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

Abstract. Numerous cryptographic applications require efficient non-interactive zero-knowledge proofs of knowledge (NIZK PoK) 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 universal composition (UC) model of Canetti is the best guarantee that nothing will go wrong when a system is part of a larger whole. In this paper, we show how to achieve efficient UC-secure NIZK PoK in the global random-oracle model of Canetti, Jain, and Scafuro.

1 Introduction

Non-interactive zero-knowledge proofs of knowledge (NIZK PoK) [6, 26, 41] 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 [45] 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 [10] and e-token [9] systems use them to compute proofs of validity of an e-coin or e-token. In group signatures [16, 2] they are used to ensure that the signer is in possession of a group signing key. In anonymous credential constructions [11, 12], 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 NIZK PoK 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 (e.g. those performed on the blockchain). It is also extremely important that the NIZK PoK be efficient. Thus, the constructions cited above use efficient Σ-protocols [24] made non-interactive via the Fiat-Shamir heuristic [27] to instantiate the NIZK PoK in the random-oracle model (ROM) [4]. Recall that a Σ-protocol for a relation R is, in a nutshell, a $(1 - negl)$-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 it non-interactive by replacing the message from the verifier by the output of the random oracle (RO).

Recently, a better understanding of how such NIZK PoK fare in the *concurrent* setting emerged [43, 25, 5, 37]. Allowing for secure concurrent executions is of vital importance for the real-world application of any of the cryptographic protocols mentioned above, especially for distributed protocols where participants can join or leave at any time. But Drijvers et al. [25] 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 [43] and Benhamouda et al.'s exploitation of the ROS problem [5].

One way to circumvent the unique subtleties of composing cryptographic primitives is to prove that each primitive is *universally composable* (UC-secure) using Canetti's universal composition (UC) framework [17]. 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 idea of an "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, including shared state, a shared RO or common reference string (CRS). In real-world applications—and especially in complex decentralized systems—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 et al. 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 [18]. This model is more general than the earlier "universal composition with joint state" (JUC) model developed by Canetti and Rabin [20]. Canetti et al. later showed several practical applications of the GUC framework with a global RO $\mathcal{G}_{RO}$ as the only trusted setup, including commitment, oblivious transfer, and secure function evaluation protocols, all GUC-secure in the $\mathcal{G}_{RO}$-hybrid model [19].

Thus, the $\mathcal{G}_{RO}$-hybrid model is an attractive one for constructing and analyzing practical and provably secure non-interactive zero-knowledge proofs. Obtaining an efficient NIZK PoK (for a relation R) in the $\mathcal{G}_{RO}$-hybrid model from an efficient Σ-protocol (for the same relation) seems like a natural goal. Unfortunately, Canetti et al. point out that it is not possible to construct ZK PoK in less than two rounds of communication using *only* a global functionality, due to the fact that there is no way for the simulator in the security experiment to exercise control over it, for instance by programming the RO or having a trapdoor to the CRS [19, 38]. Therefore, we set our sights on the combined $\mathcal{G}_{RO}$-CRS hybrid model.

The $\mathcal{G}_{RO}$-CRS hybrid model is a good compromise for the analysis of NIZK PoK in the GUC framework. While the CRS is technically a local functionality (and therefore subject to the same rigid separation as local functionalities in the original UC model), Canetti et al.'s results on GUC-secure computation with a global RO [19] would enable parties to pre-compute the CRS for their session

using only $\mathcal{G}_{RO}$. This would offload the computation to a one-time cost at the beginning of a protocol session, which could cover an entire system of protocols involving NIZK PoK among various parties. Notably since the environment in the GUC framework is allowed to spawn arbitrary parties to take part in any session, the security of the pre-computed CRS is maintained with participant turnover—parties would not need to recompute the CRS or start a new session every time a party joined or left the session.

In this work, we ask the question, "what kind of composable security guarantees can we get for Σ-protocols in a setting where all parties share the same random oracle?" We answer by introducing a *GUC-transform* that takes *any* Σ-protocol and turns it into a non-interactive, GUC-secure NIZK PoK in the $\mathcal{G}_{RO}$-CRS hybrid model. In other words, we show that any Σ-protocol can be made to emulate, or GUC-realize, the desired ideal functionality of a NIZK PoK in the $\mathcal{G}_{RO}$-CRS hybrid model. While traditionally Σ-protocols are statistical zero-knowledge, we show that our transform still works in the case that the underlying Σ-protocol is only computational zero-knowledge. The efficiency of our construction reduces to the efficiency of the Fischlin transform [28], which requires only a linear increase in the size of the proofs for small multiplicative and additive constants.

In the remainder of the introduction, we provide general background information about Σ-protocols, the GUC framework, the $\mathcal{G}_{RO}$-hybrid model, and the Fischlin transform, discuss our contributions, and present the organization of the rest of the paper.

1.1 Σ-Protocols

A Σ-protocol for a 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, it 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, special *simulation* soundness, says that special soundness must still hold even if an adversary has seen polynomially-many proofs from the simulator. The purpose of the special soundness property is to afford

the verifier security against a cheating prover, who might try to prove a statement for which it does not know a witness. This is a variant of the "proof of knowledge" property, which quantifies what it means for a prover to "know" a witness [31, 3].

The Σ-protocol format captures many practical zero-knowledge proof systems. For example, Wikström [45] 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 [21, 15]. A rich body of literature exists giving Σ-protocols for proving that values committed using Pedersen [39] and Fujisaki-Okamoto [29] commitments satisfy general algebraic and Boolean circuits [8, 13, 14] and lie in certain integer ranges [7, 35]. 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. These Σ-protocols (and their Fiat-Shamir non-interactive counterparts) are used in numerous cryptographic protocols (as discussed above) and also made their way into implementations such as Idemix and Hyperledger.

Σ-protocols are also the most efficient technique to achieve zero-knowledge proofs of knowledge of a commitment opening in the lattice setting [36, 23], 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-composable NIZK PoK in the $\mathcal{G}_{RO}$-CRS hybrid model.

1.2 The General Universal Composability (GUC) Model

Our security experiment is that of the GUC model of Canetti et al. [18], 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 whose behavior is directed by $\mathcal{Z}$.

The ideal world additionally contains an ideal functionality $\mathcal{F}$. 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. 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 (G)UC-emulates a functionality $\mathcal{F}$ if for any real-world adversary $\mathcal{A}$ 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 in the cryptographic protocol.

In our case, the ideal functionality is the ideal NIZK PoK functionality, or $\mathcal{F}_{\texttt{NZP}}$, which works as follows. An honest participant can compute a proof π of

knowledge of w such that $R(x, w)$ by querying $\mathcal{F}_{\texttt{NZP}}$'s `Prove` interface and giving it (x, w). The string π itself is computed according to the algorithm `Simulate` provided by the ideal adversary $\mathcal{S}$. The functionality guarantees the zero-knowledge property because `Simulate` is independent of w. An honest participant can also verify a purported proof π for x by querying $\mathcal{F}_{\texttt{NZP}}$'s `Verify` interface and giving it (x, π). $\mathcal{F}_{\texttt{NZP}}$ ensures the soundness of the proof system as follows: if the proof π was *not* issued by $\mathcal{F}$, 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 the ideal world, the ideal adversary $\mathcal{S}$ has two functions. First, it specifies the algorithms that the ideal functionality will use in order to create responses for the honest parties, i.e., simulate proofs of statements, and extract witnesses from adversarially-created proofs. Second, it serves as the communication channel for the corrupted parties: all communications going in to and coming out of corrupted parties are passed through $\mathcal{S}$. This is the "ideal" version of $\mathscr{A}$ who, in the real-world model, serves as the communication channel between $\mathcal{Z}$ and the corrupted parties. Of course in order to avoid tipping off the environment, $\mathcal{S}$ cannot treat $\mathcal{Z}$'s instructions to the corrupted parties any differently than $\mathscr{A}$ would. It can, however, pass the communications of the corrupted parties—and specifically the parties' queries to the RO—to the ideal functionality through a private channel upon request. As an added constraint, we allow only *non-adaptive* corruptions—that is, the environment must decide at the time of a party's initialization whether or not it is corrupted.

In Canetti's original UC framework [17], all communications between instances of ITMs 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 $\mathcal{G}_{RO}$, 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. [18].

1.3 The Global Random Oracle $\mathcal{G}_{RO}$

As defined by Canetti, Jain, and Scafuro [19], the global functionality [18] $\mathcal{G}_{RO}$ 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. Significantly, $\mathcal{G}_{RO}$ allows us to capture the realistic scenario in which the same RO is reused by many parties over many executions in numerous distinct protocols. The global ROM is both more general and affords stronger security guarantees than the traditional ROM.

Like in the traditional ROM, adversarial queries to $\mathcal{G}_{RO}$ are of essential importance in the security experiment. The simulator (ideal adversary) in the security proof of a protocol Π emulating an ideal functionality $\mathcal{F}$ in the $\mathcal{G}_{RO}$-hybrid model is able to access the queries of corrupted parties using the same mechanism

as in the traditional ROM—by monitoring the communication wires between the corrupted parties and the rest of the experiment. The *environment's* queries to $\mathcal{G}_{RO}$, on the other hand, are not directly monitored by the simulator. Since $\mathcal{G}_{RO}$ 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 these protocols.

In order to ensure these queries are still available to the simulator, Canetti et al. designed $\mathcal{G}_{RO}$ to check whether the SID for a query matches the SID of the querying party [19]. In the event that it does not, this query is labelled "illegitimate." $\mathcal{G}_{RO}$ 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 in the case of NIZK PoK, the algorithm that the ideal functionality $\mathcal{F}$ uses to verify proofs can use these queries to extract witnesses.

1.4 The Fischlin Transform

The Fischlin transform [28] is a non-interactive transform for Σ-protocols that allows for *straight-line* (or *online*) extraction, a process by which the extractor can produce a witness straight from a proof that passes verification 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 adversarial prover to a previous state and hopes for a certain pattern of interaction before a witness can be output. Straight-line extraction is necessary in the UC framework because it is structurally infeasible to rewind the environment (the definition of the UC framework simply does not allow it). Furthermore, straight-line extraction produces a tight reduction, which avoids security nuances surrounding the forking lemma [30].

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. We describe the process informally below, and include Fischlin's original definition in Appendix A.5.

First, the prover generates a vector of r commitments, where r is a parameter of the system. For each commitment, the prover iterates through each t-bit challenge $0, 1, \ldots, 2^{t-1}$, 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.

1.5 Our Contributions

Our main contribution is the GUC-transform for Σ-protocols in the $\mathcal{G}_{RO}$-CRS hybrid model. As a secondary result, we prove that our transform still works when the underlying Σ-protocol is only computational zero-knowledge. In addition to our construction, we develop the $\mathcal{G}_{RO}$-CRS hybrid model, which may be of independent interest when analyzing the composability of more complex protocols in the global ROM. Throughout the paper we reframe existing definitions as algorithms with precise inputs and outputs, making them readily transformable and compatible with the modularity of the UC framework.

Our transform works as follows. To guarantee the security of the prover's witness (and achieve the special honest-verifier zero-knowledge property) without programming $\mathcal{G}_{RO}$, our construction uses an OR-protocol [22, 24], where the prover proves that either it knows a real witness to a statement, or else it knows the simulator's secret trapdoor to the CRS. To guarantee security against a cheating prover (and achieve the special soundness property) without rewinding the environment, we leverage Fischlin's straight-line extractor [28].

In proving our construction GUC-secure in the $\mathcal{G}_{RO}$-CRS hybrid model, we unearth an interesting nuance about straight-line extraction. Because straight-line extractors work solely based on the adversary's previous queries to the RO, the adversary cannot learn anything new from interacting with the extractor that it could not compute itself. Unlike decryption oracles in chosen ciphertext attack experiments, the self-simulatable nature of straight-line extraction means that adversarial access to an extraction oracle does not impact the rest of the reduction, and will compose easily with other desirable security properties such as statistical and computational zero-knowledge. Similar to Fischlin's observation that his construction "decouples" the protocol from the RO, we observe that his construction also decouples the extraction process from the rest of the reduction.

This decoupling property, and 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 [44, 34]. Notably, Unruh [42] and Katsumata [33] have developed straight-line extractable transforms in the (programmable) QROM. A transform similar to ours might leverage these straight-line extractable transforms to achieve a version of GUC-security for quantum-secure NIZK PoK in the global (non-programmable) QROM.

Organization. In Section 2, we present notation and preliminary definitions we will use throughout the paper. We formulate the ideal functionality of NIZK PoK as well as the $\mathcal{G}_{RO}$-CRS hybrid model in Section 3, followed by our main construction of straight-line extractable OR-protocols and the GUC-transform.

In Section 4, we prove that our construction GUC-realizes the NIZK PoK functionality in the $\mathcal{G}_{RO}$-CRS hybrid model.

2 Preliminaries

Notation. We use λ for the security parameter, and say an algorithm is efficient (in λ) if its runtime can be expressed as a polynomial on input λ, or $\mathsf{poly}(\lambda)$. An efficient algorithm that uses randomness in its computation is called probabilistic polynomial-time, abbreviated PPT.

By $y \leftarrow \mathscr{A}(x)$, we mean the output y is obtained by running algorithm $\mathscr{A}$ on input x. By $y \leftarrow Z$ where Z is a set or a probability distribution, we mean an element y sampled from Z. When we enclose a variable y in brackets $\{y\}$, we reference the probability distribution over all possible values for y. 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 $Y \approx_c Z$.

A protocol is a collection of algorithms executed by multiple parties. We say a party is "honest" in its execution of a protocol if it follows the specification of the protocol. Following the UC framework [17], we model our protocol participants as interactive Turing machines (ITMs). Each instance of an ITM has a unique identity tape including a personal identifier (PID) and a session identifier (SID). In the spirit of the precision and modularity of the UC model, we distill the functionalities existing primitives into tuples of algorithms. Where the algorithmic definition is simply a reorganization of an existing definition, we state the source and provide the original version in the Appendix.

We adopt several assumptions for the convenience of simple notation and security analysis. To avoid the repeated inclusion of public parameters and state information in the specifications of algorithms, we assume the calling ITMs are *stateful*. When algorithms reference variables that are not explicitly included as input, we assume these variables were previously initialized (for instance by some setup algorithm), and are available in memory. Similarly, we assume that all messages exchanged as part of the security experiment implicitly include the sender's PID and SID. Finally, when an algorithm is designed to reject a request, rather than output nothing and stall the experiment indefinitely, we use the special "empty" symbol, $\perp$.

2.1 Σ-protocols Revisited

Recall from the introduction 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. If the verifier accepts all of the values $(\texttt{com}, \texttt{chl}, \texttt{res})$ as a "proof" of the statement x, it outputs 1 to accept the proof. Otherwise, it outputs 0 to reject. In this section, we define a "protocol template" τ to precisely capture this interaction.

Recall that Σ-protocols must also satisfy the properties of completeness, special honest-verifier zero-knowledge, and special soundness. Therefore, the full definition of a Σ-protocol must not only include specifications of `Prove` and `Verify` algorithms, but also of the `Simulate` and `Extract` algorithms required by the zero-knowledge and soundness properties, respectively. Once the protocol template is defined, we introduce the Σ-protocol object, which uses the protocol template τ to specify the `Prove` and `Verify` algorithms and additionally includes specifications of the `Simulate` and `Extract` algorithms.

Formally, let R be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call x the statement and w the witness. We consider two-party protocols between a prover P and a verifier V, both efficient ITMs, of the general form defined by Damgård [24]. Damgård's original version is in Appendix A.2, and we present our algorithmic version below.

Definition 1 (Protocol Template). *The* protocol template *for a relation R is a tuple of efficient algorithms $\tau =$ (`Setup`, `Commit`, `Challenge`, `Respond`, `Verdict`), 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.*
- $\texttt{com} \leftarrow \texttt{Commit}(x, w)$*: P sends V a message* `com`*.*
- $\texttt{chl} \leftarrow \texttt{Challenge}(x, \texttt{com})$*: V sends P a random ℓ-bit string* `chl`*.*
- $\texttt{res} \leftarrow \texttt{Respond}(x, w, \texttt{com}, \texttt{chl})$*: P sends V a reply* `res`*.*
- $\{0, 1\} \leftarrow \texttt{Verdict}(x, \texttt{com}, \texttt{chl}, \texttt{res})$*: V decides to accept (output 1) or reject (output 0) based on the input $(x,$* `com`*,* `chl`*,* `res`*).*

The tuple (`com`, `chl`, `res`) *is called a* transcript*. A transcript is* accepting *for x if* $\texttt{Verdict}(x, \texttt{com}, \texttt{chl}, \texttt{res})$ *outputs 1.*

A Σ-protocol is a protocol of the form in Definition 1 with the completeness, special honest-verifier zero-knowledge, and special soundness properties. Since we will later introduce a transform for Σ-protocols to make them GUC-secure, it will be useful to think of Σ-protocols as well-defined objects that can be passed as input to a transform. We therefore create the object $\Sigma_R =$ (`Setup`, `Prove`, `Verify`, `Simulate`, `Extract`) that is the precise algorithmic specification of a Σ-protocol over a relation R. When an algorithm or ITM gets the object Σ_R as input, it gains access to the full specification of each algorithm in Σ_R, defined as follows.

Note that in the following definition, `Prove` is a two-party protocol between a prover and a verifier, where the first input to the algorithm is the prover's input, and the second input is the verifier's. After running `Prove`, which consists of the template protocol algorithms `Commit`, `Challenge`, and `Respond`, both parties obtain a copy of the output proof π.

Definition 2 (Σ-protocol). *A* Σ-protocol *for a relation R based on template τ (see Definition 1) is a tuple of efficient procedures $\Sigma_{R,\tau} =$ (`Setup`, `Prove`, `Verify`, `Simulate`, `Extract`), where* `Prove` *is a two-party protocol between P and V and the rest are algorithms. The procedures are defined as follows.*

- $\texttt{ppm} \leftarrow \texttt{Setup}(1^\lambda)$*: Given a security parameter* 1^λ*, invoke* $\tau.\texttt{Setup}(1^\lambda)$ *to obtain* ppm*.*
- $\pi \leftarrow \texttt{Prove}((x, w), x)$*: Let the first (resp. second) argument to protocol* Prove *be the input of the prover (resp. verifier). P and V run* $\tau.\texttt{Commit}$, $\tau.\texttt{Challenge}$*, and* $\tau.\texttt{Respond}$*. We use* π *to represent the transcript* (com, chl, res)*.*
- $\{0, 1\} \leftarrow \texttt{Verify}(x, \pi)$*: V runs* $\tau.\texttt{Verdict}$ *on input* $(x, \texttt{com}, \texttt{chl}, \texttt{res})$*. The* completeness property *requires that if* $(x, w) \in R$ *and* $\pi \leftarrow \texttt{Prove}((x, w), x)$*,* $\texttt{Verify}(x, \pi) = 1$*.*
- $\pi \leftarrow \texttt{Simulate}(x, \texttt{chl})$: *For every* x *and* w *such that* $(x, w) \in R$ *and every* $\texttt{chl} \in \{0, 1\}^\ell$*, the* special honest-verifier zero-knowledge property *requires that for all* $\pi' \leftarrow \texttt{Simulate}(x, \texttt{chl})$ *and* $\pi \leftarrow \texttt{Prove}((x, w), x)$ *where V's random tape is* chl*,* $\{\pi'\} \approx_s \{\pi\}$*, where* $\{\pi'\}$ *(resp.* $\{\pi\}$*) is the distribution of proofs made by the simulator (resp. prover).*
- $w \leftarrow \texttt{Extract}(x, \texttt{com}, \texttt{chl}, \texttt{chl'}, \texttt{res}, \texttt{res'})$: *The* special soundness property *requires that as long as* $\texttt{chl} \neq \texttt{chl'}$*,* $\tau.\texttt{Verdict}(x, \texttt{com}, \texttt{chl}, \texttt{res}) = 1$*, and* $\tau.\texttt{Verdict}(x, \texttt{com}, \texttt{chl'}, \texttt{res'}) = 1$*,* Extract *outputs* w *such that* $(x, w) \in R$*.*

For convenience, we drop τ *from the notation and use* Σ_R *to represent* $\Sigma_{R,\tau}$*.*

The non-interactive version of a Σ-protocol is almost the same as a regular Σ-protocol, except Prove is an algorithm executed solely by the prover. The first non-interactive transform for Σ-protocols was given by Fiat and Shamir [27] and proven secure in the ROM by Pointcheval and Stern [40]. Fischlin's transform [28] uses the RO differently, but draws on the work of Fiat and Shamir. We present a version of the Fiat-Shamir transform for our model in Appendix A.4.

Significantly, non-interactivity in the global ROM makes the proof tuple (x, π) *universally verifiable*. Following Canetti et al., we note that the universal verifiability property makes all non-interactive proofs in the global ROM inherently *transferable* [19]. The special soundness property, however, still guarantees that a cheating prover produce a *fresh* proof of a statement.

2.2 OR-protocols

In this section, we introduce the interactive version of a Σ-protocol that we will need for our construction, called an OR-protocol [22, 24]. Rather than producing a proof corresponding to a single statement x, the prover in an OR-protocol proves that it knows a witness for *either* one statement x_0 *or* another statement x_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. The original idea, introduced as "proofs of partial knowledge" by Cramer et al. [22] and again formalized for Σ-protocols by Damgård [24], works as follows.

Given both statements x_0, x_1 and a witness w_b for one of the statements x_b, the prover first samples a random challenge $\texttt{chl}_{1-b}$ to correspond to the statement for which it does not have a witness, x_{1-b}. It then invokes the Simulate

algorithm on input $(x_{1-b}, \texttt{chl}_{1-b})$ to obtain the entire simulated proof transcript $(\texttt{com}_{1-b}, \texttt{chl}_{1-b}, \texttt{res}_{1-b})$. The prover then forms the first message commitment $\texttt{com}_b$ for x_b honestly according to the `Commit` algorithm, and sends the tuple $(\texttt{com}_0, \texttt{com}_1)$ 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 $\texttt{chl}_b = \texttt{chl}_{1-b} \oplus \texttt{CHL}$. Note this step "fixes" the challenge $\texttt{chl}_b$ such that the prover cannot cheat and simulate the proof of both statements. Given $\texttt{chl}_b$, the prover can compute $\texttt{res}_b$ according to the `Respond` algorithm. Finally, the prover sends both transcripts $(\texttt{com}_0, \texttt{chl}_0, \texttt{res}_0)$ and $(\texttt{com}_1, \texttt{chl}_1, \texttt{res}_1)$ to the verifier, who checks that both are transcripts are valid and also that $\texttt{chl}_0 \oplus \texttt{chl}_1 = \texttt{CHL}$.

As is, the definition of Damgård (provided in Appendix A.3) 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 a fully modular design, 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 `Simulate` 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 and avoid 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, `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 $W = (w, 0)$. Otherwise if $(x_1, w) \in R_1$, then $b = 1$ and the tuple $W = (w, 1)$. 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 `Extract` operation must be legitimate—either $R_0(x_0, w_0) = 1$ or $R_1(x_1, w_1) = 1$. The formal specification is as follows.

Definition 3 (OR-Protocol). *An* OR-protocol *for a relation $R_{OR} = R_0 \vee R_1$ based on Σ-protocols Σ_{R_0} and Σ_{R_1} (see Definition 2) is a tuple of procedures $\Sigma_{OR} =$ (`Setup`, `Prove`, `Verify`, `Simulate`, `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 $\texttt{ppm}_1$. Return $\texttt{ppm} = (\texttt{ppm}_0, \texttt{ppm}_1)$.*
- $\Pi \leftarrow \texttt{Prove}(X, W)$*: 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* $\tau_{R_b}.\texttt{Commit}(x_b, w)$*. 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* 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* $\tau_{R_b}.\texttt{Respond}(x_b, w, \texttt{com}_b, \texttt{chl}_b)$*. P sends* (Chl, 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:*

- $\{0,1\} \leftarrow \texttt{Verdict}(X, \texttt{Com}, \texttt{Chl}, \texttt{Res})$*: If* $\tau_{R_0}.\texttt{Verdict}(x_0, \texttt{com}_0, \texttt{chl}_0, \texttt{res}_0) = 1$ *and* $\tau_{R_1}.\texttt{Verdict}(x_1, \texttt{com}_1, \texttt{chl}_1, \texttt{res}_1) = 1$*, return 1 (accept). Otherwise, return 0 (reject).*

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

– $\Pi \leftarrow \texttt{Simulate}(X, \texttt{CHL})$: *Parse* $X = (x_0, x_1)$*. 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* $\Pi = (\pi_0, \pi_1, \texttt{CHL})$*.*

– $W \leftarrow \texttt{Extract}(X, \texttt{Com}, \texttt{Chl}, \texttt{Chl'}, \texttt{Res}, \texttt{Res'}, \texttt{CHL}, \texttt{CHL'})$*: Given two "accepting transcripts"* (Com, Chl, Res, CHL) *and* (Com, Chl', Res', CHL')*, parse X as* (x_0, x_1)*,* Com *as* $(\texttt{com}_0, \texttt{com}_1)$*, ..., and* Res′ *as* $(\texttt{res}'_0, \texttt{res}'_1)$*. Obtain* w_0 *by running* $\Sigma_{R_0}.\texttt{Extract}(x_0, \texttt{com}_0, \texttt{chl}_0, \texttt{chl}'_0, \texttt{res}_0, \texttt{res}'_0)$ *and* w_1 *by running* $\Sigma_{R_1}.\texttt{Extract}(x_1, \texttt{com}_1, \texttt{chl}_1, \texttt{chl}'_1, \texttt{res}_1, \texttt{res}'_1)$*. Return* $W = (w_0, w_1)$*.*

Theorem 1. *Given Σ-protocols Σ_{R_0} for a relation R_0 and Σ_{R_1} for relation R_1, the protocol Σ_{OR} 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 [24]. □

The independence of the bit b creates a property called *witness indistinguishability*, which makes it impossible for the verifier to tell which proof was computed "honestly" (with the witness w_b), and which proof was simulated. As we will see in the next section, this property will allow us to build a Simulate algorithm for the non-programmable $\mathcal{G}_{RO}$-CRS hybrid model.

2.3 $\mathcal{G}_{RO}$ Revisited

Building on the overview of Canetti et al.'s global RO $\mathcal{G}_{RO}$ [19] given in the introduction, we now present the formal details of its functionality.

As with traditional ROs, $\mathcal{G}_{RO}$ responds to each query string $x \in \{0,1\}^*$ with a consistent, uniformly random, and fixed-length string $v \in \{0,1\}^\ell$. We call

this original query algorithm `RandQuery`, since the caller is querying the oracle for *randomness*. In order to keep track of the "illegitimate" queries made by the environment, all queries must also include the SID for which the query was issued. $\mathcal{G}_{RO}$ then checks to see if the querying party's SID matches the SID in the query. If it doesn't, $\mathcal{G}_{RO}$ records the query in the special list of illegitimate queries for each SID s, denoted $\mathcal{Q}_s$. If at any time an ideal functionality $\mathcal{F}$ wants to see the list of illegitimate queries for s, $\mathcal{G}_{RO}$ checks whether $\mathcal{F}$ is indeed the correct functionality for s. If it is, $\mathcal{G}_{RO}$ returns the list $\mathcal{Q}_s$. Otherwise it returns $\perp$. We call this type of query algorithm `IllQuery`, since the caller is querying the oracle for *illegitimate queries*.

Our definition is notationally different but functionally identical to Canetti et al.'s [19], which is available in Appendix A.8. Recall that in order to avoid messy notation, the query to $\mathcal{G}_{RO}$ is assumed to include the querying party's PID and SID. $\mathcal{G}_{RO}$ is also assumed to maintain state, which in this case includes the output length ℓ, ideal functionality list $\overline{\mathcal{F}}$, query list $\mathcal{Q}$, and illegitimate query lists $\mathcal{Q}_{\texttt{sid}}$ for all `sid`.

Definition 4 (Global Random Oracle). *The* global random oracle (global RO) $\mathcal{G}_{RO}$ *is a tuple of algorithms* (`Setup`, `RandQuery`, `IllQuery`) *defined as follows.*

- $\perp \leftarrow \texttt{Setup}(1^\ell, \overline{\mathcal{F}})$: *Set the output length of* `RandQuery` *to be* ℓ*. Store the list of ideal functionalities* $\overline{\mathcal{F}}$*.*
- $v \leftarrow \texttt{RandQuery}(x)$: *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*.*
- $\mathcal{Q}_s \leftarrow \texttt{IllQuery}(s)$: *If the caller's PID is in the list of ideal functionalities* $\overline{\mathcal{F}}$ *and its SID* $= s$*, return* $\mathcal{Q}_s$*. Otherwise, output* $\perp$*.*

For convenience, the default query $\mathcal{G}_{RO}(\cdot)$ *to* $\mathcal{G}_{RO}$ *is shorthand for* $\texttt{RandQuery}(\cdot)$*.*

3 Construction

Our construction takes any interactive Σ-protocol and transforms it into a non-interactive GUC-secure Σ-protocol. Because of the impossibility of proving the GUC-security of non-interactive Σ-protocols using *only* a global setup [38, 18, 19], we prove the security of our construction in a hybrid model that combines the global RO functionality with a local CRS functionality. In short, we use the local CRS to give our simulator the minimal "special power" needed to simulate NIZK PoK in the global ROM. Importantly, special knowledge about the CRS is restricted to the simulator—all of the "real world" parties executing the protocol prove statements using real witnesses.

The organization of this section is as follows. First, we specify the NIZK PoK ideal functionality, which captures the special honest-verifier zero-knowledge and special soundness properties of Σ-protocols. Next, we introduce the $\mathcal{G}_{RO}$-CRS

hybrid model, which is a combination of the global RO and local CRS functionalities discussed above. Finally, we compose all of the building blocks from Section 2 to obtain a straight-line extractable OR-protocol, which we use as the basis for our GUC-transform. In the next section, we will prove that our construction GUC-realizes the NIZK PoK functionality in the $\mathcal{G}_{RO}$-CRS hybrid model.

3.1 NIZK PoK Ideal Functionality

In the "ideal" world, honest parties executing the Σ-protocol are "dummy parties" who do not perform any computations of their own. Instead, they pass all of their inputs to an ideal functionality $\mathcal{F}_{\texttt{NZP}}$, who instructs them on how to respond. As is standard in the UC framework [17], there is one ideal functionality for each SID s. A dummy party with SID s can only send input and receive output from the $\mathcal{F}_{\texttt{NZP}}$ with the same SID, denoted $\mathcal{F}^s_{\texttt{NZP}}$.

In our setting, each $\mathcal{F}^s_{\texttt{NZP}}$ will need to process proofs and verifications of the Σ-protocols from Definition 2. Recall that the proofs must be *zero-knowledge* (satisfying the special honest-verifier zero-knowledge property) and also *proofs of knowledge* (satisfying the special soundness property). These properties require the construction of a special honest-verifier zero-knowledge simulator algorithm `Simulate` and a special soundness extractor algorithm `Extract`, respectively.

Note that in the following definition, there are two conditions in which $\mathcal{F}_{\texttt{NZP}}$ can output `Fail`. The first is if the `Simulate` algorithm fails to produce a proof π of a statement x such that $\texttt{Verify}(x, \pi) = 1$. The second is if the `Extract` algorithm fails to produce a witness w from a non-simulated proof π of a statement x such that $R(x, w) = 1$. In our security proof in the next section, we will need to argue that for our construction, the probability of these failures is negligible in the security parameter. $\mathcal{F}^s_{\texttt{NZP}}$ works as follows.

Definition 5 (The NIZK PoK Ideal Functionality). *The ideal functionality $\mathcal{F}^s_{NZP}$ 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 $\mathcal{F}_{NZP}$ will use. $\mathcal{F}_{NZP}$ 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`*.*

3.2 The $\mathcal{G}_{RO}$-CRS Hybrid Model

In order to enable $\mathcal{F}_{\texttt{NZP}}$ to simulate proofs of statements without real witnesses, it will need some minimal "special power" that a real prover does not have. Because $\mathcal{G}_{RO}$ is not programmable, the traditional "special power" of the simulator in the RO model—getting the challenge in advance then programming the RO to output that challenge for the desired commitment—will not work. Instead, we will give our simulator the trapdoor to a local CRS, and have provers prove that either they know a "real" witness to the statement x_0, *or* they know a trapdoor to the CRS. Since our construction uses an OR-protocol, we must be able to interpret the CRS as a statement x with a corresponding trapdoor witness w such that the pair (x, w) satisfies some binary $\mathcal{NP}$ relation S.

In the "real" world, there is no CRS trapdoor—only a CRS "ideal functionality" for SID s that returns the $\texttt{CRS}_s$ to any querying party with SID s. In practice, parties might use secure multi-party computation to generate a CRS for each session in a way that guarantees the CRS is truly random and that no party can compute a trapdoor. As mentioned in the introduction, Canetti et al. [19] showed this can be done in the $\mathcal{G}_{RO}$-hybrid model without trusted setup.

We first present the ideal CRS functionality for an SID s, which relies on a generic "GenCRS" algorithm, and then we discuss the properties that GenCRS must have in order to satisfy the requirements of our construction.

Definition 6 (The CRS Ideal Functionality). *The ideal functionality $\mathcal{F}_{CRS}^{s}$ of a* common reference string (CRS) *for a particular SID s and 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* $(\texttt{CRS}, s, x)$*.*

In order to produce a CRS that can function as a statement in an OR-protocol, the GenCRS algorithm in our construction must generate the CRS as a statement x in a language over some relation S, with a corresponding witness w such that $S(x, w) = 1$. Efficiency dictates that the GenCRS algorithm must be able to efficiently generate the CRS and trapdoor (or statement-witness) pair, while security dictates that the trapdoor (witness) should not be efficiently computable from the CRS (statement). 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 [28].

Definition 7 (Samplable Hard Relation). *A binary $\mathcal{NP}$ relation S is* samplable *and* 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 that can be used as part of the OR-protocol.[1] We call such a relation a Σ-friendly relation.

Definition 8 (Σ-Friendly Relation). *A* Σ-friendly relation S *is a binary* $\mathcal{NP}$ *relation with a corresponding* efficient Σ*-protocol* Σ_S*.*

Therefore, `GenCRS` for our construction consists of sampling (x, w) using $\kappa_S(1^\lambda)$, where S is a samplable hard, Σ-friendly relation. We combine the local CRS functionality $\mathcal{F}_{\texttt{CRS}}$ based on the above `GenCRS` mechanism with the global RO functionality $\mathcal{G}_{RO}$ to create the $\mathcal{G}_{RO}$-CRS hybrid model.

Definition 9 (GUC-security in the $\mathcal{G}_{RO}$-CRS Hybrid Model). *A protocol Σ with security parameter λ* GUC-realizes *an ideal functionality $\mathcal{F}$ in the GRO-CRS hybrid model if for all efficient $\mathscr{A}$, there exists an ideal adversary $\mathcal{S}$ efficient in expectation such that for all efficient environments $\mathcal{Z}$,*

$$\text{IDEAL}^{\mathcal{G}_{RO}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda, \texttt{aux}) \approx_c \text{REAL}^{\mathcal{G}_{RO},\mathcal{F}_{\texttt{CRS}}}_{\Sigma,\mathscr{A},\mathcal{Z}}(1^\lambda, \texttt{aux}),$$

where $\mathcal{G}_{RO}$ is the global RO functionality from Definition 4, $\mathcal{F}_{\texttt{CRS}}$ is the local CRS functionality from Definition 6, and `aux` *is any auxiliary information provided to the environment.*

3.3 Straight-Line Extractable OR-protocol

The core of our construction is a straight-line extractable OR-protocol in which a prover demonstrates using the Fischlin proof technique [28] that it either knows a witness to a statement, or else it knows the trapdoor to the CRS for its SID. Before we can apply the Fischlin transform, we need a random oracle that maps to b bits. Since $\mathcal{G}_{RO}$ 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 [28] that maps the output of $\mathcal{G}_{RO}$ to b bits by cutting off all but b bits of the output.

Definition 10 (Bit Truncation Function). *The $\mathcal{G}_{RO}$* bit truncation function $\texttt{trunc} : \{0,1\}^\ell \to \{0,1\}^b$ *maps the ℓ-bit output of $\mathcal{G}_{RO}$ to a b-bit output by cutting off the $\ell - b$ leading bits.*

[1] While all relations are guaranteed to have a corresponding Σ-protocol, they are not necessarily guaranteed to have an *efficient* one.

The RO functionality in the straight-line extractable OR-protocol is therefore a modification of the Fischlin transform where the RO $H : \{0,1\}^* \to \{0,1\}^b$ is replaced by $\texttt{trunc}(\mathcal{G}_{RO}) : \{0,1\}^* \to \{0,1\}^b$. We apply the modified $\mathcal{G}_{RO}$-Fischlin transform to the interactive OR-protocol in Definition 3 in order to make it non-interactive and straight-line extractable. In the following section, we will instantiate the straight-line extractable OR-protocol using the Σ-protocols Σ_R and Σ_S, where Σ_R is any Σ-protocol over any relation R, and Σ_S is the Σ-protocol corresponding to the samplable hard, Σ-friendly relation S. We will demonstrate how to use this OR-protocol in order to achieve a transform that makes any Σ-protocol Σ_R GUC-secure in the global hybrid model.

The straight-line extractable OR-protocol works as follows. The public parameters `ppm` set during the `Setup` algorithm must now include the public parameters of the underlying OR-protocol Σ_{OR} *and* the parameters b, r, S, t described in the Fischlin transformation [28], as well as the $\mathcal{G}_{RO}$ bit truncation function `trunc`. The `Prove` algorithm uses the Fischlin technique with Σ_{OR} as the underlying Σ-protocol. We again use capital letters for the statements, commitments, challenges, responses, and proofs to denote they are really a *pair* of each. The overall `Prove` process is the same as the process described in Section 1.4 on the Fischlin transform: the prover generates a vector of commitments, tests challenges until it finds one that causes the RO to return an all-zero (or else minimal) bit string, and returns a vector of proofs with only one minimal-return transcript included for each commitment. Similarly, the `Verify` algorithm parses and verifies each proof and checks to make sure the sum of the returns from the RO is less than some maximal value. Since this is an OR-protocol, the verifier must also check to make sure $\texttt{CHL}_i = \texttt{chl}_{i0} \oplus \texttt{chl}_{i1}$ for $i = 1, \ldots, r$.

In order to simulate proofs, the traditional Fischlin construction relies on the special honest-verifier zero-knowledge simulator of the underlying Σ-protocol. For each of the r proofs, the `Simulate` algorithm samples 2^t random b-bit strings and assigns them to the t-bit challenges $0, 1, \ldots, 2^{t-1}$. It then picks the challenge $\texttt{CHL}_i$ that maps to the minimal string and runs the simulator $\Sigma_{OR}.\texttt{Simulate}$ on input $(X, \texttt{CHL})$ to obtain the proof Π. Once it has completed this process for all r proofs, it take the commitment vector $\overline{\texttt{Com}}$ and programs $\mathcal{G}_{RO}$ at each input $(X, \overline{\texttt{Com}}, i, \texttt{Chl}_i, \texttt{Res}_i, \texttt{CHL}_i)$ to output a random string ending in the b minimal bits. Note that since $\mathcal{G}_{RO}$ is not programmable, we will not be able to use Fischlin's simulator in our construction. We will, however, demonstrate that our simulator is indistinguishable from the simulator that *can* program $\mathcal{G}_{RO}$.

Finally, the Fischlin extractor relies on a list $\mathcal{Q}_{P^*}$ of adversarial provers' queries to $\mathcal{G}_{RO}$. In our definition of the straight-line extractable OR-protocol, we give this list as input to the `Extract` algorithm, and will later demonstrate in the definition of our GUC-transform how exactly the adversary's query list is generated in the GUC model. Given the list and a proof from which to extract, the algorithm searches the list for two queries with matching commitment vectors and i values, but different challenges. If it finds two such queries, it runs the underlying extractor algorithm $\Sigma_{OR}.\texttt{Extract}$ on the transcripts. Otherwise, if it cannot find enough information to run the OR-protocol extractor, the Fischlin extractor outputs `Fail`.

Fischlin demonstrates that the success of the simulator is directly proportional to the success of the underlying Σ-protocol simulator, and that with sufficiently-sized parameters b, r, S, and t the extractor will fail with negligible probability [28]. We use Fischlin's results as a key part of our proof in the next section.

Definition 11 (The Straight-Line Extractable OR-protocol). *Let Σ_{OR} be an interactive OR-protocol with challenge length $\ell = O(\log \lambda)$ bits. Then the* straight-line extractable OR-protocol *Σ^*_{OR} is a non-interactive, straight-line extractable transform of Σ_{OR}, defined as follows.*

- $\mathtt{ppm} \leftarrow \mathtt{Setup}(1^\lambda)$: *Let b, r, S, t be set according to the Fischlin transform (see Definition 17 in Appendix A.5 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 $\Sigma_{OR}.\mathtt{Setup}(1^\lambda)$ and $\mathtt{trunc}$ is the bit truncation function from Definition 10.*
- $(X, \Pi) \leftarrow \mathtt{Prove}(X, W)$: *To compute the vector of r commitments $\overline{\mathtt{Com}} = \langle \mathtt{Com}_0, \mathtt{Com}_1, \ldots, \mathtt{Com}_r \rangle$, the prover repeats the commitment step of $\Sigma_{OR}.\mathtt{Prove}(X, W)$ r times. Recall that as part of the OR-protocol commitment step, the prover generates $\mathtt{chl}_{1-b}$ at random. To compute each response $\mathtt{Res}_i$, the prover tests t-bit challenges $\mathtt{CHL}_i$ as follows. First, it sets $\mathtt{chl}_b = \mathtt{chl}_{1-b} \oplus \mathtt{CHL}$. Then, it repeats the response step of $\Sigma_{OR}.\mathtt{Prove}(X, W)$ until it finds one such that $\mathtt{trunc}(\mathcal{G}_{RO}(X, \overline{\mathtt{Com}}, i, \mathtt{Chl}_i, \mathtt{Res}_i, \mathtt{CHL}_i)) = 0^b$, 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 $\Pi_i = (\pi_{0i}, \pi_{1i}, \mathtt{CHL}_i)$ for $1 \leq i \leq r$.*
- $\{0, 1\} \leftarrow \mathtt{Verify}(X, \Pi)$: *Parse $\Pi = (\Pi_1, \ldots, \Pi_r)$. The verifier outputs 1 (accepts) if and only if $\Sigma_{OR}.\mathtt{Verify}(X, \Pi_i) = 1$ and $\mathtt{CHL}_i = \mathtt{chl}_{i0} \oplus \mathtt{chl}_{i1}$ for all $1 \leq i \leq r$, and $\sum_{i=1}^{r} \mathtt{trunc}(\mathcal{G}_{RO}(X, \overline{\mathtt{Com}}, i, \mathtt{Chl}_i, \mathtt{Res}_i, \mathtt{CHL}_i)) \leq S$. Otherwise, the verifier outputs 0 (rejects).*
- $(X, \Pi) \leftarrow \mathtt{Simulate}(X)$: *For each proof $1 \leq i \leq r$, sample 2^t random b-bit strings and assign them to the t-bit challenges $\mathtt{CHL}_i \in \{0, 1\}^t$. Let $\mu : \{0, 1\}^t \rightarrow \{0, 1\}^b$ represent the map between the challenges and the b-bit outputs, which are potential outputs of the RO. 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 $\Sigma_{OR}.\mathtt{Simulate}(X, \mathtt{CHL})$ 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 $\mathcal{G}_{RO}$ on input $(X, \overline{\mathtt{Com}}, i, \mathtt{Chl}_i, \mathtt{Res}_i, \mathtt{CHL}_i)$ 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 $\Pi = (\Pi_1, \ldots, \Pi_r)$.*
- $W \leftarrow \mathtt{Extract}(X, \Pi, \mathcal{Q}_{P^*})$: *Parse $\Pi = (\Pi_1, \ldots, \Pi_r)$ and each $\Pi_i = (\overline{\mathtt{Com}}, \mathtt{Chl}_i, \mathtt{Res}_i, \mathtt{CHL}_i)$. Given a list $\mathcal{Q}_{P^*}$ of adversarial provers' queries to $\mathcal{G}_{RO}$, the extractor searches for two queries $(X, \overline{\mathtt{Com}}, i, \mathtt{Chl}_i, \mathtt{Res}_i, \mathtt{CHL}_i)$ and $(X, \overline{\mathtt{Com}}, i, \mathtt{Chl}'_i, \mathtt{Res}'_i, \mathtt{CHL}'_i)$ such that $\mathtt{CHL}_i \neq \mathtt{CHL}'_i$, $\Sigma_{OR}.\mathtt{Verify}(X, (\mathtt{Com}_i, \mathtt{Chl}_i, \mathtt{Res}_i, \mathtt{CHL}_i)) = 1$, and $\Sigma_{OR}.\mathtt{Verify}(X, (\mathtt{Com}_i, \mathtt{Chl}'_i, \mathtt{Res}'_i, \mathtt{CHL}'_i)) = 1$. If no two queries exist, output $\mathtt{Fail}$. Otherwise, obtain W by running $\Sigma_{OR}.\mathtt{Extract}(X,$ Com, Chl, Chl', Res, Res', CHL, CHL'$)$. If there exists a witness $w \in W$ such*

that $R(x, w) = 1$*, outputs* W*. Otherwise, if* `Extract` *outputs* `Fail` *or if* $R(x, w) = 0$ *for both* $w \in W$*, output* `Fail`*.*

Theorem 2. *Let* Σ_{OR} *be a* Σ*-protocol for relation* $R_{OR} = R_0 \vee R_1$*. Then the protocol* Σ^*_{OR} *is a non-interactive, straight-line extractable* Σ*-protocol with the stronger property of special* simulation *soundness for relation* R_{OR} *in the random oracle model.*

Proof. We refer the reader to Fischlin's proof [28]. □

Finally, we are ready to introduce construction of our main result, the GUC-transform for Σ-protocols.

3.4 Non-Interactive GUC-Secure Σ-protocols

We propose a transform that makes any Σ-protocol Σ_R non-interactive and GUC-secure in the $\mathcal{G}_{RO}$-CRS hybrid model. The transform works as follows.

First, the `Setup` functionality picks a samplable hard, Σ-friendly relation S and defines a new Σ-protocol Σ_{OR} over the relation $R_{OR} = R \vee S$ using the specifications of Σ_R and Σ_S. This OR-protocol is used as input to the straight-line extractable OR-protocol transform given in the previous definition. The public parameters of our system are the same as the public parameters from the straight-line extractable OR-protocol, obtained by running $\Sigma^*_{OR}.\texttt{Setup}(1^\lambda)$.

For each SID s, provers query the CRS ideal functionality $\mathcal{F}^s_{\texttt{CRS}}$ to obtain $\texttt{CRS}_s$. Each time a prover with SID s needs to create a proof of a statement x, it generates a proof using the straight-line extractable OR-protocol Σ^*_{OR} that either they know a witness w for the statement x, or they know the trapdoor $\texttt{trap}_s$ to $\texttt{CRS}_s$. Since no real prover will ever have the CRS trapdoor, it must prove knowledge of the witness w. In other words, the compound statement to be proven is $X = (x, \texttt{CRS}_s)$, and the compound witness $W = (w, 0)$ denotes a "genuine" proof of the statement x using the corresponding witness w. The prover then uses $\Sigma^*_{OR}.\texttt{Prove}(X, W)$ to obtain the proof (X, Π), and returns it.

In order to verify the proof, a verifier checks whether $\Sigma^*_{OR}.\texttt{Verify}(X, \Pi) = 1$. Recall that from the witness indistinguishability property of OR-protocols (see Theorem 1), a verifier cannot tell from the proof whether the prover used a real witness or the CRS trapdoor. This property will be important to the proof of GUC-security, since our `Simulate` algorithm uses the CRS trapdoor rather than a real witness in all of its proofs.

In particular, rather than use the original `Simulate` algorithm to simulate proofs (which would require the global random oracle to be programmable), the simulator in our construction works as follows. The first time the simulator is invoked for an SID s, it generates $(\texttt{CRS}_s, \texttt{trap}_s)$ itself using the sampling algorithm κ_S on input 1^λ. It then sets $X = (x, \texttt{CRS}_s)$ as before, but rather than use a real witness, it sets $W = (\texttt{trap}_s, 1)$, denoting a "genuine" proof of the statement $\texttt{CRS}_s$. The simulator then uses $\Sigma_{OR}.\texttt{Prove}(X, W)$ to obtain the proof (X, Π), and returns it.

Given a proof (X, Π), the `Extract` algorithm first prepares the list of adversarial queries to $\mathcal{G}_{RO}$ as follows. It obtains the corrupted parties' queries by

querying the ideal adversary $\mathcal{S}$, who gets to observe the corrupted parties' communications. It obtains the environment's queries by issuing an illegitimate list query to $\mathcal{G}_{RO}$ for SID s. Finally, it takes the combined list of adversarial proof queries, denoted $\mathcal{Q}^s_{P^*}$, and runs $\Sigma^*_{OR}.\texttt{Extract}(X, \Pi, \mathcal{Q}^s_{P^*})$.

Note that while it is possible for $\Sigma^*_{OR}.\texttt{Extract}$ to output `Fail`, the probability is negligible by Fischlin's result (see Theorem 2). This result will also be important to the proof of GUC-security, since a verification process that outputs `Fail` would instantly tip off $\mathcal{Z}$ that it is living in the ideal world. As we will see in the next section, the straight-line extraction process lends itself to security for proofs that are only computationally zero-knowledge, since $\mathcal{Z}$ can already see all of its own oracle queries and there is nothing new to learn from interacting with the extractor.

Our main result, the GUC-transform $\Sigma^{\texttt{uc}}_R$ for any Σ-protocol Σ_R, is formalized below.

Definition 12 (GUC-Transform for Σ_R). *Let Σ_R be any Σ-protocol. Our transform $\Sigma^{\texttt{uc}}_R$ is a non-interactive, straight-line extractable transform of Σ_R, defined as follows.*

- $\texttt{ppm} \leftarrow \texttt{Setup}(1^\lambda)$*: Choose a samplable hard, Σ-friendly relation S with corresponding efficient Σ-protocol Σ_S. Let the compound relation $R_{OR} = R \vee S$, and let Σ_{OR} be the OR-protocol based on Σ_R and Σ_S. Generate* `ppm` *by running* $\Sigma^*_{OR}.\texttt{Setup}(1^\lambda)$.
- $(X, \Pi) \leftarrow \texttt{Prove}(x, w)$*: Let the prover have identity $P = (\texttt{pid}, \texttt{sid})$. If $R(x, w) \neq 1$, return $\perp$. Otherwise, if this is P's first execution of* `Prove`*, obtain the CRS* $\texttt{CRS}_{\texttt{sid}}$ *by calling* $\mathcal{F}_{CRS}.\texttt{Query}$*. The compound statement X to be proven is the logical OR of x and* $\texttt{CRS}_{\texttt{sid}}$*, or $X = (x, \texttt{CRS}_{\texttt{sid}})$. The compound witness W is a combination of the bit 0, denoting a "genuine" proof of statement x, and the witness w corresponding to x, or $W = (w, 0)$. Obtain (X, Π) by running $\Sigma^*_{OR}.\texttt{Prove}(X, W)$ and return it.*
- $\{0, 1\} \leftarrow \texttt{Verify}(X, \Pi)$*: The verifier outputs 1 (accept) if and only if $\Sigma^*_{OR}.\texttt{Verify}(X, \Pi) = 1$. Otherwise, it outputs 0 (reject).*
- $(X, \Pi) \leftarrow \texttt{Simulate}(x)$ *: If this is the first time* `Simulate` *is being called for a particular SID s, generate $(\texttt{CRS}_s, \texttt{trap}_s)$ using the sampling algorithm κ_S on input 1^λ and store it. The compound statement X to be proven is the same as in* `Prove`*, or $X = (x, \texttt{CRS}_s)$. The compound witness $W = (\texttt{trap}_s, 1)$ is now a combination of the bit 1, denoting a "genuine" proof of statement $\texttt{CRS}_s$, and the CRS trapdoor $\texttt{trap}_s$. Obtain (X, Π) by running $\Sigma^*_{OR}.\texttt{Prove}(X, W)$ and return it.*
- $W \leftarrow \texttt{Extract}(X, \Pi)$*: Parse $X = (x, \texttt{CRS}_s)$. First, obtain the adversary's queries $\mathcal{Q}^s_{\mathscr{A}}$ for SID s by querying $\mathcal{S}$, who gets to observe corrupted parties' communications, including their queries to $\mathcal{G}_{RO}$. Next, obtain the environment's queries $\mathcal{Q}^s_{\mathcal{Z}}$ for s by issuing an illegitimate list query $\texttt{IllQuery}(s)$ to $\mathcal{G}_{RO}$. Let the combined list of adversarial provers' queries for SID s be $\mathcal{Q}^s_{P^*} = \mathcal{Q}^{\mathscr{A}}_s \cup \mathcal{Q}^{\mathcal{Z}}_s$. Run $\Sigma^*_{OR}.\texttt{Extract}(X, \Pi, \mathcal{Q}^s_{P^*})$ and return the output.*

In the following section, we prove that our construction GUC-realizes the NIZK PoK functionality $\mathcal{F}_{\mathtt{NZP}}$ in the $\mathcal{G}_{RO}$-CRS hybrid model.

4 Security Analysis

We wish to show that our transform $\Sigma_R^{\mathtt{uc}}$ takes any Σ-protocol Σ_R and turns it into a generally universally composable (GUC-secure) Σ-protocol.

Theorem 3. *If Σ_R is a Σ-protocol, $\Sigma_R^{\mathtt{uc}}$ is a non-interactive Σ-protocol that GUC-realizes $\mathcal{F}_{NZP}$ in the $\mathcal{G}_{RO}$-CRS hybrid model.*

Proof. Recall that in order to show that $\Sigma_R^{\mathtt{uc}}$ GUC-realizes $\mathcal{F}_{\mathtt{NZP}}$ in the $\mathcal{G}_{RO}$-CRS hybrid model, we must satisfy Definition 9. In short, 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 $\mathcal{Z}$,

$$\text{IDEAL}^{\mathcal{G}_{RO}}_{\mathcal{F}_{\mathtt{NZP}},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathtt{aux}) \approx_c \text{REAL}^{\mathcal{G}_{RO},\mathcal{F}_{\mathtt{CRS}}}_{\Sigma,\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}).$$

We review the standard GUC model real- and ideal-world experiments [18] in Appendix A.7, noting that we are working in the *non-adaptive* corruption model in which $\mathcal{Z}$ must decide at the time of party invocation whether or not the party is corrupted. The construction of the ideal adversary $\mathcal{S}$ is as follows.

Construction of the Ideal Adversary. The ideal adversary (also known as the simulator) $\mathcal{S}$, works as follows. When the ideal functionality $\mathcal{F}_{\mathtt{NZP}}$ asks it for the specification of algorithms, $\mathcal{S}$ returns the algorithms in $\Sigma_R^{\mathtt{uc}}$. When $\mathcal{F}_{\mathtt{NZP}}$ asks it for the queries of adversarial provers for an SID s, $\mathcal{S}$ returns the corrupted parties' $\mathcal{G}_{RO}$ 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 proves statements using the trapdoor to the CRS 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 (including the subroutine calls to $\mathcal{F}_{\mathtt{CRS}}$) *and* $\mathcal{G}_{RO}$. 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 1 (REAL = Experiment A). *In the view of the environment, Experiment A is identical to the real world experiment. Formally,*

$$\text{REAL}^{\mathcal{G}_{RO},\mathcal{F}_{CRS}}_{\Sigma_R^{\mathtt{uc}},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}) = \text{Exp}\text{A}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}).$$

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" $\mathcal{G}_{RO}$, or whether it is interacting with a puppet master who simulates all of the parties, including $\mathcal{G}_{RO}$. We will leverage this identical view to show in the next experiment that the "traditional" simulator of the straight-line extractable OR-protocol Σ^*_{OR}, which uses a programmable RO, is statistically indistinguishable from the unconventional simulator in $\Sigma^{\mathtt{uc}}_R$, 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 $\mathcal{G}_{RO}$ 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 obtains the proof (X, Π) by running the simulator of the straight-line extractable OR-protocol, $\Sigma^*_{OR}.\mathtt{Simulate}(X)$. Finally, it checks that $\Sigma^{\mathtt{uc}}_R.\mathtt{Verify}(X, \Pi) = 1$. If it doesn't, $\mathcal{C}$ outputs `Fail`. Otherwise, it outputs (X, Π).

Lemma 2 (Experiment A $\approx_s$ Experiment B). *Experiment B is statistically indistinguishable from Experiment A. Formally,*

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

Proof. Note that in both experiments, the challenger is programming convincingly (pseudo)random random strings as the output of $\mathcal{G}_{RO}$. As long as the domain of $\mathcal{G}_{RO}$ is exponential in the security parameter (as is standard for random oracles) and $\mathcal{Z}$ is only allowed to make polynomially-many queries to $\mathcal{G}_{RO}$ throughout the duration of the experiment, $\mathcal{Z}$'s view of the random oracle in experiment A is statistically close to its view in experiment B. Moreover, simulated proofs are statistically indistinguishable from non-simulated proofs by the honest-verifier zero-knowledge property of Σ^*_{OR}. The only other potential difference between the environment's view in experiments A and B is that the challenger in experiment B can output `Fail`, while the challenger in experiment A never does. However, experiment B only fails if the Fischlin simulator fails. Since the security of the Fischlin simulator tightly reduces to the statistical zero-knowledge of the underlying Σ-protocol [28], the failure of the Fischlin simulator is statistically close to the failure of the simulator for the underlying Σ-protocol, which is negligible. Therefore, the distributions representing $\mathcal{Z}$'s view of experiment A and experiment B are statistically indistinguishable. □

Remark. We demonstrate below via a reduction that experiment A is still computationally indistinguishable from experiment B in the case that Σ_{OR} is only computationally honest-verifier zero-knowledge.

In the next experiment (experiment C), we replace the real-world verification mechanism with extraction. Given a proof (X, Π), the challenger $\mathcal{C}$ runs through

a series of checks identical to the checks performed by $\mathcal{F}_{\texttt{NZP}}$. If $\mathcal{C}$ did not previously simulate the proof and $\Sigma_R^{\texttt{uc}}.\texttt{Verify}(X, \Pi) = 1$, $\mathcal{C}$ runs $\Sigma_R^{\texttt{uc}}.\texttt{Extract}(X, \Pi)$ and outputs the result. Recall that this process can fail in a few different ways. First, the straight-line OR-protocol extractor $\Sigma_{OR}^*.\texttt{Extract}$ can fail to find enough query transcripts to run the OR-protocol's underlying extractor $\Sigma_{OR}.\texttt{Extract}$. Second, the witnesses $W = (w_0, w_1)$ that $\Sigma_{OR}^*.\texttt{Extract}$ produces might fail to satisfy any relation, such that both $R(x, w_0) = 0$ and $S(\texttt{CRS}_s, w_1) = 0$. Finally, it might be the case that just $S(\texttt{CRS}_s, w_1) = 1$, which means that an adversarial prover somehow got access to the simulator's CRS trapdoor $\texttt{trap}_s$.

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 functionality of $\mathcal{F}_{\texttt{NZP}}$, can be used to contradict 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 protocol on non-simulated (adversarial) proofs, the challenger $\mathcal{C}$ attempts to extract a witness as follows. For a proof (X, Π) that it did not simulate previously, $\mathcal{C}$ first checks whether $\Sigma_R^{\texttt{uc}}.\texttt{Verify}(X, \Pi) = 1$. If it doesn't, $\mathcal{C}$ simply outputs 0. Otherwise, $\mathcal{C}$ runs $\Sigma_R^{\texttt{uc}}.\texttt{Extract}(X, \Pi)$ to extract W. If $\texttt{Extract}$ outputs a witness w_0 such that $R(x, w_0) = 1$, $\mathcal{C}$ outputs 1. Otherwise, if $\texttt{Extract}$ outputs $\texttt{Fail}$ or if $R(x, w_0) = 0$, $\mathcal{C}$ outputs $\texttt{Fail}$.

Lemma 3 (Experiment B $\approx_c$ Experiment C). *Experiment C is computationally indistinguishable from Experiment B. Formally,*

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

Proof. Given an environment $\mathcal{Z}_B$ that can distinguish between experiment B and experiment C, we construct a reduction that contradicts either the results of Fischlin, the special soundness property of Σ_{OR}, or the hardness property of the samplable hard relation S used to construct the CRS. Note that the only difference between experiment B and experiment C is that experiment C sometimes outputs $\texttt{Fail}$ while verifying a proof, while Experiment C never does. Therefore, if $\mathcal{Z}_B$ is able to determine the difference between experiment B and experiment C with non-negligible probability, the output $\texttt{Fail}$ must occur with non-negligible probability.

Consider the circumstances in which experiment C outputs $\texttt{Fail}$. The first condition in which the experiment C challenger outputs $\texttt{Fail}$ is if the $\texttt{Extract}$ algorithm outputs $\texttt{Fail}$, which only happens if the $\Sigma_{OR}^*.\texttt{Extract}$ algorithm cannot find two transcripts with the same prefix $(\overline{\texttt{Com}}, i)$ but different challenges $\texttt{CHL}_i \neq \texttt{CHL}'_i$. Note that since $\mathcal{C}$ is in control of all of the wires in the experiment and $\mathcal{G}_{RO}$, it gets to see all of the adversarial provers' queries to $\mathcal{G}_{RO}$, enabling it to construct a full adversarial query set $\mathcal{Q}_{P^*}^s$ for each session s, and provide that list to $\Sigma_{OR}^*.\texttt{Extract}$ along with the proof (X, Π). Therefore, the case in which $\Sigma_{OR}^*.\texttt{Extract}(X, \Pi, \mathcal{Q}_{P^*}^s)$ fails contradicts Fischlin's result from Theorem 2, which guarantees that if $\Sigma_{OR}^*.\texttt{Verify}(X, \Pi) = 1$, the prover must have queried $\mathcal{G}_{RO}$ enough times to produce at least two transcripts that satisfy

the requirements of the extractor. So, with all but negligible probability, the challenger is guaranteed to extract *something* for the witness W.

The second `Fail` condition happens when the first witness w_0 that `Extract` produces does not satisfy $R(x, w_0) = 1$. If neither of the witnesses satisfy R *and* neither of the witnesses satisfy S, that is if $R(x, w_0) = 0$ and $S(\mathtt{CRS}_s, w_1) = 0$, then $\mathcal{C}$ can produce a proof (X, Π) for which $\Sigma_{OR}.\mathtt{Verify}(X, \Pi) = 1$ but for which extraction fails, contradicting the special soundness property of Σ_{OR}. Since this condition also guaranteed to happen with negligible probability, one of the witnesses must satisfy $S(\mathtt{CRS}, w) = 1$.

Finally if it is true that $S(\mathtt{CRS}, w) = 1$ for some $w \in W$, $\mathcal{C}$ can construct a reduction that is able to produce a witness w for x sampled according to the CRS sampling algorithm $\kappa_S(1^\lambda)$. The reduction sets $\mathtt{CRS}_s = x$ and proceeds as experiment C. If the adversary produces w such that $S(x, w) = 1$, then the reduction breaks the hardness property of the samplable hard relation S.

Therefore, the output `Fail`—and consequently the event that $\mathcal{Z}_B$ distinguishes between experiment B and experiment C with non-negligible advantage—must occur with negligible probability. □

Finally, we replace the simulated proof process from experiment B, which uses the traditional OR-protocol simulator $\Sigma^*_{OR}.\mathtt{Simulate}$, with the GUC-transform simulator $\Sigma^{\mathtt{uc}}_R.\mathtt{Simulate}$, 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 $\Sigma^{\mathtt{uc}}_R.\mathtt{Prove}$ algorithm, only this time with the witness $W = (\mathtt{trap}_s, 1)$ 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 $\mathcal{G}_{RO}$, the challenger computes proofs of statements x for the honest parties by running $\Sigma^{\mathtt{uc}}_{OR}.\mathtt{Simulate}(x)$. Recall that this process consists of generating the CRS and trapdoor pair $(\mathtt{CRS}_s, \mathtt{trap}_s)$ for each session s according to the sampling algorithm $\kappa_S(1^\lambda)$, then running $\Sigma^*_{OR}.\mathtt{Prove}(X, W)$ for $X = (x, \mathtt{CRS}_s)$ and $W = (\mathtt{trap}_s, 1)$.

Lemma 4 (Experiment C $\approx_s$ Experiment D). *Experiment D is statistically indistinguishable from Experiment C. Formally,*

$$\textsc{Exp}\mathrm{C}_{\mathcal{C},\mathcal{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}) \approx_s \textsc{Exp}\mathrm{D}_{\mathcal{C},\mathcal{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}).$$

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 $\Sigma_{OR}.\mathtt{Prove}(X, W)$ for $X = (x, \mathtt{CRS}_s)$ and $W = (\mathtt{trap}_s, 1)$. Moreover, $\mathcal{Z}$'s view of the CRS is the same as in experiment A, since we defined $\mathcal{F}_{\mathtt{CRS}}$ 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 identical to that used to prove experiment A $\approx_s$ experiment B. □

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 $\mathcal{F}_{\texttt{NZP}}$ for both the `Prove` and `Verify` procedures. Therefore, we can replace $\mathcal{C}$ with $\mathcal{F}_{\texttt{NZP}}$ and $\mathcal{S}$, who takes over keeping track of the corrupted parties' communications with $\mathcal{G}_{RO}$.

Lemma 5 (Experiment D = IDEAL). *In the view of the environment, Experiment D is identical to the ideal world experiment. Formally,*

$$\text{EXPD}_{\mathcal{C},\mathscr{A},\mathcal{Z}}(1^\lambda, \texttt{aux}) = \text{IDEAL}^{\mathcal{G}_{RO}}_{\mathcal{F}_{\texttt{NZP}},\mathcal{S},\mathcal{Z}}(1^\lambda, \texttt{aux}).$$

Proof. Note that in experiment D, the challenger $\mathcal{C}$ answers honest parties' `Prove` queries by running $\Sigma^{\texttt{uc}}_R.\texttt{Simulate}(x)$, and `Verify` queries by running $\Sigma^{\texttt{uc}}_R.\texttt{Extract}(X, \Pi)$, with the same surrounding checks and procedures. Therefore, we can replace $\mathcal{C}$ in experiment D with $\mathcal{F}_{\texttt{NZP}}$ 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 $\mathcal{F}_{\texttt{NZP}}$ through a private channel upon request. Furthermore, since $\mathcal{C}$ is no longer programming $\mathcal{G}_{RO}$ in order to simulate proofs in experiment D, the functionality of $\mathcal{G}_{RO}$ is identical in both experiments. Therefore, the environment's view of experiment D is identical to its view of the ideal world experiment. □

In demonstrating the indistinguishability of the environment's view of the real and ideal experiments, we have shown that our construction GUC-realizes $\mathcal{F}_{\texttt{NZP}}$ in the $\mathcal{G}_{RO}$-CRS hybrid model. Finally, we argue that $\Sigma^{\texttt{uc}}_R$ is indeed a non-interactive Σ-protocol, completing the proof that $\Sigma^{\texttt{uc}}_R$ is a GUC-*transform* of Σ_R that maintains the overall Σ-protocol structure.

Clearly, $\Sigma^{\texttt{uc}}_R$ is non-interactive because `Prove` is an algorithm that the prover executes without interacting with the verifier. The algorithmic breakdown of $\Sigma^{\texttt{uc}}_R$ given in Definition 12 clearly resembles the overall structure of a Σ-protocol, with one key difference: the `Simulate` algorithm does not receive a challenge, and must simulate proofs using the CRS trapdoor instead. Since $\mathcal{G}_{RO}$ is not programmable, the traditional notion of Σ-protocol simulation cannot work in the GUC framework. The spirit of special honest-verifier zero-knowledge property, however, is maintained in that the simulated proofs are still statistically indistinguishable from regular proofs due to the witness indistinguishability property of OR-protocols. Therefore, we argue our construction is as close as it is possible to get to the traditional Σ-protocol simulation functionality within the GUC framework.

While the $\Sigma^{\texttt{uc}}_R.\texttt{Extract}$ algorithm takes different arguments as input, the functionality is effectively the same as the traditional Σ-protocol extractor. Our constructions of $\Sigma^{*}_{OR}.\texttt{Extract}$ and $\Sigma^{\texttt{uc}}_R.\texttt{Extract}$ simply spell out *where* the extractor is supposed to get the transcripts to invoke the special soundness property. The underlying extraction process still boils down to the OR-protocol extractor $\Sigma_{OR}.\texttt{Extract}$, which takes the traditional arguments as input. Similarly, the `Prove` algorithm takes as input a single statement and witness pair (x, w) and produces a compound proof (X, Π) where $X = (x, \texttt{CRS}_s)$ and Π is

an OR-protocol proof. That way, $\Sigma_R^{\mathtt{uc}}$ truly feels like a "transform"—able to take any statement x in the language of R as input, and produce a proof that GUC-realizes $\mathcal{F}_{\mathtt{NZP}}$. Unlike the simulator, these differences are merely cosmetic, allowing for cleaner explanations and notation—it would be possible to rearrange both functionalities to exactly match the original notation of the Σ-protocol.

We have now shown that $\Sigma_R^{\mathtt{uc}}$ is itself a Σ-protocol that GUC-realizes $\mathcal{F}_{\mathtt{NZP}}$, completing the proof of the theorem. □

Finally, we demonstrate that GUC-security also holds in the relaxed case where the underlying Σ-protocols are only computational honest-verifier zero-knowledge. That is, we consider the case where simulated proofs are only computationally indistinguishable from real ones, replacing $\{\pi\} \approx_s \{\pi'\}$ with $\{\pi\} \approx_c \{\pi'\}$ in the `Simulate` algorithm of Σ-protocols in Definition 2. The interesting thing in this case is that even though the zero-knowledge property only holds computationally, it still will not interfere with the knowledge extraction process necessary to satisfy security in the GUC model. In general, computational zero-knowledge may fail to compose with a knowledge extractor for the same proof system; but here, since extraction is exclusively through the oracle query history (such that anything $\mathcal{Z}$ can compute from interacting with the extractor it can compute on its own), the two security properties—computational zero-knowledge and knowledge extraction—will compose with each other.

Theorem 4. *If Σ_R is a Σ-protocol with only computational honest-verifier zero-knowledge, $\Sigma_R^{\mathtt{uc}}$ is a Σ-protocol with computational honest-verifier zero-knowledge that GUC-realizes $\mathcal{F}_{\mathtt{NZP}}$ in the $\mathcal{G}_{RO}$-CRS hybrid model.*

Experiment A and the proof that REAL = ExpA is the same. Experiment B is the same as above, only this time $\Sigma_{OR}^*.\mathtt{Simulate}(X)$ is only guaranteed to produce proofs that are *computationally* indistinguishable from real proofs. We replace the lemma ExpA $\approx_s$ ExpB as follows.

Lemma 6 (Experiment A $\approx_c$ Experiment B). *Experiment B is computationally indistinguishable from Experiment A. Formally,*

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

Proof. Given an environment $\mathcal{Z}_A$ that can distinguish between experiment A and experiment B, we construct a reduction that breaks the (computational) honest-verifier zero-knowledge property of Σ_{OR}. In particular, we consider a hybrid argument in which $\mathcal{Z}_A$ can distinguish between a world in which the first j proofs are constructed according to experiment A and the rest are constructed according to experiment B, and a world in which the first $j+1$ proofs are constructed according to experiment A and the rest are constructed according to experiment B. The reduction works as follows.

Upon getting an initial input of the security parameter from the honest-verifier zero-knowledge challenger, the reduction initializes $\mathcal{Z}_A$ and runs the experiment according to experiment A up to the $j+1^{st}$ prove query by letting $\mathcal{Z}_A$ and $\mathscr{A}$ interact freely with the honest parties. When it gets the $j+1^{st}$

prove query $\texttt{Prove}(x, w)$ from $\mathcal{Z}_A$, it issues the same challenge $\texttt{Prove}(x, w)$ to its challenger and either obtains a real proof $(X, \Pi) \leftarrow \Sigma^*_{OR}.\texttt{Prove}(X, W)$, or a simulated proof $(X, \Pi) \leftarrow \Sigma^*_{OR}.\texttt{Simulate}(X)$. The reduction then programs $\mathcal{G}_{RO}$ to make the output consistent with the proof (X, Π) it receives from its challenger, and returns (X, Π) to $\mathcal{Z}_A$.

For the remaining prove queries, the reduction proceeds according to experiment B, simulating each proof and programming $\mathcal{G}_{RO}$ to maintain a consistent view. If $\mathcal{Z}_A$ outputs j, the reduction outputs "Real" to indicate that the proof was calculated using the `Prove` algorithm. If $\mathcal{Z}_A$ outputs $j + 1$, the reduction outputs "Ideal" to indicate that the proof was calculated using the `Simulate` algorithm.

Analysis. If $\mathcal{Z}_A$ is able to distinguish between the hybrids, it must be able to tell whether the proof in the $j + 1^{st}$ slot was calculated according to experiment A—in other words, according to the "real world" `Prove` algorithm—or according to experiment B—in other words, according to the "ideal world" `Simulate` algorithm. This is in direct correspondence to whether the challenge the reduction received was generated using `Prove` or `Simulate`. Therefore the reduction is tight, and will break the honest-verifier zero-knowledge property with the same probability as $\mathcal{Z}_A$ is able to distinguish hybrid j from hybrid $j + 1$. As there can only be polynomially-many `Prove` queries, the probability that $\mathcal{Z}_A$ can distinguish experiment A from experiment B overall is negligible. □

Since the special soundness property remains unchanged, experiment C is unchanged. Note that since there is no change in the structure of the proofs between experiments B and C, the proof that ExpB $\approx_c$ ExpC is identical to the proof that ExpB $\approx_c$ ExpC from Theorem 3.

In experiment D, the simulated proofs are again only computationally zero-knowledge. Because there is computational wiggle room between the proofs in experiments C and D, *and* the experiment C-D distinguisher environment $\mathcal{Z}_C$ is now dealing with the extractor rather than a normal verifier, we cannot use the exact same argument as in the proof that ExpA $\approx_c$ ExpB. In particular, we have to make sure that the *only* way that $\mathcal{Z}_C$ can distinguish between hybrid j and $j + 1$ is if it can tell the difference between a real and a simulated proof in the $j+1^{st}$ slot. We argue that because anything $\mathcal{Z}_C$ can learn from the extractor it can learn from itself, it must not learn anything new about the proofs between experiments C and D.

Lemma 7 (Experiment C $\approx_c$ Experiment D). *Experiment D is computationally indistinguishable from Experiment C. Formally,*

$$\text{ExpC}_{\mathcal{C},\mathcal{A},\mathcal{Z}}(1^\lambda, \texttt{aux}) \approx_s \text{ExpD}_{\mathcal{C},\mathcal{A},\mathcal{Z}}(1^\lambda, \texttt{aux}).$$

Proof. Like the proof of ExpC $\approx_s$ ExpD in Theorem 3 above, this step reverts the proofs to being effectively non-simulated, and the reduction argument is similar to that used to prove Experiment A $\approx_c$ Experiment B. However since the distribution of the proofs are no longer statistically indistinguishable between experiments C and D, it might be the case that the environment $\mathcal{Z}_C$ can learn something from playing with the simulated proofs in experiment D that it

did not learn in experiment C, and vice versa. In particular, unlike experiment A, the verification process for the honest parties in experiment D is that the challenger tries to extract a witness using $\Sigma_R^{\mathsf{uc}}.\texttt{Extract}$. If for some reason the extraction process causes the challenger to output `Fail` in experiment C but not in experiment D, $\mathcal{Z}_C$ would be able to distinguish the experiments regardless of the nature of the $j+1^{st}$ proof. Therefore, in order to use the same argument as in ExpA $\approx_c$ ExpB, we must first argue that the reduction is still well-formed, and that the *only* way $\mathcal{Z}_C$ will distinguish the j from the $j+1^{st}$ hybrid is if it can tell the difference between a real proof and a simulated one.

Consider the inputs $(X, \Pi, \mathcal{Q}^s_{P*})$ to the algorithm $\Sigma^*_{OR}.\texttt{Extract}$ that is responsible for the verification procedure in experiments C and D. (X, Π) is a proof that $\mathcal{Z}_C$ itself produced. Similarly, $\mathcal{Q}^s_{P*}$ is a list of queries to $\mathcal{G}_{RO}$ made by either $\mathcal{Z}_C$ itself, or by the corrupted parties through $\mathscr{A}$ at $\mathcal{Z}_C$'s request. Therefore, $\mathcal{Z}_C$ 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 $\mathcal{Z}_C$ 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 proof that ExpA $\approx_c$ ExpB. □

Finally, the proof that ExpD = IDEAL is the same. □

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 Oded Goldreich. On defining proofs of knowledge. In *Annual International Cryptology Conference*, pages 390–420. Springer, 1992.
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
5. 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.
6. Manuel Blum, Alfredo De Santis, Silvio Micali, and Guiseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.
7. Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT '00*, pages 431–444, 2000.
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, 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.
10. 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.

11. 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.
12. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *SCN 2002*, volume 2576, pages 268–289, 2003.
13. 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.
14. Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO '99*, volume 1666, pages 413–430, 1999.
15. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO '03*, volume 2729, pages 126–144, 2003.
16. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *CRYPTO '97*, pages 410–424. Springer Verlag, 1997.
17. 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.
18. 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.
19. 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.
20. Ran Canetti and Tal Rabin. Universal composition with joint state. In *Annual International Cryptology Conference*, pages 265–281. Springer, 2003.
21. 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.
22. 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.
23. 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.
24. Ivan Damgård. On σ-protocols. University of Aarhus, Department of Computer Science, 2002.
25. 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.
26. Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. 29(1):1–28, 1999.
27. 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.
28. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Annual International Cryptology Conference*, pages 152–168. Springer, 2005.
29. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO '97*, pages 16–30, 1997.
30. 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.

31. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.
32. Carmit Hazay and Yehuda Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.
33. 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.
34. Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In *Advances in Cryptology - EUROCRYPT 2018*, volume 10822 of *Lecture Notes in Computer Science*, pages 552–586. Springer, 2018.
35. Helger Lipmaa. Statistical zero-knowledge proofs from diophantine equations. Manuscript. Available from http://eprint.iacr.org/2001/086, 2001.
36. 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.
37. Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: simple two-round schnorr multi-signatures. In *Annual International Cryptology Conference*, pages 189–221. Springer, 2021.
38. Rafael Pass. On deniability in the common reference string and random oracle model. In *Annual International Cryptology Conference*, pages 316–337, 2003.
39. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '92*, volume 576, pages 129–140, 1992.
40. 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.
41. 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.
42. 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.
43. David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.
44. John Watrous. Zero-knowledge against quantum attacks. *SIAM Journal on Computing*, 39(1):25–58, 2009.
45. Douglas Wikström. A commitment-consistent proof of a shuffle. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, pages 407–421. Springer, 2009.

A Supplementary materials

A.1 Protocol Template

Definition 13 (Protocol Template for Relation R). [24, 32] *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.2 Σ-protocol

Definition 14 (Σ-protocol). [24, 32] *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* (com, chl, res) *and* (com, chl', res') *for x where* chl $\neq$ 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* chl *outputs a transcript of the form* (com, chl, 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* chl $\in \{0,1\}^\ell$ *it holds that*

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

where $M(x,$ chl*$)$ denotes the output of simulator M on input x and* 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* chl*.*

The value ℓ is called the challenge length*.*

A.3 The OR-protocol

Definition 15 (OR-protocol for Relation R). [24, 32], [32] *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.

A.4 The Fiat-Shamir Transform

Recall that in a Σ-protocol, `Prove` protocol of consists of the commitment, challenge, and response procedures specified by the protocol template. The step requiring interaction with the verifier is the challenge phase, during which the verifier gets to see the prover's commitment. Issuing the challenge *after* the commitment is fixed ensures that the prover cannot cheat and just change the commitment to fit the challenge. In order to make the `Prove` procedure non-interactive and still maintain security for the verifier, the prover needs to truly commit to its commitment *before* the challenge is issued. In addition, the "honest-verifier" zero-knowledge property—the prover's guarantee that answering the challenge will not reveal anything about its secret—requires the challenge to be *random*.

Fiat and Shamir [27] proposed a non-interactive transform that was later shown to satisfy these conditions in the random oracle model [40]. Rather than asking the verifier for a challenge based on its commitment, the prover queries the random oracle $H : \{0,1\}^* \to \{0,1\}^\ell$ on input $(x, \texttt{com})$, and receives the random string `chl` in return. This process "fixes" the challenge as the oracle's response to the prover's commitment. To confirm that the prover is not cheating, the verification algorithm queries the oracle also and confirms that $H(x, \texttt{com}) = \texttt{chl}$. `Commit` and `Respond` are similarly modified to eliminate interaction—rather than sending `com` and `res` one by one to the verifier, the prover computes and stores the outputs of each algorithm until they are ready to be included in the final transcript. In order to avoid requiring any prior interaction between the prover and the verifier, the output of `Prove` (and therefore `Simulate`) includes the statement as well as the proof, or (x, π).

Definition 16 (Fiat-Shamir Transform). *The* non-interactive transform *for a Σ-protocol Σ_R based on protocol template τ_R transforms Σ_R into a tuple of non-interactive* algorithms *Σ_R^* based on τ_R^* in the random oracle model. The transform is made up of the following modifications to the algorithms in Σ_R and τ_R. Otherwise, Σ_R^* and τ_R^* are identical to Σ_R and τ_R, respectively.*

- *Modifications to τ_R: Before or during the execution of* `Setup`*, the* random oracle $H : \{0,1\}^* \to \{0,1\}^\ell$ *must be initialized. The specification of* `Challenge` *is replaced as follows: P obtains the challenge* `chl` *by querying H on input $(x, \texttt{com})$. During the execution of* `Commit` *and* `Respond`*, P does not send anything to V, but rather computes and stores the values. The acceptance conditions of* `Verdict` *additionally include a check that $H(x, \texttt{com}) = \texttt{chl}$.*
- *Modifications to Σ_R: Σ_R^* takes as input the modified protocol template τ_R^*.* `Prove` *is now an* algorithm *that P executes independently of V. The output of* `Prove` *and* `Simulate` *contains both the statement and proof, written (x, π).*

A.5 The Fischlin Transform

Definition 17 (Fischlin Transform). [28] *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).*

Online Extractor. *For all polynomial-time algorithms $\mathscr{A}$, let $\mathcal{Q}_H(\mathscr{A})$ denote the set of queries $\mathscr{A}$ makes to the random oracle H. Given $\pi = (a_i, e_i, z_i)$ for $1 \leq i \leq r$, search through $\mathcal{Q}_H(\mathscr{A})$ for a pair of queries $(x, \overline{a}, i, e_i, z_i)$ and $(x, \overline{a}, i, e_i', z_i')$ for $e_i \neq e_i'$ and such that $V_{FS}(x, a_i, e_i', z_i') = 1$. If two such queries are found, run the knowledge extractor of the underlying FS proof system on these values and output the witness w. Otherwise, output $\perp$.*

Zero-Knowledge Simulator. *Because the zero-knowledge simulator requires the random oracle H to be programmable, it is not applicable to our setting and therefore omitted. We refer the interested reader to [28].*

A.6 The GUC Real- and Ideal-World Experiments

Real-World Experiment. The real-world experiment $\mathrm{REAL}^{\mathcal{G}_{RO}, \mathcal{F}_{\mathsf{CRS}}}_{\Sigma_R^{\mathsf{uc}}, \mathscr{A}, \mathcal{Z}}(1^\lambda, \mathtt{aux})$ 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 $\mathcal{G}_{RO}$ on input 1^ℓ.[2]
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,[3] 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.

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

[3] 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. [17].

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 $\mathcal{G}_{RO}$ 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 $\Sigma_R^{\mathtt{uc}}.\mathtt{Setup}$, $\Sigma_R^{\mathtt{uc}}.\mathtt{Prove}$, and $\Sigma_R^{\mathtt{uc}}.\mathtt{Verify}$, respectively.

Ideal-World Experiment. The ideal world experiment $\mathrm{IDEAL}^{\mathcal{G}_{RO}}_{\Sigma_R^{\mathtt{uc}},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathtt{aux})$ 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[4] and $\mathcal{G}_{RO}$ 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 $\mathcal{F}_{\mathtt{NZP}}$ for their SID s. At initialization, $\mathcal{F}_{\mathtt{NZP}}$ 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 $\mathcal{F}_{\mathtt{NZP}}$, which responds according to its specification, given in Definition 5.

A.7 GUC-Security in the Global ROM

Formally, the definition of GUC-security in the global ROM states that for all efficient adversaries $\mathscr{A}$, we must be able to construct an ideal adversary $\mathcal{S}$ that can direct the ideal functionality $\mathcal{F}$ to simulate a protocol Σ. $\mathcal{S}$ must be efficient in expectation, and must function the same for all efficient environments $\mathcal{Z}$. We say Σ *GUC-realizes* an ideal functionality $\mathcal{F}$ if the probability distribution representing the environment's view of the "real" experiment (with Σ, $\mathscr{A}$, and $\mathcal{Z}$) is computationally indistinguishable from the distribution representing its view of the "ideal" experiment (with $\mathcal{F}$, $\mathcal{S}$, and $\mathcal{Z}$).

Definition 18 (GUC-Security in the Global ROM). [18, 19] *A protocol Σ with security parameter λ* GUC-realizes *an ideal functionality $\mathcal{F}_{\mathtt{NZP}}$ in the global ROM if for all efficient $\mathscr{A}$, there exists an ideal adversary $\mathcal{S}$ efficient in expectation such that for all efficient environments $\mathcal{Z}$,*

$$\mathrm{IDEAL}^{\mathcal{G}_{RO}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathtt{aux}) \approx_c \mathrm{REAL}^{\mathcal{G}_{RO}}_{\Sigma,\mathscr{A},\mathcal{Z}}(1^\lambda, \mathtt{aux}),$$

where $\mathcal{G}_{RO}$ is the global RO functionality in Definition 4, and aux *is any auxiliary information provided to the environment.*

[4] 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.8 Global Random Oracle

Definition 19 (Global Random Oracle). [19] *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.*