Adaptively-Sound Succinct Arguments for NP from Indistinguishability Obfuscation

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

David J. Wu
UT Austin
dwu4@cs.utexas.edu

Abstract

A succinct non-interactive argument (SNARG) for NP allows a prover to convince a verifier that an NP statement x is true with a proof of size $o(|x| + |w|)$, where w is the associated NP witness. A SNARG satisfies adaptive soundness if the malicious prover can choose the statement to prove *after* seeing the scheme parameters. In this work, we provide the first adaptively-sound SNARG for NP in the plain model assuming sub-exponentially-hard indistinguishability obfuscation, sub-exponentially-hard one-way functions, and either the (polynomial) hardness of the discrete log assumption or the (polynomial) hardness of factoring. This gives the first adaptively-sound SNARG for NP from *falsifiable* assumptions. All previous SNARGs for NP in the plain model either relied on non-falsifiable cryptographic assumptions or satisfied a weak notion of non-adaptive soundness (where the adversary has to choose the statement it proves before seeing the scheme parameters).

1 Introduction

A succinct non-interactive argument (SNARG) for NP allows a (computationally-bounded) prover to convince a verifier that an NP statement x is true with a proof whose size scales with $o(|x| + |w|)$, where w is the associated NP witness. While succinct arguments for NP can be constructed unconditionally in the *random oracle* model [Kil92, Mic94], the same is not true in the plain model. In the plain model, we assume the prover and the verifier have access to a common reference string (CRS). SNARGs for NP that do not have a common reference string are unlikely to exist [BP04, Wee05].

Existing constructions of SNARGs in the plain model either rely on strong, *non-falsifiable* cryptographic assumptions [Gro10, BCCT12, DFH12, Lip13, GGPR13, BCI+13, BCPR14, BISW17, BCC+17, BISW18, ACL+22, CLM23] or only support subsets of NP [KR09, KP16, BHK17, KPY19, JKKZ21, KVZ21, CJJ21a, CJJ21b, WW22, JJ22, KLV23, BBK+23, CGJ+23]. To date, the only exception is the construction of Sahai and Waters of a SNARG for NP from indistinguishability obfuscation ($i\mathcal{O}$) and one-way functions [SW14]. While the existence of $i\mathcal{O}$ is itself not a falsifiable assumption, recent work has shown how to base $i\mathcal{O}$ on a collection of falsifiable assumptions [JLS21, JLS22]. However, the Sahai-Waters construction only achieves a weak notion of *non-adaptive* soundness, where soundness only holds against an adversary that declares its false statement *before* seeing the common reference string. The more natural notion of security is adaptive soundness which allows the adversary to choose its statement after seeing the scheme parameters. Achieving adaptively-sound SNARGs for NP from standard falsifiable cryptographic assumptions has proven to be an elusive goal and any such construction must overcome black-box separations [GW11, CGKS23].

This work. In this work, we construct the first adaptively-sound SNARG for NP assuming the existence of a sub-exponentially-hard $i\mathcal{O}$ scheme,[1] a sub-exponentially-hard one-way function, and polynomial hardness of a standard number-theoretic assumption (e.g., the hardness of discrete log or the hardness of factoring). In conjunction with the results basing $i\mathcal{O}$ on falsifiable assumptions [JLS21, JLS22], this yields the first adaptively-sound SNARG for NP from falsifiable assumptions. We summarize our construction below and provide a technical overview of our construction in Section 1.1.

Theorem 1.1 (Informal)**.** *Assuming (1) either the polynomial hardness of computing discrete logs in a prime-order group or the polynomial hardness of factoring, (2) the existence of a sub-exponentially-secure indistinguishability obfuscation scheme for Boolean circuits, and (3) the existence of a sub-exponentially-secure one-way function, there exists an adaptively-sound SNARG for* NP*. Specifically, we construct a SNARG for* NP *with the following properties:*

- **Preprocessing SNARG:** *Similar to [SW14], we work in the preprocessing model where there is a large CRS that depends on the Boolean circuit* $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ *that computes the* NP *relation (i.e., n is the statement length and h is the witness length for the* NP *relation). The size of the common reference string is* $\mathsf{poly}(\lambda + |C|)$*, where λ denotes a security parameter.*
- **Proof size:** *The size of the proof is* $\mathsf{poly}(\lambda)$*.*

Moreover, the SNARG satisfies perfect zero-knowledge.

The Gentry-Wichs separation. A classic result of Gentry and Wichs [GW11] rules out adaptively-sound SNARGs for NP whose security can be based on a black-box reduction to a falsifiable cryptographic assumption (and in some settings, even with a CRS whose size grows with the size of the NP relation [CGKS23]). A critical assumption in the Gentry-Wichs separation is that the running time of the SNARG security reduction is *insufficient* to decide the associated NP language. The existing reductions of $i\mathcal{O}$ to falsifiable assumptions [JLS21, JLS22] run in time that is *exponential* in the input length of the obfuscated program. In our construction, the CRS contains obfuscated programs that take the statement *and* the witness as input. Correspondingly, our security reduction runs in time that is sufficient to decide membership in the underlying NP language. For this reason, the Gentry-Wichs separation does not apply to our construction. As was noted in [JJ22], this caveat is also true for the original Sahai-Waters SNARG based on $i\mathcal{O}$, so the Gentry-Wichs separation also does not say anything about the Sahai-Waters construction.

One interpretation of the Gentry-Wichs separation is that to build adaptively-sound SNARGs for NP from falsifiable assumptions, we need *some* form of sub-exponential hardness (and complexity leveraging). In this case, either the size of the CRS or the size of the proof must grow with the size of the statement or witness. The challenge then is to offload the complexity leveraging overhead in the construction entirely to the CRS in order to keep the proofs succinct. This is precisely what our new approach achieves.

1.1 Technical Overview

We begin by describing a variant of the Sahai-Waters SNARG for NP based on indistinguishability obfuscation (and one-way functions) [SW14]. We work with the language of Boolean circuit satisfiability, where the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is fixed ahead of time (i.e., as part of the CRS). A statement is a string $x \in \{0,1\}^n$ and the statement is true if there exists a witness $w \in \{0,1\}^h$ such that $C(x,w) = 1$.

[1]The notion of sub-exponential hardness we use in this work says that for a security parameter λ, a *polynomial-time* adversary cannot break the assumption except with probability at most $2^{-\lambda^c}$ for some positive constant $c < 1$.

In addition to $i\mathcal{O}$, the Sahai-Waters construction requires a one-way function $f: \mathcal{Y} \to \mathcal{Z}$ and a puncturable pseudorandom function (PRF) F with domain $\{0,1\}^n$ and output space $\mathcal{Y}$. A puncturable PRF [BW13, KPTZ13, BGI14] is a pseudorandom function where the PRF key k can be "punctured" at an input point x. We write $k^{(x)}$ to denote a key punctured at input x. The punctured key can be used to evaluate the PRF $\mathsf{F}(k, \cdot)$ on all inputs $x' \neq x$ (i.e., $\mathsf{F}(k, x') = \mathsf{F}(k^{(x)}, x')$ for all $x' \neq x$). Moreover, the value of the PRF $\mathsf{F}(k, x)$ should remain pseudorandom even given the punctured key $k^{(x)}$.

The Sahai-Waters construction. In the Sahai-Waters construction, the common reference string consists of two obfuscated programs: a program Prove for generating proofs and a Verify program for validating proofs. Here, we describe a variant where the Verify program is replaced by an "instance-generator" $\mathsf{GenInst}$, which will be a useful stepping stone to our construction. The two programs are defined as follows:

```
Prove(x, w):
  • On input the statement $x \in \{0,1\}^n$ and the witness $w \in \{0,1\}^h$, if $C(x, w) = 1$, output $\pi = \mathsf{F}(k, x)$.
  • Otherwise, output $\bot$.
```

```
GenInst(x):
  • On input the statement $x \in \{0,1\}^n$, output $f(\mathsf{F}(k, x))$.
```

The CRS contains obfuscations of the Prove and $\mathsf{GenInst}$ programs. To construct a proof for a statement x and witness w, the prover simply runs the (obfuscated) Prove program on input (x, w) and obtain a proof π. To check the proof π on statement x, the verifier runs the (obfuscated) $\mathsf{GenInst}$ program on input x to obtain a challenge $z \in \mathcal{Z}$. The verifier then checks that $f(\pi) = z$. In the original Sahai-Waters construction [SW14, §5.5], the CRS contained a Verify program that combines $\mathsf{GenInst}$ and the verification check $f(\pi) = z$. We separate it out because it will be helpful for understanding our SNARG construction.

At a high-level, we can view the $\mathsf{GenInst}$ as generating a challenge (for the one-way function) for each statement x and the Prove program as generating solutions to those challenges. Informally, soundness follows from the fact that the Prove program only solves instances associated with the true statements $x \in \mathcal{L}_C$, where we define $\mathcal{L}_C = \left\{x \in \{0,1\}^n \mid \exists w \in \{0,1\}^h : C(x, w) = 1\right\}$. In order to construct a proof for a false statement $x^* \notin \mathcal{L}_C$, the adversary essentially has to invert the one-way function f on a (pseudorandom) input $\mathsf{F}(k, x^*)$, which should be hard. This is formalized via the following hybrid argument:

- In the non-adaptive soundness game, the adversary first commits to the false statement $x^* \notin \mathcal{L}_C$. Now, we can construct an equivalent pair of programs that use a punctured key $k^{(x^*)}$ in place of k. In the case of the Prove program, since $x^* \notin \mathcal{L}_C$, the program never needs to evaluate $\mathsf{F}(k, x^*)$. Thus, we can replace the obfuscated programs in the CRS with obfuscations of the following programs (which compute identical functionality as the previous programs):

  ```
  Prove(x, w):
    – If $C(x, w) = 1$, output $\pi = \mathsf{F}(k^{(x^*)}, x)$.
    – Otherwise, output $\bot$.
  ```

  ```
  GenInst(x, π):
    – If $x^* = x$ output $f(\mathsf{F}(k, x^*))$.
    – If $x^* \neq x$, output $f(\mathsf{F}(k^{(x^*)}, x))$.
  ```

- By puncturing security of the PRF, the value of $\mathsf{F}(k, x^*)$ is pseudorandom even given the punctured key $k^{(x^*)}$. More precisely, the distribution of $f(\mathsf{F}(k, x^*))$ is computationally indistinguishable from the distribution of $z = f(y)$ where $y \xleftarrow{\text{R}} \mathcal{Y}$ is a uniformly-random string sampled from the codomain of the PRF. This means that the following two programs are computationally indistinguishable:

  ```
  Prove(x, w):
    – If $C(x, w) = 1$, output $\pi = \mathsf{F}(k^{(x^*)}, x)$.
    – Otherwise, output $\bot$.
  ```

  ```
  GenInst(x, π):
    – If $x^* = x$, output $z$.
    – If $x^* \neq x$, output $f(\mathsf{F}(k^{(x^*)}, x))$.
  ```

In this experiment, the only way the adversary produces a valid proof for the statement x^* is by outputting a π such that $f(\pi) = z$. Thus, to come up with a valid proof for x^*, the adversary must invert f at a random $z = f(y)$. This is computationally infeasible by security of the one-way function, and so we conclude that an efficient adversary is unable to produce a valid proof for x^*.

The challenge of adaptivity. The reduction strategy described above critically relies on knowing the false statement $x^* \in \{0,1\}^n$ in advance. Indeed, the first step in the security reduction is to replace the PRF key k with a key punctured at x^*. Puncturing the PRF at x^* enables us to program a random challenge (for the one-way function) at x^*. This ensures that any successful adversary that comes up with a proof for x^* must be able to invert the one-way function. This strategy breaks down if the statement x^* is *not* known in advance (i.e., the setting of adaptive soundness). Notably, it is not clear *where* to puncture the PRF key and embed the challenge for the one-way function.

Why not complexity leverage? One possible way to argue adaptive soundness is to complexity leverage and *guess* the statement x^* and rely on sub-exponential hardness. Here, the security reduction would guess a random statement $x^* \xleftarrow{\text{R}} \{0,1\}^n$ and then apply the previous reduction as if the adversary had committed to x^*. The security reduction succeeds if the guess was correct, which occurs with probability $1/2^n$. In turn, we rely on sub-exponential hardness of the underlying cryptographic primitives and assume that the advantage of any computationally-bounded adversary breaking each primitive should be much smaller than even $1/2^n$. In particular, this means that the probability that an efficient adversary succeeds in inverting the one-way function f must also be smaller than $1/2^n$. But this means the length of the preimage of the one-way function must be at least n, which is the statement size (otherwise, the preimage can be guessed with probability better than $1/2^n$). Since the proof is precisely the preimage, this means the proof size is now at least $\Omega(n)$. Thus, while the standard complexity-leveraging strategy suffices to prove adaptive soundness, the resulting proof system is no longer succinct.[2]

Starting point: embedding a second challenge. To build an adaptively-sound SNARG, we will need a different proof technique (and construction). Our starting point is to modify the Sahai-Waters variant described above by having the GenInst program output *two* independent challenges $(z_{x,0}, z_{x,1})$ for each statement $x \in \{0,1\}^n$. In the modified scheme, the verifier accepts if the prover solves *either* challenge: namely, a proof $\pi = (b, y)$ is valid if $b \in \{0,1\}$ and $f(y) = z_{x,b}$. The critical property is that the Prove program will only output a solution to *one* of the challenges. In more detail, we consider three different (puncturable) PRFs: $\mathsf{F}_{\mathsf{sel}}$, F_0, and F_1. The selector PRF $\mathsf{F}_{\mathsf{sel}}$ takes as input a statement $x \in \{0,1\}^n$ and outputs a selection bit $b \in \{0,1\}$. The PRFs F_0 and F_1 takes as input a statement $x \in \{0,1\}^n$ and outputs a value $y \in \mathcal{Y}$ in the domain of the one-way function. We now define the Prove and GenInst programs as follows:

```
Prove($x, w$):
  • If $C(x, w) = 1$, compute $b = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and output
    $\pi = (b, \mathsf{F}_b(k_b, x))$.
  • Otherwise, output $\perp$.
```

```
GenInst($x$):
  • Compute $y_{x,b} = \mathsf{F}_b(k_b, x))$ for $b \in \{0,1\}$.
  • Output $(z_{x,0}, z_{x,1}) = (f(y_{x,0}), f(y_{x,1}))$.
```

[2]Some works (e.g., [JJ22]) only require the size of the proof be sublinear in the length of the witness, and allow for a polynomial dependence in the length of the statement. Under this weaker notion of succinctness, the Sahai-Waters construction would be adaptively-sound via complexity leveraging. Our focus in this work is the more stringent and common notion of succinctness that require the proof size to be sublinear in *both* the statement and the witness. This was the notion studied in [GW11].

Before proceeding, we provide some brief intuition for why having two challenges is beneficial for arguing adaptive security. In the non-adaptive soundness analysis above, the adversary has to pre-commit to the statement x^* and the security reduction then embeds the one-way function challenge at x^* in the $\mathsf{GenInst}$ program. When considering adaptive security, the security reduction does *not* know which statement the adversary will choose so it is not clear where to embed the one-way function challenge (and guessing does not work for the reasons outlined above).

One possible approach is to change $\mathsf{GenInst}$ to output the one-way function challenge on *every* statement $x \in \{0,1\}^n$ in the security proof.[3] Then, an adversary that outputs a proof for *any* false statement would successfully invert the one-way function. However, this leads to a correctness issue: we still need to be able to generate proofs for true statements (i.e., the Prove program needs to give out preimages for the challenges associated with true statements). This is no longer possible if $\mathsf{GenInst}$ outputs a one-way function challenge for which the Prove algorithm does not know a corresponding preimage on every input $x \in \{0,1\}^n$. Ideally, we would only embed the challenge instances on inputs $x \notin \mathcal{L}_C$. However, $\mathsf{GenInst}$ cannot decide the language itself to determine whether it should output a challenge instance or one with a known preimage.

Our solution is to associate two challenges with every x. One of the challenges (as determined by $\mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$) will be sampled with a known preimage while the other can be arbitrary (and will be used to embed a challenge instance in the reduction). This way, the Prove program still has the capability to produce a proof for *every* statement, while simultaneously ensuring that an adversary that succeeds on *any* instance x breaks security of the one-way function. This is conceptually similar to other "two-challenge" approaches used to argue adaptive security for digital signatures [KW03], broadcast encryption [GW09], or registered attribute-based encryption [FWW23], albeit in the random oracle model. In fact, as we discuss in Remark 4.21, our techniques can be used to "instantiate" the random oracle in the Katz-Wang signature scheme with an obfuscated PRF to obtain a provably-secure variant of their scheme in the *plain* model.

Proving adaptive security. Our proof of adaptive security proceeds in a sequence of hybrid experiments. Our reduction still relies on complexity leveraging (and specifically, an exponential number of hybrids), but the parameter blowup from complexity leveraging only factors into the CRS size, and *not* the proof size. We now survey our main hybrids and refer to the proof of Theorem 4.3 for the full details.

- Hyb_0: This corresponds to the real adaptive soundness game. In this game, the adversary is considered successful if it outputs a false statement $x \notin \mathcal{L}_C$ along with a proof $\pi = (b, y)$ where $f(y) = z_{x,b}$ and $(z_{x,0}, z_{x,1}) \leftarrow \mathsf{GenInst}(x)$. Notably, there is no requirement on the value of b (i.e., the adversary wins if it can invert f on *either* $z_{x,0}$ or $z_{x,1}$).

- Hyb_1: This is the same experiment as before, except we consider the adversary successful only if it outputs a false statement $x \notin \mathcal{L}_C$ and a proof $\pi = (b, y)$ where $b \neq \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$.

  We claim that this can only reduce the adversary's advantage in winning the game by a factor of 2. This is because for all $x \notin \mathcal{L}_C$, the value of $\mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ is hidden from the adversary (i.e., never computed by the Prove algorithm). If the adversary's advantage decreased by more than a factor of 2 between Hyb_0 and Hyb_1, then the adversary is able to predict the value of $\mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ with probability better than $1/2$, thereby breaking security of the PRF.

  Formally, we argue this by considering 2^n possible experiments (for each possible statement $x \in \{0,1\}^n$ the adversary could output). By relying on (sub-exponential) security of $i\mathcal{O}$ and the puncturable PRF, we can show that in each of these experiments, the adversary's success probability is always

[3]Ignore for a moment that $\mathsf{GenInst}$ is technically supposed to output independent values on each input x.

within a factor of 2 of the adversary's success probability in Hyb_0 (up to negligible differences). We refer to Lemma 4.4 for the full details. Note that even though we rely on complexity leveraging and sub-exponential-hardness, we are only complexity leveraging on the $i\mathcal{O}$ scheme and the puncturable PRF, *not* on the security of the one-way function. This means the cost of complexity leveraging is only incurred in the CRS size (now larger by a $\mathrm{poly}(n)$ factor), but not in the proof size (whose length is governed by the preimage length for the one-way function).

At this point in the proof, we are in a conceptually-similar end-point as in the non-adaptive soundness proof of Sahai-Waters. In order to win, the adversary needs to invert the one-way function at a (pseudorandom) point: to give a proof for x, the adversary needs to invert $z_{x,1-b_x} = f(\mathsf{F}_{1-b_x}(k_{1-b_x}, x))$ where $b_x = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$. All we need is a way to plant a one-way function challenge instance at each $z_{x,1-b_x}$.

Rerandomizable one-way functions. To complete the proof, we want to argue that any adversary that succeeds at inverting $z_{x,1-b_x}$ for *any* (false) statement $x \in \{0,1\}^n$ translates into one that inverts a one-way function challenge z^*. Moreover, the challenges $z_{x,1-b_x}$ for different x should look indistinguishable from fresh challenges. Phrased differently, we require a way to derive fresh challenges $z_{x,1-b_x}$ from z^* such that a solution y_x where $f(y_x) = z_{x,1-b_x}$ for any x implies a solution y^* where $f(y^*) = z^*$. We refer to one-way functions with this property as *rerandomizable* one-way functions (see Section 3). Suppose we have a rerandomizable one-way function. Then, we define the final hybrid Hyb_2 as follows:[4]

- Hyb_2: In this experiment, the challenger first samples $y^* \xleftarrow{\text{R}} \mathcal{Y}$ and sets the challenge $z^* = f(y^*)$. The challenger also samples a new (puncturable) PRF key $\mathsf{F}_{\mathsf{rerand}}$ that will be used for rerandomizing the challenge z^*. The CRS then consists of obfuscations of the following programs:

```
Prove($x, w$):
  – If $C(x, w) = 1$, compute $b = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and output $\pi = (b, \mathsf{F}_b(k_b, x))$.
  – Otherwise, output $\perp$.
```

```
GenInst($x$):
  – Compute $b = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$.
  – Compute $y_{x,b} = \mathsf{F}_b(k_b, x))$ and $z_{x,b} = f(y_{x,b})$.
  – Compute the rerandomized challenge $z_{x,1-b} = \mathsf{Rerandomize}(z^*; \mathsf{F}_{\mathsf{rerand}}(k_{\mathsf{rerand}}, x))$.
  – Output $(z_{x,0}, z_{x,1})$.
```

  To argue that the distributions of Hyb_1 and Hyb_2 are computationally indistinguishable, we again use a sequence of 2^n different hybrids, one for each value of $x \in \{0,1\}^n$. In the i^{th} hybrid, we change the distribution of $z_{i,1-b}$ from being a freshly-sampled challenge (as in Hyb_1) to being a rerandomized challenge derived from z^* (as in Hyb_2). Each of these intermediate transitions relies on security of $i\mathcal{O}$, security of the underlying puncturable PRFs, and the rerandomizability of the one-way function. Since there are 2^n hybrids, we require sub-exponential hardness of each of the underlying primitives.

  An astute reader might observe that unlike the transition from Hyb_0 to Hyb_1, the transition from Hyb_1 to Hyb_2 *does* rely on security of the one-way function, specifically, the property that a rerandomized instance is indistinguishable from a fresh instance. Thus it might seem like we need to increase the parameters of the one-way function to achieve this. However, this need not be the case. By considering algebraic one-way functions (e.g., based on discrete log or factoring), it is possible to *statistically* rerandomize an instance so that the statistical distance between a fresh instance and a rerandomized instance is at most $2^{-\Omega(n)}$ even though the length of the instances is $\mathrm{poly}(\lambda)$, where λ is the security parameter (independent of the statement length n). Importantly, this hybrid transition

[4] In the technical section, we introduce an intermediate hybrid between Hyb_1 and Hyb_2 to simplify the technical exposition. We elide this intermediate hybrid in this informal overview.

only relies on rerandomization and not security of the one-way function. This is the reason we are able to consider an exponential number of hybrids without affecting the proof length. Thus, once again, we are able to complexity leverage, but not incur *any* overhead in the length of the proof. We provide the full details in Lemma 4.10.

In Hyb_2, a successful prover succeeds only if it inverts the one-way function on the value $z_{x,1-b_x}$ for some $x \in \{0,1\}^n$. But in Hyb_2, the value of $z_{x,1-b_x}$ was derived by rerandomizing the instance z^*. By the rerandomization property of the one-way function, a preimage of $z_{x,1-b}$ can be used to recover a preimage of z^*. In other words, inverting $z_{x,1-b_x}$ for *any* x is sufficient to invert z^*. By security of the one-way function, the advantage of the adversary in Hyb_2 must be negligible, and adaptive soundness holds. This is the *only* step in the security proof where we rely on security of the one-way function. As such, *polynomial* hardness of the one-way function suffices for the analysis. Since the proof is still just a preimage of the one-way function, the size of the proof is $\mathrm{poly}(\lambda)$, independent of the statement length n (or the witness length). The overhead from the complexity leveraging only manifests in the CRS size and *not* the proof size. We provide the formal description and analysis in Section 4.

Constructing rerandomizable one-way functions. The remaining ingredient we need to complete the construction is a rerandomizable one-way function. We describe two constructions here based on the hardness of computing discrete logs and the hardness of factoring (specifically, the hardness of computing modular square roots). Both constructions rely on the random self-reducibility of the underlying assumption. We give the formal constructions and analysis in Section 5.

- **Construction from discrete log:** Let $\mathbb{G}$ be a group of prime order p and generated by g. The discrete log assumption in $\mathbb{G}$ says that given $h \xleftarrow{R} \mathbb{G}$, it is hard to find $x \in \mathbb{Z}_p$ such that $g^x = h$. In our construction, we sample the challenge h as $h \xleftarrow{R} \mathbb{G} \setminus \{g^0\}$.[5] This allows for *perfect* rerandomization: given any challenge $h \in \mathbb{G} \setminus \{g^0\}$, the distribution of h^r where $r \xleftarrow{R} \mathbb{Z}_p^*$ is *exactly* the uniform distribution over the original challenge space $\mathbb{G} \setminus \{g^0\}$. Moreover, given the discrete log s of h^r (i.e., $s \in \mathbb{Z}_p^*$ where $g^s = h^r$), we can recover the discrete log of h by computing $sr^{-1} \bmod p$. This yields a perfectly rerandomizable one-way function. We give the full details in Section 5.1.
- **Construction from factoring:** We obtain a second rerandomizable one-way function based on the hardness of computing square roots modulo $N = pq$, where p, q are distinct primes (i.e., given $x^2 \in \mathbb{Z}_N$ where $x \xleftarrow{R} \mathbb{Z}_N$, find $z \in \mathbb{Z}_N$ such that $z^2 = x^2 \bmod N$). This problem is equivalent to the hardness of factoring N [Rab79]. This problem also has a random self-reduction: namely, given a challenge $y \in \mathbb{Z}_N$, we can construct a new instance by sampling $r \xleftarrow{R} \mathbb{Z}_N$ and outputting $yr^2 \bmod N$. Any solution $s \in \mathbb{Z}_N$ where $s^2 = yr^2$ yields a solution $sr^{-1} \bmod N$ for the original challenge y (provided that r is invertible modulo N). Some extra care is needed to ensure that the statistical distance of the rerandomized distribution and the original challenge distribution is at most $2^{-n} \ll 2^{-\lambda}$. As we show in Section 5.2, this is possible by rejection sampling (since membership in $\mathbb{Z}_N^*$ is efficiently-checkable).

[5]As discussed above, security of our construction requires that the distance between the original distribution and the rerandomized distribution to be small compared to 2^n, where n is the statement size. However, the size of the group should only be a function of the security parameter (to preserve succinctness). In this case, if $p = 2^{O(\lambda)}$, the statistical distance between the uniform distribution on $\mathbb{G}$ and the uniform distribution on $\mathbb{G} \setminus \{g^0\}$ is $1/p = 2^{-O(\lambda)}$, which is *not* small enough compared to 2^n. Thus, the distinction between sampling from $\mathbb{G}$ vs. sampling from $\mathbb{G} \setminus \{g^0\}$ is essential to our construction.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. We write $\mathsf{poly}(\lambda)$ to denote a *fixed* polynomial in the security parameter λ. We say a function $f(\lambda)$ is negligible in λ if $f(\lambda) = o(\lambda^{-c})$ for all $c \in \mathbb{N}$ and denote this by writing $f(\lambda) = \mathsf{negl}(\lambda)$. When $x, y \in \{0,1\}^n$, we will view x and y as both bit-strings of length n as well as the binary representation of an integer between 0 and $2^n - 1$. We write "$x \leq y$" to refer to the comparison of the integer representations of x and y. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input.

Our construction will rely on sub-exponential hardness assumptions, so we will formulate some of our security definitions using (t, ε)-notation. Generally, we say that a primitive is (t, ε)-secure, if for all adversaries $\mathcal{A}$ running in time at most $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, the adversary's advantage is bounded by $\varepsilon(\lambda)$. We say a primitive is polynomially-secure if it is $(1, \mathsf{negl}(\lambda))$-secure for some negligible function $\mathsf{negl}(\cdot)$ and we say that it is sub-exponentially secure if it is $(1, 2^{-\lambda^c})$-secure for some constant $c \in \mathbb{N}$. We now recall the main cryptographic primitives we use in this work.

Definition 2.1 (Indistinguishability Obfuscation [BGI+01])**.** An indistinguishability obfuscator for Boolean circuits is an efficient algorithm $i\mathcal{O}(\cdot,\cdot,\cdot)$ with the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, circuit size parameters $s \in \mathbb{N}$, all Boolean circuits C of size at most s, and all inputs x,

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(1^\lambda, 1^s, C)] = 1.$$

- **Security:** For a bit $b \in \{0,1\}$ and a security parameter λ, we define the program indistinguishability game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input the security parameter 1^λ, the adversary outputs a size parameter 1^s and two Boolean circuits C_0, C_1 of size at most s.
  - If there exists an input x such that $C_0(x) \neq C_1(x)$, then the challenger halts with output $\perp$. Otherwise, the challenger replies with $i\mathcal{O}(1^\lambda, 1^s, C_b)$.
  - The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that $i\mathcal{O}$ is (t, ε)-secure if for all adversaries $\mathcal{A}$ running in time at most $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, we have that

$$\mathrm{iOAdv}_{\mathcal{A}}(\lambda) := |\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \varepsilon(\lambda)$$

  in the program indistinguishability game defined above.

Definition 2.2 (Puncturable PRF [BW13, KPTZ13, BGI14])**.** A puncturable pseudorandom function consists of a tuple of efficient algorithms $\Pi_{\mathsf{PPRF}} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Puncture})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to k$: On input the security parameter λ, an input length ℓ_{in}, and an output length ℓ_{out}, the key-generation algorithm outputs a key k. We assume that the key k contains an implicit description of ℓ_{in} and ℓ_{out}.

- $\mathsf{Puncture}(k, x^*) \to k^{(x^*)}$: On input a key k and a point $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$, the puncture algorithm outputs a punctured key $k^{(x^*)}$. We assume the punctured key also contains an implicit description of ℓ_{in} and ℓ_{out} (same as the key k).

- $\mathsf{Eval}(k, x) \to y$: On input a key k and an input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, the evaluation algorithm outputs a value $y \in \{0,1\}^{\ell_{\mathsf{out}}}$:

In addition, Π_{PPRF} should satisfy the following properties:

- **Functionality-preserving:** For all $\lambda, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, every input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, and every $x \in \{0,1\}^{\ell_{\mathsf{in}}} \setminus \{x^*\}$,

$$\Pr\left[\mathsf{Eval}(k, x) = \mathsf{Eval}(k^{(x^*)}, x) : \begin{array}{r} k \leftarrow \mathsf{KeyGen}(1^\lambda) \\ k^{(x^*)} \leftarrow \mathsf{Puncture}(k, x^*) \end{array}\right] = 1.$$

- **Punctured pseudorandomness:** For a bit $b \in \{0,1\}$ and a security parameter λ, we define the (selective) punctured pseudorandomness game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the input length $1^{\ell_{\mathsf{in}}}$, the output length $1^{\ell_{\mathsf{out}}}$, and commits to a point $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$.
  - The challenger samples $k \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ and gives $k^{(x^*)} \leftarrow \mathsf{Puncture}(k, x^*)$ to $\mathcal{A}$.
  - If $b = 0$, the challenger gives $y^* = \mathsf{Eval}(k, x^*)$ to $\mathcal{A}$. If $b = 1$, then it gives $y^* \xleftarrow{\text{R}} \{0,1\}^{\ell_{\mathsf{out}}}$ to $\mathcal{A}$.
  - At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{PPRF} satisfies (t, ε)-punctured pseudorandomness if for all adversaries $\mathcal{A}$ running in time at most $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, it holds that

$$\mathsf{PPRFAdv}_{\mathcal{A}}(\lambda) := |\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \varepsilon(\lambda)$$

  in the punctured pseudorandomness security game.

Theorem 2.3 (Puncturable PRFs [GGM84, BW13, KPTZ13, BGI14])**.** *Assuming the existence of polynomially-secure (resp., sub-exponentially-secure) one-way functions, then there exists a selective polynomially-secure (resp., sub-exponentially-secure) puncturable PRF.*

Succinct non-interactive arguments. We now recall the definition of a succinct non-interactive argument for the language of Boolean circuit satisfiability. We start by defining the language of Boolean circuit satisfiability:

Definition 2.4 (Boolean Circuit Satisfiability)**.** We define the circuit satisfiability language $\mathcal{L}_{\mathsf{SAT}}$ as

$$\mathcal{L}_{\mathsf{SAT}} = \left\{ (C, x) \;\middle|\; \begin{array}{c} C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}, x \in \{0,1\}^n \\ \exists w \in \{0,1\}^h : C(x, w) = 1 \end{array} \right\}.$$

Definition 2.5 (Succinct Non-Interactive Argument)**.** A succinct non-interactive argument (SNARG) in the preprocessing model for Boolean circuit satisfiability is a tuple $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, C) \to \mathsf{crs}$: On input the security parameter λ and a Boolean circuit C, the setup algorithm outputs a common reference string crs.

- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: On input a common reference string crs, a statement x, and a witness w, the prove algorithm outputs a proof π.

- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: On input a common reference string crs, a statement x and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

Moreover, Π_{SNARG} should satisfy the following properties:

- **Completeness:** For all security parameters $\lambda \in \mathbb{N}$, all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all instances (x, w) where $C(x, w) = 1$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array}\right] = 1.$$

- **Adaptive soundness:** For a security parameter λ, we define the adaptive soundness game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input the security parameter 1^λ, the adversary $\mathcal{A}$ starts by outputting a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
  - The challenger replies with $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C)$.
  - The adversary outputs a statement $x \in \{0,1\}^n$ and a proof π.
  - The output is $b = 1$ if $(C, x) \notin \mathcal{L}_{\mathsf{SAT}}$ and $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$. The output is $b = 0$ otherwise.

  We say that Π_{SNARG} is adaptively sound if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the adaptive soundness game. When $b = 1$, we say that "$\mathcal{A}$ wins the adaptive soundness game."

- **Succinctness:** There exist a polynomial p such that for all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and all crs in the support of $\mathsf{Setup}(1^\lambda, C)$, all statements $x \in \{0,1\}^n$, and all witnesses $w \in \{0,1\}^h$, the size of the proof π output by $\mathsf{Prove}(\mathsf{crs}, x, w)$ satisfies $|\pi| \le p(\lambda + \log|C|)$.

Definition 2.6 (Perfect Zero-Knowledge)**.** A preprocessing SNARG $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for Boolean circuit satisfiability satisfies perfect zero-knowledge if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ such that for all adversaries $\mathcal{A}$, all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and all $(x, w) \in \{0,1\}^n \times \{0,1\}^h$ where $C(x, w) = 1$, we have that

$$\left\{(\mathsf{crs}, x, \pi) : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array}\right\} \equiv \left\{(\mathsf{crs}, x, \pi) : \begin{array}{c} (\mathsf{crs}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_0(1^\lambda, C) \\ \pi \leftarrow \mathcal{S}_1(\mathsf{st}_{\mathcal{S}}, x) \end{array}\right\}.$$

Remark 2.7 (Fast Verification)**.** In a preprocessing SNARG, the length of the common reference string crs can depend polynomially on the size of C (i.e., $|\mathsf{crs}| = \mathsf{poly}(\lambda + |C|)$). Correspondingly, this means the running time of $\mathsf{Verify}(\mathsf{crs}, \cdot, \cdot)$ can be as large as $\mathsf{poly}(\lambda + |C|)$. We can compose the SNARG with a RAM delegation scheme (i.e., a SNARG for P) [CJJ21b, KVZ21, KLVW23] to obtain a SNARG for NP where the verification time is $\mathsf{poly}(\lambda + |x| + \log|C|)$. Instead of computing Verify itself, the verifier delegates the computation of $\mathsf{Verify}(\mathsf{crs}, x, \pi)$ to the prover and verifies the proof that $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$. We sketch the full construction below:

- Let M be an arbitrary RAM machine that takes two inputs x and y and outputs a bit $b \in \{0,1\}$.[6] A RAM delegation scheme allows a prover to convince the verifier that $M(x, y) = \beta$ with a proof π of

[6]The approach in [CJJ21b] considers a RAM machine with a single input x (i.e., the initial state of the RAM program) and the verification algorithm needs to read a digest of x. The same approach extends to the case where the RAM machine takes two inputs x and y, and the verification algorithm is provided a digest for x and y individually.

size $|\pi| = \text{poly}(\lambda + \log|M| + \log|x| + \log|y| + \log T)$, where T is the running time of M. The verification algorithm $\text{Verify}_{\text{RAM}}$ takes as input a common reference string crs_{RAM} for the delegation scheme, a hash digest dig of (M, x), the value y, the proof π, and the claimed value β and either accepts (with output 1) or rejects (with output 0). The length of the verification key and the length of the proof satisfies $|\text{vk}|, |\pi| = \text{poly}(\lambda + \log|M| + \log|x| + \log|y| + \log T)$. The soundness requirement says that if dig is an honestly-generated digest of (M, x), then an efficient prover cannot produce (y, π, b) such that $\text{Verify}_{\text{RAM}}(\text{crs}_{\text{RAM}}, \text{dig}, y, \pi, b) = 1$ and $M(x, y) \neq b$, except with negligible probability.

- To support fast verification for the SNARG, we define the new common reference string to be $\text{crs} = (\text{crs}_{\text{SNARG}}, \text{crs}_{\text{RAM}}, \text{dig})$, where $\text{crs}_{\text{SNARG}}$ is a CRS for the underlying SNARG for NP, crs_{RAM} is the CRS for the RAM delegation scheme, and dig is a digest for $(M, \text{crs}_{\text{SNARG}})$, where $M(\text{crs}_{\text{SNARG}}, (x, \pi))$ is the RAM machine that computes the verification algorithm $\text{Verify}_{\text{SNARG}}(\text{crs}_{\text{SNARG}}, x, \pi)$ for the underlying SNARG.

- A proof for a statement x consists of a SNARG proof π_{SNARG} together with a RAM delegation proof π_{RAM} that $M(\text{crs}_{\text{SNARG}}, (x, \pi_{\text{SNARG}})) := \text{Verify}_{\text{SNARG}}(\text{crs}_{\text{SNARG}}, x, \pi) = 1$. The verification algorithm simply runs $\text{Verify}_{\text{RAM}}(\text{crs}_{\text{RAM}}, \text{dig}, (x, \pi_{\text{SNARG}}), \pi_{\text{RAM}}, 1)$.

Adaptive computational soundness follows from the fact that if $\text{Verify}_{\text{RAM}}(\text{crs}_{\text{RAM}}, \text{dig}, (x, \pi_{\text{SNARG}}), \pi_{\text{RAM}}, 1)$, then with all but negligible probability, $\text{Verify}_{\text{SNARG}}(\text{crs}_{\text{SNARG}}, x, \pi_{\text{SNARG}}) = 1$, and soundness reduces to that of the underlying SNARG. Moreover, the size of π_{RAM} is $\text{poly}(\lambda + \log|C|)$, so the composed scheme remains succinct. In the composed scheme, the verification algorithm only needs crs_{RAM} and dig (but *not* $\text{crs}_{\text{SNARG}}$). Thus, we can define a separate verification key for the composed scheme $\text{vk} = (\text{crs}_{\text{RAM}}, \text{dig})$, which has size $\text{poly}(\lambda + \log|C|)$. The running time of the composed verification algorithm is then $\text{poly}(\lambda + |x| + \log|C|)$.

3 Rerandomizable One-Way Functions

In this section, we introduce the notion of a rerandomizable one-way function, which is one of the main building blocks we use in our construction. Then, in Section 5, we show that rerandomizable one-way functions can be based on standard number-theoretic assumptions.

Definition 3.1 (Rerandomizable One-Way Functions)**.** A rerandomizable one-way function is a tuple of efficient algorithms $\Pi_{\text{ROWF}} = (\text{Setup}, \text{GenInstance}, \text{Rerandomize}, \text{Verify}, \text{RecoverSolution})$ with the following syntax:

- $\text{Setup}(1^\lambda, 1^m) \to \text{crs}$: On input a security parameter λ and a rerandomization parameter m, the setup algorithm outputs a common reference string crs.

- $\text{GenInstance}(\text{crs}) \to (y, z)$: On input the common reference string crs, the instance-generator algorithm outputs an instance y together with a solution z.

- $\text{Rerandomize}(\text{crs}, y) \to (y', \text{st})$: On input the common reference string crs, the rerandomize algorithm outputs a new instance y' and a randomization state st.

- $\text{Verify}(\text{crs}, y, z) \to b$: On input the common reference string crs, an instance y, and a solution z, the verification algorithm outputs a bit $b \in \{0, 1\}$.

- $\text{RecoverSolution}(\text{crs}, z', \text{st}) \to z$: On input the common reference string crs, a solution z' and a randomization state st, the solution-recovery algorithm outputs a solution z.

We require that Π_{ROWF} satisfy the following properties:

- **Correctness:** For all $\lambda, m \in \mathbb{N}$, it holds that
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, y, z) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m) \\ (y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs}) \end{array}\right] = 1.$$

- **Rerandomization correctness:** For all $\lambda, m \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, all (y, z) in the support of $\mathsf{GenInstance}(\mathsf{crs})$, all (y', st) in the support of $\mathsf{Rerandomize}(\mathsf{crs}, y)$, and for all z' where $\mathsf{Verify}(\mathsf{crs}, y', z') = 1$, it holds that
$$\Pr[\mathsf{Verify}(\mathsf{crs}, y, z) = 1 : z \leftarrow \mathsf{RecoverSolution}(\mathsf{crs}, z', \mathsf{st})] = 1.$$

- **One-wayness:** For an adversary $\mathcal{A}$, a security parameter λ, and a rerandomization parameter m, we define the one-wayness security game as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the rerandomization parameter 1^m.
  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y^*, z^*) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. It gives (crs, y^*) to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a solution z. The challenger outputs $b = \mathsf{Verify}(\mathsf{crs}, y^*, z)$.

  We say that the rerandomizable one-way function is one-way if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\mathsf{OWFAdv}_{\mathcal{A}}(\lambda) := \Pr[b = 1] \leq \varepsilon(\lambda)$$
  in the one-wayness game.

- **Rerandomization security:** For an adversary $\mathcal{A}$, a bit $b \in \{0, 1\}$, a security parameter λ, and a rerandomization parameter m, we define the rerandomization security game as follows:
  - The challenger starts by sampling $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$.
  - If $b = 0$, the challenger samples $(y^*, z^*) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. If $b = 1$, the challenger samples $(y^*, \mathsf{st}) \leftarrow \mathsf{Rerandomize}(\mathsf{crs}, y_{\mathsf{base}})$. The challenger gives $(1^\lambda, 1^{m(\lambda)}, \mathsf{crs}, y_{\mathsf{base}}, y^*)$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  We say that the rerandomizable one-way function satisfies (t, ε)-rerandomizable security if for all polynomials $m = m(\lambda)$, all adversaries $\mathcal{A}$ running in time $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists $\lambda_{\mathcal{A},m} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A},m}$,
$$\mathsf{RerandAdv}_{\mathcal{A},m}(\lambda) := |\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \varepsilon(m(\lambda))$$
  in the rerandomization security game. In particular, the distinguishing advantage ε is a function of the *rerandomization parameter m*. We say that Π_{ROWF} satisfies ε-statistical rerandomizable security if for all polynomials $m = m(\lambda)$, all (possibly unbounded) adversaries $\mathcal{A}$, and all $\lambda \in \mathbb{N}$,
$$\mathsf{RerandAdv}_{\mathcal{A},m}(\lambda) \leq \varepsilon(m(\lambda)).$$
  We say Π_{ROWF} satisfies perfect rerandomizable security if it satisfies ε-statistical rerandomizable security for $\varepsilon = 0$.

- **Succinctness:** There exists a polynomial p such that for all $\lambda, m \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, and all (y, z) in the support of $\mathsf{GenInstance}(\mathsf{crs})$, it holds that $|z| \leq p(\lambda + \log m)$.

4 Constructing Adaptively-Sound SNARGs for NP

In this section, we show how to construct an adaptively-sound SNARG from indistinguishability obfuscation together with a rerandomizable one-way function.

Construction 4.1 (Adaptively-Sound SNARGs)**.** Our construction relies on the following primitives:

- Let iO be an indistinguishability obfuscator for Boolean circuits.
- Let $\Pi_{\mathsf{ROWF}} = (\mathsf{R.Setup}, \mathsf{R.GenInstance}, \mathsf{R.Rerandomize}, \mathsf{R.Verify}, \mathsf{R.RecoverSolution})$ be a rerandomizable one-way function.
- Let $\Pi_{\mathsf{PPRF}} = (\mathsf{F.KeyGen}, \mathsf{F.Eval}, \mathsf{F.Puncture})$ be a puncturable PRF. For a key k and an input x, we will write $\mathsf{F}(k, x)$ to denote $\mathsf{F.Eval}(k, x)$.

Our construction will leverage sub-exponential hardness of iO and Π_{PPRF}. In the following, let $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$, $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$, and $m = m(\lambda, n)$ be fixed polynomials in the scheme's security parameter λ and the statement length n. We will describe how to define the polynomials λ_{obf}, λ_{PRF}, and m in the security analysis. We construct a (preprocessing) succinct non-interactive argument $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for Boolean circuit satisfiability as follows:

- $\mathsf{Setup}(1^\lambda, C)$: On input the security parameter λ and a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, the setup algorithm does the following:
  - Let $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^\lambda, 1^m)$.
  - Sample a "selector" PRF key $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$.
  - Let ρ be a bound on the number of bits of randomness the $\mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$ algorithm takes. Sample two additional PRF keys $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$.
  - Define the following programs GenProof and GenInst:

Input: statement x and witness w
Hard-coded: Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and common reference string $\mathsf{crs}_{\mathsf{ROWF}}$ for the rerandomizable one-way function, puncturable PRF keys $k_{\mathsf{sel}}, k_0, k_1$

On input a statement $x \in \{0,1\}^n$ and a witness $w \in \{0,1\}^h$:

- If $C(x, w) = 0$, output $\perp$.
- If $C(x, w) = 1$, then compute $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and $(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$. Output (b, z_b).

Figure 1: The proof-generation program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1]$.

Input: statement x
Hard-coded: Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and common reference string $\mathsf{crs}_{\mathsf{ROWF}}$ for the rerandomizable one-way function, puncturable PRF keys k_0, k_1

On input a statement $x \in \{0,1\}^n$:

- Compute $(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$ for $b \in \{0,1\}$. Output (y_0, y_1).

Figure 2: The instance-generation program $\mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1]$.

Let $s = s(\lambda, n, |C|)$ be the maximum size of the GenProof and GenInst programs as well as those appearing in the proof of Theorem 4.3 (specifically, the programs in Figs. 3 to 5). By construction, we note that $s = \mathrm{poly}(\lambda, |C|)$ is polynomially-bounded.

  – Construct the obfuscated programs $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathrm{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathrm{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathrm{obf}}}, 1^s, \mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1])$. Output the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$.

- $\mathsf{Prove}(\mathsf{crs}, x, w)$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$, the prove algorithm outputs the proof $\pi = (b, z_b) = \mathsf{ObfProve}(x, w)$.

- $\mathsf{Verify}(\mathsf{crs}, x, \pi)$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$, the statement $x \in \{0,1\}^n$, and the proof $\pi = (b, z)$, the verification algorithm runs $(y_0, y_1) = \mathsf{ObfVerify}(x)$. It outputs $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z)$.

Theorem 4.2 (Completeness). *If $i\mathcal{O}$ and Π_{ROWF} are correct, then Construction 4.1 is complete.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, any Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and any instance-witness pair (x, w) where $C(x, w) = 1$. Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify}) \leftarrow \mathsf{Setup}(1^\lambda, C)$ and $\pi = (b, z) \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)$. Consider the output of $\mathsf{Verify}(\mathsf{crs}, x, \pi)$:

- By construction, ObfProve is an obfuscation of the program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathrm{sel}}, k_0, k_1]$, where $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^\lambda, 1^m)$, $k_{\mathrm{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$. In this case (b, z) is obtained by evaluating ObfProve on input (x, w). By correctness of $i\mathcal{O}$ and definition of GenProof, this means that $b = \mathsf{F}(k_{\mathrm{sel}}, x)$ and $(y, z) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$. By correctness of Π_{ROWF}, it follows that $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y, z) = 1$

- By construction ObfVerify is an obfuscation of the program $\mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1]$. The verification algorithm first evaluates GenInst on input x to obtain (y_0', y_1'). By correctness of $i\mathcal{O}$ and definition of GenInst, for $\beta \in \{0,1\}$, we have that $(y_\beta', z_\beta') = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_\beta, x))$.

- The verification algorithm now outputs $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b', z)$. By definition $y_b' = y$ and the verification algorithm outputs 1. □

Theorem 4.3 (Adaptive Soundness). *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathrm{obf}}^{\varepsilon_{\mathrm{obf}}}})$-secure, Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security, and Π_{ROWF} satisfies $(1, 2^{-m^{\varepsilon_{\mathsf{ROWF}}}})$-rerandomization security for constants $\varepsilon_{\mathrm{obf}}, \varepsilon_{\mathsf{PRF}}, \varepsilon_{\mathsf{ROWF}} \in (0,1)$. Let $\lambda_{\mathrm{obf}} = (\lambda + n)^{1/\varepsilon_{\mathrm{obf}}}$, $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$, and $m = (\lambda + n)^{1/\varepsilon_{\mathsf{ROWF}}}$. In addition, suppose $i\mathcal{O}$ is correct, Π_{PPRF} satisfies punctured correctness, and Π_{ROWF} satisfies rerandomization correctness. Then, Construction 4.1 is adaptively sound.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the adaptive soundness game for Construction 4.1 that succeeds with (non-negligible) advantage $\varepsilon = \varepsilon(\lambda)$. We first claim that without loss of generality, we can assume that for every security parameter λ, algorithm $\mathcal{A}$ always outputs a Boolean circuit C with statements of a fixed length $n = n(\lambda)$. To argue this formally, we first use the fact that $\mathcal{A}$ is a polynomial-time algorithm, so on input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs a Boolean circuit of size at most $s_{\max}(\lambda)$, where $s_{\max}(\lambda) = \mathrm{poly}(\lambda)$. This in turn define a maximum statement length $n_{\max}(\lambda) \le s_{\max}(\lambda)$. In an execution of the adaptive soundness game, let E_i be the event that algorithm $\mathcal{A}$ outputs a Boolean circuit C with statements of length i. Then,

$$\Pr[\mathcal{A} \text{ wins the soundness game}] = \sum_{i \in [n_{\max}]} \Pr[\mathcal{A} \text{ wins the soundness game} \wedge \mathsf{E}_i].$$

If $\mathcal{A}$ wins the soundness game with advantage $\varepsilon(\lambda)$, then it must be the case that there exists some index $i^* \in [n_{\max}(\lambda)]$ such that

$$\Pr[\mathcal{A} \text{ wins the soundness game} \wedge \mathsf{E}_{i^*}] \geq \frac{\varepsilon(\lambda)}{n_{\max}(\lambda)}. \tag{4.1}$$

For each security parameter λ, define $n(\lambda) := i^*$ to be the smallest index i^* where Eq. (4.1) holds. We can now construct a new (non-uniform) adversary $\mathcal{A}'$ that functions as a wrapper around $\mathcal{A}$. Namely, algorithm $\mathcal{A}'$ takes as input the security parameter 1^λ and the non-uniform advice $n(\lambda)$. Algorithm $\mathcal{A}'$ runs $\mathcal{A}$ on the same security parameter 1^λ. If $\mathcal{A}$ outputs a Boolean circuit C where the statement length is not $n(\lambda)$, then algorithm $\mathcal{A}'$ aborts. Otherwise, algorithm $\mathcal{A}'$ simply follows the behavior of $\mathcal{A}$ (and outputs whatever $\mathcal{A}$ outputs). By construction,

$$\Pr[\mathcal{A}' \text{ wins the soundness game}] = \Pr[\mathcal{A} \text{ wins the soundness game} \wedge \mathsf{E}_{n(\lambda)}] \geq \frac{\varepsilon(\lambda)}{n_{\max}(\lambda)}.$$

The advantage of $\mathcal{A}'$ is only polynomially-smaller than that of $\mathcal{A}$ and moreover, algorithm $\mathcal{A}'$ always outputs a Boolean circuit with fixed statement $n(\lambda)$ size. Thus, if there exists an adaptive soundness adversary $\mathcal{A}$ that succeeds with non-negligible probability, then we can construct from $\mathcal{A}$ an efficient (non-uniform) adversary $\mathcal{A}'$ that also succeeds with non-negligible probability. For the remainder of this proof, we will thus assume that the adaptive soundness adversary always outputs a circuit C for statements of length exactly $n = n(\lambda)$. We now define a sequence of hybrid experiments:

- Hyb_0: This is the real adaptive soundness experiment. Namely, the adversary starts by outputting a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$. The challenger then constructs the CRS as follows:
  - Sample $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^\lambda, 1^m)$.
  - Sample PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$.
  - The challenger then constructs $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}$ on the programs from Figs. 1 and 2, and s is the same size parameter from Construction 4.1.

  The challenger gives the $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ then outputs a statement x and a proof $\pi = (b, z)$. The challenger then computes $(y_0, y_1) = \mathsf{ObfVerify}(x)$ and the output is 1 if
  $$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1.$$

- Hyb_1: Same as Hyb_0 except the output of the experiment is 1 if the following hold:
  $$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1 \quad \text{and} \quad b \neq \mathsf{F}(k_{\mathsf{sel}}, x).$$

- Hyb_2: Same as Hyb_1 except when computing the output, the challenger no longer checks that $(C, x) \notin \mathcal{L}_{\mathsf{SAT}}$. Namely, the output of the experiment is 1 if
  $$\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1 \quad \text{and} \quad b \neq \mathsf{F}(k_{\mathsf{sel}}, x).$$

- Hyb_3: Same as Hyb_2, except the challenger changes how it constructs $\mathsf{ObfVerify}$. During setup, the challenger now does the following:

- Sample an instance $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$.
- Let κ be the number of bits of randomness the $\mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y)$ takes. Sample a PRF key $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\kappa)$. Define the following program $\mathsf{GenInst}_1$:

Input: statement x
Hard-coded: Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and common reference string $\mathsf{crs}_{\mathsf{ROWF}}$ for the rerandomizable one-way function, puncturable PRF keys $k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}$, and instance y_{base}

On input a statement $x \in \{0,1\}^n$:

* Compute $b = \mathsf{F}(k_{\mathsf{sel}}, x)$.
* Compute $(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$.
* Compute $(y_{1-b}, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, x))$.
* Output (y_0, y_1).

Figure 3: The instance-generation program $\mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}]$.

The challenger sets $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}])$ in crs. The rest of the experiment proceeds exactly as in Hyb_2.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output distribution of an execution of hybrid Hyb_i with the adversary $\mathcal{A}$. We now analyze each adjacent pair of hybrid distributions.

Lemma 4.4. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure and suppose Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security for constants $\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}} \in (0,1)$. In addition, suppose that $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. Finally, suppose Π_{PPRF} satisfies punctured correctness. Then,*

$$\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \geq \frac{1}{2} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - 2^{-\Omega(\lambda)}.$$

Proof. Consider an execution of Hyb_0 or Hyb_1. For an index $i \in \{0,1\}^n$, let E_i be the event that the adversary $\mathcal{A}$ outputs i as its statement in an execution of Hyb_0 or Hyb_1. By definition, we can now write

$$\begin{aligned} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] &= \sum_{i \in \{0,1\}^n} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1 \wedge \mathsf{E}_i] \\ \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] &= \sum_{i \in \{0,1\}^n} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_i]. \end{aligned} \tag{4.2}$$

To show the claim, we show that for all $i \in \{0,1\}^n$,

$$\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_i] \geq \frac{1}{2} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \frac{1}{2^n} \cdot \frac{O(1)}{2^\lambda}. \tag{4.3}$$

To show this, we consider two cases.

Case 1. Suppose $(C, i) \in \mathcal{L}_{\mathsf{SAT}}$. If the adversary outputs i as its statement (i.e., if E_i occurs), then the output in Hyb_0 and Hyb_1 are both 0. Thus,

$$\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1 \wedge \mathsf{E}_i] = 0 = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_i].$$

Correspondingly, Eq. (4.3) holds.

Case 2. Suppose $(C, i) \notin \mathcal{L}_{\mathsf{SAT}}$. In this case, we proceed by defining a sequence of hybrids:

- $\mathsf{Hyb}_{0,i}^{(0)}$: Same as Hyb_0 except the challenger outputs 1 only if
$$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1 \quad \text{and} \quad x = i.$$

- $\mathsf{Hyb}_{0,i}^{(1)}$: Same as $\mathsf{Hyb}_{0,i}^{(0)}$ except when setting up the CRS, the challenger defines the modified solution-generation problem $\mathsf{GenProof}_1$ as follows:

Input: statement x and witness w
Hard-coded: Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and common reference string $\mathsf{crs}_{\mathsf{ROWF}}$ for the rerandomizable one-way function, puncturable PRF keys $k_{\mathsf{sel}}, k_0, k_1$, statement $i \in \{0,1\}^n$

On input a statement $x \in \{0,1\}^n$ and a witness $w \in \{0,1\}^h$:

- If $x = i$, output $\perp$.
- If $C(x, w) = 0$, output $\perp$.
- If $C(x, w) = 1$, then compute $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and $(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$. Output (b, z_b).

Figure 4: The solution-generation program $\mathsf{GenProof}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, i]$.

Next, after sampling $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, the challenger computes $k_{\mathsf{sel}}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{sel}}, i)$. It then constructs the prover program $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i])$. The remainder of the program proceeds as in $\mathsf{Hyb}_{0,i}^{(0)}$.

- $\mathsf{Hyb}_{0,i}^{(2)}$: Same as $\mathsf{Hyb}_{0,i}^{(1)}$, except after the adversary outputs its statement x and the proof $\pi = (b, z)$, the challenger samples a random bit $b' \xleftarrow{\mathrm{R}} \{0,1\}$ and outputs 1 if
$$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1 \quad \text{and} \quad x = i \quad \text{and} \quad b \neq b'.$$

- $\mathsf{Hyb}_{0,i}^{(3)}$: Same as $\mathsf{Hyb}_{0,i}^{(2)}$, except after the adversary outputs its statement x and the proof $\pi = (b, z)$, the challenger outputs 1 if
$$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1 \quad \text{and} \quad x = i \quad \text{and} \quad b \neq \mathsf{F}(k_{\mathsf{sel}}, i).$$

- $\mathsf{Hyb}_{0,i}^{(4)}$: Same as $\mathsf{Hyb}_{0,i}^{(3)}$, except when setting up the CRS, the challenger reverts to computing $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1])$.

By definition,
$$\Pr[\mathsf{Hyb}_{0,i}^{(0)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1 \wedge \mathsf{E}_i] \quad \text{and} \quad \Pr[\mathsf{Hyb}_{0,i}^{(4)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_i].$$

We now consider each pair of adjacent distributions.

Claim 4.5. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$. In addition, suppose $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ and Π_{PPRF} satisfies punctured correctness. Then, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*
$$|\Pr[\mathsf{Hyb}_{0,i}^{(1)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{0,i}^{(0)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda + n}.$$

Proof. We first show that $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1]$ in Hyb_0 and $\mathsf{GenProof}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i]$ in Hyb_1 compute identical functionality. We consider the possibilities. Let (x, w) be an input to the two programs.

- Suppose $x = i$. We are analyzing the case $(C, i) \notin \mathcal{L}_{\mathsf{SAT}}$, so $C(i, w) = 0$. In this case, both programs output $\perp$.
- Suppose $C(x, w) = 0$. Then both programs output $\perp$.
- Suppose $C(x, w) = 1$. In this case $x \neq i$. Since the key $k_{\mathsf{sel}}^{(i)}$ is punctured at input i, it follows that $\mathsf{F}(k_{\mathsf{sel}}, x) = \mathsf{F}(k_{\mathsf{sel}}^{(i)}, x)$. Once again, the behavior of the two programs are identical.

We conclude that the two programs output identical functionality. The claim now follows by $i\mathcal{O}$ security. Formally, suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$,

$$|\Pr[\mathsf{Hyb}_{0,i}^{(1)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{0,i}^{(0)}(\mathcal{A}) = 1]| > 1/2^{\lambda + n(\lambda)}. \tag{4.4}$$

Let $\Lambda_{\mathcal{B}} = \left\{(\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{obf}}} : \lambda \in \Lambda_{\mathcal{A}}\right\}$. Since n is a non-negative function, $\Lambda_{\mathcal{B}}$ is also an infinite set. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ such that for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) > 1/2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$. For each value of $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular λ_{obf}, we pick the largest such λ; note that since $\varepsilon_{\mathsf{obf}} < 1$ and $n(\lambda) > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{obf}}$). Algorithm $\mathcal{B}$ proceeds as follows:

1. On input the security parameter $1^{\lambda_{\mathsf{obf}}}$ (and advice string 1^{λ}), algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on security parameter 1^{λ} to get a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^{\lambda}, 1^m)$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$. It computes $k_{\mathsf{sel}}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{sel}}, i)$.
3. Algorithm $\mathcal{B}$ computes s as in Construction 4.1 and gives 1^s, $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1]$, and $\mathsf{GenProof}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i]$ to the challenger to receive the obfuscated program $\mathsf{ObfProve}$.
4. Algorithm $\mathcal{B}$ computes $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1])$ and gives $\mathcal{A}$ the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$.
5. After algorithm $\mathcal{A}$ outputs a statement x and a proof $\pi = (b, z)$, algorithm $\mathcal{B}$ computes $(y_0, y_1) = \mathsf{ObfVerify}(x)$ and outputs 1 if $x = i$ and $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1$.

If the challenger obfuscates the program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1]$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,i}^{(0)}$. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_{0,i}^{(0)}(\mathcal{A}) = 1]$. Alternatively, if the challenger obfuscates the program $\mathsf{GenProof}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i]$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{0,i}^{(1)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{0,i}^{(1)}(\mathcal{A}) = 1]$. By Eq. (4.4),

$$\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) > 2^{-(\lambda + n(\lambda))} = 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}. \qquad \square$$

Claim 4.6. *It holds that* $\Pr[\mathsf{Hyb}_{0,i}^{(2)}(\mathcal{A}) = 1] = \frac{1}{2} \Pr[\mathsf{Hyb}_{0,i}^{(1)}(\mathcal{A}) = 1]$.

Proof. The only difference between $\mathsf{Hyb}_{0,i}^{(1)}$ and $\mathsf{Hyb}_{0,i}^{(2)}$ is the extra condition $b \neq b'$ in $\mathsf{Hyb}_{0,i}^{(2)}$. Since the challenger samples $b' \xleftarrow{\mathrm{R}} \{0,1\}$ *after* the adversary outputs b, we have that $b' = b$ with probability $1/2$. $\square$

Claim 4.7. *Suppose Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security for some constant $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, it holds that*

$$|\Pr[\mathsf{Hyb}_{0,i}^{(2)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{0,i}^{(3)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

Proof. Suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$,

$$|\Pr[\mathsf{Hyb}_{0,i}^{(2)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{0,i}^{(3)}(\mathcal{A}) = 1]| > 1/2^{\lambda+n(\lambda)}.$$

Let $\Lambda_{\mathcal{B}} = \left\{(\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{PRF}}} : \lambda \in \Lambda_{\mathcal{B}}\right\}$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ such that for all $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, $\mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) > 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}$. For each $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular λ_{PRF}, we pick the largest such λ). Algorithm $\mathcal{B}$ now proceeds as follows:

1. On input the security parameter $1^{\lambda_{\mathsf{PRF}}}$ (and advice 1^{λ}), algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input 1^{λ} to get a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^{\lambda}, 1^m)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$. It gives the input length 1^n, the output length 1^1, and the point $i \in \{0,1\}^n$ to the punctured PRF challenger. The challenger replies with a punctured key $k_{\mathsf{sel}}^{(i)}$ and a challenge bit $b' \in \{0,1\}$.
3. Algorithm $\mathcal{B}$ sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$, and computes

$$\mathsf{ObfProve} \leftarrow i\mathcal{O}\big(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i]\big)$$
$$\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1]).$$

   It gives $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$.
4. After algorithm $\mathcal{A}$ outputs a statement x and a proof $\pi = (b, z)$, algorithm $\mathcal{B}$ computes $(y_0, y_1) = \mathsf{ObfVerify}(x)$ and outputs 1 if $x = i$, $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1$, and $b = b'$.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{0,i}^{(2)}$ and $\mathsf{Hyb}_{0,i}^{(3)}$ for $\mathcal{A}$. If the challenger samples $b' \xleftarrow{\mathrm{R}} \{0,1\}$, then algorithm $\mathcal{B}$ computes its output according to the specification of $\mathsf{Hyb}_{0,i}^{(2)}$. If the challenger computes $b' = \mathsf{F}(k_{\mathsf{sel}}, i)$, then algorithm $\mathcal{B}$ computes its output according to the specification of $\mathsf{Hyb}_{0,i}^{(3)}$. Correspondingly, $\mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) > 2^{-(\lambda+n(\lambda))} = 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}$. □

Claim 4.8. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$. In addition, suppose $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ and Π_{PPRF} satisfies punctured correctness. Then, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

$$|\Pr[\mathsf{Hyb}_{0,i}^{(3)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{0,i}^{(4)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

Proof. This follows by an analogous argument as the proof of Claim 4.5. □

Combining Claims 4.5 to 4.8, we have demonstrated that for all $i \in \{0,1\}^n$ where $(C, i) \notin \mathcal{L}_{\mathsf{SAT}}$, Eq. (4.3) holds. Combined with Eq. (4.2), we can now write

$$\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \sum_{i \in \{0,1\}^n} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_i] \geq \frac{1}{2} \sum_{i \in \{0,1\}^n} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \frac{2^n}{2^n} \cdot \frac{O(1)}{2^{\lambda}}$$
$$= \frac{1}{2} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - 2^{-\Omega(\lambda)}.$$
□

Lemma 4.9. *It holds that* $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

Proof. This follows by construction since the conditions for outputting 1 in Hyb_2 is a *strict* subset of those in Hyb_1. Thus, whenever the challenger outputs 1 in Hyb_1, it would also do so in Hyb_2, and the lemma follows. □

Lemma 4.10. *Suppose* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathrm{obf}}^{\varepsilon_{\mathrm{obf}}}})$*-secure,* Π_{PPRF} *satisfies selective* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-punctured security, and* Π_{ROWF} *satisfies* $(1, 2^{-m^{\varepsilon_{\mathsf{ROWF}}}})$*-rerandomization security for constants* $\varepsilon_{\mathrm{obf}}, \varepsilon_{\mathsf{PRF}}, \varepsilon_{\mathsf{ROWF}} \in (0,1)$*. Let* $\lambda_{\mathrm{obf}} = (\lambda+n)^{1/\varepsilon_{\mathrm{obf}}}$, $\lambda_{\mathsf{PRF}} = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}$*, and* $m = (\lambda+n)^{1/\varepsilon_{\mathsf{ROWF}}}$*. Finally, suppose* Π_{PPRF} *satisfies punctured correctness. Then,*

$$|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq 2^{-\Omega(\lambda)}.$$

Proof. We define a sequence of intermediate hybrids indexed by $i \in \{0, \ldots, 2^n\}$:

- $\mathsf{Hyb}_{2,i}^{(0)}$: Same as Hyb_2, except the challenger defines the following program $\mathsf{GenInst}_2$:

Input: statement x
Hard-coded: Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and common reference string $\mathsf{crs}_{\mathsf{ROWF}}$ for the rerandomizable one-way function, puncturable PRF keys $k_{\mathrm{sel}}, k_0, k_1, k_{\mathrm{rerand}}$, instances y_{base}, y^*, index $i \in \{0,1\}^n$

On input a statement $x \in \{0,1\}^n$:

```
- Compute $b = \mathsf{F}(k_{\mathrm{sel}}, x)$.
- Compute $(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$.
- Compute $y_{1-b}$ as follows:
    * If $x < i$, let $(y_{1-b}, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathrm{base}}; \mathsf{F}(k_{\mathrm{rerand}}, x))$.
    * If $x = i$, let $y_{1-b} = y^*$.
    * If $x > i$, let $(y_{1-b}, z_{1-b}) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_{1-b}, x))$.
- Output $(y_0, y_1)$.
```

Figure 5: The instance-generation program $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathrm{sel}}, k_0, k_1, k_{\mathrm{rerand}}, y_{\mathrm{base}}, y^*, i]$.

Then, the challenger samples $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^\lambda, 1^m)$ and PRF keys $k_{\mathrm{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$, $k_{\mathrm{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\kappa)$. The challenger also samples the following additional components:

- Sample $(y_{\mathrm{base}}, z_{\mathrm{base}}) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$.
- Let $b^* = 1 - \mathsf{F}(k_{\mathrm{sel}}, i)$. Compute $r^* = \mathsf{F}(k_{b^*}, i)$ and $(y^*, z^*) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; r^*)$.

The challenger computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathrm{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathrm{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathrm{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathrm{sel}}, k_0, k_1, k_{\mathrm{rerand}}, y_{\mathrm{base}}, y^*, i])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}_2$ are the programs from Figs. 1 and 5 and s is the bound on the program size from Construction 4.1. Algorithm $\mathcal{B}$ gives $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$. After $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$, the challenger computes $(y_0, y_1) = \mathsf{ObfVerify}(x)$ and outputs 1 if

$$\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1 \quad \text{and} \quad b \neq \mathsf{F}(k_{\mathrm{sel}}, x).$$

- $\mathsf{Hyb}_{2,i}^{(1)}$: Same as $\mathsf{Hyb}_{2,i}^{(0)}$, except after computing $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$, the challenger punctures k_{b^*} and k_{rerand} at index i. Namely, it computes $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$ and $k_{\mathsf{rerand}}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{rerand}}, i)$ It still sets $r^* = \mathsf{F}(k_{b^*}, i)$ and $(y^*, z^*) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; r^*)$. Then, it uses the punctured keys $k_{b^*}^{(i)}$ and $k_{\mathsf{rerand}}^{(i)}$ in place of k_{b^*} and k_{rerand} in ObfProve and ObfVerify. Specifically, ObfProve and ObfVerify are now defined as follows:
  - If $b^* = 0$, then the challenger sets $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1, k_{\mathsf{rerand}}^{(i)}, y_{\mathsf{base}}, y^*, i])$.
  - If $b^* = 1$, then the challenger sets $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_{b^*}^{(i)}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_{b^*}^{(i)}, k_{\mathsf{rerand}}^{(i)}, y_{\mathsf{base}}, y^*, i])$.
- $\mathsf{Hyb}_{2,i}^{(2)}$: Same as $\mathsf{Hyb}_{2,i}^{(1)}$, except the challenger samples $r^* \xleftarrow{\mathrm{R}} \{0,1\}^\rho$.
- $\mathsf{Hyb}_{2,i}^{(3)}$: Same as $\mathsf{Hyb}_{2,i}^{(2)}$, except the challenger first samples $r^* \xleftarrow{\mathrm{R}} \{0,1\}^\kappa$ and rerandomizes y_{base} to obtain y^*: $(y^*, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}}; r^*)$.
- $\mathsf{Hyb}_{2,i}^{(4)}$: Same as $\mathsf{Hyb}_{2,i}^{(3)}$, except the challenger sets $r^* = \mathsf{F}(k_{\mathsf{rerand}}, i)$.

We now show that each pair of adjacent experiments are indistinguishable.

Claim 4.11. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda^{\varepsilon_{\mathsf{obf}}}})$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and suppose $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$. Suppose Π_{PPRF} satisfies punctured correctness. Then, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

$$|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,0}^{(0)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

Proof. We start by showing that the program $\mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1]$ in Hyb_2 and the corresponding program $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, 0]$ in $\mathsf{Hyb}_{2,0}^{(0)}$ compute identical functionalities. Take any input $x \in \{0,1\}^n$, and consider the program $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, 0]$ in $\mathsf{Hyb}_{2,0}^{(0)}$:

- Let $b = \mathsf{F}(k_{\mathsf{sel}}, x)$. Then $\mathsf{GenInst}_2$ computes $(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$, which is exactly how the program GenInst computes (y_b, z_b).
- Consider the distribution of y_{1-b}. In $\mathsf{Hyb}_{2,0}^{(0)}$, when x satisfies $x > 0$, the program $\mathsf{GenInst}_2$ computes $(y_{1-b}, z_{1-b}) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_{1-b}, x)))$, which matches the behavior of GenInst. When $x = 0$, $\mathsf{GenInst}_2$ sets $y_{1-b} = y^*$, where $(y^*, z^*) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; r^*)$ and $r^* = \mathsf{F}(k_{1-b}, x)$. Once again, this is the behavior of GenInst.

We conclude that on all inputs x, the verification programs GenInst and $\mathsf{GenInst}_2$ in Hyb_2 and $\mathsf{Hyb}_{2,0}^{(0)}$ have identical input/output behavior. The claim now holds by security of $i\mathcal{O}$. Formally, suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$,

$$|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,0}^{(0)}(\mathcal{A}) = 1]| > 1/2^{\lambda+n(\lambda)}.$$

Let $\Lambda_{\mathcal{B}} = \left\{(\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{obf}}} : \lambda \in \Lambda_{\mathcal{A}}\right\}$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ such that for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) > 1/2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$. For each value of $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular λ_{obf}, we pick the largest such λ). Algorithm $\mathcal{B}$ works as follows:

1. On input the security parameter $1^{\lambda_{\mathrm{obf}}}$ (and advice string 1^{λ}), algorithm $\mathcal{B}$ runs $\mathcal{A}$ on security parameter λ to get a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^{\lambda}, 1^m)$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and then samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$, $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\kappa})$.

3. Algorithm $\mathcal{B}$ then computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, 0)$, $r^* = \mathsf{F}(k_{b^*}, 0)$ and $(y^*, z^*) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; r^*)$. It also samples $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$.

4. Algorithm $\mathcal{B}$ computes the parameter s as in Construction 4.1 and gives 1^s, $\mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1]$, and $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, 0]$ to the challenger. The challenger replies with an obfuscated program $\mathsf{ObfVerify}$.

5. Algorithm $\mathcal{B}$ computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathrm{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and gives the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$.

6. After $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$, the challenger computes $(y_0, y_1) = \mathsf{ObfVerify}(x)$ and outputs 1 if $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1$ and $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$.

If the challenger obfuscates the program $\mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_0, k_1]$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_2. If the challenger obfuscates the program $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, 0]$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,0}^{(0)}$. Correspondingly, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathrm{obf}}) > 2^{-(\lambda + n(\lambda))} = 2^{-\lambda_{\mathrm{obf}}^{\varepsilon_{\mathrm{obf}}}}$. □

Claim 4.12. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda^{\varepsilon_{\mathrm{obf}}}})$-secure for some constant $\varepsilon_{\mathrm{obf}} \in (0,1)$ and suppose $\lambda_{\mathrm{obf}} = (\lambda + n)^{1/\varepsilon_{\mathrm{obf}}}$. Suppose Π_{PPRF} satisfies punctured correctness. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

$$\left|\Pr[\mathsf{Hyb}_{2,i}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(1)}(\mathcal{A}) = 1]\right| \leq 2/2^{\lambda+n}.$$

Proof. Take any $i \in \{0, \ldots, 2^n - 1\}$. Consider an execution of $\mathsf{Hyb}_{2,i}^{(0)}$ and $\mathsf{Hyb}_{2,i}^{(1)}$. Let $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$. We first show that if $b^* = 0$, then the program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}, k_1]$ in $\mathsf{Hyb}_{2,i}^{(0)}$ has the same functionality as the program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1]$ in $\mathsf{Hyb}_{2,i}^{(0)}$:

- First, the key $k_{b^*}^{(i)}$ is punctured on input i, so it follows that $\mathsf{F}(k_{b^*}^{(i)}, x) = \mathsf{F}(k_{b^*}, x)$ for all $x \neq i$. Thus, on all inputs (x, w) where $x \neq i$, the two programs behave identically.
- Consider an input (x, w) where $x = i$. In this case, both programs first computes $b = \mathsf{F}(k_{\mathsf{sel}}, i)$ and then evaluate $\mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, i))$. However, by definition, $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i) = 1 - b \neq b$. In this case, both programs derive the randomness using $\mathsf{F}(k_{1-b^*}, x) = \mathsf{F}(k_1, x)$. Once again, the two programs have identical functionality.

Next, we show that the program $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, i]$ in $\mathsf{Hyb}_{2,i}^{(0)}$ has the same functionality as the program $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1, k_{\mathsf{rerand}}^{(i)}, y_{\mathsf{base}}, y^*, i]$ in $\mathsf{Hyb}_{2,i}^{(1)}$:

- By punctured correctness, for all $x \neq i$, it follows that

$$\mathsf{F}\big(k_{b^*}^{(i)}, x\big) = \mathsf{F}(k_{b^*}, x) \quad \text{and} \quad \mathsf{F}\big(k_{\mathsf{rerand}}^{(i)}, x\big) = \mathsf{F}(k_{\mathsf{rerand}}, x).$$

  Thus, for all inputs $x \neq i$, the two programs have identical behavior.

- Suppose $x = i$. Then, both programs compute $b = \mathsf{F}(k_{\mathsf{sel}}, i)$ and $\mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, i))$. By definition, $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i) = 1 - b \neq b$. In this case, both programs derive the randomness using $\mathsf{F}(k_{1-b^*}, x) = \mathsf{F}(k_1, x)$. Once again, the two programs have identical functionality.

An analogous argument shows that the GenProof and GenInst programs in $\mathsf{Hyb}_{2,i}^{(0)}$ and $\mathsf{Hyb}_{2,i}^{(1)}$ have identical behavior when $b^* = 1$. To complete the proof, we first introduce an intermediate hybrid:

- iHyb_i: Same as $\mathsf{Hyb}_{2,i}^{(1)}$ except the challenger computes the ObfVerify as in $\mathsf{Hyb}_{2,i}^{(0)}$. Namely, it computes $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, i])$.

Suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$,

$$|\Pr[\mathsf{Hyb}_{2,i}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_i(\mathcal{A}) = 1]| > 1/2^{\lambda + n(\lambda)}.$$

Let $\Lambda_{\mathcal{B}} = \left\{(\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{obf}}} : \lambda \in \Lambda_{\mathcal{A}}\right\}$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ such that for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) > 1/2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$. For each value of $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular λ_{obf}, we pick the largest such λ). Algorithm $\mathcal{B}$ works as follows:

1. On input the security parameter $1^{\lambda_{\mathsf{obf}}}$ (and advice 1^λ), algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input 1^λ to get a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^\lambda, 1^m)$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$, and $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\kappa)$.

3. Algorithm $\mathcal{B}$ then computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$, $r^* = \mathsf{F}(k_{b^*}, i)$ and $(y^*, z^*) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; r^*)$. It also samples $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$ and $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$.

4. Algorithm $\mathcal{B}$ computes the parameter s as in Construction 4.1. It then constructs its challenge as follows:

   - If $b^* = 0$, it gives 1^s, $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1]$, and $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1]$ to the challenger.
   - If $b^* = 1$, it gives 1^s, $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1]$, and $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_{b^*}^{(i)}]$ to the challenger.

   The challenger replies with an obfuscated program ObfProve.

5. Algorithm $\mathcal{B}$ computes $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, i])$ and gives the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$.

6. After $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$, the challenger computes $(y_0, y_1) = \mathsf{ObfVerify}(x)$ and outputs 1 if $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1$ and $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$.

If the challenger obfuscates the program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1]$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,i}^{(0)}$. If the challenger obfuscates the program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1]$ (in the case where $b^* = 0$) or $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_{b^*}^{(i)}]$ (in the case where $b^* = 1$), algorithm $\mathcal{B}$ perfectly simulates iHyb_i. Correspondingly, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) > 2^{-(\lambda+n(\lambda))} = 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$. As such, algorithm $\mathcal{B}$ breaks $(1, 2^{-\lambda^{\varepsilon_{\mathsf{obf}}}})$-security of $i\mathcal{O}$. Thus, for all sufficiently-large $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{2,i}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_i(\mathcal{A}) = 1]| \le 1/2^{\lambda+n(\lambda)}. \tag{4.5}$$

By an analogous argument (where the reduction algorithm obtains ObfVerify from the challenger), we can show that for all sufficiently-large $\lambda \in \mathbb{N}$, it holds that

$$|\Pr[\mathsf{Hyb}_{2,i}^{(1)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_i(\mathcal{A}) = 1]| \le 1/2^{\lambda+n(\lambda)}. \tag{4.6}$$

Combining Eqs. (4.5) and (4.6), we conclude that for all sufficiently-large $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{2,i}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(1)}(\mathcal{A}) = 1]| \le 2/2^{\lambda+n(\lambda)}. \qquad \square$$

Claim 4.13. *Suppose Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security for some constant $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and $\lambda_{\mathsf{PRF}} = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \ge \lambda_{\mathcal{A}}$, it holds that*

$$|\Pr[\mathsf{Hyb}_{2,i}^{(1)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(2)}(\mathcal{A}) = 1] \le 1/2^{\lambda+n}$$

Proof. Take any $i \in \{0, \ldots, 2^n - 1\}$ and suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$,

$$|\Pr[\mathsf{Hyb}_{2,i}^{(1)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(2)}(\mathcal{A}) = 1]| > 1/2^{\lambda+n(\lambda)}.$$

Let $\Lambda_{\mathcal{B}} = \left\{(\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{PRF}}} : \lambda \in \Lambda_{\mathcal{A}}\right\}$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ such that for all $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, $\mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) > 1/2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}$. For each value of $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular λ_{PRF}, we pick the largest such λ). Algorithm $\mathcal{B}$ works as follows:

1. On input the security parameter $1^{\lambda_{\mathsf{PRF}}}$ (and advice string 1^λ), algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input 1^λ to obtain a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^\lambda, 1^m)$, $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$, and $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$. It computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$. It samples $k_{1-b^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$ and $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\kappa)$. Algorithm $\mathcal{B}$ also computes $k_{\mathsf{rerand}}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{rerand}}, i)$.

3. Algorithm $\mathcal{B}$ submits the input length 1^n, the output length 1^ρ, and a point $i \in \{0,1\}^n$ to the punctured PRF challenger. It receives the punctured key $k_{b^*}^{(i)}$ as well as the challenge value $r^* \in \{0,1\}^\rho$.

4. Algorithm $\mathcal{B}$ now samples $(y^*, z^*) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; r^*)$. Then, algorithm $\mathcal{B}$ sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$ and constructs the programs ObfProve and ObfVerify as follows:

   - If $b^* = 0$, then it computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_{1-b^*}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_{1-b^*}, k_{\mathsf{rerand}}^{(i)}, y_{\mathsf{base}}, y^*, i])$.
   - If $b^* = 1$, then it computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{1-b^*}, k_{b^*}^{(i)}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{1-b^*}, k_{b^*}^{(i)}, k_{\mathsf{rerand}}^{(i)}, y_{\mathsf{base}}, y^*, i])$.

   Algorithm $\mathcal{B}$ gives the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$.

5. After algorithm $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$, algorithm $\mathcal{B}$ computes $(y_0, y_1) \leftarrow \mathsf{ObfVerify}(x)$ and outputs 1 if $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1$ and $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$.

By definition, the punctured PRF challenger constructs key $k_{b^*}^{(i)}$ by first sampling $k_{b^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$ and setting $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$. This matches the specification in $\mathsf{Hyb}_{2,i}^{(1)}$ to $\mathsf{Hyb}_{2,i}^{(2)}$. Consider now the distribution of the challenge value r^*:

- Suppose $r^* = \mathsf{F}(k_{b^*}, i)$. Then, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{2,i}^{(1)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{2,i}^{(1)}(\mathcal{A}) = 1]$.
- Suppose $r^* \xleftarrow{\mathrm{R}} \{0,1\}^\rho$. Then, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{2,i}^{(2)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{2,i}^{(2)}(\mathcal{A}) = 1]$.

Then $\mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) > 2^{-(\lambda + n(\lambda))} = 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}$, and the claim follows. □

Claim 4.14. *Suppose* Π_{ROWF} *satisfies* $(1, 2^{-m^{\varepsilon_{\mathsf{ROWF}}}})$*-rerandomization security for some constant* $\varepsilon_{\mathsf{ROWF}} \in (0, 1]$*, and suppose* $m = (\lambda + n)^{1/\varepsilon_{\mathsf{ROWF}}}$*. Then, for all* $i \in \{0, \ldots, 2^n - 1\}$*, there exists* $\lambda_{\mathcal{A}} \in \mathbb{N}$ *such that for all* $\lambda \geq \lambda_{\mathcal{A}}$*, it holds that*

$$|\Pr[\mathsf{Hyb}_{2,i}^{(2)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(3)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

Proof. Take any $i \in \{0, \ldots, 2^n - 1\}$ and suppose there exists an infinite set $\Lambda \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$,

$$|\Pr[\mathsf{Hyb}_{2,i}^{(2)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(3)}(\mathcal{A}) = 1]| > 1/2^{\lambda+n(\lambda)}.$$

Let $m(\lambda) = (\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{ROWF}}}$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ such that for all $\lambda \in \Lambda$, $\mathsf{RerandAdv}_{\mathcal{B},m}(\lambda) > 1/2^{-m(\lambda)^{\varepsilon_{\mathsf{ROWF}}}}$. Algorithm $\mathcal{B}$ works as follows:

1. On input the challenge $(1^\lambda, 1^{m(\lambda)}, \mathsf{crs}, y_{\mathsf{base}}, y^*)$, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input 1^λ to obtain the circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
2. Algorithm $\mathcal{B}$ computes $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^\lambda, 1^n, 1^\rho)$, and $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\kappa)$.
3. Algorithm $\mathcal{B}$ computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$. It then computes the punctured keys $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$ and $k_{\mathsf{rerand}}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{rerand}}, i)$. Finally, it sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$ and constructs $\mathsf{ObfProve}$ and $\mathsf{ObfVerify}$ as follows:
   - If $b^* = 0$, then it computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1, k_{\mathsf{rerand}}^{(i)}, y_{\mathsf{base}}, y^*, i])$.
   - If $b^* = 1$, then it computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_{b^*}^{(i)}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_{b^*}^{(i)}, k_{\mathsf{rerand}}^{(i)}, y_{\mathsf{base}}, y^*, i])$.
4. After algorithm $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$, algorithm $\mathcal{B}$ computes $(y_0, y_1) \leftarrow \mathsf{ObfVerify}(x)$ and outputs 1 if $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1$ and $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$.

The challenger samples $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^\lambda, 1^m)$ and $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$. This matches the distribution of $\mathsf{crs}_{\mathsf{ROWF}}$ and y_{base} in $\mathsf{Hyb}_{2,i}^{(2)}$ and $\mathsf{Hyb}_{2,i}^{(3)}$. Consider the distribution of y^*:

- Suppose the challenger samples $r^* \xleftarrow{R} \{0,1\}^\rho$ and computes $(y^*, z^*) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; r^*)$. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,i}^{(2)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{2,i}^{(2)}(\mathcal{A}) = 1]$.
- Suppose the challenger samples $r^* \xleftarrow{R} \{0,1\}^\kappa$ and sets $(y^*, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}}; r^*)$. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,i}^{(3)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{2,i}^{(3)}(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ succeeds with advantage $\mathsf{RerandAdv}_{\mathcal{B},m}(\lambda) > 2^{-(\lambda+n(\lambda))} = 2^{-m^{\varepsilon_{\mathsf{ROWF}}}}$. ☐

Claim 4.15. *Suppose Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security for some constant $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and $\lambda_{\mathsf{PRF}} = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, it holds that*

$$|\Pr[\mathsf{Hyb}_{2,i}^{(3)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(4)}(\mathcal{A}) = 1] \leq 1/2^{\lambda+n}$$

Proof. This follows by a similar argument as in the proof of Claim 4.13, except the reduction algorithm outputs 1^κ as the output length and programs $k_{\mathsf{rerand}}^{(i)}$ to be the punctured key (and samples k_0, k_1 itself). The rest of the argument proceeds analogously. ☐

Claim 4.16. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda^{\varepsilon_{\mathsf{obf}}}})$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and suppose $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}$. Suppose Π_{PPRF} satisfies punctured correctness. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

$$|\Pr[\mathsf{Hyb}_{2,i}^{(4)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i+1}^{(0)}(\mathcal{A}) = 1]| \leq 2/2^{\lambda+n}.$$

Proof. This follows by a similar argument as the proof of Claim 4.12. For completeness, we show that the programs associated with ObfProve and ObfVerify have identical behavior in the two experiments. The claim then follows by security of $i\mathcal{O}$ (as in the proof of Claim 4.12). Take any $i \in \{0, \ldots, 2^n - 1\}$ and consider an execution of $\mathsf{Hyb}_{2,i}^{(4)}$ and $\mathsf{Hyb}_{2,i+1}^{(0)}$. Let $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$. First, consider the case where $b^* = 0$.

The GenProof programs. When $b^* = 0$, by the identical analysis as in the proof of Claim 4.12, the programs $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1]$ in $\mathsf{Hyb}_{2,i}^{(4)}$ computes the same functionality as the program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}, k_1]$ in $\mathsf{Hyb}_{2,i+1}^{(0)}$.

The GenInst programs. Consider the programs $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1, k_{\mathsf{rerand}}^{(i)}, y_{\mathsf{base}}, y^*, i]$ in $\mathsf{Hyb}_{2,i}^{(4)}$ and $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, i+1]$ in $\mathsf{Hyb}_{2,i+1}^{(0)}$. Again, suppose $b^* = 0$:

- By punctured correctness, for all $x \neq i$, it follows that

$$\mathsf{F}(k_{b^*}^{(i)}, x) = \mathsf{F}(k_{b^*}, x) \quad \text{and} \quad \mathsf{F}(k_{\mathsf{rerand}}^{(i)}, x) = \mathsf{F}(k_{\mathsf{rerand}}, x).$$

  Thus, for all inputs $x \notin \{i, i+1\}$, the two programs have identical behavior.
- Suppose $x = i$. In this case, the $\mathsf{GenInstance}_2$ program in $\mathsf{Hyb}_{2,i}^{(4)}$ sets $y_{1-b} = y^*$ where $(y^*, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}}; r^*)$ and $r^* = \mathsf{F}(k_{\mathsf{rerand}}, i)$. This coincides with the behavior of the program in $\mathsf{Hyb}_{2,i+1}^{(0)}$.
- Suppose $x = i+1$. Let $b = \mathsf{F}(k_{\mathsf{sel}}, i+1)$. Then, the program in $\mathsf{Hyb}_{2,i}^{(4)}$ sets y_{1-b} as follows:
  - If $1-b = b^* = 0$, it computes $(y_{1-b}, z_{1-b}) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_{b^*}^{(i)}, i+1))$. By punctured correctness, $\mathsf{F}(k_{b^*}^{(i)}, i+1) = \mathsf{F}(k_{b^*}, i+1) = \mathsf{F}(k_0, i+1)$.

– If $1 - b = 1 - b^* = 1$, it computes $(y_{1-b}, z_{1-b}) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_1, i+1))$.

In particular, the program in $\mathsf{Hyb}_{2,i}^{(4)}$ sets $y_{1-b} = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_{1-b}, i+1))$. In $\mathsf{Hyb}_{2,i+1}^{(0)}$, the challenger sets $y_{1-b} = y^*$, where $y^* = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_{1-b}, i+1))$. Once more, the two programs have identical behavior.

Completing the proof of Claim 4.15. The above analysis shows that when $b^* = 0$, the GenProof and GenInst programs in $\mathsf{Hyb}_{2,i}^{(4)}$ and $\mathsf{Hyb}_{2,i+1}^{(0)}$ compute identical functionality. An analogous argument applies when $b^* = 1$. The claim now follows by security of $i\mathcal{O}$ (following the exact same structure as in the proof of Claim 4.12). □

Claim 4.17. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and suppose $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$. Suppose Π_{PPRF} satisfies punctured correctness. Then, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

$$|\Pr[\mathsf{Hyb}_{2,2^n}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

Proof. This follows by a similar argument as the proof of Claim 4.11. We first show that the programs $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}, y^*, 2^n]$ and $\mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}]$ in hybrids $\mathsf{Hyb}_{2,2^n}^{(0)}$ and Hyb_3, respectively, compute identical functionalities. Take any input $x \in \{0,1\}^n$. Let $b = \mathsf{F}(k_{\mathsf{sel}}, x)$.

- Consider the behavior of $\mathsf{GenInst}_2$. Since $x \in \{0,1\}^n$, it follows that $x < 2^n$. In this case, $\mathsf{GenInst}_2$ computes

$$(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$$
$$(y_{1-b}, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, x)).$$

- Consider the behavior of $\mathsf{GenInst}_1$. By definition, $\mathsf{GenInst}_1$ sets

$$(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$$
$$(y_{1-b}, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, x)).$$

Both experiments sample the quantities $\mathsf{crs}_{\mathsf{ROWF}}$, k_{sel}, k_0, k_1, k_{rerand}, and y_{base} using identical procedures. We conclude that the two programs compute identical functionality. The claim now follows via $i\mathcal{O}$ security (as in the proof of Claim 4.11). □

We now return to the proof of Lemma 4.10. By Claims 4.12 to 4.16, for all $i \in \{0, \ldots, 2^n - 1\}$, and all sufficiently-large $\lambda \in \mathbb{N}$, it follows that

$$|\Pr[\mathsf{Hyb}_{2,i}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i+1}^{(0)}(\mathcal{A}) = 1]| \leq 7/2^{\lambda + n(\lambda)}.$$

By the triangle inequality, this means that

$$|\Pr[\mathsf{Hyb}_{2,0}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,2^n}^{(0)}(\mathcal{A}) = 1]| \leq 2^{n(\lambda)} \cdot \frac{7}{2^{\lambda+n(\lambda)}} = \frac{7}{2^\lambda}.$$

Combined with Claims 4.11 and 4.17, we conclude that

$$|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq \frac{O(1)}{2^\lambda} = 2^{-\Omega(\lambda)}.$$ □

Lemma 4.18. *Suppose* Π_{ROWF} *is one-way,* $i\mathcal{O}$ *satisfies correctness, and* Π_{ROWF} *satisfies rerandomizable correctness. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \le \mathsf{negl}(\lambda).$$

Proof. Suppose $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] > \varepsilon(\lambda)$ for some non-negligible function ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on 1^λ and obtains the circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
2. Algorithm $\mathcal{B}$ computes $m = m(\lambda, n)$ and gives 1^m to the challenger. The challenger replies with a challenge $(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}})$.
3. Algorithm $\mathcal{B}$ computes $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$, and $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\kappa)$.
4. Algorithm $\mathcal{B}$ sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$ and constructs the obfuscated programs
$$\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1])$$
$$\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}]).$$
It gives $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$.
5. After $\mathcal{A}$ outputs a statement $x \in \{0,1\}^n$ and a proof $\pi = (b, z)$, algorithm $\mathcal{B}$ computes $(y, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, x))$ and outputs $\mathsf{R.RecoverSolution}(\mathsf{crs}_{\mathsf{ROWF}}, z, \mathsf{st})$.

By definition, the one-wayness challenger samples $\mathsf{crs}_{\mathsf{ROWF}} \leftarrow \mathsf{R.Setup}(1^\lambda, 1^m)$ and $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$, which matches the distribution in Hyb_3. Thus, with probability ε, algorithm $\mathcal{A}$ outputs (x, b, z) with the following properties:

$$b \ne \mathsf{F}(k_{\mathsf{sel}}, x) \quad \text{and} \quad \mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1,$$

where $(y_0, y_1) = \mathsf{ObfVerify}(x)$. By correctness of $i\mathcal{O}$,

$$(y_0, y_1) = \mathsf{ObfVerify}(x) = \mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{rerand}}, y_{\mathsf{base}}](x).$$

By definition of $\mathsf{GenInst}_1$, the instance y_b (for $b \ne \mathsf{F}(k_{\mathsf{sel}}, x)$) is computed as

$$(y_b, \mathsf{st}) = \mathsf{R.Rerandomize}(\mathsf{crs}_{\mathsf{ROWF}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, x)).$$

Since $\mathsf{R.Verify}(\mathsf{crs}_{\mathsf{ROWF}}, y_b, z) = 1$, we appeal to rerandomization correctness of Π_{ROWF} to conclude that $\mathsf{R.Verify}(\mathsf{crs}, y_{\mathsf{base}}, z^*) = 1$ when $z^* = \mathsf{R.RecoverSolution}(\mathsf{crs}_{\mathsf{ROWF}}, z, \mathsf{st})$. In this case, algorithm $\mathcal{B}$ wins the one-wayness game and $\mathsf{OWFAdv}_{\mathcal{B}}(\lambda) > \varepsilon(\lambda)$. □

Combining Lemmas 4.4, 4.9 and 4.10, we have for all sufficiently-large $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \ge \frac{1}{2} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - 2^{-\Omega(\lambda)}.$$

By Lemma 4.18, we have $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$. We conclude that

$$\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \le \mathsf{negl}(\lambda).$$

Since Hyb_0 corresponds to the real adaptive soundness security game, Theorem 4.3 follows. □

Theorem 4.19 (Succinctness)**.** *If Π_{ROWF} is succinct, then Construction 4.1 is succinct.*

Proof. A proof π in Construction 4.1 consists of a bit $b \in \{0,1\}$ and an element z output by algorithm $\mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}})$. Since Π_{ROWF} is succinct, there exists a fixed polynomial p such that $|z| \leq p(\lambda + \log m)$. Since $m(\lambda, n)$ in Construction 4.1 is a fixed polynomial in the security parameter λ and the statement length n and the statement length is always upper-bounded by the circuit size, it follows that $|\pi| \leq \mathrm{poly}(\lambda + \log |C|)$. □

Theorem 4.20 (Perfect Zero-Knowledge)**.** *If $i\mathcal{O}$ is correct, then Construction 4.1 satisfies perfect zero-knowledge.*

Proof. We construct the simulator as follows:

- $\mathcal{S}_0(1^\lambda, C)$: On input the security parameter λ and a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, the simulator samples the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C)$ exactly as in the real scheme. Let $k_{\mathsf{sel}}, k_0, k_1$ be the underlying PRF keys sampled in Setup. The simulator algorithm outputs crs together with the state $\mathsf{st}_{\mathcal{S}} = (k_{\mathsf{sel}}, k_0, k_1)$.
- $\mathcal{S}_1(\mathsf{st}_{\mathcal{S}}, x)$: On input the state $\mathsf{st}_{\mathcal{S}} = (k_{\mathsf{sel}}, k_0, k_1)$ and a statement $x \in \{0,1\}^n$, the simulator computes $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and $(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$. It then outputs (b, z_b).

Take any Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$. First, observe that the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{ROWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ output by $\mathcal{S}_0$ is distributed exactly as $\mathsf{Setup}(1^\lambda, C)$. Thus, it suffices to consider the simulated proofs. Consider any pair (x, w) where $C(x, w) = 1$. By construction, the proof $\pi = (b, z)$ output by $\mathsf{Prove}(\mathsf{crs}, x, w)$ is obtained by evaluating ObfProve on input (x, w). By correctness of $i\mathcal{O}$, the output of ObfProve on input (x, w) is the output of $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{ROWF}}, k_{\mathsf{sel}}, k_0, k_1]$ on input (x, w). By construction of GenProof, it computes $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and $(y_b, z_b) = \mathsf{R.GenInstance}(\mathsf{crs}_{\mathsf{ROWF}}; \mathsf{F}(k_b, x))$. This is how the simulator $\mathcal{S}_1$ constructs the proof and perfect zero-knowledge follows. □

Remark 4.21 (Katz-Wang Signatures in the Plain Model)**.** Our two-challenge approach for constructing adaptively-sound SNARGs shares a similar structure as the approach from Katz and Wang [KW03] for constructing adaptively-secure digital signatures in the random oracle model with a tight security reduction. In fact, our approach can be viewed as a way to "implement" the Katz-Wang proof strategy using an obfuscated PRF in place of the random oracle; as such, we obtain an adaptively-secure digital signature scheme in the plain model. We provide an overview of this relationship below:

- **The Katz-Wang signature scheme.** The core signature scheme is the short signature scheme of Boneh, Lynn, and Shacham [BLS01]. Let $(\mathbb{G}, \mathbb{G}_T)$ be a pairing group of prime order p. Let g be a generator of $\mathbb{G}$ and $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be an efficiently-computable non-trivial bilinear map. The public verification key in the signature scheme is $\mathsf{vk} = g^\alpha$ and the secret key is the exponent $\alpha \in \mathbb{Z}_p$ along with a PRF key k_{sel} (for a PRF F with one-bit outputs). A signature on a message m is then $(b, H(m,b)^\alpha)$, where $b = \mathsf{F}(k_{\mathsf{sel}}, m)$ and H is a hash function with codomain $\mathbb{G}$ (and modeled as a random oracle). To verify a signature (b, σ) on a message m with respect to the verification key $\mathsf{vk} = g^\alpha$, the verifier checks that
$$e(g^\alpha, H(m,b)) = e(g, \sigma).$$

  In the signature security proof, the reduction algorithm needs a way to (1) create a signature for *any* message (to answer signing queries), and (2) convert a successful forgery into a solution to a computational problem (in this case, the computational Diffie-Hellman problem (CDH) in $\mathbb{G}$). Katz and Wang achieve this through a two-challenge approach. For each message m, there are two possible signatures: $(0, H(m,0)^{\alpha})$ and $(1, H(m,1)^{\alpha})$. For each message, the reduction algorithm programs the outputs of the random oracle so for every m it knows $(b_m, H(m, b_m)^{\alpha})$ for some $b_m \in \{0,1\}$. It embeds the computational challenge into the value of $H(m, 1-b_m)$. This way, it has the ability to answer all signing queries, and simultaneously, if the adversary produces a valid forgery for any m^* with respect to bit $1-b_{m^*}$, then it solves the hard problem. Since the bit b_{m^*} associated with each message is pseudorandom and hidden from the adversary, this happens with probability $1/2$.

- **Replacing the random oracle with an obfuscated PRF.** Instead of using a random oracle to construct the challenge, we can replace it with an indistinguishability obfuscation of a puncturable PRF (i.e., as in the program $\mathsf{GenInst}$ from Construction 4.1). Then, by relying on (sub-exponential) hardness of $i\mathcal{O}$ and the puncturable PRF (by following an analogous structure as the proof of Theorem 4.3) as well as the hardness of CDH (as in the Katz-Wang construction), we obtain an adaptively-secure digital signature scheme with a tight reduction to the CDH problem.

Thus, our techniques provide a way to instantiate the Katz-Wang techniques for arguing adaptive security in the *plain* model *without* random oracles through the use of obfuscation. Of course, using indistinguishability obfuscation in place of the random oracle will incur significant overhead in the size of the public verification key. Nonetheless, our result highlights an interesting conceptual point that it is possible to instantiate the random oracle with a concrete hash function and base hardness on (standard) cryptographic assumptions in the plain model. Previously, [HSW14] showed how to replace the random oracle with indistinguishability obfuscation in the setting of full-domain hash signatures.

5 Constructing Rerandomizable One-Way Functions

In this section, we describe two constructions of rerandomizable one-way functions from classic number-theoretic assumptions. Our first construction is based on the discrete log assumption and the second is based on the hardness of computing modular square roots (which reduces to factoring [Rab79]). Both constructions rely on random self-reducibility.

5.1 Rerandomizable One-Way Function from Discrete Log

In this section, we show how to construct a rerandomizable one-way function from discrete log. We begin by recalling the discrete log assumption in prime-order groups.

Notation. For a positive integer $p > 1$, we write $\mathbb{Z}_p$ to denote the set of integers $\{0, \ldots, p-1\}$. We write $\mathbb{Z}_p^*$ to denote the multiplicative group of integers modulo p.

Definition 5.1 (Prime-Order Group Generator). Let λ be a security parameter. A prime-order group generator is an efficient algorithm GroupGen that takes as input a security parameter 1^λ and outputs the description $\mathcal{G} = (\mathbb{G}, p, g)$ of a group $\mathbb{G}$ of prime order $p = 2^{\Theta(\lambda)}$ and generated by $g \in \mathbb{G}$. Moreover, we require that the group operation in $\mathbb{G}$ be efficiently-computable.

Definition 5.2 (Discrete Log Assumption). Let GroupGen be a prime-order group generator. We say that the discrete log assumption holds with respect to GroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(1^\lambda, \mathcal{G}, g^x) = x : \mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda), x \xleftarrow{\mathrm{R}} \mathbb{Z}_p\right] \le \mathsf{negl}(\lambda).$$

Construction 5.3 (Rerandomizable One-Way Functions from Discrete Log)**.** Let GroupGen be a prime-order group generator. We construct a rerandomizable one-way function $\Pi_{\mathsf{ROWF}} = (\mathsf{Setup}, \mathsf{GenInstance}, \mathsf{Rerandomize}, \mathsf{Verify}, \mathsf{RecoverSolution})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^m)$: On input the security parameter λ and a rerandomization parameter m, the setup algorithm samples $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and outputs $\mathsf{crs} = \mathcal{G}$.
- $\mathsf{GenInstance}(\mathsf{crs})$: On input the common reference string $\mathsf{crs} = (\mathbb{G}, p, g)$, the instance-generator algorithm samples $z \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ and outputs (g^z, z).[7]
- $\mathsf{Rerandomize}(\mathsf{crs}, y)$: On input the common reference string $\mathsf{crs} = (\mathbb{G}, p, g)$ and an instance $y \in \mathbb{G}$, the rerandomization algorithm samples $r \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$ and outputs (y^r, r).
- $\mathsf{Verify}(\mathsf{crs}, y, z)$: On input the common reference string $\mathsf{crs} = (\mathbb{G}, p, g)$, an instance $y \in \mathbb{G}$, and a candidate solution $z \in \mathbb{Z}_p^*$, the verification algorithm outputs 1 if $y = g^z$.
- $\mathsf{RecoverSolution}(\mathsf{crs}, z', \mathsf{st})$: On input the common reference string $\mathsf{crs} = (\mathbb{G}, p, g)$, a solution $z' \in \mathbb{Z}_p^*$, and the rerandomization state $\mathsf{st} = r \in \mathbb{Z}_p^*$, the solution-recovery algorithm outputs $z' r^{-1} \in \mathbb{Z}_p^*$.

Theorem 5.4 (Correctness)**.** *Construction 5.3 is correct.*

Proof. Take any $\lambda, m \in \mathbb{N}$ and let $\mathsf{crs} = (\mathbb{G}, p, g) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. By construction, this means $z \in \mathbb{Z}_p^*$ and $y = g^z$. As such, $\mathsf{Verify}(\mathsf{crs}, y, z) = 1$. □

Theorem 5.5 (Rerandomization Correctness)**.** *Construction 5.3 satisfies rerandomization correctness.*

Proof. Take any $\lambda, m \in \mathbb{N}$ and any common reference string $\mathsf{crs} = (\mathbb{G}, p, g)$ in the support of $\mathsf{Setup}(1^\lambda, 1^m)$. Take any (y, z) in the support of $\mathsf{GenInstance}(\mathsf{crs})$ and (y', st) in the support of $\mathsf{Rerandomize}(\mathsf{crs}, y)$. By construction of GenInstance, this means that $z \in \mathbb{Z}_p^*$ and $y = g^z$. Similarly, by construction of Rerandomize, we have that $\mathsf{st} = r \in \mathbb{Z}_p^*$ and $y' = y^r$. Consider any z' where $\mathsf{Verify}(\mathsf{crs}, y', z') = 1$. This means $z' \in \mathbb{Z}_p^*$ and moreover, $g^{z'} = y' = y^r = g^{zr}$. Since $r \in \mathbb{Z}_p^*$, this means that $z = z' r^{-1} \in \mathbb{Z}_p^*$ and $\mathsf{Verify}(\mathsf{crs}, y, z' r^{-1}) = 1$. Since $\mathsf{RecoverSolution}(\mathsf{crs}, z', \mathsf{st})$ outputs $z' r^{-1}$, the claim holds. □

Theorem 5.6 (One-Wayness)**.** *If the discrete log assumption holds with respect to* GroupGen*, then Construction 5.3 is one-way.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ where $\mathsf{OWFAdv}_{\mathcal{A}}(\lambda) > \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the discrete log problem:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives the security parameter 1^λ, the group $\mathcal{G} = (\mathbb{G}, p, g)$ and the challenge $h \in \mathbb{G}$. If $h = g^0$, then algorithm $\mathcal{B}$ outputs 0.
2. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ on the security parameter 1^λ. Algorithm $\mathcal{A}$ outputs the rerandomization parameter 1^m, and algorithm $\mathcal{B}$ replies with $\mathsf{crs} = (\mathbb{G}, p, g)$ and the instance $h \in \mathbb{G}$.
3. After algorithm $\mathcal{A}$ outputs a solution z, algorithm $\mathcal{B}$ also outputs z.

[7] It is important that GenInstance samples the challenge from $\mathbb{Z}_p^*$ and *not* $\mathbb{Z}_p$. Our proof of rerandomization security will critically rely on this distinction.

By construction, the discrete log challenger samples $\mathsf{crs} = (\mathbb{G}, p, g) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$, $x \xleftarrow{R} \mathbb{Z}_p$, and sets $h = g^x$. If $h = g^0$, then algorithm $\mathcal{B}$ solves the discrete log problem. If $x \neq 0$, then x is uniformly distributed over $\mathbb{Z}_p^*$, so algorithm $\mathcal{B}$ perfectly simulates the one-wayness game for $\mathcal{A}$. In this case, with probability at least ε, algorithm $\mathcal{A}$ outputs $z \in \mathbb{Z}_p^*$ such that $\mathsf{Verify}(\mathsf{crs}, h, z) = 1$, or equivalently, z such that $h = g^z$. But in this case, algorithm $\mathcal{B}$ also solves the discrete log problem. We conclude that algorithm $\mathcal{B}$ succeeds in solving the discrete log problem with the same non-negligible advantage ε. □

Theorem 5.7 (Rerandomization Security)**.** *Construction 5.3 satisfies perfect rerandomizable security. Namely, for all polynomials $m = m(\lambda)$ and all adversaries $\mathcal{A}$,* $\mathsf{RerandAdv}_{\mathcal{A},m}(\lambda) = 0$.

Proof. Take any polynomial $m = m(\lambda)$. Let $\mathsf{crs} = (\mathbb{G}, p, g) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$. Sample $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. This means that $z_{\mathsf{base}} \xleftarrow{R} \mathbb{Z}_p^*$ and $y_{\mathsf{base}} = g^{z_{\mathsf{base}}}$. Suppose $(y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$ and $(y', \mathsf{st}) \leftarrow \mathsf{Rerandomize}(\mathsf{crs}, y_{\mathsf{base}})$. We argue that $(\mathsf{crs}, y_{\mathsf{base}}, y)$ is distributed identically to $(\mathsf{crs}, y_{\mathsf{base}}, y')$:

- By construction of $\mathsf{GenInstance}$, the distribution of z is uniform over $\mathbb{Z}_p^*$ so $y = g^z$ is uniform over $\mathbb{G} \setminus \{g^0\}$.
- By construction of $\mathsf{Rerandomize}$, the distribution of $\mathsf{st} = r$ is uniform over $\mathbb{Z}_p^*$. Next $y' = y_{\mathsf{base}}^r = g^{r z_{\mathsf{base}}}$. Since $z_{\mathsf{base}} \in \mathbb{Z}_p^*$, it follows that $z_{\mathsf{base}} \neq 0$.[8] This means that the distribution of $z_{\mathsf{base}} r$ is uniform over $\mathbb{Z}_p^*$ and so y' is uniform over $\mathbb{G} \setminus \{g^0\}$ (and *independent* of y_{base}).

We conclude that the joint distribution of $(\mathsf{crs}, y_{\mathsf{base}}, y)$ and $(\mathsf{crs}, y_{\mathsf{base}}, y')$ are identically distributed and the claim follows. □

Theorem 5.8 (Succinctness)**.** *Construction 5.3 is succinct.*

Proof. Take any $\lambda, m \in \mathbb{N}$ and let $\mathsf{crs} = (\mathbb{G}, p, g) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. By construction of Setup, $(\mathbb{G}, p, g)$ is output by $\mathsf{GroupGen}(1^\lambda)$. This means $p = 2^{\Theta(\lambda)}$. By construction of $\mathsf{GenInstance}$, this means $z \in \mathbb{Z}_p^*$ so $|z| = \Theta(\lambda)$. □

5.2 Rerandomizable One-Way Functions from Computing Modular Square Roots

In this section, we show how to construct a rerandomizable one-way function from factoring. Specifically, we base hardness on the hardness of computing modular square roots, which is equivalent to the factoring problem. We begin by recalling the computational assumptions we use:

Definition 5.9 (Composite Modulus Sampler)**.** Let λ be a security parameter. A composite-modulus sampler is an efficient algorithm $\mathsf{SampleN}$ that takes as input the security parameter 1^λ and outputs (N, p, q) where $N = pq$ and p, q are *distinct* λ-bit primes (i.e., $p, q \in [2^{\lambda-1}, 2^\lambda - 1]$).

Definition 5.10 (Hardness of Factoring)**.** Let $\mathsf{SampleN}$ be a composite-modulus sampler. Factoring is hard with respect to $\mathsf{SampleN}$ if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathcal{A}(1^\lambda, N) \in \{p, q\} : (N, p, q) \leftarrow \mathsf{SampleN}(1^\lambda)] = \mathsf{negl}(\lambda).$$

[8]This is where we use the fact that z_{base} is drawn from $\mathbb{Z}_p^*$ and not $\mathbb{Z}_p$. If we sampled z_{base} from $\mathbb{Z}_p$, then these two distributions have a statistical distance of $1/p = 1/2^{\Theta(\lambda)}$, which may not be small enough relative to the rerandomization parameter m.

Definition 5.11 (Hardness of Computing Modular Square Roots)**.** Let SampleN be a composite-modulus sampler. Computing modular square roots is hard if with respect to SampleN if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[z^2 = x^2 \bmod N : \begin{array}{c} (N,p,q) \leftarrow \mathsf{SampleN}(1^\lambda), x \xleftarrow{\mathrm{R}} \mathbb{Z}_N \\ z \leftarrow \mathcal{A}(1^\lambda, N, x^2) \end{array}\right] = \mathsf{negl}(\lambda).$$

Fact 5.12 (Computing Modular Square Roots is Equivalent to Factoring [Rab79])**.** For every composite-modulus sampler SampleN, factoring is hard with respect to SampleN if and only if computing modular square roots is hard with respect to SampleN.

Construction 5.13 (Rerandomizable One-Way Functions from Factoring)**.** Let SampleN be a composite-modulus sampler. We construct a rerandomizable one-way function $\Pi_{\mathsf{ROWF}} = (\mathsf{Setup}, \mathsf{GenInstance}, \mathsf{Rerandomize}, \mathsf{Verify}, \mathsf{RecoverSolution})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^m)$: On input the security parameter $\lambda \in \mathbb{N}$ and a rerandomization parameter $m \in \mathbb{N}$, the setup algorithm samples $(N,p,q) \leftarrow \mathsf{SampleN}(1^\lambda)$. It outputs the common reference string $\mathsf{crs} = (1^m, N)$.
- $\mathsf{GenInstance}(\mathsf{crs})$: On input the common reference string $\mathsf{crs} = (1^m, N)$, the instance-generator algorithm proceeds as follows:
  - Sample $x_1, \ldots, x_m \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. If $\gcd(x_i, N) \neq 1$ for all $i \in [m]$, then output $(1,1)$.
  - Otherwise, take the smallest such $i \in [m]$ where $\gcd(x_i, N) = 1$ and output $(x_i^2 \bmod N, x_i)$.
- $\mathsf{Rerandomize}(\mathsf{crs}, y)$: On input the common reference string $\mathsf{crs} = (1^m, N)$ and an instance $y \in \mathbb{Z}_N$, the rerandomizable algorithm does the following:
  - Sample $r_1, \ldots, r_m \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. If $\gcd(r_i, N) \neq 1$ for all $i \in [m]$, then output $(y, 1)$.
  - Otherwise, take the smallest such $i \in [m]$ where $\gcd(r_i, N) = 1$ and output $(y r_i^2 \bmod N, r_i)$.
- $\mathsf{Verify}(\mathsf{crs}, y, z)$: On input the common reference string $\mathsf{crs} = (1^m, N)$, an instance $y \in \mathbb{Z}_N$, and a candidate solution $z \in \mathbb{Z}_N$, the verification algorithm outputs 1 if $y = z^2 \bmod N$.
- $\mathsf{RecoverSolution}(\mathsf{crs}, z', \mathsf{st})$: On input the common reference string $\mathsf{crs} = (1^m, N)$, a solution $z' \in \mathbb{Z}_N$, and the rerandomizable state $\mathsf{st} = r \in \mathbb{Z}_N^*$, the solution-recovery algorithm outputs $z'/r \bmod N$.

Theorem 5.14 (Correctness)**.** *Construction 5.13 is correct.*

Proof. Take any $\lambda, m \in \mathbb{N}$ and let $\mathsf{crs} = (1^m, N) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. By construction, this means $y = z^2 \bmod N$, and $\mathsf{Verify}(\mathsf{crs}, y, z) = 1$. ☐

Theorem 5.15 (Rerandomization Correctness)**.** *Construction 5.13 satisfies rerandomization correctness.*

Proof. Take any $\lambda, m \in \mathbb{N}$ and any common reference string $\mathsf{crs} = (1^m, N)$ in the support of $\mathsf{Setup}(1^\lambda, 1^m)$. Take any (y, z) in the support of $\mathsf{GenInstance}(\mathsf{crs})$ and (y', st) in the support of $\mathsf{Rerandomize}(\mathsf{crs}, y)$. By construction of GenInstance, this means $\gcd(N, z) = 1$. This means $z \in \mathbb{Z}_N^*$ and $y = z^2 \bmod N$. By construction of Rerandomize, we have that $\mathsf{st} = r \in \mathbb{Z}_N^*$ and $y' = yr^2 \bmod N$. Consider any z' where $\mathsf{Verify}(\mathsf{crs}, y', z') = 1$. This means $y' = (z')^2 \bmod N$. Thus,

$$yr^2 = (z')^2 \bmod N \implies y = (z'/r)^2 \bmod N.$$

Here, r is invertible since $r \in \mathbb{Z}_N^*$. Now $\mathsf{RecoverSolution}(\mathsf{crs}, z', \mathsf{st})$ outputs z'/r and $\mathsf{Verify}(\mathsf{crs}, y, z'/r) = 1$, so the claim holds. ☐

Theorem 5.16 (One-Wayness)**.** *If computing modular square roots is hard, then Construction 5.13 is one-way.*

Proof. We start by defining a sequence of hybrid experiments:

- Hyb_0: This is the real one-wayness experiment. Namely, on input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs 1^m. Then, the challenger does the following:
  - Sample $(N, p, q) \leftarrow \mathsf{SampleN}(1^\lambda)$.
  - Sample $x_1, \ldots, x_m \xleftarrow{R} \mathbb{Z}_N$. If $\gcd(x_i, N) \neq 1$ for all $i \in [m]$, the challenger sets the challenge to be $y = 1$. Otherwise, it takes the smallest such $i \in [m]$ where $\gcd(x_i, N) = 1$ and sets the challenge to be $y = x_i^2 \bmod N$.

  The challenger gives crs and y to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a solution $z \in \mathbb{Z}_N$ and the output of the experiment is 1 if $y = z^2 \bmod N$.
- Hyb_1: Same as Hyb_0, except to generate the challenge $y \in \mathbb{Z}_N$, the challenger instead samples $x \xleftarrow{R} \mathbb{Z}_N^*$ and sets $y = x^2 \bmod N$. The rest of the experiment proceeds as in Hyb_0.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output distribution of Hyb_i with adversary $\mathcal{A}$. We now analyze the experiments.

Lemma 5.17. *For all adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| \leq 2^{-\Omega(\lambda)}.$$

Proof. We argue that the distribution of y in the two experiments are statistically close. Consider the distribution of y in Hyb_0. Let $x_1, \ldots, x_k \xleftarrow{R} \mathbb{Z}_N$ be the values sampled by the challenger in Hyb_0. We consider two possibilities:

- Suppose for all $i \in [m]$, $\gcd(x_i, N) \neq 1$. Since $N = pq$, the primes p, q are λ-bits each, and each $x_i \xleftarrow{R} \mathbb{Z}_N$, the probability that $\gcd(x_i, N) \neq 1$ is

$$\Pr[\gcd(x_i, N) \neq 1 : x_i \xleftarrow{R} \mathbb{Z}_N] = \frac{p + q - 1}{pq} = 2^{-\Omega(\lambda)}. \tag{5.1}$$

  Thus, this case happens with probability at most $2^{-\Omega(\lambda)}$.
- Suppose there exists an $i \in [m]$ where $\gcd(x_i, N) = 1$. For all such i, the distribution of x_i is uniform over $\mathbb{Z}_N^*$. In this case, the challenger sets $y = x_i^2 \bmod N$. This is exactly the distribution of y in Hyb_1.

We conclude that the statistical distance between the distribution of $(1^m, N, y)$ in Hyb_0 and Hyb_1 is at most $2^{-\Omega(\lambda)}$ and the claim holds. □

To complete the proof of Theorem 5.16, we show that under the hardness of computing modular square roots with respect to SampleN, $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$ for all efficient adversaries $\mathcal{A}$. To show this, suppose there exists an efficient adversary $\mathcal{A}$ where $\mathsf{OWFAdv}_{\mathcal{A}}(\lambda) > \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for computing modular square roots:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives the security parameter 1^λ, the common reference string $\mathsf{crs} = (1^m, N)$, and the challenge $y \in \mathbb{Z}_N$. Algorithm $\mathcal{B}$ first checks if $\gcd(y, N) = 1$. If $\gcd(y, N) \neq 1$, then algorithm $\mathcal{B}$ aborts with output $\perp$.[9]

[9]Technically, algorithm $\mathcal{B}$ learns a factor of N in this case and can use the factorization of N to obtain a square root y. However, since the event $\gcd(y, N) = 1$ happens with negligible probability, it also suffices to ignore this case and simplify the analysis.

2. If $\gcd(y, N) = 1$, then algorithm $\mathcal{B}$ runs $\mathcal{A}$ on the security parameter 1^λ. Algorithm $\mathcal{A}$ outputs the rerandomization parameter 1^m and algorithm $\mathcal{B}$ replies with $\mathsf{crs} = (1^m, N)$ and the instance $y \in \mathbb{Z}_N$.

3. After algorithm $\mathcal{A}$ outputs a solution z, algorithm $\mathcal{B}$ also outputs z.

By construction, the challenger samples $(N, p, q) \leftarrow \mathsf{SampleN}(1^\lambda)$ and $y = x^2$ where $x \xleftarrow{R} \mathbb{Z}_N$. We consider two possibilities:

- Suppose $\gcd(x, N) \neq 1$. From Eq. (5.1), this case happens with probability $2^{-\Omega(\lambda)}$.
- Suppose $\gcd(x, N) = 1$. Then the distribution of x is uniform over $\mathbb{Z}_N^*$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_1 for $\mathcal{A}$. By assumption, with probability at least ε, algorithm $\mathcal{A}$ outputs $z \in \mathbb{Z}_N$ such that $y = z^2 \bmod N$. In this case, algorithm $\mathcal{B}$ successfully outputs a square root of $y \bmod N$.

Thus, we conclude that algorithm $\mathcal{B}$ succeeds with probability at least $\varepsilon(1 - 2^{-\Omega(\lambda)}) = \varepsilon - \mathsf{negl}(\lambda)$, which is non-negligible. Thus, we conclude that for all efficient adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$. Combined with Lemma 5.17, this means that for all efficient adversaries $\mathcal{A}$,

$$\mathsf{OWFAdv}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \mathsf{negl}(\lambda). \qquad \square$$

Theorem 5.18 (Rerandomization Security). *Construction 5.13 satisfies $2^{-\Omega(m)}$-statistical rerandomizable security. Namely, for all polynomials $m = m(\lambda)$ and all adversaries $\mathcal{A}$, $\mathsf{RerandAdv}_{\mathcal{A},m}(\lambda) \leq 2^{-\Omega(m(\lambda))}$.*

Proof. Take any polynomial $m = m(\lambda)$. We define three distributions:

- $\mathcal{D}_0$: Sample $\mathsf{crs} = (1^m, N) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$, $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$, and $(y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. Output $(\mathsf{crs}, y_{\mathsf{base}}, y)$.
- $\mathcal{D}_1$: Sample $\mathsf{crs} = (1^m, N) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. Sample $x \xleftarrow{R} \mathbb{Z}_N^*$ and set $y = x^2 \bmod N$. Output $(\mathsf{crs}, y_{\mathsf{base}}, y)$.
- $\mathcal{D}_2$: Sample $\mathsf{crs} = (1^m, N) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$, $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$, and $(y, \mathsf{st}) \leftarrow \mathsf{Rerandomize}(\mathsf{crs}, y_{\mathsf{base}})$. Output $(\mathsf{crs}, y_{\mathsf{base}}, y)$.

We argue that the statistical distance between each pair of distributions is bounded by $2^{-\Omega(m)}$:

- Consider distributions $\mathcal{D}_0$ and $\mathcal{D}_1$. Since crs and y_{base} are identically distributed in the two distributions, it suffices to consider the distribution of y. In $\mathcal{D}_0$, the $\mathsf{GenInstance}$ algorithm samples $x_1, \ldots, x_m \xleftarrow{R} \mathbb{Z}_N$ and outputs $y = x_i^2 \bmod N$ if there exists some $i \in [m]$ where $\gcd(x_i, N) = 1$. In this case, the distribution of x_i (given crs and y_{base}) in $\mathcal{D}_0$ is uniform over $\mathbb{Z}_N^*$, and the distribution of y is distributed exactly as in $\mathcal{D}_1$. The only setting where the two distributions differ is if in $\mathcal{D}_0$, for all $i \in [m]$, $\gcd(x_i, N) \neq 1$. Since $N = pq$, the primes p, q are λ-bits each, and each $x_i \xleftarrow{R} \mathbb{Z}_N$, the probability that $\gcd(x_i, N) = 1$ for all $i \in [m]$ is

$$\Pr[\forall i \in [m] : \gcd(x_i, N) = 1 \mid x_i \xleftarrow{R} \mathbb{Z}_N] = \left(\frac{p + q - 1}{pq}\right)^m \leq 2^{-\Omega(m)}.$$

  Thus, the statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_1$ is at most $2^{-\Omega(m)}$.

- Consider distributions $\mathcal{D}_1$ and $\mathcal{D}_2$. Again, it suffices to consider the distribution of y in the two experiments. By construction of GenInstance, in distribution $\mathcal{D}_2$, it is the case that $z_{\mathrm{base}} \in \mathbb{Z}_N^*$ and $y_{\mathrm{base}} = z_{\mathrm{base}}^2 \bmod N$. In $\mathcal{D}_2$, the Rerandomize algorithm samples $r_1, \ldots, r_m \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and outputs $y = y_{\mathrm{base}} r_i^2$ if there exists some $i \in [m]$ where $\gcd(r_i, N) = 1$. In this case, we can write y as $y = (z_{\mathrm{base}} r_i)^2$ where $r_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N^*$, which coincides exactly with the distribution of y in $\mathcal{D}_1$. Thus, the only setting where the two distributions differ is if in $\mathcal{D}_2$, for all $i \in [m]$, $\gcd(r_i, N) \neq 1$. By the same calculation as in the previous case, this happens with probability at most $2^{-\Omega(m)}$, and so the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is at most $2^{-\Omega(m)}$.

Since the statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_1$ as well as that between $\mathcal{D}_1$ and $\mathcal{D}_2$ is at most $2^{-\Omega(m)}$, it follows that the statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_2$ is also bounded by $2^{-\Omega(m)}$. □

Theorem 5.19 (Succinctness). *Construction 5.13 is succinct.*

Proof. Take any $\lambda, m \in \mathbb{N}$ and let $\mathsf{crs} = (1^m, N) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. By construction of Setup, $N = pq$ where p, q are λ-bit primes. By construction of GenInstance, this means $z \in \mathbb{Z}_N$ so $|z| = \log N \leq 2\lambda$. □

Acknowledgments

Brent Waters is supported by NSF CNS-1908611, CNS-2318701, and a Simons Investigator award. David J. Wu is supported by NSF CNS-2151131, CNS-2140975, CNS-2318701, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[ACL+22] Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri Aravinda Krishnan Thyagarajan. Lattice-based SNARKs: Publicly verifiable, preprocessing, and recursively composable - (extended abstract). In *CRYPTO*, pages 102–132, 2022.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, pages 252–283, 2023.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *STOC*, pages 505–514, 2014.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519, 2014.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *STOC*, pages 474–482, 2017.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. In *EUROCRYPT*, pages 247–277, 2017.

[BISW18] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal SNARGs via linear multi-prover interactive proofs. In *EUROCRYPT*, pages 222–255, 2018.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*, pages 514–532, 2001.

[BP04] Boaz Barak and Rafael Pass. On the possibility of one-message weak zero-knowledge. In *TCC*, pages 121–132, 2004.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, pages 280–300, 2013.

[CGJ$^+$23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and snargs from sub-exponential DDH. In *CRYPTO*, pages 635–668, 2023.

[CGKS23] Matteo Campanelli, Chaya Ganesh, Hamidreza Khoshakhlagh, and Janno Siim. Impossibilities in succinct arguments: Black-box extraction and more. In *AFRICACRYPT*, pages 465–489, 2023.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *CRYPTO*, pages 394–423, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *FOCS*, pages 68–79, 2021.

[CLM23] Valerio Cini, Russell W. F. Lai, and Giulio Malavolta. Lattice-based succinct arguments from vanishing polynomials - (extended abstract). In *CRYPTO*, pages 72–105, 2023.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered abe, flexible broadcast, and more. In *CRYPTO*, pages 498–531, 2023.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *CRYPTO*, pages 276–288, 1984.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, pages 626–645, 2013.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In *EUROCRYPT*, pages 171–188, 2009.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *EUROCRYPT*, pages 201–220, 2014.

[JJ22] Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *FOCS*, pages 1023–1034, 2022.

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In *STOC*, pages 708–721, 2021.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, pages 60–73, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In *EUROCRYPT*, pages 670–699, 2022.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[KLV23] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs and PPAD hardness from the decisional diffie-hellman assumption. In *EUROCRYPT*, pages 470–498, 2023.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In *STOC*, pages 1545–1552, 2023.

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In *TCC*, pages 91–118, 2016.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, pages 669–684, 2013.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *STOC*, pages 1115–1124, 2019.

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO*, pages 143–159, 2009.

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In *TCC*, pages 330–368, 2021.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In *ACM CCS*, pages 155–164, 2003.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *ASIACRYPT*, pages 41–60, 2013.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

[Rab79] Michael Rabin. Digitalized signatures and public-key functions as intractable as factorization. 1979.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[Wee05] Hoeteck Wee. On round-efficient argument systems. In *ICALP*, pages 140–152, 2005.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, pages 433–463, 2022.