Resettable Non-Interactive Zero-Knowledge: Attacks and Defenses

Behzad Abdolmaleki[1], Matteo Campanelli[2], Quang Dao[3], Shadman Mohammadi[4], and Nahid Roustaeifar[1]

[1] University of Sheffield
{behzad.abdolmaleki, nroustaeifar1}@sheffield.ac.uk
[2] Offchain Labs and University of Tartu
binarywhalesinternaryseas@gmail.com
[3] Carnegie Mellon University
qvd@andrew.cmu.edu
[4] Independent Researcher
shadman.mohammadi1996@gmail.com

Abstract. As zero-knowledge proofs are increasingly deployed in real-world systems, they face new security threats beyond traditional theoretical guarantees. One important threat is resetting attacks, where an adversary exploits side-channel vulnerabilities or fault injection to manipulate a prover's randomness generation. While resettable zero-knowledge has been extensively studied for interactive protocols, it remains unclear whether modern non-interactive arguments (e.g., zkSNARKs) are secure against resetting attacks.

We present the first systematic study of resettable security for non-interactive zero-knowledge (NIZK) arguments. We make three contributions:

- **New Definition:** We formalize *strong resettable zero-knowledge* (srZK), which captures adversaries that can selectively reset portions of the prover's randomness while leaving other parts unchanged. This models practical attacks such as fault injection on secure hardware or partial state corruption in virtualized environments, which are not captured by the standard rZK definition.
- **Concrete Attacks:** We demonstrate that widely-used NIZK constructions are vulnerable to resetting attacks. We show witness-recovery attacks against Fiat-Shamir-compiled versions of (i) Σ-protocols (e.g., Schnorr), (ii) PIOP-based SNARKs (e.g., PlonK), and (iii) salted Fiat-Shamir compilation of rewindable protocols.
- **Generic Defense:** We present a simple compiler that transforms any NIZK into one satisfying srZK by modifying only the randomness generation. The prover now derives all necessary randomness by applying a pseudorandom function (PRF) to the public parameters, the statement, and the witness, using a short secret seed as the PRF key, i.e., $\tilde{r} = \mathsf{F}_r(\mathsf{pp}, \mathsf{x}, \mathsf{w})$. This approach prevents resetting attacks without increasing proof size and with only negligible proving overhead.

Our results demonstrate that resetting attacks must be considered in NIZK systems, and provide a practical defense with negligible overhead.

Keywords: Resettable zero-knowledge · Resetting attacks · zkSNARKs

1 Introduction

Non-Interactive Zero-Knowledge (NIZK) proofs are a fundamental component in both the theory and practice of cryptography. A NIZK scheme enables a prover to convince a verifier of a statement's validity through a single message, without revealing any information beyond the truth of the statement itself. NIZKs have numerous applications, including the construction of advanced cryptographic protocols [9, 28] and core blockchain technologies [52].

As NIZKs see increasing deployment in real-world systems, they face novel attacks that are not covered by standard security definitions, such as knowledge soundness and zero-knowledge. A particularly concerning threat model emerges when adversaries can *manipulate* or *reset* the prover's randomness—a capability that is becoming increasingly realistic. Consider the following scenarios:

- *Trusted Execution Environments (TEEs)* offering proving-as-a-service, where a malicious server might compromise the TEE's input-output operations, including randomness generation.
- *Virtual machines* in cloud environments, where VM snapshots enable adversaries to reset the prover to a previous state with the same randomness [50].
- *Smartcards and hardware tokens*, where physical access or side-channel attacks like electromagnetic fault injection can manipulate internal randomness [47, 55].

The concept of *resettable zero-knowledge* (rZK) [14] was introduced for interactive protocols to address this threat, ensuring the zero-knowledge property holds even if a malicious verifier can reset the prover and force the reuse of randomness. While significant progress has been made in constructing resettable *interactive* zero-knowledge systems (see Section 1.3), the security of NIZK proofs[5] in this setting remains unexplored: even related basic questions have remained open so far, for example whether classical NIZK constructions, such as those derived from the Fiat-Shamir transform, are secure even under resetting attacks; modern (and deployed) succinct proof systems like Plonk [32], Spartan [53], and Bulletproofs [12] have not been studied at all under the lens of resettable security.

A first critical set of questions that we seek to answer in this work thus is:

Q: Are existing NIZK constructions secure against resetting attacks? If not, what general techniques can make them secure with minimal overhead?

A Broader Class of Resetting Attacks. Additionally, we look at new and practically-motivated strengthening of the resetting attacks. The standard model for such attacks assumes an adversary can reset the prover's *entire* randomness tape. In contrast, realistic attack scenarios often grant the adversary more fine-grained control:

[5] For simplicity we occasionally talk about "proofs" but we focus exclusively on computationally-sound NIZKs [44], i.e., *arguments*.

- **Partial Randomness Manipulation:** Fault injection attacks like voltage glitching or laser injection can flip specific bits while leaving others intact, allowing an adversary to reset portions of the randomness [10, 11]. Side-channel attacks, such as template attacks, can also reveal or modify parts of a device's secret state, including the output of random number generators [4, 15].
- **Incomplete State Capture:** In virtualized environments, snapshots may only preserve partial states of random number generators (RNGs) [50]. This can occur due to system constraints or because the adversary lacks the capability to capture all random number sources, leading to a hybrid state where some randomness is fixed while other parts remain fresh.

Therefore, a second question this work seeks to answer is:

Q: How to protect NIZKs against resetting adversaries that can manipulate even only part of the randomness?

1.1 Our Results

This paper presents the first comprehensive study of resettable security for non-interactive zero-knowledge proofs (NIZKs). Our contributions are threefold: we (1) introduce a stronger security notion, (2) demonstrate concrete attacks on widely-used NIZK constructions, and (3) provide a generic, efficient compiler to achieve security against these attacks.

New Definition We introduce *strong resettable zero-knowledge (srZK)*, a security notion that generalizes classical resettable zero-knowledge (rZK) [14]. While classical rZK allows an adversary to reset the prover's entire randomness tape, our model grants the adversary fine-grained control to selectively reset arbitrary portions of it.

Definition 1.1 (srNIZK Informal). *A NIZK argument system Π is* strong resettable zero-knowledge (srZK) *if there exists a simulator $\mathcal{S}$ such that for every efficient* strong resetting adversary $\mathcal{A}$*, the following experiments are computationally indistinguishable.*

Real Experiment: *The adversary $\mathcal{A}$ interacts with the real prover as follows. Let ℓ denote the prover's randomness tape length and let $r \in \{0,1\}^\ell$ be a bitstring from which all internal coins (e.g., field elements) are deterministically derived. The adversary runs a polynomial number of sessions and adaptively chooses inputs based on the transcripts so far. In each session, $\mathcal{A}$ selects an instance $\mathsf{x} \in L$ and a valid witness w for x (not given to $\mathcal{A}$). The adversary also specifies a prior session and a subset of bit indices $I \subseteq [\ell]$. Let r be the (hidden) randomness used in that prior session. For the first session, r is uniformly random. All internal coins used by the prover are deterministically derived from r. The prover forms a new randomness tape r' by combining the bits r_I from the prior tape with fresh randomness $r'_{[\ell]\setminus I}$ for the positions not in I, then generates a proof for x using the composite randomness $r' = r_I \| r'_{[\ell]\setminus I}$.*

Ideal Experiment: *The adversary $\mathcal{A}$ interacts with the simulator $\mathcal{S}$ (which does not have access to any witness) using the same session protocol.*

Attacks on Practical NIZKs in the ROM While our definitions are general, our attack results focus on NIZKs in the random oracle model (ROM). We identify vulnerabilities in both the standard and strong resettable settings.

Theorem 1.2 (Standard Resetting Attack, Informal). *Let Π_{FS} be a Fiat-Shamir-compiled NIZK in the ROM. Π_{FS} is vulnerable to standard resetting attacks in the following cases.*

Maurer's Proof of Homomorphism Preimage: *For any rZK simulator $\mathcal{S}$ for this protocol, there exists an efficient meta-reduction that uses $\mathcal{S}$ as a black box to break the one-wayness of the underlying homomorphism, violating zero-knowledge.*

PIOP-Based NIZKs: *For a NIZK constructed from a Polynomial Interactive Oracle Proof (PIOP) system (e.g., Plonk), if a resetting adversary $\mathcal{A}$ has access to a known set of valid instance-witness pairs as auxiliary input, then there exists a witness-recovering attack with auxiliary input (Definition 3.5) under standard linear-response structure and deterministic commitments under fixed randomness.*

Theorem 1.3 (Strong Resetting Attack, Informal). *Let Π be a NIZK in the ROM. Π is vulnerable to* strong *resetting attacks in the following cases.*

Rewindable Arguments with Salted Fiat-Shamir: *Let Π be an interactive argument satisfying black-box rewinding knowledge soundness, and let Π_{sFS} be its non-interactive version obtained via salted Fiat-Shamir (Definition 2.5). Then Π_{sFS} admits a witness-recovering attack (Definition 3.4) under the same minimal structural assumptions.*

PIOP-Based NIZKs: *For a NIZK constructed from a PIOP system (e.g., Plonk), there exists a witness-recovering attack (Definition 3.4).*

Generic Compiler for NIZKs We propose a simple and efficient compiler that defends against both standard and strong resetting attacks for any NIZK system. Our approach modifies only the prover's randomness generation, introducing no proof size overhead and only negligible proving time overhead.

Theorem 1.4 (Generic Compiler for srNIZK, Informal). *Let $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a NIZK argument for an NP relation $\mathcal{R}$. We construct a compiled argument system $\tilde{\Pi} = (\tilde{\mathcal{G}}, \tilde{\mathcal{P}}, \tilde{\mathcal{V}})$ from Π as follows. The setup algorithm $\tilde{\mathcal{G}}$ and verifier $\tilde{\mathcal{V}}$ are identical to $\mathcal{G}$ and $\mathcal{V}$, respectively. The new prover $\tilde{\mathcal{P}}$, on input $(\mathsf{pp}, \mathsf{x}, \mathsf{w})$, samples a seed $r \leftarrow \{0,1\}^\lambda$ and derives its random tape as $\tilde{r} = \mathsf{F}_r(\mathsf{pp}, \mathsf{x}, \mathsf{w})$, where F is a pseudorandom function. It then outputs the proof $\pi \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}; \tilde{r})$.*

The resulting system $\tilde{\Pi}$ satisfies the following:

1. *If F is a standard PRF, then $\tilde{\Pi}$ is resettable non-interactive zero-knowledge (rNIZK) in the* ***standard model****.*

2. *If* F *is instantiated as a random oracle, then* $\tilde{\Pi}$ *is strong resettable non-interactive zero-knowledge (srNIZK) in the* ***random oracle model****.*
3. *If* F *is a strong resettable-key pseudorandom function (sRK-PRF), then* $\tilde{\Pi}$ *is strong resettable non-interactive zero-knowledge (srNIZK) in the* ***standard model****.*

The strong resettable-key PRF (sRK-PRF) is a new cryptographic primitive, which remains pseudorandom even against adversaries that can partially reset the PRF key. We demonstrate that this primitive can be efficiently instantiated in the random oracle model. We believe sRK-PRF may be of independent interest for constructing other resettable cryptographic protocols.

1.2 Technical Overview

This section provides a high-level overview of our approach to resettable and strong resettable zero-knowledge for NARGs. To ground our discussion in a concrete setting, we will use the Okamoto Σ-protocol [46] as a running example. The non-interactive version of this protocol, obtained via the Fiat-Shamir transformation, operates as follows:

1. **Public Parameters:** A group $\mathbb{G}$ of prime order p with generators g, h. The relation is $\mathcal{R}_{\mathsf{Oka}} = \{(\mathsf{x}, (\mathsf{w}_0, \mathsf{w}_1)) \mid \mathsf{x} = g^{\mathsf{w}_0} \cdot h^{\mathsf{w}_1}\}$.
2. **Prover $\mathcal{P}$:** Samples $r_0, r_1 \leftarrow \mathbb{F}_p$, computes the commitment $a = g^{r_0} \cdot h^{r_1}$, the challenge $c = \mathsf{H}(\mathsf{x}, a)$, and the responses $z_0 = r_0 + c \cdot \mathsf{w}_0$, $z_1 = r_1 + c \cdot \mathsf{w}_1$. The proof is $\pi = (a, z_0, z_1)$.
3. **Verifier $\mathcal{V}$:** Given x and $\pi = (a, z_0, z_1)$, recomputes $c = \mathsf{H}(\mathsf{x}, a)$ and checks $g^{z_0} \cdot h^{z_1} \stackrel{?}{=} a \cdot \mathsf{x}^c$.

The Distinction Between Resetting and Rewinding: A central conceptual point is that resetting—even in its strong form—does not automatically grant rewinding capabilities, which are typically reserved for extractors in security proofs. If resetting were equivalent to rewinding, witnesses could be extracted from any proof system, making simulation impossible for non-trivial languages.

To illustrate this distinction, consider the standard rewinding extractor for the Okamoto protocol. This extractor obtains two accepting transcripts (a, c, z_0, z_1) and (a, c', z_0', z_1') for the same x and a, where $c \neq c'$. It can then compute the witness as:

$$\mathsf{w}_0 = (z_0 - z_0') \cdot (c - c')^{-1}, \quad \mathsf{w}_1 = (z_1 - z_1') \cdot (c - c')^{-1}$$

Crucially, the extractor can rewind the protocol execution to a point after the first message a is fixed and assign a *new* independent challenge c'. This direct control over the challenge enables witness extraction.

In contrast, a (strong) resetting adversary cannot directly manipulate the challenge. For example, in the Okamoto protocol, if the adversary resets the prover while keeping all other parameters fixed, the challenge remains identical

$(c = c')$, and witness extraction fails. The adversary can only change the challenge by resetting the prover to generate a new instance x or a new first message a; it cannot change the challenge independently while keeping other components fixed. This limitation makes the design of (strong) resetting attacks against non-interactive arguments non-trivial. Our key contribution is identifying the precise conditions that enable these attacks.

Why Resetting Attacks Matter for NIZKs: In the interactive setting, *concurrent zero-knowledge* [29] is challenging because an adversary can manage multiple simultaneous protocol executions and interfere with the simulator's ability to produce a convincing transcript.

In contrast, NIZKs are naturally secure against concurrent attacks since each proof is a single, independent message, eliminating adversarial timing or manipulation. However, the situation becomes more complex in the resetting model, where an adversary can force the prover to generate proofs using fixed or partially fixed randomness.

To demonstrate the importance of studying NIZKs in the resettable setting, we analyze the Okamoto protocol under both resetting and strong resetting adversaries. This analysis highlights the vulnerabilities and clarifies the notions introduced in subsequent sections.

Breaking Zero-Knowledge: The standard simulator for Okamoto works as follows: it randomly selects $c, z_0, z_1 \leftarrow \mathbb{F}_p$, computes $a = g^{z_0} \cdot h^{z_1} \cdot \mathsf{x}^{-c}$, programs the random oracle so that $\mathsf{H}(\mathsf{x}, a) := c$, and returns the simulated proof $\pi_{\text{sim}} = (a, z_0, z_1)$.

However, in the resettable setting, this simulator is insufficient. An adversary may reset the prover multiple times using the same randomness $r = (r_0, r_1)$—and thus the same initial message a—while requesting proofs for new statements x_i. Consequently, the rNIZK simulator must produce multiple non-interactive proofs with the *same* first message a, a capability not supported by the standalone NIZK simulator.

Witness Recovery: As shown, resetting attacks on non-interactive protocols can disrupt a simulator's performance and break the zero-knowledge property. However, this is not the only concern; in many cases, such attacks can enable an adversary to recover the witness directly.

Reconsider the Okamoto protocol. In a three-round interaction, a resetting adversary can easily recover the witness by fixing all the prover's inputs—including $(\mathsf{pp}, \mathsf{x}, \mathsf{w}; r)$—and selecting fresh challenges c_i. This attack does not apply directly to non-interactive protocols, since fixing all prover inputs yields the same challenge and thus the same proof, leaking no new information. Nevertheless, a resetting adversary can overcome this limitation.

Suppose a resetting adversary has access to an auxiliary statement-witness pair $(\mathsf{x}', \mathsf{w}' = (\mathsf{w}'_0, \mathsf{w}'_1))$. The adversary can efficiently generate such a pair itself by sampling w' and computing $\mathsf{x}' = g^{\mathsf{w}'_0} \cdot h^{\mathsf{w}'_1}$, requiring *no additional assumptions* (e.g., no need to assume that these auxiliary pairs are leaked from other attacks). The adversary first requests a proof $\pi' = (z'_0, z'_1, a)$ from the prover using these auxiliary inputs. Knowing w', the adversary recovers the randomness $r = (r_0, r_1)$

used in the proof as:

$$r_0 = (z'_0 - c \cdot \mathsf{w}'_0), \quad r_1 = (z'_1 - c \cdot \mathsf{w}'_1)$$

The adversary then resets the prover, forcing it to reuse the same randomness r, and obtains a new proof $\pi = (z_0, z_1, a)$ for a different statement x and the secret witness w. Using the known randomness r, the adversary recovers the secret witness $\mathsf{w} = (\mathsf{w}_0, \mathsf{w}_1)$ as:

$$\mathsf{w}_0 = (z_0 - r_0) \cdot c^{-1}, \quad \mathsf{w}_1 = (z_1 - r_1) \cdot c^{-1}$$

This illustrates the concept of a witness-recovering attack with auxiliary input, similar to the attack we apply to PIOP-based NIZKs in Section 4.1.

Now consider a scenario where the adversary cannot participate in the experiment corresponding to a witness-recovering attack with auxiliary input. Instead, the adversary can fix part of the prover's randomness while the remainder is freshly generated, as in our *strong* resetting model. The adversary proceeds as follows:

The adversary requests a proof, and the prover, using inputs $(\mathsf{pp}, \mathsf{x}, \mathsf{w}; r_0, r_1)$, generates the proof $\pi = (a, z_0, z_1)$. Then, the adversary selectively resets the prover, forcing it to reuse r_0 while fresh randomness r'_1 is generated for the second part. Consequently, the prover produces a new proof $\pi' = (a', z'_0, z'_1)$. This strategy enables the adversary to recover the first part of the witness w_0 as follows:

$$\mathsf{w}_0 = (z_0 - z'_0) \cdot (c - c')^{-1}$$

Similarly, by resetting the prover on the second part r_1, the adversary recovers w_1. This is a witness-recovering attack in the strong resettable setting, similar to the attack on PIOP-based NARGs in Section 4.2.

Finally, suppose the Okamoto protocol is made non-interactive via the salted Fiat-Shamir transform, where the challenge is computed as $c = \mathsf{H}(\mathsf{x}, a; \tau)$ and the salt τ is part of the prover's randomness. A strong resetting adversary can fix the primary randomness $r = (r_0, r_1)$ while allowing the salt τ to be freshly generated, producing different challenges even when x and a remain fixed . In this case, the adversary can precisely imitate the extractor's behavior to recover the witness, making rewinding and resetting equivalent for this construction. We extend this attack to rewindable black-box interactive arguments compiled with the salted Fiat–Shamir transform in Section 4.2.

Generic Compiler: To defend against the aforementioned attacks, we structure the prover's randomness to satisfy two key properties. First, randomness cannot be altered "locally"; any change in the initial randomness must have a global effect. This property effectively reduces the security requirement from srNIZK to rNIZK. Second, the randomness must be bound to the specific instance and witness being proven; a change in the instance or witness should result in a completely different random tape. This further reduces the requirement from rNIZK to standard zero-knowledge.

A crucial requirement is that the base NIZK must be *multi-theorem* zero-knowledge, meaning the public parameters pp can be safely reused for any polynomial number of proofs. This aligns perfectly with the resettable setting, where an adversary can initiate multiple proof sessions, all using the same pp.

We achieve both defense objectives through a simple yet powerful modification to the prover's randomness generation. The prover derives its full random tape by applying a pseudorandom function to the public parameters, the instance, and the witness, using a short secret seed as the PRF key. That is, it computes $\tilde{r} = \mathsf{F}_r(\mathsf{pp}, \mathsf{x}, \mathsf{w})$.

In the standard resettable setting, it is sufficient that F is a standard PRF. For the strong resettable setting, however, we require a *strong resettable-key PRF (*$\mathsf{sRK\text{-}PRF}$*)*, which ensures pseudorandomness is maintained even when an adversary forces the prover to reuse previous key states—the core of a resetting attack. Including the witness in the PRF input guarantees that different witnesses for the same statement produce distinct random tapes, which reduces the problem to witness indistinguishability in the standalone setting.

A complete security proof of our compiler is provided in Section 5.

1.3 Related Works

Strong Notions of Security Against Malicious Provers. To address vulnerabilities in the soundness of NIZKs, particularly concerning knowledge-soundness against a malicious prover, Sahai's seminal work [51] introduced a stronger notion known as *simulation soundness*. This was later extended to *simulation-extractability* [23]. These notions essentially assert that no cheating prover can compromise (knowledge) soundness, even after obtaining simulated proofs for adaptively chosen statements from a zero-knowledge simulator. A significant body of research focuses on generically enhancing NIZK systems to achieve simulation soundness and simulation-extractability [1–3,16,22,23,30,33,36,40–43,48].

Resettable Zero-Knowledge in the Interactive Setting. The notions of *concurrent zero-knowledge* [29] and *resettable zero-knowledge* [14] were proposed for interactive protocols. Concurrent Zero-Knowledge (cZK) guarantees that the zero-knowledge property holds even when multiple protocol sessions are arbitrarily interleaved. Resettable Zero-Knowledge (rZK), our primary focus, ensures that zero-knowledge is preserved even when an adversary can reset the prover and force it to reuse the same randomness across multiple executions.

Following the introduction of rZK in [14], there has been significant progress in constructing proof systems with this type of security. For instance, [6] proposed a transformation that converts an *admissible proof system*—zero-knowledge in a hybrid model—into a resettable zero-knowledge system. Building on this, Deng, Goyal, and Sahai [25] provided the first construction of *simultaneously resettable zero-knowledge* (i.e., resettable-sound and resettable-zero-knowledge) arguments for NP based on standard hardness assumptions.

A substantial body of literature explores resettable security in the relaxed *bare public key* model, which assumes all verifiers deposit a public key in a public

file before any interaction occurs [6, 14, 20, 24, 26, 27, 45, 60]. Other works have proposed resettable *statistical* zero-knowledge [34, 38], which guarantees security against unbounded malicious resetting adversaries, even when the prover reuses randomness.

However, all these efforts have focused on *interactive* protocols. The security of *non-interactive* arguments in the resettable setting remains formally unexplored, a gap our work aims to fill.

Hedging Cryptography Against Randomness Failures: Since randomness failures are widespread and unlikely to be completely eliminated, a fundamental approach is to design cryptographic primitives that hedge against such failures. A simple and elegant method for hedged public-key encryption (PKE) is the *Randomized-Encrypt-With-Hash* (REwH) technique introduced by Bellare et al. [7]. This approach synthesizes fresh random bits by hashing all encryption inputs, using the resulting hash output as randomness for the underlying PKE scheme. While their goal of mitigating bad randomness aligns with ours, their definitions do not capture the full adversarial control of the resettable setting.

In subsequent work aligned with our goal, Yilek [59] addressed the case of reused (but not adversarially chosen) randomness, suggesting a construction where the hash is replaced by a keyed pseudorandom function (PRF). Building on this, Ristenpart and Yilek [50] proposed a backwards-compatible framework for hedging deployed cryptography, protecting primitives like PKE, symmetric encryption, digital signatures, and key exchange against a wide range of randomness failures. Similar techniques were later applied to authenticated key exchange (AKE) protocols [58].

Motivated by the history of randomness failures, Paterson, Schuldt, and Sibborn [49] discuss hedging in the context of PKE to protect against *related randomness attacks*, where the randomness used after a reset is correlated with previously used values.

Our compiler is conceptually similar to these hedging transformations, which aim to guarantee security in the presence of imperfect randomness. A key distinction is that our compiler assumes access to a small, reliable source of true randomness (the PRF seed) and demonstrates that this alone is sufficient to achieve the strong, formal guarantee of resettable zero-knowledge, a notion previously unaddressed for non-interactive proofs.

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \dots\}$ denote the set of positive integers. For $\eta \in \mathbb{N}$, we write $[\eta] := \{1, 2, \dots, \eta\}$. For $1 \le k \le \eta$, we write $[k, \eta] := \{k, k+1, \dots, \eta\}$.

For a list L, we write $L[l]$ to denote the l-th element of L (indexing starts at 1), so elements of L are accessed with indices in $\{1, \dots, |L|\}$, where $|L|$ denotes its length. Each element of L is an ℓ-bit string.

For an ℓ-bit string r, we write $r[i]$ to denote the i-th bit of r (indexing also starts at 1), so bit positions are identified with indices in $[\ell] = \{1, \dots, \ell\}$.

The security parameter is denoted by λ. A function $f(\lambda)$ is *negligible* if for any positive polynomial $\mathsf{poly}(\lambda)$, $f(\lambda) \leq 1/\mathsf{poly}(\lambda)$ for all sufficiently large λ. We write $f(\lambda) \leq \mathsf{negl}(\lambda)$ to indicate that f is negligible.

By $y \leftarrow A(x;\rho)$, we denote that a randomized algorithm A outputs y on input x using random coins ρ sampled uniformly from its randomness space.

For a finite field $\mathbb{F}$, $\mathbb{F}^{\leq d}[X]$ denotes the set of polynomials over $\mathbb{F}$ of degree at most d.

Indexed Relations. An indexed relation $\mathcal{R}_{\mathsf{i}}$ is a set of triples $(\mathsf{i}, \mathsf{x}, \mathsf{w})$, where i is the index, x is the instance, and w is the witness. We assume there exists an efficient algorithm to check whether $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathsf{i}}$.

2.1 Interactive Arguments

Definition 2.1 (Interactive Argument). *An interactive argument (*IARG*) for a relation $\mathcal{R}$ is a tuple of PPT algorithms $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ with the following syntax:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: *outputs public parameters* pp *given security parameter* 1^λ.
- $\langle \mathcal{P}(\mathsf{w}), \mathcal{V} \rangle (\mathsf{pp}, \mathsf{x}) \to \{0,1\}$: *an interactive protocol whereby the prover $\mathcal{P}$, holding a witness* w*, interacts with the verifier $\mathcal{V}$ on common input* $(\mathsf{pp}, \mathsf{x})$ *to convince $\mathcal{V}$ that* $(\mathsf{pp}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$*. At the end, $\mathcal{V}$ outputs a bit to accept or reject the proof.*

We define the following properties for interactive arguments:

- **Completeness.** For any adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{l|l} (\mathsf{pp}, \mathsf{x}, \mathsf{w}) \notin \mathcal{R} \vee & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \langle \mathcal{P}(\mathsf{w}), \mathcal{V} \rangle (\mathsf{pp}, \mathsf{x}) = 1 & (\mathsf{x}, \mathsf{w}) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] = 1.$$

- **Knowledge Soundness.** There exists a expected polynomial time extractor $\mathcal{E}$ such that for any stateful PPT adversary $\mathcal{P}^*$, the following probability is negligible in λ:[6]

$$\Pr\left[\begin{array}{r:l} & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ b = 1 \wedge & (\mathsf{x}, \mathsf{st}_{\mathcal{P}^*}) \leftarrow \mathcal{P}^*(\mathsf{pp}) \\ (\mathsf{pp}, \mathsf{x}, \mathsf{w}) \notin \mathcal{R} & b \leftarrow \langle \mathcal{P}^*(\mathsf{st}_{\mathcal{P}^*}), \mathcal{V} \rangle (\mathsf{pp}, \mathsf{x}) \\ & \mathsf{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathsf{pp}, \mathsf{x}) \end{array}\right]$$

  Here $\mathcal{E}$ gets black-box access to each of the next-message functions of $\mathcal{P}^*$ in the interactive protocol, and can rewind $\mathcal{P}^*$ to any point in the interaction.
- **Public Coin.** In each round i, the verifier $\mathcal{V}$ samples its message uniformly at random from some challenge space Ch_i.

[6] We do not define soundness, but this implies soundness with the same advantage.

2.2 Non-Interactive Arguments

Definition 2.2 (Non-Interactive Argument (NARG)). *A* NARG $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ *for an NP relation* $\mathcal{R}$ *is a triple of PPT algorithms with the following syntax:*

- $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$*: The setup algorithm takes the security parameter* λ *and generates public parameters* pp*.*
- $\pi \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w})$*: The prover algorithm takes the public parameters* pp *a statement* x*, and a witness* w*. It outputs a a proof* π *asserting* $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$*.*
- $b \leftarrow \mathcal{V}(\mathsf{pp}, \mathsf{x}, \pi)$*: The verifier algorithm is deterministic. It takes the public parameters* pp*, a statement* x*, and a proof* π*, and outputs a bit* b*, where* $b = 1$ *signifies "accept" and* $b = 0$ *signifies "reject".*

Remark 2.3 (On Generality). This definition provides a unified syntactic framework for non-interactive arguments. The algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ can be instantiated in various models:

- **CRS/SRS:** $\mathcal{G}$ generates a common (or structured) reference string, possibly including a trapdoor.
- **ROM:** $\mathcal{P}$ and $\mathcal{V}$ have black-box access to a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$. We apply our attacks from Section 4 in this model.
- **Transparent:** $\mathcal{G}$ outputs a public random string.

Additionally, there exists the notion of *indexed* (or preprocessing) NARGs, where a deterministic indexer algorithm is introduced for efficiency. Any indexed NARG can be transformed into a standard NARG by incorporating the indexer into both the prover and verifier. Our attacks also apply to some indexed NARGs (e.g., popular SNARKs).

While NARGs typically satisfy completeness and (knowledge) soundness, our focus is on the zero-knowledge property. We now define a non-interactive zero-knowledge argument (NIZK) as a NARG that also satisfies zero-knowledge.

Definition 2.4 (Non-Interactive Zero-Knowledge (NIZK)). *Let* $\Pi = (\mathcal{G}, \mathsf{P}, \mathsf{V})$ *be a* NARG *for an NP relation* $\mathcal{R}$*. We say* Π *is a* NIZK *if it satisfies* computational adaptive multi-theorem zero-knowledge*. Specifically, there exists a PPT simulator* $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ *such that for any PPT adversary* $\mathcal{B}$*:*

$$\mathsf{Adv}^{\mathsf{NIZK}}_{\mathcal{B},\Pi}(\lambda) := \left| \Pr\left[\mathsf{Real}^{\Pi}_{\mathcal{B}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}^{\Pi}_{\mathcal{S},\mathcal{B}}(1^\lambda) = 1\right] \right| \leq \mathrm{negl}(\lambda)$$

where the experiments are defined in Figure 1, and $\mathcal{B}$ *makes polynomially many adaptive queries to its oracle.*

It is well-known that a NIZK with adaptive single-theorem zero-knowledge can be generically converted into a NIZK with adaptive multi-theorem zero-knowledge using a pseudo-random generator (PRG) [31] or simulatable verifiable random function (sVRF) [17]. It is easy to see that the same construction works in the non-adaptive setting.

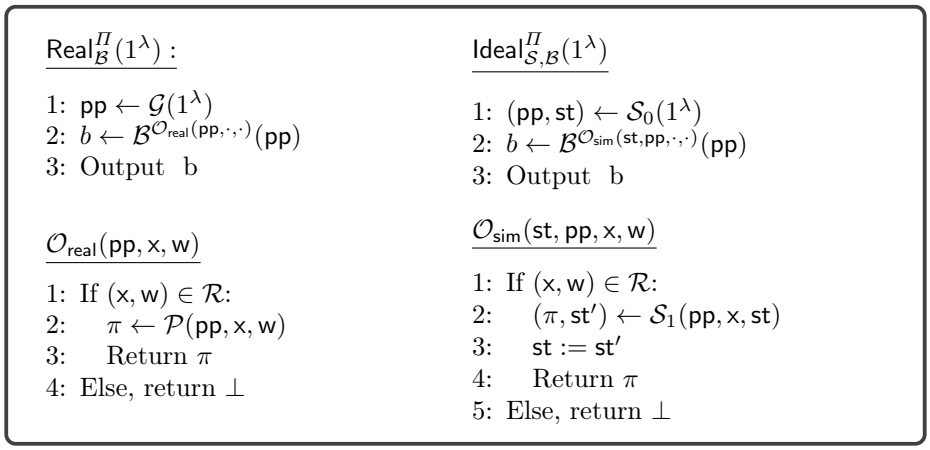

Fig. 1: Experiments for NIZK.

2.3 Fiat-Shamir transformation

We now describe the Fiat-Shamir transformation, which turns a public-coin interactive argument (Definition 2.1) into a non-interactive argument in the ROM (Definition 2.4). Our description is the more efficient version of Fiat-Shamir in [19], which is also the approach taken in popular implementations [21, 56].

Definition 2.5 ((Salted) Fiat-Shamir Transformation). *Let* $\mathsf{IARG} = (\mathcal{G},\mathcal{P},\mathcal{V})$ *be a* r*-round interactive argument. We say that* $\mathsf{NARG} := \mathsf{FS}^*_{salt}[\mathsf{IARG},\lambda]$ *is the non-interactive argument in the ROM obtained from* IARG *via the* (salted) Fiat-Shamir transformation *if* NARG *is defined as follows:*

- $\mathsf{pp} \leftarrow \mathsf{NARG}.\mathcal{G}(1^\lambda)$*: Return* $\mathsf{pp} \leftarrow \mathsf{IARG}.\mathcal{G}(1^\lambda)$.
- $\pi \leftarrow \mathsf{NARG}.\mathcal{P}^{\mathsf{H}}(\mathsf{pp},\mathsf{x},\mathsf{w})$*:*
  1. *For* $i = [\mathsf{r}]$*:*
     (a) *Compute the* i*-th message* a_i
     (b) *Sample a random salt* $\tau_i \in \{0,1\}^\lambda$.
     (c) *Derive the* i*-th random challenge* c_i *as:*

$$c_i := \begin{cases} \mathsf{H}(1,\mathsf{x},a_1;\tau_1) & \text{if } i = 1, \\ \mathsf{H}(i,c_{i-1},a_i;\tau_i) & \text{if } i > 1. \end{cases}$$

  2. *Output the proof* $\pi := \big((a_i)_{i\in[\mathsf{r}]},(\tau_i)_{i\in[\mathsf{r}]}\big)$.
- $b \leftarrow \mathsf{NARG}.\mathcal{V}^{\mathsf{H}}(\mathsf{pp},\mathsf{x},\pi)$*:*
  1. *Parse the proof* $\pi = \big((a_i)_{i\in[\mathsf{r}]},(\tau_i)_{i\in[\mathsf{r}]}\big)$.
  2. *For each round* $i = [\mathsf{r}]$*, derive the challenge* c_i *exactly as in Step 1(c) of the prover.*
  3. *Return* $b \leftarrow \mathsf{IARG}.\mathcal{V}\big(\mathsf{pp},\mathsf{x},(a_i)_{i\in[\mathsf{r}]},(c_i)_{i\in[\mathsf{r}]}\big)$.

Clearly, if the parts highlighted in orange are not executed, the transformation corresponds to the standard (non-salted) Fiat-Shamir.

2.4 Polynomial Interactive Oracle Proofs

We define Polynomial Interactive Oracle Proofs (PIOP) with preprocessing, following the formulation in [39], which builds upon earlier works [18] and [13].

Definition 2.6 (PIOP). *A Polynomial Interactive Oracle Proof (*PIOP*) for an indexed relation* $\mathcal{R}_{\mathsf{i}}$ *is a tuple of algorithms* $\mathsf{PIOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$*. The protocol execution proceeds as follows:*

- ***Offline Phase:*** *Given an index* i *for* $\mathcal{R}_{\mathsf{i}}$*, the indexer* $\mathbf{I}$ *outputs* $\mathsf{s}(0)$ *preprocessed polynomials* $p_{0,1}, \ldots, p_{0,\mathsf{s}(0)} \in \mathbb{F}^{\leq d}[X]$*.*
- ***Online phase*** *Given an instance* x *and witness* w *such that* $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathsf{i}}$*, the prover* $\mathbf{P}$ *receives* $(\mathsf{i}, \mathsf{x}, \mathsf{w})$*, and the verifier* $\mathbf{V}$ *receives* x *and has oracle access to the polynomials output by* $\mathbf{I}(\mathsf{i})$*. The parties engage in an* ρ*-round interaction. In round* r*, the prover* $\mathbf{P}$ *sends* $\mathsf{s}(r)$ *oracle polynomials* $p_{r,1}, \ldots, p_{r,\mathsf{s}(r)} \in \mathbb{F}^{\leq d}[X]$ *to the verifier* $\mathbf{V}$*, who then replies with a challenge* $c_r \in \mathsf{Ch}$*. The protocol is* public-coin*, meaning all challenges* c_r *are public and uniformly sampled from the challenge space* Ch*.*
  Query phase*, let* $\mathbf{p} = (p_{r,t})_{r \in [\rho], t \in [\mathsf{s}(r)]}$ *denote all polynomials sent by the prover. The verifier executes a query subroutine* $\mathbf{Q}_{\mathbf{v}}$ *on input* $(\mathsf{x}; c_1, \ldots, c_\rho)$ *and outputs a vector of evaluation points* $\mathbf{z} = (z_{r,t})_{r \in [\rho], t \in [\mathsf{s}(r)]}$[7]*, and obtains the corresponding evaluation vector* $\mathbf{y}_{r,t} = p_{r,t}(z_{r,t})$*.*
- ***Decision phase*** *the verifier executes a decision subroutine* $\mathbf{D}_{\mathbf{V}}$ *that receives* $(\mathsf{x}, \mathbf{p}(\mathbf{z}); c_1, \ldots, c_\rho)$ *as input and outputs a decision bit* b *indicating acceptance or rejection.*

Remark 2.7 (PIOP with Multilinear Polynomials). Definition 2.6 assumes that the polynomials sent are univariate. There is a growing body of works on SNARKs built from PIOPs with *multilinear* polynomials [5,35,53,54,57]. To accommodate this, we simply need to replace "univariate" with "multilinear" or "multivariate" in our definition above, with the degree restrictions extended properly.

We require PIOP to satisfy following properties:

Definition 2.8 (Completeness). *A* PIOP *is complete if for any* $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathsf{i}}$*,*

$$\Pr\left[b \leftarrow \langle \mathbf{P}(\mathsf{i}, \mathsf{x}, \mathsf{w}), \mathbf{V}^{\mathbf{I}(\mathsf{i})}(\mathsf{x}) \rangle : b = 1\right] = 1.$$

Definition 2.9 (Knowledge Soundness). *A* PIOP *for* $\mathcal{R}_{\mathsf{i}}$ *is knowledge sound if there exists a PPT extractor* E *such that for all admissible provers* $\mathbf{P}^*$[8]*, every*

[7] In each round $r \in [\rho]$ and for each polynomial $t \in [\mathsf{s}(r)]$, it is possible that the verifier queries $\mathsf{s}(r, t)$ evaluation points. In such cases, the notation should be $\mathbf{z} = (z_{r,t,u})_{u \in [\mathsf{s}(r,t)]}$. However, for simplicity, we consider $\mathsf{s}(r, t) = 1$ and omit this parameter.

[8] An admissible prover is a possibly malicious prover that sends polynomials with a maximum degree of d (see [39]).

index i*, and every statement* x*, the following holds:*

$$\Pr\left[\begin{array}{c} b \leftarrow \langle \mathbf{P}^*(\mathsf{i},\mathsf{x}), \mathbf{V}^{\mathbf{I}(\mathsf{i})}(\mathsf{x})\rangle; \\ \mathsf{w}^* \leftarrow \mathrm{E}^{\mathbf{P}^*}(\mathsf{i},\mathsf{x}) \mid \\ b = 1 \wedge (\mathsf{i},\mathsf{x},\mathsf{w}^*) \notin \mathcal{R}_{\mathsf{i}} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We also provide definitions of honest-verifier zero-knowledge (HVZK) in Section A.1. Additionally, Section A.2 includes definitions of a polynomial commitment scheme (PCOM). Section A.3 describes how a PIOP can be compiled with a PCOM to yield a (public-coin) interactive argument, which can then be transformed into a *PIOP-based NARG* via the Fiat-Shamir transformation.

3 Non-Interactive Arguments in the Resettable Setting

In interactive protocols, a malicious verifier may execute multiple interleaved sessions with a prover, sending adaptively chosen messages to gain an advantage. This motivates the study of *concurrent zero-knowledge*. For non-interactive arguments, however, where each proof consists of a single message, the concept of concurrency is less meaningful. An adversary can already request proofs for arbitrary statements adaptively and in any order—a capability captured by the standard adaptive security game for NIZKs. Thus, concurrency does not introduce substantially new adversarial capabilities in the non-interactive setting.

In contrast, the ability to *reset* the prover—forcing it to reuse randomness across multiple proof generations—represents a meaningful and practically significant threat. In this section, we introduce a general resetting model where the adversary can partially reset the prover's randomness, controlling exactly which specific bits are reused across sessions. This constitutes a strong resettable setting that subsumes the classical notion. Within this general model, we first define resettable NIZKs and then introduce witness-recovering attacks.

3.1 Strong Resettable NIZK

As discussed, a strong resetting adversary can selectively reset arbitrary portions of the prover's randomness. Crucially, while the adversary chooses which bit positions to fix, it does not learn the actual values of those bits. Therefore, the adversary cannot compute the resulting composite randomness itself.

To formally model this capability, we define a stateful algorithm $\mathsf{RandGen}$ that manages the randomness generation process according to the adversary's reset instructions. The algorithm works by maintaining a list of previously generated randomness strings. On each query, it constructs a new randomness string where specified bit positions are fixed to values from a previous string, while the remaining bits are freshly sampled. The algorithm initializes itself automatically on the first call—when the list is empty—and thereafter retrieves existing randomness values by index.

Definition 3.1 (Randomness Generator for Strong Resetting Attacks). *Let $\ell = \ell(\lambda)$ be the length of the prover's randomness strings. The algorithm* $\mathsf{RandGen}$ *is stateful and operates as follows:*

- ***State:*** *A list L (initially empty). Each element of L is an ℓ-bit string.*
- ***Input:*** *A pair (l, I) where $l \in \mathbb{N}$ and $I \subseteq [\ell]$ [9] is a subset of bit indices.*
- ***Output:***
  - *If L is empty (first call), sample $r_1 \leftarrow \{0,1\}^\ell$, set $L := [r_1]$, and return r_1.*
  - *Else, if $l > |L|$, return $\perp$.*
  - *Otherwise:*
    - *Retrieve $r := L[l]$*[10]
    - *Sample fresh randomness $r'_{\mathsf{fresh}} \leftarrow \{0,1\}^{\ell - |I|}$.*
    - *Construct a new ℓ-bit string r' as follows:*
      - *For each position $i \in [\ell]$:*
        - *If $i \in I$, set $r'[i] := r[i]$.*
        - *Else, set $r'[i]$ from r'_{fresh} (in increasing order of i).*
    - *Append r' to L.*
    - *Return r'.*

We are now ready to define the zero-knowledge property in the resettable setting. As in the standalone setting (Definition 2.4), the core requirement is the indistinguishability between real proofs generated by an honest prover and simulated proofs generated by a simulator. The key difference lies in the proof generation mechanism under resetting attacks.

In the real world, the adversary's queries are handled by an oracle that executes the honest prover algorithm $\mathcal{P}$ using the randomness generation process defined by $\mathsf{RandGen}$. In the ideal world, a simulator must produce convincing proofs for the same query pattern without access to the witness.

The Consistency Challenge. In the resetting model, simulators face a critical challenge: maintaining consistency across repeated queries. In a real execution, when an adversary queries the same statement-witness pair (x, w) and the $\mathsf{RandGen}$ algorithm produces identical randomness r', the honest prover deterministically generates the same proof π (assuming public parameters are unchanged). A naive simulator that generates a fresh proof for each query would fail to replicate this behavior, becoming easily distinguishable.

To resolve this, we rely on the oracle simulator $\mathcal{O}_{\mathcal{S}}$. To emulate the prover's deterministic nature, $\mathcal{O}_{\mathcal{S}}$ first runs its own subroutine $\mathsf{RandGen}_{\mathsf{sim}}$, which is initialized with independent randomness but otherwise behaves identically to the real $\mathsf{RandGen}$ algorithm (it maintains its own list and samples fresh bits for positions not in I). It then consults an internal dictionary $\mathcal{D}$ that maps a pair $(\mathsf{pp}, \mathsf{x}, \mathsf{w}r'_{\mathsf{sim}})$ to its corresponding simulated proof π. If an entry exists for a given

[9] Recall that for a positive integer ℓ, $[\ell] := \{1, 2, \ldots, \ell\}$ is a subset of bit indices.
[10] Recall that for a list L, $L[l]$ denotes the l-th element of L, which is an ℓ-bit string.

key, the oracle returns the stored simulated proof, thereby perfectly replicating the real prover's consistency.

A crucial observation is that this consistency check must be performed by the oracle, not by the simulator itself. The simulator $\mathcal{S}_1$ does not have access to w or r'_{sim}; it only receives $(\mathsf{pp}, \mathsf{x})$ as input. Therefore, it cannot determine whether a query is a repeat or not. The oracle must maintain the dictionary externally and decide when to invoke $\mathcal{S}_1$ versus return a cached proof. This makes the dictionary an explicit part of the definition.

What consistency imposes and why it is good. In the standard NIZK definition (Definition 2.4), the oracles check that $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$ before producing a proof. This validity check is necessary for the definition to work—otherwise a trivial attack exists. This formulation is equivalent to running the prover or simulator directly while restricting the adversary to query only valid instance-witness pairs. This restriction is without loss of generality, because invalid queries reveal no information about any valid witness.

Similarly, in the resettable setting, we add a consistency mechanism to avoid a trivial distinguishing attack. This mechanism is equivalent to providing a normal simulator oracle as in the NIZK definition, while restricting the adversary to non-duplicate queries.[11] This assumption of non-duplicate queries is standard in resettable cryptography. For instance, in resettable public-key encryption [59], it is assumed that the adversary never makes duplicate queries to the encryption oracle because "all such queries will return the same response." The same reasoning applies to our setting. An adversary making duplicate queries receives the same proof repeatedly; such an adversary can be transformed into one that makes only non-duplicate queries and simply copies the received proof locally. Therefore, this restriction to non-duplicate queries does not impose a loss of generality.

Defining *strong resettable witness indistinguishability for NIZKs* would also require a non-duplicated query condition, as this condition appears in the definition of resettable witness indistinguishability in the interactive setting [6, 14]. However, we do not need this notion for our results. Without this condition, an adversary could trivially distinguish by issuing duplicate queries (i.e., the same $(\mathsf{pp}, \mathsf{x}, \mathsf{w}, r')$) and observing whether the returned proofs are identical. Since the real prover would return the same proof for duplicate queries, the repetition pattern would leak information about witness equality, breaking witness indistinguishability. Therefore, the non-repeating query condition is essential for a meaningful definition of strong resettable witness indistinguishability.

Another point is that our definition has the necessary generality: if a duplicate query occurs, the consistency mechanism handles it; if no duplicate occurs, there is no problem. However, removing the consistency mechanism would only work when duplicates never occur. For example, suppose the prover uses a unique

[11] A duplicate query is a query with the same $(\mathsf{pp}, \mathsf{x}, \mathsf{w}, r')$, not the same $(\mathsf{pp}, \mathsf{x}, \mathsf{w}, l, I)$, because identical query pairs (l, I) to $\mathsf{RandGen}$ may result in different randomness r' since $\ell - |I|$ bits are freshly sampled.

identifier to select randomness for each proof. Then duplicate queries never occur, and no consistency mechanism would be needed. But we do not consider this unique-identifier assumption to be meaningful, because it implicitly assumes a reset-resistant source. To achieve a resettable structure, we assume that part of the protocol is itself resettable. This only reduces the scale of the problem; it does not solve the fundamental issue.

Thus, we adopt the dictionary-based approach to address the consistency challenge, as described in the ideal-world oracle in Figure 2.

Definition 3.2 ((Strong) Resettable Non-Interactive Zero-Knowledge). *Let $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument system for an NP relation $\mathcal{R}$. Π is* (strong) resettable non-interactive zero-knowledge (srNIZK) *if there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ such that for every PPT (strong) resetting adversary $\mathcal{A}$, it holds that:*

$$\mathsf{Adv}^{\mathit{srNIZK}}_{\mathcal{A},\Pi}(\lambda) := \left|\Pr\left[\mathsf{srReal}^{\Pi}_{\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{srIdeal}^{\Pi}_{\mathcal{S},\mathcal{A}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where the experiments $\mathsf{srReal}^{\Pi}_{\mathcal{A}}$ *and* $\mathsf{srIdeal}^{\Pi}_{\mathcal{S},\mathcal{A}}$ *are defined in Figure 2.*

Remark 3.3. The notion of a strong resetting adversary generalizes the standard resettable setting. When the adversary is restricted to queries where either $I = [\ell]$ (all bits are fixed, representing a full reset) or $I = \emptyset$ (all bits are fresh), this is equivalent to a **standard resetting adversary**. Our definition therefore captures and extends the standard resettable security notion.

3.2 Witness-Recovering Attack

We now define a powerful class of attacks in which adversaries not only compromise zero-knowledge properties but also extract sensitive witness information.

Attack Model. We consider NP relations $\mathcal{R}$ that are *efficiently samplable*, i.e., there exists a PPT algorithm Sample that outputs $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$. This ensures the challenger can generate the target pair $(\mathsf{x}^*, \mathsf{w}^*)$ for the attack. The challenger samples $(\mathsf{x}^*, \mathsf{w}^*) \leftarrow \mathsf{Sample}(1^\lambda)$ and provides x^* to the adversary, while keeping w^* secret. The adversary is granted access to an oracle $\mathcal{I}_{\mathcal{P}}$ that returns proofs for the target statement x^* using the internal witness w^* (the oracle is implemented by the challenger, who knows w^*). The adversary may query this oracle multiple times, each time specifying a resetting pattern (l, I). The goal of the adversary is to output any witness w such that $(\mathsf{x}^*, \mathsf{w}) \in \mathcal{R}$.

Another attack model could be considered where the adversary chooses x^* on its own and then interacts with the oracles. However, for such a model to be meaningful, the challenger would need to provide a witness for any statement the adversary chooses, which would require either a trapdoor or restricting the relation to be efficiently samplable on demand. Moreover, even when such sampling is possible, this model is unrealistically strong and not necessary for our purposes.

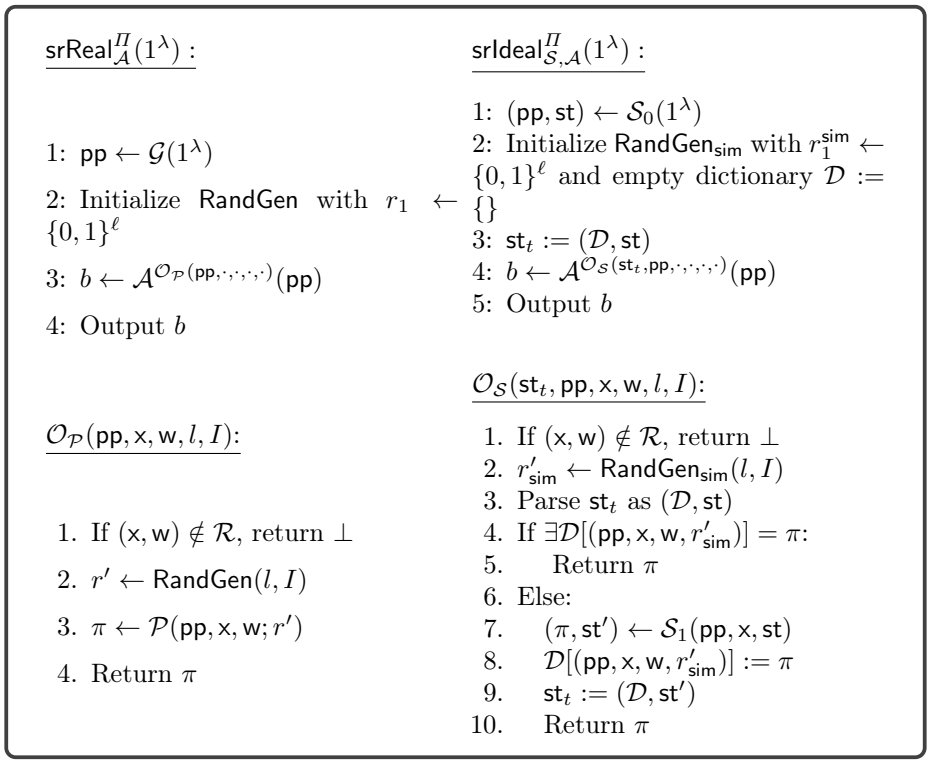

Fig. 2: Experiments for srNIZK. Note that when $\mathcal{A}$ is a standard resetting adversary (only queries with $I = \emptyset$ or $I = [\ell]$), this definition corresponds to rNIZK.

For example, consider a one-way function relation $\mathcal{R} = \{(\mathsf{x},\mathsf{w}) \mid f(\mathsf{w}) = \mathsf{x}\}$: an adversary could simply pick an arbitrary w, compute $\mathsf{x} = f(\mathsf{w})$, and then trivially "recover" w as the witness. Therefore, we present the fixed-target model where x^* is sampled by the challenger.

Auxiliary Input. In many realistic scenarios, the adversary may have access to additional statement-witness pairs $\mathsf{aux} = \{(\mathsf{x}_1,\mathsf{w}_1),\ldots,(\mathsf{x}_k,\mathsf{w}_k)\}$ that are valid for the same relation $\mathcal{R}$. These can arise in two ways. First, they may be leaked from previous protocol executions or obtained through side-channel attacks, which requires additional assumptions about the adversary's capabilities. Second, and more importantly for our purposes, the adversary can generate such pairs itself without any additional assumptions, by simply sampling an arbitrary w and computing x (see Section 4.1 for concrete examples of this approach).

To model this, we define a stronger variant of the attack where the adversary has access to another oracle $\mathcal{O}_{\mathcal{P}}$ that, given input $(\mathsf{x},\mathsf{w},l,I)$, returns a proof with respect to the specified resetting parameters. The adversary can use aux to query this oracle. A crucial point is that the oracles $\mathcal{O}_{\mathcal{P}}$ and $\mathcal{I}_{\mathcal{P}}$ share a

common $\mathsf{RandGen}$ state, so information obtained from $\mathcal{O}_{\mathcal{P}}$ can help the adversary in the games corresponding to $\mathcal{I}_{\mathcal{P}}$. This is fundamentally different from the case where the adversary simply produces a proof on its own using aux. Finally, the adversary, like in the WRA case, outputs a witness corresponding to the target statement.

In order for the attack to be meaningful, the target statement must be different from all statements in aux. Since x^* is freshly sampled by the challenger, the probability that x^* coincides with any $\mathsf{x}_i \in \mathsf{aux}$ is negligible for relations with super-polynomially large statement spaces. Therefore, we do not need to explicitly state $\mathsf{x}^* \notin \mathsf{aux}$ in the definition; this condition holds with overwhelming probability.

Definition 3.4 (Witness-Recovering Attack). *Let $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument system for an NP relation $\mathcal{R}$, and let Sample be an efficient algorithm that outputs $(\mathsf{x}^*, \mathsf{w}^*) \in \mathcal{R}$. We say that adversary $\mathcal{A}$ performs a* witness-recovering attack *against Π if:*

$$\Pr\left[\mathsf{WRA}^{\Pi}_{\mathcal{A}}(1^\lambda) = 1\right] \geq 1 - \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{WRA}^{\Pi}_{\mathcal{A}}$ is defined in Figure 3.

Definition 3.5 (Witness-Recovering Attack with Auxiliary Input). *Let $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument system for an NP relation $\mathcal{R}$, and let Sample be an efficient algorithm that outputs $(\mathsf{x}^*, \mathsf{w}^*) \in \mathcal{R}$. Consider an adversary $\mathcal{A}$ that has auxiliary statement-witness pairs $\mathsf{aux} = \{(\mathsf{x}_1, \mathsf{w}_1), \ldots, (\mathsf{x}_k, \mathsf{w}_k)\}$. We say that $\mathcal{A}$ performs a* witness-recovering attack with auxiliary input *against Π if:*

$$\Pr\left[\mathsf{WRAaux}^{\Pi}_{\mathcal{A}}(1^\lambda) = 1\right] \geq 1 - \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{WRAaux}^{\Pi}_{\mathcal{A}}$ is defined in Figure 3.

Remark 3.6 (Standard vs. Strong Resetting). The above definitions capture witness-recovering attacks in both the standard and strong resettable settings. In the standard resettable setting, the adversary is restricted to queries where either $I = [\ell]$ (a full reset) or $I = \emptyset$ (a fresh execution).

3.3 Relation Between srNIZK and Witness-Recovering Attacks

In this subsection, we show that the srNIZK property guarantees security against both witness-recovering attacks. This is important for two reasons. First, it demonstrates that these attacks are realistic, yet preventing them does not require extra assumptions or unusual properties beyond srNIZK. Second, it clarifies our goal: to achieve security in the resettable setting, we must construct a compiler that achieves the srNIZK property.

$\underline{\mathsf{WRA}^{\Pi}_{\mathcal{A}}(1^\lambda)}$:

1. $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$
2. $(\mathsf{x}^*, \mathsf{w}^*) \leftarrow \mathsf{Sample}(1^\lambda)$
3. $\mathsf{w} \leftarrow \mathcal{A}^{\mathcal{I}_{\mathcal{P}}(\mathsf{pp},\mathsf{x}^*,\mathsf{w}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{x}^*)$
4. If $(\mathsf{x}^*, \mathsf{w}) \in \mathcal{R}$: Output 1, else Output 0

$\underline{\mathsf{WRAaux}^{\Pi}_{\mathcal{A}}(1^\lambda)}$:

1. $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$
2. $(\mathsf{x}^*, \mathsf{w}^*) \leftarrow \mathsf{Sample}(1^\lambda)$
3. $\mathsf{w} \leftarrow \mathcal{A}^{\mathcal{I}_{\mathcal{P}},\mathcal{O}_{\mathcal{P}}(\mathsf{pp},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pp}, \mathsf{x}^*, \mathsf{aux})$
4. If $(\mathsf{x}^*, \mathsf{w}) \in \mathcal{R}$: Output 1, else Output 0

$\underline{\mathcal{I}_{\mathcal{P}}(\mathsf{pp}, \mathsf{x}^*, \mathsf{w}^*, l, I)}$:

1. $r' \leftarrow \mathsf{RandGen}(l, I)$
2. $\pi \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}^*, \mathsf{w}^*; r')$
3. Return π

$\underline{\mathcal{O}_{\mathcal{P}}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, l, I)}$:

1. If $(\mathsf{x}, \mathsf{w}) \notin \mathcal{R}$, return $\perp$
2. $r' \leftarrow \mathsf{RandGen}(l, I)$
3. $\pi \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}; r')$
4. Return π

Fig. 3: Witness-Recovering Attack experiments. The oracles $\mathcal{I}_{\mathcal{P}}$ and $\mathcal{O}_{\mathcal{P}}$ share the same $\mathsf{RandGen}$ instance (a single initialized state with a list of previously generated randomness strings). $\mathcal{I}_{\mathcal{P}}$ only answers queries for the target statement x^* using the target witness w^* (hardcoded by the challenger); the adversary only provides (l, I). $\mathcal{O}_{\mathcal{P}}$ handles arbitrary $(\mathsf{x}, \mathsf{w}, l, I)$ pairs supplied by the adversary

Theorem 3.7 (srNIZK Implies Security Against WRAaux). *Let $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ be an srNIZK argument system for an NP relation $\mathcal{R}$. Then for every PPT adversary $\mathcal{A}$,*

$$\Pr\left[\mathsf{WRAaux}^{\Pi}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

where the experiment $\mathsf{WRAaux}^{\Pi}_{\mathcal{A}}$ is defined in Figure 3.

Proof. Assume for contradiction that there exists a PPT adversary $\mathcal{A}$ that succeeds in WRAaux with non-negligible probability. We construct a distinguisher $\mathcal{D}$ that breaks the srNIZK property.

Construction of $\mathcal{D}$:

- $\mathcal{D}$ samples a random target pair $(\mathsf{x}^*, \mathsf{w}^*) \leftarrow \mathsf{Sample}(1^\lambda)$.
- $\mathcal{D}$ runs $\mathcal{A}$, answering its oracle queries by forwarding them to its own srNIZK oracle $\mathcal{O}$ (which is either the real prover $\mathcal{O}_{\mathcal{P}}$ or the simulator $\mathcal{O}_{\mathcal{S}}$).
- Target queries (l, I) from $\mathcal{A}$ are forwarded as $\mathcal{O}(\mathsf{x}^*, \mathsf{w}^*, l, I)$.
- Auxiliary queries $(\mathsf{x}, \mathsf{w}, l, I)$ from $\mathcal{A}$ are forwarded as $\mathcal{O}(\mathsf{x}, \mathsf{w}, l, I)$.
- When $\mathcal{A}$ outputs a candidate witness w, $\mathcal{D}$ outputs 1 if $(\mathsf{x}^*, \mathsf{w}) \in \mathcal{R}$, otherwise 0.

Analysis:

- In the **real world** ($\mathcal{O} = \mathcal{O}_{\mathcal{P}}$), $\mathcal{D}$ perfectly simulates the WRAaux experiment. By assumption, $\mathcal{A}$ succeeds with non-negligible probability, so $\Pr\left[\mathcal{D}^{\mathcal{O}_{\mathcal{P}}} = 1\right]$ is non-negligible.
- In the **ideal world** ($\mathcal{O} = \mathcal{O}_{\mathcal{S}}$), the simulator $\mathcal{S}_1$ never receives any witness. For target queries, it receives only x^*; for auxiliary queries, it ignores the provided w. Thus, $\mathcal{A}$'s view is independent of w^* except for the public x^*. Consequently, $\mathcal{A}$ can only output a valid witness with negligible probability, so $\Pr\left[\mathcal{D}^{\mathcal{O}_{\mathcal{S}}} = 1\right]$ is negligible.

The non-negligible gap between these probabilities contradicts the srNIZK property. Therefore, no such $\mathcal{A}$ exists.

Theorem 3.7 shows that srNIZK is sufficient to guarantee security against witness-recovering attacks with auxiliary input in the strong resettable setting. This attack is the strongest version we have introduced, where the adversary gains both auxiliary input and the ability to perform strong resetting attacks. We therefore conclude that srNIZK is sufficient for security against all introduced witness-recovering attacks.

4 Resetting Attacks on Non-Interactive Arguments

In this section, we present our resetting attacks in both the standard and strong models defined in Section 3. We specifically focus on Fiat-Shamir NIZKs in the ROM, as they represent a significant class of practical non-interactive zero knowledge systems. The standard resetting attacks are described in Section 4.1, while the strong resetting attacks are detailed in Section 4.2.

4.1 Resetting Attacks

We begin by demonstrating a standard resetting attack on a classic Fiat-Shamir proof system.

Maurer's Proof of Homomorphism Preimage

Definition 4.1 (One-Way Group Homomorphism). *We define a* group-and-homomorphism generator *to be an efficient algorithm* $\mathsf{GroupHomGen}$*, that on input the security parameter* 1^λ*, outputs the description of two groups* $(\mathbb{G}, \star)$ *and* $(\mathbb{H}, \otimes)$*, whose elements are efficiently representable and operations efficiently computable, along with an efficiently computable group homomorphism* $\phi : \mathbb{G} \to \mathbb{H}$*. Furthermore, we assume that* $\mathbb{G}$ *and* $\mathbb{H}$ *are efficiently sampleable.*

We say that $\mathsf{GroupHomGen}$ *is* (average-case) one-way *if for all PPT adversary* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\phi(\mathsf{w}) = \mathsf{x} \;\middle|\; \begin{array}{l}(\mathbb{G}, \mathbb{H}, \phi) \leftarrow \mathsf{GroupHomGen}(1^\lambda) \\ \mathsf{x} \leftarrow \mathbb{H} \\ \mathsf{w} \leftarrow \mathcal{A}(\mathsf{x})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Relation: $\mathcal{R}_{\mathsf{hom\text{-}preim}} := \{(\mathsf{x}, \mathsf{w}) \mid \mathsf{x} = \phi(\mathsf{w})\}$.

Parameters. Challenge set $\mathcal{C} \subseteq \mathbb{N}$ and repetition parameter κ. The random oracle H outputs values in $\mathcal{C}^\kappa$.

Non-Interactive Protocol Π_{hom}:

- $\mathcal{G}(1^\lambda)$: Return $\mathsf{pp} := (\mathbb{G}, \mathbb{H}, \phi) \leftarrow \mathsf{GroupHomGen}(1^\lambda)$.
- $\mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w})$:
  1. For $i \in [\kappa]$, sample $r_i \leftarrow \mathbb{G}$ and compute $a_i := \phi(r_i)$.
  2. Compute $(c_1, \ldots, c_\kappa) := \mathsf{H}(\mathsf{x}, a_1, \ldots, a_\kappa) \in \mathcal{C}^\kappa$.
  3. For $i \in [\kappa]$, compute $z_i := r_i \star \mathsf{w}^{c_i}$.
  4. Output $\pi = (a_i, z_i)_{i \in [\kappa]}$.
- $\mathcal{V}(\mathsf{pp}, \mathsf{x}, \pi)$:
  1. Parse $\pi = (a_i, z_i)_{i \in [\kappa]}$.
  2. Compute $(c_1, \ldots, c_\kappa) := \mathsf{H}(\mathsf{x}, a_1, \ldots, a_\kappa)$.
  3. For each $i \in [\kappa]$, check $\phi(z_i) = a_i \otimes \mathsf{x}^{c_i}$.
  4. Accept if all checks pass; else reject.

Fig. 4: Maurer's non-interactive proof of knowledge for a homomorphism preimage.

Theorem 4.2. *The Maurer protocol Π_{hom} (Figure 4) is not resettable zero-knowledge (rNIZK) for the relation $\mathcal{R}_{\mathsf{hom\text{-}preim}}$, assuming* GroupHomGen *is a one-way group homomorphism generator. Specifically, for any rNIZK simulator $\mathcal{S}$ for Π_{hom}, there exists an efficient meta-reduction $\mathcal{M}$ that uses $\mathcal{S}$ as a black box to break the one-wayness of* GroupHomGen.

Proof. Assume for contradiction that an rNIZK simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ exists for Π_{hom}. We construct a meta-reduction $\mathcal{M}$ that uses $\mathcal{S}$ to break the one-wayness of GroupHomGen.

$\mathcal{M}$ receives a one-wayness challenge $(\mathbb{G}, \mathbb{H}, \phi, \mathsf{x})$, where $\mathsf{x} \leftarrow \mathbb{H}$ is the target element. Its goal is to compute $\mathsf{w} \in \mathbb{G}$ such that $\phi(\mathsf{w}) = \mathsf{x}$. $\mathcal{M}$ proceeds as follows:

1. Run $(\mathsf{pp}, \mathsf{st}) \leftarrow \mathcal{S}_0(1^\lambda)$.
2. **First query (target statement):** Query $\mathcal{S}_1$ on x to obtain $\pi \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{x}, \mathsf{st})$. Parse $\pi = (a_i, z_i)_{i \in [\kappa]}$.
3. **Reset:** Reset $\mathcal{S}_1$ to its previous state st.
4. **Second query (auxiliary statement):** Sample $\mathsf{w}' \leftarrow \mathbb{G}$, compute $\mathsf{x}' := \phi(\mathsf{w}')$, and query $\mathcal{S}_1$ on x' to obtain $\pi' \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{x}', \mathsf{st})$. Parse $\pi' = (a'_i, z'_i)_{i \in [\kappa]}$.
5. **Extraction:** Find an index i where $c_i \neq 0$ and output:

$$\mathsf{w} := \left(z_i \star (z'_i)^{-1} \star (\mathsf{w}')^{c'_i} \right)^{c_i^{-1}}.$$

Why extraction works. Because $\mathcal{S}_1$ was reset to the same state st, its internal randomness is identical across both executions. Hence, the first messages are the same: $a'_i = a_i$ for all $i \in [\kappa]$.

From the verification conditions, we have:

$$\phi(z_i) = a_i \otimes \mathsf{x}^{c_i}, \qquad \phi(z'_i) = a_i \otimes (\mathsf{x}')^{c'_i}.$$

Dividing the first equation by the second (i.e., multiplying by the inverse in $\mathbb{H}$):

$$\phi(z_i) \otimes \phi(z'_i)^{-1} = \mathsf{x}^{c_i} \otimes (\mathsf{x}')^{-c'_i}.$$

By the homomorphism property, $\phi(z_i) \otimes \phi(z'_i)^{-1} = \phi(z_i \star (z'_i)^{-1})$. Thus:

$$\phi(z_i \star (z'_i)^{-1}) = \mathsf{x}^{c_i} \otimes (\mathsf{x}')^{-c'_i}.$$

Now multiply both sides on the right by $(\mathsf{x}')^{c'_i}$ (which equals $\phi(\mathsf{w}')^{c'_i} = \phi((\mathsf{w}')^{c'_i})$):

$$\phi(z_i \star (z'_i)^{-1}) \otimes (\mathsf{x}')^{c'_i} = \mathsf{x}^{c_i}.$$

Since $\phi(z_i \star (z'_i)^{-1}) \otimes (\mathsf{x}')^{c'_i} = \phi(z_i \star (z'_i)^{-1}) \otimes \phi((\mathsf{w}')^{c'_i}) = \phi(z_i \star (z'_i)^{-1} \star (\mathsf{w}')^{c'_i})$, we obtain:

$$\phi\Big(z_i \star (z'_i)^{-1} \star (\mathsf{w}')^{c'_i}\Big) = \mathsf{x}^{c_i}.$$

Now compute $\phi(\mathsf{w})$ using the extracted w:

$$\phi(\mathsf{w}) = \phi\left(\left(z_i \star (z'_i)^{-1} \star (\mathsf{w}')^{c'_i}\right)^{c_i^{-1}}\right) = \phi\Big(z_i \star (z'_i)^{-1} \star (\mathsf{w}')^{c'_i}\Big)^{c_i^{-1}} = (\mathsf{x}^{c_i})^{c_i^{-1}} = \mathsf{x}.$$

Thus $\phi(\mathsf{w}) = \mathsf{x}$, meaning w is a valid witness for the target instance x.

Therefore, $\mathcal{M}$ successfully breaks the one-wayness of $\mathsf{GroupHomGen}$, contradicting the assumption that $\mathcal{S}$ is a PPT simulator for Π_{hom}.

Adaptation to a Witness-Recovering Attack. The above attack can be transformed into a witness-recovering attack (Definition 3.5) against a real prover $\mathcal{P}$ in the resettable setting. The adversary generates an auxiliary pair $(\mathsf{x}', \mathsf{w}')$ on its own, queries $\mathcal{O}_{\mathcal{P}}$ to obtain a proof π' under a fixed randomness state, then resets $\mathcal{I}_{\mathcal{P}}$ to the same randomness to obtain π for the target x^*, and finally extracts w^* using the same algebraic formula. The shared $\mathsf{RandGen}$ state between the two oracles (as specified in Figure 3) is essential for the attack, as it allows the adversary to force randomness reuse across queries.

PIOP-Based Non-Interactive Arguments We now describe witness-recovering attacks against Fiat-Shamir NARGs (FS-NARGs) compiled from PIOPs. These attacks exploit the algebraic structure of PIOPs through resetting capabilities to recover witnesses.

Attack Intuition. In the PIOP model (Definition 2.6), the prover outputs polynomials $\mathbf{p} = (p_{r,t})_{r\in[\rho],t\in[\mathsf{s}(r)]}$, and the verifier executes query algorithm $\mathbf{Q}_{\mathbf{v}}$ on input $(\mathsf{x}; c_1, \ldots, c_\rho)$ to obtain evaluation points $\mathbf{z} = (z_{r,t})_{r\in[\rho],t\in[\mathsf{s}(r)]}$ and corresponding evaluations $\mathbf{y}_{r,t} = p_{r,t}(z_{r,t})$.

Assume there exists a subset of polynomials $\Phi \subseteq \mathbf{p}$ that completely encodes the witness. For simplicity, we consider the case where there is a single polynomial Φ that is evaluated at a single point z per execution, with the prover returning $\Phi(z)$.

When this PIOP is compiled into an FS-NARG, the query points $\mathbf{z}$ are derived via a random oracle H. The attack strategy leverages resetting to force the prover to generate multiple proofs for the same statement-witness pair with different evaluation points. By collecting sufficiently many distinct evaluations of the witness-encoding polynomials, the adversary can solve the resulting system of equations to recover the complete witness.

Technical Challenge and Approach. The fundamental difficulty arises from a contradiction: successful witness extraction requires the prover to produce identical witness-encoding polynomials across sessions while receiving distinct evaluation points. However, under normal operation, identical inputs yield identical random oracle queries, resulting in identical evaluation points. To resolve this, we develop two techniques:

1. **Standard resetting attacks with auxiliary inputs**: Leveraging known instance-witness pairs to recover unknown witnesses.
2. **Strong resetting attacks**: Exploiting partial control over randomness to enforce polynomial reuse while varying evaluation points.

These attacks are formally presented in Theorem 4.3 and Theorem 4.6, respectively.

Theorem 4.3. *Let Π_{FS} be a Fiat-Shamir non-interactive argument system compiled from a* PIOP*. Let $\mathbf{p}$ be the set of all polynomials sent by the prover in the underlying* PIOP*, and let $\Phi \subseteq \mathbf{p}$ denote the subset of polynomials that are evaluated by the verifier via the random oracle H. If the witness of Π_{FS} is fully encoded within Φ, then Π_{FS} admits a witness-recovering attack with auxiliary input (Definition 3.5) in the **standard** resettable setting.*

Proof. Figure 5 outlines the adversary's strategy. The adversary conducts n sessions, each comprising $k+1$ resetting sub-sessions using the same randomness r_i. The adversary first queries $\mathcal{O}_{\mathcal{P}}$ on k known auxiliary pairs $\{(\mathsf{x}_1, \mathsf{w}_1), \ldots, (\mathsf{x}_k, \mathsf{w}_k)\}$, then queries $\mathcal{I}_{\mathcal{P}}$ on the target statement x_{k+1} (provided by the challenger), resulting in $\eta = n \cdot (k+1)$ total queries.

Assume each session yields a single polynomial evaluation: the prover computes oracle polynomial Φ at point z, returning $\Phi(z)$. The polynomial decomposes as:

$$\Phi(X) = \hat{\Phi}(X) + r(X)$$

Input: Let $\mathcal{A}$ be a resetting adversary with auxiliary statement-witness pairs $\mathsf{aux} = \{(\mathsf{x}_1, \mathsf{w}_1), \ldots, (\mathsf{x}_k, \mathsf{w}_k)\}$ and public parameters pp, where $(\mathsf{x}_j, \mathsf{w}_j) \in \mathcal{R}$ for all $j \in [k]$. Let x_{k+1} be the target statement, which is provided to $\mathcal{A}$ by the challenger as per Definition 3.5.

Output: Witness w_{k+1} such that $(\mathsf{x}_{k+1}, \mathsf{w}_{k+1}) \in \mathcal{R}$.

Attack Procedure:

1. For $i \in [n]$:
   (a) For $j \in [k]$:
      - Query $\mathcal{O}_{\mathcal{P}}$ with $(\mathsf{x}_j, \mathsf{w}_j, l = i, I = [\ell])$
      - Receive $\pi_{i,j} \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}_j, \mathsf{w}_j; r_i)$ containing evaluation $\Phi_{i,j}(z_{i,j})$

   (b) Solve for $r_i(X)$ using evaluations $\{\Phi_{i,1}(z_{i,1}), \ldots, \Phi_{i,k}(z_{i,k})\}$
   (c) Query $\mathcal{I}_{\mathcal{P}}$ with $(l = i, I = [\ell])$
   (d) Receive $\pi_{i,k+1} \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}_{k+1}, \mathsf{w}_{k+1}; r_i)$ containing evaluation $\Phi_{i,k+1}(z_{i,k+1})$
2. Solve for w_{k+1} using evaluations $\{\Phi_{1,k+1}(z_{1,k+1}), \ldots, \Phi_{n,k+1}(z_{n,k+1})\}$ and recovered $\{r_1(X), \ldots, r_n(X)\}$
3. Return w_{k+1}

Fig. 5: Resetting attack against PIOP with auxiliary input

where $\hat{\Phi}(X)$ (the *witness-carrying polynomial*) is derived from $(\mathsf{pp}, \mathsf{x}, \mathsf{w})$ and $r(X)$ (the *randomness-carrying polynomial*) is derived from $(\mathsf{pp}; r)$.

Within each session $i \in [n]$, the randomness r_i remains fixed across all $k+1$ sub-sessions due to resetting queries with $I = [\ell]$. Thus:

$$\forall j \in [k+1], \quad r_{i,j}(X) = r_i(X)$$

Moreover, for each fixed $j \in [k+1]$, the same statement-witness pair $(\mathsf{x}_j, \mathsf{w}_j)$ is used across all sessions $i \in [n]$, giving:

$$\forall i \in [n], \quad \hat{\Phi}_{i,j}(X) = \hat{\Phi}_j(X)$$

The oracle polynomial in session i, sub-session j is therefore:

$$\Phi_{i,j}(X) = \hat{\Phi}_j(X) + r_i(X)$$

Crucially, for any two distinct sessions $i \neq i'$, we have $r_i(X) \neq r_{i'}(X)$ with overwhelming probability, as the randomness polynomials r_i are sampled independently. Similarly, for $j \neq j'$, the witness-carrying polynomials $\hat{\Phi}_j(X)$ and $\hat{\Phi}_{j'}(X)$ differ whenever the pairs $(\mathsf{x}_j, \mathsf{w}_j) \neq (\mathsf{x}_{j'}, \mathsf{w}_{j'})$. Therefore, the oracle polynomials $\Phi_{i,j}(X)$ are pairwise distinct across different (i, j) indices. Since evaluation point $z_{i,j}$ is derived via random oracle from the transcript (which includes

$\Phi_{i,j}$), distinct polynomials yield distinct evaluation points with overwhelming probability. This gives the fundamental equation:

$$\forall i \in [n], j \in [k+1], \quad \Phi_{i,j}(z_{i,j}) = \hat{\Phi}_j(z_{i,j}) + r_i(z_{i,j})$$

The attack proceeds in two phases:

1. **Recovering Randomness Polynomials via Auxiliary Inputs**: For each session $i \in [n]$ and auxiliary input $j \in [k]$, the adversary receives evaluation $\Phi_{i,j}(z_{i,j})$ and computes the witness-carrying evaluation $\hat{\Phi}_j(z_{i,j})$ using the known witness w_j. The adversary then computes:

$$\forall i \in [n], j \in [k], \quad r_i(z_{i,j}) = \Phi_{i,j}(z_{i,j}) - \hat{\Phi}_j(z_{i,j})$$

   This provides k distinct evaluations of each randomness polynomial $r_i(X)$. When $k \geq \deg(r_i(X)) + 1$, the adversary can uniquely interpolate the polynomial $r_i(X)$. This is sufficient because the subsequent step only requires evaluating $r_i(X)$ at the point $z_{i,k+1}$.
2. **Recovering Witness via Recovered Randomness**: Using the interpolated polynomials $r_i(X)$ and the received evaluations $\Phi_{i,k+1}(z_{i,k+1})$, the adversary computes:

$$\forall i \in [n], \quad \hat{\Phi}_{k+1}(z_{i,k+1}) = \Phi_{i,k+1}(z_{i,k+1}) - r_i(z_{i,k+1})$$

   This yields n evaluations of the witness-carrying polynomial $\hat{\Phi}_{k+1}(X)$. Since $\hat{\Phi}_{k+1}(X)$ is constructed from the witness vector w_{k+1} of length m (e.g., as a linear combination of basis polynomials with coefficients being witness elements), these evaluations form a system of n equations in m unknowns. When $n \geq m$ and the equations are linearly independent, solving this system uniquely reveals the witness w_{k+1}.

Both degrees are polynomial in λ by PIOP efficiency, making n and k polynomial in λ. The adversary makes $\eta = n \cdot (k+1) = \mathsf{poly}(\lambda)$ oracle queries and performs efficient polynomial interpolation, completing the witness-recovering attack.

Concrete Example with PLONK. We now demonstrate the witness-recovery attack by applying it to PLONK [32], a prominent PIOP-based system. In PLONK, the prover $\mathbf{P}$ takes as input an index i, a public statement $\mathsf{x} = (w_t)_{t \in [d']}$, and a private witness $\mathsf{w} = (w_t)_{t \in [d'+1, 3d]}$. The verifier $\mathbf{V}$ receives x and has oracle access to polynomials output by the indexer $\mathbf{I}$.

During the online phase, the prover computes the public polynomial $\hat{\Phi}_{pub}(X) = \sum_{t\in[d']} w_t \cdot L_t(X)$, and the witness-carrying polynomials:

$$\hat{\Phi}_L(X) = \sum_{t\in[d]} w_t \cdot L_t(X)$$
$$\hat{\Phi}_R(X) = \sum_{t\in[d]} w_{t+d} \cdot L_t(X)$$
$$\hat{\Phi}_O(X) = \sum_{t\in[d]} w_{t+2d} \cdot L_t(X)$$

To ensure zero-knowledge, these witness-carrying polynomials are masked by adding randomness-blinded vanishing polynomial terms:

$$\Phi_L(X) = \hat{\Phi}_L(X) + (r_0 + r_1 X) \cdot Z_H(X)$$
$$\Phi_R(X) = \hat{\Phi}_R(X) + (r_2 + r_3 X) \cdot Z_H(X)$$
$$\Phi_O(X) = \hat{\Phi}_O(X) + (r_4 + r_5 X) \cdot Z_H(X)$$

The prover sends the evaluations $\Phi_L(z), \Phi_R(z), \Phi_O(z)$ to the verifier for a random challenge $z \leftarrow \mathbf{Q_v}(\mathsf{x}; c)$, where c is the output of the random oracle H.

The adversary's goal is to extract a witness w_3 for a target statement x_3. The attack requires two auxiliary statement-witness pairs $\{(\mathsf{x}_1, \mathsf{w}_1), (\mathsf{x}_2, \mathsf{w}_2)\}$. Since the witness length is $n = 3d - d'$, the adversary executes n sessions.

The detailed attack procedure is as follows:

Inputs: Adversary $\mathcal{A}$ has public parameters pp, auxiliary inputs $\{(\mathsf{x}_1, \mathsf{w}_1), (\mathsf{x}_2, \mathsf{w}_2)\}$, and a target statement x_3 (provided by the challenger).

Output: A valid witness w_3 for x_3.

Attack procedure:

1. For each session $i \in [n]$:
   Step 1 – Query auxiliary oracles: The adversary queries $\mathcal{O}_{\mathcal{P}}(\mathsf{pp}, ., ., ., .)$ with $(\mathsf{x}_1, \mathsf{w}_1, l = i, I = [\ell])$ and receives evaluations $\Phi_{1_L}(z_{i,1}), \Phi_{1_R}(z_{i,1}), \Phi_{1_O}(z_{i,1})$. Similarly, the adversary queries $\mathcal{O}_{\mathcal{P}}(\mathsf{pp}, ., ., ., .)$ with $(\mathsf{x}_2, \mathsf{w}_2, l = i, I = [\ell])$ and receives evaluations $\Phi_{2_L}(z_{i,2}), \Phi_{2_R}(z_{i,2}), \Phi_{2_O}(z_{i,2})$.
   Step 2 – Recover session randomness: Using the known witnesses w_1 and w_2, the adversary computes the unmasked values $\hat{\Phi}_{1_L}(z_{i,1})$ and $\hat{\Phi}_{2_L}(z_{i,2})$. It then solves for (r_0, r_1) from the linear system:

$$r_0 + r_1 z_{i,1} = Z_H(z_{i,1})^{-1} \cdot (\Phi_{1_L}(z_{i,1}) - \hat{\Phi}_{1_L}(z_{i,1}))$$
$$r_0 + r_1 z_{i,2} = Z_H(z_{i,2})^{-1} \cdot (\Phi_{2_L}(z_{i,2}) - \hat{\Phi}_{2_L}(z_{i,2}))$$

   Similarly, the adversary recovers (r_2, r_3) from the Φ_R evaluations and (r_4, r_5) from the Φ_O evaluations. The full randomness vector for session i is $r_i = (r_0, r_1, r_2, r_3, r_4, r_5)$.
   Step 3 – Query target oracle: The adversary queries $\mathcal{I}_{\mathcal{P}}(\mathsf{pp}, \mathsf{x}_3, \mathsf{w}_3, ., .)$ with the reset parameters: $(l = i, I = [\ell])$. The adversary receives evaluations $\Phi_L(z_{i,3})$, $\Phi_R(z_{i,3})$, $\Phi_O(z_{i,3})$.

Step 4 – Unmask target evaluations: Using the recovered r_i, the adversary computes the unmasked witness-carrying evaluations:

$$\sum_{t\in[d]} w_t \cdot L_t(z_{i,3}) = \Phi_L(z_{i,3}) - (r_0 + r_1 z_{i,3}) \cdot Z_H(z_{i,3})$$

$$\sum_{t\in[d]} w_{t+d} \cdot L_t(z_{i,3}) = \Phi_R(z_{i,3}) - (r_2 + r_3 z_{i,3}) \cdot Z_H(z_{i,3})$$

$$\sum_{t\in[d]} w_{t+2d} \cdot L_t(z_{i,3}) = \Phi_O(z_{i,3}) - (r_4 + r_5 z_{i,3}) \cdot Z_H(z_{i,3})$$

2. After completing $n = 3d - d'$ sessions, the adversary has obtained n linear equations in terms of the unknown witness variables $(w_t)_{t\in[d'+1,3d]}$.
3. The adversary solves this linear system to recover the witness $\mathsf{w}_3 = (w_t)_{t\in[d'+1,3d]}$.
4. Return w_3.

4.2 Strong Resetting Attacks

Rewindable Black-Box Interactive Arguments Compiled with Salted Fiat-Shamir While resetting generally does not imply rewinding due to limited control over challenges, we identify a specific setting where strong resetting enables simulation of black-box rewinding. This occurs when rewindable interactive arguments are compiled using the *salted* Fiat-Shamir transform, where the salt is treated as fresh prover randomness in each session while other prover randomness remains fixed. By varying the salt, the adversary can induce different random oracle outputs (challenges) for identical prover states, effectively mimicking a rewinding extractor's capability to try multiple challenges on fixed states.

Theorem 4.4. *Let Π be an interactive argument satisfying black-box rewinding knowledge soundness, and let Π_{sFS} be its non-interactive compilation via salted Fiat-Shamir. Then Π_{sFS} admits a witness-recovering attack in the strong resettable model.*

Proof. Since Π satisfies black-box rewinding knowledge soundness, there exists an extractor $\mathcal{E}$ that recovers witnesses by rewinding the prover to fixed states and querying with distinct challenges. In Π_{sFS}, challenges are generated by hashing the transcript with a salt τ treated as prover randomness.

The strong resetting adversary $\mathcal{A}$ operates as follows: it resets the prover to identical internal states while varying only the salt τ_i across sessions. This generates different challenges for identical prover states. By collecting transcripts for distinct salts, $\mathcal{A}$ obtains multiple challenge-response pairs for fixed prover states, effectively simulating the rewinding extractor $\mathcal{E}$ from the original protocol.

$\mathcal{A}$ then invokes $\mathcal{E}$ on these transcripts to recover the witness. Thus, Π_{sFS} admits a witness-recovering attack under strong resetting.

Input: Strong resetting adversary $\mathcal{A}$ with public parameters pp and target statement x^* (provided by the challenger as per Definition 3.4)

Output: Witness w^* such that $(\mathsf{x}^*, \mathsf{w}^*) \in \mathcal{R}$

Attack Procedure:

1. For $i \in [\eta]$:
   (a) Query $\mathcal{I}_{\mathcal{P}}$ with $(l = 1, I = [\ell - |\tau_i|])$
   (b) Receive $\pi_i \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}^*, \mathsf{w}^*; r_I \| \tau_i)$ containing evaluation $\Phi_i(z_i)$
2. Recover witness w^* from evaluations $\{\Phi_1(z_1), \ldots, \Phi_\eta(z_\eta)\}$
3. Return w^*

Fig. 6: Strong resetting attack against PIOP. The query with $l = 1, I = [\ell - |\tau_i|]$ fixes the first $\ell - |\tau_i|$ bits of initialized randomness r_1, where this fixed part is denoted r_I. The remaining bits of r_1 are replaced with the salt τ_i in each session.

Remark 4.5. For non-salted Fiat-Shamir, rewinding and strong resetting are generally not equivalent. However, if varying other prover randomness components effectively functions as salting, similar vulnerabilities may arise in non-salted compilations.

PIOP-Based Non-Interactive Arguments We now revisit the attack from Figure 5 in the strong resettable setting. The key advantage is that the adversary no longer requires auxiliary inputs. Instead, the adversary can exploit fine-grained control over randomness by fixing the main prover randomness while varying only the salt component.

Theorem 4.6. *Let Π_{sFS} be a salted Fiat-Shamir non-interactive argument compiled from a PIOP. Let $\mathbf{p}$ be all polynomials sent by the prover in the underlying PIOP, and $\Phi \subseteq \mathbf{p}$ be the subset of polynomials queried by the verifier via random oracle H. If the witness of Π_{sFS} is fully encoded within Φ, then Π_{sFS} admits a witness-recovering attack in the **strong** resettable model.*

Proof. Figure 6 outlines the attack procedure. The strong resetting adversary $\mathcal{A}$ makes η queries to $\mathcal{I}_{\mathcal{P}}$ with inputs $(\mathsf{x}^*, l = 1, I = [\ell - |\tau_i|])$. This fixes the first $\ell - |\tau_i|$ bits of the initial randomness r_1; this fixed portion is denoted by r_I and remains the same across all sessions, while the remaining $|\tau_i|$ bits come from a fresh salt τ_i for each session. Thus, the full randomness used in session i is $r^{(i)} = r_I \| \tau_i$, where r_I is fixed and τ_i is freshly sampled.

Since $(\mathsf{pp}, \mathsf{x}^*, \mathsf{w}^*, r_I)$ remain identical across all sessions, the witness-encoding polynomials are identical:

$$\Phi_1(X) = \Phi_2(X) = \cdots = \Phi_\eta(X) = \Phi(X)$$

The salt τ_i varies freshly across sessions and, by Definition 2.5, serves as input to the random oracle. This generates distinct evaluation points $z_1, \ldots, z_\eta$. The adversary collects evaluations $\Phi(z_1), \ldots, \Phi(z_\eta)$ of the identical polynomial $\Phi(X)$.

Since the witness w^* is fully encoded in $\Phi(X)$ and the witness vector has length $m = \mathsf{poly}(\lambda)$, the adversary obtains η evaluations $\Phi(z_1), \ldots, \Phi(z_\eta)$ of the same polynomial. When $\eta \geq m$, these evaluations yield a system of η linear equations in the m unknown witness variables. Solving this system uniquely reveals the witness w^*. The attack requires $\eta = \mathsf{poly}(\lambda)$ sessions and succeeds with overwhelming probability.

Concrete Example with PLONK. In this example, we apply the attack illustrated in Figure 6 to the PLONK protocol [32]. The details of the polynomials in PLONK were presented in the previous resetting attack analysis.

Unlike the standard resetting attack, the strong resetting attack does not require auxiliary inputs or nested executions. The adversary simply makes queries with different salts to obtain distinct evaluations of the masked polynomials. Because the evaluation points differ, this yields linearly independent equations. If the number of equations matches the number of unknown values, the system can be solved.

Specifically, for each masked polynomial, the unknown values consist of the witness components of length $3d - d'$ and two components of the randomness vector r_I. Therefore, the total number of sessions required is $\eta = 3d - d' + 2$.

The attack proceeds as follows:

Inputs: The strong resetting adversary $\mathcal{A}$ has public parameters pp and a target statement x^* (provided by the challenger).

Output: A valid witness w^* such that $(\mathsf{x}^*, \mathsf{w}^*) \in \mathcal{R}$.

Attack Procedure: Adversary $\mathcal{A}$ aims to extract the witness w^* corresponding to the statement x^*.

1. For each $i \in [\eta]$:
   (a) Query $\mathcal{I}_{\mathcal{P}}(\mathsf{pp}, \mathsf{x}^*, \mathsf{w}^*, ., .)$ with $(l = 1, I = [\ell - |\tau_i|])$.
   (b) Receive $\pi_i \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}^*, \mathsf{w}^*; r_I \| \tau_i)$ containing the following evaluations of the masked polynomials:

$$\Phi_L(z_i) = \sum_{t \in [d]} w_t \cdot L_t(z_i) + (r_0 + r_1 z_i) \cdot Z_H(z_i)$$
$$\Phi_R(z_i) = \sum_{t \in [d]} w_{t+d} \cdot L_t(z_i) + (r_2 + r_3 z_i) \cdot Z_H(z_i)$$
$$\Phi_O(z_i) = \sum_{t \in [d]} w_{t+2d} \cdot L_t(z_i) + (r_4 + r_5 z_i) \cdot Z_H(z_i)$$

   where the blinding coefficient vector $r_I = (r_0, r_1, r_2, r_3, r_4, r_5)$ remains fixed across all sessions.

2. For each of the masked polynomials Φ_L, Φ_R, and Φ_O, the adversary obtains η linearly independent equations in terms of $\mathsf{w}^* = (w_t)_{t\in[d'+1,3d]}$ of length $3d - d'$ and two components of the blinding vector r_I. Therefore, if $\eta = 3d - d' + 2$, the system of equations is solvable, allowing the adversary to recover the witness w^*.
3. Return w^*.

Does the Strong Resetting Attack Work on Non-Salted Fiat-Shamir NARGs? To explain this, consider the standard compiler introduced in [13] and [39], which constructs a non-interactive argument system by combining a PIOP (Definition 2.6) with Polynomial Commitments (Definition A.2), as described in Section A.3.

In this setting, the adversary can fix the randomness used in the PIOP block ($\mathsf{rnd}_{\mathsf{PIOP}}$) across different sessions, while the randomness associated with PCOM ($\mathsf{rnd}_{\mathsf{PCOM}}$) remains fresh and independent in each session. Consequently, when the prover executes with fixed inputs ($\mathsf{pp}, \mathsf{x}, \mathsf{w}; \mathsf{rnd}_{\mathsf{PIOP}}$) and fresh $\mathsf{rnd}_{\mathsf{PCOM}}$, the oracle polynomials generated in the online phase of the PIOP remain identical, while the evaluation points derived from these commitments vary across sessions. This variation in evaluation points effectively functions as fresh random salt in the Fiat-Shamir transform. The attack therefore proceeds similarly to the salted Fiat-Shamir NARG scenario described previously.

We emphasize that while we have presented strong resetting attacks in the context of salted Fiat-Shamir transforms for generality and simplicity, these attacks also apply to non-salted settings where some randomness component serves a salt-like function. The essential requirement is that varying this component across sessions produces different challenges. Unlike a formal salt, however, this randomness component need not be known to the verifier.

5 Generic Compiler for Strong Resettable NIZK

In this section, we introduce a simple and generic compiler that transforms *any* zero-knowledge NARG into a strong resettable zero-knowledge NARG. The compiler equips the prover with a short random key $r \in \{0,1\}^\lambda$ for a PRF. For each proof on an instance x with public parameters pp and witness w, the prover derives its full-length randomness as $\tilde{r} = \mathsf{F}_r(\mathsf{pp}, \mathsf{x}, \mathsf{w})$. The transformed prover then executes the original prover algorithm with this derived randomness $\tilde{r}$. The full compiler details are provided in Figure 7.

A subtlety arises because the PRF key r is part of the prover's state and is subject to exposure in a strong resetting attack. Standard PRF security assumes the key remains secure without manipulation. To address this, we require a PRF that remains secure even when the adversary can query the function on arbitrary inputs under a key that is subject to reset attacks. We formalize this stronger security notion in Section 5.1.

5.1 PRF Security under Strong Resetting Key Attacks

This security notion extends related-key attacks [8] to a setting where adversaries can partially reset keys by fixing specific bit positions from previous keys while randomizing the rest. Unlike standard RKA that applies transformations to a single key, our model allows reset operations across multiple key generations, making it a distinct security notion.

We say that $\mathcal{F}$ is a *strong resettable-key PRF* if for any PPT adversary, its advantage in the following game is negligible. The challenger picks $b \leftarrow \{0,1\}$. For each query $(x,(l,I))$ from the adversary, the oracle computes $k' \leftarrow \mathsf{RandGen}(l,I)$ and returns $\mathsf{F}_{k'}(x)$ if $b=1$, or $\mathsf{G}(k',x)$ for a random function G if $b=0$. Finally, the adversary outputs b' and wins with advantage $\left|\Pr[b'=b]-\frac{1}{2}\right|$. Without reset queries, this reduces to standard PRF security.

Definition 5.1 (Strong Resettable-key PRF). *A PRF family* $\mathcal{F}=\{\mathsf{F}_k : \{0,1\}^* \to \{0,1\}^\lambda\}_{k\in\{0,1\}^\ell}$ *is a* strong resettable-key PRF (sRK-PRF) *if for all PPT adversaries* $\mathcal{C}$*:*

$$\mathsf{Adv}^{\mathsf{sRK-PRF}}_{\mathcal{C},\mathcal{F}}(\lambda) := \left|\Pr\left[\mathcal{C}^{\mathcal{O}_{\mathsf{real}}}(1^\lambda)=1\right]-\Pr\left[\mathcal{C}^{\mathcal{O}_{\mathsf{rand}}}(1^\lambda)=1\right]\right| \le \mathsf{negl}(\lambda)$$

Where the oracles for each query $(x,(l,I))$ *with* $x\in\{0,1\}^*$, $l\in\mathbb{N}$, $I\subseteq[\ell]$ *operate as follows:*

- $\mathcal{O}_{\mathsf{real}}$*:* $k' \leftarrow \mathsf{RandGen}(l,I)$*; return* $\mathsf{F}_{k'}(x)$
- $\mathcal{O}_{\mathsf{rand}}$*:* $k' \leftarrow \mathsf{RandGen}(l,I)$*; return* $\mathsf{G}(k',x)$ *for random* $\mathsf{G}:\{0,1\}^\ell\times\{0,1\}^*\to\{0,1\}^\lambda$

Theorem 5.2. *Let* $\mathsf{H}:\{0,1\}^*\to\{0,1\}^\lambda$ *be a random oracle. Define the function family* $\mathcal{F}=\{\mathsf{F}_k:\{0,1\}^*\to\{0,1\}^\lambda\}_{k\in\{0,1\}^\ell}$ *where* $\mathsf{F}_k(x)=\mathsf{H}(k\|x)$*. Then* $\mathcal{F}$ *is a* sRK-PRF*.*

Proof. In the random oracle model, the outputs for distinct (k',x) pairs are independent uniform strings, making $\mathcal{O}_{\mathsf{real}}$ and $\mathcal{O}_{\mathsf{rand}}$ perfectly indistinguishable. Formal proof in Section A.4.

5.2 The Compiler: From NIZK to srNIZK

The main theorem establishing the effectiveness of this compiler is as follows:

Theorem 5.3. *Let* $\Pi=(\mathcal{G},\mathcal{P},\mathcal{V})$ *be a* NARG *for a relation* $\mathcal{R}$*. Let* $\tilde{\Pi}=(\tilde{\mathcal{G}},\tilde{\mathcal{P}},\tilde{\mathcal{V}})$ *be the resulting argument system obtained from* Π *via the compiler in Figure 7. If* Π *is a NIZK and* $\mathcal{F}$ *is a secure* sRK-PRF *family, then* $\tilde{\Pi}$ *is a srNIZK.*

Proof. Let $\mathcal{A}$ be a PPT strong resetting adversary with non-negligible advantage $\epsilon(\lambda)$ against $\tilde{\Pi}$. We construct a sequence of hybrid experiments to bound $\mathcal{A}$'s advantage.

Input: A NIZK $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ for relation $\mathcal{R}$ and a sRK-PRF family $\mathcal{F} = \{\mathsf{F}_r : \{0,1\}^* \to \{0,1\}^\lambda\}_{r \in \{0,1\}^\ell}$ where $\ell = \mathsf{poly}(\lambda)$.

Output: A compiled NIZK $\tilde{\Pi} = (\tilde{\mathcal{G}}, \tilde{\mathcal{P}}, \tilde{\mathcal{V}})$ defined as follows:

1. $\tilde{\mathcal{G}}(1^\lambda)$: Identical to $\mathcal{G}(1^\lambda)$.
2. $\tilde{\mathcal{V}}(\mathsf{pp}, \mathsf{x}, \pi)$: Identical to $\mathcal{V}(\mathsf{pp}, \mathsf{x}, \pi)$.
3. $\tilde{\mathcal{P}}(\mathsf{pp}, \mathsf{x}, \mathsf{w}; r)$: On input seed $r \in \{0,1\}^\lambda$:
   (a) Derive randomness: $\tilde{r} \leftarrow \mathsf{F}_r(\mathsf{pp}, \mathsf{x}, \mathsf{w})$[a]
   (b) Emulate original prover: $\pi \leftarrow \mathcal{P}(\mathsf{pp}, \mathsf{x}, \mathsf{w}; \tilde{r})$.
   (c) Output proof π.

[a] This PRF evaluation can be efficiently instantiated in the random oracle model as $\mathsf{H}(\mathsf{pp}, \mathsf{x}, \mathsf{w}, r)$ where $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ is a random oracle.

Fig. 7: Generic compiler for strong resettable NIZK

Hybrid H_0: This is the real experiment $\mathsf{srReal}^{\tilde{\Pi}}_{\mathcal{A}}(1^\lambda)$ where the prover uses the compiled system.
Hybrid H_1: Same as H_0, but we replace the PRF $\mathsf{F}_{r'}(\mathsf{pp}, \mathsf{x}, \mathsf{w})$ with an independent random function $\mathsf{G}(r', \mathsf{pp}, \mathsf{x}, \mathsf{w})$ in all prover queries.
Hybrid H_2: This is the ideal experiment $\mathsf{srIdeal}^{\tilde{\Pi}}_{\tilde{\mathcal{S}}, \mathcal{A}}(1^\lambda)$.

Claim. $|\Pr[\mathsf{H}_0 = 1] - \Pr[\mathsf{H}_1 = 1]| \leq \mathsf{Adv}^{\mathsf{sRK-PRF}}_{\mathcal{C}, \mathcal{F}}(\lambda)$ for some PPT adversary $\mathcal{C}$.

Proof. Construct $\mathcal{C}$ that answers $\mathcal{A}$'s prover queries using its own oracle. If the oracle is $\mathsf{F}_{k'}$, $\mathcal{C}$ simulates H_0; if random, it simulates H_1.

Claim. $|\Pr[\mathsf{H}_1 = 1] - \Pr[\mathsf{H}_2 = 1]| \leq \mathsf{Adv}^{\mathsf{NIZK}}_{\mathcal{B}, \Pi}(\lambda)$ for some PPT adversary $\mathcal{B}$.

Proof. We construct a NIZK adversary $\mathcal{B}$ that distinguishes real proofs from simulated proofs. $\mathcal{B}$ works as follows:

1. $\mathcal{B}$ receives pp from its NIZK challenger and forwards it to $\mathcal{A}$.
2. $\mathcal{B}$ initializes an empty dictionary $\mathcal{D} := \{\}$
3. For each prover query $(\mathsf{x}, \mathsf{w}, l, I)$ from $\mathcal{A}$:
   (a) Compute $r' \leftarrow \mathsf{RandGen}(l, I)$.
   (b) Compute $\tilde{r} \leftarrow \mathsf{G}(r', \mathsf{pp}, \mathsf{x}, \mathsf{w})$.
   (c) If $\mathcal{D}[\tilde{r}]$ exists, retrieve the stored proof $\pi := \mathcal{D}[\tilde{r}]$.
   (d) Else:
      i. Query $\mathcal{B}$'s NIZK challenger on (x, w) to obtain π.
      ii. Store $\mathcal{D}[\tilde{r}] := \pi$.
   (e) Return π to $\mathcal{A}$.
4. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

If $\mathcal{B}$'s challenger provides real proofs, $\mathcal{B}$ simulates H_1; if simulated proofs, it simulates H_2.

Putting everything together:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{srNIZK}}_{\mathcal{A},\tilde{\Pi}}(\lambda) &= |\Pr[\mathsf{H}_0 = 1] - \Pr[\mathsf{H}_2 = 1]| \\ &\leq |\Pr[\mathsf{H}_0 = 1] - \Pr[\mathsf{H}_1 = 1]| + |\Pr[\mathsf{H}_1 = 1] - \Pr[\mathsf{H}_2 = 1]| \\ &\leq \mathsf{Adv}^{\mathsf{sRK-PRF}}_{\mathcal{C},\mathcal{F}}(\lambda) + \mathsf{Adv}^{\mathsf{NIZK}}_{\mathcal{B},\Pi}(\lambda) \\ &\leq \mathsf{negl}(\lambda)\end{aligned}$$

This contradicts the assumption that $\mathcal{A}$ has non-negligible advantage, completing the proof.

5.3 Future Directions

Our compiler achieves rNIZK in the standard model and srNIZK in the random oracle model. It also reduces constructing srNIZK in the standard model to constructing an sRK-PRF. We leave this as an open problem for future research. This direction is particularly important, as constructing an sRK-PRF in the standard model would not only standardize our compiler but also enable the construction of strong resettable public-key encryption using methods similar to those of Yilek [59] and Paterson et al. [49].

Another attractive direction is *implementing strong resetting attacks* against real-world systems—for instance, via fault injection on secure hardware or by exploiting virtual machine snapshots in cloud environments. Such implementations would validate our threat model and uncover practical vulnerabilities in deployed cryptographic systems. We leave experimental validation of these attacks to future work.

Acknowledgments. Matteo Campanelli was supported by the Estonian Research Council grant PRG2531.

References

1. Abdolmaleki, B., Chevalier, C., Ebrahimi, E., Malavolta, G., Vu, Q.H.: On quantum simulation-soundness. IACR Communications in Cryptology **1**(4) (2025)
2. Abdolmaleki, B., Ramacher, S., Slamanig, D.: Lift-and-shift: Obtaining simulation extractable subversion and updatable SNARKs generically. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1987–2005. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417228
3. Abdolmaleki, B., Ramacher, S., Slamanig, D.: Sok: Lifting transformations for simulation extractable subversion and updatable snarks. In: The 3rd ZKProof Workshop (2020)

4. Archambeau, C., Peeters, E., Standaert, F.X., Quisquater, J.J.: Template attacks in principal subspaces. In: Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8. pp. 1–14. Springer (2006)
5. Arun, A., Setty, S., Thaler, J.: Jolt: Snarks for virtual machines via lookups. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 3–33. Springer (2024)
6. Barak, B., Goldreich, O., Goldwasser, S., Lindell, Y.: Resettably-sound zero-knowledge and its applications. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 116–125. IEEE (2001)
7. Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., Yilek, S.: Hedged public-key encryption: How to protect against bad randomness. In: Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15. pp. 232–249. Springer (2009)
8. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 491–506. Springer (2003)
9. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22. pp. 614–629. Springer (2003)
10. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: International conference on the theory and applications of cryptographic techniques. pp. 37–51. Springer (1997)
11. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. Journal of cryptology **14**, 101–119 (2001)
12. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
13. Bünz, B., Fisch, B., Szepieniec, A.: Transparent snarks from dark compilers. In: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39. pp. 677–706. Springer (2020)
14. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing. pp. 235–244 (2000)
15. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International workshop on cryptographic hardware and embedded systems. pp. 13–28. Springer (2002)
16. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 78–96. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_5
17. Chase, M., Lysyanskaya, A.: Simulatable vrfs with applications to multi-theorem nizk. In: Annual International Cryptology Conference. pp. 303–322. Springer (2007)
18. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zksnarks with universal and updatable srs. In: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39. pp. 738–768. Springer (2020)

19. Chiesa, A., Yogev, E.: Building Cryptographic Proofs from Hash Functions (2024), https://github.com/hash-based-snargs-book
20. Cho, C., Ostrovsky, R., Scafuro, A., Visconti, I.: Simultaneously resettable arguments of knowledge. In: Theory of Cryptography Conference. pp. 530–547. Springer (2012)
21. arkworks contributors: `arkworks` zksnark ecosystem (2022), https://arkworks.rs
22. Dao, Q., Grubbs, P.: Spartan and bulletproofs are simulation-extractable (for free!). In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 531–562. Springer, Heidelberg (Apr 2023). https://doi.org/10.1007/978-3-031-30617-4_18
23. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_33
24. Deng, Y., Feng, D., Goyal, V., Lin, D., Sahai, A., Yung, M.: Resettable cryptography in constant rounds–the case of zero knowledge. In: Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17. pp. 390–406. Springer (2011)
25. Deng, Y., Goyal, V., Sahai, A.: Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In: 2009 50th Annual IEEE Symposium on Foundations of Computer Science. pp. 251–260. IEEE (2009)
26. Deng, Y., Lin, D.: Resettable zero knowledge with concurrent soundness in the bare public-key model under standard assumption. In: International Conference on Information Security and Cryptology. pp. 123–137. Springer (2007)
27. Di Crescenzo, G., Persiano, G., Visconti, I.: Constant-round resettable zero knowledge with concurrent soundness in the bare public-key model. In: Annual International Cryptology Conference. pp. 237–253. Springer (2004)
28. Dwork, C., Naor, M.: Zaps and their applications. In: Proceedings 41st Annual Symposium on Foundations of Computer Science. pp. 283–293. IEEE (2000)
29. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. Journal of the ACM (JACM) **51**(6), 851–898 (2004)
30. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34931-7_5
31. Feige, U., Lapidot, D., Shamir, A.: Multiple noninteractive zero knowledge proofs under general assumptions. SIAM Journal on computing **29**(1), 1–28 (1999)
32. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
33. Garay, J.A., MacKenzie, P., Yang, K.: Strengthening zero-knowledge protocols using signatures. Journal of Cryptology **19**(2), 169–209 (2006)
34. Garg, S., Ostrovsky, R., Visconti, I., Wadia, A.: Resettable statistical zero knowledge. In: Theory of Cryptography Conference. pp. 494–511. Springer (2012)
35. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 193–226. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38545-2_7

36. Jain, A., Pandey, O.: Non-malleable zero knowledge: Black-box constructions and definitional relationships. In: Abdalla, M., Prisco, R.D. (eds.) SCN 14. LNCS, vol. 8642, pp. 435–454. Springer, Heidelberg (Sep 2014). https://doi.org/10.1007/978-3-319-10879-7_25
37. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16. pp. 177–194. Springer (2010)
38. Kiyoshima, S.: Resettable statistical zero-knowledge for np. In: Annual International Cryptology Conference. pp. 288–320. Springer (2024)
39. Kohlweiss, M., Pancholi, M., Takahashi, A.: How to compile polynomial iop into simulation-extractable snarks: a modular approach. In: Theory of Cryptography Conference. pp. 486–512. Springer (2023)
40. Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., Shelat, A., Shi, E.: C∅C∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093 (2015)
41. Lipmaa, H.: Plonk is simulation extractable in rom under falsifiable assumptions. In: Theory of Cryptography Conference. pp. 3–36. Springer (2025)
42. Lysyanskaya, A., Rosenbloom, L.N.: Efficient and universally composable non-interactive zero-knowledge proofs of knowledge with security against adaptive corruptions. Cryptology ePrint Archive (2022)
43. Lysyanskaya, A., Rosenbloom, L.N.: Universally composable Σ-protocols in the global random-oracle model. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022, Part I. LNCS, vol. 13747, pp. 203–233. Springer, Heidelberg (Nov 2022). https://doi.org/10.1007/978-3-031-22318-1_8
44. Micali, S.: CS proofs (extended abstracts). In: 35th FOCS. pp. 436–453. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365746
45. Micali, S., Reyzin, L.: Min-round resettable zero-knowledge in the public-key model. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 373–393. Springer (2001)
46. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Annual international cryptology conference. pp. 31–53. Springer (1992)
47. Otto, M.: Fault attacks and countermeasures. Ph.D. thesis, University of Paderborn, Germany (2005)
48. Pass, R., Rosen, A.: New and improved constructions of non-malleable cryptographic protocols. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 533–542. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060670
49. Paterson, K.G., Schuldt, J.C., Sibborn, D.L.: Related randomness attacks for public key encryption. In: Public-Key Cryptography–PKC 2014: 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings 17. pp. 465–482. Springer (2014)
50. Ristenpart, T., Yilek, S.: When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In: Ndss (2010)
51. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th FOCS. pp. 543–553. IEEE Computer Society Press (Oct 1999). https://doi.org/10.1109/SFFCS.1999.814628
52. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE symposium on security and privacy. pp. 459–474. IEEE (2014)

53. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 704–737. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56877-1_25
54. Setty, S., Thaler, J., Wahby, R.: Unlocking the lookup singularity with lasso. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 180–209. Springer (2024)
55. Silur: Zk11: Zk in restricted environments - can state-of-the-art systems run on smartcards? https://www.youtube.com/watch?v=s2BVwde--AM (April 2024), accessed: 2024-05-24
56. de Valence, H.: Merlin: composable proof transcripts for public-coin arguments of knowledge. https://merlin.cool/ (2024)
57. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 299–328. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15985-5_11
58. Yang, G., Duan, S., Wong, D.S., Tan, C.H., Wang, H.: Authenticated key exchange under bad randomness. In: Financial Cryptography and Data Security: 15th International Conference, FC 2011, Gros Islet, St. Lucia, February 28-March 4, 2011, Revised Selected Papers 15. pp. 113–126. Springer (2012)
59. Yilek, S.: Resettable public-key encryption: How to encrypt on a virtual machine. In: Cryptographers' Track at the RSA Conference. pp. 41–56. Springer (2010)
60. Yung, M., Zhao, Y.: Generic and practical resettable zero-knowledge in the bare public-key model. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 129–147. Springer (2007)

Supplementary material hereafter

A Additional Preliminaries

A.1 HVZK for PIOP

We define the notion of Ψ-honest-verifier zero-knowledge for polynomial IOPs.

Definition A.1 (Ψ-Honest-Verifier Zero-Knowledge). *Let* PIOP *be a polynomial IOP for relation $\mathcal{R}_{\mathsf{i}}$. let $\mathbb{D}$ denote the domain of honest polynomial oracle queries. Define an auxiliary query vector $\chi = (\chi_{i,j})_{i\in[\mathsf{r}],j\in[\mathsf{s}(i)]}$ denote an auxiliary query vector to be valid if for every $i \in [\mathsf{r}], j \in [\mathsf{s}(i)], |\chi_{i,j}| \leq \Psi$ and each query in $\chi_{i,j}$ comes from $\mathbb{D}$.* PIOP *is statistical Ψ-honest-verifier zero-knowledge (Ψ-HVZK), if there exists a* PPT *simulator* $\mathbf{S}$ *such that for any distinguisher $\mathcal{B}$, and for all valid auxiliary query vectors χ, it holds that:*

$$\mathsf{Adv}^{\Psi\text{-}\mathit{HVZK}}_{\mathcal{B}}(\lambda) := \left|\Pr\left[\mathit{HVZK}\text{-}0_{\mathcal{B}}(1^\lambda,\chi)=0\right] - \Pr\left[\mathit{HVZK}\text{-}1_{\mathcal{B}}(1^\lambda,\chi)=0\right]\right| \leq \mathsf{negl}(\lambda)$$

where the games HVZK-0 and HVZK-1 are defined in Figure 8. If the operations highlighted in orange are not executed, then we simply say PIOP *satisfies HVZK.*

$\underline{\mathsf{HVZK\text{-}0}_{\mathcal{B}}(1^\lambda, \chi):}$

1: $b \leftarrow \mathcal{B}^{\mathbf{O_0}}(1^\lambda)$
2: **return** b

$\underline{\mathbf{O_0}(\mathsf{i}, \mathsf{x}, \mathsf{w}):}$

1: **if** $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \notin \mathcal{R}_\mathsf{i}$ **then return** $\perp$
2: $(\mathsf{view}, \mathbf{p}) \leftarrow [\mathbf{P}(\mathsf{i}, \mathsf{x}, \mathsf{w}), \mathbf{V}^{\mathbf{I}(\mathsf{i})}(\mathsf{x})]$
3: $(c_1, \ldots, c_\rho, \mathbf{y}) := \mathsf{view}$
4: $\mathbf{z} \leftarrow \mathbf{Q}_\mathbf{v}(\mathsf{x}; c_1, \ldots, c_\rho)$
5: **return** $(\mathsf{view}, \mathbf{p}(\chi), \mathbf{p}(\mathbf{z}))$

$\underline{\mathsf{HVZK\text{-}1}_{\mathcal{B}}(1^\lambda, \chi):}$

1: $b \leftarrow \mathcal{B}^{\mathbf{O_1}}(1^\lambda)$
2: **return** b

$\underline{\mathbf{O_1}(\mathsf{i}, \mathsf{x}, \mathsf{w}):}$

1: **if** $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \notin \mathcal{R}_\mathsf{i}$ **then return** $\perp$
2: $(\mathsf{view}, \mathbf{p}) \leftarrow \mathbf{S}(\mathsf{i}, \mathsf{x})$
3: $(c_1, \ldots, c_\rho, \mathbf{y}) := \mathsf{view}$
4: $\mathbf{z} \leftarrow \mathbf{Q}_\mathbf{v}(\mathsf{x}; c_1, \ldots, c_\rho)$
5: **if** $\exists p_{r,t} \in \mathbf{p}, \deg(p_{r,t}) > \mathsf{d}(|\mathsf{i}|)$ **then return** $\perp$
6: **return** $(\mathsf{view}, \mathbf{p}(\chi), \mathbf{p}(\mathbf{z}))$

Fig. 8: HVZK games for PIOP

A.2 Polynomial Commitments

Definition A.2 (Polynomial Commitment Scheme). *A polynomial commitment scheme denoted by* PCOM *is a tuple of algorithms* $(\mathsf{KGen}, \mathsf{Com}, \mathsf{Eval}, \mathsf{Check})$*:*

1. $\mathsf{ck} \leftarrow \mathsf{KGen}(1^\lambda, D)$*: It takes as input the security parameter* λ *and the maximum degree bound* D *and generates commitment key* ck *as output. We assume* ck *to include a description of the finite field* $\mathbb{F}$*.*
2. $C \leftarrow \mathsf{Com}(\mathsf{ck}, f; r)$*: It takes as input* ck*, the polynomial* $f \in \mathbb{F}^D[X]$*, and using randomness* r*, outputs a commitment* C*. In case the commitment scheme is deterministic, then* $r = \perp$*.*
3. $\pi \leftarrow \mathsf{Eval}(\mathsf{ck}, C, \mathbf{z}, f; r)$*: It takes as input* ck*, the commitment* c*, evaluation point* $\mathbf{z} \in \mathbb{F}$*, the polynomial* f*, and outputs a non-interactive proof of evaluation* π*. The randomness* r *must equal the one previously used in* Com*.*
4. $b \leftarrow \mathsf{Check}(\mathsf{ck}, C, \mathbf{z}, y, \pi)$*: It takes as input the statement* $(\mathsf{ck}, C, \mathbf{z}, y)$*, where* $y \in \mathbb{F}$ *is a claimed polynomial evaluation, and the proof of evaluation* π*, and outputs a bit* b*.*

Typically, PCOM should satisfy *correctness*, *extractability*, and *hiding*. For formal definitions, we refer to, e.g., [18,37]. Additionally, milder properties such as *weak hiding* and *weak evaluation binding* are discussed in [39].

A.3 Compiling Polynomial IOPs

The compiler takes following building blocks as input:

- PIOP$= (\mathbf{I}, \mathbf{P}, \mathbf{V})$
- PCOM$= (\mathsf{KGen}, \mathsf{SCom}, \mathsf{Eval}, \mathsf{Check})$

It then outputs iNARG $\Pi_{\mathsf{H}} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ described in *Figure 9*.

At a high level, the outer proverr $\mathcal{P}$ internally runs a PIOP prover $\mathbf{P}$ in order to obtain polynomials and then commit to them using the polynomial commitment scheme. By hashing the transcript obtained up to the i-th round, $\mathcal{P}$ obtains PIOP challenge c_i, which is fed to $\mathbf{P}$ to advance to the next round. When the PIOP prover terminates, $\mathcal{P}$ runs a PIOP query algorithm to sample query points $\mathbf{z}$ and evaluates the polynomial oracles at these points. Finally, $\mathcal{P}$ produces evaluation proofs to guarantee that polynomial evaluations are done correctly with respect to commitments produced in earlier rounds.

A.4 Proof of Theorem 5.2

Let $\mathcal{C}$ be any PPT adversary in the sRKA security game, making at most $q = \mathsf{poly}(\lambda)$ queries to its oracle. We must prove that:

$$\mathsf{Adv}^{\mathsf{sRK-PRF}}_{\mathcal{C},\mathcal{F}}(\lambda) := \left|\Pr\left[\mathcal{C}^{\mathcal{O}_{\mathsf{real}}}(1^\lambda) = 1\right] - \Pr\left[\mathcal{C}^{\mathcal{O}_{\mathsf{rand}}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

Following the Definition 5.1, the oracles are defined as:

- $\mathcal{O}_{\mathsf{real}}(x, l, I)$: $k' \leftarrow \mathsf{RandGen}(l, I)$; Return $\mathsf{H}(k' \| x)$
- $\mathcal{O}_{\mathsf{rand}}(x, l, I)$: $k' \leftarrow \mathsf{RandGen}(l, I)$; Return $\mathsf{G}(k', x)$.

We define one hybrid $\mathbf{H}$ that on query (x, l, I) operates as:

1. Maintain table T
2. $k' \leftarrow \mathsf{RandGen}(l, I)$
3. If $T[k'][x]$ is undefined, sample $T[k'][x] \xleftarrow{\$} \{0,1\}^\lambda$
4. Return $T[k'][x]$

We now analyze the relationships between these experiments:

Claim. $\mathbf{H}$ perfectly simulates $\mathcal{O}\mathsf{real}$.

Proof. In the random oracle model, H is implemented via lazy sampling. Define the random oracle implementation that maintains a table T_{H} and on input s returns $T_{\mathsf{H}}[s]$ if defined, else samples $T_{\mathsf{H}}[s] \xleftarrow{\$} \{0,1\}^\lambda$. If we set $T[k'][x] = T_{\mathsf{H}}[k' \| x]$ for all (k', x), then $\mathbf{H}$ behaves identically to $\mathcal{O}\mathsf{real}$. Both initialize $\mathsf{RandGen}$ with independent uniform keys, and both use the same lazy sampling process for generating outputs.

Claim. $\mathbf{H}$ perfectly simulates $\mathcal{O}\mathsf{rand}$.

Proof. The ideal oracle $\mathcal{O}\mathsf{rand}$ maintains independent random functions $\mathsf{G}_{k'}$ for each key k'. This is implemented by maintaining a table T_{rand} where for each new (k', x), we sample $T_{\mathsf{rand}}[k'][x] \xleftarrow{\$} \{0,1\}^\lambda$. If we set $T[k'][x] = T_{\mathsf{rand}}[k'][x]$ for all (k', x), then $\mathbf{H}$ behaves identically to $\mathcal{O}_{\mathsf{rand}}$. Both initialize $\mathsf{RandGen}$ with independent uniform keys, and both use the same process of assigning fresh random values to new (k', x) pairs.

Since the hybrid experiment perfectly simulates both the real and random worlds, the adversary's advantage is zero.

- $\mathcal{G}(1^\lambda)$
  1. Let $D = \max_{(\mathsf{i},\cdot,\cdot)\in R_\lambda}^{d(|\mathsf{i}|)}$.
  2. Run $\mathsf{ck} \leftarrow \mathsf{PCOM.KGen}(1^\lambda, D)$ and output $\mathsf{srs} := \mathsf{ck}$.
- $\mathcal{I}(\mathsf{i}, \mathsf{srs})$
  1. Compute $(p_{0,1}, \ldots, p_{0,s(0)}) \leftarrow \mathsf{PIOP.I}(\mathsf{i})$ where each polynomial $p_{0,t}$, for $t \in [s(0)]$ is of degree at most $d(|\mathsf{i}|)$.
  2. For $t \in [s(r)]$: Compute $C_{0,t} \leftarrow \mathsf{PCOM.Com}(\mathsf{ck}, p_{0,t}; \perp)$ with empty randomness.
  3. Set $\mathsf{ipk} := (\mathsf{i}, \mathsf{srs}, (C_{0,1}, \ldots, C_{0,s(0)}), (p_{0,1}, \ldots, p_{0,s(0)}))$, and $\mathsf{ivk} := (\mathsf{i}, \mathsf{srs}, (C_{0,1}, \ldots, C_{0,s(0)})$.
- $\mathcal{P}^{\mathsf{H}}(\mathsf{ipk}, \mathsf{x}, \mathsf{w})$
  1. Initialize round $r := 1$, transcript $\pi := \varnothing$, polynomials $p := \varnothing$, evaluation proofs $\pi_{\mathsf{PCOM}} := \varnothing$, initial state $\mathsf{st}_p := (\mathsf{i}, \mathsf{x}, \mathsf{w})$.
  2. **Online phase:** While $r \leq \rho$,
     (a) Compute $(p_{r,1}, \ldots, p_{r,s(r)}, \mathsf{st}_p) \leftarrow \mathsf{PIOP.P}(\mathsf{st}_p; \mathsf{rnd}_{\mathsf{PIOP}})$.
     (b) For $t \in [s(r)]$: Compute $C_{r,t} \leftarrow \mathsf{PCOM.Com}(\mathsf{ck}, p_{r,t}; \mathsf{rnd}_{\mathsf{PCOM}})$.
     (c) Update $\pi := \pi \| (C_{r,1}, \ldots, C_{r,s(r))}$, $\mathbf{p} := \mathbf{p} \| (p_{r,1}, \ldots, p_{r,s(r)})$.
     (d) Derive challenge $c_r \leftarrow \mathsf{H}(\mathsf{srs}, \mathsf{i}, \mathsf{x}, \pi)$.
     (e) Update $\pi := \pi || c_r$, and $r := r + 1$.
  3. **Query phase:** if $r > \rho$,
     (a) Compute $\mathbf{z} \leftarrow \mathsf{PIOP.Q}_{\mathbf{v}}(\mathsf{x}; c_1, \ldots, c_\rho)$
     (b) For each $r \in [0, \rho], t \in [s(r)], u \in [s(r, t)]$: Compute $\pi_{r,t,u} \leftarrow \mathsf{PCOM.Eval}(\mathsf{ck}, C_{r,t}, \mathbf{z}_{r,t,u}, p_{r,t}; \mathsf{rnd}_{\mathsf{PCOM}})$ and $\pi_{\mathsf{PCOM}} := \pi_{\mathsf{PCOM}} \| \pi_{r,t,u}$.
     (c) $\mathbf{y} := \mathbf{p}(\mathbf{z})$.
     (d) Update $\pi := \pi \| (\mathbf{y}, \pi_{\mathsf{PCOM}})$.
  4. Output π.
- $\mathcal{V}^{\mathsf{H}}(\mathsf{ivk}, \mathsf{x}, \pi)$
  1. Initialize round $r := 1$.
  2. Parse $\pi := (C_{1,1}, \ldots, c_\rho, \mathbf{y}, \pi_{\mathsf{PCOM}})$.
  3. **Online phase:** While $r \leq \rho$,
     (a) Define $a_r := (C_{1,1}, \ldots, C_{r,s(r)})$.
     (b) Derive challenge $c'_r \leftarrow \mathsf{H}(\mathsf{srs}, \mathsf{i}, \mathsf{x}, a_r)$. If $c'_r \neq c_r$, abort by outputting 0.
     (c) Update round $r := r + 1$.
  4. **Query phase:** Compute $\mathbf{z} \leftarrow \mathbf{Q}_{\mathbf{v}}(\mathsf{x}; c_1, \ldots, c_\rho)$.
  5. **Decision phase:** Output 1 if $\mathsf{PCOM.Check}(\mathsf{ck}, C, \mathbf{z}, \mathbf{y}, \pi_{\mathsf{PCOM}}) = 1$ and $\mathbf{D}_{\mathbf{v}}(\mathsf{x}, \mathbf{y}; c_1, \ldots, c_\rho) = 1$.

Fig. 9: Compilation of PIOPs into NARGs using PCOM