Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and Leah Namisa Rosenbloom

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

11 June 2022

Abstract. Numerous cryptographic applications require efficient non-interactive zero-knowledge proofs of knowledge (NIZKPoK) as a building block. Typically they rely on the Fiat-Shamir heuristic to do so, as security in the random-oracle model is considered good enough in practice. However, there is a troubling disconnect between the stand-alone security of such a protocol and its security as part of a larger, more complex system where several protocols may be running at the same time. Provable security in the general universal composition (GUC) model of Canetti et al. is the best guarantee that nothing will go wrong when a system is part of a larger whole, even when all parties share a common random oracle. In this paper, we prove the minimal necessary properties of generally universally composable (GUC-secure) NIZKPoK in any global random-oracle model, and show how to achieve efficient and GUC-secure NIZKPoK in both the restricted programmable and restricted observable (non-programmable) global random-oracle models.

1 Introduction

Non-interactive zero-knowledge proofs of knowledge (NIZKPoK) [5,28,43] form the basis of many cryptographic protocols that are on the cusp of widespread adoption in practice. For example, the Helios voting system [1] and other efficient systems employing cryptographic shuffles [47] use zero-knowledge proofs of knowledge to ensure that each participant in the system correctly followed the protocol and shuffled or decrypted its inputs correctly. Anonymous e-cash [12] and e-token [11] systems use them to compute proofs of validity of an e-coin or e-token. In group signatures [18,2] they are used to ensure that the signer is in possession of a group signing key. In anonymous credential constructions [13,14], they are used to ensure that the user identified by a given pseudonym is in possession of a credential issued by a particular organization.

The non-interactive aspect of NIZKPoK is especially important to most of these applications—it enables a prover to form a proof of some attribute for a *general* verifier rather than forcing the prover to talk to each verifier individually, which is inefficient in most cases and infeasible for some applications. It is also extremely important that the NIZKPoK be efficient. Thus, the constructions cited above use efficient Σ-protocols [26] made non-interactive via the Fiat-Shamir heuristic [29] to instantiate the NIZKPoK in the random-oracle model

(ROM) [3]. Recall that a Σ-protocol for a relation R is, in a nutshell, a $(1-\mathsf{negl})$-sound honest-verifier three-move proof system in which the single message from the verifier to the prover is a random ℓ-bit string. The Fiat-Shamir transform makes the proof system non-interactive by replacing the message from the verifier with the output of a random oracle (RO).

Recently, a better understanding of how badly such NIZKPoK fare in the *concurrent* setting emerged [45,27,4,39]. Allowing for secure concurrent executions is of vital importance for the real-world application of any of the cryptographic protocols mentioned above, and especially for distributed protocols. But Drijvers et al. [27] demonstrated subtleties in the proofs of security for concurrent protocol executions that often go undetected, leaving building-block cryptographic protocols vulnerable to attacks like Wagner [45] and Benhamouda et al.'s exploitation of the ROS problem [4].

One way to circumvent the unique subtleties of composing cryptographic primitives is to prove that each primitive is *universally composable* using Canetti's universal composition (UC) framework [19]. In the UC framework, the security of a particular session of a protocol is analyzed with respect to an environment, which represents an arbitrary set of concurrent protocols. The environment in the UC framework can talk to and collude with the traditional "adversary" in cryptographic protocols, directing it to interfere with the protocol. However, the original UC framework did not provide a mechanism for parties in different settings to use a shared global functionality, for instance a shared RO or common reference string (CRS). In real-world applications, it is virtually guaranteed that parties will share setup and state between sessions.

To address the issue of shared state and concurrency in the UC framework, Canetti, Dodis, Pass, and Walfish developed the *general* UC (GUC) framework, which considers "global" functionalities $\mathcal{G}$ that can be queried by any party in any session at any time, including the environment [20]. Canetti, Jain, and Scafuro later showed several practical applications of the GUC framework with an restricted observable global RO $\mathcal{G}_{\mathtt{roRO}}$ as the only trusted setup. They include commitment, oblivious transfer, and secure function evaluation protocols, all GUC-secure in the $\mathcal{G}_{\mathtt{roRO}}$-hybrid model [22]. Building on Canetti et al.'s framework, Camenisch, Drijvers, Gagliardoni, Lehmann, and Neven developed a restricted *programmable* observable global RO, denoted $\mathcal{G}_{\mathtt{rpoRO}}$, that allows for more efficient GUC-secure commitments in the $\mathcal{G}_{\mathtt{rpoRO}}$-hybrid model [10].

Thus, the $\mathcal{G}_{\mathtt{rpoRO}}$- and $\mathcal{G}_{\mathtt{roRO}}$-hybrid models are attractive ones for constructing and analyzing practical and provably secure non-interactive zero-knowledge proofs. Obtaining an efficient NIZKPoK (for a relation R) in the $\mathcal{G}_{\mathtt{rpoRO}}$-hybrid model from an efficient Σ-protocol (for the same relation) is a natural goal. We show that it is possible to transform *any* Σ-protocol into a GUC-secure NIZKPoK in the (programmable) $\mathcal{G}_{\mathtt{rpoRO}}$-hybrid model using an algorithm called a straight-line compiler, which we develop from the definition of NIZKPoK with online extractors due to Fischlin [30]. While the programming property of $\mathcal{G}_{\mathtt{rpoRO}}$ is helpful in proving security, it also localizes aspects of the global RO by providing a programming verification interface that concurrent protocols cannot access. It is unclear how localized interfaces that are vital to the security of component

protocols would impact the security of composed protocols, especially when composing protocols that are provably secure in less restrictive models.

Therefore, we also consider NIZKPoK in the less restrictive (non-programmable) $\mathcal{G}_{\texttt{roRO}}$-hybrid model, where $\mathcal{G}_{\texttt{roRO}}$'s interfaces are completely public. Unfortunately, Pass [40] and Canetti et al. [22] point out that it is not possible to construct ZKPoK in less than two rounds of communication using *only* a global functionality, because there is no way for the simulator in the security experiment to exercise control over it. We introduce a new model called the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model, in which participants have access to a trusted common reference string (CRS) functionality. Similar to the programming interface, the CRS functionality is localized such that concurrent protocols do not have access to the local session's CRS generation mechanism. However, unlike localizing aspects of the global RO, localizing aspects of the protocol will not impact the modeling of other protocols that may be composed with our GUC-secure NIZKPoKs in the future. Fortunately, protocol participants can compute such a trusted CRS for a one-time cost at the beginning of the session using only $\mathcal{G}_{\texttt{roRO}}$, due to Canetti et al.'s GUC-secure non-interactive secure computation (NISC) protocol [22]. We prove that any straight-line compiler in conjunction with our new construction, which uses a special type of Σ-protocol called an OR-protocol [26,24], is sufficient to transform any Σ-protocol into a GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model.

We realize our GUC transforms for Σ-protocols using Kondi and shelat's randomized version of the Fischlin transform [36,30], demonstrating that it is possible to construct *efficient* GUC-secure NIZKPoK from a broad class of Σ-protocols in both the $\mathcal{G}_{\texttt{rpoRO}}$ and $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid models. Along the way, we uncover theoretical results and observations that may be of independent interest. First, that straight-line extractors afford strong security guarantees: because they work exclusively using information the adversary already knows, it is possible to compose them with other building blocks such as zero-knowledge simulators without compromising the security of the overall system. This "decoupling" property [30], and security properties of non-rewinding extractors in general, are of interest in the quantum random-oracle model (QROM), where rewinding is tricky because of the no-cloning theorem [46,35,44]. Second, we prove that there are several security properties that are *strictly necessary* for a protocol to have in order for it to be a GUC-secure NIZKPoK in any global ROM. Interestingly, a straight-line compiler—and in particular the randomized Fischlin transform—is uniquely positioned to realize these properties. It is the subject of future work to explore whether other mechanisms of straight-line extraction [17,40,35,44] are sufficient to bootstrap Σ-protocols into GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$- or $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid models, a different global ROM, or the QROM.

Organization. In the remainder of the introduction, we provide general background information on Σ-protocols, the GUC model, the global ROM(s), and straight-line extraction. In Section 2, we give formal definitions of Σ-protocols and straight-line compilers. Section 3 contains definitions of GUC-security in various global ROMs and a proof that any NIZKPoK that is GUC-secure in

any global ROM must have the security properties afforded by straight-line compilers. In Section 4, we prove that any straight-line compiler is sufficient to transform any Σ-protocol into a GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model, and in Section 5 we prove that any straight-line compiler in conjunction with our OR-protocol construction is sufficient to complete the transform in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model. Finally in Section 6, we leverage the randomized Fischlin transform to efficiently realize our constructions in both global ROMs.

Σ-Protocols. A Σ-protocol for a binary $\mathcal{NP}$ relation R is a three-round, public-coin proof system. On input x and w such that $(x, w) \in R$, the prover generates his first message $\texttt{com}$ (in the literature on Σ protocols, this first message is often referred to as a "commitment"). In response, the honest verifier sends a unique ℓ-length *random* "challenge" $\texttt{chl}$ to the prover. Finally, the prover "responds" with a value $\texttt{res}$. The resulting transcript $(\texttt{com}, \texttt{chl}, \texttt{res})$ is then fed to a verification algorithm that determines whether the verifier accepts or rejects.

Σ-protocols must additionally satisfy three properties. First, they must satisfy *completeness*: if the prover has a valid witness and both parties engage in the protocol honestly, the verifier always accepts. Next, they must be *special honest-verifier zero-knowledge*: there must exist a simulator algorithm that on input x and $\texttt{chl} \in \{0,1\}^\ell$ outputs an accepting transcript $(\texttt{com}, \texttt{chl}, \texttt{res})$ for x such that, if $\texttt{chl}$ was chosen uniformly at random, $(\texttt{com}, \texttt{chl}, \texttt{res})$ is statistically indistinguishable from that output by an honest verifier on input x. Finally, they must have *special soundness*: if there are two accepting transcripts for any statement with the same commitment $\texttt{com}$ but different challenges, there exists an extractor algorithm that can produce a valid witness from the transcripts. The stronger version of soundness, special *simulation* soundness, says that special soundness must still hold even if an adversary has oracle access to the simulator.

The Σ-protocol format captures many practical zero-knowledge proof systems. For example, Wikström [47] shows Σ-protocols for proving a rich set of relations between ElGamal ciphertexts, which in turn allow proving that a set of ciphertexts was shuffled correctly; similar protocols exist for Paillier ciphertexts [23,17]. A robust body of literature exists giving Σ-protocols for proving that values committed using Pedersen [41] and Fujisaki-Okamoto [32] commitments satisfy general algebraic and Boolean circuits [8,15,16] and lie in certain integer ranges [6,37]. For all the Σ-protocols listed above, the size and complexity of the proof system is a $O(1)$ factor of the complexity of verifying the underlying relation $R(x, w)$, making Σ-protocols extremely desirable in practice.

Σ-protocols are also the most efficient technique to achieve zero-knowledge proofs of knowledge of a commitment opening in the lattice setting [38,25], where the complexity grows by a factor of $O(k)$ in order to achieve soundness $(1 - 2^{-k})$. Thus, for all the relations R cited above, our results immediately yield the most efficient known GUC-secure NIZKPoK in the global ROM.

The General Universal Composability (GUC) Model. Our security experiment is that of the GUC model of Canetti et al. [20], which enables the UC-security analysis of protocols with global functionalities.

Briefly, the UC and GUC modeling of the world envisions an adversarial environment $\mathcal{Z}$, which provides inputs to honest participants, observes their outputs, and (on a high level) directs the order in which messages are passed between different system components. Additionally, the world includes honest participants (that receive inputs from $\mathcal{Z}$ and let $\mathcal{Z}$ observe their outputs) and adversarial participants controlled by the adversary $\mathscr{A}$ (whose behavior is also directed and observed by $\mathcal{Z}$).

The ideal world additionally contains an ideal functionality $\mathcal{F}$ and an ideal adversary $\mathcal{S}$, also called the simulator. In the ideal world, the honest participants pass their inputs directly to $\mathcal{F}$ and receive output from it. The real world does not contain such a functionality; instead, the honest participants run a cryptographic protocol. The corrupted participants in the ideal world always communicate through $\mathcal{S}$, who simulates their view and may pass their inputs to $\mathcal{F}$ through a private channel. There are also worlds in between these two: in a $\mathcal{G}$-hybrid world, the honest participants run a protocol that can make calls to an ideal functionality $\mathcal{G}$. In the GUC model, $\mathcal{G}$ is accessible not only to the honest participants, but also to $\mathcal{Z}$. A cryptographic protocol is said to be (G)UC-secure with respect to a functionality $\mathcal{F}$ (or (G)UC-emulate $\mathcal{F}$) if for any real-world adversary $\mathscr{A}$, there exists an "ideal" adversary $\mathcal{S}$ (also called the simulator), which creates a view for the environment (in the ideal world) that is indistinguishable from its view of the cryptographic protocol.

In our case, the ideal functionality is the NIZKPoK ideal functionality, or $\mathcal{F}_{\texttt{NIZK}}$, which works as follows. An honest participant can compute a proof π of knowledge of w such that $(x, w) \in R$ by querying $\mathcal{F}_{\texttt{NIZK}}$'s `Prove` interface and giving it (x, w). The string π itself is computed according to the algorithm `SimProve` provided by the ideal adversary $\mathcal{S}$. The functionality guarantees the zero-knowledge property because `SimProve` is independent of w. An honest participant can also verify a supposed proof π for x by querying $\mathcal{F}_{\texttt{NIZK}}$'s `Verify` interface on input (x, π). $\mathcal{F}_{\texttt{NIZK}}$ ensures the soundness of the proof system as follows: if the proof π was *not* issued by $\mathcal{F}_{\texttt{NIZK}}$, then it runs an extractor algorithm `Extract` provided by $\mathcal{S}$ to try to compute a witness w from the proof π. `Extract` may require additional inputs that may need to be supplied by $\mathcal{S}$ as well.

In Canetti's original UC framework [19], all communications between participants are passed through a special controller that determines whether the communication is valid in the model of analysis. For example in the standard UC model, $\mathcal{Z}$ can only control participants and direct inputs corresponding to a particular session identifier (SID) s—any message between $\mathcal{Z}$ and a party with $\texttt{sid} \neq s$ is rejected. In the GUC model, $\mathcal{Z}$ is allowed to engage with participants under any SID and in particular query a global functionality, for instance $\mathcal{G}_{\texttt{rpoRO}}$ or $\mathcal{G}_{\texttt{roRO}}$, under any SID. The global functionality answers the queries of all parties in all sessions in both real and ideal experiments. For a full specification of the GUC model, we refer readers to Canetti et al. [20].

The Global Random-Oracle Models (Global ROMs). The traditional random oracle (RO) $H : \{0,1\}^* \to \{0,1\}^\ell$ is a function that takes any string as input and returns a uniformly random ℓ-bit string as output [3]. The global

random-oracle model (global ROM) allows us to capture the realistic scenario in which the same RO is reused by many parties over many (potentially concurrent) executions of numerous distinct protocols. As envisioned by Canetti et al. [22] and formalized by Camenisch et al. [10], the "strict" global RO functionality $\mathcal{G}_{\mathtt{sRO}}$ is a public, universally-accessible RO that can be queried by any party in any protocol execution, including by the arbitrary concurrent protocols modeled by the environment in the UC framework [20].

Pass [40], Canetti and Fischlin [21], Canetti et al. [20,22], and Camenisch et al. [10] have all discussed the limitations of $\mathcal{G}_{\mathtt{sRO}}$. In particular, Canetti and Fischlin [21] demonstrated that it is impossible to achieve UC commitments with *only* a global setup, and Canetti et al. extended this argument to commitments and zero knowledge in the GUC framework [20] and the $\mathcal{G}_{\mathtt{roRO}}$-hybrid model [22]. The limitation stems from the fact that in a "strict" setup, the simulator does not have any special advantage over a regular protocol participant. In our setting, $\mathcal{F}_{\mathtt{NIZK}}$ needs access to the adversary's RO queries in order to extract witnesses and ensure the special soundness property. Most zero-knowledge simulators also rely on the extra ability to *program* the RO at selected points in order to simulate proofs of statements without witnesses. In the rest of this section, we will discuss and draw a distinction between these two privileges in the global ROM.

In order to resolve the former issue with extraction and realize GUC-secure commitments in the (non-programmable) global ROM, Canetti et al. introduced a global RO $\mathcal{G}_{\mathtt{roRO}}$ with a restricted "observability" property [22]. The simulator (ideal adversary) $\mathcal{S}$ in the security proof of a protocol Π emulating an ideal functionality $\mathcal{F}$ in the $\mathcal{G}_{\mathtt{roRO}}$-hybrid model is able to observe all adversarial queries to $\mathcal{G}_{\mathtt{roRO}}$ as follows. First, $\mathcal{S}$ can observe the corrupted parties' queries to $\mathcal{G}_{\mathtt{roRO}}$ by directly monitoring their input and output wires (recall that in the ideal world, corrupted parties communicate through $\mathcal{S}$). The *environment's* queries to $\mathcal{G}_{\mathtt{roRO}}$, on the other hand, are not directly monitored by $\mathcal{S}$. Since $\mathcal{G}_{\mathtt{roRO}}$ is completely public, the environment is free to query it anytime; however, the environment is not free to query it with the same SID as the participants in Π or $\mathcal{F}$, because it is external to Π by definition.

In order to ensure the environment's queries are still available to the simulator, $\mathcal{G}_{\mathtt{roRO}}$ checks whether the SID for a query matches the SID of the querying party. In the event that it does not, this query is labelled "illegitimate," creating the restriction. $\mathcal{G}_{\mathtt{roRO}}$ makes a record of all illegitimate queries available to an ideal functionality $\mathcal{F}$ with the correct SID, if it exists. We will see that for our construction of GUC-secure NIZKPoK in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{NIZK}}$-hybrid model, $\mathcal{F}_{\mathtt{NIZK}}$ can leverage these queries to extract witnesses from the environment's proofs.

A key feature of this relaxation of the strict global setup is that it does not hide any of its interfaces from the environment. $\mathcal{G}_{\mathtt{roRO}}$ might be checking querents' SIDs and disclosing information to $\mathcal{F}_{\mathtt{NIZK}}$, but its "front-facing" interface looks no different to the environment than it does to any other party. While in the original formulation of the definition $\mathcal{G}_{\mathtt{roRO}}$ makes the list of illegitimate queries available to $\mathcal{F}$, it is reasonable to imagine a world in which all of the illegitimate queries are simply posted to a global public bulletin—honest parties will never attempt to interfere with other parties' sessions, so their queries will

never be disclosed to anyone. Put differently, since the list of illegitimate queries contains adversarial queries only, the environment (who is also puppet-mastering the corrupted parties) cannot learn anything from seeing the list of illegitimate queries that it did not already know—any information it would glean from the global bulletin would be *self-simulatable*. Therefore, the observability property of $\mathcal{G}_{\texttt{roRO}}$ does not functionally change the view of the environment.

We contrast this with the "programmability" property of the restricted programmable observable global RO, $\mathcal{G}_{\texttt{rpoRO}}$ [10]. Technically since $\mathcal{G}_{\texttt{rpoRO}}$ is public, anybody can program it. While there are uses for a non-restricted programmable global RO [10], it would not work for NIZKPoK since anybody could forge a proof. In order to ensure that programming is restricted to the simulator only, $\mathcal{G}_{\texttt{rpoRO}}$ has an `IsProgrammed` interface that allows participants with a particular SID to check whether the output of $\mathcal{G}_{\texttt{rpoRO}}$ was programmed on some input pertaining to the *same session*. Honest parties in the challenge session can therefore check whether the adversary has programmed $\mathcal{G}_{\texttt{rpoRO}}$, and can refuse to continue the protocol if so. Camenisch et al. argue that since the simulator controls the corrupted parties' view of the experiment in the ideal world, it can pretend that the simulator did not program anything and return "false" to all of the corrupted parties' `IsProgrammed` queries. Since only parties running a legitimate protocol session s are allowed to use the `IsProgrammed` interface for s, the environment cannot make `IsProgrammed` queries for s—if it could, it would easily be able to distinguish between the real and ideal experiments by checking whether honest parties' responses were programmed. Unlike the former observability property, the programmability property afforded by the `IsProgrammed` interface creates a *local* restriction—it does not allow the environment to interact freely with the interfaces of the RO just like any other party would.

We show how to construct efficient, GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model. However, we believe there are downsides to $\mathcal{G}_{\texttt{rpoRO}}$: it is not clear how compromising the fully-public aspect of the global ROM with a locally-restricted interface might impact the overall composability of protocols proven secure in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model. In order to achieve efficient GUC-secure NIZKPoK *without* this localized interface, we build a new hybrid model called the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model. The $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model shifts the localized interface from inside of the global RO to *inside of the protocol*—as long participants realize the functionality $\mathcal{F}_{\texttt{CRS}}$ with a secure common reference string (CRS), our GUC-secure NIZKPoK are guaranteed to retain composability with primitives that are provably secure in (fully) global ROMs. For a one-time cost at the beginning of the protocol execution, participants can compute this CRS securely and realize $\mathcal{F}_{\texttt{NIZK}}$ *using only the observable global RO* $\mathcal{G}_{\texttt{roRO}}$ by leveraging Canetti et al.'s GUC-secure NISC protocol [22]. Similar mechanisms are used in practice to obtain practical NIZKPoK in other ROMs [7].

In the real world, our ideal CRS functionality $\mathcal{F}_{\texttt{CRS}}$ returns a random string `CRS` (the CRS our real-world participants might compute using the NISC protocol). In the ideal world, the simulator generates `CRS` itself, along with a trapdoor `trap` that only it knows. The proof-generation process in our construction of GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model is to show that the prover

either knows a "real" witness w for a statement x such that $(x, w) \in R$, *or* it knows the trapdoor to the CRS. The `Prove` and `SimProve` algorithms differ only in the witness used: a real prover must use a real witness, while the simulator can use the CRS trapdoor in a way that we will show is imperceptible to the environment. We formalize this intuition using an OR-protocol [24,26] over the original relation R and what we call a samplable-hard relation for the CRS.

Straight-Line Extraction and the Fischlin Transform. The original Fischlin transform [30] is a non-interactive transform for Σ-protocols in the standard ROM that allows for *straight-line* (or *online*) extraction. Straight-line extraction is a process by which the extractor can produce a witness straight from a valid proof without any further interaction with the prover. (In order to do so, it will need additional, auxiliary information available to the extractor algorithm only.) This is in contrast to extraction in the "rewinding" model, in which the extractor resets the prover to a previous state and hopes for a certain pattern of interaction before it can obtain a witness. Straight-line extraction is necessary in the (G)UC model, which does not allow the simulator to rewind the environment [20]. Furthermore, straight-line extraction produces a tight reduction, which avoids security nuances surrounding the forking lemma [33].

In order to create a straight-line extractable proof system from a Σ-protocol, the Fischlin transform essentially forces the prover to rewind itself, requiring multiple proofs on repeated commitments until the probability that the prover has generated at least two responses to different challenges on the same commitment is overwhelming. Kondi and shelat recently showed that because the Fischlin prover is deterministic—that is, because it tests challenges by iterating from zero to some fixed constant—the original transform is open to a "replay" attack that breaks the the witness indistinguishability property of OR-protocols [36]. To avoid the attack, Fischlin's original construction requires the underlying Σ-protocols to have a property called *quasi-unique responses*, which Kondi and shelat demonstrate precludes the transformation of OR-protocols. Kondi and shelat show this property can be omitted (and most OR-protocols transformed) as long as one randomizes the challenge selection process.

We review the resulting "randomized" Fischlin transform [31,36] that we will leverage in our constructions. First, the prover generates a vector of r commitments, where r is a parameter of the system. For each commitment, the prover draws challenges uniformly at random from the t-bit challenge space, computes responses, and queries the RO on the complete transcript until it finds one that causes the RO to return a value with b leading zeroes, where t and b are also parameters. If the prover does not find such a response, it chooses the transcript such that, on input this transcript, the RO returns the smallest value in lexicographic order.

In the end, the prover sends *only* the responses with minimal return values for each of the r repetitions to the verifier. The verifier is therefore only able to see a single transcript for each commitment, and can check the validity of the transcripts and oracle queries as usual. Since the transform allows the prover some flexibility in choosing a minimal oracle response value (rather than forcing

all b bits to be leading zeroes), the verifier checks that the sum of the oracle's responses to the transcripts is less than some maximal parameter, S.

The parameters b, r, S, and t are set such that there are guaranteed to be (with overwhelming probability) two matching transcript queries, $(x, \texttt{com}, \texttt{chl}, \texttt{res})$ and $(x, \texttt{com}, \texttt{chl}', \texttt{res}')$, with the same commitment but different challenges. When the extractor obtains these oracle queries via either the simulator in the security experiment or the observability interface of the global RO, it is able to extract a witness w such that $(x, w) \in R$ with overwhelming probability, as guaranteed by the special soundness property.

2 Preliminaries

We use standard notation, available for review in Appendix A.1.

2.1 Σ-protocols, Revisited

Recall from Section 1 that Σ-protocols are three-round, public-coin transactions in which a prover "commits" to a value com, the verifier sends a unique ℓ-length *random* "challenge" chl to the prover, and the prover "responds" with a value res that must convince the verifier it knows a value corresponding to some public statement x. If the verifier accepts the values $(\texttt{com}, \texttt{chl}, \texttt{res})$ as a "proof" of x, it outputs 1 to accept the proof. Otherwise, it outputs 0 to reject.

Formally, let R be any efficiently computable binary relation. For pairs $(x, w) \in R$, or equivalently such that $R(x, w) = 1$, we call x a statement in the language of R, denoted L_R, and say w is a witness to $x \in L_R$. We consider Σ-protocols over a relation R between a prover P and a verifier V that have the general form discussed above, which Damgård formalizes as a Σ-protocol template [26].

Since we will later introduce compilers for Σ-protocols—first to make them non-interactive and straight-line extractable and then to make them GUC-secure—it will be helpful to define Σ-protocol interfaces with precise inputs and outputs. We therefore present an algorithmic version of Damgård's definitions, and include the originals in Appendix A.3.

Definition 1 (Σ-protocol Template). *The* 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 the challenge length ℓ.*
- $\texttt{com} \leftarrow \texttt{Commit}(\texttt{ppm}, x, w)$*: P sends V a message* com*.*
- $\texttt{chl} \leftarrow \texttt{Challenge}(\texttt{ppm}, x, \texttt{com})$*: V sends P a random ℓ-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.*

Recall that Σ-protocols must also satisfy the properties of completeness, special honest-verifier zero-knowledge (SHVZK), and special soundness (SS). The SHVZK property requires the existence of a simulator algorithm for simulating proofs, and the SS property requires an extractor algorithm for extracting witnesses. Therefore, our algorithmic specification of a Σ-protocol includes three additional algorithms: `SimSetup`, `SimProve`, and `Extract`.

In order to more easily translate our definition of Σ-protocols into the non-interactive setting, we combine the `Commit`, `Challenge`, and `Respond` algorithms of the protocol template into a `Prove` interface. For now we are still dealing with the interactive version, and the specification of `Prove` below is a two-party protocol where the first input to the algorithm is the prover's input, and the second input is the verifier's. After running `Prove`, both parties obtain the same copy of the proof transcript $\pi = (\texttt{com}, \texttt{chl}, \texttt{res})$. In the next section, we will introduce a non-interactive straight-line compiler that makes the `Prove` interface a non-interactive algorithm in the random-oracle model.

The non-interactive, straight-line extractable (NISLE) versions of Σ-protocols have different formulations of the SHVZK and SS properties. Because we will work almost exclusively with the NISLE versions of SHVZK and SS, we defer a formal definition and discussion of the original versions to Appendix A.4.

Definition 2 (Σ-protocol). *A* Σ-protocol for a relation R *based on a protocol template* τ *(Definition 1) is a tuple of efficient procedures* $\Sigma_{R,\tau} = (\texttt{Setup}, \texttt{Prove}, \texttt{Verify}, \texttt{SimSetup}, \texttt{SimProve}, \texttt{Extract})$*, defined as follows.*

- $\texttt{ppm} \leftarrow \texttt{Setup}(1^\lambda)$*: Given a security parameter* 1^λ*, invoke* $\tau.\texttt{Setup}(1^\lambda)$ *to obtain the public parameters* `ppm`*.*
- $\pi \leftarrow \texttt{Prove}((\texttt{ppm}, x, w), (\texttt{ppm}, x))$*: Let the first (resp. second) argument to* `Prove` *be the input of the prover (resp. verifier), where both parties get* `ppm` *and the statement* x*, but only the prover gets* w*. 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* π *for statemenet* x*, parse* π *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 18 in Appendix A.4.*
- $(\texttt{ppm}, z) \leftarrow \texttt{SimSetup}(1^\lambda)$*: Generate* `ppm` *and the simulation trapdoor* z*. Together,* `SimSetup` *and* `SimProve` *must satisfy the* special honest-verifier zero-knowledge property *from Definition 19 in Appendix A.4.*
- $\pi \leftarrow \texttt{SimProve}(\texttt{ppm}, z, x, \texttt{chl})$: *Given public parameters* `ppm`*, trapdoor* z*, statement* x*, and a challenge* `chl`*, produce a proof* $\pi = (\texttt{com}, \texttt{chl}, \texttt{res})$*.*
- $w \leftarrow \texttt{Extract}(\texttt{ppm}, x, \pi, \pi')$: *Given two proofs* π *and* π' *for a statement* x*, output a witness* w*.* `Extract` *must satisfy the* special soundness property *from Definition 20 in Appendix A.4.*

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

2.2 Straight-Line Compilers

The notion of a straight-line compiler (SLC) for Σ-protocols in the random-oracle model (ROM) was introduced by Fischlin [31,30]. Fischlin's work makes use of the first non-interactive (NI) transform for Σ-protocols given by Fiat and Shamir [29], which was proven secure in the standard ROM by Pointcheval and Stern [42]. In Fiat and Shamir's original NI transform, they make the `Prove` protocol an algorithm by replacing the challenge issued by the verifier with the output of a random oracle (RO) H on input $(x, \texttt{com})$. The simulation process is similarly non-interactive, whereby the simulator obtains the challenge in advance by programming the RO (as opposed to rewinding the prover).

An SLC is a special kind of non-interactive transform for Σ-protocols in the ROM where the extraction process is also non-interactive, such that after `Prove` outputs π for a statement x, `Extract` can output a valid witness for x without any further interaction with the prover. Fischlin defines a new special soundness (SS) game for the ROM in which $\mathscr{A}$ has oracle access to H, and `Extract` works on input $(x, \pi, \mathcal{Q}_{\mathscr{A}})$, where $\mathcal{Q}_{\mathscr{A}}$ are $\mathscr{A}$'s queries to H. Fischlin's approach is not the only one for achieving straight-line extraction. Verifiable encryption [17,9] provides a different mechanism: the parameters `ppm` contain a public key, and the proof π contains an encryption of the witness under this key. The extractor's trapdoor is the decryption key. This approach requires additional machinery: it needs a proof systems for proving that a plaintext of a particular ciphertext is a witness w, and thus cannot just be constructed directly from Σ_R. While our constructions follow Fischlin and interpret $\mathcal{Q}_{\mathscr{A}}$ to be $\mathscr{A}$'s queries to H, the definition below covers either approach: one can simply consider $\mathcal{Q}_{\mathscr{A}}$ to be the minimal privileged information needed for the extractor to function.

A non-interactive, straight-line extractable (NISLE) zero-knowledge proof system must also satisfy a non-interactive equivalent of SHVZK. In particular, its definition must reflect the fact that the simulator might be programming the RO. Fischlin's conception of the SHVZK property for SLCs allows the challenger in the SHVZK game to update the RO after every `Prove` query, and gives the adversary $\mathscr{A}$ oracle access to this new RO. The SHVZK property must continue to hold even as the RO is updated, meaning that if the simulator changes the RO at all, it must be done in a way that is imperceptible to $\mathscr{A}$. Note that the definition does not imply that the simulator *has* to program the RO—just that if it does, it must do so imperceptibly. This nuance is important because we will later give a construction in Section 5.3 for GUC-secure NIZKPoK in the (non-programmable) $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model—this construction should not (and does not) contradict our result from Section 3.5, which says that any GUC-secure NIZKPoK must meet the requirements of non-interactive (multiple) SHVZK.

Similarly, Fischlin presents a slightly weaker version of completeness that allows for the optimization of other parameters; other version of SLCs, for instance Kondi and shelat's randomized version of Fischlin's transform [36], do not require this property. In Fischlin's weaker version (which we call overwhelming completeness), any proof of a valid statement-witness pair computed according to the `Prove` algorithm is accepted by `Verify` with probability that is overwhelming in the security parameter (rather than strictly 1). Certainly any SLC that

satisfies the regular notion of completeness will also satisfy the weaker version of overwhelming completeness, so we use the weaker version in order to identify the *minimal* properties necessary for a protocol be a GUC-secure NIZKPoK.

To that end, we will show in Section 3.5 that the security properties guaranteed by an SLC are strictly necessary to transform a Σ-protocol into a GUC-secure NIZKPoK in any global ROM. Later in Section 4, we will show that an SLC is sufficient to transform any Σ-protocol into a GUC-secure NIZKPoK in the (programmable) $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model, and in Section 5 we will show that it is sufficient in conjunction with our OR-protocol construction to complete the transformation in the $\mathcal{G}_{\texttt{rpoRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model.

Definition 3 (Straight-Line Compiler). *An algorithm* SLC *is a* straight-line compiler (SLC) *in the random-oracle model if given any Σ-protocol Σ_R for relation R (Definition 2) as input, it outputs a tuple of* algorithms $\Sigma_R^{\texttt{SLC}} = (\texttt{Setup}^H, \texttt{Prove}^H, \texttt{Verify}^H, \texttt{SimSetup}, \texttt{SimProve}, \texttt{Extract})$ *based on Σ_R that satisfy the following properties: overwhelming completeness (Definition 4), non-interactive multiple special honest-verifier zero-knowledge (Definition 5), and non-interactive special simulation soundness (Definition 6).*

We refer to $\Sigma_R^{\texttt{SLC}}$ as a non-interactive, straight-line extractable (NISLE) Σ-protocol, and proofs generated by $\Sigma_R^{\texttt{SLC}}$ as non-interactive, straight-line extractable zero-knowledge proofs of knowledge (NISLE ZKPoK) in the random-oracle model.

Definition 4 (Overwhelming Completeness). *For any security parameter λ, any $(x, w) \in R$, and any π produced by running $\Sigma_R^{\texttt{SLC}}.\texttt{Prove}^H(x, w)$,*

$$\Pr[\Sigma_R^{\texttt{SLC}}.\texttt{Verify}^H(x, \pi) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

In the standard definition of SHVZK, $\mathscr{A}$ is only permitted to issue *one* Prove query. In the GUC security experiment (and in most natural applications of Σ-protocols), the environment is allowed to issue polynomially-many Prove queries, and we will still need the SHVZK property to hold. Therefore, we present our version of Fischlin's definition of non-interactive *multiple* SHVZK (NIM-SHVZK).

Definition 5 (Non-Interactive Multiple SHVZK). *A NISLE Σ-protocol $\Sigma_R^{\texttt{SLC}}$ based on Σ_R is* non-interactive multiple special honest-verifier zero-knowledge (NIM-SHVZK) *in the random-oracle model if there exist algorithms $\Sigma_R^{\texttt{SLC}}$.* SimSetup *and $\Sigma_R^{\texttt{SLC}}$.*SimProve *such that for any security parameter λ, any* PPT *adversary $\mathscr{A}$, and a bit $b \leftarrow_{\$} \{0, 1\}$, there exists some negligible function* negl *such that $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where b' is the result of running the game $\mathsf{NIM\text{–}SHVZK}_{\mathscr{A}, \Sigma_R^{\texttt{SLC}}}(1^\lambda, b)$ from Figure 1. We say $\mathscr{A}$ wins the* NIM–SHVZK *game if $\Pr[b' = b] > \frac{1}{2} + \mathsf{negl}(\lambda)$.*

Similarly, the environment in the ideal-world GUC experiment will have access to polynomially-many proofs generated by the SimProve algorithm, which $\mathcal{F}_{\texttt{NIZK}}$ will use to simulate proofs. We therefore define our straight-line compilers to have the NI special *simulation* soundness property (NI-SSS), which says that special soundness must still hold even after an adversary has seen polynomially-many proofs from the simulator. Fischlin's original construction is both NIM-SHVZK and NI-SSS [30]. We will use his results in Section 6.1 to prove that the randomized Fischlin transform [36,30] is also NIM-SHVZK and NI-SSS.

$\mathsf{NIM\text{–}SHVZK}^{H}_{\mathscr{A},\Sigma_R^{\mathsf{SLC}}}(1^\lambda, 0)$

1 : $\mathtt{ppm} \leftarrow \Sigma_R^{\mathsf{SLC}}.\mathtt{Setup}^H(1^\lambda)$
2 : $\mathtt{st} \leftarrow \mathscr{A}^H(1^\lambda, \mathtt{ppm})$
3 : $i \leftarrow 0$
4 : **while** $\mathtt{st} \notin \{0,1\}$:
5 : $\quad(\mathtt{Prove}, x, w), \mathtt{st} \leftarrow \mathscr{A}^H(\mathtt{st})$
6 : $\quad$**if** $R(x,w) = 1$:
7 : $\quad\quad\pi \leftarrow \Sigma_R^{\mathsf{SLC}}.\mathtt{Prove}^H((x,w),(x,\mathtt{chl}))$
8 : $\quad$**else** :
9 : $\quad\quad\pi \leftarrow \bot$
10 : $\quad\mathtt{st} \leftarrow \mathscr{A}^H(\mathtt{st}, \pi)$
11 : **return** b'

$\mathsf{NIM\text{–}SHVZK}^{H}_{\mathscr{A},\Sigma_R^{\mathsf{SLC}}}(1^\lambda, 1)$

1 : $\mathtt{ppm}, z \leftarrow \Sigma_R^{\mathsf{SLC}}.\mathtt{SimSetup}(1^\lambda)$
2 : $H_0 \leftarrow H$
3 : $\mathtt{st} \leftarrow \mathscr{A}^{H_0}(1^\lambda, \mathtt{ppm})$
4 : $i \leftarrow 0$
5 : **while** $\mathtt{st} \notin \{0,1\}$:
6 : $\quad(\mathtt{Prove}, x, w), \mathtt{st} \leftarrow \mathscr{A}^{H_i}(\mathtt{st})$
7 : $\quad$**if** $R(x,w) = 1$:
8 : $\quad\quad i \leftarrow i + 1$
9 : $\quad\quad\pi, H_i \leftarrow \Sigma_R^{\mathsf{SLC}}.\mathtt{SimProve}(x, z, \mathtt{chl})$
10 : $\quad$**else** :
11 : $\quad\quad\pi \leftarrow \bot$
12 : $\quad\mathtt{st} \leftarrow \mathscr{A}^{H_i}(\mathtt{st}, \pi)$
13 : **return** b'

Fig. 1. Non-Interactive Multiple SHVZK (NIM-SHVZK) Game.

Definition 6 (Non-Interactive Special Simulation Soundness). *A NISLE Σ-protocol Σ_R^{SLC} based Σ_R is* non-interactive special simulation sound *in the random-oracle model if there exists an algorithm $\Sigma_R^{\mathsf{SLC}}.\mathtt{Extract}$ such that for any security parameter λ and any $\mathtt{PPT}$ adversary $\mathscr{A}$,*

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

where H is any random oracle and $\mathsf{NI\text{–}SSS}$ is the NI-SS game described in Figure 2. We say $\mathscr{A}$ wins if $\Pr[\mathtt{Fail} \leftarrow \mathsf{NI\text{–}SSS}^{H}_{\mathscr{A},\Sigma_R^{\mathsf{SLC}}}(1^\lambda)] > \mathsf{negl}(\lambda)$.

Σ-protocols that maintain the zero-knowledge property under NI transforms in the ROM must additionally have $\mathtt{com}$ messages with entropy that is super-logarithmic in the security parameter [31], such that the adversary cannot pre-emptively query the prefix $(x, \mathtt{com})$ to the RO and check whether the challenge supplied by the prover matches what it receives. We define and discuss Fischlin's *superlogarithmic commitment entropy* property further in Appendix A.6.

2.3 OR-protocols

Rather than producing a proof corresponding to a single statement x in a language L_R, the prover in an OR-protocol proves that it knows a witness for *either* a statement x_0 in L_{R_0} *or* another statement x_1 in L_{R_1}. At a high level, the prover does this by simulating the proof of the statement for which it does not have a witness, while computing the proof of the statement for which it *does* have a witness honestly.

$\mathrm{SSS}^{H}_{\mathscr{A},\Sigma_R^{\mathtt{SLC}}}(1^\lambda)$

1 : $\mathtt{ppm}, z \leftarrow \Sigma_R^{\mathtt{SLC}}.\mathtt{SimSetup}(1^\lambda)$
2 : $H_0 \leftarrow H;\ i \leftarrow 0;\ \mathtt{Response} \leftarrow \perp$
3 : $\mathtt{st} \leftarrow \mathscr{A}^{H_0}(1^\lambda, \mathtt{ppm})$
4 : **while** $\mathtt{st} \neq \mathtt{halt}$:
5 : $\quad \mathtt{Query}, \mathcal{Q}_i^{\mathscr{A}}, \mathtt{st} \leftarrow \mathscr{A}^{H_i}(\mathtt{st})$
6 : $\quad$ **if** $\mathtt{Query} = (\mathtt{Prove}, x, w)$:
7 : $\quad\quad$ **if** $R(x, w) = 1$:
8 : $\quad\quad\quad \pi, H_{++i} \leftarrow \Sigma_R^{\mathtt{SLC}}.\mathtt{SimProve}(x, z, \mathtt{chl})$
9 : $\quad\quad\quad \mathtt{Response} \leftarrow \pi$
10 : $\quad$ **elseif** $\mathtt{Query} = (\mathtt{Challenge}, x, \pi)$
11 : $\quad\quad$ **if** $\Sigma_R^{\mathtt{SLC}}.\mathtt{Verify}^{H_i}(x, \pi) = 1$:
12 : $\quad\quad\quad w \leftarrow \Sigma_R^{\mathtt{SLC}}.\mathtt{Extract}(x, \pi, \mathcal{Q}_i^{\mathscr{A}})$
13 : $\quad\quad\quad$ **if** $R(x, w) = 0$: **return** $\mathtt{Fail}$
14 : $\quad \mathtt{st} \leftarrow \mathscr{A}^{H_i}(\mathtt{st}, \mathtt{Response})$
15 : **return** $\mathtt{Success}$

Fig. 2. Non-Interactive Special Simulation Soundness (NI-SSS) Game.

Our definition is adapted directly from Damgård's [26], with a few minor tweaks to make it more general. Since we will use the OR-protocol functionality as a black box in our construction, it suffices for the purpose of understanding our results to treat the OR-protocol as a Σ-protocol (according to Definition 2) with *compound inputs*. For example, we represent the compound statement $x_0 \vee x_1$ with the upper-case variable $X = (x_0, x_1)$. The witness $W = (w, b)$ includes a witness along with a bit b such that $(x_b, w) \in R_b$. We provide the detailed version of our definition alongside Damgård's, as well as a discussion of the minor differences between them, in Appendix A.7.

3 Properties of GUC-secure NIZKPoK

In this section we formalize the definitions of the programmable global RO $\mathcal{G}_{\mathtt{rpoRO}}$ and the observable global RO $\mathcal{G}_{\mathtt{roRO}}$, the ideal NIZKPoK functionality $\mathcal{F}_{\mathtt{NIZK}}$, the CRS ideal functionality $\mathcal{F}_{\mathtt{CRS}}$, and the security requirements for protocols that GUC-realize $\mathcal{F}_{\mathtt{NIZK}}$ in the $\mathcal{G}_{\mathtt{rpoRO}}$- and $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid models. We then show that the non-interactive multi-SHVZK and non-interactive special simulation soundness properties are *strictly necessary* to obtain GUC-secure NIZKPoK in any global ROM. In Sections 4 and 5, we will show how to use any straight-line compiler to realize GUC compilers for Σ-protocols in the $\mathcal{G}_{\mathtt{rpoRO}}$- and $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid models, respectively.

3.1 $\mathcal{G}_{\texttt{roRO}}$ and $\mathcal{G}_{\texttt{rpoRO}}$, Revisited

Building on the overview of the global ROM given in Section 1, we now formalize Canetti et al.'s restricted observable global RO $\mathcal{G}_{\texttt{roRO}}$ [22] and Camenisch et al.'s restricted programmable observable global RO $\mathcal{G}_{\texttt{rpoRO}}$. As with traditional ROs, both oracles act as functions that respond to each input string $x_i \in \{0,1\}^*$ with a uniformly random ℓ-bit string $v_i \in \{0,1\}^\ell$. We call this original query algorithm `Query`, and will now define the other types of queries handled by each oracle. Since $\mathcal{G}_{\texttt{rpoRO}}$ builds on the interfaces of $\mathcal{G}_{\texttt{roRO}}$, we will start with the specification of $\mathcal{G}_{\texttt{roRO}}$ and follow with the extra interfaces of $\mathcal{G}_{\texttt{rpoRO}}$.

Recall from Section 1 that since we are working in the GUC model, both oracles will expect to see a calling party's unique session identifier (SID). The first thing $\mathcal{G}_{\texttt{roRO}}$ does when it receives a query is to check whether the calling party's SID `sid` matches the session s for which it has requested randomness. If $\texttt{sid} \neq s$, $\mathcal{G}_{\texttt{roRO}}$ assumes this is an "illegitimate" query made by the environment, and records the query in its special list of illegitimate queries for s, denoted $\mathcal{Q}_s$. In the original version of the definition (provided in Appendix A.10), only the ideal functionality $\mathcal{F}^s$ for session s can query $\mathcal{G}_{\texttt{roRO}}$ using the `Observe` interface to get the list of illegitimate queries for s. However, note that no honest provers' queries will ever be recorded in this list, as they will only ever be querying $\mathcal{G}_{\texttt{roRO}}$ for randomness sessions in which they are participating legitimately. Therefore, we follow Camenisch et al.'s version of the restricted observability property [10] and simply release the list $\mathcal{Q}_s$ to anyone who wants it.

Definition 7 (Observable Global RO $\mathcal{G}_{\texttt{roRO}}$). [22] *The* observable global RO $\mathcal{G}_{\texttt{roRO}}$ *is a tuple of algorithms* $(\texttt{Query}, \texttt{Observe})$ *defined over an output length* ℓ *and an initially empty list of queries* $\mathcal{Q}$*:*

- $v \leftarrow \texttt{Query}(x)$: *Parse* x *as* (s, x') *where* s *is an SID. If the caller's SID* $\neq s$*, add* (x, v) *to the list* $\mathcal{Q}_s$ *of illegitimate queries for* 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*.*
- $\mathcal{Q}_s \leftarrow \texttt{Observe}(s)$: *If* $\mathcal{Q}_s$ *does not yet exist, set* $\mathcal{Q}_s = \emptyset$*. Return* $\mathcal{Q}_s$*.*

In addition to the `Query` and `Observe` interfaces, Camenisch et al.'s restricted *programmable* observable global RO $\mathcal{G}_{\texttt{rpoRO}}$ has two extra interfaces, `Program` and `IsProgrammed`. $\mathcal{G}_{\texttt{rpoRO}}$ keeps track of which queries have been programmed using the set `Program`. `Program` takes a potential programming (x, v) as input and checks whether the input x is already programmed; if it is, `Program` outputs 0 to indicate a failed programming. Otherwise, it adds (x, v) to $\mathcal{Q}$ and the list of programmed queries `prog` and and returns 1. Note that there are no checks on who can program $\mathcal{G}_{\texttt{rpoRO}}$, since privileged (simulator-only) programming is not allowed in the GUC model. In order to functionally restrict this privilege to the simulator, Camenisch et al. introduces the `IsProgrammed` interface, which reveals whether or not $\mathcal{G}_{\texttt{rpoRO}}$ was programmed on an index $x = (s, x')$, but only to a calling party with $\texttt{sid} = s$. Notably, this interface directly restricts the environment from ever seeing whether or not the oracle was programmed (since the environment is by definition not part of any legitimate protocol session),

and indirectly restricts the adversary from ever seeing whether or not the oracle was programmed (since the simulator is in charge of its view in the ideal-world experiment in which programming is employed).

Definition 8 (Restricted Programmable Observable Global RO $\mathcal{G}_{\texttt{rpoRO}}$). [10] *The* restricted programmable observable global random oracle $\mathcal{G}_{\texttt{rpoRO}}$ *is a tuple of algorithms* $(\texttt{Query}, \texttt{Observe}, \texttt{Program}, \texttt{IsProgrammed})$ *defined over an output length* ℓ *and initially empty lists* $\mathcal{Q}$ *(queries) and* `prog` *(programmed queries):*

- $v \leftarrow \texttt{Query}(x)$: *Parse* x *as* (s, x') *where* s *is an SID. If the caller's SID* $\neq s$*, add* (x, v) *to the list* $\mathcal{Q}_s$ *of illegitimate queries for* 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*.*
- $\mathcal{Q}_s \leftarrow \texttt{Observe}(s)$: *If* $\mathcal{Q}_s$ *does not yet exist, set* $\mathcal{Q}_s = \emptyset$*. Return* $\mathcal{Q}_s$*.*
- $\{0,1\} \leftarrow \texttt{Program}(x, v)$: *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*.*

3.2 The NIZKPoK Ideal Functionality

We now formalize the NIZKPoK ideal functionality $\mathcal{F}_{\texttt{NIZK}}$. Recall from Section 1 that in the "ideal" world, the honest parties who would execute protocol Π are actually dummy parties who do not perform any computations of their own. Instead, they pass all of their inputs to an ideal functionality $\mathcal{F}_{\texttt{NIZK}}$, who instructs them on how to respond. As is standard in the UC framework [19], there is one ideal functionality for each SID s. A dummy party with SID s can only send input and receive output from the $\mathcal{F}_{\texttt{NIZK}}$ with the same SID, denoted $\mathcal{F}^s_{\texttt{NIZK}}$.

Each $\mathcal{F}^s_{\texttt{NIZK}}$ will need to run some kind of setup, then process proofs and verifications on behalf of the honest parties in its session. Recall that in order to be NIZKPoK, the proofs must be *non-interactive*, *zero-knowledge* (satisfying the SHVZK property), and *proofs of knowledge* (satisfying the SS property). These properties imply the existence of SHVZK simulator algorithms `SimSetup` and `SimProve` that do not take the prover's witness as input, as well as of the SS algorithm `Extract` that can compute witnesses from adversarially-created proofs. During $\mathcal{F}_{\texttt{NIZK}}$'s **Setup** procedure, $\mathcal{F}_{\texttt{NIZK}}$ requests the specifications of these algorithms from the ideal adversary (simulator) $\mathcal{S}$.

Note that there are two conditions in which $\mathcal{F}_{\texttt{NIZK}}$ can output `Fail`. The first is a completeness error, where $\mathcal{F}_{\texttt{NIZK}}$'s execution of the `SimProve` algorithm on input $(x, w) \in R$ fails to produce a proof π such that $\texttt{Verify}(x, \pi) = 1$. The second is an extraction error, where $\mathcal{F}_{\texttt{NIZK}}$'s execution of the `Extract` algorithm on input a valid, non-simulated proof tuple (x, π) fails to produce a witness w such that $R(x, w) = 1$. In the proof of Theorem 1 in Section 3.5, we will draw a direct correspondence between these failures and the functionality of a Σ-protocol in order to prove that any protocol that GUC-realizes $\mathcal{F}_{\texttt{NIZK}}$ must be non-interactive multi-SHVZK and special simulation-sound.

Definition 9 (NIZKPoK Ideal Functionality). *The ideal functionality* $\mathcal{F}_{\texttt{NIZK}}^{s}$ *of a* non-interactive zero-knowledge proof of knowledge (NIZK PoK) *for a particular SID* s *is defined as follows.*

Setup: *Upon receiving the request* $(\texttt{Setup}, s)$ *from a party* $P = (\texttt{pid}, \texttt{sid})$*, first check whether* $\texttt{sid} = s$*. If it doesn't, output* $\perp$*. Otherwise, if this is the first time that* $(\texttt{Setup}, s)$ *was received, pass* $(\texttt{Setup}, s)$ *to the ideal adversary* $\mathcal{S}$*, who returns the tuple* $(\texttt{Algorithms}, \texttt{Setup}, \texttt{Prove}, \texttt{Verify}, \texttt{Simulate}, \texttt{Extract})$ *with definitions for the algorithms* $\mathcal{F}_{\texttt{NIZK}}$ *will use.* $\mathcal{F}_{\texttt{NIZK}}$ *stores the tuple.*

Prove: *Upon receiving a request* $(\texttt{Prove}, s, x, w)$ *from a party* $P = (\texttt{pid}, \texttt{sid})$*, check that* $\texttt{sid} = s$ *and* $R(x, w) = 1$*. If not, output* $\perp$*. Otherwise, compute* π *according to the* `Simulate` *algorithm and check that* $\texttt{Verify}(x, \pi) = 1$*. If it doesn't, output* `Fail`*. Otherwise, record then output the message* $(\texttt{Proof}, s, x, \pi)$*.*

Verify: *Upon receiving a request* $(\texttt{Verify}, s, x, \pi)$ *from a party* $P = (\texttt{pid}, \texttt{sid})$*, first check that* $\texttt{sid} = s$*. If it doesn't, output* $\perp$*. Otherwise if* $\texttt{Verify}(x, \pi) = 0$*, output* $(\texttt{Verification}, s, x, \pi, 0)$*. Otherwise if* $(\texttt{Proof}, s, x, \pi)$ *is already stored, output* $(\texttt{Verification}, s, x, \pi, 1)$*. Otherwise, compute* w *according to the* `Extract` *algorithm. If* $R(x, w) = 1$*, output* $(\texttt{Verification}, s, x, \pi, 1)$ *for a successful extraction. Else if* $R(x, w) = 0$*, output* `Fail`*.*

For convenience and when the context is clear, we drop the message type and `sid` *from the notation.*

3.3 The CRS Ideal Functionality

Below is the ideal common reference string (CRS) functionality for a session s, which relies on a generic "`GenCRS`" algorithm. In Section 5.1, we will articulate the properties that `GenCRS` must have for the purposes of our construction.

Definition 10 (CRS Ideal Functionality). *The ideal functionality* $\mathcal{F}_{\texttt{CRS}}^{s}$ *of a* common reference string (CRS) *for a particular SID* s *and a CRS generation mechanism* `GenCRS` *is defined as follows.*

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

3.4 GUC Security Definitions

We are now ready to formalize what it means for a protocol Π to be a GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$- and $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid models. We review the standard GUC model real- and ideal-world experiments given by Canetti et al. [20] in Appendix A.9.

Definition 11 (GUC NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid Model). *A protocol* $\Pi = (\texttt{Setup}, \texttt{Prove}, \texttt{Verify}, \texttt{SimSetup}, \texttt{SimProve}, \texttt{Extract})$ *with security parameter* λ GUC-realizes the NIZKPoK ideal functionality $\mathcal{F}_{\texttt{NIZK}}$ in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid

model *if for all efficient* $\mathscr{A}$*, there exists an ideal adversary* $\mathcal{S}$ *efficient in expectation such that for all efficient environments* $\mathcal{Z}$*,*

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

where $\mathcal{G}_{\mathtt{rpoRO}}$ *is the global RO from Definition 8 and* aux *is any auxiliary information provided to the environment.*

Definition 12 (GUC NIZKPoK in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid Model). *A protocol* $\Pi = (\mathtt{Setup}, \mathtt{Prove}, \mathtt{Verify}, \mathtt{SimSetup}, \mathtt{SimProve}, \mathtt{Extract})$ *with security parameter* λ GUC-realizes *the NIZKPoK ideal functionality* $\mathcal{F}_{\mathtt{NIZK}}$ *in the* $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{NIZK}}$ *hybrid model if for all efficient* $\mathscr{A}$*, there exists an ideal adversary* $\mathcal{S}$ *efficient in expectation such that for all efficient environments* $\mathcal{Z}$*,*

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

where $\mathcal{G}_{\mathtt{roRO}}$ *is the global RO functionality from Definition 7,* $\mathcal{F}_{\mathtt{CRS}}$ *is the local CRS functionality from Definition 10, and* aux *is any auxiliary information provided to the environment.*

3.5 GUC NIZKPoK Are NIM-SHVZK and NI-SSS

We prove in this section that any protocol Π that GUC-realizes $\mathcal{F}_{\mathtt{NIZK}}$ in *any* global ROM must be non-interactive multiple special honest-verifier zero-knowledge (NIM-SHVZK) and non-interactive special simulation simulation sound (NI-SSS) according to the definitions in Section 2.2. In other words, the NIM-SHVZK and NI-SSS properties guaranteed by the straight-line compiler (SLC) in Definition 3 are *strictly necessary* to create GUC NIZKPoK in the global ROM.

As shown in Appendix B.1, any ordinary Σ-protocol that is regular SHVZK is also multi-SHVZK, and any multi-SHVZK can be made NIM-SHVZK. The more interesting result is the necessity of special simulation soundness, since that is not a property guaranteed by all Σ-protocols—it will be up to the SLC to create a special simulation sound NISLE Σ-protocol even when the underlying Σ-protocol is only regular special sound. In the proof of Theorem 3 in the full version of his paper [30], Fischlin shows that the NISLE Σ-protocols resulting from his transform satisfy both NIM-SHVZK and NI-SSS. A key element in Fischlin's proof that will surface again in the proof of Lemma 3 below, as well as in the proofs of Theorem 3 in Section 5.3 and Theorem 4 in Section 6.1, is the observation that a non-interactive `Extract` algorithm functionally decouples the extraction process from the rest of the experiment—interacting with the extractor does not influence the adversary's view in any way. Intuitively, this is because `Extract` works solely using adversarial outputs $(X, \Pi, \mathcal{Q}_{\mathscr{A}})$ that the adversary already knows. We use some of Fischlin's ideas in the proofs below, and extend his result to Kondi and shelat's randomized transform in Section 6.1.

Since we wish to prove our result for *any* global ROM, we recall the strict global RO $\mathcal{G}_{\mathtt{sRO}}$ outlined by Canetti et al. [22] and formalized by Camenisch et al. [10]. $\mathcal{G}_{\mathtt{sRO}}$ has the same parameters as $\mathcal{G}_{\mathtt{rpoRO}}$ and $\mathcal{G}_{\mathtt{roRO}}$ but only one interface, `Query`, that can be accessed freely by any party in the GUC experiment. `Query`

is defined similarly to the `Query` functionalities of $\mathcal{G}_{\mathtt{roRO}}$ and $\mathcal{G}_{\mathtt{rpoRO}}$ above, except that it does not need to store any extra lists of illegitimate queries. The functionality of $\mathcal{G}_{\mathtt{sRO}}$ is the minimal-most assumption of an RO in the GUC model, creating a direct correspondence to the standard RO H in the NIM–SHVZK and NI–SSS experiments. Because the point of using $\mathcal{G}_{\mathtt{sRO}}$ here is to convey the *minimal* assumption needed and not to prove the result *only* for $\mathcal{G}_{\mathtt{sRO}}$, we use the generic notation $\mathcal{G}_{\mathtt{RO}}$, which represents any global RO with a minimum of $\mathcal{G}_{\mathtt{sRO}}$'s `Query` interface. The GUC security definition in the $\mathcal{G}_{\mathtt{RO}}$-hybrid model is the same as in Definition 11, except that $\mathcal{G}_{\mathtt{rpoRO}}$ is replaced with $\mathcal{G}_{\mathtt{RO}}$ in the notation.

In the proof of the following theorem, our reduction will be playing the role of the environment against a GUC experiment challenger using the NIM–SHVZK and NI–SSS adversaries as black boxes. Using $\mathcal{G}_{\mathtt{RO}}$ makes sense in this context given that the reduction/environment does not have any special control over $\mathcal{G}_{\mathtt{RO}}$, and cannot make use of any privileged interfaces. We still assume, however, that the adversary playing the NI–SSS game outputs the information $\mathcal{Q}_{\mathscr{A}}$ for the reduction to use. Recall from Section 2.2 that while we use the notation $\mathcal{Q}_{\mathscr{A}}$ for convenience in our constructions (which all assume $\mathcal{Q}_{\mathscr{A}}$ is a list of $\mathscr{A}$'s queries to the RO), it can represent *any* information that the extractor needs. Finding a sufficient global RO to bootstrap other forms of SLCs that do not rely on $\mathscr{A}$'s RO query history into the GUC setting is left for future work.

Theorem 1. *Let Π be a protocol that GUC-realizes $\mathcal{F}_{\mathtt{NIZK}}$ in the $\mathcal{G}_{\mathtt{RO}}$-hybrid model (Definition 11 where $\mathcal{G}_{\mathtt{rpoRO}}$ is replaced with $\mathcal{G}_{\mathtt{RO}}$. Then Π must be both NI multi-SHVZK (Definition 5) and NI special simulation-sound (Definition 6).*

Proof Sketch. We proceed by cases and show that if Π is not NIM-SHVZK then it does not GUC-realize $\mathcal{F}_{\mathtt{NIZK}}$, and similarly that if Π is not NI-SSS then it does not GUC-realize $\mathcal{F}_{\mathtt{NIZK}}$. The full version of this proof is available in Appendix B.2.

In the first half of the proof, we construct a reduction that uses an adversary that can win the NIM–SHVZK experiment from Figure 1 with non-negligible advantage to determine whether it is talking to the real- or ideal-world GUC experiment. The reduction forwards $\mathscr{A}$'s oracle queries to and from $\mathcal{G}_{\mathtt{RO}}$ and $\mathscr{A}$'s prove queries to the GUC challenger, returning the proofs it receives back to $\mathscr{A}$. We note that since the reduction has no control over $\mathcal{G}_{\mathtt{RO}}$, its view of $\mathcal{G}_{\mathtt{RO}}$ is exactly the same as $\mathscr{A}$'s, so anything $\mathscr{A}$ can learn about the proofs from interacting with $\mathcal{G}_{\mathtt{RO}}$, the reduction can also learn. Furthermore if the GUC challenger is running the ideal-world experiment and $\mathcal{F}_{\mathtt{NIZK}}$ outputs `Fail` (indicating that `SimProve` failed to compute a valid proof for a statement-witness pair $(x, w) \in R$), the reduction can immediately tell it is living in the ideal world, since the real proof system is always complete (unless we are working in Fischlin's overwhelming completeness model, in which case we allow for a negligible completeness error). As long the `SimProve` algorithm does not produce `Fail`, the reduction simulates $\mathscr{A}$'s exact view of the challenger in the NIM–SHVZK game and succeeds in distinguishing the real- from ideal-world GUC experiments with the same probability as $\mathscr{A}$, arriving at the contradiction.

The second reduction uses an $\mathscr{A}$ that can win the NI–SSS game from Figure 2 with non-negligible advantage in order to distinguish between the GUC

experiments. This reduction proceeds similarly to the last, forwarding all of $\mathscr{A}$'s queries to the relevant parties. The argument regarding the reduction's view of $\mathcal{G}_{\mathtt{RO}}$ is identical to the argument above. In this case, however, there is a nuance to $\mathscr{A}$'s view: the regular NI–SSS challenger always produces *simulated* proofs, while the reduction will only produce simulated proofs if the GUC challenger is running the ideal-world experiment. We argue that in the case that the GUC challenger is running the real-world experiment, $\mathscr{A}$'s view from the reduction reduces to the regular non-interactive special soundness game NI–SS given in Appendix A.5, in which $\mathscr{A}$ gets to query a regular `Prove` oracle rather than querying `SimProve`. The reduction therefore runs two copies of $\mathscr{A}$, returning proofs from the GUC challenger to the first copy $\mathscr{A}$ and generating proofs for the second copy $\mathscr{A}'$ itself using Π.`Prove`. If the GUC challenger is running the ideal-world experiment, the reduction is able to simulate $\mathscr{A}$'s exact view of the NI–SSS game, and the reduction will be able to determine that it is living in the ideal-world experiment with the same probability that $\mathscr{A}$ is able to output a proof that causes $\mathcal{F}_{\mathtt{NIZK}}$'s `Extract` algorithm to output `Fail`. If the GUC challenger is running the real-world experiment and $\mathscr{A}'$ can output a valid proof such that Π.`Extract` fails but the GUC challenger outputs 1, the reduction knows it is the real-world challenger, and can therefore distinguish the GUC experiments with the same probability that $\mathscr{A}'$ succeeds in winning the NI–SS game.

Note this proof requires the reduction to be able to compute Π.`Extract` itself, which it can only do because $\mathscr{A}'$ provides the auxiliary information $\mathcal{Q}_{\mathscr{A}'}$ somewhere the reduction can see.[1] Similarly, the reduction would not work if Π was *only* special sound, since the adversary in the NI–SS game does not have well-defined behavior with respect to simulated proofs. □

4 GUC NIZKPoK in the Programmable Global ROM

We will now prove that a straight-line compiler (SLC) is sufficient to transform any Σ-protocol into a GUC-secure NIZKPoK in the the $\mathcal{G}_{\mathtt{rpoRO}}$-hybrid model.

Theorem 2. *Let Σ_R be any Σ-protocol for relation R (Definition 2), $\mathcal{G}_{\mathtt{rpoRO}}$ be the restricted programmable observable global random oracle (Definition 7), and* SLC *be any straight-line compiler (Definition 3). Then the NISLE Σ-protocol $\Sigma_R^{\mathtt{SLC}} \leftarrow \mathtt{SLC}(\Sigma_R)$ GUC-realizes $\mathcal{F}_{\mathtt{NIZK}}$ in the $\mathcal{G}_{\mathtt{rpoRO}}$-hybrid model (Definition 11).*

Proof Sketch. In the ideal-world experiment, our simulator $\mathcal{S}$ hands the ideal functionality $\mathcal{F}_{\mathtt{NIZK}}$ the tuple of algorithms $\Sigma_R^{\mathtt{SLC}}$, returns `false` to the corrupted parties' `IsProgrammed` queries, and otherwise functions as a dummy adversary, forwarding communications between the environment and the protocol.

We proceed by creating a hybrid reduction starting in the real-world experiment that replaces each piece of the real-world protocol $\Sigma_R^{\mathtt{SLC}}$ with the functionality of $\mathcal{F}_{\mathtt{NIZK}}$. First, we replace all of the environment's and adversary's

[1] As for the GUC challenger, the only hope it has of successfully running the extractor against the reduction in this case is to find information about $\mathscr{A}'$'s proofs somewhere outside of the entire experiment—in the $\mathcal{G}_{\mathtt{RO}}$-hybrid setting, the only option is for the GUC challenger to get special access to $\mathcal{G}_{\mathtt{RO}}$.

connections to the real-world protocol participants with the "challenger" of our reduction, $\mathcal{C}$. This difference is syntactic, so the first two hybrids are identical.

In the next hybrid, we replace $\mathcal{C}$'s `Prove` functionality with the **Prove** interface of $\mathcal{F}_{\texttt{NIZK}}$, and show the environment's views are indistinguishable between these experiments as long as Σ_R is non-interactive multiple special honest-verifier zero-knowledge (NIM-SHVZK). Recall that the **Prove** functionality of $\mathcal{F}_{\texttt{NIZK}}$ uses the $\Sigma_R^{\texttt{SLC}}.\texttt{SimProve}$ algorithm, which simulates proofs the same way as $\Sigma_R.\texttt{Prove}$—by programming the RO to return a challenge it gets in advance. In addition, $\mathcal{C}$ always returns `false` to any of the adversary's `IsProgrammed` queries. As long as $\Sigma_R^{\texttt{SLC}}.\texttt{SimProve}$ 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 superlogarithmic commitment entropy property in Defintion 22 and the restriction of the `IsProgrammed` interface) and the real and simulated proofs are also statistically indistinguishable (which follows from the statistical NIM-SHVZK property of the straight-line extractor), the two hybrids are statistically indistinguishable. In the event that Σ_R is only *computationally* SHVZK, we construct a (tight) reduction that uses an environment that can distinguish the two hybrids to win the NIM–SHVZK game from Figure 1. The reduction simply proceeds by forwarding all of the environment's RO queries to $\mathcal{G}_{\texttt{rpoRO}}$, all `Prove` queries to the NIM–SHVZK challenger, and answering `Verify` queries itself by running $\Sigma_R^{\texttt{SLC}}.\texttt{Verify}$. If the NIM–SHVZK challenger playing with bit $b = 0$ and the proofs are according to $\Sigma_R^{\texttt{SLC}}.\texttt{Prove}$, this is exactly what the environment expects from the first hybrid; otherwise if $b = 1$ and the proofs are according to $\Sigma_R^{\texttt{SLC}}.\texttt{SimProve}$, the environment knows it is living in the second hybrid. Therefore, our reduction succeeds with the same probability as the hybrid-distinguisher environment, contradicting the NIM-SHVZK property of $\Sigma_R^{\texttt{SLC}}$ and proving that the hybrids must be computationally indistinguishable.

In the penultimate hybrid, we replace $\mathcal{C}$'s `Verify` functionality with the **Verify** functionality of $\mathcal{F}_{\texttt{NIZK}}$, and show the environment's views are computationally indistinguishable between these hybrids as long as Σ_R is non-interactive special simulation-sound (NI-SSS). Recall that the **Verify** functionality of $\mathcal{F}_{\texttt{NIZK}}$ uses the $\Sigma_{\texttt{R}\vee\texttt{S}}^{\texttt{SLC}}.\texttt{Extract}$ algorithm, and fails whenever the witness extracted from a valid (non-simulated) proof is not valid. Our reduction uses an environment that can distinguish the simulate-only hybrid from the simulate-and-extract hybrid as a black box to produce a proof that wins the NI–SSS game from Figure 2 as follows. For `Prove` queries, the reduction simulates proofs according to either hybrid (both use $\Sigma_R^{\texttt{SLC}}.\texttt{SimProve}$). Any time the environment wants to verify a proof that the reduction did not create itself, it gathers the environment's queries (which are freely available—recall that all of the environment's wires pass through our challenger $\mathcal{C}$) and sends the proof along with the environment's queries to the NI–SSS challenger. Note that since the only difference between the hybrids is that the second hybrid can output `Fail` while the first never does, the only way for the environment to distinguish between them is to produce such a failure by outputting a valid (non-simulated) proof that causes $\Sigma_R^{\texttt{SLC}}.\texttt{Extract}$ to fail. Since the challenger in the NI–SSS game also uses the

$\Sigma_R^{\mathtt{SLC}}.\mathtt{Extract}$ algorithm, the reduction succeeds with the probability as the environment, contradicting the NI–SSS property and proving that the hybrids must be computationally indistinguishable.

The final step is to replace the challenger $\mathcal{C}$ with $\mathcal{F}_{\mathtt{NIZK}}$ and $\mathcal{S}$. Note that since $\mathcal{C}$ already runs the algorithms of $\mathcal{F}_{\mathtt{NIZK}}$ and returns `false` to corrupted parties' `IsProgrammed` queries, this is again only a syntactic difference, and the last two hybrids are identical. The full version of this proof is in Appendix B.3. □

5 GUC NIZKPoK in the Observable Global ROM

Recall from Section 1 that in order to avoid the session-localized `IsProgrammed` interface, we pursue GUC-secure NIZKPoK in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model, where $\mathcal{F}_{\mathtt{CRS}}$ outputs a CRS that real-world participants might generate at the beginning of a session using Canetti et al.'s GUC-secure NISC protocol [22].

We begin by discussing the properties we need from our CRS generation mechanism, then introduce a GUC compiler `guc` that creates GUC-secure NIZKPoK in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model.

5.1 Generating a CRS that Plays Nice with Σ-protocols

In our construction, the prover convinces the verifier that either it knows a "real" witness w_0 to the statement x_0, or it knows the trapdoor `trap` to the CRS. In the real world, nobody knows `trap` (as long as the CRS is generated securely, for instance using the NISC protocol). Therefore, all proofs executed by the regular `Prove` algorithm will be using the real witness w_0 for the statement x_0. In the ideal world, the simulator gets to generate the CRS for each session s with a trapdoor as part of the `SimProve` algorithm. `SimProve` is otherwise the same as `Prove`, except the witness is always the trapdoor $\mathtt{trap}_s$ for the statement $\mathtt{CRS}_s$.

In order for this OR-proof to work, `Prove` and `SimProve` must be able to interpret the CRS as a statement x_1 with a corresponding trapdoor witness w_1 such that the pair (x_1, w_1) satisfies some binary $\mathcal{NP}$ relation S. For efficiency purposes (since the simulator must run in polynomial-time) the CRS must be efficiently computable, and for security purposes, the trapdoor must be difficult to compute. We call a relation that satisfies the efficiency property *samplable* and a relation that satisfies the security property *hard*. The intuition is similar to that of Fischlin's one-way instance generator [31].

Definition 13 (Samplable-Hard Relation). *A binary $\mathcal{NP}$ relation S is* samplable-hard *with respect to a security parameter λ if it has the following properties.*

1. **Sampling a statement-witness pair is easy.** *There exists a sampling algorithm κ_S that on input 1^λ 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 $\mathscr{A}$,

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

Finally, we require that the relation S underlying the CRS has an efficient corresponding Σ-protocol Σ_S. Our construction will instantiate an OR-protocol $\Sigma_{\mathtt{R}\vee\mathtt{S}}$ based on Σ_R and Σ_S for the relation $R \vee S$.

Putting all of the pieces together, the CRS generation mechanism $\mathtt{GenCRS}$ for our construction fixes S as a samplable-hard relation with corresponding efficient Σ-protocol Σ_S, and consists of sampling (x, w) using $\kappa_S(1^\lambda)$. We combine the local CRS functionality $\mathcal{F}_{\mathtt{CRS}}$ based on the above $\mathtt{GenCRS}$ mechanism with the restricted observable global RO $\mathcal{G}_{\mathtt{roRO}}$ to instantiate the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model, and are now ready to introduce our GUC compiler algorithm.

5.2 GUC Compiler

We propose a compiler that uses any SLC in conjunction with the OR-protocol described in Section 5.1 above to transform any Σ-protocol into a GUC-secure NIZKPoK in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model. The compiler works as follows.

The samplable-hard relation S and corresponding efficient Σ-protocol Σ_S underlying the CRS are fixed, and the algorithm $\mathtt{GenCRS}$ for $\mathcal{F}_{\mathtt{CRS}}$ is fixed as $(x, w) \leftarrow \kappa_S(1^\lambda)$. The real-world $\mathtt{Setup}$ functionality runs the OR-protocol $\Sigma_{\mathtt{R}\vee\mathtt{S}}$ based on Σ_R and Σ_S through the SLC to obtain $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}$, and returns the same setup parameters as $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}$.

For each SID s, provers in the real world query the CRS ideal functionality $\mathcal{F}^s_{\mathtt{CRS}}$ to obtain $\mathtt{CRS}_s$. Each time a prover with SID s needs to create a proof of a statement x using witness w, it generates a proof that either it knows a witness or the CRS trapdoor using $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Prove}^{\mathcal{G}_{\mathtt{roRO}}}(X, W)$, where $X = (x, \mathtt{CRS}_s)$ and $W = (w, 0)$ to denote the supplied witness is for the first statement, x. In order to verify the proof, a verifier first parses $X = (x_0, x_1)$, obtains $\mathtt{CRS}_s$ from $\mathcal{F}^s_{\mathtt{CRS}}$, then checks whether $x_1 = \mathtt{CRS}_s$. If it does, it the verifier outputs the result of running $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Verify}^{\mathcal{G}_{\mathtt{roRO}}}(X, \Pi) = 1$.

In the ideal world, the $\mathtt{SimSetup}$ algorithm is the same as the real-world $\mathtt{Setup}$, except that it additionally generates an empty list in which to store the simulated CRS for each session, denoted $\mathtt{simcrs}$. When it is time to prove a statement x on behalf of an honest party in session s, the $\mathtt{SimProve}$ algorithm generates $(\mathtt{CRS}_s, \mathtt{trap}_s) \leftarrow \kappa_S(1^\lambda)$ (if one has not been generated already), and computes the proof using $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Prove}^{\mathcal{G}_{\mathtt{roRO}}}(X, W)$, where $X = (x, \mathtt{CRS}_s)$ and $W = (\mathtt{trap}_s, 1)$ to denote the supplied witness is for the second statement, $\mathtt{CRS}_s$.

Given a proof (X, Π) for session s, a list $\mathcal{Q}_{\mathscr{A}}$ of adversarial provers' queries for session s, and the CRS list $\mathtt{simcrs}$, the $\mathtt{Extract}$ algorithm first runs the same checks as $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Verify}$. If they pass, $\mathtt{Extract}$ runs $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Extract}(X, \Pi, \mathcal{Q}^s_{P^*})$ and tests the witness $W = (w_0, w_1)$. If $R_{\mathtt{R}\vee\mathtt{S}}(X, W) = 1$ but $R(x_0, w_0) = 0$, $\mathtt{Extract}$ outputs $\mathtt{Fail}$. Otherwise, it outputs W.

Note that this formulation diverges from the general intuition of an OR-protocol extractor (see Appendix A.7) in that we require any valid witness W

to imply that $R(x_0, w_0) = 1$, not that either $R(x_0, w_0) = 1$ *or* $S(x_1, w_1) = 1$. This is because we need to account for the fact that $\mathcal{F}_{\mathtt{NIZK}}$ will never invoke the $\mathtt{Extract}$ algorithm on proofs it has generated using $\mathtt{SimProve}$, and nobody else should ever have access to the CRS trapdoor. If $\mathcal{F}_{\mathtt{NIZK}}$ gets a proof that verifies because $S(\mathtt{CRS}_s, w_1) = 1$, it must be the case that an adversarial prover has acquired the trapdoor, and $\mathtt{Extract}$ forms its output in such a way that $\mathcal{F}_{\mathtt{NIZK}}$ will output $\mathtt{Fail}$. In our proof of security, we will bound the probability of this failure by constructing a reduction to the hardness property of S.

We give a candidate construction of the compiler below, and prove in Section 5.3 that it creates GUC-secure NIZKPoK in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model.

Definition 14 (Candidate Compiler). *Let Σ_R be any Σ-protocol for relation R (Definition 2), $\mathcal{G}_{\mathtt{roRO}}$ be the restricted observable global random oracle (Definition 7), $\mathcal{F}_{\mathtt{CRS}}$ be the ideal CRS functionality (Definition 10), Σ_S be a Σ-protocol for samplable-hard relation S (Definition 13), and $\mathtt{SLC}$ be any straight-line compiler (Definition 3). Then our* candidate compiler $\mathtt{guc}$ *is an algorithm that on input Σ_R and $\mathtt{SLC}$, produces a tuple of algorithms $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}} = (\mathtt{Setup}^{\mathcal{G}_{\mathtt{roRO}}}_{\Sigma_R}, \mathtt{Prove}^{\mathcal{G}_{\mathtt{roRO}},\mathcal{F}_{\mathtt{CRS}}}_{\Sigma_R}, \mathtt{Verify}^{\mathcal{G}_{\mathtt{roRO}},\mathcal{F}_{\mathtt{CRS}}}_{\Sigma_R}, \mathtt{SimSetup}_{\Sigma_R}, \mathtt{SimProve}_{\Sigma_R}, \mathtt{Extract}_{\Sigma_R})$, defined in Figure 3.*

5.3 Realizing $\mathcal{F}_{\mathtt{NIZK}}$ in the $\mathcal{G}_{\mathtt{RO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid Model

We now prove that the algorithm $\mathtt{guc}$ compiles any Σ-protocol into a GUC-secure NIZKPoK in the $\mathcal{G}_{\mathtt{RO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model.

Theorem 3. *Let Σ_R be any Σ-protocol for relation R (Definition 2), $\mathcal{G}_{\mathtt{roRO}}$ be the restricted observable global random oracle (Definition 7), $\mathtt{SLC}$ be any straight-line compiler (Definition 3), $\mathcal{F}_{\mathtt{CRS}}$ be the ideal CRS functionality (Definition 10), Σ_S be a Σ-protocol for a samplable-hard relation S (Definition 13), and $\mathtt{guc}$ be the algorithm described in Definition 14. Then the NISLE Σ-protocol $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}} \leftarrow \mathtt{guc}(\Sigma_R, \mathtt{SLC})$ GUC-realizes $\mathcal{F}_{\mathtt{NIZK}}$ in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model (Definition 12).*

Proof Sketch. The proof proceeds similarly to that of Theorem 2 in Section 4, where we construct a sequence of hybrid experiments that transition between the real- and ideal-world executions of the compiled Σ-protocol $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}} \leftarrow \mathtt{guc}(\Sigma_R, \mathtt{SLC})$. In the ideal-world experiment, our simulator $\mathcal{S}$ hands the ideal functionality $\mathcal{F}_{\mathtt{NIZK}}$ the tuple of algorithms $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}$ and otherwise functions as a dummy adversary, forwarding communications between the environment and the protocol. Throughout this proof when we say an argument is identical to an argument from the proof of Theorem 2, we mean identical up the handling of $\mathcal{G}_{\mathtt{rpoRO}}$'s $\mathtt{IsProgrammed}$ interface, which does not exist in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model.

The first hybrid is identical to the first hybrid in the proof of Theorem 2: we replace all of the real-world protocol participants, $\mathcal{G}_{\mathtt{roRO}}$, and now $\mathcal{F}_{\mathtt{CRS}}$ with a challenger $\mathcal{C}$ who controls all of the wires in and out of the environment and the adversary, noting this step permits $\mathcal{C}$ to program $\mathcal{G}_{\mathtt{roRO}}$.[2] The second hybrid is also

[2] As discussed by Camenish et al. [10], the challenger in such a hybrid experiment can make use of techniques like programming and rewinding that are "illegal" for the ideal-world simulator to employ in the GUC model.

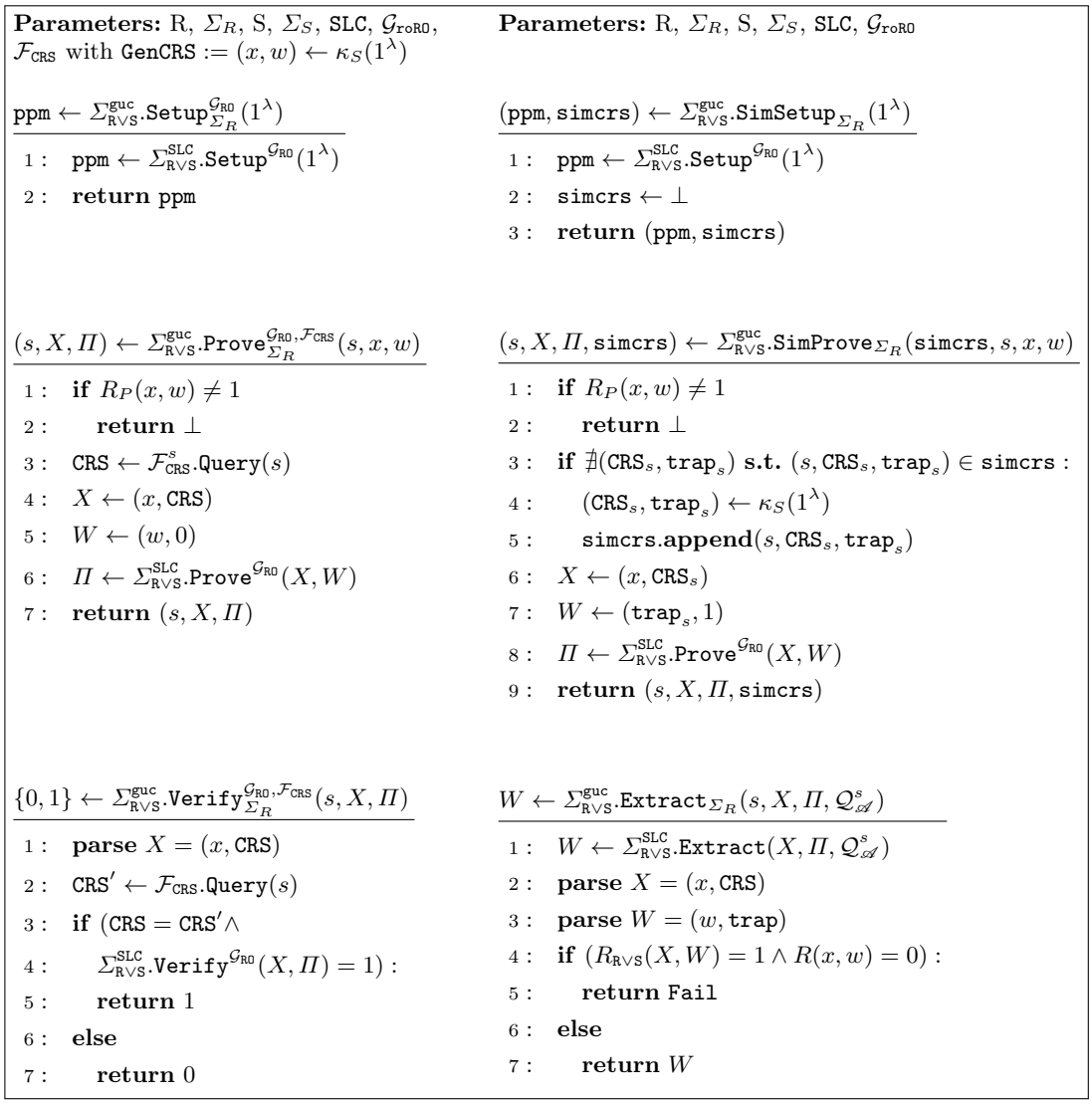

Fig. 3. Compiler $\Sigma^{\text{guc}}_{\text{R}\vee\text{S}} \leftarrow \text{guc}(\Sigma_R, \text{SLC})$ for Σ_R in the $\mathcal{G}_{\text{roRO}}$-$\mathcal{F}_{\text{CRS}}$-hybrid Model

identical to the one in the proof of Theorem 2 above, except instead of jumping straight to replacing $\mathcal{C}$'s real-world $\texttt{Prove}$ algorithm with the **Prove** interface of the ideal functionality, which will use $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{SimSetup}$ and $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{SimProve}$, we instead replace $\texttt{Prove}$ with $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{SimSetup}$ and $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{SimProve}$. This step allows us to postpone giving $\mathcal{C}$ access to the CRS trapdoors, since we will need to ensure that any adversarially-created proofs in the next hybrid will only avoid extraction if the adversary is somehow able to generate the trapdoor itself. By the arguments used in the proof of Theorem 2, we can reduce the indistinguishability of the first two hybrids to the NIM-SHVZK property of $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}$.

The third hybrid is identical in that we replace $\mathcal{C}$'s $\texttt{Verify}$ procedure with $\mathcal{F}_{\texttt{NIZK}}$'s **Verify** interface, which uses $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{Extract}$. The proof of indistinguishability of the second and third hybrids will differ slightly due to the new failure condition in the $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{Extract}$ algorithm: namely, the clause that says if the overall witness $W = (w, \texttt{trap}_s)$ is a valid witness for the statement $X = (x, \texttt{CRS}_s)$ but w is not a valid witness for x, output $\texttt{Fail}$. We can limit the probability of this failure by constructing a reduction to the hardness property of the samplable-hard relation: if the environment is able to produce a proof that meets the failure condition, the reduction can produce a tuple $(\texttt{CRS}_s, \texttt{trap}_s)$ given only $\texttt{CRS}_s$ that was the output of running the sampling algorithm $\kappa_S(1^\lambda)$. Since the probability of generating such a tuple is negligible by the hardness property of S, the probability of such a failure is similarly negligible. The only other way for the environment to distinguish the hybrids is to produce a valid, non-extractable proof of a statement X, such that $R_{\texttt{R}\vee\texttt{S}}(X, W) = 0$ for $W \leftarrow \Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{Extract}(X, W)$. In this case, $\mathcal{C}$ can use this proof to contradict the NI-SSS (or NI-SS) property of $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}$ in the exact same way as the parallel reduction in the proof of Theorem 2.

Finally, the penultimate hybrid replaces $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{SimSetup}$ and $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{SimProve}$ with the candidate compiler's algorithms $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{SimSetup}$ and $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{SimProve}$. This step effectively reverts the proofs back to the real-world $\texttt{Prove}$ mechanism, except $\mathcal{C}$ is using trapdoors rather than real witnesses. If the underlying OR-protocol $\Sigma_{\texttt{R}\vee\texttt{S}}$ is statistical SHVZK (and statistically witness-indistinguishable according to Theorem 5 in Appendix A.7), then there is automatically negligible difference in view between the third and penultimate hybrids. If, however, there is computational wiggle room between the proofs in the two experiments, *and* the distinguisher environment now has access to the extractor, we must ensure that the *only* way the environment can distinguish the hybrids is by the contents of the proofs (as opposed to somehow using its view of the new proofs to cause the extractor to fail). We argue here that because the straight-line extractor works exclusively based on statements, proofs, and oracle queries that the environment made itself, anything the environment can learn from the extractor it could have learned on its own. Therefore, it cannot have possibly learned anything new about the hybrids from the extractor, and the reduction to computational NIM-SHVZK proceeds the same as before.

The last hybrid replaces $\mathcal{C}$ with $\mathcal{F}_{\texttt{NIZK}}$ and $\mathcal{S}$—this is again a syntactic rearrangement, and is functionally identical to the ideal-world experiment. The full version of this proof is in Appendix B.4. □

6 Constructions via the Randomized Fischlin Transform

We demonstrated in the last two sections that any special straight-line compiler (SLC) that satisfies Definition 3 is sufficient to transform any Σ-protocol Σ_R into a GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model, and sufficient in conjunction with our OR-protocol construction to make Σ_R a GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model. In this section, we will show that the randomized Fischlin transform [31,36] meets our definition of an SLC for a broad class of Σ-protocols, and therefore enables us to practically instantiate both GUC compilers. The efficiency of the resulting proof systems reduce to the efficiency of the randomized Fischlin transform, which requires only a linear increase in the size of the proofs for small multiplicative and additive constants.

In this section, we review the randomized Fischlin transform `Fis` and show that it meets our definition of an SLC. We then apply `Fis` to efficiently realize GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$- and $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid models, respectively.

6.1 The Randomized Fischlin Transform, Revisited

Recall from Section 1 that the randomized Fischlin transform due to Kondi and shelat [36] is a version of the Fischlin transform [31,30] in which the challenges are selected uniformly at random from the challenge space. In order to use the randomized Fischlin transform in a way that preserves the security properties of the SLC given in Definition 3, the Σ-protocols used as input must have the *strong* special soundness property, which is a common property of most Σ-protocols [36] and notably does not preclude us from transforming an OR-protocol. The strong special soundness property 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 $\texttt{chl} = \texttt{chl}'$, as long as $\texttt{res} \neq \texttt{res}'$. We provide a definition of strong special soundness below, and a brief discussion of the necessity of strong special soundness in Appendix A.11.

Definition 15 (Strong Special Soundness). *A Σ-protocol Σ_R for relation R has* strong special soundness *if the condition $\texttt{chl} \neq \texttt{chl}'$ in the special soundness game (Definition 20) is replaced with the condition $(\texttt{chl}, \texttt{res}) \neq (\texttt{chl}', \texttt{res}')$.*

In the full version of his paper, Fischlin proves that the standard Fischlin transform over a Σ-protocol with the quasi-unique responses property creates a protocol that is both NIM-SHVZK and NI-SSS in the standard ROM [30]. Kondi and shelat show that the randomized Fischlin transform over a Σ-protocol with the more general strong special soundness property creates a protocol that is standard (non-multi) NI-SHVZK and standard (non-simulation) strong NI-SS [36]. Therefore, it suffices to show that the NI *multi*-SHVZK and strong special *simulation* soundness are similarly preserved under the randomized transform for strong special sound Σ-protocols. Our proof of the theorem below draws heavily on arguments from Fischlin [30] and Kondi and shelat [36]; the only novelty is in the (nearly verbatim) application of Fischlin's arguments for NIM-SHVZK and NI-SSS to the randomized transform. We therefore defer the technical details of

the randomized Fischlin transform to Definition 28 in Appendix A.12, and the full version of the proof to Appendix B.5.

Theorem 4. *Let Σ_R be a Σ-protocol for relation R (Definition 2) with strong special soundness (Definition 15). Then the randomized Fischlin transform* `Fis` *(Definition 28) is a straight-line compiler for Σ_R (Definition 3).*

Proof sketch. Recall that a straight-line compiler according to our definition must create protocols that are NIM-SHVZK and NIM-SSS. Kondi and shelat prove in Theorem 6.4 [36] that the tuple of algorithms $\Sigma_R^{\texttt{Fis}}$ (denoted $\pi_{\texttt{NIZK}}^{\texttt{F-rand}}$ in their paper) produced by running the randomized Fischlin transform on any strong special sound Σ-protocol Σ_R for relation R is a non-interactive straight-line extractable zero-knowledge proof of knowledge for L_R in the standard random-oracle model. Since Kondi and shelat use the standard definitions of SHVZK and strong special soundness (Definitions 19 and 15, respectively), it remains to show that $\Sigma_R^{\texttt{Fis}}$ satisfies NIM-SHVZK and NIM-SSS.

Fischlin shows in the proof of Theorem 3 [30] that the standard Fischlin transform satisfies the NIM-SHVZK and NI-SSS properties. Since the strong special soundness property replaces the quasi-unique responses property and the challenges in the randomized version are identically distributed to those in the original version, the proof of NIM-SHVZK and NI-SSS for the randomized Fischlin transform is almost identical to Fischlin's proof of Theorem 3. We discuss the minor differences in the full proof (Appendix B.5). □

6.2 Efficient, GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid Model

We demonstrated in Section 4 that any SLC is a GUC compiler for any Σ-protocol Σ_R in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model, and argued in Section 6.1 above that the transform `Fis` is an SLC that transforms any strong special sound Σ-protocol into a NISLE Σ-protocol. Therefore, $\texttt{Fis}(\Sigma_R)$ is a GUC compiler for any strong special sound Σ-protocol Σ_R in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model.

Corollary 1. *Let Σ_R be any strong special sound Σ-protocol for a relation R (Definitions 2 and 15) and* `Fis` *be the algorithm from Definition 28. Then $\Sigma_R^{\texttt{SLC}} \leftarrow \texttt{Fis}(\Sigma_R)$ GUC-realizes $\mathcal{F}_{\texttt{NIZK}}$ in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model.*

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

6.3 Efficient, GUC-secure NIZKPoK in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid Model

Our construction for the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model requires two layered compilers: any SLC, and the transform `guc` from Definition 14. We proved in Theorem 3 that $\texttt{guc}(\Sigma_R, \texttt{SLC})$ GUC-realizes $\mathcal{F}_{\texttt{NIZK}}$ for any Σ-protocol Σ_R, and again in Section 6.1 that `Fis` is an SLC. Therefore, $\texttt{guc}(\Sigma_R, \texttt{Fis})$ is a GUC compiler for any strong special sound Σ-protocol Σ_R in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model.

Corollary 2. *Let Σ_R be any strong special sound Σ-protocol for a relation R (Definitions 2 and 15) and* `guc` *be the algorithm from Definition 14. Then $\Sigma_{\texttt{R}\vee\texttt{S}}^{\texttt{guc}} \leftarrow \texttt{guc}(\Sigma_R, \texttt{Fis})$ GUC-realizes $\mathcal{F}_{\texttt{NIZK}}$ in the $\mathcal{G}_{\texttt{roRO}}$-$\mathcal{F}_{\texttt{CRS}}$-hybrid model.*

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

Acknowledgements

Many thanks to Yashvanth Kondi and abhi shelat for crucial security analysis of our original OR-protocol construction, and to Jack Doerner for insightful discussions about $\mathcal{F}_{\texttt{NIZK}}$ that inspired our results in Section 3.5. Thank you to all three for many productive conversations surrounding the interesting properties of straight-line extractors.

References

1. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium*, pages 335–348, 2008.
2. Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880, pages 255–270, 2000.
3. 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.
4. Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in) security of ros. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 33–53. Springer, 2021.
5. Manuel Blum, Alfredo De Santis, Silvio Micali, and Guiseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.
6. Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT '00*, pages 431–444, 2000.
7. Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *ePrint Archive*, 2017.
8. Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates—Building in Privacy*. PhD thesis, Eindhoven Inst. of Tech., The Netherlands, 1999.
9. 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.
10. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–312. Springer, 2018.
11. 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.
12. 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.
13. 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.
14. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *SCN 2002*, volume 2576, pages 268–289, 2003.
15. Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In *EUROCRYPT '99*, pages 107–122, 1999.

16. Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO '99*, volume 1666, pages 413–430, 1999.
17. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO '03*, volume 2729, pages 126–144, 2003.
18. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *CRYPTO '97*, pages 410–424. Springer Verlag, 1997.
19. 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.
20. 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.
21. Ran Canetti and Marc Fischlin. Universally composable commitments. In *Annual International Cryptology Conference*, pages 19–40. Springer, 2001.
22. 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.
23. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045, pages 280–300. Springer Verlag, 2001.
24. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Annual International Cryptology Conference*, pages 174–187. Springer, 1994.
25. Ronald Cramer, Ivan Damgård, Chaoping Xing, and Chen Yuan. Amortized complexity of zero-knowledge proofs revisited: Achieving linear soundness slack. In *Advances in Cryptology - EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 479–500, 2017.
26. Ivan Damgård. On σ-protocols. University of Aarhus, Department of Computer Science, 2002.
27. Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE, 2019.
28. Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. 29(1):1–28, 1999.
29. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.
30. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. 2005.
31. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Annual International Cryptology Conference*, pages 152–168. Springer, 2005.
32. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO '97*, pages 16–30, 1997.
33. Eu-Jin Goh and Stanisław Jarecki. A signature scheme as secure as the diffie-hellman problem. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 401–415. Springer, 2003.
34. Carmit Hazay and Yehuda Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.
35. Shuichi Katsumata. A new simple technique to bootstrap various lattice zero-knowledge proofs to qrom secure nizks. In *Annual International Cryptology Conference*, pages 580–610. Springer, 2021.

36. Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. *Cryptology ePrint Archive*, 2022.
37. Helger Lipmaa. Statistical zero-knowledge proofs from diophantine equations. Manuscript. Available from http://eprint.iacr.org/2001/086, 2001.
38. Vadim Lyubashevsky. Lattice signatures without trapdoors. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755. Springer, 2012.
39. Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: simple two-round schnorr multi-signatures. In *Annual International Cryptology Conference*, pages 189–221. Springer, 2021.
40. Rafael Pass. On deniability in the common reference string and random oracle model. In *Annual International Cryptology Conference*, pages 316–337, 2003.
41. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '92*, volume 576, pages 129–140, 1992.
42. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *International conference on the theory and applications of cryptographic techniques*, pages 387–398. Springer, 1996.
43. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 566–598. Springer, 2001.
44. Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 755–784. Springer, 2015.
45. David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.
46. John Watrous. Zero-knowledge against quantum attacks. *SIAM Journal on Computing*, 39(1):25–58, 2009.
47. Douglas Wikström. A commitment-consistent proof of a shuffle. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, pages 407–421. Springer, 2009.

Appendix

A Supplementary Definitions

A.1 Notation

We use λ for the security parameter, and say an algorithm $\mathscr{A}$ is efficient in λ if its runtime can be expressed as a polynomial $\mathsf{poly}(\lambda)$ on input λ. We say a function negl is negligible in λ 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 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 λ, 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 Protocol Template

Definition 16 (Protocol Template for Relation R). [26,34] *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 ℓ-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).*

A.3 Σ-protocols

Definition 17 (Σ-protocol). [26,34] *A protocol Π is a* Σ-protocol *for relation R if it is a three-round public-coin protocol of the form in Definition 1 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 $(\texttt{com}, \texttt{chl}, \texttt{res})$ and $(\texttt{com}, \texttt{chl}', \texttt{res}')$ for x where $\texttt{chl} \neq \texttt{chl}'$, outputs w such that $(x, w) \in R$.*
- **Special honest verifier zero knowledge**: *There exists a PPT simulator M, which on input x and $\texttt{chl}$ outputs a transcript of the form $(\texttt{com}, \texttt{chl}, \texttt{res})$ 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 $\texttt{chl} \in \{0,1\}^\ell$ it holds that*

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

where $M(x, \texttt{chl})$ denotes the output of simulator M on input x and $\texttt{chl}$, 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 $\texttt{chl}$.

The value ℓ is called the challenge length.

A.4 Standard Σ-protocol Security Definitions

We will now formalize the completeness, special honest-verifier zero-knowledge, and special soundness properties. Other than notational differences, our formulation is due to Damgård [26].

The completeness property requires that any proof computed using the `Prove` algorithm on a valid statement-witness pair should induce the `Verify` algorithm to accept.

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

The special honest-verifier zero-knowledge (SHVZK) property essentially says that no efficient algorithm should be able to distinguish between proofs generated using the `Setup` and `Prove` algorithms from proofs generated using the `SimSetup` and `SimProve` algorithms. We formalize the SHVZK property as a game between an adversary algorithm $\mathscr{A}$ and a challenger $\mathcal{C}$ who is running one of two experiments: the $b = 0$ experiment in which $\mathcal{C}$ responds to $\mathscr{A}$'s queries $(\texttt{Prove}, x, w)$ by returning $\pi \leftarrow \Sigma_R.\texttt{Prove}((x, w), (x, \texttt{chl}))$, and the $b = 1$ experiment in which $\mathcal{C}$ responds to `Prove` queries by returning $\pi \leftarrow \Sigma_R.\texttt{SimProve}(x, z, \texttt{chl})$ for z generated using $\Sigma_R.\texttt{SimSetup}$. Note that since we are in the honest-verifier (public-coin) setting, we can assume $\mathcal{C}$ runs the proof process with the correct verifier algorithm whose challenge $\texttt{chl}$ is the contents of its random tape. As a result, the challenges in $\Sigma_R.\texttt{Prove}((x, w), (x, \texttt{chl}))$ and $\Sigma_R.\texttt{SimProve}(x, z, \texttt{chl})$ are identically distributed. The word "special" here refers to the fact that the `SimProve` algorithm gets to see the honest verifier's random tape, and thus the challenge, prior to computing the simulated proof. That is the simulator's advantage over a real prover who gets its challenge from the verifier only after it has issued its

commit message. We say a Σ-protocol is SHVZK with respect to a security parameter λ if the probability that $\mathscr{A}$ can distinguish between the two experiments is only negligibly better than a random guess.

In the original definition of Σ-protocols given in Appendix A.3, SHVZK is a *statistical* security property—that is, the experiments described above are statistically indistinguishable, and even an unbounded (non-PPT) $\mathscr{A}$ cannot distinguish them. Later in the paper we will prove that our GUC-compiler works for both statistical and computational SHVZK Σ-protocols. Therefore, we write our definition to permit both versions. Where there is no qualifier before SHVZK, the reader can assume we mean the traditional (statistical) notion of SHVZK.

Definition 19 (Special Honest-Verifier Zero-Knowledge). *A Σ-protocol Σ_R for relation R is statistical (resp. computational)* special honest-verifier zero-knowledge (SHVZK) *if there exist algorithms* `SimSetup` *and* `SimProve` *such that for any security parameter λ, any adversary (resp. any* PPT *adversary) $\mathscr{A}$, and a bit $b \leftarrow_{\$} \{0,1\}$, there exists some negligible function* 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* $\Pr[b' = b] > \frac{1}{2} + \mathsf{negl}(\lambda)$.

$\mathsf{SHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, 0)$: Real	$\mathsf{SHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, 1)$: Ideal
1 : $\texttt{ppm} \leftarrow \Sigma_R.\texttt{Setup}(1^\lambda)$	1 : $(\texttt{ppm}, z) \leftarrow \Sigma_R.\texttt{SimSetup}(1^\lambda)$
2 : $(\texttt{Prove}, x, w), \texttt{st} \leftarrow \mathscr{A}(1^\lambda, \texttt{ppm})$	2 : $(\texttt{Prove}, x, w), \texttt{st} \leftarrow \mathscr{A}(1^\lambda, \texttt{ppm})$
3 : **if** $R(x, w) = 1$:	3 : **if** $(x, w) \in R$:
4 : $\quad \pi \leftarrow \Sigma_R.\texttt{Prove}((x, w), (x, \texttt{chl}))$	4 : $\quad \pi \leftarrow \Sigma_R.\texttt{SimProve}(x, z, \texttt{chl})$
5 : **else** :	5 : **else** :
6 : $\quad \pi \leftarrow \perp$	6 : $\quad \pi \leftarrow \perp$
7 : $b' \leftarrow \mathscr{A}(\texttt{st}, \pi)$	7 : $b' \leftarrow \mathscr{A}(\texttt{st}, \pi)$
8 : **return** b'	8 : **return** b'

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

Note that in the standard definition of SHVZK, $\mathscr{A}$ is only permitted to issue *one* `Prove` query. In the GUC security experiment (and in most natural applications of Σ-protocols), the adversary is allowed to issue polynomially-many `Prove` queries and we will still need the SHVZK property to hold. Therefore, the definition we care about is *multiple* SHVZK (multi-SHVZK), also called multi-proof or multi-theorem SHVZK. We will formalize a version of multi-SHVZK for non-interactive straight-line extractable Σ-protocols in the next section, and prove later in Section 3.5 that any protocol that GUC-realizes the NIZKPoK ideal functionality in any global ROM must have this property. To justify our switch to the multi-proof setting for the basic (interactive) Σ-protocols and show that it does not compromise the generality of our results, we define multi-SHVZK

for interactive Σ-protocols in Appendix B.1, and prove using a standard hybrid argument that if Σ_R is SHVZK, it is also multi-SHVZK.

Finally, the special soundness property essentially says that for any pair of valid proofs generated by an adversary $\mathscr{A}$ for a statement x that have the same commitment but different challenges, the $\texttt{Extract}$ algorithm can extract a witness such that $R(x, w) = 1$ with overwhelming probability. The word "special" here refers to the fact that the $\texttt{Extract}$ algorithm relies on access to multiple valid transcripts in order to obtain a witness; by default, "special" soundness actually refers to "two-special" soundness (such that $\texttt{Extract}$ needs two transcripts), but Σ-protocols can also be n-special sound for some integer $n > 2$. To maintain consistency with the original definitions and keep things simple in the proofs, we have left our definition as two-special, but it is easy to replace any mention of two transcripts π, π' throughout the paper with $\pi_1, \ldots, \pi_n$.

We again formalize the intuition of special soundness with a game in which $\mathscr{A}$ issues a challenge tuple (x, π, π') that is designed to force the $\texttt{Extract}$ algorithm to fail—that is, the witness w returned by $\texttt{Extract}(x, \pi, \pi')$ is such that $R(x, w) = 0$.

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

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

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

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

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

Fig. 5. Special Soundness (SS) Game.

Similar to our need for the multi-SHVZK property, we will later show in Section 3.5 that GUC security requires a stronger form of soundness called special

simulation soundness (SSS), which guarantees that the `Extract` algorithm will still succeed with overwhelming probability even if $\mathscr{A}$ has oracle access to the proof simulator algorithm `SimProve` instead of `Prove`. Unlike multi-SHVZK, SSS does not follow from regular special soundness (SS). However, a regular SS Σ-protocol can be bootstrapped into an SSS one via a non-interactive straight-line compiler discussed in the next section, and Fischlin shows that for non-interactive straight-line extractable (NISLE) Σ-protocols, multi-SHVZK implies special simulation soundness. We formalize the notion of non-interactive SSS for the compiler in the next section, and prove in Section 3.5 that any protocol that GUC-realizes the NIZKPoK ideal functionality in any global ROM must have the non-interactive SSS property.

A.5 Non-Interactive Special Soundness

In the traditional definition of the special soundness experiment, $\mathscr{A}$ computes proofs during this experiment by himself, without access to proofs computed by non-adversarial participants. We make a syntactic modification to the traditional definition by allowing $\mathscr{A}$ to outsource the computation of proofs to the `Prove` oracle that just follows the correct algorithm for generating proofs. This syntactic modification does not make the definition any different (since the adversary can easily run `Prove` himself), but it will be helpful to us in the argument of Theorem 1.

Definition 21 (Non-Interactive Special Soundness). *A NISLE Σ-protocol Σ_R^{SLC} based on any Σ-protocol Σ_R for relation R is* non-interactive special sound (NI-SS) *if there exists an algorithm $\Sigma_R^{\mathsf{SLC}}.\mathtt{Extract}$ such that for any security parameter λ and any* `PPT` *adversary $\mathscr{A}$,*

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

where H is any random oracle and NI–SS *is the NI-SS game described in Figure 6. We say $\mathscr{A}$ wins the* NI–SS *game if* $\Pr[\mathtt{Fail} \leftarrow \mathsf{NI\text{–}SS}^H_{\mathscr{A},\Sigma_R^{\mathsf{SLC}}}(1^\lambda)] > \mathsf{negl}(\lambda)$.

A.6 Additional Properties of NISLC-Compliant Σ-protocols

By introducing a random oracle into the security experiment, NI transforms of any kind open up a new sort of security vulnerability for Σ-protocols rooted in the adversary's ability to freely interact with the RO. In particular, if an adversary $\mathscr{A}$ can predict how the prover is going to query the oracle in order to generate a proof of a statement x, $\mathscr{A}$ can go through this process itself and "predict" the challenge that will be returned. In other words, if $\mathscr{A}$ is able to predict `com` and query the RO on $(x, \mathtt{com})$ before the prover does, it will be able to learn the RO's original response $\mathtt{chl}^*$ before the simulator has had a chance to program a different one. $\mathscr{A}$ will then be able to distinguish the NIM-SHVZK games based on whether or not the `chl` returned by the SHVZK challenger matches $\mathtt{chl}^*$. To handle this vulnerability, we follow Fischlin [31] in assuming that the

$\mathsf{NI\text{–}SS}^{H}_{\mathscr{A},\Sigma_R^{\mathtt{SLC}}}(1^\lambda)$

1 : $\mathtt{ppm} \leftarrow \Sigma_R^{\mathtt{SLC}}.\mathtt{Setup}(1^\lambda)$
2 : $\mathtt{st} \leftarrow \mathscr{A}^H(1^\lambda, \mathtt{ppm})$
3 : **while** $\mathtt{st} \neq \mathtt{halt}$:
4 : $\quad \mathtt{Query}, \mathcal{Q}^{\mathscr{A}}, \mathtt{st} \leftarrow \mathscr{A}^H(\mathtt{st})$
5 : $\quad \mathtt{Response} \leftarrow \bot$
6 : $\quad$ **if** $\mathtt{Query} = (\mathtt{Prove}, x, w)$:
7 : $\quad\quad$ **if** $R(x, w) = 1$:
8 : $\quad\quad\quad \pi \leftarrow \Sigma_R^{\mathtt{SLC}}.\mathtt{Prove}^H(x, \mathtt{chl})$
9 : $\quad\quad\quad \mathtt{Response} \leftarrow \pi$
10 : $\quad$ **elseif** $\mathtt{Query} = (\mathtt{Challenge}, x, \pi)$
11 : $\quad\quad$ **if** $\Sigma_R^{\mathtt{SLC}}.\mathtt{Verify}^H(x, \pi) = 1$:
12 : $\quad\quad\quad w \leftarrow \Sigma_R^{\mathtt{SLC}}.\mathtt{Extract}(x, \pi, \mathcal{Q}^{\mathscr{A}})$
13 : $\quad\quad\quad$ **if** $R(x, w) = 0$:
14 : $\quad\quad\quad\quad$ **return** $\mathtt{Fail}$
15 : $\quad \mathtt{st} \leftarrow \mathscr{A}^H(\mathtt{st}, \mathtt{Response})$
16 : **return** $\mathtt{Success}$

Fig. 6. Non-Interactive Special Soundness (NI-SS) Game.

com messages of the underlying Σ-protocols have entropy that is superlogarithmic in the security parameter. We stress that any Σ-protocol that maintains the SHVZK property under any NI transform in the ROM, including the plain Fiat-Shamir transform, must have this property.

Definition 22 (Superlogarithmic Commitment Entropy). *Let Σ_R be any Σ-protocol for binary $\mathcal{NP}$ relation R and template τ as specified in Definition 2. Σ_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 λ.*

Recall from Section 1 that Fischlin additionally requires a "quasi-unique responses" property that would preclude us from using his transform on an OR-protocol. In order to solve this problem we exchange the Fischlin transform for the *randomized* Fischlin transform due to Kondi and shelat [36], which relies on a weaker assumption called strong special soundness. We define and discuss this property as well as the randomized Fischlin transform in Section 6.1.

A.7 The OR-protocol

The following definition is the original OR-protocol as imagined by Cramer [24] and formalized by Damg(å)rd [26]. Given both statements x_0, x_1 and a witness w_b for one of the statements x_b, the OR-protocol prover first samples a random challenge $\mathtt{chl}_{1-b}$ to correspond to the statement for which it does not have a witness, x_{1-b}. It then invokes the Simulate algorithm on input $(x_{1-b}, \mathtt{chl}_{1-b})$ to obtain the entire simulated proof transcript $(\mathtt{com}_{1-b}, \mathtt{chl}_{1-b}, \mathtt{res}_{1-b})$. The prover then forms the first message commitment $\mathtt{com}_b$ for x_b honestly according to the Commit algorithm, and sends the tuple $(\mathtt{com}_0, \mathtt{com}_1)$ to the verifier, who returns the overall protocol challenge CHL.

Once it receives CHL from the verifier, the prover sets the second individual Σ-protocol challenge $\mathtt{chl}_b = \mathtt{chl}_{1-b} \oplus \mathtt{CHL}$. Note this step "fixes" the challenge $\mathtt{chl}_b$ such that the prover cannot cheat and simulate the proof of both statements. Given $\mathtt{chl}_b$, the prover can compute $\mathtt{res}_b$ according to the Respond algorithm. Finally, the prover sends both transcripts $(\mathtt{com}_0, \mathtt{chl}_0, \mathtt{res}_0)$ and $(\mathtt{com}_1, \mathtt{chl}_1, \mathtt{res}_1)$ to the verifier, who checks that both are transcripts are valid and also that $\mathtt{chl}_0 \oplus \mathtt{chl}_1 = \mathtt{CHL}$.

Definition 23 (Original OR-Protocol). [26] *Let the common input to P and V be a pair (x_0, x_1), and the private input to P be a value w and a bit b such that $(x_b, w) \in R$. The* OR-protocol *is the following transaction:*

1. P computes the first message a_b according to the template using (x_b, w) as input. P chooses e_{1-b} at random and runs the simulator M on input (x_{1-b}, e_{1-b}); let $(a_{1-b}, e_{1-b}, z_{1-b})$ be the output of M. P sends V (a_0, a_1).
2. V sends P a random ℓ-bit string s.
3. P sets $e_b = s \oplus e_{1-b}$ and computes the answer z_b to challenge e_b according to the template using (x_b, a_b, e_b, w) as input. P sends (e_0, z_0, e_1, z_1) to V.
4. V checks that $e_0 \oplus e_1 = s$ and that both transcripts (a_0, e_0, z_0) and (a_1, e_1, z_1) are accepting on inputs x_0 and x_1, respectively.

Note that the original formulation does not explicitly state which template or Σ-protocol specification the prover uses at each step of the protocol execution. Since the proofs of statements x_0 and x_1 are computed independently, it is reasonable to consider the case in which x_0 and x_1 are associated with different relations, protocol templates, and Σ-protocols. In the spirit of keeping the CRS generation mechanism that we will introduce for our OR-protocol construction in Section 5.1 as general as possible, we consider the more general case where R_0 and R_1 are independent. Our version of the OR-protocol therefore depends on two different Σ-protocols Σ_{R_0} and Σ_{R_1}, allowing the prover to differentiate its instructions depending on the witness it has. For example, if the prover has w_b for a statement x_b, it would use the $\texttt{SimProve}$ algorithm of $\Sigma_{R_{1-b}}$ to obtain the transcript for x_{1-b}, then use the algorithms in the protocol template τ_{R_b} to generate the transcript for x_b.

In order to keep the notation consistent with Σ-protocols while avoiding variable clutter, we use capital letters to represent compound objects as follows. The statement X to be proven in an OR-protocol consists of a tuple representing *both* statements x_0 and x_1, or $X = (x_0, x_1)$. The compound proof Π is a tuple including $\pi_0 = (\texttt{com}_0, \texttt{chl}_0, \texttt{res}_0)$ and $\pi_1 = (\texttt{com}_1, \texttt{chl}_1, \texttt{res}_1)$, as well as the verifier's challenge, $\texttt{CHL}$. We write this tuple $\Pi = (\pi_0, \pi_1, \texttt{CHL})$. Similarly, the witness W must include not only the witness w for one of the statements, but also a bit b indicating the statement to which w corresponds. In other words, if $(x_0, w) \in R_0$ then $b = 0$ and the witness tuple is $W = (w, 0)$. Otherwise if $(x_1, w) \in R_1$, then the tuple is $W = (w, 1)$. In the special case that W is returned from the extractor, we let $W = (w_0, w_1)$, with the acknowledgement that only one of the witnesses produced by the $\texttt{Extract}$ operation must be legitimate—either $R_0(x_0, w_0) = 1$ or $R_1(x_1, w_1) = 1$.

Definition 24 (OR-Protocol). *An* OR-protocol *for a relation $R_{OR} = R_0 \vee R_1$ based on Σ-protocols Σ_{R_0,τ_0} and Σ_{R_1,τ_1} (see Definition 2) is a tuple of procedures $\Sigma_{OR} = (\texttt{Setup}, \texttt{Prove}, \texttt{Verify}, \texttt{SimSetup}, \texttt{Simulate}, \texttt{Extract})$ defined as follows.*

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

  The output "proof" Π is a tuple $(\pi_0, \pi_1, \texttt{CHL})$, where $\pi_b = (\texttt{com}_b, \texttt{chl}_b, \texttt{res}_b)$.

– $\{0,1\} \leftarrow \texttt{Verify}(X, \Pi)$: *Parse* Π *as* $(\pi_0, \pi_1, \texttt{CHL})$, *where* $\pi_b = (\texttt{com}_b, \texttt{chl}_b, \texttt{res}_b)$. *Execute the following:*

  - $\{0,1\} \leftarrow \texttt{Decision}(X, \texttt{Com}, \texttt{Chl}, \texttt{Res})$: *If* $\tau_0.\texttt{Decision}(x_0, \texttt{com}_0, \texttt{chl}_0, \texttt{res}_0) = 1$ *and* $\tau_1.\texttt{Decision}(x_1, \texttt{com}_1, \texttt{chl}_1, \texttt{res}_1) = 1$, *return 1. Otherwise, return 0.*

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

– $(\texttt{PPM}, Z) \leftarrow \texttt{SimSetup}(1^\lambda)$: *Generate* $(\texttt{ppm}_0, z_0)$ *by running* $\Sigma_{R_0}.\texttt{SimSetup}(1^\lambda)$ *and* $(\texttt{ppm}_1, z_1)$ *by running* $\Sigma_{R_1}.\texttt{SimSetup}(1^\lambda)$. *Return* $(\texttt{PPM}, Z)$ *where* $Z = (z_0, z_1)$.

– $\Pi \leftarrow \texttt{SimProve}(X, Z, \texttt{CHL})$: *Parse* $X = (x_0, x_1)$ *and* $Z = (z_0, z_1)$. *Generate* $\texttt{chl}_0$ *uniformly at random and set* $\texttt{chl}_1 = \texttt{chl}_0 \oplus \texttt{CHL}$. *Obtain* π_0 *by running* $\Sigma_{R_0}.\texttt{Simulate}(x_0, \texttt{chl}_0)$ *and* π_1 *by running* $\Sigma_{R_1}.\texttt{Simulate}(x_1, \texttt{chl}_1)$. *Return* $\Pi = (\pi_0, \pi_1, \texttt{CHL})$.

– $W \leftarrow \texttt{Extract}(X, \Pi, \Pi')$: *Parse* $X = (x_0, x_1)$, $\Pi = (\pi_0, \pi_1)$, *and* $\Pi' = (\pi'_0, \pi'_1)$. *Obtain* w_0 *by running* $\Sigma_{R_0}.\texttt{Extract}(x_0, \pi_0, \pi'_0)$ *and* w_1 *by running* $\Sigma_{R_1}.\texttt{Extract}(x_1, \pi_1, \pi'_1)$. *Return* $W = (w_0, w_1)$.

Theorem 5. *Given* Σ*-protocols* Σ_{R_0} *for a relation* R_0 *and* Σ_{R_1} *for relation* R_1, *the protocol* Σ_{OR} *from Definition 24 is a* Σ*-protocol for relation* $R_{OR} = R_0 \vee R_1$. *Moreover, for any verifier* V^*, *the probability distribution of conversations between* P *and* V^* *where* w *is such that* $(x_b, w) \in R_b$ *is independent of* b.

Proof. We refer the reader to Damgård's proof [26]. □

A.8 The Original Fischlin Transform

Definition 25 (Original Fischlin Transform). [31] *Let* (P_{FS}, V_{FS}) *be an interactive Fiat-Shamir (FS) proof of knowledge over relation* R *with challenge length* $\ell = O(\log \lambda)$ *bits. Let* b *be the number of test bits,* r *be the number of repetitions,* S *be the maximum sum over all repetitions, and* t *be the number of bits per trial such that* $br = \omega(\log \lambda)$, $2^{t-b} = \omega(\log \lambda)$, $b, r, t = O(\log \lambda)$ *and* $b \leq t \leq \ell$. *Let* $H : \{0,1\}^* \to \{0,1\}^b$ *be a random oracle that maps to* b *bits. Define the following NI proof system for relation* R *in the ROM as follows.*

Prover. *The prover* P^H *runs the prover of the underlying FS proof system* $P_{FS}(x, w)$ *in* r *independent repetitions to obtain the commitment vector* $\overline{a} = a_1, \ldots, a_r$. *Then for each repetition* $1 \leq i \leq r$, P^H *tests* t*-bit challenges* $e_i = 0, 1, \ldots 2^t - 1$ *and computes the response* z_i *using* P_{FS} *until it finds one such that* $H(x, \overline{a}, i, e_i, z_i) = 0^b$. *If no such tuple is found, the prover picks the minimal value over all* 2^t *oracle queries. The prover outputs the proof* (x, π) *where* $\pi = (a_i, e_i, z_i)$ *for* $1 \leq i \leq r$.

Verifier. *The verifier* V^H *accepts (outputs 1) if and only if* $V_{1,FS}(x, \pi_i) = 1$ *for* $1 \leq i \leq r$ *where* $\pi_i = (a_i, e_i, z_i)$, *and if* $\sum_{i=1}^{r} H(x, \overline{a}, i, e_i, z_i) \leq S$. *Otherwise, the verifier rejects (outputs 0).*

A.9 The GUC Real- and Ideal-World Experiments

Real-World Experiment. The real-world experiment $\mathsf{REAL}^{\mathcal{G}_{\mathtt{RO}},\mathcal{F}_{\mathtt{CRS}}}_{\Sigma_R^{\mathsf{guc}},\mathscr{A},\mathcal{Z}}(1^\lambda,\mathtt{aux})$ is executed as follows.

1. The experiment invokes the environment $\mathcal{Z}$ on input $(1^\lambda,\mathtt{aux})$.
2. $\mathcal{Z}$ invokes $\mathscr{A}$ on input of its choice and $\mathcal{G}_{\mathtt{RO}}$ on input 1^ℓ.[3]
3. $\mathcal{Z}$ invokes arbitrary parties with arbitrary SIDs. $\mathcal{Z}$ can corrupt up to all but one of the parties by sending messages through $\mathscr{A}$. $\mathcal{Z}$ can invoke new parties whenever it chooses,[4] but must decide at the time of invocation whether or not they are corrupted (passive corruption model).
4. As is standard in the UC and GUC models, $\mathcal{Z}$ passes inputs and receives outputs to the input-output tapes of all parties to the protocol on its own. Additionally, it communicates with corrupted parties through $\mathscr{A}$. In particular (briefly), $\mathcal{Z}$ can send arbitrary `Setup`, `Prove`, and `Verify` requests to any party, and have corrupted parties send any corrupted `Setup`, `Prove`, and `Verify` requests on its behalf. It can also arbitrarily query $\mathcal{G}_{\mathtt{RO}}$ using any SID, and execute any version of `Setup`, `Prove`, and `Verify` itself.
5. In order to respond honestly to `Setup`, `Prove`, and `Verify` requests, the parties run the protocols $\Sigma_R^{\mathsf{guc}}.\mathtt{Setup}$, $\Sigma_R^{\mathsf{guc}}.\mathtt{Prove}$, and $\Sigma_R^{\mathsf{guc}}.\mathtt{Verify}$, respectively.

Ideal-World Experiment. The ideal world experiment $\mathsf{IDEAL}^{\mathcal{G}_{\mathtt{RO}}}_{\Sigma_R^{\mathsf{guc}},\mathcal{S},\mathcal{Z}}(1^\lambda,\mathtt{aux})$ is executed as follows.

1. The experiment invokes the environment $\mathcal{Z}$ on input $(1^\lambda,\mathtt{aux})$.
2. $\mathcal{Z}$ invokes $\mathcal{S}$ on input of its choice[5] and $\mathcal{G}_{\mathtt{RO}}$ on input 1^ℓ.
3. Same as Step 3 in the real world experiment.
4. Same as Step 4 in the real world experiment.
5. Rather than respond to `Setup`, `Prove`, and `Verify` requests themselves, honest parties invoke the (local) ideal functionality $\mathcal{F}_{\mathtt{NIZK}}$ for their SID s. At initialization, $\mathcal{F}_{\mathtt{NIZK}}$ obtains specifications for the algorithms `Setup`, `Prove`, `Verify`, `Simulate`, and `Extract` from $\mathcal{S}$. After the ideal functionality is set up, honest parties with SID s forward all `Prove` and `Verify` requests directly to $\mathcal{F}_{\mathtt{NIZK}}$, which responds according to its specification, given in Definition 9.

[3] One can also imagine that $\mathcal{G}_{\mathtt{RO}}$ with output length ℓ already exists, or was invoked by the experiment. Since a precise invocation of $\mathcal{G}_{\mathtt{RO}}$ is not clear in the literature, we chose to maintain internal consistency with the rest of the definition and have the experiment initialize $\mathcal{G}_{\mathtt{RO}}$ during setup.

[4] In order to guarantee that the experiment runs in time polynomial in the security parameter, the UC model places certain restrictions on the runtime of the arbitrary parties $\mathcal{Z}$ invokes. For a full discussion, we refer readers to Canetti et al. [19].

[5] To the environment, this process looks exactly the same as in the real world. However in the ideal world, the simulator comes pre-programmed with special instructions to help the ideal functionality simulate the protocol.

A.10 Global Random Oracle

Definition 26 (Global Random Oracle). [22] *The* global random oracle (global RO) *is a public random oracle functionality parameterized by the output length $\ell(\lambda)$ and a list $\overline{\mathcal{F}}$ of ideal functionalities. The oracle works as follows.*

1. *Upon receiving a query x from some party $P = (\texttt{pid}, \texttt{sid})$ or from the simulator $\mathcal{S}$, do:*
   - *If there already exists a pair (x, v) for some $v \in \{0,1\}^{\ell(\lambda)}$ in the (initially empty) list $\mathcal{Q}$ of past queries, return v to P. Otherwise, chose v uniformly from $\{0,1\}^{\ell(\lambda)}$, store the pair (x, v) in $\mathcal{Q}$, and return v to P.*
   - *Parse x as (s, x'). If $\texttt{sid} \neq s$ then add (s, x', v) to the (initially empty) list of* illegitimate queries *for SID s, denoted $\mathcal{Q}_s$.*
2. *Upon receiving a request from an instance of an ideal functionality in the list $\overline{\mathcal{F}}$ with SID s, return to this instance the list $\mathcal{Q}_s$ of illegitimate queries for SID s.*

A.11 Discussion of Strong Special Soundness

Recall Definition 15: A Σ-protocol Σ_R has *strong special soundness* if the condition $\texttt{chl} \neq \texttt{chl}'$ in the `Extract` algorithm from Definition 20 is replaced with the condition $(\texttt{chl}, \texttt{res}) \neq (\texttt{chl}', \texttt{res}')$.

The strengthening of the extractor afforded by strong special soundness allows the randomized Fischlin prover to iterate over the same challenge without compromising soundness. If, for example, the Σ-protocol allowed both $(\texttt{com}, \texttt{chl}, \texttt{res}||0)$ and $(\texttt{com}, \texttt{chl}, \texttt{res}||1)$ to verify without extraction, repeating the protocol for the same challenge (as is the case with the Fischlin prover) would not guarantee soundness, since a cheating prover could simply simulate one instance and tack on some extra bits at the end.

Fischlin navigated around this issue using the quasi-unique responses property, which requires that if $\Sigma_R.\texttt{Verify}(x, \texttt{com}, \texttt{chl}, \texttt{res}) = \Sigma_R.\texttt{Verify}(x, \texttt{com}, \texttt{chl}, \texttt{res}') = 1$, then $\texttt{res} = \texttt{res}'$ with overwhelming probability. As noted by Kondi and shelat, this also prevents two provers with different witnesses, w_0 and w_1 for the same statement x, from answering the same challenge in a different way. This situation always occurs when the simulator is using a different witness than a real prover (as is the case with our OR-protocol transform from Definition 14), and also occurs during the normal functioning of most OR-protocols. For a more in depth discussion on the Fischlin transform applied to OR-protocols and the strong special soundness property, we refer the reader to Section 6 of Kondi and shelat [36].

A.12 Randomized Fischlin Transform

Recall from Section 1 that the (randomized) Fischlin transform works by checking for each completed transcript whether the output of the RO for that transcript maps to a b-bit string. Therefore before we can apply the transform, we

need a random oracle that maps to b bits. For the purposes of this general defintion, we let the global RO be the general global RO $\mathcal{G}_{\mathtt{RO}}$ described in Section 3.5. Since $\mathcal{G}_{\mathtt{RO}}$ is global and can be reused for different setups, rather than alter the output length or introduce a second RO, we construct the truncation function suggested by Fischlin [31] that maps the output of $\mathcal{G}_{\mathtt{RO}}$ to b bits by cutting off all but b bits of the output.

Definition 27 (Bit Truncation Function). *The RO* bit truncation function $\mathtt{trunc} : \{0,1\}^{\ell} \to \{0,1\}^{b}$ *maps the ℓ-bit output of H to a b-bit output by cutting off the $\ell - b$ leading bits.*

The RO functionality of the (randomized) Fischlin transform where the RO $H : \{0,1\}^{*} \to \{0,1\}^{b}$ is therefore replaced by $\mathtt{trunc}(H) : \{0,1\}^{*} \to \{0,1\}^{b}$. We omit overly-technical details such as the precise generation of the public parameters, which can be found in brief in Appendix A.8. For a more in depth treatment of the Fischlin and randomized Fischlin transforms, we invite readers to peruse [31] and [36], respectively.

Definition 28 (Randomized Fischlin Transform). *Let $\Sigma_{R,\tau}$ be any special Σ-protocol for relation R and protocol template τ as given in Definition 2 with the strong special soundness property from Definition 15 and a challenge length $\ell = O(\log \lambda)$ bits. Let H be any random oracle. Then the* randomized Fischlin transform *of $\Sigma_{R,\tau}$, denoted* $\mathtt{Fis}$*, is an algorithm that takes $\Sigma_{R,\tau}$ as input and creates a tuple of algorithms $\Sigma_R^{\mathtt{Fis}} = (\mathtt{Setup}^H, \mathtt{Prove}^H, \mathtt{Verify}^H, \mathtt{SimSetup}, \mathtt{SimProve}, \mathtt{Extract})$, defined as follows.*

- $\mathtt{ppm} \leftarrow \mathtt{Setup}^H(1^\lambda)$: *H is fixed. Let b, r, S, t be set according to the Fischlin transform (see Appendix A.8 for details). Then the public parameters are $\mathtt{ppm} = (\mathtt{ppm}_\Sigma, b, r, S, t, \mathtt{trunc})$, where $\mathtt{ppm}_\Sigma$ is obtained by running $\tau.\mathtt{Setup}(1^\lambda)$ and $\mathtt{trunc}$ is the bit truncation function from Definition 27.*
- $(x, \Pi) \leftarrow \mathtt{Prove}^H(x, w)$: *Compute the vector of r commitments $\overline{\mathtt{com}} = \langle \mathtt{com}_0, \mathtt{com}_1, \ldots, \mathtt{com}_r \rangle$, by running $\tau.\mathtt{Commit}(x, w)$ r times. To compute each response $\mathtt{res}_i$, the prover tests each t-bit challenge $\mathtt{chl}_i$ as follows. First, it selects $\mathtt{chl}_i$ uniformly at random from the challenge space. Then, it repeats $\tau.\mathtt{Respond}(x, w, \mathtt{com}, \mathtt{chl})$ until it finds one such that $\mathtt{trunc}(\mathcal{G}_{\mathtt{RO}}(x, \overline{\mathtt{com}}, i, \mathtt{chl}_i, \mathtt{res}_i)) = 0^b$, or else it takes the minimal over all of the responses. Finally, it returns (x, Π), where $\Pi = (\pi_1, \ldots, \pi_r)$, and each $\pi_i = (\mathtt{com}_i, \mathtt{chl}_i, \mathtt{res}_i)$.*
- $\{0,1\} \leftarrow \mathtt{Verify}^H(x, \Pi)$: *Parse $\Pi = (\pi_1, \ldots, \pi_r)$. Output 1 (accept) if and only if $\Sigma_R.\mathtt{Verify}(x, \pi_i) = 1$ and $\sum_{i=1}^{r} \mathtt{trunc}(\mathcal{G}_{\mathtt{RO}}(x, \overline{\mathtt{com}}, i, \mathtt{chl}_i, \mathtt{res}_i)) \leq S$ for $1 \leq i \leq r$. Otherwise, outputs 0 (reject).*
- $(\mathtt{ppm}, z) \leftarrow \mathtt{SimSetup}(1^\lambda)$: *Fix H and generate $\mathtt{ppm}$ the same as in $\Sigma_R^{\mathtt{Fis}}.\mathtt{Setup}$. Generate the simulator state information z by running $\Sigma_{R,\tau}.\mathtt{SimSetup}$ and return $(\mathtt{ppm}, z)$.*
- $(x, \Pi) \leftarrow \mathtt{SimProve}(x, z, \mathtt{chl}_1, \ldots, \mathtt{chl}_r)$: *For each proof $1 \leq i \leq r$, sample 2^t random b-bit strings and assign them to the t-bit challenges $\mathtt{chl}_i$. Let $\mu : \{0,1\}^t \to \{0,1\}^b$ represent the map between the challenges and the b-bit*

outputs, which are potential outputs of H. Let the final challenge for the i^{th} proof $\mathtt{chl}_i$ be the first challenge in lexicographic order to map to the minimal response. Run $\Sigma_{R,\tau}.\mathtt{Simulate}(x, z, \mathtt{chl})$ to obtain $\pi_i = (\mathtt{com}_i, \mathtt{chl}_i, \mathtt{res}_i)$. Repeat this process for all r proofs. For each proof, program the output of H on input $(x, \overline{\mathtt{com}}, i, \mathtt{chl}_i, \mathtt{res}_i)$ to end with the b-bit output $\mu_i(\mathtt{chl}_i)$, and let the $\ell - b$ leading bits be random. Finally, output the proof tuple (x, Π), where $\Pi = (\Pi_1, \ldots, \Pi_r)$.

– $w \leftarrow \mathtt{Extract}(X, \Pi, \mathcal{Q}_{\mathscr{A}})$: *Parse $\Pi = (\pi_1, \ldots, \pi_r)$ 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 $\Sigma_R.\mathtt{Verify}(x, (\mathtt{com}_i, \mathtt{chl}_i, \mathtt{res}_i)) = \Sigma_R.\mathtt{Verify}(x, (\mathtt{com}_i, \mathtt{chl}'_i, \mathtt{res}'_i)) = 1$. If no such queries exist, output $\mathtt{Fail}$. Otherwise, obtain w by running $\Sigma_R.\mathtt{Extract}(x, (\mathtt{com}_i, \mathtt{chl}_i, \mathtt{res}_i), (\mathtt{com}_i, \mathtt{chl}'_i, \mathtt{res}'_i))$.*

The full proof that the randomized Fischlin transform described above is a straight-line compiler can be found in Appendix B.5.

A.13 Compiling Σ-protocols into GUC-secure NIZKPoK

We introduce the idea of a GUC compiler for Σ-protocols in the global ROM. Similar to the specification of a straight-line compiler in Section 3, our GUC compiler takes any Σ-protocol as input, and produces a tuple of algorithms $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}$ based on Σ_R that have access to a generic global random oracle $\mathcal{G}_{\mathtt{RO}}$ from Section 3.5. In Section 4 we will show that an SLC is a sufficient GUC compiler for Σ-protocols in the programmable $\mathcal{G}_{\mathtt{rpoRO}}$-hybrid model, and in Section 5 we will demonstrate how to use an SLC in conjunction with an OR-protocol in order to construct a new GUC compiler for Σ-protocols in the non-programmable $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model.

Definition 29 (GUC Compiler). *An algorithm $\mathtt{GUC}_{\Sigma}$ is a GUC-compiler for Σ-protocols in the global random-oracle model if given any Σ-protocol Σ_R from Definition 2 as input, it outputs a tuple of algorithms $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}} = (\mathtt{Setup}^{\mathcal{G}_{\mathtt{RO}}}, \mathtt{Prove}^{\mathcal{G}_{\mathtt{RO}}}, \mathtt{Verify}^{\mathcal{G}_{\mathtt{RO}}}, \mathtt{SimSetup}, \mathtt{SimProve}, \mathtt{Extract})$ based on Σ_R with oracle access to the global random oracle $\mathcal{G}_{\mathtt{RO}}$ such that $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}$ GUC-realizes the NIZKPoK ideal functionality $\mathcal{F}_{\mathtt{NIZK}}$ in the $\mathcal{G}_{\mathtt{RO}}$-hybrid model.*

B Supplementary Proofs

B.1 SSHVZK Implies Multi-SSHVZK

Definition 30 (Multiple SSHVZK). *A Σ-protocol Σ_R for relation R is* multiple special honest-verifier zero-knowledge (multi-SSHVZK) *if there exist algorithms $\Sigma_R.\mathtt{SimSetup}$ and $\Sigma_R.\mathtt{SimProve}$ such that for any security parameter λ, any $\mathtt{PPT}$ adversary $\mathscr{A}$, and a bit $b \leftarrow_{\$} \{0,1\}$, there exists some negligible function negl such that $\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, where b' is the result of running the game $\text{M-SSHVZK}_{\mathscr{A},\Sigma_R}(1^{\lambda}, b)$ from Figure 1. We say $\mathscr{A}$ wins the M–SHVZK game if $\Pr[b' = b] > \frac{1}{2} + \mathsf{negl}(\lambda)$.*

$\mathsf{M\text{–}SHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, 0)$

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

$\mathsf{M\text{–}SHVZK}_{\mathscr{A},\Sigma_R}(1^\lambda, 1)$

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

Fig. 7. Multiple SSHVZK (Multi-SSHVZK) Game.

Lemma 1. *If a Σ-protocol Σ_R is SSHVZK according to Definition 19, then it is multi-SSHVZK according to Definition 30.*

Proof. We proceed by contrapositive and show that a protocol that is not multi-SSHVZK cannot be SSHVZK. In particular, consider an adversary $\mathscr{A}$ who can distinguish the following worlds: world 1) the first j proofs returned by the multi-SSHVZK challenger are real and the $j + 1^{st}$ onward are simulated, and world 2) the first $j+1$ proofs are real and the $j+2^{nd}$ onward are simulated. We construct a reduction that uses $\mathscr{A}$ as a black box to win the regular SSHVZK game from Figure 4 as follows. The reduction proceeds by answering the first j of $\mathscr{A}$'s queries $(\mathtt{Prove}, x, w)$ by running $\Sigma_R.\mathtt{Prove}(x, w)$. On the $j + 1^{st}$ query $(\mathtt{Prove}, x_j, w_j)$, $\mathscr{A}$ issues $(\mathtt{Prove}, x_j, w_j)$ to its SSHVZK challenger and receives π_j that is either a result of running $\Sigma_R.\mathtt{Prove}((x_j, w_j), (x_j, \mathtt{chl}_j))$ or a result of running $\Sigma_R.\mathtt{SimProve}(x_j, z, \mathtt{chl}_j)$. It returns π_j to $\mathscr{A}$, sets up the simulator state z by running $\Sigma_R.\mathtt{SimSetup}(1^\lambda)$, and proceeds to answer the rest of $\mathscr{A}$'s queries $(\mathtt{Prove}, x, w)$ by running $\Sigma_R.\mathtt{SimProve}(x, z, \mathtt{chl})$ (note that since challenges are guaranteed to be independently distributed in the honest-verifier model, the reduction can simulate the rest of the proofs for $\mathscr{A}$ without "cheating" on its own challenge instance $\mathtt{chl}_j$). The reduction continues until $\mathscr{A}$ returns b' at which point the reduction also outputs b'. Clearly if $\mathscr{A}$ has distinguished the proof in the $j+1^{st}$ slot as real or simulated, so has the reduction—the reduction wins the SSHVZK game with the same probability that $\mathscr{A}$ distinguishes the j- and $j + 1$-hybrids of the multi-SSHVZK game. Therefore, the probability that $\mathscr{A}$ can distinguish the j^{th} from the $j + 1^{st}$ hybrid must be negligible in λ. Since $\mathscr{A}$ is PPT and the reduction is tight, the overall probability that $\mathscr{A}$ can win the multi-SSHVZK game is similarly negligible.

B.2 Full Proof of Theorem 1

The following is the full proof of Theorem 1 from Section 3.5.

Recall Theorem 1: Let Π be a protocol that GUC-realizes $\mathcal{F}_{\texttt{NIZK}}$ according to Definition 11 above, where $\mathcal{G}_{\texttt{rpoRO}}$ is replaced with $\mathcal{G}_{\texttt{RO}}$. Then Π must be both non-interactive multi-SSHVZK according to Definition 5 and non-interactive special simulation sound according to Definition 6.

Proof. We proceed by contrapositive and demonstrate that any protocol Π that is not both NIM-SSHVZK and NI-SSS cannot possibly GUC-realize $\mathcal{F}_{\texttt{NIZK}}$. We begin by showing that if Π is not NIM-SSHVZK, it does not GUC-realize $\mathcal{F}_{\texttt{NIZK}}$ in any global ROM.

Lemma 2. *Any protocol Π that is not NI-multi-SSHVZK in the $\mathcal{G}_{\texttt{RO}}$-hybrid model according to Definition 5 does not GUC–realize $\mathcal{F}_{\texttt{NIZK}}$ in the $\mathcal{G}_{\texttt{RO}}$-hybrid model.*

Proof. We construct a reduction that uses an algorithm $\mathscr{A}^{\mathcal{G}_{\texttt{RO}}}$ that wins the game NIM–SHVZK from Figure 1 with non-negligible advantage as a black box to distinguish between the real- and ideal-world GUC experiments. The reduction gets $\texttt{ppm}$ from its GUC challenger $\mathcal{C}$, who either calculates $\texttt{ppm} \leftarrow \Pi.\texttt{Setup}^{\mathcal{G}_{\texttt{RO}}}(1^\lambda)$ if it is running the real-world experiment or $\texttt{ppm}, z \leftarrow \Pi.\texttt{SimSetup}(1^\lambda)$ if it is running the ideal-world experiment, and the reduction initializes $\mathscr{A}$ on $(1^\lambda, \texttt{ppm})$. The reduction passes all of $\mathscr{A}$'s random oracle queries to and from $\mathcal{G}_{\texttt{RO}}$ and all of $\mathscr{A}$'s queries $(\texttt{Prove}, x_i, w_i)$ to $\mathcal{C}$ under some protocol session s. In response to the query $(\texttt{Prove}, s, x_i, w_i)$, $\mathcal{C}$ returns π that is either the result of running $\Pi.\texttt{Prove}^{\mathcal{G}_{\texttt{RO}}}(x_i, w_i)$ or the result of running $\Pi.\texttt{SimProve}(x_i, z, \texttt{chl}_i)$ (where $\mathcal{G}_{\texttt{RO}}$ is potentially programmed). Clearly if the simulator hands $\mathcal{F}_{\texttt{NIZK}}$ algorithms $\texttt{SimSetup}$ and $\texttt{SimProve}$ that cause a completeness error (such that $R(x_i, w_i) = 1$ but $\texttt{Verify}(x_i, \pi_i) = 0$) and $\mathcal{F}_{\texttt{NIZK}}$ outputs $\texttt{Fail}$, the reduction can tell immediately that it is living in the ideal-world experiment without any further interaction with $\mathscr{A}$, and we arrive at the contradiction. Similarly if the reduction notices any inconsistencies in $\mathcal{G}_{\texttt{RO}}$ it can immediately output "ideal"—the reduction itself does not have any control over $\mathcal{G}_{\texttt{RO}}$ and therefore $\mathscr{A}$'s view of $\mathcal{G}_{\texttt{RO}}$ in the NI-Multi-SSHVZK experiment directly depends on whether the GUC experiment is real-world (such that $\mathcal{G}_{\texttt{RO}}$ does not change) or ideal-world (such that $\mathcal{G}_{\texttt{RO}}$ may change depending on the specification of $\texttt{SimSetup}$ and $\texttt{SimProve}$ and whether or not $\mathcal{G}_{\texttt{RO}}$ is programmable).

The reduction proceeds in the manner above until $\mathscr{A}$ outputs a bit b' to indicate whether it is talking to $\mathsf{M}\text{–}\mathsf{SHVZK}(1^\lambda, 0)$ or $\mathsf{M}\text{–}\mathsf{SHVZK}(1^\lambda, 1)$, and the reduction outputs whatever $\mathscr{A}$ outputs. Note that if $b = 0$ and the reduction is getting proofs from the standard $\Pi.\texttt{Prove}^{\mathcal{G}_{\texttt{RO}}}$ algorithm, the reduction produces $\mathscr{A}$'s exact view in the experiment $\mathsf{NIM}\text{–}\mathsf{SHVZK}^{\mathcal{G}_{\texttt{RO}}}(1^\lambda, 0)$ which also generates proofs using $\Pi.\texttt{Prove}^{\mathcal{G}_{\texttt{RO}}}$. If $b = 1$ and the reduction is getting proofs from an $\mathcal{F}_{\texttt{NIZK}}$ whose $\texttt{Prove}$ functionality never outputs $\texttt{Fail}$, this is exactly what $\mathscr{A}$ expects to see from the experiment $\mathsf{M}\text{–}\mathsf{SHVZK}(1^\lambda, 1)$, which generates proofs using $\Pi.\texttt{SimSetup}$ and $\Pi.\texttt{SimProve}$. Therefore, the reduction succeeds in distinguishing the real from ideal experiments with the same (non-negligible) probability as $\mathscr{A}$, completing the contradiction. □

We now show that if Π is not NI special simulation sound, it does not GUC-realize $\mathcal{F}_{\texttt{NIZK}}$ in any global ROM.

Lemma 3. *Any protocol Π that is not special simulation sound via query history in the $\mathcal{G}_{\texttt{RO}}$-hybrid model according to Definition 6 does not GUC–realize $\mathcal{F}_{\texttt{NIZK}}$ in the $\mathcal{G}_{\texttt{RO}}$-hybrid model according to Definition 11, where $\mathcal{G}_{\texttt{rpoRO}}$ is replaced with $\mathcal{G}_{\texttt{RO}}$.*

Proof. We again construct a reduction that uses an algorithm $\mathscr{A}^{\mathcal{G}_{\texttt{RO}}}$—this time one that wins the special simulation soundness via query history game NI–SSS from Figure 2—as a black box to distinguish between the real- and ideal-world GUC experiments. The reduction gets $\texttt{ppm}$ from its GUC challenger $\mathcal{C}$, where $\texttt{ppm} \leftarrow \Pi.\texttt{Setup}^{\mathcal{G}_{\texttt{RO}}}(1^\lambda)$ if $\mathcal{C}$ is running the real-world experiment or $\texttt{ppm}, z \leftarrow \Pi.\texttt{SimSetup}(1^\lambda)$ if $\mathcal{C}$ is running the ideal-world experiment, and the reduction initializes $\mathscr{A}$ on $(1^\lambda, \texttt{ppm})$. The reduction passes all of $\mathscr{A}$'s random oracle queries to and from $\mathcal{G}_{\texttt{RO}}$ and all of $\mathscr{A}$'s queries $(\texttt{Prove}, x_i, w_i)$ to $\mathcal{C}$ as $(\texttt{Prove}, s, x_i, w_i)$ under some challenge protocol session s. In response to the query $(\texttt{Prove}, s, x_i, w_i)$, $\mathcal{C}$ returns π that is either the result of running $\Pi.\texttt{Prove}^{\mathcal{G}_{\texttt{RO}}}(x_i, w_i)$ or the result of running $\Pi.\texttt{SimProve}(x_i, z, \texttt{chl}_i)$ (where $\mathcal{G}_{\texttt{RO}}$ is potentially programmed). Let the set of proofs returned by $\mathcal{C}$ up to query i be denoted $P = \pi_1, \ldots, \pi_i$. The argument surrounding the reduction's view of $\mathcal{G}_{\texttt{RO}}$ is the same as above—if the $\mathcal{C}$ is running the ideal-world experiment and $\mathcal{F}_{\texttt{NIZK}}$'s **Prove** interface makes any noticeable changes to $\mathcal{G}_{\texttt{RO}}$, the reduction will be able to tell immediately that it is living in the ideal world. Similarly if $\mathcal{F}_{\texttt{NIZK}}$'s **Prove** interface has a completeness error that causes $\mathcal{F}_{\texttt{NIZK}}$ to output $\texttt{Fail}$, the reduction outputs "ideal" without any further interaction with $\mathscr{A}$.

When $\mathscr{A}$ issues a query $(\texttt{Challenge}, x_i, \pi_i)$, $\mathscr{B}$ issues the query $(\texttt{Verify}, s, x_i, \pi_i)$ to $\mathcal{C}$. By assumption, $\mathscr{A}$ will eventually issue a challenge proof (x_i, π_i) such that $\Pi.\texttt{Verify}(x_i, \pi_i) = 1$ and $(x_i, \pi_i) \notin P$ but $R(x_i, w_i) = 0$ for $w \leftarrow \Pi.\texttt{Extract}(x_i, \pi_i, \mathcal{Q}_{\mathscr{A}})$, causing the NI–SSS experiment to output $\texttt{Fail}$. When the reduction outputs this proof to the $\mathcal{C}$, we argue that it will succeed in distinguishing the real from ideal worlds with the same probability as $\mathscr{A}$. Note that if the reduction is talking to the ideal-world GUC experiment then the challenger's responses to the queries $(\texttt{Prove}, s, x_i, w_i)$ and $(\texttt{Verify}, s, x_i, \pi_i)$ will be distributed identically to what $\mathscr{A}$ is expecting from the queries $(\texttt{Prove}, x_i, w_i)$ and $(\texttt{Challenge}, x_i, \pi_i)$ in the NI–SSS game for the following reasons. First, assuming the $\texttt{Prove}$ interface of $\mathcal{F}_{\texttt{NIZK}}$ does not output $\texttt{Fail}$ and $\mathscr{A}$'s view of $\mathcal{G}_{\texttt{RO}}$ remains consistent as discussed above, $\mathcal{F}_{\texttt{NIZK}}$'s $\texttt{SimSetup}$ and $\texttt{SimProve}$ algorithms must respond to queries $(\texttt{Prove}, s, x_i, w_i)$ with proofs π_i that are indistinguishable from the π_i produced by $\Pi.\texttt{SimSetup}(1^\lambda)$ and $\Pi.\texttt{SimProve}(x_i, z, \texttt{chl}_i)$ via the same argument as in Lemma 2 above. Second, $\mathcal{F}_{\texttt{NIZK}}$'s $\texttt{Extract}$ algorithm makes the same checks on $\texttt{Extract}$ as the challenger makes on $\Pi.\texttt{Extract}$ in the NI–SSS game. Therefore, if the reduction is talking to the ideal-world experiment, $\mathscr{A}$'s proof will cause $\mathcal{F}_{\texttt{NIZK}}$ to output $\texttt{Fail}$ with the same non-negligible advantage as $\mathscr{A}$ has in the NI–SSS game.

If the reduction is talking to the real-world experiment, we argue that the reduction succeeds with the same probability as an $\mathscr{A}$ playing the regular special

soundness with query history (SS-Q) game from Figure 6 in Appendix A.5. Note that if the reduction is talking to the real-world GUC experiment then the challenger's responses to the queries $(\texttt{Prove}, s, x_i, w_i)$ and $(\texttt{Verify}, s, x_i, \pi_i)$ will be distributed identically to what $\mathscr{A}$ is expecting from the queries $(\texttt{Prove}, x_i, w_i)$ and $(\texttt{Challenge}, x_i, \pi_i)$ in the NI–SS game for the following reasons. First, $\mathcal{G}_{\texttt{RO}}$ remains consistent throughout the protocol and $\mathcal{C}$ responds to queries $(\texttt{Prove}, s, x_i, w_i)$ with proofs $\pi_i \leftarrow \Pi.\texttt{Prove}(s, x_i, w_i)$, exactly as $\mathscr{A}$ expects from the NI–SS challenger. Whenever $\mathscr{A}$ issues a query $(\texttt{Challenge}, x_i, \pi_i)$ for a proof $(x_i, \pi_i) \notin P$ where $\Pi.\texttt{Verify}(x_i, \pi_i) = 1$ but $(x_i, \pi_i) \notin P$, the reduction runs $\Pi.\texttt{Extract}(x_i, \pi_i, \mathcal{Q}_{\mathscr{A}})$ itself (recall from the NI–SS and NI–SSS experiments that we assume $\mathscr{A}$ outputs its RO query history whenever it issues a challenge). If $\Pi.\texttt{Extract}(x_i, \pi_i, \mathcal{Q}_{\mathscr{A}})$ outputs `Fail`, then the reduction knows it has a proof that succeeds in breaking the regular special soundness property. When it queries this proof $(\texttt{Verify}, s, x_i, \pi_i)$ to $\mathcal{C}$ and gets a response $(\texttt{Verify}, s, x_i, \pi_i, 1)$ rather than a message `Fail`, it knows it is living in the real-world experiment, since $\mathcal{F}_{\texttt{NIZK}}$ would have made the same checks as the reduction. Therefore, the reduction succeeds in this case with the same probability as $\mathscr{A}$ can win the NI–SS game, completing the contradiction. □

To see why it was necessary for us to use the special *simulation* soundness property in the proof of Lemma 3, consider the case in which the reduction is talking to the ideal-world GUC challenger: the regular special soundness adversary is not defined to handle proofs from the simulator, so its behavior in this case is undefined and therefore useless to the reduction. To see why it was necessary for us to use the *non-interactive* versions of multi-SSHVZK and special simulation soundness definitions, note that the `Prove` and `Verify` interfaces of $\mathcal{F}_{\texttt{NIZK}}$ are non-interactive with respect to the oracle $\mathcal{G}_{\texttt{RO}}$—in order for the simulation and extraction algorithms of Π to correspond with the interfaces of $\mathcal{F}_{\texttt{NIZK}}$ they must be similarly non-interactive with respect to $\mathcal{G}_{\texttt{RO}}$.

We have now shown that both the NI multi-SSHVZK and NI special simulation soundness properties are necessary for a protocol Π to GUC-realize $\mathcal{F}_{\texttt{NIZK}}$ in the $\mathcal{G}_{\texttt{RO}}$-hybrid model, completing the proof of Theorem 1. □

B.3 Full Proof of Theorem 2

Recall Theorem 2: Let Σ_R be any Σ-protocol for relation R (Definition 2), $\mathcal{G}_{\texttt{rpoRO}}$ be the restricted programmable observable global random oracle (Definition 7), and `SLC` be any straight-line compiler (Definition 3). Then the NISLE Σ-protocol $\Sigma_R^{\texttt{SLC}} \leftarrow \texttt{SLC}(\Sigma_R)$ GUC-realizes $\mathcal{F}_{\texttt{NIZK}}$ in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model (Definition 11).

Proof. We must demonstrate that $\Sigma_{\texttt{R}\vee\texttt{S}}^{\texttt{SLC}} \leftarrow \texttt{SLC}(\Sigma_R)$ GUC-realizes $\mathcal{F}_{\texttt{NIZK}}$ in the $\mathcal{G}_{\texttt{rpoRO}}$-hybrid model—that is, we must satisfy Definition 11. Briefly, we must show that for all efficient $\mathscr{A}$, there exists an ideal adversary $\mathcal{S}$ efficient in expectation such that for all efficient environments $\mathcal{Z}$,

$$\mathsf{IDEAL}^{\mathcal{G}_{\texttt{RO}}}_{\mathcal{F}_{\texttt{NIZK}},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathsf{aux}) \approx_c \mathsf{REAL}^{\mathcal{G}_{\texttt{RO}},\mathcal{F}_{\texttt{CRS}}}_{\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathsf{aux}).$$

We review the precise formulation of the GUC experiment in Appendix 3.

Construction of the Simulator $\mathcal{S}$. The simulator (also known as the ideal adversary) $\mathcal{S}$, works as follows. When the ideal functionality $\mathcal{F}_{\texttt{NIZK}}$ asks it for the specification of algorithms, $\mathcal{S}$ returns the algorithms in $\Sigma_R^{\texttt{SLC}}$. When $\mathcal{F}_{\texttt{NIZK}}$ asks it for the queries of adversarial provers for an SID s, $\mathcal{S}$ returns the corrupted parties' $\mathcal{G}_{\texttt{rpoRO}}$ queries $\mathcal{Q}^s_{\mathscr{A}}$. If any of the corrupted parties issue an `IsProgrammed` query to $\mathcal{G}_{\texttt{rpoRO}}$ through $\mathcal{S}$ (recall that the environment cannot issue such queries to $\mathcal{G}_{\texttt{rpoRO}}$ directly, but must instruct a corrupted party with the correct `sid` to do so, and this way the query must go through $\mathcal{S}$), $\mathcal{S}$ "lies" as described by Camenisch et al. [10] and outputs `false` regardless of whether $\mathcal{G}_{\texttt{rpoRO}}$ was programmed or not. Otherwise, $\mathcal{S}$ behaves identically to the dummy adversary $\mathscr{A}$, forwarding communications between $\mathcal{Z}$ and the corrupted parties.

Now we wish to show that the real world, in which parties prove statements using real witnesses and verify proofs according to the protocol, is indistinguishable from the ideal world, in which the ideal functionality (with help from the simulator) proves statements by programming the RO and verifies proofs by extracting witnesses. We start with the real-world experiment and show it is possible to construct a series of hybrid experiments, each negligibly different from the last, that transform the real world experiment into the ideal world experiment.

Experiment A. The first experiment is the same as the real world experiment, except there is a "challenger" $\mathcal{C}$ who controls the environment's and adversary's views of the rest of the protocol. In particular, the challenger simulates all of the honest parties *and* $\mathcal{G}_{\texttt{rpoRO}}$. The challenger does everything on behalf of all parties exactly the same as the parties would do for themselves in the real world experiment.

Lemma 4 (REAL = Experiment A). *In the view of the environment, Experiment A is identical to the real world experiment. Formally,*

$$\mathsf{REAL}^{\mathcal{G}_{\texttt{rpoRO}}}_{\Sigma_R^{\texttt{guc}},\mathscr{A},\mathcal{Z}}(1^\lambda, \texttt{aux}) = \mathsf{ExpA}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \texttt{aux}).$$

Proof. The challenger simulates all of the real world parties in Experiment A, and the simulated output is defined to be identical to the output of the parties in the real world. □

In other words, there is no way for $\mathcal{Z}$ to tell whether it is interacting with separate parties, including the "real" $\mathcal{G}_{\texttt{rpoRO}}$, or whether it is interacting with a puppet master who simulates all of the parties, including $\mathcal{G}_{\texttt{rpoRO}}$.

Experiment B. Experiment B is the same as Experiment A, except that instead of executing real proofs on behalf of the honest parties, the challenger $\mathcal{C}$ runs the `Simulate` algorithm of $\Sigma_R^{\texttt{SLC}}$. That is, given a statement x to prove for a session s, $\mathcal{C}$ runs $\Sigma_R^{\texttt{SLC}}.\texttt{Simulate}^{\mathcal{G}_{\texttt{rpoRO}}}(x)$ to obtain π. $\mathcal{C}$ then checks to make sure that $\Sigma_R^{\texttt{SLC}}.\texttt{Verify}(x,\pi) = 1$. If it does not, $\mathcal{C}$ outputs `Fail`; otherwise, it outputs (x,π). If any of the corrupted parties make `IsProgrammed` queries, `chl` simply returns `false`, regardless of whether $\mathcal{G}_{\texttt{rpoRO}}$ was programmed on the queried index.

Lemma 5 (Experiment A $\approx_c$ Experiment B). *Provided $\Sigma_R^{\mathtt{SLC}}$ is SHVZK, Experiment B is computationally indistinguishable from Experiment A. Formally,*

$$\mathsf{ExpA}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}) \approx_c \mathsf{ExpB}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}).$$

Proof. Note that in both experiments, the challenger $\mathcal{C}$ returns random strings as the output of $\mathcal{G}_{\mathtt{rpoRO}}$. However, in Experiment B, $\mathcal{C}$ must program $\mathcal{G}_{\mathtt{rpoRO}}$'s outputs *after* the adversary begins issuing `Prove` queries, in order to maintain consistency with the simulated proofs. Recall from Definition 3 that the SLC simulator $\Sigma_R^{\mathtt{SLC}}.\mathtt{Simulate}$ essentially forks the RO by programming it, such that the adversary sees either the "normal" RO $\mathcal{G}^0_{\mathtt{rpoRO}}$ used by a real-world prover, or it sees the programmed RO $\mathcal{G}^1_{\mathtt{rpoRO}}$ that contains programmed outputs. Therefore, in order to guarantee that the hybrids are indistinguishable, we must first argue that there is only a negligible difference between $\mathcal{G}^0_{\mathtt{rpoRO}}$ and $\mathcal{G}^1_{\mathtt{rpoRO}}$.

As long as the domain of $\mathcal{G}_{\mathtt{rpoRO}}$ is exponential in the security parameter (as is standard for random oracles), $\mathcal{Z}$ is only allowed to make polynomially-many queries to $\mathcal{G}_{\mathtt{rpoRO}}$ throughout the duration of the experiment (as is required by a polynomial-time experiment), and commitments have superlogarithmic min-entropy (as described in Definition 22), the probability that the environment can predict a commitment for some statement x and query it to $\mathcal{G}_{\mathtt{roRO}}$ before obtaining a proof (such that it is able to check the challenge in the proof against the challenge it got previously from $\mathcal{G}_{\mathtt{roRO}}$) is negligible. Furthermore, recall from Definition 8 of $\mathcal{G}_{\mathtt{rpoRO}}$ that $\mathcal{Z}$ is not part of any legitimate protocol session and is therefore not allowed to make `IsProgrammed` queries of its own, and $\mathcal{C}$ answers all of the corrupted parties' `IsProgrammed` queries by returning `false`. Therefore, $\mathcal{Z}$'s view of the random oracle in Experiment A is computationally close to its view in Experiment B.

The only other potential difference between Experiments A and B is the contents of the proofs, and that Experiment B can output `Fail`, while Experiment A never does. Assume for a contradiction that $\mathcal{Z}_{AB}$ can distinguish the proof process in Experiment B from the proof process in Experiment A. We can use $\mathcal{Z}_{AB}$ as a black box to break the SHVZK property of $\Sigma_R^{\mathtt{SLC}}$ as follows.

Note first that Experiment B only outputs `Fail` if there is some internal inconsistency with the simulator, such that a proof of (x, π) for some $x \in L_R$ does not verify. In this case, after receiving a query $\mathtt{Prove}(x, \pi)$ from $\mathcal{Z}_{AB}$ and a corresponding response (x, π) from its challenger, the reduction can tell immediately if the challenger is running $\Sigma_R^{\mathtt{SLC}}.\mathtt{Simulate}$, triggering the contradiction.

Otherwise, $\mathcal{Z}_{AB}$ must be able to tell the difference between Experiments A and B by looking at the proofs themselves. If Σ_R is statistical SHVZK, then the outputs of $\Sigma_R^{\mathtt{SLC}}.\mathtt{Simulate}(x)$ are statistically close to the outputs of $\Sigma_R^{\mathtt{SLC}}.\mathtt{Prove}(x, w)$, and we are done. If Σ_R is only computational SHVZK, the reduction continues as follows.

When $\mathcal{Z}$ issues any query $(\mathtt{Prove}, x, w)$ for a proof of some statement x, $\mathcal{C}$ forwards the query to its SHVZK challenger and receives either a simulated proof (produced by running $\Sigma_R^{\mathtt{SLC}}.\mathtt{Simulate}(x)$) or a real proof (produced by running $\Sigma_R^{\mathtt{SLC}}.\mathtt{Prove}(x, w)$). $\mathcal{C}$ forwards the response back to $\mathcal{Z}$, and repeats until

$\mathcal{Z}$ outputs a bit indicating that it is living either in Experiment A or in Experiment B. If $\mathcal{Z}$ outputs "A", $\mathcal{C}$ outputs "Real" to indicate its challenger was using $\Sigma_R^{\mathtt{SLC}}.\mathtt{Prove}$; otherwise if $\mathcal{Z}$ outputs "B", $\mathcal{C}$ outputs "Simulated" to indicate its challenger was using $\Sigma_R^{\mathtt{SLC}}.\mathtt{Simulate}$. $\mathcal{C}$ succeeds in breaking the (computational) SHVZK property of $\Sigma_R^{\mathtt{SLC}}$ with this method whenever $\mathcal{Z}$ succeeds in distinguishing Experiments A and B, completing the contradiction. Therefore, the distributions representing $\mathcal{Z}$'s view of Experiment A and Experiment B are computationally indistinguishable. □

In the next experiment, Experiment C, we replace the real-world verification mechanism with extraction.

Experiment C. Experiment C is the same as Experiment B, except now instead of running the normal verification protocol on non-simulated (adversarial) proofs, the challenger $\mathcal{C}$ attempts to extract a witness as follows. Given a proof (x, π) for a session s that $\mathcal{C}$ did not previously simulate itself, $\mathcal{C}$ proceeds as follows. If $\Sigma_R^{\mathtt{SLC}}.\mathtt{Verify}(x, \pi) = 0$, $\mathcal{C}$ simply outputs 0. Otherwise if $\Sigma_R^{\mathtt{SLC}}.\mathtt{Verify}(x, \pi) = 1$, $\mathcal{C}$ gathers the environment's and adversary's queries $\mathcal{Q}_{P^*}^s$ to $\mathcal{G}_{\mathtt{rpoR0}}$ from reviewing the traffic on its wires. It then runs $\Sigma_R^{\mathtt{SLC}}.\mathtt{Extract}^{\mathcal{G}_{\mathtt{rpoR0}}}(x, \pi, \mathcal{Q}_{P^*}^s)$ to obtain w. If $R(x, w) = 1$, $\mathcal{C}$ outputs 1. Otherwise, it outputs $\mathtt{Fail}$.

Lemma 6 (Experiment B $\approx_c$ Experiment C). *Provided $\Sigma_R^{\mathtt{SLC}}$ is special simulation sound, Experiment C is computationally indistinguishable from Experiment B. Formally,*

$$\mathsf{ExpB}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}) \approx_c \mathsf{ExpC}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}).$$

Proof. Given an environment $\mathcal{Z}_{BC}$ that can distinguish between Experiment B and Experiment C, we construct a reduction that contradicts the special simulation soundness property of $\Sigma_R^{\mathtt{SLC}}$.

Consider the circumstances under which it is possible for $\mathcal{Z}_{BC}$ to notice a difference between Experiment B and Experiment C. The only difference in output between Experiments B and C is that Experiment C can fail, while Experiment B never does. In particular, Experiment C fails only when $\mathcal{Z}_{BC}$ is able to produce a proof tuple (x, π) such that $\Sigma_R^{\mathtt{SLC}}.\mathtt{Verify}^{\mathcal{G}_{\mathtt{rpoR0}}}(x, \pi) = 1$ but $R(x, w) = 0$, where w was obtained by running $\Sigma_R^{\mathtt{SLC}}.\mathtt{Extract}^{\mathcal{G}_{\mathtt{rpoR0}}}(x, \pi, \mathcal{Q}_{P^*}^s)$. Given oracle access to its challenger the $\Sigma_R^{\mathtt{SLC}}.\mathtt{Extract}$ algorithm, the reduction uses $\mathcal{Z}_{BC}$ to break the special soundness property as follows.

For $\mathtt{Prove}$ queries, the reduction proceeds as Experiment B (identical to Experiment C). Any time $\mathcal{Z}_{BC}$ wants to verify a proof tuple (x, π) for session s that the reduction did not create itself, the reduction gathers the queries $\mathcal{Q}_{P^*}^s$ and sends $(x, \pi, \mathcal{Q}_{P^*}^s)$ to its challenger, who returns w. By the logic in the preceding paragraph, an environment that can distinguish Experiments B and C with non-negligible advantage must eventually issue some proof tuple (x, π) such that $\Sigma_R^{\mathtt{SLC}}.\mathtt{Verify}^{\mathcal{G}_{\mathtt{rpoR0}}}(x, \pi) = 1$, but the witness returned by $\Sigma_R^{\mathtt{SLC}}.\mathtt{Extract}^{\mathcal{G}_{\mathtt{rpoR0}}}(x, \pi, \mathcal{Q}_{P^*}^s)$ is such that $R(x, w) = 0$. By passing this tuple to the extractor, the reduction has also successfully produced a proof (x, π) such that $\Sigma_R^{\mathtt{SLC}}.\mathtt{Verify}^{\mathcal{G}_{\mathtt{rpoR0}}}(x, \pi) = 1$, but $R(x, w) = 0$. The non-negligible existence of such a proof tuple contradicts

the special soundness property, which says if $\Sigma_R^{\mathtt{SLC}}.\mathtt{Verify}^{\mathcal{G}_{\mathtt{rpoRO}}}(x,\pi)=1$, $R(x,w)$ must equal 1 with overwhelming probability. Therefore, Experiment B must be computationally indistinguishable from Experiment C. □

Finally, we show that Experiment C is identical to the ideal-world experiment by rearranging the components to get rid of the challenger. Note that at this point, the functionality of the challenger is identical to that of $\mathcal{F}_{\mathtt{NIZK}}$ for both the `Prove` and `Verify` procedures. Therefore, we can replace $\mathcal{C}$ with $\mathcal{F}_{\mathtt{NIZK}}$ and $\mathcal{S}$, who keeps track of the corrupted parties' communications with $\mathcal{G}_{\mathtt{rpoRO}}$.

Lemma 7 (Experiment C = IDEAL). *In the view of the environment, Experiment C is identical to the ideal world experiment. Formally,*

$$\mathsf{ExpC}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda,\mathtt{aux}) = \mathsf{IDEAL}^{\mathcal{G}_{\mathtt{RO}}}_{\mathcal{F}_{\mathtt{NIZK}},\mathcal{S},\mathcal{Z}}(1^\lambda,\mathtt{aux}).$$

Proof. Note that in Experiment C, the challenger $\mathcal{C}$ answers honest parties' `Prove` queries by running $\Sigma_R^{\mathtt{SLC}}.\mathtt{Simulate}^{\mathcal{G}_{\mathtt{rpoRO}}}(x)$, and `Verify` queries by running $\Sigma_R^{\mathtt{SLC}}.\mathtt{Extract}^{\mathcal{G}_{\mathtt{rpoRO}}}(x,\pi,\mathcal{Q}^s_{P^*})$, with the same surrounding checks and procedures. Therefore, we can replace $\mathcal{C}$ in Experiment C with $\mathcal{F}_{\mathtt{NIZK}}$ in the ideal-world experiment. Since there is no longer a challenger controlling the wires in and out of the adversary, we must additionally replace $\mathscr{A}$ with the ideal adversary $\mathcal{S}$. Recall that $\mathscr{A}$ is the dummy adversary, and that $\mathcal{S}$ behaves exactly like $\mathscr{A}$ throughout the execution of the experiment, except that it forwards $\mathcal{Z}$'s communications with the corrupted parties to $\mathcal{F}_{\mathtt{NIZK}}$ through a private channel upon request, and also returns `false` to `IsProgrammed` queries. Furthermore, since $\mathcal{C}$ programs $\mathcal{G}_{\mathtt{rpoRO}}$ the same way as $\mathcal{S}$, the environment's view of $\mathcal{G}_{\mathtt{rpoRO}}$ is identical in both experiments. Therefore, the environment's view of Experiment C is identical to its view of the ideal-world experiment. □

We have now shown that the real-world experiment, which uses our construction $\Sigma_R^{\mathtt{SLC}}$, and the ideal-world experiment, which uses $\mathcal{F}_{\mathtt{NIZK}}$, are indistinguishable, completing the proof of Theorem 3. □

B.4 Full Proof of Theorem 3

Recall Theorem 3: Let Σ_R be any Σ-protocol for relation R (Definition 2), $\mathcal{G}_{\mathtt{roRO}}$ be the restricted observable global random oracle (Definition 7), `SLC` be any straight-line compiler (Definition 3), $\mathcal{F}_{\mathtt{CRS}}$ be the ideal CRS functionality (Definition 10), Σ_S be a Σ-protocol for a samplable-hard relation S (Definition 13), and `guc` be the algorithm described in Definition 14. Then the NISLE Σ-protocol $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}} \leftarrow \mathtt{guc}(\Sigma_R,\mathtt{SLC})$ GUC-realizes $\mathcal{F}_{\mathtt{NIZK}}$ in the $\mathcal{G}_{\mathtt{roRO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model (Definition 12).

Proof. We must show that $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}} \leftarrow \mathtt{guc}(\Sigma_R,\mathtt{SLC})$ GUC-realizes $\mathcal{F}_{\mathtt{NIZK}}$ in the $\mathcal{G}_{\mathtt{RO}}$-$\mathcal{F}_{\mathtt{CRS}}$-hybrid model—that is, we must satisfy Definition 12. Briefly, we must show that for all efficient $\mathscr{A}$, there exists an ideal adversary $\mathcal{S}$ efficient in expectation such that for all efficient environments $\mathcal{Z}$,

$$\mathsf{IDEAL}^{\mathcal{G}_{\mathtt{RO}}}_{\mathcal{F}_{\mathtt{NIZK}},\mathcal{S},\mathcal{Z}}(1^\lambda,\mathtt{aux}) \approx_c \mathsf{REAL}^{\mathcal{G}_{\mathtt{RO}},\mathcal{F}_{\mathtt{CRS}}}_{\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}},\mathscr{A},\mathcal{Z}}(1^\lambda,\mathtt{aux}).$$

Construction of the Simulator $\mathcal{S}$. The simulator (also known as the ideal adversary) $\mathcal{S}$, works as follows. When the ideal functionality $\mathcal{F}_{\mathtt{NIZK}}$ asks it for the specification of algorithms, $\mathcal{S}$ returns the algorithms in $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}$. When $\mathcal{F}_{\mathtt{NIZK}}$ asks it for the queries of adversarial provers for an SID s, $\mathcal{S}$ returns the corrupted parties' $\mathcal{G}_{\mathtt{roRO}}$ queries $\mathcal{Q}^s_{\mathscr{A}}$. Otherwise, $\mathcal{S}$ behaves identically to the dummy adversary $\mathscr{A}$, forwarding communications between $\mathcal{Z}$ and the corrupted parties.

Now we wish to show that the real world, in which parties prove statements using real witnesses and verify proofs according to the protocol, is indistinguishable from the ideal world, in which the ideal functionality (with help from the simulator) proves statements using the trapdoor to the CRS and verifies proofs by extracting witnesses. We again start with the real world experiment and show it is possible to construct a series of hybrid experiments, each negligibly different from the last, that transform the real world experiment into the ideal world experiment.

Experiment A. The first experiment is the same as the real-world experiment, except there is again a "challenger" $\mathcal{C}$ who controls the environment's and adversary's views of the rest of the protocol. In particular, the challenger simulates all of the honest parties (including the subroutine calls to $\mathcal{F}_{\mathtt{CRS}}$) *and* $\mathcal{G}_{\mathtt{roRO}}$. The challenger does everything on behalf of all parties exactly the same as the parties would do for themselves in the real world experiment.

Lemma 8 (REAL = Experiment A). *In the view of the environment, Experiment A is identical to the real world experiment. Formally,*

$$\mathsf{REAL}^{\mathcal{G}_{\mathtt{RO}},\mathcal{F}_{\mathtt{CRS}}}_{\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}},\mathscr{A},\mathcal{Z}}(1^\lambda,\mathsf{aux}) = \mathsf{ExpA}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda,\mathsf{aux}).$$

Proof. The challenger simulates all of the real world parties in Experiment A, and the simulated output is defined to be identical to the output of the parties in the real world. □

In other words, there is no way for $\mathcal{Z}$ to tell whether it is interacting with separate parties, including the "real" $\mathcal{G}_{\mathtt{roRO}}$, or whether it is interacting with a puppet master who simulates all of the parties, including $\mathcal{G}_{\mathtt{roRO}}$. In the next experiment, the challenger will leverage this identical view to invoke the "traditional" simulator of the straight-line extractable OR-protocol $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}$, which uses a programmable RO. Hiding the CRS trapdoor from the challenger while allowing it to simulate proofs via programming will allow us to construct a reduction (in future steps) that can either break special soundness or extract the CRS trapdoor. Eventually, we will arrive at the conclusion that the programming view in Experiment B is indistinguishable from the "unconventional" simulator in $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}$, which uses the trapdoor to the CRS.

Experiment B. Experiment B is the same as Experiment A, except that instead of executing real proofs on behalf of the honest parties, the challenger $\mathcal{C}$ programs $\mathcal{G}_{\mathtt{roRO}}$ in order to simulate both components of the OR-protocol. That is, given a statement x to prove for a session s, $\mathcal{C}$ prepares the compound statement $X = (x, \mathtt{CRS}_s)$ by simulating the functionality of $\mathcal{F}^s_{\mathtt{CRS}}$. It then computes the proof

(X, Π) by running the simulator of the straight-line extractable OR-protocol, $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Simulate}(X)$, and outputs (X, Π).

Lemma 9 (Experiment A $\approx_s$ Experiment B). *Provided $\Sigma_{\mathtt{R}\vee\mathtt{S}}$ is statistical (resp. computational) SHVZK, Experiment B is statistically (resp. computationally) indistinguishable from Experiment A. Formally,*

$$\mathsf{ExpA}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}) \approx_s \mathsf{ExpB}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}).$$

Proof. The proof is the same the proof of Lemma 5 in Section 4, except that we do not have to consider the `Fail` condition. □

In the next experiment, Experiment C, we again replace the real-world verification mechanism with extraction. We proceed to show via a reduction that an environment that can distinguish between Experiment B, which uses real-world verification, and Experiment C, which uses the same extraction functionality as $\mathcal{F}_{\mathtt{NIZK}}$, can be used to contradict either the special simulation soundness property of $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}$ or the hardness property of the samplable hard relation used to construct the CRS.

Experiment C. Experiment C is the same as Experiment B, except now instead of running the normal verification algorithm on non-simulated (adversarial) proofs (X, Π), the challenger $\mathcal{C}$ proceeds as follows. If $\mathcal{C}$ previously simulated (X, Π), $\mathcal{C}$ outputs 1. If $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Verify}(X, \Pi) = 0$, $\mathcal{C}$ outputs 0. Otherwise if (X, Π) is not a simulated proof and $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Verify}(X, \Pi) = 1$, $\mathcal{C}$ runs $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Extract}(X, \Pi)$ to obtain $W = (w_0, w_1)$. If W is such that $R_{\mathtt{R}\vee\mathtt{S}}(X, W) = 1$, $\mathcal{C}$ outputs 1. Otherwise if $R_{\mathtt{R}\vee\mathtt{S}}(X, W) = 0$ or $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Extract}$ outputs `Fail`, $\mathcal{C}$ outputs `Fail`.

Lemma 10 (Experiment B $\approx_c$ Experiment C). *Experiment C is computationally indistinguishable from Experiment B. Formally,*

$$\mathsf{ExpB}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}) \approx_c \mathsf{ExpC}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}).$$

Proof. Note that there are now two conditions under which Experiment C can fail and behave differently from Experiment B. Given an environment $\mathcal{Z}_{BC}$ that can distinguish between Experiment B and Experiment C, we construct a reduction that contradicts either the special simulation soundness property of $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}$ or the hardness property of the samplable hard relation used to construct the CRS. The first part of the reduction—the reduction to special soundness—is identical to the reduction from Lemma 6 in Section 4 above. This rules out the first failure: that $\Sigma^{\mathtt{SLC}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Verify}(X, \Pi) = 1$, but $R_{\mathtt{R}\vee\mathtt{S}}(X, W) = 0$.

The second failure condition occurs when $\mathcal{Z}_{BC}$ is able to produce some proof (X, Π) in some session s that causes $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Extract}(X, \Pi)$ to output `Fail`. Recall that this condition happens when $R_{\mathtt{R}\vee\mathtt{S}}(X, W) = 1$ but $R(x_0, w_0) = 0$—that is, $R_{\mathtt{R}\vee\mathtt{S}}(X, W) = 1$ because $S(x_1, w_1) = 1$, where $x_1 = \mathtt{CRS}_s$ and $w_1 = \mathtt{trap}_s$. In other words, the failure occurs if $\mathcal{Z}_{BC}$ is able to produce a proof that verifies using the CRS trapdoor that should only be available to the simulator for session s in the ideal world.

We use this $\mathcal{Z}_{BC}$ as a black box to construct a reduction that breaks the hardness property of the samplable hard relation S as follows. For $\texttt{Prove}$ queries, the reduction proceeds as Experiment B/C, except that it obtains the CRS $\texttt{CRS}_s$ for each SID s from its challenger the samplable-hard CRS sampling algorithm κ_S. The reduction sets $\texttt{CRS}_s = x$ and answers $\texttt{Prove}(x, w)$ queries as usual, by setting $X = (x, \texttt{CRS}_s)$ and running $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{Simulate}(X)$. It answers queries $\texttt{Verify}(X, \Pi)$ for $X = (x, \texttt{CRS}_s)$ according to Experiment C until $\mathcal{Z}_{BC}$ produces a proof (X, Π) for session s that causes $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{Extract}(X, \Pi)$ to return a $W = (w_0, w_1)$ such that $S(\texttt{CRS}_s, w_1) = 1$. The reduction can now produce a witness w_1 such that $S(\texttt{CRS}_s, w_1) = 1$, contradicting the hardness property of S, which says that the probability of computing a w' for some $x \leftarrow \kappa_S(1^\lambda)$ such that $S(x, w') = 1$ is negligible in λ.

Therefore, both failure conditions happen with negligible probability, and Experiment B is computationally indistinguishable from Experiment C. □

Finally, we replace the simulated proof process from Experiment B, which uses straight-line extractable OR-protocol simulator $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{Simulate}$, with the GUC-transform simulator $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{Simulate}$, which proceeds as a "genuine" prover using the trapdoor to the CRS rather than a witness to the statement x. This process essentially reverts the change between Experiments A and B, since the challenger is going back to using the $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{Prove}$ algorithm, only this time with the witness $W = (\texttt{trap}_s, 1)$ rather than the witness $W = (w, 0)$.

Experiment D. Experiment D is the same as Experiment C, except in how it generates the honest participants' proofs. Rather than programming $\mathcal{G}_{\texttt{roRO}}$, the challenger computes proofs of statements x for the honest parties by running $\Sigma^{\texttt{guc}}_{\texttt{R}\vee\texttt{S}}.\texttt{Simulate}(x)$. Recall that this process consists of generating the CRS and trapdoor pair $(\texttt{CRS}_s, \texttt{trap}_s)$ for each session s according to the sampling algorithm $\kappa_S(1^\lambda)$, then running $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{Prove}(X, W)$ for $X = (x, \texttt{CRS}_s)$ and $W = (\texttt{trap}_s, 1)$.

Lemma 11 (Experiment C $\approx_s$ Experiment D). *Provided $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}$ is statistical (resp. computational) SHVZK, Experiment D is statistically (resp. computationally) indistinguishable from Experiment C. Formally,*

$$\mathsf{ExpC}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \texttt{aux}) \approx_{s(c)} \mathsf{ExpD}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \texttt{aux}).$$

Proof. This step reverts the proofs to being effectively non-simulated as in experiment A, since using the trapdoor witness involves computing the OR-protocol honestly according to $\Sigma^{\texttt{SLC}}_{\texttt{R}\vee\texttt{S}}.\texttt{Prove}(X, W)$ for $X = (x, \texttt{CRS}_s)$ and $W = (\texttt{trap}_s, 1)$. Moreover, the environment's view of the CRS is the same as in Experiment A, since we defined $\mathcal{F}_{\texttt{CRS}}$ to use $\kappa_S(1^\lambda)$ as the CRS generation functionality. Therefore, the argument for *statistical* indistinguishability between the OR-protocol-simulated proofs in Experiment C and the GUC-transform-simulated proofs in Experiment D is again identical to the (statistical) argument from Lemma 5.

If there is computational wiggle room between the proofs in Experiments C and D, *and* the Experiment C-D distinguisher environment $\mathcal{Z}_{CD}$ is now dealing with the extractor rather than a normal verifier, we cannot use the exact same

argument from the proof of Lemma 5. In particular, we have to make sure that the *only* way that $\mathcal{Z}_{CD}$ can distinguish between hybrid j and hybrid $j+1$ is if it can tell the difference between a real and a simulated proof in the $j+1^{st}$ slot. Otherwise—if $\mathcal{Z}_{CD}$ could somehow compose its knowledge of the simulated proofs with whatever it obtains from the extractor in order to construct a proof that causes the extractor to fail—$\mathcal{Z}_{CD}$ would be able to distinguish the experiments immediately, regardless of the nature of the $j+1^{st}$ proof.

We argue that because anything $\mathcal{Z}_{CD}$ can learn from a straight-line extractor it can learn from itself, it must not learn anything new about the proofs between Experiments C and D. This is a substantial bonus of straight-line extraction—it stops the the extractor from getting in the way of other desirable properties of the system.

Consider the inputs $(X, \Pi, \mathcal{Q}^s_{P^*})$ to the algorithm $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Extract}$ that is responsible for the verification procedure in Experiments C and D. (X, Π) is a proof that $\mathcal{Z}_{CD}$ itself produced. Similarly, $\mathcal{Q}^s_{P^*}$ is a list of queries to $\mathcal{G}_{\mathtt{roRO}}$ made by either $\mathcal{Z}_{CD}$ itself, or by the corrupted parties through $\mathscr{A}$ at $\mathcal{Z}_{CD}$'s request. Therefore, $\mathcal{Z}_{CD}$ can fully simulate its own view of the extractor, and cannot possibly learn anything new about whether it is living in Experiment C or Experiment D from the proof verification process.

We have shown that if $\mathcal{Z}_{CD}$ is able to distinguish between the hybrids, it must be able to distinguish whether the proof in the $j+1^{st}$ slot is real or simulated. The rest of the argument is the same as the computational section of the proof of Lemma 5.

Finally, we show that Experiment D is identical to the ideal world experiment by rearranging the components to get rid of the challenger. Note that at this point, the functionality of the challenger is identical to that of $\mathcal{F}_{\mathtt{NIZK}}$ for both the `Prove` and `Verify` procedures. Therefore, we can replace $\mathcal{C}$ with $\mathcal{F}_{\mathtt{NIZK}}$ and $\mathcal{S}$, who takes over keeping track of the corrupted parties' communications with $\mathcal{G}_{\mathtt{RO}}$.

Lemma 12 (Experiment D = IDEAL). *In the view of the environment, Experiment D is identical to the ideal world experiment. Formally,*

$$\mathsf{ExpD}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}) = \mathsf{IDEAL}^{\mathcal{G}_{\mathtt{roRO}}}_{\mathcal{F}_{\mathtt{NIZK}},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathtt{aux}).$$

Proof. Note that in Experiment D, the challenger $\mathcal{C}$ answers honest parties' `Prove` queries by running $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Simulate}(x)$, and `Verify` queries by running $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}.\mathtt{Extract}(X, \Pi)$, with the same surrounding checks and procedures. Therefore, we can replace $\mathcal{C}$ in Experiment D with $\mathcal{F}_{\mathtt{NIZK}}$ in the ideal-world experiment. Since there is no longer a challenger controlling the wires in and out of the adversary, we must additionally replace $\mathscr{A}$ with the ideal adversary $\mathcal{S}$. Recall that $\mathscr{A}$ is the dummy adversary, and that $\mathcal{S}$ behaves exactly like $\mathscr{A}$ throughout the execution of the experiment, except that it forwards $\mathcal{Z}$'s communications with the corrupted parties to $\mathcal{F}_{\mathtt{NIZK}}$ through a private channel upon request. Furthermore, since $\mathcal{C}$ is no longer programming $\mathcal{G}_{\mathtt{roRO}}$ in order to simulate proofs in Experiment D, the functionality of $\mathcal{G}_{\mathtt{roRO}}$ is identical in both experiments. Therefore,

the environment's view of Experiment D is identical to its view of the ideal-world experiment. □

We have now shown that the real-world experiment, which uses our construction $\Sigma^{\mathtt{guc}}_{\mathtt{R}\vee\mathtt{S}}$, and the ideal-world experiment, which uses $\mathcal{F}_{\mathtt{NIZK}}$, are indistinguishable, completing the proof of Theorem 3. □

B.5 Full Proof of Theorem 4

Recall Theorem 4: Provided Σ_R is a Σ-protocol for relation R according to Definition 2 with strong special soundness as given in Definition 15, the randomized Fischlin transform $\mathtt{Fis}$ for Σ_R described in Definition 28 is a straight-line compiler according to Definition 3.

Proof. Kondi and shelat prove in Theorem 6.4 of their work [36] that the tuple of algorithms $\Sigma_R^{\mathtt{Fis}}$ (denoted $\pi_{\mathtt{NIZK}}^{\mathtt{F-rand}}$ in their paper) produced by running the randomized Fischlin transform on any strong special sound Σ-protocol Σ_R for relation R is a non-interactive straight-line extractable zero-knowledge proof of knowledge for L_R in the random-oracle model. Since Kondi and shelat use the standard definitions of special SHVZK and strong special soundness (Definitions 19 and 15, respectively), it remains to show that $\Sigma_R^{\mathtt{Fis}}$ satisfies the special multi-SHVZK property from Definition 5 and the special simulation soundness property from Definition 6.

We argue that almost the exact same arguments from the proof of Theorem 3 of the full version of Fischlin's paper [30] can be used to show that Kondi and shelat's transform also satisfies special multi-SHVZK and special simulation soundness. We briefly review the identical aspects of the proof and discuss the differences in depth below.

The multi-SHVZK property follows identically regular (single-proof) SHVZK because of the independence and superlogarithmic entropy of the commitments (such that the oracles in both experiments are still indistinguishable), along with a hybrid argument that distinguishing the j from the $j+1^{st}$ proof of a multi-proof simulator would allow a reduction to distinguish whether the $j+1^{st}$ proof from its SHVZK challenger was real or simulated.

Simulation soundness follows from a reduction to the multi-SHVZK property and the regular special soundness extractor as follows. First, Fischlin rules out some trivial attacks in which the adversary modifies an existing proof $\pi = (\mathtt{com}, \mathtt{chl}, \mathtt{res})$ for a statement x to produce some new accepting transcript $\pi' = (\mathtt{com}, \mathtt{chl}, \mathtt{res}')$ where $\mathtt{res} \neq \mathtt{res}'$. In Fischlin's proof this attack is ruled out by the unique responses property, which guarantees that if $\Sigma_R.\mathtt{Verify}(x, \pi) = \Sigma_R.\mathtt{Verify}(x, \pi') = 1$, $\mathtt{res} = \mathtt{res}'$ with overwhelming probability. In our proof, this attack is ruled out by strong special soundness, which guarantees that $\Sigma_R.\mathtt{Extract}(x, \pi, \pi')$ will still produce a witness w such that $R(x, w) = 1$ for $\mathtt{res} \neq \mathtt{res}'$. The extractor still works in both cases for proofs $\pi = (\mathtt{com}, \mathtt{chl}, \mathtt{res})$ and $\pi' = (\mathtt{com}, \mathtt{chl}', \mathtt{res}')$ where $\mathtt{chl} \neq \mathtt{chl}'$. Therefore, Fischlin proceeds assuming the adversary has generated a proof with a fresh commitment vector for its statement x, and shows that for such a proof, the multi-SHVZK property implies special simulation soundness.

Te rest of the argument is identical to the proof of Theorem 3. Briefly, Fischlin proceeds by contradiction, using an algorithm B with oracle access to the simulator that can produce a proof causing the extractor to fail as a black box in order to contradict either the multi-SHVZK property or the regular special soundness property of $\Sigma_R^{\texttt{Fis}}$. First, the reduction creates a distinguisher algorithm D^H with oracle access to H to encompass the "real" B that simply passes inputs and outputs between B, the reduction, and H, and returns 1 whenever B is able to produce a valid but non-extractable proof (x_i, π_i) such that $\Sigma_R^{\texttt{Fis}}.\texttt{Verify}(x_i, \pi_i) = 1$ but $R(x, w_i) = 0$. Next, the reduction creates a second distinguisher algorithm A^H also with oracle access to H that simulates a different copy of B, denoted B', that similarly passes inputs and outputs and returns 1 whenever B' is able to produce a valid but non-extractable proof.

When the real B issues the i^{th} query $(\texttt{Prove}, x_i, w_i)$, D passes this query to the reduction, who queries its multi-SHVZK challenger for a proof π that is either the result of running $\Sigma_R^{\texttt{Fis}}.\texttt{Prove}(x_i, w_i)$ or $\Sigma_R^{\texttt{Fis}}.\texttt{SimProve}(x_i, z_i, \texttt{chl}_i)$ for some $\texttt{chl}$. The reduction returns π to D who returns it to B. When the simulated copy B' running inside of A issues queries $(\texttt{Prove}, x_i, w_i)$, A creates a real proof $\pi' \leftarrow \Sigma_R^{\texttt{Fis}}.\texttt{Prove}^H(x_i, w_i)$ and returns π' to B'. This step essentially reduces B' to the adversary in the regular special soundness experiment for Σ_R, who is free to run the $\texttt{Prove}$ algorithm on anything it wants. We note here that since the challenges produced by Kondi and shelat's transform are distributed identically to those produced by Fischlin's, B and B' views of the proofs in this experiment are identical to those in the original experiment conducted by Fischlin. Any time B (res. B') issues a proof (x_i, π_i), D (resp. A) checks that $\Sigma_R^{\texttt{Fis}}.\texttt{Verify}(x_i, \pi_i) = 1$, obtains w by running $\Sigma_R^{\texttt{Fis}}.\texttt{Extract}(x_i, \pi_i)$ and outputs 1 if $R(x, w_i) = 0$. The reduction outputs whatever D outputs.

Consider the eventual outputs of D and A. If the reduction is communicating with the simulator in the ideal-world multi-SHVZK experiment, then the reduction successfully outputs 1 to indicate it is running inside Multi-SHVZK-IDEAL whenever D successfully outputs 1 to indicate that B has produced a valid non-extractable proof. Since we assumed this probability to be non-negligible by assumption, we arrive at a contradiction of the multi-SHVZK property. If the reduction is communicating with a real prover in the real-world multi-SHVZK experiment, D and therefore B must succeed with the same probability as A by the following logic. Because extraction relies only on the relevant adversary's queries, the functionality of the extractor for adversary B' is independent of whether the multi-SHVZK challenger is also running real $\texttt{Prove}$ queries—in other words, A's output does not rely on the RO queries issued by the multi-SHVZK challenger and vice versa. D and A are therefore essentially identical, parallel (independent) experiments, such that D's output (sourced from B) and A's output (sourced from B') must be identically distributed. Therefore if B succeeds with non-negligible probability, so does A, contradicting the underlying special soundness property of $\Sigma_R^{\texttt{Fis}}$. □