A Pure Indistinguishability Obfuscation Approach to Adaptively-Sound SNARGs for NP

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

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

Abstract

We construct an adaptively-sound succinct non-interactive argument (SNARG) for NP in the CRS model from sub-exponentially-secure indistinguishability obfuscation ($i\mathcal{O}$) and sub-exponentially-secure one-way functions. Previously, Waters and Wu (STOC 2024), and subsequently, Waters and Zhandry (CRYPTO 2024) showed how to construct adaptively-sound SNARGs for NP by relying on sub-exponentially-secure indistinguishability obfuscation, one-way functions, and an *additional* algebraic assumption (i.e., discrete log, factoring, or learning with errors). In this work, we show that no additional algebraic assumption is needed and vanilla (sub-exponentially-secure) one-way functions already suffice in combination with $i\mathcal{O}$.

We first give a direct construction of an adaptively-sound SNARG for NP assuming (sub-exponentially-secure) $i\mathcal{O}$ and an *injective* one-way function. Then, we show that it suffices to have an injective one-way function that has an *inefficient* sampler (i.e., sampling a challenge for the one-way function requires super-polynomial time). Because we rely on the existence of injective one-way functions only in the security proof and *not* in the actual construction, having an inefficient sampling procedure does not impact correctness. We then show that injective one-way functions with an inefficient sampler can be built generically from any vanilla one-way function. Our approach may be independently useful in other settings to replace injective one-way functions with standard one-way functions in applications of $i\mathcal{O}$.

1 Introduction

Succinct non-interactive arguments (SNARGs) for NP [Mic94, GW11] allow a prover to convince a verifier that an NP statement is true with a proof that is much shorter than the length of the NP statement and witness. Micali [Mic94] (building on the work of Kilian [Kil92]) gave the first SNARG for NP in the random oracle model. Subsequently, a long sequence of works have constructed SNARGs for NP from various *non-falsifiable* assumptions in the common reference string (CRS) model [Gro10, BCCT12, DFH12, BCCT13, Lip13, GGPR13, BCI+13, BCPR14, BISW17, BCC+17, BISW18, ACL+22, CLM23]. In the CRS model, the prover and verifier have access to a (trusted) reference string (or random string). Another line of work has shown how to construct SNARGs for subsets of NP, such as P [GZ21, CJJ21b, KVZ21, KLVW23, WW24b], batch NP [CJJ21a, CJJ21b, WW22, CGJ+23, KLVW23], monotone policy batch NP [BBK+23, NWW23], and NP languages with propositional proofs of non-membership [JKLV24] from standard falsifiable assumptions.

A natural question is whether we can construct SNARGs for general NP from *falsifiable* assumptions. The first construction of a SNARG for NP from (sub-exponentially) falsifiable assumptions was due to Sahai and Waters [SW14], who gave a construction from indistinguishability obfuscation ($i\mathcal{O}$) and one-way functions. A limitation of the Sahai-Waters scheme was that it was only shown to be *non-adaptively* sound, where soundness only holds against an adversary that must declare the false statement it provides a proof for *before* seeing the CRS. The natural notion of security is *adaptive* soundness, where the adversary can choose which statement it wants to prove *after* seeing the CRS.

Adaptively-sound SNARGs. Building adaptively-sound SNARGs for NP from falsifiable assumptions is a challenging problem, and any such result must circumvent known black-box separations [GW11, CGKS23]. Very recently, Waters and Wu showed how to construct an adaptively-sound SNARG for NP from $i\mathcal{O}$ together with a *re-randomizable* one-way function [WW24a]; they also showed how to construct the re-randomizable one-way function from the discrete log or factoring assumptions. Subsequently, Waters and Zhandry [WZ24] showed how to replace the rerandomizable one-way function with a lossy function, which notably enabled an instantiation from standard lattice assumptions. Both of these works circumvent the Gentry-Wichs impossibility by relying on security reductions whose running time is *exponential* in the length of the NP witness. The challenge addressed in these works is to ensure the overhead of the complexity leveraging only manifests in the size of the CRS and *not* the size of the proof.

Both constructions of adaptively-sound SNARGs for NP rely on $i\mathcal{O}$ in conjunction with a number-theoretic assumption, or analogously, some source of algebraic structure. In the case of [WW24a], the algebraic assumption is used to construct a re-randomizable one-way function. The security analysis from [WW24a] critically relied on perfect (or statistical) re-randomizability for the one-way function. Constructing a one-way function with this property seems challenging from an unstructured assumption. In fact, it was not even apparent how to use lattice assumptions (e.g., short integer solutions (SIS) [Ajt96] or learning with errors (LWE) [Reg05]) to construct the necessary re-randomizable one-way function. Indeed, to obtain an adaptively-sound SNARG for NP from $i\mathcal{O}$ and standard lattice assumptions, the subsequent work of Waters and Zhandry [WZ24] showed instead how to replace the re-randomizable one-way function in the original [WW24a] construction with a *lossy* function, which can in turn be built from LWE. At the same time, the [WZ24] approach replaced one algebraic primitive with a simpler, but still algebraic, primitive.

Motivating a pure $i\mathcal{O}$ approach for adaptive soundness. A simpler assumption we could use alongside $i\mathcal{O}$ would be (vanilla) one-way functions (i.e., an *unstructured* source of hardness).[1] We refer to such an approach as a "pure $i\mathcal{O}$" approach which we motivate below.

First, while our current $i\mathcal{O}$ constructions require algebraic assumptions [JLS21, JLS22], constructing provably-secure $i\mathcal{O}$ is a continually-evolving field as witnessed by the recent work of Ragavan, Vafa, and Vaikuntanathan [RVV24]. In the future, we might have new $i\mathcal{O}$ constructions based on general assumptions or possibly algebraic assumptions that do not imply the strong versions of re-randomizable one-way functions or lossy functions needed by the [WW24a, WZ24] constructions. For example, neither of these primitives are known to follow from the learning parity with noise (LPN) assumption. A pure $i\mathcal{O}$ approach to building SNARGs (or for that matter, any cryptographic primitive) would not impose any further assumptions than those already needed to achieve indistinguishability obfuscation.

Second, we believe that achieving or attempting to achieve a pure $i\mathcal{O}$ approach is critical to obtaining a deep understanding of a primitive. While many primitives can be built from $i\mathcal{O}$ and one-way functions, there are notable exceptions such as collision-resistant hash functions and homomorphic encryption [AS15].

Finally, achieving a pure $i\mathcal{O}$ solution to adaptively-secure SNARGs matches what is known for SNARGs in weaker models and shows that we do not need to make any concessions to achieve adaptive soundness.

[1] Recall that $i\mathcal{O}$ alone cannot imply one-way functions since in a world where $\mathsf{P} = \mathsf{NP}$, indistinguishability obfuscation exists unconditionally while one-way functions do not exist. Thus, to leverage $i\mathcal{O}$ in cryptographic applications, we typically need to additionally assume a source of cryptographic hardness (e.g., the existence of one-way functions). The extra assumption can sometimes be replaced by a weaker worst-case assumption [KMN+14].

Specifically, the original non-adaptively-sound SNARG by Sahai and Waters [SW14] was based on a pure $i\mathcal{O}$ approach, and more recently, the work of [MPV24] showed that the Sahai-Waters construction is also adaptively-secure in the *designated-verifier* model (i.e., where a secret key is needed to verify proofs). Note though that the [MPV24] adaptive soundness analysis critically relies on working in the designated-verifier model, and it is not known how to prove adaptive soundness of the original Sahai-Waters construction.

This work. In this work, we give the first pure $i\mathcal{O}$ approach for constructing adaptively-sound SNARGs for NP. Specifically, we show that (sub-exponentially-secure) $i\mathcal{O}$ and (sub-exponentially-secure) one-way functions imply a SNARG for NP. As we elaborate in Section 1.1, we put forward a new realization of the "two-challenge" paradigm from [WW24a] that only relies on $i\mathcal{O}$ in combination with sub-exponentially-secure one-way functions. We summarize our main result in the following informal theorem:

Theorem 1.1 (Informal). *Let λ be a security parameter and $\mathcal{R}$ be any* NP *relation. Assuming the existence of sub-exponentially-secure iO for Boolean circuits and sub-exponentially-secure one-way functions, there exists a SNARG for $\mathcal{R}$ in the CRS model with the following properties:*

- **CRS size:** *The size of the CRS is* $\mathsf{poly}(\lambda, |\mathcal{R}|)$*, where* $|\mathcal{R}|$ *is the size of the Boolean circuit computing* $\mathcal{R}$.
- **Proof size:** *The size of the proof is* $\mathsf{poly}(\lambda)$.

Moreover, the SNARG satisfies perfect zero-knowledge.

1.1 Technical Overview

The starting point of this work is the adaptively-sound SNARG for NP by Waters and Wu [WW24a]. In their construction, the CRS contains two programs: the first is used to generate proofs while the second is used to generate challenges. At a high-level, [WW24a] takes the following "two-challenge" approach:

- Each NP statement x is associated with two (pseudorandom) challenges $z_{x,0}$ and $z_{x,1}$ for a one-way function f. Specifically, for each bit $b \in \{0, 1\}$, $z_{x,b} = f(\mathsf{F}(k_b, x))$ where F is a pseudorandom function (PRF) and f is a one-way function. The challenge-generation program takes a statement x and outputs the two associated challenges $(z_{x,0}, z_{x,1})$.
- A proof π for a statement x is a preimage to either $z_{x,0}$ or $z_{x,1}$. Specifically, we can write $\pi = (b, y)$ and π is valid if $f(y) = z_{x,b}$.
- The prover program takes as input a statement x and a witness w. If the witness is valid, then the prover program outputs the preimage $y_{x,b_x} = \mathsf{F}(k_{b_x}, x)$ of z_{x,b_x}, where $b_x \in \{0, 1\}$ is a pseudorandom bit derived from x. Specifically, the prover program computes $b_x = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$, where $\mathsf{F}_{\mathsf{sel}}$ is a PRF with 1-bit outputs. Importantly, for every choice of statement x, the prover program never outputs $y_{x,\bar{b}_x}$ where $\bar{b}_x = 1 - b_x$ is the complement of the bit associated with x.

More precisely, the proof-generation program and the challenge-generation program in the CRS are obfuscations of the following programs (where we hard-wire the circuit C for the associated NP relation, the PRF keys $k_{\mathsf{sel}}, k_0, k_1$, and the description of the one-way function f):

$\mathsf{GenProof}(x, w)$:	$\mathsf{GenChal}(x)$:
• On input the statement x and the witness w, if $C(x, w) = 1$, then compute $b_x = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and output $(b_x, \mathsf{F}(k_{b_x}, x))$. • Otherwise, output $\perp$.	• For $b \in \{0, 1\}$, compute $z_{x,b} = f(\mathsf{F}(k_b, x))$. • Output the pair of challenges $(z_{x,0}, z_{x,1})$.

The idea in the [WW24a] soundness analysis is that for a false statement x, the value b_x is computationally unpredictable. Thus, if an adversary outputs a proof $\pi = (b, y)$ for a false statement x and a bit $b \in \{0, 1\}$, with probability close to $1/2$, it will be the case that $b = \bar{b}_x$. More precisely, in [WW24a], they first move to an experiment where the adversary wins the (adaptive) soundness game only if it outputs a proof $\pi = (b, y)$ where

$$b \neq \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x) \quad \text{and} \quad f(y) = z_{x,b} \text{ where } (z_{x,0}, z_{x,1}) \leftarrow \mathsf{GenChal}(x). \tag{1.1}$$

Since the GenProof program *never* needs to output a preimage $y_{x,\bar{b}_x}$ for $z_{x,\bar{b}_x}$, the [WW24a] reduction replaces each challenge $z_{x,\bar{b}_x}$ output by the challenge-generation program with a challenge for which the reduction algorithm (and GenProof program) does *not* know the associated preimage. In the case of [WW24a], they consider a re-randomizable one-way function, which is a one-way function equipped with a statistical re-randomization algorithm. The re-randomization algorithm takes as input *any* challenge for the one-way function and outputs a fresh instance; moreover, given the re-randomization randomness together with a solution to the re-randomized instance, it is possible to recover a solution to the original instance. The [WW24a] reduction uses an exponential number of hybrids to replace every challenge $z_{x,\bar{b}_x}$ with a re-randomization of a fresh (but fixed) challenge z^*. Any preimage of $z_{x,\bar{b}_x}$ immediately yields a preimage of z^*. This would in turn break one-wayness. To construct a re-randomizable one-way function, the work of [WW24a] relies on algebraic assumptions: either the hardness of discrete log or the hardness of factoring. In this work, we introduce a new approach that does *not* need to rely on additional algebraic structure.

Bundling challenge-generation and proof verification. The first change we make is syntactic, but essential to realizing our new approach. In [WW24a, WZ24], the verification algorithm (on input a statement x and purported proof π) proceeds as follows:

- The algorithm first invokes the obfuscated challenge-verification program GenChal on the statement x to obtain two challenges $(0, z_{x,0})$ and $(1, z_{x,1})$.
- Then, it checks whether the provided preimage $\pi = (b, y)$ satisfies $f(y) = z_{x,b}$.

While [WW24a, WZ24] decouple the challenge-generation and the proof-verification processes, there is no need to do this. In this work, and as was done in the original construction of Sahai and Waters of a non-adaptively-sound SNARG [SW14], we publish a single obfuscated program that combines challenge-generation and proof verification. The CRS now contains obfuscations of the following programs:

GenProof(x, w):	VerProof(x, π):
• On input the statement x and the witness w, if $C(x, w) = 1$, then compute $b_x := \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and output $\big(b_x, \mathsf{F}(k_{b_x}, x)\big)$. • Otherwise, output $\perp$.	• On input the statement x and the proof $\pi = (b, y)$, output 1 if $y = \mathsf{F}(k_b, x)$ and 0 otherwise.

Since the verification logic is now embedded in the verification program VerProof itself, the program only needs to check whether $y = \mathsf{F}(k_b, x)$. In previous approaches [WW24a, WZ24], the GenChal program outputted one-way function challenges directly: $z_{x,0} = f(\mathsf{F}(k_0, x))$ and $z_{x,1} = f(\mathsf{F}(k_1, x))$. In these constructions, it is important that there is an efficient algorithm for sampling a challenge for the one-way function: namely, the challenge is sampled by first deriving a (pseudorandom) domain element $y_{x,b} = \mathsf{F}(k_b, x)$ and setting $z_{x,b} = f(y_{x,b})$. As we show later, it will be important in our construction that the real scheme does *not* need to sample a challenge for the one-way function. In fact, neither of the programs in the CRS need to compute the one-way function f. The one-way function f will only appear in the security proof itself.

Proving adaptive soundness. By the same analysis as in [WW24a], we first move to an experiment where the adversary wins if the adversary outputs a statement x and a proof $\pi = (b, y)$ where the following analog of Eq. (1.1) holds:

$$b \neq \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x) \quad \text{and} \quad y = \mathsf{F}(k_b, x). \tag{1.2}$$

Note that the condition $y = \mathsf{F}(k_b, x)$ is the same as the condition $\mathsf{VerProof}(x, \pi) = 1$. Formally, this relies on security of the selector PRF $\mathsf{F}_{\mathsf{sel}}$. Specifically, for a false statement x, the value of $\mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ is pseudorandom (and thus, computationally unpredictable) from the view of the adversary (it is never computed or output by $\mathsf{GenProof}$). An adversary that outputs a valid proof $\pi = (b, y)$ for a false statement x where $b = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ with probability far from $1/2$ would imply an adversary that can predict the value of $\mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$.

We now devise a sequence of hybrid experiments to embed a *fixed* string $y^* \in \{0,1\}^t$ (where t is the output length of F) into the verification check for *every* statement x. To do so, we start by rewriting the logic of the verification program in the following more convenient form:

$\mathsf{GenProof}(x, w)$:	$\mathsf{VerProof}_1(x, \pi)$:
• On input the statement x and the witness w, if $C(x, w) = 1$, then compute $b_x := \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and output $\big(b_x, \mathsf{F}(k_{b_x}, x)\big)$. • Otherwise, output $\bot$.	• On input the statement x and the proof $\pi = (b, y)$, compute $b_x = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$. Let $\bar{b}_x = 1 - b_x$. • If $b = b_x$, output 1 if $y = \mathsf{F}\big(k_{b_x}, x\big)$. • If $b = \bar{b}_x$, output 1 if $y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big) = 0^t$. • Otherwise, output 0.

By inspection, the modified $\mathsf{VerProof}_1$ program is functionally equivalent to the real $\mathsf{VerProof}$ program, so we can appeal to security of indistinguishability obfuscation to argue that the new CRS is computationally indistinguishable from the real CRS.

Planting a challenge. Since the $\mathsf{GenProof}$ program never evaluates $\mathsf{F}\big(k_{\bar{b}_x}, x\big)$ for any input x, we can appeal to (punctured) pseudorandomness[2] of $\mathsf{F}\big(k_{\bar{b}_x}, x\big)$ to argue that for *any fixed* string $y^* \in \{0,1\}^t$ (sampled independent of $k_{\bar{b}_x}$), the distribution of $\mathsf{F}\big(k_{\bar{b}_x}, x\big)$ is computationally indistinguishable from the distribution of $\mathsf{F}\big(k_{\bar{b}_x}, x\big) \oplus y^*$. In particular, for every input x, we can substitute the check $y \oplus \mathsf{F}\big(\bar{b}_x, x\big) = 0^t$ with the following (computationally indistinguishable) one:

$$y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big) \oplus y^* = 0^t \iff y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big) = y^*,$$

so long as $k_{\bar{b}_x}$ is sampled independently of y^*. Thus, using a hybrid argument where we step through each possible input x, we can show that the obfuscated programs $(\mathsf{GenProof}, \mathsf{VerProof}_1)$ in the CRS are computationally indistinguishable from the obfuscations of the following programs $(\mathsf{GenProof}, \mathsf{VerProof}_2)$:

$\mathsf{GenProof}(x, w)$:	$\mathsf{VerProof}_2(x, \pi)$:
• On input the statement x and the witness w, if $C(x, w) = 1$, then compute $b_x := \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and output $\big(b_x, \mathsf{F}(k_{b_x}, x)\big)$. • Otherwise, output $\bot$.	• On input the statement x and the proof $\pi = (b, y)$, compute $b_x = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$. Let $\bar{b}_x = 1 - b_x$. • If $b = b_x$, output 1 if $y = \mathsf{F}\big(k_{b_x}, x\big)$. • If $b = \bar{b}_x$, output 1 if $y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big) = y^*$. • Otherwise, output 0.

[2]In a puncturable PRF [BW13, KPTZ13, BGI14], the PRF key k can be punctured at a special point x^* to derive a punctured key $k^{(x^*)}$ with the property that for all $x \neq x^*$, $\mathsf{F}(k^{(x^*)}, x) = \mathsf{F}(k, x)$. The security requirement is that the value of $\mathsf{F}(k, x^*)$ is pseudorandom even given the punctured key $k^{(x^*)}$.

The programmed value y^* can be *any* value, as long as it is independent of the PRF keys k_0 and k_1. In this experiment, the adversary wins only if it outputs a statement x and a proof $\pi = (b, y)$ where $b \neq \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and $\mathsf{VerProof}_2(x, \pi) = 1$. By construction of $\mathsf{VerProof}_2$, this means that

$$y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big) = y^* \text{ where } \bar{b}_x = 1 - \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x).$$

The component y output by the adversary can be viewed as an "encryption" of the special string y^*, and moreover, given knowledge of k_0 and k_1, it is possible to recover y^* from any valid proof (*regardless* of the statement x).

Adaptive soundness via *injective* one-way functions. To complete the proof, we use the string y^* to embed a computational challenge. We begin with an approach using any *injective* one-way function f (with t-bit inputs). Since f is injective, it holds that

$$y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big) = y^* \iff f\big(y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big)\big) = f(y^*). \tag{1.3}$$

Then, by security of indistinguishability obfuscation, the programs in the CRS are computationally indistinguishable from obfuscations of the following programs:

<table>
<tr>
<td>

$\mathsf{GenProof}(x, w)$:

- On input the statement x and the witness w, if $C(x, w) = 1$, then compute $b_x := \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and output $\big(b_x, \mathsf{F}(k_{b_x}, x)\big)$.
- Otherwise, output $\bot$.

</td>
<td>

$\mathsf{VerProof}_3(x, \pi)$:

- On input the statement x and the proof $\pi = (b, y)$, compute $b_x = \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$. Let $\bar{b}_x = 1 - b_x$.
- If $b = b_x$, output 1 if $y = \mathsf{F}\big(k_{b_x}, x\big)$.
- If $b = \bar{b}_x$, output 1 if $f\big(y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big)\big) = f(y^*)$.
- Otherwise, output 0.

</td>
</tr>
</table>

First, observe that the description of GenProof and $\mathsf{VerProof}_3$ can be simulated given only the description of f and a one-way function challenge $f(y^*)$; the reduction algorithm would sample the PRF keys $k_{\mathsf{sel}}, k_0, k_1$ itself. Suppose the adversary outputs a statement x and a proof $\pi = (b, y)$ where $b \neq \mathsf{F}_{\mathsf{sel}}(k_{\mathsf{sel}}, x)$ and $\mathsf{VerProof}_3(x, \pi) = 1$. Then the value y it outputs must satisfy

$$f\big(y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big)\big) = f(y^*).$$

Using $k_{\bar{b}_x}$, the reduction algorithm would compute and output $y \oplus \mathsf{F}\big(k_{\bar{b}_x}, x\big)$ as its solution to the one-way function challenge. Observe that we have programmed the challenge string $f(y^*)$ into *every* verification check, so a successful proof of *any* statement implies a solution to the one-way function challenge. This gives an adaptively-secure SNARG for NP from a (sub-exponentially-secure) indistinguishability obfuscation scheme, a sub-exponentially-secure one-way function (to construct a puncturable PRF) and an *injective* one-way function.

Relaxing injectivity. Our construction above critically relies on injectivity of the one-way function (so Eq. (1.3) holds). If the one-way function was not injective, then we are not able to argue that the programs $\mathsf{VerProof}_2$ and $\mathsf{VerProof}_3$ are computationally indistinguishable by security of indistinguishability obfuscation. Injective one-way functions are significantly more structured than plain one-way functions, and standard constructions typically rely on algebraic assumptions such as discrete log or factoring [GLN11]. To obtain a construction from indistinguishability obfuscation and *unstructured* hardness assumptions (i.e., a pure $i\mathcal{O}$ approach), the goal would be to only rely on the existence of *plain* one-way functions.

To do so, we first rely on the fact that neither the prover program GenProof nor the verification program VerProof in the real scheme depends on the one-way function f.[3] The one-way function f only shows up in the security proof (specifically in the description of $\mathsf{VerProof}_3$), and critically, the only requirement we needed in the security proof was injectivity and one-wayness. Interestingly, we do *not* need the ability to *efficiently* sample a challenge for the injective one-way function. Observe that if the Setup algorithm which generates the CRS (or the Prove/Verify algorithms used to generate and verify proofs) needed to sample from the input space of the one-way function, then to have an efficient SNARG, we would additionally require the one-way function to support efficient sampling. However, since we only rely on the injective one-way function in the security proof, a construction with an *inefficient* sampling procedure would still suffice for the security reduction. Of course, this means that the intermediate reduction algorithms have to run in super-polynomial time or take in non-uniform advice (i.e., a sample from the challenge space of the injective one-way function). For ease of exposition, we take the latter approach in this work, but using super-polynomial-time reductions also suffices. Note that the cost of complexity leveraging (due to the use of super-polynomial-time security reductions) would only affect the size of the CRS and not the proofs. Thus, assuming sub-exponentially-secure $i\mathcal{O}$, a sub-exponentially secure one-way function, and an injective one-way function with an inefficient sampler, we obtain an adaptively-sound SNARG for NP. We give the formal construction and proof of adaptive soundness in Section 4.

Constructing injective one-way functions with an inefficient sampler. While we do not know how to construct injective one-way functions from any vanilla one-way function, it is straightforward if we allow the sampling algorithm to run in super-polynomial time. The idea is to compose with a hash function to reduce the number of preimages. Namely, let $f: \{0,1\}^t \to \{0,1\}^m$ be a one-way function. Suppose that a value $z \in \{0,1\}^m$ has k preimages under f. Let $h: \{0,1\}^t \to \{0,1\}^\rho$ be a hash function with output length $\rho = \log k$. Consider now the mapping $g(h, y) := (h, f(y), h(y))$. If h is a universal hash function, then with constant probability, we would expect that there is exactly one input (h, y) where $g(h, y) = (h, f(y), h(y))$. Having multiple such tuples (h, y) would mean there was a collision in the universal hash function h (when hashing k items to $O(k)$ buckets). To *guarantee* injectivity, the sampling procedure GenChal would (repeatedly) sample random challenges $(h, f(y), h(y))$ and only output a challenge when there is *exactly* one preimage under g. Checking that there is at most a single preimage requires super-polynomial time, which is why the resulting scheme has an inefficient sampler. To argue that this sampling procedure still produces instances that are hard to invert, we use the fact that each "sampling attempt" made by the sampling algorithm succeeds with inverse polynomial probability. This way, we can construct a reduction algorithm that takes a random one-way function challenge for f and outputs a sample from the sampling procedure GenChal with inverse polynomial probability. Since we are reducing to a *search* assumption (one-wayness), this suffices to establish the one-wayness of g. We give the formal definition, construction, and analysis in Section 3.

2 Preliminaries

Throughout this work, we write $\lambda \in \mathbb{N}$ to denote the security parameter. We write $\mathsf{poly}(\lambda)$ to denote a fixed polynomial in λ. We say a function $f(\lambda)$ is negligible in λ if $f(\lambda) = o(\lambda^{-c})$ for all constants $c \in \mathbb{N}$. We denote this with $f(\lambda) = \mathsf{negl}(\lambda)$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. For a finite set S, we write $x \xleftarrow{\mathrm{R}} S$ to denote that x is sampled uniformly

[3] Technically, they size of the obfuscated circuits need to be padded to be the size of the largest circuit used in the proof (which will ultimately include the circuit that evaluates the one-way function). However, the salient point is that the injective one-way function is not used in the actual construction itself, which gives us additional flexibility in our design.

at random from S. When $\mathcal{D}$ is a distribution (or a randomized algorithm), we write $x \leftarrow \mathcal{D}$ to denote that x is a draw from $\mathcal{D}$ (or the output of the randomized algorithm on a fresh choice of randomness). For a random variable X, we write $\mathbb{E}[X]$ to denote the expected value of X. We also recall Markov's inequality: if X is a non-negative random variable, then

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}.$$

Non-uniform algorithms. We model an *efficient* non-uniform algorithm $\mathcal{A}$ for inputs of length $n = n(\lambda)$ as a pair of algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ is a (possibly unbounded) algorithm that takes as input 1^λ and outputs an advice string $\mathsf{st}_\mathcal{A}$ of length $\mathsf{poly}(\lambda)$, and $\mathcal{A}_2$ is an *efficient* algorithm that takes as input the state $\mathsf{st}_\mathcal{A}$ and the input x. Specifically, for all $\lambda \in \mathbb{N}$ and all inputs $x \in \{0,1\}^{n(\lambda)}$, we define the output $\mathcal{A}(1^\lambda, x)$ to be $\mathcal{A}(1^\lambda, x) := \mathcal{A}_2(\mathcal{A}_1(1^\lambda), x)$. We often refer to $\mathcal{A}_1$ as the "preprocessing" algorithm and $\mathcal{A}_2$ as the "online" algorithm.

Sub-exponential hardness. Similar to [WW24a], our construction relies on sub-exponential hardness assumptions. We formulate some of our security definitions using (t, ε)-notation. We say 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 that it is sub-exponentially secure with parameter $c \in (0,1)$ if it is $\left(1, 2^{-\lambda^c}\right)$-secure. When extending the notion of (t, ε)-security to non-uniform algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we only require the online algorithm $\mathcal{A}_2$ to run in time $t(\lambda) \cdot \mathsf{poly}(\lambda)$; the preprocessing algorithm $\mathcal{A}_1$ that computes the advice string can still be unbounded.

Cryptographic primitives. We reuse many of the same primitives and notation from [WW24a]. Much of the text in this section is taken verbatim from [WW24a, §2].

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 $x \in \{0,1\}^{\ell_{\mathsf{in}}} \setminus \{x^*\}$,
$$\Pr\left[\mathsf{Eval}(k,x) = \mathsf{Eval}(k^{(x^*)}, x) : \begin{array}{c} k \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \\ k^{(x^*)} \leftarrow \mathsf{Puncture}(k, x^*) \end{array}\right] = 1.$$
- **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 challenge 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 $y^* \xleftarrow{\mathrm{R}} \{0,1\}^{\ell_{\mathsf{out}}}$ 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 $t(\lambda) \cdot \mathrm{poly}(\lambda)$, 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 $|\pi| \le p(\lambda + \log |C|)$.

3 Injective One-Way Functions with an Inefficient Sampler

In this section, we introduce the notion of an injective one-way function with an *inefficient* sampling algorithm. We then show how to construct such an object from *any* one-way function. This is the main additional cryptographic primitive we use to obtain our adaptively-sound SNARG from $i\mathcal{O}$ and one-way functions.

Definition 3.1 (One-Way Function)**.** Let $t = t(\lambda)$ and $m = m(\lambda)$ be polynomials. A function $f \colon \{0,1\}^{t(\lambda)} \to \{0,1\}^{m(\lambda)}$ is one-way if it is efficiently-computable and moreover, for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\Pr\left[f(y) = f(y^*) : \begin{array}{c} y^* \xleftarrow{\mathrm{R}} \{0,1\}^{t(\lambda)} \\ y \leftarrow \mathcal{A}(1^\lambda, f(y^*)) \end{array}\right] = \mathsf{negl}(\lambda).$$

Definition 3.2 (Injective One-Way Function with an Inefficient Sampler)**.** Let $t = t(\lambda)$ and $m = m(\lambda)$ be polynomials. An injective one-way function with an inefficient sampler with input length $t = t(\lambda)$ and output length $m = m(\lambda)$ is a pair of algorithms $\Pi_{\mathsf{OWF}} = (\mathsf{GenChal}, \mathsf{Verify})$ with the following properties:

- $\mathsf{GenChal}(1^\lambda) \to (z^*, y^*)$: On input a security parameter λ, the challenge-generation algorithm outputs a challenge $z^* \in \{0,1\}^{m(\lambda)}$ together with a solution $y^* \in \{0,1\}^{t(\lambda)}$. The $\mathsf{GenChal}$ algorithm is *not* required to be efficient.
- $\mathsf{Verify}(z, y) \to b$: On input a challenge z and a solution y, the verification algorithm outputs a bit $b \in \{0,1\}$. The verification algorithm must be efficient.

Moreover, the algorithms must satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$ and all (z^*, y^*) in the support of $\mathsf{GenChal}(1^\lambda)$, it holds that $\mathsf{Verify}(z^*, y^*) = 1$.
- **Injectivity:** For all security parameters $\lambda \in \mathbb{N}$, all (z^*, y^*) in the support of $\mathsf{GenChal}(1^\lambda)$, and all $y \neq y^*$, it holds that $\mathsf{Verify}(z^*, y) = 0$.
- **One-wayness:** For all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\Pr\left[\mathsf{Verify}(z^*, y) = 1 : \begin{array}{c} (z^*, y^*) \leftarrow \mathsf{GenChal}(1^\lambda) \\ y \leftarrow \mathcal{A}(1^\lambda, z^*) \end{array}\right] = \mathsf{negl}(\lambda).$$

Remark 3.3 (Explicit Evaluation Algorithm)**.** Strictly speaking, Definition 3.2 does not conform to the usual syntax of a one-way function in that we do *not* require an evaluation algorithm that takes an element $x \in \{0,1\}^t$ in the input space and produces an element $y \in \{0,1\}^m$ in the output space. In a standard (injective) one-way function $f : \{0,1\}^t \to \{0,1\}^m$, the challenge-generation algorithm $\mathsf{GenChal}$ algorithm would sample a random domain element $x \xleftarrow{\text{R}} \{0,1\}^t$ and output $(f(x), x)$ as the challenge. In our generalized syntax, we allow for an arbitrary (and inefficient) sampling algorithm, and moreover, omit the explicit requirement for an evaluation function. In the case of our specific construction (Construction 3.5), we note that it is straightforward to adapt it to have an explicit evaluation algorithm. Since this is unnecessary for our main application, we elect to use the simpler syntax in this paper.

Construction from one-way functions. We now show how to construct an injective one-way function with an inefficient sampler from *any* one-way function (and a universal hash function). We start by recalling the definition of a universal hash function and then give our construction.

Definition 3.4 (Universal Hash Function)**.** Let $\mathcal{H}$ be a family of hash functions $h : \mathcal{Y} \to \mathcal{Z}$ with domain $\mathcal{Y}$ and range $\mathcal{Z}$. We say that $\mathcal{H}$ is *universal* if for all $y_1, y_2 \in \mathcal{Y}$ where $y_1 \neq y_2$,
$$\Pr[h(y_1) = h(y_2) : h \xleftarrow{\text{R}} \mathcal{H}] \leq \frac{1}{|\mathcal{Z}|}.$$

We say that $\mathcal{H}$ is efficiently-sampleable if there exists an explicit algorithm that outputs a sample $h \xleftarrow{\text{R}} \mathcal{H}$ in time $\mathsf{poly}(\log|\mathcal{Y}| + \log|\mathcal{Z}|)$.

Construction overview. As noted in Section 1.1, we construct our injective one-way function with an inefficient sampler by composing a vanilla one-way function with a universal hash function. Specifically, suppose $f: \{0,1\}^t \to \{0,1\}^m$ is a one-way function. Each element $v \in \{0,1\}^m$ in the image of f can have between 1 and 2^t possible preimages. Thus, we need a way to associate a "unique" solution to a challenge element v. To do so, we additionally include a hash value σ with v, and we say that a candidate preimage $y \in \{0,1\}^t$ of v is valid only if $h(y) = \sigma$. In this case, the adversary's goal is not to find any preimage of v, but rather, to find a preimage that also has the correct hash value: that is, a value y where $(f(y), h(y)) = (v, \sigma)$.

The remaining question is how to pick the output length for the hash function h. If the output length is too short and a candidate value $v \in \{0,1\}^m$ has many preimages, then there can still be multiple preimages of v that share a hash value σ. Conversely, if the output length of the hash function is too long, then giving out the hash of a preimage $\sigma = h(y)$ might leak too many bits of information about the preimage y and compromise one-wayness of the function. In particular, the output length of the hash function should be dynamically adjusted based on the number of preimages the value v has (e.g., the output length of the hash function should scale with the number of preimages the element v has). In our construction, we handle this by having the challenge-generation algorithm "guess" the number of preimages v has, and we show that whenever it guesses correctly (up to a factor of 2), then the resulting challenge is hard to invert with noticeable probability. In more detail, our approach operates as follows:

- **Challenge structure:** The challenge is a tuple $z = (\rho, h, v, \sigma)$, where $\rho \in [t+1]$ is the output length of the hash function, $h: \{0,1\}^t \to \{0,1\}^\rho$ is a hash function sampled from a universal hash family, $v \in \{0,1\}^m$ is an element in the image of f, and $\sigma \in \{0,1\}^\rho$ is the target hash value.

- **Challenge sampling and injectivity:** The challenge-generation algorithm first samples the hash length $\rho \xleftarrow{R} [t+1]$. Then, it samples the hash function h from a universal hash family (with t-bit inputs and ρ-bit outputs). Finally, it samples a random $v \in \{0,1\}^m$ in the image of f (i.e., by sampling $u \xleftarrow{R} \{0,1\}^t$ and setting $v = f(u)$) and a random tag $\sigma \xleftarrow{R} \{0,1\}^\rho$. Now, the challenge-generation algorithm checks to see if there exists *exactly* one preimage y where $(f(y), h(y)) = (v, \sigma)$. If so, it outputs the challenge $z = (\rho, h, v, \sigma)$, and otherwise, it repeats this process. By construction, any challenge z output by this sampling procedure has exactly one preimage, so injectivity follows by construction. Note also that this sampling procedure is not efficiently-computable since it needs to count the number of preimages of v.

- **One-wayness:** To argue that it remains hard to invert the challenges z output by this procedure, we first show that with inverse polynomial probability δ, the GenChal algorithm will successfully sample a valid challenge $z = (\rho, h, v, \sigma)$ on the *first* attempt. In this case, we can set up a reduction to the one-wayness of f. Suppose there exists an efficient algorithm $\mathcal{A}$ that can solve the challenges output by GenChal with probability ε. Such an algorithm can be used to break one-wayness of f as follows. Given a (random) challenge $v \in \{0,1\}^m$ for f, the reduction algorithm samples the values of ρ, h, and σ itself (according to the same distribution as GenChal), and gives the challenge $z = (\rho, h, v, \sigma)$ to the adversary $\mathcal{A}$. With probability δ, this challenge is distributed according to the output of GenChal, so if $\mathcal{A}$ succeeds with probability ε, then our reduction algorithm succeeds in inverting f with probability $\delta\varepsilon$ and the claim follows.

We now give the formal construction and analysis:

Construction 3.5 (Injective One-Way Function with an Inefficient Sampler)**.** Let $t = t(\lambda)$ be a polynomially-bounded function and let $f: \{0,1\}^{t(\lambda)} \to \{0,1\}^{m(\lambda)}$ be a one-way function. For each $\rho \in [t+1]$, let $\mathcal{H}_\rho$

be an efficiently-sampleable family of (efficiently-computable) universal hash functions with domain $\{0,1\}^t$ and range $\{0,1\}^\rho$. We use f to construct an injective one-way function with an inefficient sampler $\Pi_{\mathsf{OWF}} = (\mathsf{GenChal}, \mathsf{Verify})$ with input length $t(\lambda) + 1$ as follows:

- $\mathsf{GenChal}(1^\lambda)$: On input the security parameter λ, set $t = t(\lambda)$. Then repeat the following procedure (up to) $\lambda \cdot (t+1)$ times:
  - Sample $\rho \xleftarrow{\mathrm{R}} [t+1]$ and $h \xleftarrow{\mathrm{R}} \mathcal{H}_\rho$. Sample $u \xleftarrow{\mathrm{R}} \{0,1\}^t$ and let $v = f(u) \in \{0,1\}^m$.
  - Sample $\sigma \xleftarrow{\mathrm{R}} \{0,1\}^\rho$. If there exists $\hat{y}^* \in \{0,1\}^t$ such that $f(\hat{y}^*) = v$ and $h(\hat{y}^*) = \sigma$, and moreover, for all $\hat{y} \neq \hat{y}^* \in \{0,1\}^t$, it holds that $(f(\hat{y}), h(\hat{y})) \neq (f(\hat{y}^*), h(\hat{y}^*))$, then output the challenge $z^* = (\rho, h, f(\hat{y}^*), h(\hat{y}^*))$ together with the solution $y^* = 0 \| \hat{y}^* \in \{0,1\}^{t+1}$.

  If after $\lambda \cdot (t+1)$ attempts, the above algorithm has not produced any output, then output $z^* = \bot$ and the associated solution $y^* = 1^{t+1}$.

- $\mathsf{Verify}(z, y)$: On input the challenge z and a solution y, the verification algorithm proceeds as follows:
  - If $z = \bot$, then output 1 if $y = 1^{t+1}$ and 0 otherwise.
  - If $z = (\rho, h, v, \sigma)$ for some $\rho \in [t]$, $h \in \mathcal{H}_\rho$, $v \in \{0,1\}^m$, and $\sigma \in \{0,1\}^\rho$, then parse $y = b \| \hat{y}$ where $b \in \{0,1\}$ and $\hat{y} \in \{0,1\}^t$. Output 1 if $b = 0$ and $(f(\hat{y}), h(\hat{y})) = (v, \sigma)$ and 0 otherwise.

  In all other cases, output 0.

Theorem 3.6 (Correctness and Injectivity)**.** *Construction 3.5 is correct and injective.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$ and any (z^*, y^*) in the support of $\mathsf{GenChal}(1^\lambda)$. We consider two possibilities:

- Suppose $z^* = \bot$. In this case, $y^* = 1^{t+1}$. By construction, $\mathsf{Verify}(z^*, y^*) = 1$, and moreover, $\mathsf{Verify}(z^*, y) = 0$ for all $y \neq 1^{t+1}$.

- Suppose $z^* = (\rho, h, v, \sigma)$ and $y^* \in \{0,1\}^{t+1}$. By construction of GenChal, it must then be the case that $y^* = 0 \| \hat{y}^*$ for some $\hat{y}^* \in \{0,1\}^t$ and $(v, \sigma) = (f(\hat{y}^*), h(\hat{y}^*))$. As such, $\mathsf{Verify}(z^*, y^*) = 1$. Moreover, the GenChal algorithm outputs (z^*, y^*) only if for all $\hat{y} \neq \hat{y}^*$, it holds that $(f(\hat{y}), h(\hat{y})) \neq (f(\hat{y}^*), h(\hat{y}^*)) = (v, \sigma)$. Correspondingly, for all $y \neq 0 \| \hat{y}^*$, this means that $\mathsf{Verify}(z^*, y) = 0$. □

Theorem 3.7 (One-Wayness)**.** *If for all $\rho \in [t]$, $\mathcal{H}_\rho$ is universal and if f is one-way, then Construction 3.5 is also one-way.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the one-wayness game. We now define a sequence of hybrid experiments between the adversary $\mathcal{A}$ and the challenger:

- Hyb_0: This is the real one-wayness game. Namely, the challenger starts by sampling the challenge z^* according to the specification of $\mathsf{GenChal}(1^\lambda)$:
  - The challenger repeats the following sampling procedure until it either successfully samples a challenge-solution pair (z^*, y^*) or it fails a total of $\lambda(t+1)$ times: sample $\rho \xleftarrow{\mathrm{R}} [t+1]$, $h \xleftarrow{\mathrm{R}} \mathcal{H}_\rho$, $u \xleftarrow{\mathrm{R}} \{0,1\}^t$, $\sigma \xleftarrow{\mathrm{R}} \{0,1\}^\rho$, and set $v = f(u)$. If there exists $\hat{y}^* \in \{0,1\}^t$ such that $f(\hat{y}^*) = v$ and $h(\hat{y}^*) = \sigma$ and for all $\hat{y} \neq \hat{y}^*$, it holds that $(f(\hat{y}), h(\hat{y})) \neq (f(\hat{y}^*), h(\hat{y}^*))$, then set $z^* = (\rho, h, f(\hat{y}^*), h(\hat{y}^*)) = (\rho, h, f(u), \sigma)$.

– If the sampling procedure does not terminate after $\lambda(t+1)$ attempts, the challenger sets $z^* = \perp$.

The challenger gives the challenge z^* to the adversary $\mathcal{A}$. Algorithm $\mathcal{A}$ replies with y. The output of the experiment is $\mathsf{Verify}(z^*, y)$.

- Hyb_1: Same as Hyb_0, except the challenger first defines the following sets:
  – For each $\rho \in [t+1]$, define the set S_ρ to be
  $$S_\rho = \left\{(h, u, \sigma) : h \in \mathcal{H}_\rho, u \in \{0,1\}^t, \sigma \in \{0,1\}^\rho\right\}.$$
  – For each $\rho \in [t+1]$, define the set $T_\rho \subseteq S_\rho$ to be the subset of tuples (h, u, σ) where there exists $\hat{y}^* \in \{0,1\}^t$ such that $f(\hat{y}^*) = f(u)$ and $h(\hat{y}^*) = \sigma$ and for all $\hat{y} \neq \hat{y}^*$, it holds that $(f(\hat{y}), h(\hat{y})) \neq (f(\hat{y}^*), h(\hat{y}^*))$.

  Then, the challenger repeats the following sampling procedure until it successfully samples a challenge z^* or it fails a total of $\lambda(t+1)$ times: sample $\rho \xleftarrow{R} [t+1]$ and $(h, u, \sigma) \xleftarrow{R} S_\rho$. If $(h, u, \sigma) \in T_\rho$, set $z^* = (\rho, h, f(u), \sigma)$. If the sampling procedure does not succeed after $\lambda(t+1)$ attempts, the challenger sets $z^* = \perp$. The challenger gives z^* to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs y and the output of the experiment is $\mathsf{Verify}(z^*, y)$.

- Hyb_2: Same as Hyb_1, except the challenger continues to sample $\rho \xleftarrow{R} [t+1]$ and $(h, u, \sigma) \xleftarrow{R} S_\rho$ until $(h, u, \sigma) \in T_\rho$ (in which case it sets $z^* = (\rho, h, f(u), \sigma)$ as in Hyb_1). If it is the case that $T_\rho = \varnothing$ for all $\rho \in [t+1]$, then the experiment always outputs 0.

- Hyb_3: Same as Hyb_2, except the challenger now samples $\rho \xleftarrow{R} [t+1]$, $(h, u, \sigma) \xleftarrow{R} S_\rho$, and sets $z^* = (\rho, h, f(u), \sigma)$. In particular, the challenger no longer checks for membership in T_ρ.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output distribution of an execution of Hyb_i with adversary $\mathcal{A}$. We now show that each pair of adjacent hybrid experiments are indistinguishable. We start by proving the following claim about the sets S_ρ and T_ρ defined in Hyb_1, which will be useful for analyzing the output distributions of the hybrid experiments.

Claim 3.8. *Let S_ρ, T_ρ be the sets defined in the specification of Hyb_1, where $\rho \in [t+1]$. If $\mathcal{H}_\rho$ is universal for all $\rho \in [t+1]$, then*
$$\Pr[(h, u, \sigma) \in T_\rho : \rho \xleftarrow{R} [t+1], (h, u, \sigma) \xleftarrow{R} S_\rho] \geq \frac{1}{224(t+1)}.$$

Proof. Let $\mathcal{D}$ be the distribution over tuples (ρ, h, u, σ) where $\rho \xleftarrow{R} [t+1]$ and $(h, u, \sigma) \xleftarrow{R} S_\rho$. Equivalently, $\mathcal{D}$ samples $\rho \xleftarrow{R} [t+1]$, $h \xleftarrow{R} \mathcal{H}_\rho$, $u \xleftarrow{R} \{0,1\}^t$, and $\sigma \xleftarrow{R} \{0,1\}^\rho$. For a particular tuple (ρ, h, u, σ), we now define the following events:

- Let k_u be the number of pre-images of $f(u)$, and label these preimages $u_1, \ldots, u_{k_u} \in \{0,1\}^t$. Namely, for all $i \in [k_u]$, $f(u_i) = f(u)$. Let E_1 be the event that $2^{\rho-1} \leq 2k_u \leq 2^\rho$.
- For each $i \in [k_u]$, let N_i be the number of indices $j \in [k_u]$ where $h(u_j) = h(u_i)$. We will say that u_i is "good" if $N_i = 1$ and that it is "bad" otherwise. Let E_2 be the event that there are at least $k_u/8$ indices $i \in [k_u]$ where u_i is good.

Now we can write

$$\begin{aligned}\Pr[(h,u,\sigma)\in T_\rho] &\geq \Pr[(h,u,\sigma)\in T_\rho \wedge \mathsf{E}_1 \wedge \mathsf{E}_2] \\ &= \Pr[(h,u,\sigma)\in T_\rho \mid \mathsf{E}_1 \wedge \mathsf{E}_2]\cdot \Pr[\mathsf{E}_2 \mid \mathsf{E}_1]\cdot \Pr[\mathsf{E}_1],\end{aligned} \tag{3.1}$$

where all probabilities are taken over the choice of $(\rho,h,u,\sigma) \leftarrow \mathcal{D}$. We now analyze each of the probabilities:

- Consider event E_1. Take any $u \in \{0,1\}^t$ and let k_u be the number of preimages of $f(u)$. By definition, $1 \leq k_u \leq 2^t$. Thus, there exists some $\ell_u \in [t+1]$ such that $2^{\ell_u - 1} \leq 2k_u \leq 2^{\ell_u}$. Correspondingly,

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[\mathsf{E}_1] = \Pr[\rho = \ell_u : \rho \xleftarrow{\mathrm{R}} [t+1], u \xleftarrow{\mathrm{R}} \{0,1\}^t] = \frac{1}{t+1}. \tag{3.2}$$

- Suppose E_1 occurs. Consider now the conditional probability that E_2 occurs. For a tuple (ρ,h,u,σ), let $u_1,\ldots,u_{k_u} \in \{0,1\}^t$ be the preimages of $f(u)$. Then, for all $i,j \in [k_u]$, define the indicator random variable $b_{i,j}$ for the event $h(u_j) = h(u_i)$. Since $\mathcal{H}_\rho$ is universal,

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[b_{i,j} = 1 \mid \mathsf{E}_1] = \Pr_{h \xleftarrow{\mathrm{R}} \mathcal{H}_\rho}[h(u_i) = h(u_j)] \leq \begin{cases} 1 & i = j \\ 1/2^\rho & i \neq j \end{cases}.$$

  This means that

$$\mathop{\mathbb{E}}_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[b_{i,j} \mid \mathsf{E}_1] \leq \begin{cases} 1 & i = j \\ 1/2^\rho & i \neq j \end{cases}.$$

  By definition, $N_i = \sum_{j\in[k_u]} b_{i,j}$, so we conclude that for all $i \in [k_u]$,

$$\mathop{\mathbb{E}}_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[N_i \mid \mathsf{E}_1] = 1 + \frac{k_u - 1}{2^\rho}.$$

  By Markov's inequality,

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[N_i \geq 2 \mid \mathsf{E}_1] \leq \frac{1}{2} + \frac{k_u - 1}{2^{\rho+1}}.$$

  Finally, if E_1 occurs, $k_u \leq 2^{\rho-1}$ so $k_u/2^{\rho+1} \leq 1/4$. We conclude that for each $i \in [k_u]$,

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[N_i \geq 2 \mid \mathsf{E}_1] \leq \frac{3}{4}. \tag{3.3}$$

  Let M be the number of indices $i \in [k_u]$ where u_i is bad (i.e., where $N_i \geq 2$). Let b_i' be the indicator random variable for the event that u_i is bad. From Eq. (3.3), we have that $\mathbb{E}_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[b_i' \mid \mathsf{E}_1] \leq 3/4$. Since $M = \sum_{i\in[k_u]} b_i'$, we correspondingly have that $\mathbb{E}_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[M \mid \mathsf{E}_1] \leq 3k_u/4$. Again by Markov's inequality,

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}\left[M \geq \frac{7k_u}{8} \;\middle|\; \mathsf{E}_1\right] \leq \frac{3k_u/4}{7k_u/8} = \frac{6}{7}.$$

  Event E_2 corresponds to the case where $M < 7k_u/8$.

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[\mathsf{E}_2 \mid \mathsf{E}_1] = 1 - \Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}\left[M \geq \frac{7k_u}{8} \;\middle|\; \mathsf{E}_1\right] \geq \frac{1}{7}. \tag{3.4}$$

- Suppose events E_1 and E_2 occur. We now consider the probability that $(h, u, \sigma) \in T_\rho$. Since E_2 occurs, at least $k_u/8$ of the indices $i \in [k_u]$ are good. This means there exists a set $\Sigma_u \subseteq \{0,1\}^\rho$ of size at least $|\Sigma_u| \geq k_u/8$ such that for all $\sigma \in \Sigma_u$, there exists $i \in [k_u]$ such that $h(u_i) = \sigma$ and for all $j \neq i$, $h(u_j) \neq \sigma$. Notably, this means that for all $\hat{u} \neq u_i$, either $f(\hat{u}) \neq f(u_i)$ or $h(\hat{u}) \neq h(u_i)$. Equivalently, $(h, u_i, \sigma) \in T_\rho$ for all $\sigma \in \Sigma_u$. Thus, we can now write

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[(h,u,\sigma) \in T_\rho \mid \mathsf{E}_2 \wedge \mathsf{E}_1] = \Pr_{\sigma \xleftarrow{\mathrm{R}} \{0,1\}^\rho}[\sigma \in \Sigma_u] = \frac{|\Sigma_u|}{2^\rho} \geq \frac{k_u/8}{2^\rho}.$$

Conditioned on E_1, we have that $2^{\rho-1} \leq 2k_u$ so $2^\rho \leq 4k_u$, so we conclude that

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[(h,u,\sigma) \in T_\rho \mid \mathsf{E}_2 \wedge \mathsf{E}_1] \geq \frac{k_u/8}{2^\rho} \geq \frac{1}{32}. \tag{3.5}$$

Combining Eqs. (3.2), (3.4) and (3.5) with Eq. (3.1), we conclude that

$$\Pr_{(\rho,h,u,\sigma)\leftarrow\mathcal{D}}[(h,u,\sigma) \in T_\rho] \geq \frac{1}{7 \cdot 32 \cdot (t+1)} = \frac{1}{224(t+1)}. \qquad \square$$

Lemma 3.9. *It holds that* $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

Proof. The only difference between these two experiments is syntactic. Namely, in both cases, the challenger samples $\rho \xleftarrow{\mathrm{R}} [t+1]$, $h \xleftarrow{\mathrm{R}} \mathcal{H}_\rho$, $u \xleftarrow{\mathrm{R}} \{0,1\}^t$, and $\sigma \xleftarrow{\mathrm{R}} \{0,1\}^\rho$. Checking that $(h,u,\sigma) \in T_\rho$ in Hyb_1 is identical to the check the challenger performs in Hyb_0. $\square$

Lemma 3.10. *If $\mathcal{H}_\rho$ is universal for all $\rho \in [t+1]$, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \leq \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] + \mathsf{negl}(\lambda)$.

Proof. The only difference between Hyb_1 and Hyb_2 is the challenger sets $z^* = \perp$ if the sampling procedure fails after $\lambda(t+1)$ attempts whereas the challenger in Hyb_2 tries indefinitely until it is successful. Thus, the adversary's view in these two experiments is *identical* unless the challenger in Hyb_1 is unsuccessful in sampling a challenge z^* after $\lambda(t+1)$ iterations. By Claim 3.8, each sampling attempt is successful with probability at least $1/(224(t+1))$. Since the samples are drawn independently, the challenger in Hyb_1 sets $z^* = \perp$ with probability at most

$$\Pr[z^* = \perp \text{ in } \mathsf{Hyb}_1] \leq \left(1 - \frac{1}{224(t+1)}\right)^{\lambda(t+1)} \leq \exp\left(-\frac{\lambda(t+1)}{224(t+1)}\right) = e^{-\Omega(\lambda)} = \mathsf{negl}(\lambda),$$

where we are using the fact that for all real-valued x, it holds that $1 + x \leq e^x$. Thus, with probability $1 - \mathsf{negl}(\lambda)$, the challenger in Hyb_1 will successfully sample a challenge z^* in the first $\lambda(t+1)$ iterations. In this case, the adversary's view is identical in the two experiments. $\square$

Lemma 3.11. *If $\mathcal{H}_\rho$ is universal for all $\rho \in [t+1]$, then* $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \leq 224(t+1) \cdot \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

Proof. Let $\rho \xleftarrow{\mathrm{R}} [t+1]$ and $(h,u,\sigma) \xleftarrow{\mathrm{R}} S_\rho$. Let event E be the event that $(h,u,\sigma) \in T_\rho$. Then,

$$\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1 \wedge \mathsf{E}] = \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1 \mid \mathsf{E}] \cdot \Pr[\mathsf{E}]. \tag{3.6}$$

From Claim 3.8, we have that $\Pr[\mathsf{E}] \geq \frac{1}{224(t+1)}$. Moreover, conditioned on $(h,u,\sigma) \in T_\rho$, the challenge $z^* = (\rho, h, f(u), \sigma)$ in Hyb_3 is distributed exactly according to the distribution in Hyb_2. Thus, we conclude that

$$\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1 \mid \mathsf{E}] \geq \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1].$$

The claim now follows from Eq. (3.6). $\square$

Lemma 3.12. *If f is one-way, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ such that $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \geq \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks one-wayness of f:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $v \in \{0,1\}^m$.
2. Algorithm $\mathcal{B}$ samples $\rho \xleftarrow{\text{R}} [t+1]$, $h \xleftarrow{\text{R}} \mathcal{H}_\rho$, and $\sigma \xleftarrow{\text{R}} \{0,1\}^\rho$. It gives $z^* = (\rho, h, v, \sigma)$ to $\mathcal{A}$.
3. If algorithm $\mathcal{A}$ outputs a preimage $y \in \{0,1\}^t$, then $\mathcal{B}$ also outputs y.

By definition, the one-wayness challenger samples $u \xleftarrow{\text{R}} \{0,1\}^t$ and sets $v = f(u)$. Thus, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_3 for $\mathcal{A}$. Thus, with probability ε, algorithm $\mathcal{A}$ outputs y such that $\mathsf{Verify}(z^*, y) = 1$. This means that $f(y) = v$, in which case, algorithm $\mathcal{B}$ successfully recovers a preimage of v for f. Thus, algorithm $\mathcal{B}$ succeeds with the same advantage ε. $\square$

By Lemmas 3.9 to 3.11, we have that

$$\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \leq 224(t+1) \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] + \mathsf{negl}(\lambda).$$

By Lemma 3.12, $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$. Since $t = t(\lambda)$ is polynomially-bounded, the theorem follows. $\square$

4 Adaptively-Sound SNARGs for NP from $i\mathcal{O}$ and One-Way Functions

In this section, we show how to construct an adaptively-sound SNARG from indistinguishability obfuscation together with an injective one-way function with an inefficient sampler (Definition 3.2). Our construction closely follows the two-challenge paradigm from [WW24a, WZ24]. A key difference between our construction and the previous constructions is we move the entirety of the verification logic *into* the obfuscated program itself. In the previous constructions, verification consists of first running an obfuscated program to derive a challenge (for a one-way function) and then checking whether the proof contains a valid preimage to the challenge. In our construction, the obfuscated program checks the proof. This difference will enable a different proof strategy for arguing adaptive soundness that allows us to only rely on one-way functions.

Notation. In our construction, we will associate a bit-string $x, y \in \{0,1\}^n$ of length n with the binary representation of an integer between 0 and $2^n - 1$, and we will write "$x \leq y$" to refer to the comparison of the *integer* representations of x and y.

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

- Let $i\mathcal{O}$ be an indistinguishability obfuscator for Boolean circuits (Definition 2.1).
- Let $\Pi_{\mathsf{PPRF}} = (\mathsf{F.KeyGen}, \mathsf{F.Eval}, \mathsf{F.Puncture})$ be a puncturable PRF (Definition 2.2). For a key k and an input x, we will write $\mathsf{F}(k, x)$ to denote $\mathsf{F.Eval}(k, x)$.
- Let $\Pi_{\mathsf{OWF}} = (\mathsf{OWF.GenChal}, \mathsf{OWF.Verify})$ be an injective one-way function with an inefficient sampler (Definition 3.2). Let $t = t(\lambda)$ be the input length of Π_{OWF}. Note that our construction will not make use of Π_{OWF} (it is only used in the proof of Theorem 4.3). However, the scheme will depend on

the input length t of Π_{OWF} as well as the size of the circuit that computes OWF.Verify. Specifically, the size of the verification program VerProof in the following construction will be padded to be at least as large as a program that computes OWF.Verify.

Our construction will leverage sub-exponential hardness of $i\mathcal{O}$ and the puncturable PRF Π_{PPRF}. In the following, let $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$ and $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ 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} and λ_{PRF} 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:
  - Sample a "selector" PRF key $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$.
  - Let $t = t(\lambda)$ be the input length for Π_{OWF} and sample PRF keys $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$.
  - Define the following programs GenProof and VerProof:

    Input: statement $x \in \{0,1\}^n$ and witness $w \in \{0,1\}^h$
    Hard-coded: Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and 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 $C(x, w) = 1$, compute $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and output $(b, \mathsf{F}(k_b, x))$.

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

    Input: statement $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{t+1}$
    Hard-coded: Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and PRF keys k_0, k_1

    On input a statement $x \in \{0,1\}^n$ and a proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$,

    * Output 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise.

    Figure 2: The verification program $\mathsf{VerProof}[C, k_0, k_1]$.

    Let $s = s(\lambda, n, |C|)$ be the maximum size of the GenProof and VerProof programs as well as those appearing in the proof of Theorem 4.3 (specifically, the programs in Figs. 3 to 5 and 6). By construction, we note that $s = \mathrm{poly}(\lambda, |C|)$ is polynomially-bounded.
  - Construct the obfuscated programs $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}[C, k_0, k_1])$. Output the common reference string $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$.

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

- $\mathsf{Verify}(\mathsf{crs}, x, \pi)$: On input the common reference string $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$, the statement $x \in \{0,1\}^n$, and the proof $\pi \in \{0,1\}^{t+1}$, the verification algorithm outputs $\mathsf{ObfVerify}(x, \pi)$.

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

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

- By construction, $\mathsf{ObfProve}$ is an obfuscation of the program $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1]$, where $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$. In this case π is obtained by evaluating $\mathsf{ObfProve}$ on input (x, w). By correctness of $i\mathcal{O}$ and definition of $\mathsf{GenProof}$, this means that $\pi = (b, y)$ where $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and $y = \mathsf{F}(k_b, x)$.
- By construction $\mathsf{ObfVerify}$ is an obfuscation of the program $\mathsf{VerProof}[C, k_0, k_1]$. The verification program computes $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and checks whether $y = \mathsf{F}(k_b, x)$. Both checks hold by construction, so by correctness of $i\mathcal{O}$, the verification algorithm accepts. □

Theorem 4.3 (Adaptive Soundness)**.** *Suppose the following conditions hold:*

1. *$i\mathcal{O}$ is correct and satisfies sub-exponential security with parameter $\varepsilon_{\mathsf{obf}} \in (0,1)$ against non-uniform adversaries;*[4]
2. *Π_{PPRF} satisfies punctured correctness and selective sub-subexponential punctured security with parameter $\varepsilon_{\mathsf{PRF}} \in (0,1)$ against non-uniform adversaries;*
3. *Π_{OWF} is an injective one-way function with inefficient sampler with polynomial security against non-uniform adversaries.*

Moreover, suppose $\lambda_{\mathsf{obf}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\lambda_{\mathsf{PRF}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then, Construction 4.1 is adaptively sound against non-uniform adversaries.

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a non-uniform adversary for the adaptive soundness game for Construction 4.1 that succeeds with (non-negligible) advantage $\varepsilon = \varepsilon(\lambda)$. Without loss of generality, we assume that for every security parameter $\lambda \in \mathbb{N}$, algorithm $\mathcal{A}$ always outputs a Boolean circuit C with statements of a *fixed* length $n = n(\lambda)$; we refer to [WW24a, Theorem 4.3] for a formal argument. We now define a sequence of hybrid experiments. The initial sequence is nearly identical to those from [WW24a] with the main distinction being the specification of Hyb_3.

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

  The challenger gives $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ then outputs a statement x and a proof π. The output is 1 if

  $$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{ObfVerify}(x, \pi) = 1.$$

[4] Recall from Section 2 that we say a primitive is sub-exponential secure with parameter $\varepsilon \in (0,1)$ against non-uniform adversaries if for every non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_2$ run in time at most $\mathrm{poly}(\lambda)$, and all sufficiently-large $\lambda \in \mathbb{N}$, the advantage of $\mathcal{A}$ is at most $2^{-\lambda^\varepsilon}$.

- Hyb_1: Same as Hyb_0 except at the end of the experiment, after the adversary outputs the proof $\pi = (b, y) \in \{0,1\}^{t+1}$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, the output of the experiment is 1 if the following hold:

$$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{ObfVerify}(x, \pi) = 1 \quad \text{and} \quad b \neq \mathsf{F}(k_{\mathsf{sel}}, x).$$

- 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

$$\mathsf{ObfVerify}(x, \pi) = 1 \quad \text{and} \quad b \neq \mathsf{F}(k_{\mathsf{sel}}, x).$$

- Hyb_3: Same as Hyb_2, except the challenger changes how it constructs $\mathsf{ObfVerify}$. During setup, the challenger now does the following:
  - At the beginning of the experiment, the challenger samples $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$. Then the challenger defines the following program $\mathsf{VerProof}_1$:

Input: statement $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{t+1}$
Hard-coded: Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, puncturable PRF keys $k_{\mathsf{sel}}, k_0, k_1$, and instance $z^* \in \{0,1\}^m$

On input a statement $x \in \{0,1\}^n$ and a proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$:

* If $b = \mathsf{F}(k_{\mathsf{sel}}, x)$, output 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise.
* If $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$, output $\mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_b, x))$.

Figure 3: The verification program $\mathsf{VerProof}_1[C, k_{\mathsf{sel}}, k_0, k_1, z^*]$.

The challenger sets $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_1[C, k_{\mathsf{sel}}, k_0, k_1, z^*])$ 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 sub-exponentially-secure with parameter $\varepsilon_{\mathsf{obf}} \in (0,1)$ against non-uniform adversaries. and Π_{PPRF} satisfies selective sub-exponential punctured security with parameter $\varepsilon_{\mathsf{PRF}} \in (0,1)$ against non-uniform adversaries. Suppose $\lambda_{\mathsf{obf}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\lambda_{\mathsf{PRF}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. Finally, suppose Π_{PPRF} satisfies punctured correctness. Then,*

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

Proof. The proof is analogous to the proof of Lemma 4.4 in [WW24a]. For completeness, we include the proof in Appendix A.1. □

Lemma 4.5. *It holds that* $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \le \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

Proof. The conditions for outputting 1 in Hyb_2 are a subset of those in Hyb_1. Thus, whenever the challenger outputs 1 in Hyb_1, it also does so in Hyb_2, and the lemma follows. □

Lemma 4.6. *Suppose $i\mathcal{O}$ is sub-exponentially-secure with parameter $\varepsilon_{\text{obf}} \in (0,1)$ against non-uniform adversaries and Π_{PPRF} satisfies selective sub-exponential punctured security with parameter $\varepsilon_{\text{PRF}} \in (0,1)$ against non-uniform adversaries. Suppose $\lambda_{\text{obf}}(\lambda, n) = (\lambda+n)^{1/\varepsilon_{\text{obf}}}$ and $\lambda_{\text{PRF}}(\lambda, n) = (\lambda+n)^{1/\varepsilon_{\text{PRF}}}$. Finally, suppose Π_{PPRF} satisfies punctured correctness and Π_{OWF} satisfies injectivity. Then,*

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

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

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

  Input: statement $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{t+1}$
  Hard-coded: Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, puncturable PRF keys k_{sel}, k_0, k_1, an instance $z^* \in \{0,1\}^m$, and an index $i \in \{0,1\}^n$

  On input a statement $x \in \{0,1\}^n$ and a proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$:

  – If $b = \mathsf{F}(k_{\text{sel}}, x)$, output 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise.

  – If $b = 1 - \mathsf{F}(k_{\text{sel}}, x)$, then proceed as follows:

    * If $x < i$, output $\mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_b, x))$.
    * If $x \ge i$, output 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise.

  Figure 4: The verification program $\mathsf{VerProof}_2[C, k_{\text{sel}}, k_0, k_1, z^*, i]$.

  Then, the challenger proceeds as follows:

  – Sample PRF keys $k_{\text{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\text{PRF}}}, 1^n, 1^1)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\text{PRF}}}, 1^n, 1^t)$.
  – Sample $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$.
  – Construct the programs $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\text{obf}}}, 1^s, \mathsf{GenProof}[C, k_{\text{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\text{obf}}}, 1^s, \mathsf{VerProof}_2[C, k_{\text{sel}}, k_0, k_1, z^*, i])$ where $\mathsf{GenProof}$ and $\mathsf{VerProof}_2$ are the programs from Figs. 1 and 4 and s is the bound on the program size from Construction 4.1.

  The challenger gives $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$. After $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, the challenger outputs 1 if $\mathsf{ObfVerify}(x, \pi) = 1$ and $b \ne \mathsf{F}(k_{\text{sel}}, x)$.

- $\mathsf{Hyb}_{2,i}^{(1)}$: Same as $\mathsf{Hyb}_{2,i}^{(0)}$, except the challenger defines the following program $\mathsf{VerProof}_3$:

Input: statement $x \in \{0,1\}^n$ and proof $\pi \in \{0,1\}^{t+1}$
Hard-coded: Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, puncturable PRF keys $k_{\mathsf{sel}}, k_0, k_1$, values $r^*, y^* \in \{0,1\}^t$, $z^* \in \{0,1\}^m$, and an index $i \in \{0,1\}^n$

On input a statement $x \in \{0,1\}^n$ and a proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$:

- If $b = \mathsf{F}(k_{\mathsf{sel}}, x)$, output 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise.
- If $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$, then proceed as follows:
  - If $x < i$, output $\mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_b, x))$.
  - If $x = i$, output 1 if $y \oplus r^* = y^*$.
  - If $x > i$, output 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise.

Figure 5: The verification program $\mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_0, k_1, r^*, y^*, z^*, i]$.

The challenger now computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$ and $r^* = y^* \oplus \mathsf{F}(k_{b^*}, i)$. It constructs the verification program as $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_0, k_1, r^*, y^*, z^*, i])$. The remainder of the experiment proceeds as in $\mathsf{Hyb}_{2,i}^{(0)}$.

- $\mathsf{Hyb}_{2,i}^{(2)}$: Same as $\mathsf{Hyb}_{2,i}^{(1)}$ except the challenger punctures k_{b^*} at index i. Specifically, the challenger computes $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$. It still sets $r^* = y^* \oplus \mathsf{F}(k_{b^*}, i)$. Then, it uses the punctured key $k_{b^*}^{(i)}$ in place of k_{b^*} in $\mathsf{ObfProve}$ and $\mathsf{ObfVerify}$. Specifically, $\mathsf{ObfProve}$ and $\mathsf{ObfVerify}$ are now defined as follows:
  - If $b^* = 0$, then the challenger sets $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0^{(i)}, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_0^{(i)}, k_1, r^*, y^*, z^*, i])$.
  - If $b^* = 1$, then the challenger sets $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1^{(i)}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_0, k_1^{(i)}, r^*, y^*, z^*, i])$.
- $\mathsf{Hyb}_{2,i}^{(3)}$: Same as $\mathsf{Hyb}_{2,i}^{(2)}$, except the challenger samples $r^* \xleftarrow{\text{R}} \{0,1\}^t$.
- $\mathsf{Hyb}_{2,i}^{(4)}$: Same as $\mathsf{Hyb}_{2,i}^{(3)}$, except the challenger sets $r^* = \mathsf{F}(k_{b^*}, i)$.

We now show that each pair of hybrids are indistinguishable.

Claim 4.7. *Suppose $i\mathcal{O}$ is sub-exponentially-secure with parameter $\varepsilon_{\mathsf{obf}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathsf{obf}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$. 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 for any choice of z^*, the verification program $\mathsf{VerProof}[C, k_0, k_1]$ in Hyb_2 and the verification program $\mathsf{VerProof}_2[C, k_{\mathsf{sel}}, k_0, k_1, z^*, 0]$ in $\mathsf{Hyb}_{2,0}^{(0)}$ compute identical functionalities. Take any input $x \in \{0,1\}^n$ and proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$. Consider the behavior of the program $\mathsf{VerProof}_2[C, k_{\mathsf{sel}}, k_0, k_1, z^*, 0]$ in $\mathsf{Hyb}_{2,0}^{(0)}$:

- Suppose $b = \mathsf{F}(k_{\mathsf{sel}}, x)$. Then $\mathsf{VerProof}_2[C, k_{\mathsf{sel}}, k_0, k_1, z^*, 0]$ outputs 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise. This is the same logic as $\mathsf{VerProof}[C, k_0, k_1]$.

- Suppose $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$. Since $x \in \{0,1\}^n$, the integer value of x is between 0 and $2^n - 1$. Thus, $x \geq 0$, so $\mathsf{VerProof}_2[C, k_{\mathsf{sel}}, k_0, k_1, z^*, 0]$ outputs 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise. This is the same logic as $\mathsf{VerProof}[C, k_0, k_1]$.

We conclude that on all inputs $x \in \{0,1\}^n$ and $\pi \in \{0,1\}^{t+1}$, the verification programs VerProof and $\mathsf{VerProof}_2$ in Hyb_2 and $\mathsf{Hyb}_{2,0}^{(0)}$ have identical input/output behavior. The claim now follows 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 $\Lambda_{\mathcal{B}} = \left\{(\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{obf}}} : \lambda \in \Lambda_{\mathcal{A}}\right\}$. Since $n(\lambda)$ is non-negative and $\Lambda_{\mathcal{A}}$ is infinite, the set $\Lambda_{\mathcal{B}}$ is also infinite. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a non-uniform adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) > 1/2^{\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$. We define the (inefficient) preprocessing algorithm $\mathcal{B}_1$ as follows:

1. On input $1^{\lambda_{\mathsf{obf}}}$, algorithm $\mathcal{B}_1$ first checks if there exists $\lambda \in \Lambda_{\mathcal{A}}$ such that $\lambda_{\mathsf{obf}} = (\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{obf}}}$. If no such λ exists, algorithm $\mathcal{B}_1$ outputs $\perp$. Otherwise, it sets λ to be the smallest such value that satisfies the condition.
2. Algorithm $\mathcal{B}_1$ runs $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$. It then samples $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$. Finally, it outputs the state $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, z^*, y^*)$.

The online algorithm $\mathcal{B}_2$ now proceeds as follows:

1. On input the state $\mathsf{st}_{\mathcal{B}}$, algorithm $\mathcal{B}_2$ outputs $\perp$ if $\mathsf{st}_{\mathcal{B}} = \perp$. Otherwise, it parses $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, z^*, y^*)$ and starts running $\mathcal{A}_2$ on input $\mathsf{st}_{\mathcal{A}}$. Algorithm $\mathcal{A}_2$ outputs a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
2. Algorithm $\mathcal{B}_2$ sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$.
3. Algorithm $\mathcal{B}_2$ computes the parameter s as in Construction 4.1 and gives 1^s, $\mathsf{VerProof}[C, k_0, k_1]$, and $\mathsf{VerProof}_2[C, k_{\mathsf{sel}}, k_0, k_1, z^*, 0]$ to the challenger. The challenger replies with an obfuscated program ObfVerify.
4. Algorithm $\mathcal{B}_2$ computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1])$ and gives the common reference string $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}_2$.
5. After $\mathcal{A}_2$ outputs the statement x and the proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, algorithm $\mathcal{B}_2$ outputs 1 if $\mathsf{ObfVerify}(x, \pi) = 1$ and $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$.

We now argue that $\mathcal{B}$ is efficient and compute its advantage:

- **Efficiency:** First, we argue that the state $\mathsf{st}_{\mathcal{B}}$ output by $\mathcal{B}_1$ has polynomial size. Since $\varepsilon_{\mathsf{obf}} \in (0,1)$ and $n(\lambda) \geq 1$, we have that $\lambda \leq \lambda_{\mathsf{obf}}$. By construction, $|\mathsf{st}_{\mathcal{A}}|, |z^*|, |y^*| = \mathsf{poly}(\lambda)$, so we conclude that $|\mathsf{st}_{\mathcal{B}}| = \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda_{\mathsf{obf}})$. Next, $\mathcal{A}_2$ is efficient so algorithm $\mathcal{B}_2$ is also efficient by construction.
- **Advantage:** It suffices to analyze the advantage of $\mathcal{B}$. In this case, the challenger obfuscates the program $\mathsf{VerProof}[C, k_0, k_1]$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_2. If the challenger obfuscates the program $\mathsf{VerProof}_2[C, k_{\mathsf{sel}}, k_0, k_1, z^*, 0]$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,0}^{(0)}$. Finally,

algorithm $\mathcal{B}$ computes the output using the same procedure as in Hyb_2 and $\mathsf{Hyb}_{2,0}^{(0)}$. By assumption, for all $\lambda_{\mathrm{obf}} \in \Lambda_{\mathcal{B}}$,

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

Thus, algorithm $\mathcal{B}$ succeeds with advantage greater than $2^{-\lambda_{\mathrm{obf}}^{\varepsilon_{\mathrm{obf}}}}$ for infinitely-many security parameters $\lambda_{\mathrm{obf}} \in \Lambda_{\mathcal{B}}$. This breaks sub-exponential-security of $i\mathcal{O}$ (with parameter $\varepsilon_{\mathrm{obf}}$). ◻

Claim 4.8. *Suppose $i\mathcal{O}$ is sub-exponentially-secure with parameter $\varepsilon_{\mathrm{obf}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathrm{obf}}(\lambda, n) = (\lambda+n)^{1/\varepsilon_{\mathrm{obf}}}$. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

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

Proof. Take any $i \in \{0, \ldots, 2^n - 1\}$. In both $\mathsf{Hyb}_{2,i}^{(0)}$ and $\mathsf{Hyb}_{2,i}^{(1)}$, the challenger samples $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$ and the PRF keys $k_{\mathrm{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathrm{PRF}}}, 1^n, 1^1)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathrm{PRF}}}, 1^n, 1^t)$. In $\mathsf{Hyb}_{2,i}^{(1)}$, the challenger also sets $r^* = y^* \oplus \mathsf{F}(k_{b^*}, i)$ where $b^* = 1 - \mathsf{F}(k_{\mathrm{sel}}, i)$. We start by showing that the programs $\mathsf{VerProof}_2[C, k_{\mathrm{sel}}, k_0, k_1, z^*, i]$ in $\mathsf{Hyb}_{2,i}^{(0)}$ and $\mathsf{VerProof}_3[C, k_{\mathrm{sel}}, k_0, k_1, r^*, y^*, z^*, i]$ in $\mathsf{Hyb}_{2,i}^{(1)}$ compute identical functionalities. Take any input $x \in \{0,1\}^n$ and proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$. Consider the behavior of the program $\mathsf{VerProof}_3[C, k_{\mathrm{sel}}, k_0, k_1, r^*, y^*, z^*, i]$ in $\mathsf{Hyb}_{2,i}^{(1)}$:

- Suppose $b = \mathsf{F}(k_{\mathrm{sel}}, x)$. Then $\mathsf{VerProof}_3[C, k_{\mathrm{sel}}, k_0, k_1, r^*, y^*, z^*, i]$ outputs 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise. This is the same logic as $\mathsf{VerProof}_2[C, k_{\mathrm{sel}}, k_0, k_1, z^*, i]$.
- Suppose $b = 1 - \mathsf{F}(k_{\mathrm{sel}}, x)$ and $x \neq i$. By inspection, the programs $\mathsf{VerProof}_3[C, k_{\mathrm{sel}}, k_0, k_1, r^*, y^*, z^*, i]$ and $\mathsf{VerProof}_2[C, k_{\mathrm{sel}}, k_0, k_1, z^*, i]$ implement identical logic in this case.
- Suppose $b = 1 - \mathsf{F}(k_{\mathrm{sel}}, x)$ and $x = i$. Then the program $\mathsf{VerProof}_3[C, k_{\mathrm{sel}}, k_0, k_1, r^*, y^*, z^*, i]$ in $\mathsf{Hyb}_{2,i}^{(1)}$ outputs 1 if $y \oplus r^* = y^*$. In this experiment, the challenger sets $r^* = y^* \oplus \mathsf{F}(k_{b^*}, i)$, so the program outputs 1 if $y \oplus y^* \oplus \mathsf{F}(k_{b^*}, i) = y^*$, or equivalently, if $y = \mathsf{F}(k_{b^*}, i)$. Moreover, $b^* = 1 - \mathsf{F}(k_{\mathrm{sel}}, i) = b$, so the program $\mathsf{VerProof}_3$ in this case outputs 1 if $y = \mathsf{F}(k_b, x)$ and 0 otherwise. This is exactly the same check performed in $\mathsf{VerProof}_2[C, k_{\mathrm{sel}}, k_0, k_1, z^*, i]$.

Thus, we conclude that on all inputs $x \in \{0,1\}^n$ and $\pi \in \{0,1\}^{t+1}$, the verification programs $\mathsf{VerProof}_2$ and $\mathsf{VerProof}_3$ in $\mathsf{Hyb}_{2,i}^{(0)}$ and $\mathsf{Hyb}_{2,i}^{(1)}$, respectively, have identical input/output behavior. The claim now follows by sub-exponential security of $i\mathcal{O}$ via a similar reduction as in the proof of Claim 4.7. ◻

Claim 4.9. *Suppose $i\mathcal{O}$ is sub-exponentially-secure with parameter $\varepsilon_{\mathrm{obf}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathrm{obf}}(\lambda, n) = (\lambda+n)^{1/\varepsilon_{\mathrm{obf}}}$. 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 $\lambda \geq \lambda_{\mathcal{A}}$,*

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

Proof. Take any $i \in \{0, \ldots, 2^n - 1\}$. In both $\mathsf{Hyb}_{2,i}^{(1)}$ and $\mathsf{Hyb}_{2,i}^{(2)}$, the challenger first samples $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$ and the PRF keys $k_{\mathrm{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathrm{PRF}}}, 1^n, 1^1)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathrm{PRF}}}, 1^n, 1^t)$. In addition, the challenger in both experiments computes $b^* = 1 - \mathsf{F}(k_{\mathrm{sel}}, i)$ and $r^* = y^* \oplus \mathsf{F}(k_{b^*}, i)$. Finally, in $\mathsf{Hyb}_{2,i}^{(2)}$, the challenger additionally computes $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$. We first show that if $b^* = 0$, then the proof-generation programs $\mathsf{GenProof}[C, k_{\mathrm{sel}}, k_0, k_1]$ in $\mathsf{Hyb}_{2,i}^{(1)}$ and $\mathsf{GenProof}[C, k_{\mathrm{sel}}, k_0^{(i)}, k_1]$ in $\mathsf{Hyb}_{2,i}^{(2)}$ have identical input/output behavior. Take any input $x \in \{0,1\}^n$ and $w \in \{0,1\}^h$:

- If $C(x, w) = 0$, then both programs output $\perp$.
- If $C(x, w) = 1$ and $b = \mathsf{F}(k_{\text{sel}}, x) = 1$, then both programs output $(1, \mathsf{F}(k_1, x))$.
- If $C(x, w) = 1$ and $b = \mathsf{F}(k_{\text{sel}}, x) = 0$, then $\mathsf{GenProof}[C, k_{\text{sel}}, k_0, k_1]$ outputs $(0, \mathsf{F}(k_0, x))$ while $\mathsf{GenProof}[C, k_{\text{sel}}, k_0^{(i)}, k_1]$ outputs $(0, \mathsf{F}(k_0^{(i)}, x))$. In this case, it holds that $x \neq i$ because $\mathsf{F}(k_{\text{sel}}, i) = 1 - b^* = 1$ when $b^* = 0$. Since $x \neq i$, by punctured correctness, $\mathsf{F}(k_0, x) = \mathsf{F}(k_0^{(i)}, x)$ and the program outputs are identical.

Thus, we conclude that the programs $\mathsf{GenProof}[C, k_{\text{sel}}, k_0, k_1]$ in $\mathsf{Hyb}_{2,i}^{(1)}$ and $\mathsf{GenProof}[C, k_{\text{sel}}, k_0^{(i)}, k_1]$ in $\mathsf{Hyb}_{2,i}^{(2)}$ have identical input/output behavior. Next, we show that the same holds for the verification programs $\mathsf{VerProof}_3[C, k_{\text{sel}}, k_0, k_1, r^*, y^*, z^*, i]$ in $\mathsf{Hyb}_{2,i}^{(1)}$ and $\mathsf{VerProof}_3[C, k_{\text{sel}}, k_0^{(i)}, k_1, r^*, y^*, z^*, i]$ in $\mathsf{Hyb}_{2,i}^{(2)}$. Again we first do so for the case of $b^* = 0$. Take any input $x \in \{0,1\}^n$ and $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$:

- Suppose $b = 1$. Then, the output only depends on the values of y, z^*, and $\mathsf{F}(k_1, x)$, which is the same in both experiments.
- Suppose $b = 0$. Then we have the following two possibilities:
  - Suppose $x \neq i$. By punctured correctness, we have that $\mathsf{F}(k_0, x) = \mathsf{F}(k_0^{(i)}, x)$. The outputs in this case only depends on the value of y, z^*, and $\mathsf{F}(k_0, x) = \mathsf{F}(k_0^{(i)}, x)$.
  - Suppose $x = i$. Since $b^* = 0 = 1 - \mathsf{F}(k_{\text{sel}}, x)$, we have that $\mathsf{F}(k_{\text{sel}}, x) = 1$ in this case. Since $b = 0$, this means that $b = 1 - \mathsf{F}(k_{\text{sel}}, i)$, and so both programs output 1 if $y \oplus r^* = y^*$ and 0 otherwise.

Once more, we conclude that the verification programs $\mathsf{VerProof}_3$ in $\mathsf{Hyb}_{2,i}^{(1)}$ and $\mathsf{Hyb}_{2,i}^{(2)}$ have identical input/output behavior. An analogous argument shows that the GenProof and VerProof programs in the two experiments have identical input/output behavior when $b^* = 1 - \mathsf{F}(k_{\text{sel}}, i) = 1$. To complete the proof we introduce an intermediate hybrid:

- iHyb_i: Same as $\mathsf{Hyb}_{2,i}^{(2)}$ except the challenger computes ObfVerify as in $\mathsf{Hyb}_{2,i}^{(1)}$. Namely, it computes $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\text{obf}}}, 1^s, \mathsf{VerProof}_3[C, k_{\text{sel}}, k_0, k_1, r^*, y^*, z^*, i])$.

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

$$|\Pr[\mathsf{Hyb}_{2,i}^{(1)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_i(\mathcal{A}) = 1]| > 1/2^{\lambda + n(\lambda)}. \tag{4.1}$$

Let $\Lambda_{\mathcal{B}} = \left\{(\lambda + n(\lambda))^{1/\varepsilon_{\text{obf}}} : \lambda \in \Lambda_{\mathcal{A}}\right\}$. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a non-uniform algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that for all $\lambda_{\text{obf}} \in \Lambda_{\mathcal{B}}$, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\text{obf}}) > 1/2^{\lambda_{\text{obf}}^{\varepsilon_{\text{obf}}}}$. We define the (inefficient) preprocessing algorithm $\mathcal{B}_1$ as follows:

1. On input $1^{\lambda_{\text{obf}}}$, algorithm $\mathcal{B}_1$ first checks if there exists $\lambda \in \Lambda_{\mathcal{A}}$ such that $\lambda_{\text{obf}} = (\lambda + n(\lambda))^{1/\varepsilon_{\text{obf}}}$. If no such λ exists, algorithm $\mathcal{B}_1$ outputs $\perp$. Otherwise, it sets λ to be the smallest such value that satisfies the condition.
2. Algorithm $\mathcal{B}_1$ runs $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$. It then samples $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$ and outputs the state $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, z^*, y^*)$.

The online algorithm $\mathcal{B}_2$ now proceeds as follows:

1. On input the state $\mathsf{st}_{\mathcal{B}}$, algorithm $\mathcal{B}_2$ outputs $\perp$ if $\mathsf{st}_{\mathcal{B}} = \perp$. Otherwise, it parses $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, z^*, y^*)$ and starts running $\mathcal{A}_2$ on input $\mathsf{st}_{\mathcal{A}}$. Algorithm $\mathcal{A}_2$ outputs a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

2. Algorithm $\mathcal{B}_2$ sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$. It computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$, $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$, and $r^* = y^* \oplus \mathsf{F}(k_{b^*}, i)$.

3. Algorithm $\mathcal{B}_2$ computes the parameter s as in Construction 4.1. It constructs the challenge as follows:

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

   The challenger replies with an obfuscated program $\mathsf{ObfProve}$.

4. Algorithm $\mathcal{B}_2$ computes $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_0, k_1, r^*, y^*, z^*, i])$ and gives the common reference string $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}_2$.

5. After $\mathcal{A}_2$ outputs the statement x and the proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, algorithm $\mathcal{B}_2$ outputs 1 if $\mathsf{ObfVerify}(x, \pi) = 1$ and $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$.

We now argue that $\mathcal{B}$ is efficient and compute its advantage:

- **Efficiency:** First, we argue that the state $\mathsf{st}_{\mathcal{B}}$ output by $\mathcal{B}_1$ has polynomial size. Since $\varepsilon_{\mathsf{obf}} \in (0,1)$ and $n(\lambda) \geq 1$, we have that $\lambda \leq \lambda_{\mathsf{obf}}$. By construction, $|\mathsf{st}_{\mathcal{A}}|, |z^*|, |y^*| = \mathsf{poly}(\lambda)$, so we conclude that $|\mathsf{st}_{\mathcal{B}}| = \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda_{\mathsf{obf}})$. Next, $\mathcal{A}_2$ is efficient so algorithm $\mathcal{B}_2$ is also efficient by construction.

- **Advantage:** If the challenger obfuscates the program $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1]$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,i}^{(1)}$. If the challenger obfuscates the program $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0^{(i)}, k_1]$ (when $b^* = 0$) or the program $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1^{(i)}]$ (when $b^* = 1$), then algorithm $\mathcal{B}$ perfectly simulates iHyb_i. Finally, algorithm $\mathcal{B}_2$ computes the output using the same procedure as in $\mathsf{Hyb}_{2,i}^{(1)}$ and iHyb_i. By assumption then, for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$,

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

  Thus, algorithm $\mathcal{B}$ succeeds with advantage greater than $2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$ for infinitely-many $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$. This breaks sub-exponential-security of $i\mathcal{O}$ (with parameter $\varepsilon_{\mathsf{obf}}$).

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

$$|\Pr[\mathsf{iHyb}_i(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i}^{(2)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda + n(\lambda)}. \tag{4.2}$$

Combining Eqs. (4.1) and (4.2), we conclude that for all sufficiently-large $\lambda \in \mathbb{N}$,

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

Claim 4.10. *Suppose Π_{PPRF} satisfies selective sub-exponential puncturing security with parameter $\varepsilon_{\mathsf{PRF}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathsf{PRF}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

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

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

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

Let $\Lambda_{\mathcal{B}} = \left\{(\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{PRF}}} : \lambda \in \Lambda_{\mathcal{A}}\right\}$. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a non-uniform algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that for all $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, $\mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) > 1/2^{\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}$. We define the (inefficient) preprocessing algorithm $\mathcal{B}_1$ as follows:

1. On input $1^{\lambda_{\mathsf{PRF}}}$, algorithm $\mathcal{B}_1$ first checks if there exists $\lambda \in \Lambda_{\mathcal{A}}$ such that $\lambda_{\mathsf{PRF}} = (\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{PRF}}}$. If no such λ exists, algorithm $\mathcal{B}_1$ outputs $\perp$. Otherwise, it sets λ to be the smallest such value that satisfies the condition.

2. Algorithm $\mathcal{B}_1$ runs $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$. It then samples $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$ and outputs the state $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, z^*, y^*)$.

The online algorithm $\mathcal{B}_2$ now proceeds as follows:

1. On input the state $\mathsf{st}_{\mathcal{B}}$, algorithm $\mathcal{B}_2$ outputs $\perp$ if $\mathsf{st}_{\mathcal{B}} = \perp$. Otherwise, it parses $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, z^*, y^*)$ and starts running $\mathcal{A}_2$ on input $\mathsf{st}_{\mathcal{A}}$. Algorithm $\mathcal{A}_2$ outputs a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

2. Algorithm $\mathcal{B}_2$ samples $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$ and computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$. It samples $k_{1-b^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$.

3. Algorithm $\mathcal{B}_2$ submits the input length 1^n, the output length 1^t, and the point $i \in \{0,1\}^n$ to the punctured PRF challenger. It receives a punctured key $k_{b^*}^{(i)}$ and a challenge value $r' \in \{0,1\}^t$. Algorithm $\mathcal{B}_2$ sets $r^* = y^* \oplus r'$.

4. Algorithm $\mathcal{B}_2$ sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$ and constructs the programs $\mathsf{ObfProve}$ and $\mathsf{ObfVerify}$ as follows:

   - If $b^* = 0$, then it computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_{1-b^*}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_{1-b^*}, r^*, y^*, z^*, i])$.
   - If $b^* = 1$, then it computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, k_{\mathsf{sel}}, k_{1-b^*}, k_{b^*}^{(i)}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_{1-b^*}, k_{b^*}^{(i)}, r^*, y^*, z^*, i])$.

   Algorithm $\mathcal{B}_2$ gives the common reference string $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}_2$.

5. After algorithm $\mathcal{A}_2$ outputs the statement x and the proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, algorithm $\mathcal{B}_2$ outputs 1 if $\mathsf{ObfVerify}(x, \pi) = 1$ and $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$.

We now argue that $\mathcal{B}$ is efficient and compute its advantage:

- **Efficiency:** First, we argue that the state $\mathsf{st}_{\mathcal{B}}$ output by $\mathcal{B}_1$ has polynomial size. Since $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and $n(\lambda) \geq 1$, we have that $\lambda \leq \lambda_{\mathsf{PRF}}$. By construction, $|\mathsf{st}_{\mathcal{A}}|, |z^*|, |y^*| = \mathsf{poly}(\lambda)$, so we conclude that $|\mathsf{st}_{\mathcal{B}}| = \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda_{\mathsf{PRF}})$. Next, $\mathcal{A}_2$ is efficient so algorithm $\mathcal{B}_2$ is also efficient by construction.
- **Advantage:** By definition, the punctured PRF challenger constructs key $k_{b^*}^{(i)}$ by first sampling $k_{b^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$ and setting $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$. This matches the specification in $\mathsf{Hyb}_{2,i}^{(2)}$ and $\mathsf{Hyb}_{2,i}^{(3)}$. Consider now the distribution of the challenge value r^*:

– Suppose $r' = \mathsf{F}(k_{b^*}, i)$. In this case, algorithm $\mathcal{B}_2$ sets $r^* = y^* \oplus r' = y^* \oplus \mathsf{F}(k_{b^*}, i)$. This corresponds to the distribution of $\mathsf{Hyb}_{2,i}^{(2)}$. Moreover algorithm $\mathcal{B}_2$ computes the outputs using the same procedure as in $\mathsf{Hyb}_{2,i}^{(2)}$ and $\mathsf{Hyb}_{2,i}^{(3)}$. Thus, in this case, algorithm $\mathcal{B}_2$ outputs 1 with probability $\Pr[\mathsf{Hyb}_{2,i}^{(2)}(\mathcal{A}) = 1]$.
– Suppose $r' \xleftarrow{\text{R}} \{0,1\}^t$. In this case, algorithm $\mathcal{B}_2$ sets $r^* = y^* \oplus r'$. Since r' is sampled independently of all other quantities, the distribution of r^* in this case is also uniform over $\{0,1\}^t$. Thus, algorithm $\mathcal{B}_2$ perfectly simulates an execution of $\mathsf{Hyb}_{2,i}^{(3)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{2,i}^{(3)}(\mathcal{A}) = 1]$.

Combining the above analysis, we have for all $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$,

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

We conclude that algorithm $\mathcal{B}$ succeeds with advantage greater than $2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}$ for infinitely-many $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$. This breaks selective sub-exponential puncturing security of Π_{PPRF} (with parameter $\varepsilon_{\mathsf{PRF}}$). ☐

Claim 4.11. *Suppose Π_{PPRF} satisfies selective sub-exponential puncturing security with parameter $\varepsilon_{\mathsf{PRF}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathsf{PRF}}(\lambda, n) = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

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

Proof. Follow by an analogous argument as the proof of Claim 4.10. ☐

Claim 4.12. *Suppose $i\mathcal{O}$ is sub-exponentially-secure with parameter $\varepsilon_{\mathsf{obf}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathsf{obf}}(\lambda, n) = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}$. Suppose Π_{PPRF} satisfies punctured correctness and Π_{OWF} is correct and injective. 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 follow by a similar argument as in the proof of Claim 4.9. To argue this, we start by showing that the programs associated with ObfProve and ObfVerify have identical behavior in the two experiments. The claim then follows by sub-exponential security of $i\mathcal{O}$ (as in the proof of Claim 4.9). We emphasize here that our analysis here critically relies on *injectivity* of Π_{OWF}. Indeed, the crux of this argument is changing the verification check for $x = i$ as follows:

$$\text{output 1 if } y \oplus \mathsf{F}(k_{b^*}, i) = y^* \implies \text{output 1 if } \mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_{b^*}, i)) = 1,$$

where $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$. These two checks are identical only in the case where Π_{OWF} is injective. If Π_{OWF} is not injective, there can be multiple inputs y where $\mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_{b^*}, i)) = 1$, but only a single input where $y \oplus \mathsf{F}(k_{b^*}, i) = y^*$.

We now give the formal argument. Take any index $i \in \{0, \ldots, 2^n - 1\}$ and consider an execution of $\mathsf{Hyb}_{2,i}^{(4)}$ and $\mathsf{Hyb}_{2,i+1}^{(0)}$. In both experiments, the challenger samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$. It also samples $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$. In $\mathsf{Hyb}_{2,i}^{(4)}$, the challenger additionally computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$, $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$, and $r^* = \mathsf{F}(k_{b^*}, i)$. We analyze the proof-generation and the proof-verification programs in the two experiments. We start by analyzing the case where $b^* = 0$; the case where $b^* = 1$ follows similarly:

The $\mathsf{GenProof}$ **programs.** We start by considering the proof-generation programs. In $\mathsf{Hyb}_{2,i}^{(4)}$, the challenger obfuscates the program $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0^{(i)}, k_1]$ whereas in $\mathsf{Hyb}_{2,i+1}^{(0)}$, the challenger obfuscates the program $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1]$. By the same argument as in the proof of Claim 4.9, these two programs compute identical functionality. In particular, by punctured correctness, $\mathsf{F}(k_0, x) = \mathsf{F}(k_0^{(i)}, x)$ for all $x \neq i$, and neither program needs to evaluate the PRF with k_0 (or $k_0^{(i)}$) at i since $\mathsf{F}(k_{\mathsf{sel}}, i) = 1 \neq b^*$.

The $\mathsf{VerProof}$ **programs.** Next, we consider the verification programs. In $\mathsf{Hyb}_{2,i}^{(4)}$, the challenger obfuscates the program $\mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_0^{(i)}, k_1, r^*, y^*, z^*, i]$ whereas in $\mathsf{Hyb}_{2,i+1}^{(0)}$, the challenger obfuscates the program $\mathsf{VerProof}_2[C, k_{\mathsf{sel}}, k_0, k_1, z^*, i+1]$. We show that these two programs compute identical functionality. Take any input $x \in \{0,1\}^n$ and $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$:

- Suppose $b = 1$. Recall that when $0 = b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$, it holds that $\mathsf{F}(k_{\mathsf{sel}}, i) = 1 = b$. Thus, there are two possibilities: either (1) $b = \mathsf{F}(k_{\mathsf{sel}}, x)$; or (2) $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$ *and* $x \neq i$ (recall that when $b^* = 0$, we have that $\mathsf{F}(k_{\mathsf{sel}}, i) = 1$). We consider each one individually:
  - Suppose $b = \mathsf{F}(k_{\mathsf{sel}}, x)$. Then, both programs output 1 if $\mathsf{F}(k_1, x) = y$ and 0 otherwise.
  - Suppose $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$ and $x \neq i$. If $x < i$, both programs output $\mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_1, x))$ and if $x > i$, both programs output 1 if $\mathsf{F}(k_1, x) = y$ and 0 otherwise.
- Suppose $b = 0$. Since $\mathsf{F}(k_{\mathsf{sel}}, i) = 1$, there are two possibilities: either (1) $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$; or (2) $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ *and* $x \neq i$. We consider these possibilities:
  - Suppose $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and $x \neq i$. In this case, the program $\mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_0^{(i)}, k_1, r^*, y^*, z^*, i]$ in $\mathsf{Hyb}_{2,i}^{(4)}$ outputs 1 if $\mathsf{F}(k_0^{(i)}, x) = y$, whereas the program $\mathsf{VerProof}_2[C, k_{\mathsf{sel}}, k_0, k_1, z^*, i+1]$ in $\mathsf{Hyb}_{2,i+1}^{(0)}$ outputs 1 if $\mathsf{F}(k_0, x) = y$. Since $x \neq i$, punctured correctness of Π_{PPRF} implies that $\mathsf{F}(k_0, x) = \mathsf{F}(k_0^{(i)}, x)$, and the outputs of the two programs are identical.
  - Suppose $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$ and $x \neq i$. If $x < i$, the program in $\mathsf{Hyb}_{2,i}^{(4)}$ outputs $\mathsf{OWF.Verify}(z^*, y \oplus F(k_0^{(i)}, x))$ whereas the program in $\mathsf{Hyb}_{2,i+1}^{(0)}$ outputs $\mathsf{OWF.Verify}(z^*, y \oplus F(k_0, x))$. By punctured correctness, the outputs are equivalent. If $x > i$, the program in $\mathsf{Hyb}_{2,i}^{(4)}$ outputs 1 if $\mathsf{F}(k_0^{(i)}, x) = y$ while the program in $\mathsf{Hyb}_{2,i+1}^{(0)}$ outputs 1 if $\mathsf{F}(k_0, x) = y$. These are the same by punctured correctness.
  - Suppose $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$ and $x = i$. In this case, the program $\mathsf{VerProof}_3[C, k_{\mathsf{sel}}, k_0^{(i)}, k_1, r^*, y^*, z^*, i]$ in $\mathsf{Hyb}_{2,i}^{(4)}$ outputs 1 if $y \oplus r^* = y^*$. In this case (with $b^* = 0$), $r^* = \mathsf{F}(k_0, i)$. Since the challenger in $\mathsf{Hyb}_{2,i}^{(4)}$ sampled $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$, correctness and injectivity of Π_{OWF} states that $\mathsf{OWF.Verify}(z^*, y^*) = 1$ and for all $y \neq y^*$, $\mathsf{OWF.Verify}(z^*, y) = 0$. Equivalently,

    $$y \oplus \mathsf{F}(k_0, i) = y^* \quad \text{if and only if} \quad \mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_0, i)) = 1.$$

    In other words, the output of the verification program in $\mathsf{Hyb}_{2,i}^{(4)}$ is 1 if

    $$\mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_0, i)) = 1$$

    and is 0 otherwise. This is the same condition checked by the program in $\mathsf{Hyb}_{2,i+1}^{(0)}$. Observe that this is the case that critically relies on *injectivity* of Π_{OWF}.

We conclude that on all inputs $x \in \{0,1\}^n$ and $\pi \in \{0,1\}^t$, the behavior of the GenProof and VerProof programs in $\mathsf{Hyb}_{2,i}^{(4)}$ and $\mathsf{Hyb}_{2,i+1}^{(0)}$ is identical when $b^* = 0$. A similar analysis applies when $b^* = 1$. The claim now follows by sub-exponential security of $i\mathcal{O}$ (as in the proof of Claim 4.9). □

Claim 4.13. *Suppose $i\mathcal{O}$ is sub-exponentially-secure with parameter $\varepsilon_{\mathrm{obf}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathrm{obf}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathrm{obf}}}$. 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 4.7. We first show that the program $\mathsf{VerProof}_2[C, k_{\mathrm{sel}}, k_0, k_1, z^*, 2^n]$ in Hybrid $\mathsf{Hyb}_{2,2^n}^{(0)}$ and the program $\mathsf{VerProof}_1[C, k_{\mathrm{sel}}, k_0, k_1, z^*]$ in Hyb_3 compute identical functionalities. Take any input $x \in \{0,1\}^n$ and $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$.

- If $b = \mathsf{F}(k_{\mathrm{sel}}, x)$, then both programs output 1 if $\mathsf{F}(k_b, x) = y$ and 0 otherwise.
- If $b = 1 - \mathsf{F}(k_{\mathrm{sel}}, x)$, then both programs output $\mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_b, x))$. Note that this follows because for all $i \in \{0, \ldots, 2^n - 1\}$, it holds that $x < 2^n$.

Both experiments sample $k_{\mathrm{sel}}, k_0, k_1$, and z^* using identical procedures. We conclude that the two programs compute identical functionality. The claim now follows by sub-exponential security of $i\mathcal{O}$ (as in the proof of Claim 4.7). □

We now return to the proof of Lemma 4.6. By Claims 4.8 to 4.12, for all $i \in \{0, \ldots, 2^n - 1\}$, and all sufficiently-large $\lambda \in \mathbb{N}$, it follows that

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

By the triangle inequality,

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

Combined with Claims 4.7 and 4.13, we conclude that

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

Lemma 4.14. *Suppose Π_{OWF} is one-way against non-uniform adversaries. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \leq \mathsf{negl}(\lambda)$.

Proof. Suppose $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] > \varepsilon(\lambda)$ for some non-negligible function ε. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a non-uniform adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that breaks one-wayness of Π_{OWF}. First the preprocessing algorithm $\mathcal{B}_1$ takes the security parameter 1^λ as input, runs $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$, and outputs $\mathsf{st}_{\mathcal{B}} = \mathsf{st}_{\mathcal{A}}$. The online algorithm $\mathcal{B}_2$ then works as follows:

1. On input the state $\mathsf{st}_{\mathcal{B}} = \mathsf{st}_{\mathcal{A}}$, algorithm $\mathcal{B}_2$ runs algorithm $\mathcal{A}_2$ on the state $\mathsf{st}_{\mathcal{A}}$. Algorithm $\mathcal{A}_2$ starts by outputting a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
2. Algorithm $\mathcal{B}_2$ computes $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathrm{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$.

3. Algorithm $\mathcal{B}_2$ sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$ and constructs the obfuscated programs

$$\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1])$$
$$\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_1[C, k_{\mathsf{sel}}, k_0, k_1, z^*]).$$

   It gives $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$.

4. After $\mathcal{A}_2$ outputs a statement $x \in \{0,1\}^n$ and a proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, algorithm $\mathcal{B}_2$ outputs $y \oplus \mathsf{F}(k_b, x)$.

By definition, the one-wayness challenger samples $(z^*, y^*) \leftarrow \mathsf{OWF.GenChal}(1^\lambda)$, which matches the distribution in Hyb_3. Thus, with probability ε, algorithm $\mathcal{A}$ outputs x and $\pi = (b, y)$ where $\mathsf{ObfVerify}(x, \pi) = 1$ and $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$. By correctness of $i\mathcal{O}$ and construction of $\mathsf{VerProof}_1$, if $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$, then $\mathsf{ObfVerify}(x, \pi) = 1$ if and only if $\mathsf{OWF.Verify}(z^*, y \oplus \mathsf{F}(k_b, x))$. This means that $y \oplus \mathsf{F}(k_b, x)$ is a preimage of z^* and algorithm $\mathcal{B}$ successfully produces a preimage of z^*. □

Combining Lemmas 4.4 to 4.6, we have for all sufficiently-large $\lambda \in \mathbb{N}$,

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

By Lemma 4.14, $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$. We conclude that

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

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

Theorem 4.15 (Succinctness)**.** *Construction 4.1 is succinct.*

Proof. A proof π in Construction 4.1 consists of a bit $b \in \{0,1\}$ and an element of $\{0,1\}^t$ where $t = t(\lambda)$ is the length of the input to the injective one-way function with inefficient sampler (Definition 3.2). Since $t = t(\lambda)$ is polynomially-bounded in the security parameter, $|\pi| = \mathsf{poly}(\lambda)$ and succinctness holds. □

Remark 4.16 (Perfect Zero-Knowledge)**.** Similar to previous $i\mathcal{O}$-based SNARGs [SW14, WW24a, WZ24], Construction 4.1 satisfies perfect zero-knowledge (the proof is just the output of a PRF on the statement, which can be perfectly simulated). We refer to the previous works for a formal proof of this.

Acknowledgments

Brent Waters is supported by NSF CNS-1908611, CNS-2318701, and a Simons Investigator award. David J. Wu is supported by NSF 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*, 2022.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, 1996.

[AS15] Gilad Asharov and Gil Segev. Limits on the power of indistinguishability obfuscation and functional encryption. In *FOCS*, 2015.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, 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), 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*, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC*, 2013.

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

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *STOC*, 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*, 2001.

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

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

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

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 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*, 2023.

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

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

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

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

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

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

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

[GLN11] Oded Goldreich, Leonid A. Levin, and Noam Nisan. On constructing 1-1 one-way functions. In *Studies in Complexity and Cryptography*, volume 6650. Springer, 2011.

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

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

[GZ21] Alonso González and Alexandros Zacharakis. Fully-succinct publicly verifiable delegation from constant-size assumptions. In *TCC*, 2021.

[JKLV24] Zhengzhong Jin, Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs under LWE via propositional proofs. In *STOC*, 2024.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, 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*, 2022.

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

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

[KMN+14] Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. In *FOCS*, 2014.

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

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

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

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

[MPV24] Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Adaptively sound zero-knowledge SNARKs for UP. In *CRYPTO*, 2024.

[NWW23] Shafik Nassar, Brent Waters, and David J. Wu. Monotone policy bargs from bargs and additively homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2023.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. Cryptology ePrint Archive, Paper 2024/856, 2024. https://eprint.iacr.org/2024/856.

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

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

[WW24a] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. In *STOC*, 2024.

[WW24b] Hoeteck Wee and David J. Wu. Succinct functional commitments for circuits from k-Lin. In *EUROCRYPT*, 2024.

[WZ24] Brent Waters and Mark Zhandry. Adaptive security in SNARGs via iO and lossy functions. In *CRYPTO*, 2024.

A Analysis of Construction 4.1

In this section, we provide the proof of Lemma 4.4. Many of these proofs follow from the corresponding proofs in [WW24a], with minor adaptions accounting for the structural differences of our scheme.

A.1 Proof of Lemma 4.4

The proof of Lemma 4.4 follows by a similar argument as the proof of Lemma 4.4 in [WW24a]. We recall the full argument here. We follow the style and conventions of [WW24a, Lemma 4.4], so some of the text is taken verbatim from the previous work. Consider an execution of Hyb_0 or Hyb_1. For an index $i \in \{0,1\}^n$, let E_i be the event that adversary $\mathcal{A}$ outputs i as its statement in an execution of Hyb_0 or Hyb_1. By definition, we can write

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

We show that for all $i \in \{0,1\}^n$,

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

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 both Hyb_0 and Hyb_1 is 0. In this case,

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

Correspondingly, Eq. (A.2) 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

$$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{ObfVerify}(x, \pi) = 1 \quad \text{and} \quad x = i.$$

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

> **Input:** statement $x \in \{0,1\}^n$ and witness $w \in \{0,1\}^h$
> **Hard-coded:** Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, PRF keys $k_{\mathsf{sel}}, k_0, k_1$, and the 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 $C(x, w) = 1$, compute $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and output $(b, \mathsf{F}(k_b, x))$.

Figure 6: The proof-generation program $\mathsf{GenProof}_1[C, k_{\mathsf{sel}}, k_0, k_1, i]$.

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 $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}_1[C, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i])$, where the size parameter s is as defined in Construction 4.1. The rest of the experiment 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 $\pi = (b, y) \in \{0,1\}^{t+1}$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, the challenger samples a random bit $b' \xleftarrow{\mathrm{R}} \{0,1\}$ and outputs 1 if

$$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{ObfVerify}(x, \pi) = 1 \quad \text{and} \quad x = i \quad \text{and} \quad b \neq b'.$$

- $\mathsf{Hyb}_{0,i}^{(3)}$: Same as $\mathsf{Hyb}_{0,i}^{(2)}$, except after the adversary outputs its statement x and the proof $\pi = (b, y) \in \{0,1\}^{t+1}$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, the challenger outputs 1 if

$$(C, x) \notin \mathcal{L}_{\mathsf{SAT}} \quad \text{and} \quad \mathsf{ObfVerify}(x, \pi) = 1 \quad \text{and} \quad x = i \quad \text{and} \quad b \neq \mathsf{F}(k_{\mathsf{sel}}, i).$$

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

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 A.1. *Suppose iO is sub-exponentially-secure with parameter $\varepsilon_{\mathsf{obf}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathsf{obf}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$. Suppose Π_{PPRF} satisfies punctured correctness. Then, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

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

Proof. We start by showing that $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1]$ in Hyb_0 and $\mathsf{GenProof}_1[C, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i]$ in Hyb_1 compute identical functionalities. Take any input (x, w) to the two programs. We consider the different possibilities:

- 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 $x \neq i$ and $C(x, w) = 1$. Then GenProof computes $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and outputs $(b, \mathsf{F}(k_b, x))$ while $\mathsf{GenProof}_1$ computes $b = \mathsf{F}(k_{\mathsf{sel}}^{(i)}, x)$ and outputs $(b, \mathsf{F}(k_b, x))$. Since $x \neq i$ and 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 is identical.

We conclude that the two programs behave identically on all inputs. The claim now follows by *iO* security. Formally, suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$,

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

Let $\Lambda_{\mathcal{B}} = \{(\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{obf}}} : \lambda \in \Lambda_{\mathcal{A}}\}$. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a non-uniform algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) > 1/2^{\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$. We define the (inefficient) preprocessing algorithm $\mathcal{B}_1$ as follows:

1. On input $1^{\lambda_{\mathsf{obf}}}$, algorithm $\mathcal{B}_1$ first checks if there exists $\lambda \in \Lambda_{\mathcal{A}}$ such that $\lambda_{\mathsf{obf}} = (\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{obf}}}$. If no such λ exists, algorithm $\mathcal{B}_1$ outputs $\perp$. Otherwise, it sets λ to be the smallest such value that satisfies the condition.
2. Algorithm $\mathcal{B}_1$ runs $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$ and outputs the state $\mathsf{st}_{\mathcal{B}} = \mathsf{st}_{\mathcal{A}}$.

The online algorithm $\mathcal{B}_2$ now proceeds as follows:

1. On input the state $\mathsf{st}_{\mathcal{B}}$, algorithm $\mathcal{B}_2$ outputs $\perp$ if $\mathsf{st}_{\mathcal{B}} = \perp$. Otherwise, it parses $\mathsf{st}_{\mathcal{B}} = \mathsf{st}_{\mathcal{A}}$ and runs $\mathcal{A}_2$ on input $\mathsf{st}_{\mathcal{A}}$. Algorithm $\mathcal{A}_2$ outputs a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
2. Algorithm $\mathcal{B}_2$ sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$. It computes $k_{\mathsf{sel}}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{sel}}, i)$.

3. Algorithm $\mathcal{B}_2$ computes s as in Construction 4.1 and gives the size parameter 1^s and the programs $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1]$ and $\mathsf{GenProof}_1[C, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i]$ to the challenger. It receive the obfuscated program ObfProve.

4. Algorithm $\mathcal{B}_2$ then computes $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}[C, k_0, k_1])$ and gives $\mathcal{A}_2$ the common reference string $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$.

5. After algorithm $\mathcal{A}_2$ outputs a statement x and a proof π, algorithm $\mathcal{B}_2$ outputs 1 if $x = i$ and $\mathsf{ObfVerify}(x, \pi) = 1$.

Since $\varepsilon_{\mathsf{obf}} \in (0, 1)$ and $n(\lambda)$ is non-negative, it follows that the value of λ (if one exists) computed by $\mathcal{B}_1$ satisfies $\lambda < \lambda_{\mathsf{obf}}$. As such, $|\mathsf{st}_{\mathcal{A}}| = \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda_{\mathsf{obf}})$, so $\mathcal{B}$ satisfies the efficiency requirements. Now consider its advantage. If the challenger obfuscates the program $\mathsf{GenProof}[C, k_{\mathsf{sel}}, k_0, k_1]$, 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 $\mathsf{GenProof}_1[C, k_{\mathsf{sel}}^{(i)}, k_0, k_1, i]$, 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. (A.3), for all $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, it holds that

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

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

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 A.3. *Suppose* Π_{PPRF} *satisfies selective sub-exponential punctured security with parameter* $\varepsilon_{\mathsf{PRF}} \in (0, 1)$ *against non-uniform adversaries and* $\lambda_{\mathsf{PRF}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. *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 $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a non-uniform algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ such that for all $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, $\mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) > 1/2^{\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}}$. We define the (inefficient) preprocessing algorithm $\mathcal{B}_1$ as follows:

1. On input $1^{\lambda_{\mathsf{PRF}}}$, algorithm $\mathcal{B}_1$ first checks if there exists $\lambda \in \Lambda_{\mathcal{A}}$ such that $\lambda_{\mathsf{PRF}} = (\lambda + n(\lambda))^{1/\varepsilon_{\mathsf{PRF}}}$. If no such λ exists, algorithm $\mathcal{B}_1$ outputs $\perp$. Otherwise, it sets λ to be the smallest such value that satisfies the condition.

2. Algorithm $\mathcal{B}_1$ runs $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^{\lambda})$ and outputs $\mathsf{st}_{\mathcal{B}} = \mathsf{st}_{\mathcal{A}}$.

The online algorithm $\mathcal{B}_2$ now proceeds as follows:

1. On input the state $\mathsf{st}_{\mathcal{B}}$, algorithm $\mathcal{B}_2$ outputs $\perp$ if $\mathsf{st}_{\mathcal{B}} = \perp$. Otherwise, it parses $\mathsf{st}_{\mathcal{B}} = \mathsf{st}_{\mathcal{A}}$ and starts running $\mathcal{A}_2$ on input $\mathsf{st}_{\mathcal{A}}$. Algorithm $\mathcal{A}_2$ outputs a circuit $C \colon \{0, 1\}^n \times \{0, 1\}^h \to \{0, 1\}$.

2. Algorithm $\mathcal{B}_2$ samples $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^t)$. 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 $\mathcal{B}_2$ sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$, and computes

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

   It gives $\mathsf{crs} = (\mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}_2$.

4. After algorithm $\mathcal{A}_2$ outputs a statement x and a proof $\pi = (b, y)$ where $b \in \{0,1\}$ and $y \in \{0,1\}^t$, algorithm $\mathcal{B}_2$ outputs 1 if $x = i$, $\mathsf{ObfVerify}(x, \pi) = 1$, and $b \neq b'$.

Since $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and $n(\lambda)$ is non-negative, it follows that the value of λ (if one exists) computed by $\mathcal{B}_1$ satisfies $\lambda < \lambda_{\mathsf{PRF}}$. As such, $|\mathsf{st}_{\mathcal{A}}| = \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda_{\mathsf{PRF}})$, so $\mathcal{B}$ satisfies the efficiency requirements. Now consider its advantage. 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, for all $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$,

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

Claim A.4. *Suppose $i\mathcal{O}$ is sub-exponentially-secure with parameter $\varepsilon_{\mathsf{obf}} \in (0,1)$ against non-uniform adversaries and $\lambda_{\mathsf{obf}}(\lambda, n) = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$. Suppose also that 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 A.1. $\square$

Combining Claims A.1 to A.4, we conclude that for all $i \in \{0,1\}^n$ where $(C, i) \notin \mathcal{L}_{\mathsf{SAT}}$, Eq. (A.2) holds. Combined with Eq. (A.1), 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 4.4 follows. $\square$