ is any random oracle. In order to be considered a straight-line

$\mathsf{adSHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, 0)$: REAL

1 : $\mathtt{ppm} \leftarrow \Sigma_R.\mathtt{Setup}(1^\lambda)$
2 : $(\mathtt{Prove}, x, w, \mathtt{st}) \leftarrow \mathscr{A}(1^\lambda, \mathtt{ppm})$
3 : **if** $R(x, w) = 1$ :
4 : $\quad r \leftarrow_\$ \{0,1\}^{\lambda_r}$
5 : $\quad \mathtt{chl} \leftarrow \{0,1\}^\ell$
6 : $\quad \pi \leftarrow \Sigma_R.\mathtt{Prove}((x, w, r), (x, \mathtt{chl}))$
7 : **else** :
8 : $\quad \pi \leftarrow \perp$
9 : $b' \leftarrow \mathscr{A}(\mathtt{st}, \pi, r)$
10 : **return** $b'$

$\mathsf{adSHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, 1)$: IDEAL

1 : $(\mathtt{ppm}, z_s) \leftarrow \Sigma_R.\mathtt{SimSetup}(1^\lambda)$
2 : $(\mathtt{Prove}, x, w, \mathtt{st}) \leftarrow \mathscr{A}(1^\lambda, \mathtt{ppm})$
3 : **if** $R(x, w) = 1$ :
4 : $\quad \mathtt{chl} \leftarrow \{0,1\}^\ell$
5 : $\quad (\pi, z_\pi) \leftarrow \Sigma_R.\mathtt{SimProve}(z_s, x, \mathtt{chl})$
6 : $\quad r \leftarrow \Sigma_R.\mathtt{SimRand}(z_s, z_\pi, x, \pi, w)$
7 : **else** :
8 : $\quad \pi \leftarrow \perp$
9 : $b' \leftarrow \mathscr{A}(\mathtt{st}, \pi, r)$
10 : **return** $b'$

**Fig. 1.** Adaptive Special Honest-Verifier Zero-Knowledge (adSHVZK) Game.

compiler, $\Pi_R^{\mathtt{SLC}}$ must have the following properties: overwhelming completeness (i.e. a negligibly small completeness error is allowed), non-interactive *multiple* special honest-verifier zero-knowledge (NIM-SHVZK), and non-interactive special *simulation* soundness (NI-SSS). Our *adaptive* version of an SLC, denoted $\mathtt{aSLC}$, says that $\Pi_R^{\mathtt{aSLC}}$ must have *adaptive* NIM-SHVZK and *adaptive* NI-SSS properties—that is, NIM-SHVZK and NI-SSS must hold even when the adversary gets to compare the prover's true randomness with the output of $\mathtt{SimRand}$.

**Definition 6 (Adaptive Straight-Line Compiler).** *An algorithm* $\mathtt{SLC}$ *is an* adaptive straight-line compiler (adaptive SLC) *in the random-oracle model if given any adaptive* $\Sigma$*-protocol* $\Sigma_R$ *for relation* $R$ *(Definition 4) as input, it outputs a tuple of* algorithms $\Pi_R^{\mathtt{SLC}} = (\mathtt{Setup}^H, \mathtt{Prove}^H, \mathtt{Verify}^H, \mathtt{SimSetup}, \mathtt{SimProve}, \mathtt{SimRand}, \mathtt{Extract})$ *based on* $\Sigma_R$ *that satisfy the following properties: overwhelming completeness (Definition 7), adaptive non-interactive multiple special honest-verifier zero-knowledge (Definition 8), and adaptive non-interactive special simulation soundness (Definition 9).*

*We refer to* $\Pi_R^{\mathtt{aSLC}} \leftarrow \mathtt{aSLC}(\Sigma_R)$ *as an adaptive and non-interactive straight-line extractable (adaptive NISLE) proof system for* $R$*, and proofs generated by* $\Pi_R^{\mathtt{aSLC}}$ *as adaptive and non-interactive straight-line extractable zero-knowledge proofs of knowledge (adaptive NISLE ZKPoK).*

**Definition 7 (Overwhelming Completeness).** *An adaptive NISLE proof system* $\Pi_R^{\mathtt{aSLC}}$ *for relation* $R$ *in the random-oracle model has the* overwhelming completeness *property if for any security parameter* $\lambda$*, any* $(x, w) \in R$*, and any proof* $\pi \leftarrow \Pi_R^{\mathtt{aSLC}}.\mathtt{Prove}^H(x, w)$*,*

$$\Pr[\Pi_R^{\mathtt{aSLC}}.\mathtt{Verify}^H(x, \pi) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

The RO in the "real-world" experiment $H_f$ is parameterized by a function $f \leftarrow_\$ F$ selected from random function family $F$, while the RO in the "ideal-

world" experiment is a list oracle $H_L$ parameterized by an initially empty list $L$ that the challenger in the security experiment can program via the interface $\texttt{Prog}_L$. In order to maintain indistinguishability between the experiments and satisfy the (adaptive) NIM-SHVZK property, the ideal-world challenger must program the RO imperceptibly.[3] For a full description, see Figure 3 in A.5.

**Definition 8 (Adaptive Non-Interactive Multiple SHVZK).** *An adaptive NISLE proof system $\Pi_R^{\mathsf{aSLC}}$ for relation $R$ in the random-oracle model has the* adaptive non-interactive multiple special honest-verifier zero-knowledge (adNIM-SHVZK) *property if there exist algorithms $\Pi_R^{\mathsf{SLC}}.\texttt{SimSetup}$, $\Pi_R^{\mathsf{SLC}}.\texttt{SimProve}$, and $\Pi_R^{\mathsf{SLC}}.\texttt{SimRand}$ such that for any security parameter $\lambda$, any $\mathsf{PPT}$ adversary $\mathscr{A}$, and a bit $b \leftarrow_{\$} \{0,1\}$, there exists some negligible function $\mathsf{negl}$ such that $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where $b'$ is the result of running the game $\mathsf{adNIM\text{–}SHVZK}^{H_*,*}_{\mathscr{A},\Pi_R^{\mathsf{SLC}}}(1^\lambda, b)$ from Figure 2.*

| $\mathsf{adNIM\text{–}SHVZK}^{H_*,F}_{\mathscr{A},\Pi_R^{\mathsf{SLC}}}(1^\lambda,0)$: Real | $\mathsf{adNIM\text{–}SHVZK}^{H_*,\mathsf{Prog}}_{\mathscr{A},\Pi_R^{\mathsf{SLC}}}(1^\lambda,1)$: Ideal |
|---|---|
| $f \leftarrow_{\$} F$ | $L \leftarrow \bot$ |
| $\texttt{ppm} \leftarrow \Pi_R^{\mathsf{SLC}}.\texttt{Setup}^{H_f}(1^\lambda)$ | $(\texttt{ppm}, z_s) \leftarrow \Pi_R^{\mathsf{SLC}}.\texttt{SimSetup}^{\texttt{Prog}_L}(1^\lambda)$ |
| $\texttt{st} \leftarrow \mathscr{A}^{H_f}(1^\lambda, \texttt{ppm})$ | $\texttt{st} \leftarrow \mathscr{A}^{H_L}(1^\lambda, \texttt{ppm})$ |
| **while** $\texttt{st} \notin \{0,1\}$ : | **while** $\texttt{st} \notin \{0,1\}$ : |
| $(\texttt{Prove}, x, w, \texttt{st}) \leftarrow \mathscr{A}^{H_f}(\texttt{st})$ | $(\texttt{Prove}, x, w, \texttt{st}) \leftarrow \mathscr{A}^{H_L}(\texttt{st})$ |
| **if** $R(x,w) = 1$ : | **if** $R(x,w) = 1$ : |
| $r \leftarrow_{\$} \{0,1\}^{\lambda_r}$ | $(\pi, z_\pi) \leftarrow \Pi_R^{\mathsf{SLC}}.\texttt{SimProve}^{\texttt{Prog}_L}(z_s, x)$ |
| $\pi \leftarrow \Pi_R^{\mathsf{SLC}}.\texttt{Prove}^{H_f}(x, w, r)$ | $r \leftarrow \Pi_R^{\mathsf{SLC}}.\texttt{SimRand}^{\texttt{Prog}_L}(z_s, z_\pi, x, \pi, w)$ |
| **else** : | **else** : |
| $\pi, r \leftarrow \bot$ | $\pi, r \leftarrow \bot$ |
| $\texttt{st} \leftarrow \mathscr{A}^{H_f}(\texttt{st}, \pi, r)$ | $\texttt{st} \leftarrow \mathscr{A}^{H_L}(\texttt{st}, \pi, r)$ |
| **return** st | **return** st |

**Fig. 2.** Adaptive Non-Interactive Multiple SHVZK (adNIM-SHVZK) Game.

Similarly, we extend Lysyanskaya and Rosenbloom's definition of NI-SSS [27] by giving $\mathscr{A}$ not only the output of $\Pi_R^{\mathsf{aSLC}}.\texttt{SimProve}$, but of $\Pi_R^{\mathsf{aSLC}}.\texttt{SimRand}$ as well. The adaptive NI-SSS property says that soundness must hold even when $\mathscr{A}$ can see polynomially-many proofs from the simulator *and* can corrupt polynomially-many provers (i.e., see the contents of polynomially-many random tapes). Our work has the same limitation of Lysyanskaya and Rosenbloom [27]

[3] To satisfy NIM-SHVZK when $\Sigma_R.\texttt{SimProve}$ involves programming, the first message com will need entropy that is superlogarithmic in the security parameter [24,27].

in that we formalize the $\Pi_R^{\mathtt{aSLC}}.\mathtt{Extract}$ algorithm to work based on the adversary's queries to the random oracle, denoted $\mathcal{Q}_{\mathscr{A}}$. Extending the formalization of SLCs to include "key-based" extractors that leverage verifiable encryption schemes [5]—and determining whether or not such extractors can be used to obtain (adaptive) GUC NIZKPoK—is left for future work.

**Definition 9 (Adaptive Non-Interactive SSS).** *An adaptive NISLE proof system $\Pi_R^{\mathtt{SLC}}$ for relation $R$ in the random-oracle model has the* adaptive non-interactive special simulation sound (adNI-SSS) *property if there exists an algorithm $\Pi_R^{\mathtt{SLC}}.\mathtt{Extract}$ such that for any security parameter $\lambda$ and any* PPT *adversary $\mathscr{A}$,*

$$\Pr[\mathtt{Fail} \leftarrow \mathsf{adNI\text{–}SSS}^{H}_{\mathscr{A},\Pi_R^{\mathtt{SLC}}}(1^\lambda)] \leq \mathsf{negl}(\lambda),$$

*where $H$ is any random oracle and* adNI–SSS *is the game described in Figure 3.*

$\mathsf{adNI\text{–}SSS}^{H_L,\mathtt{Prog}_L}_{\mathscr{A},\Pi_R^{\mathtt{SLC}}}(1^\lambda)$

$L, \mathtt{pflist}, \mathtt{Response}, \mathtt{st} \leftarrow \perp;$
$\mathtt{ppm}, z \leftarrow \Pi_R^{\mathtt{SLC}}.\mathtt{SimSetup}^{\mathtt{Prog}_L}(1^\lambda)$
**while** $\mathscr{A}^{H_L}(1^\lambda, \mathtt{ppm}, \mathtt{st}) \neq \mathtt{halt}$ :
  $(\mathtt{Query}, \mathcal{Q}_{\mathscr{A}}, \mathtt{st}) \leftarrow \mathscr{A}^{H_L}(\mathtt{st})$
  **if** $\mathtt{Query} = (\mathtt{Prove}, x, w) \wedge R(x, w) = 1$ :
    $\pi, z_\pi \leftarrow \Pi_R^{\mathtt{SLC}}.\mathtt{SimProve}^{\mathtt{Prog}_L}(z, x)$
    $\mathtt{pflist}.\mathbf{append}(x, \pi)$
    $r \leftarrow \Pi_R^{\mathtt{SLC}}.\mathtt{SimRand}^{\mathtt{Prog}_L}(z, z_\pi, x, \pi, w)$
    $\mathtt{Response} \leftarrow (x, \pi, r)$
  **elseif** $\mathtt{Query} = (\mathtt{Challenge}, x, \pi)$
    **if** $\Pi_R^{\mathtt{SLC}}.\mathtt{Verify}^{H_L}(x, \pi) = 1 \wedge (x, \pi) \notin \mathtt{pflist}$ :
    $w \leftarrow \Pi_R^{\mathtt{SLC}}.\mathtt{Extract}(x, \pi, \mathcal{Q}_{\mathscr{A}})$
    **if** $R(x, w) = 0$ : **return** $\mathtt{Fail}$
  $\mathtt{st} \leftarrow \mathscr{A}^{H_L}(\mathtt{st}, \mathtt{Response})$
**return** $\mathtt{Success}$

**Fig. 3.** Adaptive Non-Interactive Special Simulation Soundness (adNI-SSS) Game.

## 4 Adaptive and Universally Composable NIZKPoK

In this section, we show that the adaptive non-interactive multiple special honest-verifier zero-knowledge (adNIM-SHVZK) and adaptive non-interactive special

simulation-soundness (adNI-SSS) properties afforded by any adaptive straight-line compiler (SLC) are *necessary* to achieve adaptive aUC NIZKPoK in any global ROM (Section 4.1), and that they are *sufficient* to transform any $\Sigma$-protocol into an adaptive aUC NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model (Section 4.2). For our construction in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model, we show how to adapt Lysyanskaya and Rosenbloom's OR-protocol compiler [27] to obtain adaptive aUC NIZKPoK without programming the global RO (Section 4.3).

### 4.1 Adaptive aUC NIZKPoK are adNIM-SHVZK and adNI-SSS

We begin by showing that any adaptive UC NIZKPoK must be adaptive NIM-SHVZK and adaptive NI-SSS. Because this result holds for any choice of global RO with the minimal `Query` functionality (as described in Section 2.1), we use the generic notation $\mathcal{G}_{\texttt{RO}}$ to represent the global RO.

**Theorem 1.** *Let $\Pi$ be a protocol that aUC-realizes $\mathcal{F}_{\texttt{aNIZK}}$ in the $\mathcal{G}_{\texttt{RO}}$-hybrid model (Definition 2 where $\mathcal{G}$ is replaced with $\mathcal{G}_{\texttt{RO}}$) with adaptive corruptions. Then $\Pi$ must be overwhelmingly complete (Definition 7), adaptive NIM-SHVZK (Definition 8), and adaptive NI-SSS (Definition 9).*

*Proof.* We proceed by cases and show that if $\Pi$ is not overwhelmingly complete and adNIM-SHVZK then it does not aUC-realize $\mathcal{F}_{\texttt{aNIZK}}$ in the $\mathcal{G}_{\texttt{RO}}$-hybrid model with adaptive corruptions, and similarly that if $\Pi$ is not adNI-SSS then it does not aUC-realize $\mathcal{F}_{\texttt{aNIZK}}$ in the same model.

Our first reduction $\mathscr{B}$ uses an adversary $\mathscr{A}$ that wins the adNIM-SHVZK game from Figure 2 with non-negligible advantage to distinguish between the real- and ideal-world aUC experiments with non-negligible advantage. $\mathscr{B}$ proceeds as follows. After passing the security and public parameters that it received from the aUC experiment to $\mathscr{A}$, $\mathscr{B}$ forwards $\mathscr{A}$'s oracle queries to $\mathcal{G}_{\texttt{RO}}$ and $\mathcal{G}_{\texttt{RO}}$'s responses back to $\mathscr{A}$.

`Prove` queries proceed as follows. Note that according to the definition of adNIM-SHVZK, any time $\mathscr{A}$ issues a `Prove` query, it is expecting not only a proof, *but also the proof's randomness*, in return. In order to accurately simulate $\mathscr{A}$'s view, $\mathscr{B}$ first issues the `Prove` query as-is to a new honest party in the aUC experiment.[4] Before returning the proof to $\mathscr{A}$, $\mathscr{B}$ then *corrupts* the prover to obtain the prover's internal tapes. Since the `Prove` operation was the first performed by the honest party, $\mathscr{B}$ simply takes the first $\lambda_r$ bits of the prover's random tape and returns these bits to $\mathscr{A}$ along with the proof.

If the aUC challenger is running the real-world experiment, the proof will be the result of running $\Pi$.`Prove` on the prover's witness and randomness. If the aUC challenger is running the ideal-world experiment, the proof will be the result of the ideal functionality $\mathcal{F}_{\texttt{aNIZK}}$ running $\Pi$.`SimProve`, and the randomness will have been generated by the simulator (ideal adversary) $\mathcal{S}$ using $\Pi$.`SimRand`.

[4] Recall that as part of the (adaptive G)UC experiment, the environment is permitted to spawn polynomially-many protocol participants, subject to polynomial run-time restrictions [14].

Therefore, $\mathscr{B}$ simulates $\mathscr{A}$'s view exactly, and succeeds in distinguishing the real- and ideal-world aUC experiments with the same probability that $\mathscr{A}$ distinguishes the real- from ideal-world adaptive NIM-SHVZK game.

The only other condition that might allow $\mathscr{B}$ to distinguish the two experiments are if the ideal functionality $\mathcal{F}_{\texttt{aNIZK}}$ in the ideal-world aUC experiment outputs `Fail` due to a completeness error. This condition occurs with negligible probability due to overwhelming completeness.

The second reduction uses two black-box algorithms: $\mathscr{A}$ that wins the adaptive NI-SSS game from Figure 3, and $\mathscr{A}'$ that wins the regular NI-SS game from Definition 21 in section A.5 of the supplementary materials (in which the adversary does not have access to simulated proofs), with non-negligible advantage. $\mathscr{B}$ answers $\mathscr{A}$'s queries the same as in the previous reduction, by forwarding all of $\mathscr{A}$'s oracle queries to $\mathcal{G}_{\texttt{RO}}$ and `Prove` queries to the aUC challenger, then making adaptive corruptions to obtain the prover's randomness. $\mathscr{B}$ forwards $\mathscr{A}'$'s queries to $\mathcal{G}_{\texttt{RO}}$ (note $\mathscr{A}'$ does not make `Prove` queries, but can run $\Pi$.`Prove` itself).

$\mathscr{B}$ proceeds the same as before, forwarding all of $\mathscr{A}$'s oracle queries to $\mathcal{G}_{\texttt{RO}}$ and `Prove` queries to the aUC challenger, then making adaptive corruptions to obtain the prover's randomness. Whenever $\mathscr{A}$ (resp. $\mathscr{A}'$) outputs a proof $\pi$ for a statement $x$ such that $\Pi.\texttt{Verify}(x,\pi) = 1$, $\mathscr{B}$ gathers $\mathscr{A}$'s (resp. $\mathscr{A}'$'s) oracle queries $\mathcal{Q}_{\mathscr{A}}$ (resp. $\mathcal{Q}_{\mathscr{A}'}$) and runs $w \leftarrow \Pi.\texttt{Extract}(x,\pi,\mathcal{Q}_{\mathscr{A}(\text{resp.}\mathscr{A}')})$. If $w$ is such that $R(x,w) = 0$, $\mathscr{B}$ invokes a new honest party and sends it the instruction $(\texttt{Verify}, x, \pi)$. If the aUC challenger is running the ideal-world experiment, then $\mathscr{B}$ has simulated $\mathscr{A}$'s expected view, and the honest (dummy) party will invoke $\mathcal{F}_{\texttt{aNIZK}}$ on $\mathscr{A}$'s proof and output `Fail`, causing $\mathscr{B}$ to output "ideal." If the aUC challenger is running the real-world experiment, then on input a proof $\pi$ from $\mathscr{A}'$, the real-world honest party will output $\Pi.\texttt{Verify}(x,\pi) = 1$, causing $\mathscr{B}$ to output "real." $\mathscr{B}$ therefore distinguishes the ideal- from real-world aUC experiments with the same probability as $\mathscr{A}$ and $\mathscr{A}'$, respectively. □

### 4.2 Adaptive aUC NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid Model

We prove in this section that running any adaptive $\Sigma$-protocol $\Sigma_R$ for relation $R$ through any adaptive straight-line compiler (adaptive SLC) results in adaptive UC NIZKPoK for relation $R$ in the (programmable) $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model.

**Theorem 2.** *Let $\Sigma_R$ be any adaptive $\Sigma$-protocol for relation $R$ (Definition 4), $\mathcal{G}_{\texttt{rpoRO}}$ be the restricted programmable observable global random oracle (Definition 1 in A.2) and* `SLC` *be any adaptive straight-line compiler (Definition 6). Then the NISLE proof system $\Pi_R^{\texttt{SLC}} \leftarrow \texttt{SLC}(\Sigma_R)$ aUC-realizes $\mathcal{F}_{\texttt{aNIZK}}$ in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model (Definition 2 where $\mathcal{G}$ is replaced with $\mathcal{G}_{\texttt{rpoRO}}$).*

*Proof.* We begin with the construction of the ideal adversary (simulator) $\mathcal{S}$.

*Construction of the Ideal Adversary.* When $\mathcal{S}$ receives $(\texttt{Setup}, s)$ from $\mathcal{F}_{\texttt{aNIZK}}$, $\mathcal{S}$ returns the tuple of algorithms $\Pi_R^{\texttt{SLC}}$. When corrupted parties issue `IsProgrammed` queries, $\mathcal{S}$ returns `false`. When $\mathcal{Z}$ issues a query $(\textbf{Corrupt}, P)$ for a party $P = (\texttt{pid}, \texttt{sid})$, $\mathcal{S}$ sends $(\textbf{Corrupt}, P)$ to $\mathcal{F}_{\texttt{aNIZK}}$ to obtain the stored tuples

$(\texttt{Proof}, P, x_1, w_1, \pi_1, z_1), \ldots, (\texttt{Proof}, P, x_q, w_q, \pi_q, z_q)$ corresponding to each query $(\texttt{Prove}, P, x_i, w_i)$ that $\mathcal{Z}$ issued to $P$. (who forwarded them to $\mathcal{F}_{\texttt{aNIZK}}$). For each tuple, $\mathcal{S}$ interprets $z_i = (z_s, z_{\pi_i})$ as the auxiliary outputs of $\Pi_R^{\texttt{SLC}}.\texttt{SimSetup}$ and $\Pi_R^{\texttt{SLC}}.\texttt{SimProve}$, respectively. $\mathcal{S}$ then runs $r_i \leftarrow \Pi_R^{\texttt{SLC}}.\texttt{SimRand}(z_s, z_{\pi_i}, x, \pi, w)$ to obtain simulated randomness $r_i$. To reconstruct the prover's random tape $R$, $\mathcal{S}$ concatenates $R = r_1 || \ldots || r_q$ and returns $R$ to $\mathcal{Z}$. Otherwise, $\mathcal{S}$ forwards all communications between $\mathcal{Z}$ and the protocol.

We proceed by creating a hybrid argument that starts in the real-world experiment and replaces each piece of the real-world protocol $\Pi_R^{\texttt{SLC}}$ with the functionality of $\mathcal{F}_{\texttt{aNIZK}}$ and $\mathcal{S}$.

**Hybrid 1.** First, we replace all of the environment's and adversary's connections to the real-world protocol participants with the "challenger" of our reduction, $\mathcal{C}$. This difference is syntactic, so Hybrid 1 is identical to the real-world experiment.

**Hybrid 2.** In the second hybrid, we replace $\mathcal{C}$'s real-world `Prove` functionality with the **Prove** interface of $\mathcal{F}_{\texttt{aNIZK}}$ and random tape simulation of $\mathcal{S}$, and show the environment's views are indistinguishable between Hybrids 1 and 2 as long as $\Sigma_R$ is adaptive and non-interactive multiple special honest-verifier zero-knowledge (adNIM-SHVZK). First, we specify $\mathcal{C}$ to simulate $\mathcal{G}_{\texttt{rpoRO}}$ according to its specification (noting that $\mathcal{C}$ can "program" its simulation) and return `false` to all of the corrupted parties' `IsProgrammed` queries. As long as $\Pi_R^{\texttt{SLC}}.\texttt{SimProve}$ produces valid proofs for statements $x \in L_R$ with overwhelming probability (which follows from overwhelming completeness), the environment's view of $\mathcal{G}_{\texttt{rpoRO}}$ remains statistically indistinguishable between the hybrids (which follows from the restriction of the `IsProgrammed` interface), $\mathcal{Z}$ is forced to distinguish the hybrids based on the only other difference—the proofs $\pi_i$ and randomness $r_i$.

We bound $\mathcal{Z}$'s probability of distinguishing the hybrids based on the proofs and randomness by constructing a reduction to the adNIM-SHVZK property as follows. Whenever $\mathcal{Z}$ issues a query $(\texttt{Prove}, P, x_i, w_i)$ to $\mathcal{C}$, $\mathcal{C}$ forwards the query to the adNIM-SHVZK challenger from Figure 2, who returns $(\pi_i, r_i)$. $\mathcal{C}$ forwards $\pi_i$ to $\mathcal{Z}$ and stores $(P, r_i)$ for later. If $\mathcal{Z}$ issues a query $(\textbf{Corrupt}, P)$, $\mathcal{C}$ retrieves all of the tuples $(P, r_i)$ and sets the random tape $R$ to be the concatenation of all $r_i$ in the order they were stored. It then returns $R$ to $\mathcal{C}$. For $\mathcal{Z}$'s queries $(\texttt{Verify}, x, \pi)$, $\mathcal{C}$ returns the result of running $\Pi_R^{\texttt{SLC}}.\texttt{Verify}(x, \pi)$. $\mathcal{C}$ outputs whatever $\mathcal{Z}$ outputs.

Note that if the adNIM-SHVZK challenger is running the 0-bit experiment (using $\Pi_R^{\texttt{SLC}}.\texttt{Prove}$ and the prover's randomness), $\mathcal{C}$ simulates $\mathcal{Z}$'s exact view of the experiment in Hybrid 1. Else if the adNIM-SHVZK challenger is running the 1-bit experiment (using $\Pi_R^{\texttt{SLC}}.\texttt{SimProve}$ and $\Pi_R^{\texttt{SLC}}.\texttt{SimRand}$), $\mathcal{C}$ simulates $\mathcal{Z}$'s view of Hybrid 2. Therefore, $\mathcal{C}$ succeeds at winning the adNIM-SHVZK game with the same probability that $\mathcal{Z}$ can distinguish the hybrids, proving Hybrid 1 is computationally indistinguishable to Hybrid 2.

**Hybrid 3.** In the third hybrid, we replace $\mathcal{C}$'s `Verify` functionality with the **Verify** functionality of $\mathcal{F}_{\texttt{aNIZK}}$, and show the environment's views are indistin-

guishable between Hybrids 2 and 3 as long as $\Pi_R^{\mathtt{SLC}}$ is adaptive non-interactive special simulation-sound (adNI-SSS). We construct a reduction that uses an environment $\mathcal{Z}$ that can distinguish Hybrids 2 and 3 with non-negligible advantage to win the adNI-SSS game from Figure 3 as follows. First, note the only difference in output between Hybrids 2 and 3 is that the Hybrid 3 experiment can output `Fail`, while the Hybrid 2 experiment never does—in particular, Hybrid 3 will output `Fail` if $\mathcal{Z}$ succeeds in producing a valid, non-simulated proof that causes $\Pi_R^{\mathtt{SLC}}.\mathtt{Extract}$ to output `Fail`. For $\mathcal{Z}$'s `Prove` queries, the reduction acts according to Hybrid 2, this time forwarding the queries to the adNI-SSS challenger, returning the proofs, and saving random bits in case $\mathcal{Z}$ issues a corruption query on the prover. When $\mathcal{Z}$ issues a query $(\mathtt{Verify}, x, \pi)$ for a proof $\pi$ that $\mathcal{C}$ did not send to $\mathcal{Z}$, $\mathcal{C}$ sends $(\mathtt{Challenge}, x, \pi, \mathcal{Q}_{P^*})$ to the adNI-SSS challenger. Since both the adNI-SSS challenger and $\mathcal{F}_{\mathtt{aNIZK}}$ use the $\Pi_R^{\mathtt{SLC}}.\mathtt{Extract}$ algorithm and fail under the same conditions, $\mathcal{C}$ succeeds in winning the adNI-SSS game with the same probability that $\mathcal{Z}$ distinguishes Hybrids 2 and 3.

**Hybrid 4.** The final hybrid replaces $\mathcal{C}$ with $\mathcal{F}_{\mathtt{aNIZK}}$ and $\mathcal{S}$. Note that since $\mathcal{C}$ now runs all of $\mathcal{F}_{\mathtt{aNIZK}}$ and $\mathcal{S}$'s procedures, this is again a syntactic difference, and Hybrid 3 is identical to Hybrid 4. □

### 4.3 Adaptive aUC NIZKPoK in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid Model

Similarly, any adaptive SLC in conjunction with the OR-protocol construction given by Lysyanskaya and Rosenbloom [27] and reviewed in Section 2.2 is sufficient to create adaptive UC NIZKPoK in the $\mathcal{G}_{\mathtt{rpoRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model. We update the algorithms $\Pi_{\mathtt{R}\vee\mathtt{S}}^{\mathtt{SLC}}$ of the OR-protocol construction below to create an *adaptive* NISLE proof system, denoted $\Pi_{\mathtt{R}\vee\mathtt{S}}^{\mathtt{aSLC}}$.

**Definition 10 (Adaptive OR-protocol Compiler).** *The* adaptive OR-protocol compiler `guc` *modifies Lysyanskaya and Rosenbloom's candidate compiler (Definition 13 in [27]) for the adaptive setting as follows.*

- *Our compiler uses any* adaptive *straight-line compiler* `aSLC` *(Definition 6).*
- *As in the definition of adaptive $\Sigma$-protocols (Definition 4), $\Pi_{\mathtt{R}\vee\mathtt{S}}^{\mathtt{aSLC}}.\mathtt{Prove}$ now takes as input sufficient randomness $r \leftarrow_{\$} \{0,1\}^{\lambda_r}$ for $\lambda_r = \lambda_{r_R} + \lambda_{r_S}$, where $\lambda_{r_R}$ is the randomness security parameter of $\Sigma_R$ and $\lambda_{r_S}$ is the randomness security parameter of $\Sigma_S$.*[5]
- *The algorithm $\Pi_{\mathtt{R}\vee\mathtt{S}}^{\mathtt{aSLC}}.\mathtt{SimSetup}$ produces an additional list* `prand` *to store the randomness used by the proof simulator $\Pi_{\mathtt{R}\vee\mathtt{S}}^{\mathtt{aSLC}}.\mathtt{SimProve}$.*
- *The algorithm $\Pi_{\mathtt{R}\vee\mathtt{S}}^{\mathtt{aSLC}}.\mathtt{SimProve}$ updates the list* `prand` *with the random bits $r$ used to compute the "simulated" OR-proof $\Phi$ (which is simply an iteration of $\Pi_{\mathtt{R}\vee\mathtt{S}}^{\mathtt{aSLC}}.\mathtt{Prove}$ using the trapdoor to the CRS rather than a "real" witness), and returns* `prand` *as additional output.*

[5] For an in-depth treatment of OR-protocols in the adaptive setting, see 25 in the supplementary materials.

– *The tuple of algorithms* $\Pi^{\mathtt{aSLC}}_{\mathtt{R}\vee\mathtt{S}}$ *contains an additional algorithm* `SimRand`*, which, on input* $(\mathtt{ppm}, \mathtt{prand}, X, \Phi, W)$*, looks up the randomness* $r$ *used to compute* $\Phi$ *in the list* `prand` *and returns it.*

**Theorem 3.** *Let* $\Sigma_R$ *be any adaptive* $\Sigma$*-protocol for relation* $R$ *(Definition 4),* $\mathcal{G}_{\mathtt{roRO}}$ *be the restricted observable global random oracle (Definition 2 in A.2)* $\Sigma_S$ *be an efficient* $\Sigma$*-protocol for samplable-hard relation* $S$ *(Definition 11 in A.8)* $\mathcal{F}_{\mathtt{CRS}}$ *be the ideal CRS functionality (Definition 10 in A.8),* `aSLC` *be any adaptive straight-line compiler (Definition 6), and* `guc` *be the adaptive OR-protocol compiler (Definition 10). Then* $\Pi^{\mathtt{aSLC}}_{\mathtt{R}\vee\mathtt{S}} \leftarrow \mathtt{guc}(\Sigma_R, \mathtt{aSLC})$ *aUC-realizes* $\mathcal{F}_{\mathtt{aNIZK}}$ *in the* $\mathcal{G}_{\mathtt{roRO}}$*-*$\mathcal{F}_{\mathtt{CRS}}$*-hybrid model (Definition 2 where* $\mathcal{G}$ *is replaced with* $\mathcal{G}_{\mathtt{roRO}}$ *and* $*$ *is replaced with* $\mathcal{F}_{\mathtt{CRS}}$*).*

*Proof.* The construction of the ideal adversary (simulator) $\mathcal{S}$ is the same as in the proof of Theorem 2, except it returns $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}$ to $\mathcal{F}_{\mathtt{aNIZK}}$ rather than $\Pi^{\mathtt{SLC}}_{R}$, and there are no `IsProgrammed` queries. (Note that the simulation and proof trapdoors, $z_s$ and $z_\pi$ respectively, are simply the simulator's CRS list `simcrs` and proof randomness list `prand`, respectively.)

We again create a hybrid reduction that starts in the real-world experiment and replaces each piece of the real-world adaptive NISLE OR-protocol $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}$ with the functionality of $\mathcal{F}_{\mathtt{aNIZK}}$ and $\mathcal{S}$.

**Hybrid 1.** Identical to Hybrid 1 in the proof of Theorem 2.

**Hybrid 2.** In the second hybrid, we replace $\mathcal{C}$'s real-world `Prove` functionality with the original straight-line compiled OR-protocol simulator algorithms $\Pi^{\mathtt{aSLC}}_{\mathtt{R}\vee\mathtt{S}}$.`SimSetup`, $\Pi^{\mathtt{aSLC}}_{\mathtt{R}\vee\mathtt{S}}$.`SimProve`, and $\Pi^{\mathtt{aSLC}}_{\mathtt{R}\vee\mathtt{S}}$.`SimRand`. This step allows us to avoid giving $\mathcal{C}$ control over the CRS trapdoors for now, such that we are able to show in the next hybrid argument that $\mathcal{C}$ can use a proof-forging environment to break either the adNI-SSS property or the hardness property of the samplable-hard relation $S$ (i.e. the reduction produces a CRS trapdoor). The proof that Hybrid 2 is indistinguishable from Hybrid 1 proceeds identically to the proof under Hybrid 2 in Theorem 2 above, modulo the `IsProgrammed` interface.

**Hybrid 3.** In the third hybrid, we replace $\mathcal{C}$'s `Verify` functionality with the **Verify** functionality of $\mathcal{F}_{\mathtt{aNIZK}}$, and show the environment's views are indistinguishable between Hybrids 2 and 3 as long as $\Pi^{\mathtt{SLC}}_{R}$ is adaptive non-interactive special simulation-sound (adaptive NI-SSS) *and* $S$ is a hard relation (i.e. given $\mathtt{CRS}_s$, the probability of finding $\mathtt{trap}_s$ such that $S(\mathtt{CRS}_s, \mathtt{trap}_s) = 1$ is negligible). This piece of the proof proceeds identically to the proof of indistinguishability between the second and third hybrids in Lysyanskaya and Rosenbloom's proof of Theorem 3 [27], where $\mathcal{Z}$'s adaptive corruption queries are handled identically to those in Hybrid 2 (i.e. $\mathcal{S}$ patches together the prover's random tape by concatenating the outputs of $\Pi^{\mathtt{aSLC}}_{\mathtt{R}\vee\mathtt{S}}$.`SimRand` returned by the adaptive NI-SSS challenger). To recap briefly, $\mathcal{C}$ plays the role of the adversary in the adaptive NI-SSS game, forwarding $\mathcal{Z}$'s queries to its challenger. When $\mathcal{Z}$ produces a valid proof that causes $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}$.`Extract` to output `Fail` (which happens with non-negligible probability by assumption, as the failure condition is the only difference between

the hybrids), either $R$ is not satisfied and $\mathcal{C}$ wins the adaptive NI-SSS game, or $S$ is not satisfied and $\mathcal{C}$ breaks the hardness property of $S$.

**Hybrid 4.** In the penultimate hybrid, $\mathcal{C}$ uses $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{SimSetup}$, $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{SimProve}$, and $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{SimRand}$. Recall that $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{SimProve}$ is essentially $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Prove}$, except that $\mathcal{C}$ generates and stores pairs $(\mathtt{CRS}_s, \mathtt{trap}_s)$ for each protocol session $s$ in the list $\mathtt{simcrs}$, and uses the witness $\mathtt{trap}_s$ as input to $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Prove}$ rather than a "real" witness $w$. Moreover, the $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{SimRand}$ algorithm is functionally identical to the "real-world" corruption process, in which the prover simply hands over the random tape used to compute its proofs. Similar to the proof that Hybrid 1 is indistinguishable from Hybrid 2, we argue that Hybrids 3 and 4 are indistinguishable as long as $\Pi^{\mathtt{aguc}}_{\mathtt{R}\vee\mathtt{S}}$ is adaptive NIM-SHVZK. The proof is the same with one caveat: we must ensure that the *only* way $\mathcal{Z}$ can distinguish the hybrids is based on the differences between the proof simulations—that is, we must rule out the possibility that $\mathcal{Z}$ can learn something new from interacting with both the simulated proofs *and* the extractor. Following LR, we note that since $\mathtt{Extract}$ functions solely using $\mathcal{Z}$'s (and $\mathscr{A}$'s) queries to $\mathcal{G}_{\mathtt{roRO}}$, $\mathcal{Z}$ cannot possibly learn anything from interacting with it, regardless of the proof simulation algorithms.

**Hybrid 5.** Identical to Hybrid 4 in Theorem 2. □

## 5 Constructions via the Randomized Fischlin Transform

In this section, we update the randomized Fischlin transform [24,26] for the adaptive setting (Section 5.1), and prove that it is an adaptive SLC (Section 5.2). We then use the adaptive randomized Fischlin transform to aUC-realize $\mathcal{F}_{\mathtt{aNIZK}}$ efficiently[6] in both the $\mathcal{G}_{\mathtt{rpoRO}}$- and $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid models (Sections 5.3).

### 5.1 The Adaptive Randomized Fischlin Transform

The standard randomized Fischlin transform [24,26] is a straight-line compiler $\mathtt{rFis}$ [27] that transforms any $\Sigma$-protocol with certain general properties into a non-interactive, straight-line extractable (NISLE) proof system $\Pi^{\mathtt{rFis}}_R$ in the ROM. The prover in the randomized Fischlin transform essentially rewinds itself, computing proofs on repeated commitments (but different challenges) until it is guaranteed with overwhelmingly probability that there are at least two transcripts queried to the RO with the same commitment but different challenges. The algorithm $\Pi^{\mathtt{rFis}}_R.\mathtt{Extract}$ takes these proofs as input (via the adversary's oracle query history $\mathcal{Q}_{\mathscr{A}}$), and can therefore extract a witness for valid, adversarially-created proofs *without any further interaction with the prover*.

We argue in this section that our *adaptive* randomized Fischlin transform $\mathtt{aFis}$, which is almost the same as $\mathtt{rFis}$ but contains a $\mathtt{SimRand}$ functionality,

[6] For a discussion about the precise efficiency of the randomized Fischlin transform, see Section A.10.

preserves the adaptive security property of the underlying $\Sigma$-protocol: we will show that as long as $\Sigma_R$ is adaptive (and conforms to standards of `rFis`), `aFis` is an adaptive SLC. The $\Pi_R^{\texttt{aFis}}$.`SimRand` functionality, which is in charge of producing a convincing version $Q$ of the prover's randomness, works as follows.

In order to reconstruct the random "first-message" section of $Q$ (i.e. the section used to run $\tau$.`Commit`), the $\Pi_R^{\texttt{aFis}}$.`SimRand` algorithm runs $\Sigma_{R,\tau}$.`SimRand` using auxiliary output from $\Pi_R^{\texttt{aFis}}$.`SimProve`. To simulate the random "challenge-selection" section of $Q$ (i.e. the section used to generate the randomly-selected challenges of the randomized Fischlin transform), $\Pi_R^{\texttt{aFis}}$.`SimRand` concatenates the random coins corresponding to all of the challenges sampled by $\Pi_R^{\texttt{aFis}}$.`SimProve`. In order for (not only the challenges in the proof tuple, but) *all* of the challenges sampled by the simulator to agree with the output of the random oracle, $\Pi_R^{\texttt{aFis}}$.`SimRand` programs the random oracle on all of the proof-output tuples it generated while executing $\Pi_R^{\texttt{aFis}}$.`SimProve`. (Recall that the simulator in the randomized Fischlin transform generates challenges and outputs of the oracle simultaneously, finds the first challenge in lexicographic order mapping to the least output, and programs the oracle on that index—we simply extend this practice to the randomness simulator.) The amount of programming is still polynomial in the security parameter and all of the challenges are identically distributed, so we are able to show that the adaptive simulator algorithms $\Pi_R^{\texttt{aFis}}$.`SimProve` and $\Pi_R^{\texttt{aFis}}$.`SimRand` are indistinguishable from $\Pi_R^{\texttt{aFis}}$.`Prove` on real randomness, as long as the underlying $\Sigma$-protocol used as input to the transform is adaptive special honest-verifier zero-knowledge (adSHVZK).

**Definition 11 (Adaptive Randomized Fischlin Transform).** *The* adaptive randomized Fischlin transform `aFis` *is an algorithm that takes a $\Sigma$-protocol $\Sigma_{R,\tau}$ based on protocol template $\tau$ for relation $R$ (Definition 4) with the required properties for* `rFis` *(Definition 29 in the supplementary materials) as input, and produces a tuple of algorithms $\Pi_R^{\texttt{aFis}} = (\texttt{Setup}^H, \texttt{Prove}^H, \texttt{Verify}^H, \texttt{SimSetup}, \texttt{SimProve}, \texttt{SimRand}, \texttt{Extract})$, where $H : \{0,1\}^* \to \{0,1\}^b$ is any random oracle. We highlight important modifications to* `rFis` *below; a full specification of the transform can be found in the supplementary materials (A.9).*

- $\Pi_R^{\texttt{aFis}}$.`Prove` *takes randomness $Q \leftarrow_{\$} \{0,1\}^{\lambda_Q}$ as additional input.* $\Pi_R^{\texttt{aFis}}$.`SimProve` *additionally returns $Z_\Phi = (z_{\pi_1}, \ldots, z_{\pi_n}, Q')$ for each proof $\Phi = \pi_1, \ldots, \pi_n$, where $Q'$ is a copy of the randomness used by* `SimProve` *during its computation.*
- *We specify the new algorithm* $\Pi_R^{\texttt{aFis}}$.`SimRand` *as follows.* $\Pi_R^{\texttt{aFis}}$.`SimRand` *takes as input public parameters* `ppm`*, simulation trapdoor $z_s$, proof trapdoor $Z_\Phi = (z_{\pi_1}, \ldots, z_{\pi_n}, Q')$, a proof $\Phi = (\pi_1, \ldots, \pi_n)$, and a witness $w$. It produces the prover's random tape $Q$ as follows. First, generate the simulated randomness of the first messages by running $\Sigma_{R,\tau}$.*`SimRand`*$(z_s, z_{\pi_i}, x, \pi_i, w, r_i')$ for each $\pi_i$, where each $r_i'$ is the same section of $Q'$ used by* $\Pi_R^{\texttt{aFis}}$.`SimProve` *to produce $\pi_i$. Next, generate the randomness of the challenge selection process as follows. For each commitment* $\texttt{com}_i$*, use the appropriate segment of $Q'$ to reconstruct the map $\mu : \{0,1\}^t \to \{0,1\}^b$ generated by* $\Pi_R^{\texttt{aFis}}$.`SimProve` *that maps $2^t$ $t$-bit challenges to $b$-bit potential outputs of $H$. For each challenge* $\texttt{chl}_j$

*in the list of* $2^t$ *challenges, compute the response* $\mathtt{res}_j \leftarrow \tau.\mathtt{Respond}(x, w, \mathtt{com}_i, \mathtt{chl}_j)$ *and program the output of* $H$ *on input* $(x, \overline{\mathtt{com}}, i, \mathtt{chl}_j, \mathtt{res}_j)$ *to be* $\mu_i(\mathtt{chl}_j)$*. Finally, concatenate* $\mathtt{chl}_j$ *to* $Q$*. Once the process as been completed for all commitments and challenges, return* $Q$*.*

### 5.2 Adaptive Randomized Fischlin is an Adaptive SLC

We now prove that the adaptive randomized Fischlin transform `aFis` qualifies as an adaptive SLC, and can therefore efficiently bootstrap adaptive $\Sigma$-protocols into adaptive UC NIZKPoK in the global ROM(s).

**Theorem 4.** *Let* $\Sigma_{R,\tau}$ *be an adaptive* $\Sigma$*-protocol based on protocol template* $\tau$ *for relation* $R$ *(Definition 4) with the required properties for* `rFis` *(Definition 29 in A.9), and* $H : \{0,1\}^* \to \{0,1\}^b$ *be any random oracle. Then the adaptive randomized Fischlin transform* `aFis` *(Definition 30 in A.9) is an adaptive straight-line compiler for* $\Sigma_{R,\tau}$ *(Definition 6) in the random-oracle model.*

*Proof Sketch.* Recall that an adaptive straight-line compiler according to our definition must create protocols that are overwhelmingly complete, adaptive non-interactive multiple special honest-verifier zero-knowledge (adNIM-SHVZK), and adaptive non-interactive special simulation-sound (adNI-SSS). First, note that since the specification of $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ does not functionally change between `rFis` and `aFis` (as our explicit treatment of the prover's randomness is syntactic), `aFis` is as complete as `rFis` [24,26]. We proceed by contrapositive to show that if $\Pi_R^{\mathtt{aFis}}$ is not additionally adNIM-SHVZK and adNI-SSS, then $\Sigma_R$ cannot be adaptive special honest-verifier zero-knowledge (adSHVZK). The full version of this proof can be found in the supplementary materials (A.11).

We begin by constructing a reduction $\mathscr{B}$ that uses an algorithm $\mathscr{A}$ that can win the adNIM-SHVZK game (Definition 8) parameterized over $\Pi_R^{\mathtt{aFis}}$ with non-negligible advantage as a black-box in order to win the adSHVZK game (Definition 5) parameterized over $\Sigma_R$ with non-negligible advantage. $\mathscr{B}$ proceeds as follows. When it obtains `ppm` from the adSHVZK challenger, $\mathscr{B}$ passes `ppm` to $\mathscr{A}$. Note that the adSHVZK challenger is expecting exactly one `Prove` query (i.e. it is not *multi*-adaptive), so we modify $\mathscr{A}$ to distinguish two hybrids $i$ and $i+1$, where in the $i^{th}$ hybrid, the first $i$ proofs and random coins are according to $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ using randomness $r \in \{0,1\}^{\lambda_r}$, and the $i+1^{st}$ onward are according to $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimRand}$.

For the first $i$ queries $(\mathtt{Prove}, x, w)$ from $\mathscr{A}$, $\mathscr{B}$ samples randomness $Q \leftarrow \{0,1\}^{\lambda_Q}$, runs $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}(x, w, Q)$, and returns $(\Phi, Q)$ to $\mathscr{A}$. On $\mathscr{A}$'s $i+1^{st}$ query $(\mathtt{Prove}, x^*, w^*)$, $\mathscr{B}$ passes $(\mathtt{Prove}, x^*, w^*)$ to its challenger and receives $(\pi^*, r^*)$ for $\pi^* = (\mathtt{com}^*, \mathtt{chl}^*, \mathtt{res}^*)$ that is either the result of running $\Sigma_R.\mathtt{Prove}$ on randomness $r^*$ or the result of running $\Sigma_R.\mathtt{SimProve}$ and $\Sigma_R.\mathtt{SimRand}$. Note that that $\mathscr{A}$ is expecting more proofs than just $\pi^*$, as the output of $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ is actually a tuple of proofs $\Phi = \pi_1, \ldots, \pi_n$. We therefore modify $\mathscr{A}$ again to distinguish the hybrids $i+1, j$ from $i+1, j+1$, where in the $i+1, j^{th}$ hybrid, the first $j$ proofs of the challenge proof tuple $\Phi^* = \pi_1, \ldots, \pi_n$

are computed according to $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ each with randomness $Q \in \{0,1\}^{\lambda_Q}$, and the $i+1, j+1^{st}$ through the $n^{th}$ proof are according to $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimRand}$. $\mathscr{B}$ constructs the entire $i+1^{st}$ hybrid proof tuple $\Phi_{i+1} = \Phi_1, \ldots, \Phi_n$ from the challenge query as follows.

Let the $j+1^{st}$ commitment $\mathtt{com}_{j+1} = \mathtt{com}^*$, and $\pi_{j+1} = (\mathtt{com}^*, \mathtt{chl}^*, \mathtt{res}^*)$. Since it must have the entire tuple $\mathtt{com}_1, \ldots, \mathtt{com}_n$ of commitments before running $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$, $\mathscr{B}$ starts by running the steps of $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ $n-(j+2)$ times. $\mathscr{B}$ then samples each $r_1, \ldots r_j$ uniformly at random from $\{0,1\}^{\lambda_r}$ and runs $\tau.\mathtt{Commit}$ $j$ times to obtain $\mathtt{com}_1, \ldots, \mathtt{com}_j$ and produce the full commitment vector $\overline{\mathtt{com}} = \mathtt{com}_1, \ldots, \mathtt{com}_n$. $\mathscr{B}$ computes each proof $\pi_l$ for $1 \leq l \leq j$ by running the steps of $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ $j$ times.

$\mathscr{B}$ generates the prover's hybrid random tape $Q^*$ as follows. The first $j\lambda_r$ bits are the $j$ $\lambda_r$-bit samples that $\mathscr{B}$ made while computing proofs according to $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$. The proceeding $\lambda_r$ bits are those returned by the adNIM-SHVZK challenger, $r^*$. To obtain the remaining bits, $\mathscr{B}$ runs $\Sigma_{R,\tau}.\mathtt{SimRand}$ on input each $\pi_l, r'_l$ for $j+2 \leq l \leq n$, where $r'_l$ is the same section of $\mathscr{B}$'s random tape that it used in running $\Sigma_{R,\tau}.\mathtt{SimProve}$ to obtain $\pi_l$. To simulate the "random challenge selection" section of $Q^*$, $\mathscr{B}$ first concatenates (in order) all of the challenges sampled while computing $\pi_1, \ldots, \pi_j$. For $\mathtt{chl}_{j+1}$ (i.e. $\mathtt{chl}^*$, which was produced by the adNIM-SHVZK challenger rather than being sampled by $\mathscr{B}$), $\mathscr{B}$ uses fresh randomness to sample new challenges and inserts $\mathtt{chl}_{j+1}$ as the first challenge in lexicographic order to map to the minimal output value, then concatenates all of the challenges (in order) to $Q^*$. Finally, $\mathscr{B}$ concatenates (again in order) all of the challenges it sampled while computing $\pi_{j+2}, \ldots, \pi_n$.

All that remains in $\mathscr{B}$'s simulation is to make sure $H$ is consistent with the proofs and randomness it has generated. For all proofs $\pi_l$ where $j+1 \leq l \leq n$, $\mathscr{B}$ programs the output of $H$ on input $(x, \overline{\mathtt{com}}, l, \mathtt{chl}_m, \mathtt{res}_m)$ to be the corresponding $b$-bit output in the challenge-to-output mappings for all challenges and responses $1 \leq m \leq 2^t$.

For the analysis, consider that $\mathscr{B}$'s execution of the proofs $\pi_1, \ldots, \pi_j$ and $\pi_{j+2}, \ldots, \pi_n$ and corresponding randomness is exactly what $\mathscr{A}$ expects in either hybrid. It remains to show that the challenge proof $\pi_{j+1}$ and corresponding randomness is also formatted according to what $\mathscr{A}$ expects. If the adNIM-SVHZK challenger is playing with bit $b = 0$ and generated $(\pi^*, r^*)$ according to $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ and randomness $r$, then $\pi_{j+1}$ and the "first message" section of $Q^*$ corresponding to $r_j$ (i.e. the randomness used to generate $\mathtt{com}_{j+1}$) will be exactly what $\mathscr{A}$ expects from the $i, j+1^{st}$ hybrid. If, on the other hand, $\pi^*, r^*$ were generated according to $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimRand}$, then $\mathscr{A}$'s view of $\pi_{j+1}$ and $r_j$ will be according to hybrid $i, j$.

Finally, we must argue that $\mathscr{A}$'s view of the "challenge-selection" section of $Q^*$ is formatted how $\mathscr{A}$ expects (since according to the hybrid experiment, $\mathscr{A}$ is expecting a "simulated" challenge selection process only when $\pi^*, r^*$ were generated according to $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimRand}$). The statistical indistinguishability of $\mathscr{B}$'s simulation for the challenge-selection section phase follows from the fact that all of the sampled challenges are identically distributed—they

are all sampled uniformly at random from $\{0,1\}^t$. Likewise, the programmed outputs of $H$ are all sampled uniformly at random from $\{0,1\}^b$. Therefore, if $\mathscr{B}$ outputs 0 when $\mathscr{A}$ outputs "Hybrid $i, j+1$" and 1 when $\mathscr{A}$ outputs "Hybrid $i, j$," it succeeds with the same probability as $\mathscr{A}$. Because $i$ is bounded by the number of prove queries and $j$ is bounded by the parameter $n$, the number of hybrids is bounded by some polynomial in the security parameter, and $\mathscr{B}$'s summed advantage over all of the hybrids is negligible.

The proof that $\Pi_R^{\mathtt{aFis}}$ is adNI-SSS follows identically to the corresponding proof that the regular randomized Fischlin transform is NI-SSS [24,27], except that $\mathscr{B}$'s passes the proof randomness $r$ from the adNI-SSS challenger to $\mathscr{A}$ any time $\mathscr{A}$ issues an adaptive corruption (in the same manner as described above). We therefore defer the argument to the full version of the proof (A.11). □

### 5.3 Realizing Efficient and Adaptive GUC NIZKPoK from $\Sigma$-protocols in the Global ROM(s)

We demonstrated in Section 4.2 that any adaptive SLC is sufficient to convert any $\Sigma$-protocol $\Sigma_R$ into a NISLE proof system $\Pi_R^{\mathtt{SLC}}$ that aUC-realizes $\mathcal{F}_{\mathtt{aNIZK}}$ in the $\mathcal{G}_{\mathtt{rpoRO}}$-hybrid model. In the previous section, we proved that the adaptive randomized Fischlin transform `aFis` is an adaptive SLC for any $\Sigma$-protocol that has either the quasi-unique responses or strong special soundness properties. Therefore, we can efficiently aUC-realize $\mathcal{F}_{\mathtt{aNIZK}}$ in the programmable global ROM using `aFis` and any such $\Sigma$-protocol.

**Corollary 1.** *Let $\Sigma_R$ be any adaptive $\Sigma$-protocol for a relation $R$ (Definition 4) with the required properties for* `aFis` *(Definition 29), and* `aFis` *be the adaptive randomized Fischlin transform (Definition 11). Then the NISLE proof system $\Pi_R^{\mathtt{SLC}} \leftarrow \mathtt{aFis}(\Sigma_R)$ aUC-realizes $\mathcal{F}_{\mathtt{aNIZK}}$ in the $\mathcal{G}_{\mathtt{rpoRO}}$-hybrid model (Definition 2 where $\mathcal{G}_{\mathtt{RO}} := \mathcal{G}_{\mathtt{rpoRO}}$).*

*Proof.* The corollary follows directly from Theorems 2 and 4. □

Similarly, we showed in Section 4.3 that any adaptive SLC is sufficient in conjunction with the adaptive OR-protocol compiler `aguc` from Definition 10 to convert any $\Sigma$-protocol into a NISLE proof system $\Pi_{\mathrm{R}\vee\mathrm{S}}^{\mathtt{aguc}}$ that aUC-realizes $\mathcal{F}_{\mathtt{aNIZK}}$ in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model. Therefore, we can efficiently aUC-realize $\mathcal{F}_{\mathtt{aNIZK}}$ in the non-programmable global ROM using `aguc`, `aFis` and any $\Sigma$-protocol that is compatible with `aFis`.

**Corollary 2.** *Let $\Sigma_R$ be any adaptive $\Sigma$-protocol for a relation $R$ (Definition 4) with the required properties for* `aFis` *(Definition 29),* `aguc` *be the adaptive OR-protocol compiler (Definition 10), and* `aFis` *be the adaptive randomized Fischlin transform (Definition 11). Then the NISLE proof system $\Pi_{\mathrm{R}\vee\mathrm{S}}^{\mathtt{aguc}} \leftarrow \mathtt{aguc}(\Sigma_R, \mathtt{aFis})$ aUC-realizes $\mathcal{F}_{\mathtt{aNIZK}}$ in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model (Definition 2 where $\mathcal{G} := \mathcal{G}_{\mathtt{roRO}}$ and $* := \mathcal{F}_{\mathtt{CRS}}$).*

*Proof.* The corollary follows directly from Theorems 3 and 4. □

# 6 Practical Adaptive $\Sigma$-protocols

In this section, we recall the abstract treatment of identification schemes from linear function families due to Hauck, Kiltz, and Loss [25], and remodel it to capture many simple $\Sigma$-protocols (Section 6.1). We then give three satisfying instantiations: proofs of knowledge of a discrete logarithm (Section 6.2) and $n$ equivalent representations of $m$ witnesses (Section 6.3). As proven in Sections 4 and 5.1, any $\Sigma$-protocol that qualifies as adaptive can be efficiently bootstrapped into an adaptive UC NIZKPoK. We therefore conclude with an abstract blueprint and three instantiations of efficient, composable, and adaptively-secure NIZKPoK for a variety of real-world applications.

## 6.1 Simple Adaptive $\Sigma$-protocol Abstraction

Hauck et al.'s construction of "canonical identification schemes" from pseudo-modules and linear function families [25] describes an abstract three-move identification scheme that bears close resemblance to the three-move form of a $\Sigma$-protocol. We remodel their construction as a $\Sigma$-protocol template and add new adaptive $\Sigma$-protocol-specific abstractions for the `SimProve`, `SimRand`, and `Extract` algorithms.

For simplicity and since all of our instantiations are parameterized over modules, we narrow the focus of our abstraction to modules rather than pseudo-modules. Briefly, we form the $\Sigma$-protocol module over the challenge space $\mathcal{C}$ and the language (statement) space $\mathcal{X}$. The definition of a module requires that $\mathcal{C}$ is a ring with multiplicative identity $1_{\mathcal{C}}$, $\mathcal{X}$ is a group with additive commutativity, and for all $\mathtt{chl}, \mathtt{chl}' \in \mathcal{C}$ and $x, y \in \mathcal{X}$, there exists a map $\mathcal{C} \times \mathcal{X} \to \mathcal{X}$ satisfying the following properties: 1) $\mathtt{chl} \cdot (x + y) = \mathtt{chl} \cdot x + \mathtt{chl} \cdot y$, 2) $(\mathtt{chl} + \mathtt{chl}') \cdot x = \mathtt{chl} \cdot x + \mathtt{chl}' \cdot x$, 3) $(\mathtt{chl} \cdot \mathtt{chl}') \cdot x = \mathtt{chl} \cdot (\mathtt{chl}' \cdot x)$, and 4) $1_{\mathcal{C}} \cdot x = x$. In order to guarantee the special soundness property, we additionally require that for any two proofs $\pi = (\mathtt{com}, \mathtt{chl}, \mathtt{res})$ and $\pi' = (\mathtt{com}, \mathtt{chl}', \mathtt{res}')$ for $x$ such that $\mathtt{Verify}(x, \pi) = \mathtt{Verify}(x, \pi') = 1$ and $\mathtt{chl} \neq \mathtt{chl}' \neq 0_{\mathcal{C}}$, the inverse of $(\mathtt{chl} - \mathtt{chl}')$ exists. We write the inversion $(\mathtt{chl} - \mathtt{chl}) \cdot (\mathtt{chl} - \mathtt{chl}')^{-1} = 1_{\mathcal{C}}$.[7] We highlight the properties of linear function families that are critical to understanding our results as part of the proof below.

**Definition 12 (Simple Adaptive $\Sigma$-protocol Candidate).** *The* simple adaptive $\Sigma$-protocol candidate $\Sigma_{\mathtt{sim}}$ *for relation $R$ is a tuple of efficient procedures* $\Sigma_{\mathtt{sim}} = (\mathtt{Setup}, \mathtt{Prove}, \mathtt{Verify}, \mathtt{SimSetup}, \mathtt{SimProve}, \mathtt{SimRand}, \mathtt{Extract})$ *that adhere to the following format:*

[7] Note that some $\Sigma$-protocol constructions that would otherwise fit this template, such as the proof of knowledge of an RSA inverse, do not quite work because the extractor cannot compute the inverse of $(\mathtt{chl} - \mathtt{chl}')$ directly (in the case of the RSA inverse, the extractor uses Shamir's trick [10]). For such constructions, it might be convenient to use the abstraction for the adaptive SHVZK property and treat the special soundness property separately.

- $\texttt{ppm} \leftarrow \texttt{Setup}(1^\lambda)$*: Define the linear function family* $\texttt{LF} = (\texttt{PGen}, \texttt{F})$ *(Definition 3.1 in [25]) such that* $\texttt{ppm} \leftarrow \texttt{PGen}(1^\lambda)$ *fixes a statement space* $\mathcal{X}$*, witness space* $\mathcal{W}$*, first message space* $\mathcal{A}$*, challenge space* $\mathcal{C}$*, response space* $\mathcal{B}$*, and randomness space* $\mathcal{R}$ *such that* $\mathcal{W} = \mathcal{R} = \mathcal{B}$*,* $\mathcal{X} = \mathcal{A}$*,* $\mathcal{W}$ *and* $\mathcal{X}$ *form* modules *over* $\mathcal{C}$*,* $|\mathcal{X}| \geq |\mathcal{C}| \geq 2^{2\lambda}$*, and the evaluation function* $\texttt{F} : \mathcal{W} \to \mathcal{X}$*.*
- $\pi \leftarrow \texttt{Prove}((\texttt{ppm}, x, w, r), (\texttt{ppm}, x))$*: P sends V a first message* $\texttt{com} = \texttt{F}(r)$*. V replies with a challenge* $\texttt{chl} \leftarrow_\$ \mathcal{C}$*. P responds with* $\texttt{res} = \texttt{chl} \cdot w + r$*.*
- $\{0,1\} \leftarrow \texttt{Verify}(\texttt{ppm}, x, \pi)$*: Parse* $\pi$ *as* $(\texttt{com}, \texttt{chl}, \texttt{res})$*. If* $\texttt{F}(\texttt{res}) = \texttt{chl} \cdot x + \texttt{com}$*, output 1 (accept). Otherwise, output 0 (reject).*
- $\texttt{ppm} \leftarrow \texttt{SimSetup}(1^\lambda)$*: Invoke* $\texttt{Setup}(1^\lambda)$ *and return* ppm*.*
- $\pi \leftarrow \texttt{SimProve}(\texttt{ppm}, x, \texttt{chl})$ : *Sample* $\texttt{res} \leftarrow_\$ \mathcal{B}$ *and compute* $\texttt{com} = F(\texttt{res} - \texttt{chl} \cdot w)$*. Return* $\pi = (\texttt{com}, \texttt{chl}, \texttt{res})$*.*
- $r \leftarrow \texttt{SimRand}(\texttt{ppm}, x, \pi, w)$ : *Parse* $\pi = (\texttt{com}, \texttt{chl}, \texttt{res})$*. Compute and output* $r = \texttt{res} - (\texttt{chl} \cdot w)$*.*
- $w \leftarrow \texttt{Extract}(\texttt{ppm}, x, \pi, \pi')$ : *Given any two proofs* $\pi = (\texttt{com}, \texttt{chl}, \texttt{res})$ *and* $\pi' = (\texttt{com}, \texttt{chl}', \texttt{res}')$ *such that* $\texttt{Verify}(x, \pi) = \texttt{Verify}(x, \pi') = 1$ *and* $\texttt{chl} \neq \texttt{chl}'$*, compute and output* $w = (\texttt{res} - \texttt{res}') \cdot (\texttt{chl} - \texttt{chl}')^{-1}$*.*

**Theorem 5.** *The simple adaptive* $\Sigma$*-protocol candidate given above (Definition 12) is an adaptive* $\Sigma$*-protocol (Definition 3).*

*Proof.* The following proof relies on two core properties of the evaluation function F, described below (for details in context, see Definition 4.1 [25]). First, the *pseudo-module homomorphism* (PMH) property states that for all $y, z \in \mathcal{W}$, $\texttt{F}(\texttt{chl} \cdot y + z) = \texttt{chl} \cdot \texttt{F}(y) + \texttt{F}(z)$. Second, the *smoothness property* guarantees that for all $y \in \mathcal{W}$, $\texttt{F}(y)$ is uniformly distributed over $\mathcal{X}$. (Recall these properties hold as well for $\mathcal{W} = \mathcal{R} = \mathcal{B}$ and $\mathcal{X} = \mathcal{A}$.) We will now show that the simple $\Sigma$-protocol format given above satisfies the necessary completeness, adaptive special honest-verifier zero-knowledge, and special soundness properties of an adaptive $\Sigma$-protocol.

*Completeness.* The verifier checks whether $F(\texttt{res}) = \texttt{chl} \cdot x + \texttt{com}$. By the PMH property, we have $F(\texttt{chl} \cdot w + r) = \texttt{chl} \cdot F(w) + F(r) = \texttt{chl} \cdot x + \texttt{com}$.

*Adaptive SHVZK.* As long as $r \leftarrow_\$ \mathcal{R}$ and $\texttt{chl} \leftarrow_\$ \mathcal{C}$, the distribution of $\texttt{res} = \texttt{chl} \cdot w + r$ computed in $\Sigma_{\texttt{sim}}.\texttt{Prove}$ is uniformly random in $\mathcal{B}$. The $\texttt{res} \leftarrow_\$ \mathcal{B}$ and $\texttt{chl} \leftarrow_\$ \mathcal{C}$ sampled by $\Sigma_{\texttt{sim}}.\texttt{SimProve}$ are therefore identical to the res and chl in a real proof. By the smoothness of F, $\Sigma_{\texttt{sim}}.\texttt{SimProve}$'s $\texttt{com} = F(\texttt{res} - \texttt{chl} \cdot w)$ is also uniformly random in $\mathcal{A}$. Since res and chl are uniformly random in $\mathcal{B}$ and $\mathcal{C}$ respectively, the distribution of $r = \texttt{res} - (\texttt{chl} \cdot w)$ will be similarly random. Note that both simulations are correct, since

$$\texttt{F}(r) = \texttt{F}(\texttt{res} - \texttt{chl} \cdot w) = -\texttt{chl} \cdot \texttt{F}(w) + \texttt{F}(\texttt{res}) = -\texttt{chl} \cdot x + \texttt{chl} \cdot x + \texttt{com} = \texttt{com}$$

by the PMH property. The outputs of $\Sigma_{\texttt{sim}}.\texttt{Prove}$ and $r$ are therefore distributed identically to the outputs of $\Sigma_{\texttt{sim}}.\texttt{SimProve}$ and $\Sigma_{\texttt{sim}}.\texttt{SimRand}$, respectively.

*Special Soundness.* Solving the two verification equations for $\mathtt{com}$ and setting them equal, we get $\mathtt{F}(\mathtt{res}) - \mathtt{chl} \cdot x = \mathtt{F}(\mathtt{res}') - \mathtt{chl}' \cdot x$, which implies

$$\mathtt{F}(w) \cdot \mathtt{chl} - \mathtt{F}(w) \cdot \mathtt{chl}' = \mathtt{F}(\mathtt{res}) - \mathtt{F}(\mathtt{res}') \text{ (substitution and commutativity)}$$

$$\mathtt{F}(w) \cdot (\mathtt{chl} - \mathtt{chl}') = \mathtt{F}(\mathtt{res}) - \mathtt{F}(\mathtt{res}') \text{ (distributivity)}$$

$$\mathtt{F}(w) = \mathtt{F}(\mathtt{res}) - \mathtt{F}(\mathtt{res}') \cdot (\mathtt{chl} - \mathtt{chl}')^{-1} \text{ (invertability of } \mathtt{chl} - \mathtt{chl}')$$

$$\Rightarrow \mathtt{F}(w) = \mathtt{F}((\mathtt{res} - \mathtt{res}') \cdot (\mathtt{chl} - \mathtt{chl}')^{-1}) \text{ (PMH property)}$$

$$\Rightarrow w = (\mathtt{res} - \mathtt{res}') \cdot (\mathtt{chl} - \mathtt{chl}')^{-1}. \square$$

### 6.2 Adaptive Proof of Knowledge of Discrete Logarithm

Proofs of knowledge of discrete logarithm (PoK DL), also known as Schnorr proofs, are the backbone of many practical constructions of group [12,5], threshold [3], blind [25], and multi- [3,22] signatures.

**Theorem 6.** *A proof of knowledge of discrete logarithm can be described as a simple adaptive $\Sigma$-protocol (Definition 12).*

*Proof.* The PoK DL can be described using the simple adaptive $\Sigma$-protocol format from Definition 12 as follows.

The public parameters $\mathtt{ppm}$ are $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a cyclic group of prime order $q$, and $g$ is a generator of $\mathbb{G}$. The statement and first-message spaces are $\mathcal{X} = \mathcal{A} = \mathbb{G}$, the witness, randomness, and response spaces are $\mathcal{W} = \mathcal{B} = \mathcal{R} = \mathbb{Z}_q$, and the challenge space is $\mathcal{C} = \mathbb{Z}_q^*$. The evaluation function $\mathtt{F} : \mathbb{Z}_q \to \mathbb{G}$ is defined such that $\mathtt{F}(w) = w \cdot g$. For elements $x$ and $y$ in $\mathbb{Z}_q$, we define the operation $x \cdot y$ to be the process of adding $y$ to itself (component-wise) $x$ times modulo $q$. For elements $x \in \mathbb{Z}_q, \mathbb{Z}_q^*$ and $y \in \mathbb{G}$, we define $x \cdot y$ to be the process of performing the group operation on $y$ with itself $x$ times. (In traditional Schnorr, this would be exponentiation $y^x$; in the elliptic curve version, it would the scalar multiplication $xy$ where $y$ is a curve point.)

$\mathbb{Z}_q^*$ and $\mathbb{G}$ satisfy the requirements of a module, and all elements in $\mathcal{C} = \mathbb{Z}_q^*$ are invertible. Furthermore, the pseudo-module homomorphism property is satisfied because $\mathtt{F}(w) := w \cdot g \Rightarrow \mathtt{F}(\mathtt{chl} \cdot w + r) = (\mathtt{chl} \cdot w + r) \cdot g = \mathtt{chl} \cdot w \cdot g + r \cdot g = \mathtt{chl} \cdot F(w) + F(r)$. Finally, $\mathtt{F}$ is smooth, since for $r \leftarrow_{\$} \mathbb{Z}_q$, $F(r) = r \cdot g$ is uniformly distributed over $\mathbb{G}$.

### 6.3 Adaptive Proof of Knowledge of Equality of $n$ Representations

Proofs of knowledge of equality of $n$ representations of $m$ witnesses (PoK EqRep) are essential building blocks in the construction of cryptographic shuffles [34,28], accumulators [2,10], and anonymous credentials [9], which form the basis of anonymous networks, voting, payment, and identification systems. Note also that PoK EqRep is a generalization of other common $\Sigma$-protocols such as standard proofs of knowledge of representation [13,11] and Okamoto-Schnorr [29], and that the following result holds for those narrower instantiations as well.

**Theorem 7 (PoK EqRep is an Adaptive $\Sigma$-protocol).** *A proof of knowledge of equality of n representations can be described as a simple adaptive $\Sigma$-protocol (Definition 12).*

*Proof.* PoK EqRep is a natural extension of PoK DL, and fits the abstract form from Definition 12 as follows. The public parameters $\mathtt{ppm}$ are $(\mathbb{G}, q, \overline{g}_1, \ldots, \overline{g}_n)$, where $\mathbb{G}$ is a cyclic group of prime order $q$, and each $\overline{g}_i$ is a vector of $m$ generators $g_{i,1}, \ldots, g_{i,m}$. The statement and first-message spaces are $\mathcal{X} = \mathcal{A} = \mathbb{G}^n$, the witness, randomness, and response spaces are $\mathcal{W} = \mathcal{R} = \mathcal{B} = \mathbb{Z}_q^m$, and the challenge space is $\mathcal{C} = \mathbb{Z}_q^*$. The evaluation function $\mathtt{F} : \mathbb{Z}_q^m \to \mathbb{G}^n$ is such that

$$\mathtt{F}(w) = \mathtt{F}(w_1, \ldots, w_m) = \prod_{i=1}^{m} w_i \cdot g_{1,i}, \ldots, \prod_{i=1}^{m} w_i \cdot g_{n,i}.$$

The operator $\cdot$ is defined the same as in the proof of Theorem 6 above (with emphasis on *component-wise*). Likewise, the pseudo-module homomorphism and smoothness properties are satisfied via component-wise application of the logic from Theorem 6. Since PoK EqRep is more involved than PoK DL, we review the specific details of the construction in the supplementary materials (A.12) to confirm they indeed fit the abstract template. □

## Acknowledgements

Many thanks to Ran Cohen for pointing our attention to the Universal Composability with Global Subroutines model.

## References

1. Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. Universal composition with global subroutines: Capturing global setup within plain uc. In *Theory of Cryptography Conference*, pages 1–30. Springer, 2020.
2. Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 301–315. IEEE, 2017.
3. Mihir Bellare, Elizabeth Crites, Chelsea Komlo, Mary Maller, S Tessaro, and C Zhu. Better than advertised security for non-interactive threshold signatures. In *Advances in Cryptology-CRYPTO*, 2022.
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
5. Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 331–345. Springer, 2000.

6. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–312. Springer, 2018.
7. Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. pages 201–210. ACM, 2006.
8. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact E-cash. In Ronald Cramer, editor, *Advances in Cryptology — Eurocrypt 2005*, volume 3494, pages 302–321. Springer, 2005.
9. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045, pages 93–118. Springer Verlag, 2001.
10. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76. Springer, 2002.
11. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO 2003*, pages 126–144. Springer, 2003.
12. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *CRYPTO 1997*, pages 410–424. Springer, 1997.
13. Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science*, 260, 1997.
14. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.
15. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography Conference*, pages 61–85. Springer, 2007.
16. Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO 2001*, pages 19–40. Springer, 2001.
17. Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical uc security with a global random oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 597–608, 2014.
18. Ran Canetti and Tal Rabin. Universal composition with joint state. In *CRYPTO 2003*, pages 265–281. Springer, 2003.
19. Ran Canetti, Pratik Sarkar, and Xiao Wang. Triply adaptive uc nizk. *ePrint Archive*, 2020.
20. Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.
21. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO 1994*, pages 174–187. Springer, 1994.
22. Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove schnorr assuming schnorr: Security of multi-and threshold signatures. *ePrint Archive*, 2021.
23. Ivan Damgård. On $\sigma$-protocols. University of Aarhus, Department of Computer Science, 2002.
24. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. 2005. Manuscript. Available from

http://www.cryptoplexity.informatik.tu-darmstadt.de/media/crypt/publications_1/fischlinonline-extractor2005.pdf.
25. Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 345–375. Springer, 2019.
26. Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. *ePrint Archive*, 2022.
27. Anna Lysyanskaya and Leah Namisa Rosenbloom. Universally composable sigma-protocols in the global random-oracle model. *ePrint Archive*, 2022.
28. C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125, 2001.
29. Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO 1992*, pages 31–53. Springer, 1992.
30. Somnath Panja and Bimal Kumar Roy. A secure end-to-end verifiable e-voting system using zero knowledge based blockchain. *ePrint Archive*, 2018.
31. Rafael Pass. On deniability in the common reference string and random oracle model. In *CRYPTO 2003*, pages 316–337, 2003.
32. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
33. Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *CRYPTO 1999*, pages 148–164. Springer, 1999.
34. Douglas Wikström. A commitment-consistent proof of a shuffle. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, pages 407–421. Springer, 2009.

## Supplementary Materials

# A Supplementary Materials

### A.1 Notation

We use $\lambda$ for the security parameter, and say an algorithm $\mathscr{A}$ is efficient in $\lambda$ if its runtime can be expressed as a polynomial $\mathsf{poly}(\lambda)$ on input $\lambda$. We say a function $\mathsf{negl}$ is negligible in $\lambda$ if for every positive polynomial $p$ there exists a threshold $N$ such that for all $\lambda > N$, $\mathsf{negl}(\lambda) < \frac{1}{p(\lambda)}$.

When we write $y \leftarrow z$ where $z$ is a quantity, we mean that $y$ is assigned the value $z$. Similarly, $y \leftarrow \mathscr{A}(x)$ means that $y$ is assigned the output of algorithm $\mathscr{A}$ on input $x$. We write $\perp \leftarrow \mathscr{A}(x)$ to indicate that $\mathscr{A}$ has halted on input $x$ with no output, such that any process that invoked $\mathscr{A}$ can resume. By $y \leftarrow_{\$} Z$ where $Z$ is a set or a probability distribution, we mean that $y$ is assigned an element sampled uniformly at random from $Z$.

If two distributions $Y$ and $Z$ are equivalent, we use the notation $Y = Z$. If $Y$ and $Z$ are statistically indistinguishable, we use the notation $Y \approx_s Z$. If $Y$ and $Z$ are only computationally indistinguishable, we use the notation $Y \approx_c Z$. When we say two distributions are statistically (resp. computationally) indistinguishable, we mean that for all $\lambda$, the probability that any algorithm $\mathscr{A}$ (resp. PPT algorithm $\mathscr{A}$) can determine whether a mystery element $x$ was sampled from $Y$ or $Z$ is only negligibly greater than a random guess, or $\frac{1}{2} + \mathsf{negl}(\lambda)$. We might also say in this case that $\mathscr{A}$ distinguishes $Y$ from $Z$ with negligible advantage over a random guess.

### A.2 The Observable and Programmable Global ROs

**Definition 13 (Observable Global RO $\mathcal{G}_{\mathtt{roRO}}$).** [17,6] *The* observable global RO $\mathcal{G}_{\mathtt{roRO}}$ *is a tuple of algorithms* $(\mathtt{Query}, \mathtt{Observe})$ *defined over an output length* $\ell$ *and an initially empty list of queries* $\mathcal{Q}$*:*

- $v \leftarrow \mathtt{Query}(x)$ : *Parse* $x$ *as* $(s, x')$ *where* $s$ *is an SID. If a list* $\mathcal{Q}_s$ *of illegitimate queries for* $s$ *does not yet exist, set* $\mathcal{Q}_s = \perp$*. If the caller's SID* $\neq s$*, add* $(x, v)$ *to* $\mathcal{Q}_s$*. If there already exists a pair* $(x, v)$ *in the query list* $\mathcal{Q}$*, return* $v$*. Otherwise, choose* $v$ *uniformly at random from* $\{0,1\}^{\ell}$*, store the pair* $(x, v)$ *in* $\mathcal{Q}$*, and return* $v$*.*
- $\mathcal{Q}_s \leftarrow \mathtt{Observe}(s)$ : *If a list* $\mathcal{Q}_s$ *of illegitimate queries for* $s$ *does not yet exist, set* $\mathcal{Q}_s = \perp$*. Return* $\mathcal{Q}_s$*.*

**Definition 14 (Restricted Programmable Observable Global RO $\mathcal{G}_{\mathtt{rpoRO}}$).** [6] *The* restricted programmable observable global random oracle $\mathcal{G}_{\mathtt{rpoRO}}$ *is a tuple of algorithms* $(\mathtt{Query}, \mathtt{Observe}, \mathtt{Program}, \mathtt{IsProgrammed})$ *defined over an output length* $\ell$ *and initially empty lists* $\mathcal{Q}$ *(queries) and* $\mathtt{prog}$ *(programmed queries):*

- $v \leftarrow \mathtt{Query}(x)$ : *Same as Definition 13 above.*

- $\mathcal{Q}_s \leftarrow \texttt{Observe}(s)$ : *Same as Definition 13 above.*
- $\{0,1\} \leftarrow \texttt{Program}(x, v)$ : *If* $\exists v' \in \{0,1\}^\ell$ *such that* $(x, v') \in \mathcal{Q}$ *and* $v \neq v'$, *output* 0. *Otherwise, add* $(x, v)$ *to* $\mathcal{Q}$ *and* `prog` *and output* 1.
- $\{0,1\} \leftarrow \texttt{IsProgrammed}(x)$ : *Parse* $x$ *as* $(s, x')$. *If the caller's SID* $\neq s$, *output* $\perp$. *Otherwise if* $x \in$ `prog`, *output* 1. *Otherwise, output* 0.

### A.3 $\Sigma$-protocols

**Definition 15 (Adaptive $\Sigma$-protocol Template).** *The* adaptive $\Sigma$-protocol template *for a relation* $R$ *is a tuple of efficient algorithms* $\tau = (\texttt{Setup}, \texttt{Commit}, \texttt{Challenge}, \texttt{Respond}, \texttt{Decision})$, *defined as follows.*

- $\texttt{ppm} \leftarrow \texttt{Setup}(1^\lambda)$: *Given a security parameter, generates a set of public parameters* `ppm` *which minimally include* $1^\lambda$, *challenge length* $\ell$, *and randomness security parameter* $\lambda_r$.
- $\texttt{com} \leftarrow \texttt{Commit}(\texttt{ppm}, x, w, r)$: *Given statement* $x$, *witness* $w$, *and randomness* $r \leftarrow_\$ \{0,1\}^{\lambda_r}$, $P$ *sends* $V$ *a message* `com`.
- $\texttt{chl} \leftarrow \texttt{Challenge}(\texttt{ppm}, x, \texttt{com})$: $V$ *sends* $P$ *a random* $\ell$*-bit string* `chl`.
- $\texttt{res} \leftarrow \texttt{Respond}(\texttt{ppm}, x, w, \texttt{com}, \texttt{chl})$: $P$ *sends* $V$ *a reply* `res`.
- $\{0,1\} \leftarrow \texttt{Decision}(\texttt{ppm}, x, \texttt{com}, \texttt{chl}, \texttt{res})$: $V$ *decides whether to output 1 (accept) or 0 (reject) based on the input* $(\texttt{ppm}, x, \texttt{com}, \texttt{chl}, \texttt{res})$.

*The tuple* $(\texttt{com}, \texttt{chl}, \texttt{res})$ *is called a* transcript *or* proof. *We say a transcript or proof is* valid *or* accepting *if* $\texttt{Decision}(\texttt{ppm}, x, \texttt{com}, \texttt{chl}, \texttt{res})$ *outputs 1.*

**Definition 16 (Damgård's Protocol Template for Relation $R$).** [23] *Let the common input to* $P$ *and* $V$ *be* $x$, *and the private input to* $P$ *be a value* $w$ *such that* $(x, w) \in R$. *The* protocol template *is the following three-round transaction:*

1. $P$ sends $V$ a message $a$.
2. $V$ sends $P$ a random $\ell$-bit string $e$.
3. $P$ sends $V$ a reply $z$.
4. $V$ decides to accept (output 1) or reject (output 0) based solely on the values $(x, a, e, z)$.

*We say a transcript* $(a, e, z)$ *is an* accepting transcript *for* $x$ *if the protocol instructs* $V$ *to accept based on the values* $(x, a, e, z)$.

**Definition 17 (Original $\Sigma$-protocol).** [23] *A protocol* $\Phi$ *is a* $\Sigma$-protocol *for relation* $R$ *if it is a three-round public-coin protocol of the form in Definition 3 and the following requirements hold:*

- **Completeness**: *If* $P$ *and* $V$ *follow the protocol on input* $x$ *and private input* $w$ *to* $P$ *where* $(x, w) \in R$, *then* $V$ *always accepts.*

- **Special Soundness**: *There exists a polynomial-time algorithm $E$ that given any $x$ and any pair of accepting transcripts* $(\mathtt{com}, \mathtt{chl}, \mathtt{res})$ *and* $(\mathtt{com}, \mathtt{chl}', \mathtt{res}')$ *for $x$ where* $\mathtt{chl} \neq \mathtt{chl}'$*, outputs $w$ such that $(x, w) \in R$.*
- **Special honest verifier zero knowledge**: *There exists a PPT simulator $M$, which on input $x$ and* $\mathtt{chl}$ *outputs a transcript of the form* $(\mathtt{com}, \mathtt{chl}, \mathtt{res})$ *with the same probability distribution as transcripts between the honest $P$ and $V$ on common input $x$. Formally, for every $x$ and $w$ such that $(x, w) \in R$ and every* $\mathtt{chl} \in \{0,1\}^\ell$ *it holds that*

$$\{M(x, \mathtt{chl})\} \equiv \{\langle P(x, w), V(x, w)\rangle\}$$

*where $M(x, \mathtt{chl})$ denotes the output of simulator $M$ on input $x$ and* $\mathtt{chl}$*, and $\langle P(x, w), V(x, w)\rangle$ denotes the output transcript of an execution between $P$ and $V$, where $P$ has input $(x, w)$, $V$ has input $x$, and $V$'s random tape (determining its query) equals* $\mathtt{chl}$*.*

*The value $\ell$ is called the* challenge length*.*

We give formal descriptions of the completeness, special honest-verifier zero-knowledge, and special soundness properties. Other than notational differences, our definitions are due to Damgård [23].

**Definition 18 (Completeness).** *A $\Sigma$-protocol $\Sigma_R$ for relation $R$ is* complete *if for all $(x, w) \in R$ and $\pi \leftarrow \Sigma_R.\mathtt{Prove}((x, w), x)$, $\Sigma_R.\mathtt{Verify}(x, \pi) = 1$.*

**Definition 19 (Special Honest-Verifier Zero-Knowledge).** *A $\Sigma$-protocol $\Sigma_R$ for relation $R$ is statistical (resp. computational)* special honest-verifier zero-knowledge (SHVZK) *if there exist algorithms* $\mathtt{SimSetup}$ *and* $\mathtt{SimProve}$ *such that for any security parameter $\lambda$, any adversary (resp. any* PPT *adversary) $\mathscr{A}$, and a bit $b \leftarrow_\$ \{0,1\}$, there exists some negligible function* $\mathsf{negl}$ *such that $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where $b'$ is the result of running the game $\mathsf{SHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, b)$ from Figure 4. We say $\mathscr{A}$ wins the* SHVZK *game if $\Pr[b' = b] > \frac{1}{2} + \mathsf{negl}(\lambda)$.*

**Definition 20 (Special Soundness).** *A $\Sigma$-protocol $\Sigma_R$ for relation $R$ is* special sound *if there exists a* PPT *algorithm* $\mathtt{Extract}$ *such that for any security parameter $\lambda$, any* PPT *adversary $\mathscr{A}$,*

$$\Pr[\mathtt{Fail} \leftarrow \mathsf{SS}_{\mathscr{A},\Sigma_R}(1^\lambda)] \leq \mathsf{negl}(\lambda),$$

*where* SS *is the special soundness game described in Figure 5. We say $\mathscr{A}$ wins the* SS *game if $\Pr[\mathtt{Fail} \leftarrow \mathsf{SS}_{\mathscr{A},\Sigma_R}(1^\lambda)] > \mathsf{negl}(\lambda)$.*

### A.4 Modeling the Generic RO $H$

The traditional RO $H_f$ is parameterized by a function $f \leftarrow_\$ F$ selected from random function family $F$. The programmable RO $H_L$, is parameterized by a *list* $L$ that can be added to (but not edited by) the simulator.

$\mathsf{SHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, 0)$: REAL

1 : $\mathtt{ppm} \leftarrow \Sigma_R.\mathtt{Setup}(1^\lambda)$
2 : $(\mathtt{Prove}, x, w), \mathtt{st} \leftarrow \mathscr{A}(1^\lambda, \mathtt{ppm})$
3 : **if** $R(x, w) = 1$ :
4 : $\quad \pi \leftarrow \Sigma_R.\mathtt{Prove}((x, w), (x, \mathtt{chl}))$
5 : **else** :
6 : $\quad \pi \leftarrow \perp$
7 : $b' \leftarrow \mathscr{A}(\mathtt{st}, \pi)$
8 : **return** $b'$

$\mathsf{SHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, 1)$: IDEAL

1 : $(\mathtt{ppm}, z) \leftarrow \Sigma_R.\mathtt{SimSetup}(1^\lambda)$
2 : $(\mathtt{Prove}, x, w), \mathtt{st} \leftarrow \mathscr{A}(1^\lambda, \mathtt{ppm})$
3 : **if** $(x, w) \in R$ :
4 : $\quad \pi \leftarrow \Sigma_R.\mathtt{SimProve}(x, z, \mathtt{chl})$
5 : **else** :
6 : $\quad \pi \leftarrow \perp$
7 : $b' \leftarrow \mathscr{A}(\mathtt{st}, \pi)$
8 : **return** $b'$

**Fig. 4.** Special Honest-Verifier Zero-Knowledge (SHVZK) Game.

$\mathsf{SS}_{\mathscr{A},\Sigma_R}(1^\lambda)$

1 : $\mathtt{ppm} \leftarrow \Sigma_R.\mathtt{Setup}(1^\lambda)$
2 : $(\mathtt{Challenge}, x, \pi, \pi') \leftarrow \mathscr{A}(1^\lambda, \mathtt{ppm})$
3 : **parse** $\pi = (\mathtt{com}, \mathtt{chl}, \mathtt{res}), \pi' = (\mathtt{com}', \mathtt{chl}', \mathtt{res}')$
4 : **if** $\Sigma_R.\mathtt{Verify}(x, \pi) = \Sigma_R.\mathtt{Verify}(x, \pi') = 1 \wedge$
5 : $\quad \mathtt{com} = \mathtt{com}' \wedge \mathtt{chl} \neq \mathtt{chl}'$ :
6 : $\quad w \leftarrow \Sigma_R.\mathtt{Extract}(x, \pi, \pi')$
7 : $\quad$ **if** $R(x, w) = 0$ :
8 : $\quad\quad$ **return** `Fail`
9 : **return** `Success`

**Fig. 5.** Special Soundness (SS) Game.

RO $H_f(x)$

1 : **return** $f(x)$

Random List Oracle $H_L(x)$

1 : **if** $\exists\ v$ **s.t.** $(x, v) \in L$ :
2 : $\quad$ **return** $v$
3 : **else** :
4 : $\quad v \leftarrow \{0,1\}^\ell$
5 : $\quad L$.**append**$(x, v)$
6 : $\quad$ **return** $v$

Interface $\mathtt{Prog}_L(x, v)$

1 : **if** $\nexists\ v'$ **s.t.** $(x, v') \in L$ :
2 : $\quad L$.**append**$(x, v)$

**Fig. 6.** Random Oracle Functionalities for NIM-SHVZK and NI-SSS Games [27].

### A.5 Non-Interactive Special Soundness

Non-interactive special soundness (NI-SS) is a weakened version of the NI-SSS game where $\mathscr{A}$ does not get to issue `Prove` queries to the simulator algorithm `SimProve`.

**Definition 21 (Non-Interactive Special Soundness).** *A NISLE proof system* $\Pi_R^{\mathsf{SLC}} = (\mathtt{Setup}^H, \mathtt{Prove}^H, \mathtt{Verify}^H, \mathtt{SimSetup}, \mathtt{SimProve}, \mathtt{Extract})$ non-interactive special sound (NI-SS) *in the random-oracle model if there exists an algorithm* $\Pi_R^{\mathsf{SLC}}.\mathtt{Extract}$ *such that for any security parameter* $\lambda$ *any random oracle* $H$*, and any* PPT *adversary* $\mathscr{A}$*,*

$$\Pr[\mathtt{Fail} \leftarrow \mathsf{NI\text{–}SS}_{\mathscr{A},\Pi_R^{\mathsf{SLC}}}(1^\lambda)] \leq \mathsf{negl}(\lambda),$$

*where* NI–SS *is the NI-SS game described in Figure 7. We say* $\mathscr{A}$ *wins the* NI–SS *game if* $\Pr[\mathtt{Fail} \leftarrow \mathsf{NI\text{–}SS}^H_{\mathscr{A},\Pi_R^{\mathsf{SLC}}}(1^\lambda)] > \mathsf{negl}(\lambda)$.

$\mathsf{NI\text{–}SS}^H_{\mathscr{A},\Pi_R^{\mathsf{SLC}}}(1^\lambda)$

```
1 :  ppm ← Π_R^SLC.Setup(1^λ)
2 :  st ← A^H(1^λ, ppm)
3 :  while st ≠ ⊥ :
4 :    (x, π, Q^A, st) ← A^H(st)
5 :    Response ← ⊥
6 :    if Π_R^SLC.Verify^H(x, π) = 1 :
7 :      w ← Π_R^SLC.Extract(x, π, Q^A)
8 :      if R(x, w) = 0 :
9 :        return Fail
10 :   st ← A^H(st, Response)
11 : return Success
```

**Fig. 7.** Non-Interactive Special Soundness (NI-SS) Game.

### A.6 Parameters for $\mathcal{F}_{\mathtt{CRS}}$

**Definition 22 (CRS Ideal Functionality).** *The ideal functionality* $\mathcal{F}_{\mathtt{CRS}}$ *of a* common reference string (CRS) *for a particular CRS generation mechanism* `GenCRS` *is defined as follows.*

**Query:** *Upon receiving a request* $(\mathtt{Query}, s)$ *from a party* $P = (\mathtt{pid}, \mathtt{sid})$*, first check whether* $\mathtt{sid} = s$*. If it doesn't, output* $\perp$*. Otherwise, if this is the first time that* $(\mathtt{Query}, s)$ *was received, compute* $x$ *according to the algorithm* `GenCRS` *and store the tuple* $(\mathtt{CRS}, s, x)$*. Return* $(\mathtt{CRS}, s, x)$*.*

**Definition 23 (Samplable-Hard Relation).** *A binary $\mathcal{NP}$ relation $S$ is* samplable-hard *with respect to a security parameter $\lambda$ if it has the following properties.*

1. **Sampling a statement-witness pair is easy.** *There exists a sampling algorithm $\kappa_S$ that on input $1^\lambda$ outputs $(x, w)$ such that $S(x, w) = 1$ and $|x| = \mathsf{poly}(\lambda)$.*
2. **Computing a witness from a statement is hard.** *For a randomly sampled statement-witness pair $(x, w) \leftarrow \kappa_S(1^\lambda)$ the probability that an efficient adversary $\mathscr{A}$ can find a valid witness given only the statement is negligible. Formally, for all PPT $\mathscr{A}$,*

$$\Pr[(x, w) \leftarrow \kappa_S(1^\lambda), w' \leftarrow \mathscr{A}(1^\lambda, x, \kappa_S) : (x, w') \in R] \leq \mathsf{negl}(\lambda).$$

**Definition 24 ($\Sigma$-Friendly Relation).** *A* $\Sigma$-friendly relation *$S$ is a binary $\mathcal{NP}$ relation with a corresponding* efficient *$\Sigma$-protocol $\Sigma_S$.*

### A.7 The Adaptive OR-protocol

**Definition 25 (Adaptive OR-Protocol).** *An* OR-protocol *for a relation $R_{OR} = R_0 \vee R_1$ based on adaptive $\Sigma$-protocols $\Sigma_{R_0,\tau_0}$ and $\Sigma_{R_1,\tau_1}$ (Definition 4) is a tuple of procedures $\Sigma_{OR} = (\mathtt{Setup}, \mathtt{Prove}, \mathtt{Verify}, \mathtt{SimSetup}, \mathtt{SimProve}, \mathtt{SimRand}, \mathtt{Extract})$ defined as follows.*

- $\mathtt{PPM} \leftarrow \mathtt{Setup}(1^\lambda)$*: Given a security parameter $1^\lambda$, run $\Sigma_{R_0}.\mathtt{Setup}(1^\lambda)$ to obtain $\mathtt{ppm}_0$ and $\Sigma_{R_1}.\mathtt{Setup}(1^\lambda)$ to obtain $\mathtt{ppm}_1$. Output $\mathtt{PPM} = (\mathtt{ppm}_0, \mathtt{ppm}_1)$.*
- $\Phi \leftarrow \mathtt{Prove}(X, W, Q)$*: Parse $X = (x_0, x_1)$, $W = (w, b)$, and $Q = (r_0, r_1)$, and let $b$ be the bit such that $(x_b, w) \in R_b$. Execute the following:*
  - $\mathtt{Com} \leftarrow \mathtt{Commit}(X, W, Q)$*: $P$ computes $\mathtt{com}_b$ according to $\tau_b.\mathtt{Commit}(x_b, w, r_b)$. $P$ chooses $\mathtt{chl}_{1-b}$ at random and generates $(\mathtt{com}_{1-b}, \mathtt{chl}_{1-b}, \mathtt{res}_{1-b})$ by running $\Sigma_{R_{1-b}}.\mathtt{Simulate}(x_{1-b}, \mathtt{chl}_{1-b})$. $P$ sends $V$ $\mathtt{Com} = (\mathtt{com}_0, \mathtt{com}_1)$.*
  - $\mathtt{CHL} \leftarrow \mathtt{Challenge}(X, \mathtt{Com})$*: $V$ sends $P$ a random $\ell$-bit string $\mathtt{CHL}$.*
  - $\mathtt{Res} \leftarrow \mathtt{Respond}(X, W, \mathtt{Com}, \mathtt{Chl})$*: $P$ sets $\mathtt{chl}_b = \mathtt{CHL} \oplus \mathtt{chl}_{1-b}$ and computes $\mathtt{res}_b$ according to $\tau_b.\mathtt{Respond}(x_b, w, \mathtt{com}_b, \mathtt{chl}_b)$. $P$ sends $(\mathtt{Chl}, \mathtt{Res})$ $= (\mathtt{chl}_0, \mathtt{chl}_1, \mathtt{res}_0, \mathtt{res}_1)$ to $V$.*

  *The output "proof" $\Phi$ is a tuple $(\pi_0, \pi_1, \mathtt{CHL})$, where $\pi_b = (\mathtt{com}_b, \mathtt{chl}_b, \mathtt{res}_b)$.*
- $\{0, 1\} \leftarrow \mathtt{Verify}(X, \Phi)$*: Parse $\Phi$ as $(\pi_0, \pi_1, \mathtt{CHL})$, where $\pi_b = (\mathtt{com}_b, \mathtt{chl}_b, \mathtt{res}_b)$. Execute the following:*
  - $\{0, 1\} \leftarrow \mathtt{Decision}(X, \mathtt{Com}, \mathtt{Chl}, \mathtt{Res})$*: If $\tau_0.\mathtt{Decision}(x_0, \mathtt{com}_0, \mathtt{chl}_0, \mathtt{res}_0)$ $= 1$ and $\tau_1.\mathtt{Decision}(x_1, \mathtt{com}_1, \mathtt{chl}_1, \mathtt{res}_1) = 1$, return 1. Otherwise, return 0.*

  *If $\mathtt{Decision}(X, \mathtt{Com}, \mathtt{Chl}, \mathtt{Res}) = 1$ and $\mathtt{chl}_0 \oplus \mathtt{chl}_1 = \mathtt{CHL}$, output 1 (accept). Otherwise, output 0 (reject).*

- $(\mathtt{PPM}, Z) \leftarrow \mathtt{SimSetup}(1^\lambda)$: *Generate* $(\mathtt{ppm}_0, z_0)$ *by running* $\Sigma_{R_0}.\mathtt{SimSetup}(1^\lambda)$ *and* $(\mathtt{ppm}_1, z_1)$ *by running* $\Sigma_{R_1}.\mathtt{SimSetup}(1^\lambda)$. *Return* $(\mathtt{PPM}, Z)$ *where* $Z = (z_0, z_1)$.
- $\Phi \leftarrow \mathtt{SimProve}(X, Z, \mathtt{CHL})$ : *Parse* $X = (x_0, x_1)$ *and* $Z = (z_0, z_1)$. *Generate* $\mathtt{chl}_0$ *uniformly at random and set* $\mathtt{chl}_1 = \mathtt{chl}_0 \oplus \mathtt{CHL}$. *Obtain* $\pi_0$ *by running* $\Sigma_{R_0}.\mathtt{Simulate}(x_0, \mathtt{chl}_0)$ *and* $\pi_1$ *by running* $\Sigma_{R_1}.\mathtt{Simulate}(x_1, \mathtt{chl}_1)$. *Return* $\Phi = (\pi_0, \pi_1, \mathtt{CHL})$.
- $q_b \leftarrow \mathtt{SimRand}(\mathtt{PPM}, z_s, Z_\pi, X, \Pi, W)$ : *Parse* $\mathtt{PPM} = (\mathtt{ppm}_0, \mathtt{ppm}_1)$, $Z_\pi = (z_{\pi_0}, z_{\pi_1})$, $X = (x_0, x_1)$, $\Phi = (\pi_0, \pi_1)$, *and* $W = (w, b)$. *Compute* $q_b \leftarrow \Sigma_b.\mathtt{SimRand}(\mathtt{ppm}_b, z_b, z_{\pi_b}, x_b, \pi_b, w_b)$ *and return it.*
- $W \leftarrow \mathtt{Extract}(X, \Phi, \Phi')$: *Parse* $X = (x_0, x_1)$, $\Phi = (\pi_0, \pi_1)$, *and* $\Phi' = (\pi'_0, \pi'_1)$. *Obtain* $w_0$ *by running* $\Sigma_{R_0}.\mathtt{Extract}(x_0, \pi_0, \pi'_0)$ *and* $w_1$ *by running* $\Sigma_{R_1}.\mathtt{Extract}(x_1, \pi_1, \pi'_1)$. *Return* $W = (w_0, w_1)$.

### A.8 Required Properties for the Randomized Fischlin Transform

To stop $\mathscr{A}$ from predicting com and querying the RO on $(x, \mathtt{com})$ before the prover does, the com messages of $\Sigma$-protocols under any non-interactive transform in the ROM need entropy that is superlogarithmic in the security parameter.

**Definition 26 (Superlogarithmic Commitment Entropy).** [24] *Let* $\Sigma_R$ *be any* $\Sigma$*-protocol for binary* $\mathcal{NP}$ *relation* $R$ *and template* $\tau$ *as specified in Definition 4.* $\Sigma_R$ *has* superlogarithmic commitment entropy *if for all* $(x, w) \in L_R$, *the min-entropy of* $\mathtt{com} \leftarrow \tau.\mathtt{Commit}(x, w)$ *is superlogarithmic in* $\lambda$.

The quasi-unique responses property [24] says that two proofs with the same commitments and challenges must have different responses.

**Definition 27 (Quasi-Unique Responses).** *A* $\Sigma$*-protocol for relation* $R$ *(Definition 4) has the* quasi-unique responses *property if for any PPT* $\mathscr{A}$, *security parameter* $\lambda$, *and* $(x, \mathtt{com}, \mathtt{chl}, \mathtt{res}, \mathtt{res}') \leftarrow \mathscr{A}(1^\lambda)$, *we have*

$$\Pr[\Sigma_R.\mathtt{Verify}(x, \mathtt{com}, \mathtt{chl}, \mathtt{res}) = \Sigma_R.\mathtt{Verify}(x, \mathtt{com}, \mathtt{chl}, \mathtt{res}') = 1 \wedge \mathtt{res} \neq \mathtt{res}'] \leq \mathsf{negl}(\lambda).$$

The strong special soundness property [26] says that the extractor must still work as long as there is *some* difference between the challenges and responses of two transcripts—in particular, it could be that $\mathtt{chl} = \mathtt{chl}'$, as long as $\mathtt{res} \neq \mathtt{res}'$.

**Definition 28 (Strong Special Soundness).** *A* $\Sigma$*-protocol* $\Sigma_R$ *for relation* $R$ *(Definition 4) has the* strong special soundness *property if the condition* $\mathtt{chl} \neq \mathtt{chl}'$ *in the specification of the* $\Sigma.\mathtt{Extract}$ *algorithm is replaced with the condition* $(\mathtt{chl}, \mathtt{res}) \neq (\mathtt{chl}', \mathtt{res}')$.

**Definition 29 (Required Properties for aFis).** *A* $\Sigma$*-protocol* $\Sigma_R$ *for relation* $R$ *(Definition 4) has the* required properties for the randomized Fischlin transform aFis *if it has the* superlogarithmic commitment entropy property *(Definition 26) and either the* quasi-unique responses property *(Definition 27) or the* strong special soundness property *(Definition 28).*

### A.9 The Adaptive Randomized Fischlin Transform

**Definition 30 (Adaptive Randomized Fischlin Transform).** *Let $\Sigma_{R,\tau}$ be any adaptive $\Sigma$-protocol for relation $R$ (Definition 4) based on protocol template $\tau$ (Definition 3) with the quasi-unique responses (Definition 27) or strong special soundness property (Definition 28). Let $H$ be any random oracle. Then the* randomized Fischlin transform *of $\Sigma_{R,\tau}$, denoted* $\texttt{aFis}$*, is an algorithm that takes $\Sigma_{R,\tau}$ as input and creates a tuple of algorithms $\Pi_R^{\texttt{aFis}} = (\texttt{Setup}^H, \texttt{Prove}^H, \texttt{Verify}^H, \texttt{SimSetup}, \texttt{SimProve}, \texttt{SimRand}, \texttt{Extract})$, defined as follows.*

- $\texttt{ppm} \leftarrow \texttt{Setup}^H(1^\lambda)$ : *$H$ is fixed. Let $b, n, S, t, \ell$ be set according to the Fischlin transform [24] $bn = \omega(\log\lambda)$, $2^{t-b} = \omega(\log\lambda)$, $b, n, t = O(\log\lambda)$ and $b \leq t \leq \ell$, where we use $n$ in place of $r$ repetitions to avoid confusion with the notation for randomness. The compound randomness security parameters $\lambda_R = n(\lambda_r + 2^t t)$ and $\lambda_{R'} = 2^t b + n((\ell - b) + \lambda_r)$, where $\lambda_r$ and $\lambda_{r'}$ are the randomness security parameters of $\Sigma_{R,\tau}$.[8] Output the public parameters $\texttt{ppm} = (\texttt{ppm}_\Sigma, b, n, S, t, \ell, \lambda_R)$, where $\texttt{ppm}_\Sigma \leftarrow \tau.\texttt{Setup}(1^\lambda)$.*
- $(x, \Phi) \leftarrow \texttt{Prove}^H(x, w, R)$ : *Compute the vector of $n$ first messages $\overline{\texttt{com}} = \langle \texttt{com}_1, \ldots, \texttt{com}_n \rangle$ by running $\tau.\texttt{Commit}(x, w, r_i)$ for $1 \leq i \leq n$, where each $r_i$ consumes $\lambda_r$ bits of the random tape $R$. To compute each response $\texttt{res}_i$, test $2^t$ challenges $\texttt{chl}_1, \ldots, \texttt{chl}_{2^t}$ as follows. First, set $\texttt{chl}_j$ to be the next $t$ bits of $R$. Then, repeat $\texttt{res}_j \leftarrow \tau.\texttt{Respond}(x, w, \texttt{com}, \texttt{chl}_j)$ until $H(x, \overline{\texttt{com}}, i, \texttt{chl}_j, \texttt{res}_j) = 0^b$, or else take the minimal over all of the responses. Set the minimal response-producing challenge $\texttt{chl}_j = \texttt{chl}_i$. Finally, return $(x, \Phi)$, where $\Phi = (\pi_1, \ldots, \pi_n)$, and each $\pi_i = (\texttt{com}_i, \texttt{chl}_i, \texttt{res}_i)$.*
- $\{0,1\} \leftarrow \texttt{Verify}^H(x, \Phi)$ : *Parse $\Phi = (\pi_1, \ldots, \pi_n)$. Output 1 (accept) if and only if $\Sigma_R.\texttt{Verify}(x, \pi_i) = 1$ and $\sum_{i=1}^n \texttt{trunc}(H(x, \overline{\texttt{com}}, i, \texttt{chl}_i, \texttt{res}_i)) \leq S$ for $1 \leq i \leq n$. Otherwise, output 0 (reject).*
- $(\texttt{ppm}, z_s) \leftarrow \texttt{SimSetup}(1^\lambda)$ : *Fix $H$ and generate $b, n, S, t$ the same as in $\Pi_R^{\texttt{aFis}}.\texttt{Setup}$. Generate $\texttt{ppm}_\Sigma$ and simulator state information $z_s$ by running $\Sigma_{R,\tau}.\texttt{SimSetup}$. Set $\texttt{ppm} = (\texttt{ppm}_\Sigma, b, n, S, t, \lambda_R)$ and return $(\texttt{ppm}, z_s)$.*
- $(x, \Phi, Z_\Phi) \leftarrow \texttt{SimProve}(x, z_s, R')$ : *For each proof $1 \leq i \leq n$, sample $2^t$ random $t$-bit challenges and $b$-bit strings from $R'$. Let $\mu : \{0,1\}^t \rightarrow \{0,1\}^b$ represent the map between the challenges and the $b$-bit outputs, which are potential truncated outputs of $H$. Let the final challenge for the $i^{th}$ proof $\texttt{chl}_i$ be the first challenge in lexicographic order to map to the minimal response. Run $\Sigma_{R,\tau}.\texttt{SimProve}(x, z_s, \texttt{chl}_i, r_i')$ to obtain $\pi_i = (\texttt{com}_i, \texttt{chl}_i, \texttt{res}_i)$ and $z_{\pi_i}$,*

[8] For each execution of $\Pi_R^{\texttt{aFis}}.\texttt{Prove}$, we will need enough randomness to compute $n$ proofs, each requiring $\lambda_r$ bits to compute the first message and $2^t t$ bits to sample $2^t$ $t$-bit random challenges. For each execution of simulator algorithms $\Pi_R^{\texttt{aFis}}.\texttt{SimProve}$ and $\Pi_R^{\texttt{aFis}}.\texttt{SimRand}$, we will need enough randomness to sample $2^t$ $b$-bit random challenges and $n(\ell - b)$ bits to program the random oracle, as well $n(\lambda_{r'})$ bits to feed into $n$ executions of $\Sigma_{S,\tau}.\texttt{SimRand}$ and $n2^t t$ bits to simulate the challenge selection process.

*where each $r'_i$ is the next $\lambda_{r'_i}$ bits of $R'$. Repeat this process for all $n$ proofs. For each proof, program the output of $H$ on input $(x, \overline{\mathtt{com}}, i, \mathtt{chl}_i, \mathtt{res}_i)$ to be $\mu_i(\mathtt{chl}_i)$. Finally, output the proof tuple $(x, \Phi)$ and auxiliary information $Z_\Phi$, where $\Phi = (\Phi_1, \ldots, \Phi_n)$ and $Z_\Phi = (z_{\pi_1}, \ldots, z_{\pi_n})$.*

- $R \leftarrow \mathtt{SimRand}(\mathtt{ppm}, z_s, Z_\Phi, x, \Phi, w, R')$ : *Parse $\Phi = (\pi_1, \ldots, \pi_n)$ and $Z_\Phi = (z_{\pi_1}, \ldots, z_{\pi_n})$. Reconstruct the proof randomness $R$ corresponding to $\Phi$ as follows. Begin reading $R'$ at the $2^t(t+b)+1^{st}$ bit. Generate the first $n\lambda_r$ bits of $R$ (i.e. the randomness of the first messages) by running $\Sigma_{R,\tau}.\mathtt{SimRand}(z_s, z_{\pi_i}, x, \pi_i, w, r'_i)$ for each $\pi_i$, where each $r'_i$ is the next $\lambda_{r'}$ bits of $R'$. Next, generate the randomness of the challenge selection process as follows. Reset the pointer reading $R'$ back to the beginning of $R'$. For each commitment $\mathtt{com}_i$, use the appropriate segment of $R'$ to reconstruct the map $\mu : \{0,1\}^t \to \{0,1\}^b$ mapping the $2^t$ $t$-bit challenges to $b$-bit outputs. For each challenge $\mathtt{chl}_j$ in the list of $2^t$ challenges, compute the response $\mathtt{res}_j \leftarrow \tau.\mathtt{Respond}(x, w, \mathtt{com}_i, \mathtt{chl}_j)$ and program the output of $H$ on input $(x, \overline{\mathtt{com}}, i, \mathtt{chl}_j, \mathtt{res}_j)$ to be $\mu_i(\mathtt{chl}_j)$, then concatenate $\mathtt{chl}_j$ to $R$. Once the process as been completed for all commitments and challenges, return $R$.*

- $w \leftarrow \mathtt{Extract}(X, \Phi, \mathcal{Q}_{\mathscr{A}})$ : *Parse $\Phi = (\pi_1, \ldots, \pi_n)$ and each $\pi_i = (\mathtt{com}_i, \mathtt{chl}_i, \mathtt{res}_i)$. Given a list $\mathcal{Q}_{\mathscr{A}}$ the adversary's queries to $H$, search for two queries $(x, \overline{\mathtt{com}}, i, \mathtt{chl}_i, \mathtt{res}_i)$ and $(x, \overline{\mathtt{com}}, i, \mathtt{chl}'_i, \mathtt{res}'_i)$ such that $(\mathtt{chl}_i, \mathtt{res}_i) \neq (\mathtt{chl}'_i, \mathtt{res}'_i)$ and $\Sigma_R.\mathtt{Verify}(x, \pi_i) = \Sigma_R.\mathtt{Verify}(x, \pi'_i) = 1$. If no such queries exist, output $\mathtt{Fail}$. Otherwise, obtain $w$ by running $\Sigma_R.\mathtt{Extract}(x, \pi, \pi')$.*

### A.10 Efficiency of the Randomized Fischlin Transform

There are several metrics by which to evaluate the efficiency of the randomized Fischlin transform. We focus on *communication* and *computational complexity*.

To understand the communication and computation costs, consider the parameters of the randomized Fischlin transform $b, n, S, t$ with respect to the security parameter $\lambda$ and (underlying $\Sigma$-protocol's) challenge length $\ell = O(\log \lambda)$. Here $b$ represents the number of bits output by the random oracle $H$, $n$ the number of repetitions (of the underlying $\Sigma$-protocol) and therefore total proof transcripts output by the prover, $S$ the maximum size of the sum over all outputs of $H$ on input each of the $n$ proof transcripts, and $t$ the number of bits in each test challenge. These quantities are related as follows: $b, r, t = O(\log \lambda)$, $b \leq t \leq \ell$, $br = \omega(\log \lambda)$, and $2^{t-b} = \omega(\log \lambda)$ [24]. The incurred communication cost is therefore logarithmic in the security parameter—precisely, $n$ times the size of the underlying $\Sigma$-protocol proof transcript (similarly, the verifier will need to run the verification algorithm of the $\Sigma$-protocol $n$ times).

With respect to the prover's computational overhead, Fischlin proves that any NISLE ZKPoK (with a non-programming extractor) must query $H$ a number of times that is *at least* superlogarithmic in the security parameter—that is, the computational complexity of *any* NISLE ZKPoK prover is lower-bounded

by $\omega(\log \lambda)$ (Proposition 2 in Section 3.4 [24]).[9] Kondi and shelat tighten this bound, and demonstrate that the standard (and randomized) Fischlin transform is a factor of $\frac{n}{(n!)^{1/n}}$ worse on average than an optimally efficient prover (Lemma 5.1 and Claim 5.3, respectively [26]). They additionally introduce a "drop-in replacement" for Fischlin's minimal hash output predicate in the form of a hash *collision* predicate (i.e. rather than finding a transcript that maps to a minimal hash output, the prover finds two transcripts that map to the same hash output). If the underlying $\Sigma$-protocol has properties called $n+1$-special soundness and $n$-simulatability (which the adaptive $\Sigma$-protocol examples given in Section 6 do), the prover can apply a *multi-collision* predicate to obtain optimal efficiency (Section 5.3 [26]). For comparison, the cut-and-choose method of straight-line extraction due to Pass [31] incurs a minimal computational overhead of $\lambda^2$ [26].

### A.11 Full Proof of Theorem 4

**Recall Theorem 4:** Let $\Sigma_{R,\tau}$ be an adaptive $\Sigma$-protocol based on protocol template $\tau$ for relation $R$ (Definition 4) with the required properties for `rFis` (Definition 29 in A.9), and $H : \{0,1\}^* \to \{0,1\}^b$ be any random oracle. Then the adaptive randomized Fischlin transform `aFis` (Definition 30 in A.9) is an adaptive straight-line compiler for $\Sigma_{R,\tau}$ (Definition 6) in the random-oracle model.

*Proof.* Recall that an adaptive straight-line compiler according to our definition must create protocols that are overwhelmingly complete, adaptive non-interactive multiple special honest-verifier zero-knowledge (adNIM-SHVZK), and adaptive non-interactive special simulation-sound (adNI-SSS). First, note that since the specification of $\Pi_R^{\texttt{aFis}}$.`Prove` does not functionally change between `rFis` and `aFis` (as our explicit treatment of the prover's randomness is syntactic), `aFis` is as complete as `rFis` [24,26]. We proceed by contrapositive to show that if $\Pi_R^{\texttt{aFis}}$ is not additionally adNIM-SHVZK and adNI-SSS, then the underlying $\Sigma$-protocol $\Sigma_R$ cannot be regular adaptive special honest-verifier zero-knowledge (adSHVZK), contradicting the assumption in the theorem statement.

We begin by constructing a reduction $\mathscr{B}$ that uses an algorithm $\mathscr{A}$ that can win the adNIM-SHVZK game (Definition 8) parameterized over $\Pi_R^{\texttt{aFis}}$ with non-negligible advantage as a black-box in order to win the adSHVZK game (Definition 5) parameterized over $\Sigma_R$ with non-negligible advantage. $\mathscr{B}$ proceeds as follows. When it obtains `ppm` from the adSHVZK challenger, $\mathscr{B}$ passes `ppm` to $\mathscr{A}$. Note that the adSHVZK challenger is expecting exactly one `Prove` query (i.e. it is not *multi*-adaptive), so we modify $\mathscr{A}$ to distinguish two hybrids $i$ and $i+1$, where in the $i^{th}$ hybrid, the first $i$ proofs and random coins are according to $\Pi_R^{\texttt{aFis}}$.`Prove` using randomness $r \in \{0,1\}^{\lambda_r}$, and the $i+1^{st}$ onward are according to $\Pi_R^{\texttt{aFis}}$.`SimProve` and $\Pi_R^{\texttt{aFis}}$.`SimRand` using randomness $r' \in \{0,1\}^{\lambda_{r'}}$. $\mathscr{B}$ models the RO with an initially empty list $L \leftarrow \perp$. To answer the first $i$ queries, $\mathscr{B}$ first samples $f \leftarrow_{\$} F$ then runs $H_f$ on the query, storing the

[9] This bound does not capture any efficiency loss due to repeating expensive operations in the underlying $\Sigma$-protocol template—it is meant to capture the additional cost incurred by the transform itself.

input-output pair in $L$. From the $i+1^{st}$ query onward, $\mathscr{B}$ programs the RO using the interface $\mathtt{Prog}_L$ according to $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimRand}$.

For the first $i$ queries $(\mathtt{Prove}, x, w)$ from $\mathscr{A}$, $\mathscr{B}$ samples randomness $Q \leftarrow \{0,1\}^{\lambda_Q}$, runs $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}(x, w, Q)$, and returns $(\Phi, Q)$ to $\mathscr{A}$. On $\mathscr{A}$'s $i+1^{st}$ query $(\mathtt{Prove}, x^*, w^*)$, $\mathscr{B}$ passes $(\mathtt{Prove}, x^*, w^*)$ to its challenger and receives $(\pi^*, r^*)$ for $\pi^* = (\mathtt{com}^*, \mathtt{chl}^*, \mathtt{res}^*)$ that is either the result of running $\Sigma_R.\mathtt{Prove}$ on randomness $r^*$ or the result of running $\Sigma_R.\mathtt{SimProve}$ and $\Sigma_R.\mathtt{SimRand}$. Note that that $\mathscr{A}$ is expecting more proofs than just $\pi^*$, as the output of $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ is actually a tuple of proofs $\Phi = \pi_1, \ldots, \pi_n$. We therefore modify $\mathscr{A}$ again to distinguish the hybrids $i+1, j$ from $i+1, j+1$, where in the $i+1, j^{th}$ hybrid, the first $j$ proofs of the challenge proof tuple $\Phi^* = \pi_1, \ldots, \pi_n$ are computed according to $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ each with randomness $Q \in \{0,1\}^{\lambda_Q}$, and the $i+1, j+1^{st}$ through the $n^{th}$ proof are according to $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimRand}$ with randomness $Q' \in \{0,1\}^{\lambda_{Q'}}$. $\mathscr{B}$ constructs the entire $i+1^{st}$ hybrid proof tuple $\Phi_{i+1} = \Phi_1, \ldots, \Phi_n$ from the challenge query as follows.

Let the $j+1^{st}$ commitment $\mathtt{com}_{j+1} = \mathtt{com}^*$, and $\pi_{j+1} = (\mathtt{com}^*, \mathtt{chl}^*, \mathtt{res}^*)$. Since it must have the entire tuple $\mathtt{com}_1, \ldots, \mathtt{com}_n$ of commitments before running $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$, $\mathscr{B}$ starts by running the steps of $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ $n-(j+2)$ times—that is, to compute each proof $\pi_l$ for $j+2 \leq l \leq n$, $\mathscr{B}$ samples $2^t$ random $t$-bit challenges and $b$-bit strings, maps the challenges to the strings, picks $\mathtt{chl}_l$ to be the first challenge in lexicographic order to map to the minimal response, and computes $\pi_l \leftarrow \Sigma_{R,\tau}.\mathtt{SimProve}(x, z_s, \mathtt{chl}_l, r'_l)$, where $r'_l$ comes from $\mathscr{B}$'s random tape. (It will later program the outputs of $H$ to remain consistent with these proofs, but first it needs to generate the rest of the commitment vector.)

$\mathscr{B}$ then samples each $r_1, \ldots r_j$ uniformly at random from $\{0,1\}^{\lambda_r}$ and runs $\tau.\mathtt{Commit}$ $j$ times to obtain $\mathtt{com}_1, \ldots, \mathtt{com}_j$ and produce the full commitment vector $\overline{\mathtt{com}} = \mathtt{com}_1, \ldots, \mathtt{com}_n$. $\mathscr{B}$ computes each proof $\pi_l$ for $1 \leq l \leq j$ by running the steps of $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ $j$ times—that is, $\mathscr{B}$ samples $j$ $\lambda_r$-bit first messages $\mathtt{com}_1, \ldots, \mathtt{com}_j$ and for each $\mathtt{com}_l$ samples $j2^t$ $t$-bit challenges, computes $\mathtt{res}_m \leftarrow \tau.\mathtt{Respond}(x, w, \mathtt{com}_l, \mathtt{chl}_m)$ for each of the $1 \leq m \leq 2^t$ challenges, and chooses the proof tuple such that the value returned by $H(x, \overline{\mathtt{com}}, l, \mathtt{chl}_m, \mathtt{res}_m)$ is minimal over all $\mathtt{chl}_1, \ldots, \mathtt{chl}_{2^t}$.

$\mathscr{B}$ generates the prover's hybrid random tape $Q^*$ as follows. The first $j\lambda_r$ bits are the $j$ $\lambda_r$-bit samples that $\mathscr{B}$ made while computing proofs according to $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$. The proceeding $\lambda_r$ bits are those returned by the adNIM-SHVZK challenger, $r^*$. To obtain the next $(n-(j+2))\lambda_r$ bits, $\mathscr{B}$ runs $\Sigma_{R,\tau}.\mathtt{SimRand}$ on input each $\pi_l, r'_l$ for $j+2 \leq l \leq n$, where $r'_l$ is the same section of $\mathscr{B}$'s random tape that it used in running $\Sigma_{R,\tau}.\mathtt{SimProve}$ to obtain $\pi_l$. To simulate the "random challenge selection" section of $Q^*$, $\mathscr{B}$ first concatenates (in order) all of the challenges sampled while computing $\pi_1, \ldots, \pi_j$. For $\mathtt{chl}_{j+1}$ (i.e. $\mathtt{chl}^*$, which was produced by the adNIM-SHVZK challenger rather than being sampled by $\mathscr{B}$), $\mathscr{B}$ uses fresh randomness to sample $2^t - 1$ $t$-bit challenges and $2^t$ $b$-bit outputs of $H$, inserts $\mathtt{chl}_{j+1}$ as first challenge in lexicographic order to map to the minimal $b$-bit output value, then concatenates all $2^t$ challenges (in order)

to $Q^*$. Finally, $\mathscr{B}$ concatenates (again in order) all of the challenges it sampled while computing $\pi_{j+2}, \ldots, \pi_n$.

All that remains in $\mathscr{B}$'s simulation is to make sure $H$ is consistent with the proofs and randomness it has generated. For all proofs $\pi_l$ where $j+1 \leq l \leq n$, $\mathscr{B}$ programs the output of $H$ on input $(x, \overline{\texttt{com}}, l, \texttt{chl}_m, \texttt{res}_m)$ to be the corresponding $b$-bit output in the challenge-to-output mappings for all challenges and responses $1 \leq m \leq 2^t$.

For the analysis, consider that $\mathscr{B}$'s execution of the proofs $\pi_1, \ldots, \pi_j$ and $\pi_{j+2}, \ldots, \pi_n$ and corresponding randomness is exactly what $\mathscr{A}$ expects in either hybrid. It remains to show that the challenge proof $\pi_{j+1}$ and corresponding randomness is also formatted according to what $\mathscr{A}$ expects. If the adNIM-SVHZK challenger is playing with bit $b = 0$ and generated $(\pi^*, r^*)$ according to $\Pi_R^{\texttt{aFis}}.\texttt{Prove}$ and randomness $r$, then $\pi_{j+1}$ and the "first message" section of $Q^*$ corresponding to $r_j$ (i.e. the randomness used to generate $\texttt{com}_{j+1}$) will be exactly what $\mathscr{A}$ expects from the $i, j+1^{st}$ hybrid. If, on the other hand, $\pi^*, r^*$ were generated according to $\Pi_R^{\texttt{aFis}}.\texttt{SimProve}$ and $\Pi_R^{\texttt{aFis}}.\texttt{SimRand}$, then $\mathscr{A}$'s view of $\pi_{j+1}$ and $r_j$ will be according to hybrid $i, j$.

Finally, we must argue that $\mathscr{A}$'s view of the "challenge-selection" section of $Q^*$ is formatted how $\mathscr{A}$ expects (since according to the hybrid experiment, $\mathscr{A}$ is expecting a "simulated" challenge selection process only when $\pi^*, r^*$ were generated according to $\Pi_R^{\texttt{aFis}}.\texttt{SimProve}$ and $\Pi_R^{\texttt{aFis}}.\texttt{SimRand}$). The statistical indistinguishability of $\mathscr{B}$'s simulation for the challenge-selection section phase follows from the fact that all of the sampled challenges are identically distributed—they are all sampled uniformly at random from $\{0,1\}^t$. Likewise, the programmed outputs of $H$ are all sampled uniformly at random from $\{0,1\}^b$. Therefore, if $\mathscr{B}$ outputs 0 when $\mathscr{A}$ outputs "Hybrid $i, j+1$" and 1 when $\mathscr{A}$ outputs "Hybrid $i, j$," it succeeds with the same probability as $\mathscr{A}$. Because $i$ is bounded by the number of prove queries and $j$ is bounded by the parameter $n$, the number of hybrids is bounded by some polynomial in the security parameter, and $\mathscr{B}$'s summed advantage over all of the hybrids is negligible.

We now use the fact that $\Pi_R^{\texttt{aFis}}$ is adaptive NIM-SHVZK to argue that $\Pi_R^{\texttt{aFis}}$ must also be adaptive NI-SSS. Consider a new reduction $\mathscr{B}$ that uses an adversary $\mathscr{A}$ that can win the adNI-SSS game (Definition 9), as well as a second adversary $\mathscr{A}'$ that can win the regular NI-SS game (Definition 21 in Appendix A.5) with non-negligible advantage as black boxes to win the adNIM-SHVZK game (Definition 8). $\mathscr{B}$ forwards all of $\mathscr{A}$'s $\texttt{Prove}$ queries to and from the adNIM-SHVZK challenger, and all of $\mathscr{A}$'s random oracle queries to and from $H$. Similarly, $\mathscr{B}$ forwards all of $\mathscr{A}'$'s oracle queries to and from $H$. Whenever $\mathscr{A}$ produces a fresh (non-simulated) proof $(x, \Phi)$ such that $\Pi_R^{\texttt{aFis}}.\texttt{Verify}(x, \Phi) = 1$, $\mathscr{B}$ computes $w \leftarrow \Pi_R^{\texttt{aFis}}.\texttt{Extract}(x, \Phi, \mathcal{Q}_{\mathscr{A}})$ and checks whether $R(x, w) = 0$, which happens with non-negligible probability by assumption. $\mathscr{B}$ does the same for $\mathscr{A}'$'s proofs $(x', \Phi')$. (The argument that there must be sufficient queries in $\mathcal{Q}_{\mathscr{A}}$ or $\mathcal{Q}_{\mathscr{A}'}$ to invoke $\Pi_R^{\texttt{aFis}}.\texttt{Extract}$ in the first place is identical to the arguments given by Fischlin [24] and Kondi and shelat [26]; in brief, the quasi-unique responses property (Definition 27 in A.8) (resp. strong special soundness prop-

erty (Definition 28 in A.8) stops $\mathscr{A}$ from being able to "tweak" an old proof to create a forgery, for instance by changing one response.)

If the adNIM-SVHZK challenger is playing with bit $b = 1$ and the proofs and randomness being passed to $\mathscr{A}$ are according to $\Pi_R^{\mathtt{aFis}}.\mathtt{SimProve}$ and $\Pi_R^{\mathtt{aFis}}.\mathtt{SimRand}$, this is exactly what $\mathscr{A}$ expects from the adNI-SSS game, and $\mathscr{B}$'s advantage is the same as $\mathscr{A}$'s. If the adNIM-SHVZK challenger is playing with bit $b = 0$, then $\mathscr{A}$'s advantage reduces to $\mathscr{A}'$'s advantage in the standard NI-SS game (since $\mathscr{A}'$ can always generate honest proofs and randomness according to $\Pi_R^{\mathtt{aFis}}.\mathtt{Prove}$ itself). Assume for a contradiction that there is a non-negligible difference between the extraction failures produced by $\mathscr{A}$ when $b = 1$ and $\mathscr{A}/\mathscr{A}'$ when $b = 0$. If $\mathscr{B}$ observes a difference in output betwen $\mathscr{A}$ and $\mathscr{A}'$, $\mathscr{B}$ knows $\mathscr{A}$ has an extra advantage due to seeing simulated proofs and randomness and outputs 1; otherwise if there is no difference, $\mathscr{B}$ outputs 0. Therefore, the difference must be negligible, implying that as long as $\Pi_R^{\mathtt{aFis}}$ is adNIM-SHVZK (as proven in the preceeding paragraphs) and NI-SS (which follows directly from the special soundness of $\Sigma_{R,\tau}$ and the randomness of $H$), $\Pi_R^{\mathtt{aFis}}$ must be adNI-SSS.

We have shown that the tuple $\Pi_R^{\mathtt{aFis}} \leftarrow \mathtt{aFis}(\Sigma_{R,\tau})$ is overwhelmingly complete, adNIM-SVHZK, and adNI-SSS, completing the proof that $\mathtt{aFis}$ is an adaptive SLC. □

### A.12 Full Proof of Theorem 7

**Recall Theorem 7:** A proof of knowledge of equality of $n$ representations can be described as a simple adaptive $\Sigma$-protocol (Definition 12).

*Proof.* We recall the setup. The public parameters $\mathtt{ppm}$ are $(\mathbb{G}, q, \overline{g}_1, \ldots, \overline{g}_n)$, where $\mathbb{G}$ is a cyclic group of prime order $q$, and each $\overline{g}_i$ is a vector of $m$ generators $g_{i,1}, \ldots, g_{i,m}$. The statement and first-message spaces are $\mathcal{X} = \mathcal{A} = \mathbb{G}^n$, the witness, randomness, and response spaces are $\mathcal{W} = \mathcal{R} = \mathcal{B} = \mathbb{Z}_q^m$, and the challenge space is $\mathcal{C} = \mathbb{Z}_q^*$. The evaluation function $\mathtt{F} : \mathbb{Z}_q^m \to \mathbb{G}^n$ is defined such that

$$\mathtt{F}(w) = \mathtt{F}(w_1, \ldots, w_m) = \prod_{i=1}^{m} w_i \cdot g_{1,i}, \ldots, \prod_{i=1}^{m} w_i \cdot g_{n,i}.$$

The operator $\cdot$ is defined, component-wise, the same as in the proof of Theorem 6.

The prover's first message $\mathtt{com} = \mathtt{F}(r)$ becomes

$$\mathtt{F}(r_1, \ldots, r_m) = \prod_{i=1}^{m} w_i \cdot g_{1,i}, \ldots, \prod_{i=1}^{m} w_i \cdot g_{n,i}$$

for $r_1, \ldots, r_m \leftarrow_{\$} \mathbb{Z}_q$. We can therefore write $\mathtt{com} = \langle \mathtt{com}_1, \ldots, \mathtt{com}_n \rangle$. The challenge is a uniformly random element $\mathtt{chl} \leftarrow_{\$} \mathbb{Z}_q^*$, and the prover's response is $\mathtt{res} = \langle \mathtt{res}_1, \ldots, \mathtt{res}_m$, where each $\mathtt{res}_i = \mathtt{chl} \cdot w_i + r_m$. The verifier accepts the proof $(\mathtt{com}, \mathtt{chl}, \mathtt{res})$ if and only if

$$\mathtt{F}(\mathtt{res}) = \mathtt{F}(\mathtt{res}_1, \ldots, \mathtt{res}_m) = \prod_{i=1}^{m} \mathtt{res}_i \cdot g_{1,i}, \ldots, \prod_{i=1}^{m} \mathtt{res}_i \cdot g_{n,i} =$$

$$\prod_{i=1}^{m}(\mathtt{chl}\cdot w_i + r_i)\cdot g_{1,i},\ldots,\prod_{i=1}^{m}(\mathtt{chl}\cdot w_i + r_i)\cdot g_{n,i} =$$

$$\mathtt{chl}\cdot \mathtt{F}(w) + \mathtt{F}(r) = \mathtt{chl}\cdot x + \mathtt{com}.$$

The `SimProve` algorithm samples a random $m$-length response vector from $\mathbb{Z}_q^m$ and computes each component of the $n$-length first message vector $\mathtt{com}_1,\ldots,\mathtt{com}_n$ as

$$\mathtt{com} = \mathtt{F}(\mathtt{res} - \mathtt{chl}\cdot w) = \prod_{i=1}^{m}(\mathtt{res}_i - \mathtt{chl}\cdot w_i)\cdot g_{1,i},\ldots,\prod_{i=1}^{m}(\mathtt{res}_i - \mathtt{chl}\cdot w_i)\cdot g_{n,i},$$

and each random value $r_i$ output by `SimRand` is computed $\mathtt{res}_i - (\mathtt{chl}\cdot w_i)$ for $1 \le i \le m$. Similarly, each witness $w_i$ is extracted by computing $(\mathtt{res}_i - \mathtt{res}_i')\cdot(\mathtt{chl} - \mathtt{chl}')^{-1}$, where $(\mathtt{chl} - \mathtt{chl}') \neq 0$ is again invertible in $\mathbb{Z}_q^*$.

The pseudo-module homomorphism and smoothness properties follow from the component-wise application of the arguments from Theorem 6:

$$\mathtt{F}(\mathtt{chl}\cdot w + r) = \mathtt{F}(\mathtt{chl}\cdot w_1,\ldots,w_m + r_1,\ldots,r_m) =$$

$$\prod_{i=1}^{m}(\mathtt{chl}\cdot w_i + r_i)\cdot g_{1,i},\ldots,\prod_{i=1}^{m}(\mathtt{chl}\cdot w_i + r_i)\cdot g_{n,i} =$$

$$\prod_{i=1}^{m}(\mathtt{chl}\cdot w_i)\cdot g_{1,i} + r_i\cdot g_{1,i},\ldots,\prod_{i=1}^{m}(\mathtt{chl}\cdot w_i)\cdot g_{n,i} + r_i\cdot g_{n,i} =$$

$$\mathtt{chl}\cdot\prod_{i=1}^{m} w_i\cdot g_{1,i} + r_i\cdot g_{1,i},\ldots,\mathtt{chl}\cdot\prod_{i=1}^{m} w_i\cdot g_{n,i} + r_i\cdot g_{n,i} =$$

$$\mathtt{chl}\cdot\mathtt{F}(w) + \mathtt{F}(r).$$

Finally, since each $r_i \leftarrow \mathbb{Z}_q$ and $r_i\cdot g_{1,i}$ is a uniformly random element in $\mathbb{G}$, $\mathtt{F}(r) = \mathtt{F}(r_1,\ldots,r_m) = \prod_{i=1}^{m} r_i\cdot g_{1,i},\ldots,\prod_{i=1}^{m} r_i\cdot g_{n,i}$ is distributed uniformly over $\mathbb{G}^n$. □