# Batching Adaptively-Sound SNARGs for NP

Lalita Devadas
MIT
lali@mit.edu

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

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

**Abstract**

A succinct non-interactive argument (SNARG) for NP allows a prover to convince a verifier that an NP statement $x$ is true with a proof whose size is *sublinear* in the length of the traditional NP witness. Moreover, a SNARG is adaptively sound if the adversary can choose the statement it wants to prove after seeing the scheme parameters. Very recently, Waters and Wu (STOC 2024) showed how to construct adaptively-sound SNARGs for NP in the plain model from falsifiable assumptions (specifically, sub-exponentially-secure indistinguishability obfuscation, sub-exponentially-secure one-way functions, and polynomial hardness of discrete log).

We consider the *batch* setting where the prover wants to prove a collection of $T$ statements $x_1, \ldots, x_T$ and its goal is to construct a proof whose size is sublinear in both the size of a single witness *and* the number of instances $T$. In this setting, existing constructions either require the size of the public parameters to scale linearly with $T$ (and thus, can only support an a priori bounded number of instances), or only provide non-adaptive soundness, or have proof size that scales linearly with the size of a single NP witness. In this work, we give two approaches for batching adaptively-sound SNARGs for NP, and in particular, show that under the same set of assumptions as those underlying the Waters-Wu adaptively-sound SNARG, we can obtain an adaptively-sound SNARG for batch NP where the size of the proof is $\mathsf{poly}(\lambda)$ and the size of the CRS is $\mathsf{poly}(\lambda + |C|)$, where $\lambda$ is a security parameter and $|C|$ is the size of the circuit that computes the associated NP relation.

Our first approach builds directly on top of the Waters-Wu construction and relies on indistinguishability obfuscation and a *homomorphic* re-randomizable one-way function. Our second approach shows how to combine ideas from the Waters-Wu SNARG with the chaining-based approach by Garg, Sheridan, Waters, and Wu (TCC 2022) to obtain a SNARG for batch NP.

## 1 Introduction

Succinct non-interactive arguments (SNARGs) for NP allow an efficient prover to convince a verifier that an NP statement $x$ (with associated witness $w$) is true with a proof whose size scales with $o(|x| + |w|)$. The main security requirement is computational soundness which says that a computationally-bounded prover should not be able to convince a verifier of a false statement. SNARGs were first constructed in the random oracle model [Kil92, Mic94]. Many works have subsequently shown how to construct SNARGs in the plain model assuming the prover and the verifier have access to a common reference string (CRS).

Until recently, SNARGs for NP in the CRS model have either relied on non-falsifiable cryptographic assumptions (c.f., [Gro10,BCCT12,DFH12,Lip13,GGPR13,BCI+13,BCPR14,BISW17,BCC+17,ACL+22,CLM23] and the references therein) or satisfied the weaker notion of *non-adaptive* soundness [SW14], where soundness only holds against an adversary that declares its false statement before seeing the CRS. In contrast, the standard or "adaptive" notion of soundness allows the malicious prover to choose the statement *after*

seeing the CRS. Very recently, several works gave the first adaptively-sound SNARGs for NP using indistinguishability obfuscation ($i\mathcal{O}$) and either a sub-exponentially-secure re-randomizable one-way function [WW24a] or a sub-exponentially-secure lossy function [WZ24].[1] Moreover, in the *designated-verifier* model where a secret key is needed to verify proofs, the work of [MPV24] shows that the original Sahai-Waters scheme (based on $i\mathcal{O}$ and one-way functions) [SW14] is also adaptively sound. In conjunction with constructions of $i\mathcal{O}$ from falsifiable cryptographic assumptions [JLS21,JLS22], these works provide the first adaptively-sound SNARGs for NP from falsifiable assumptions.

**Batch arguments.** Existing constructions of adaptively-sound SNARGs for NP focus on the single-statement setting where the prover constructs a proof for a single statement. In many settings (e.g., incrementally verifiable computation [Val08] or proof-carrying data [CT10]), a prover might have a batch of $T$ (possibly correlated) statements $x_1, \ldots, x_T$ that it wants to prove to the verifier, and the goal is to construct a single short proof (whose size is sublinear in $T$ *and* in the size of the associated NP relation) of all $T$ statements. There are two main approaches to constructing batch arguments:

- **Using BARGs for NP:** Non-interactive batch arguments (BARGs) for NP [KPY19, CJJ21, KVZ21, CJJ22] provide one possible approach. Namely, a BARG for NP allows a prover to prepare a proof on $T$ statements with a proof whose size scales sublinearly (ideally, polylogarithmically) with the number of statements $T$. Moreover, many recent works have shown how to construct BARGs for NP from a broad range of cryptographic assumptions [CJJ21, KVZ21, CJJ22, WW22, HJKS22, DGKV22, PP22, CGJ+23, KLV23, KLVW23]. However, in these existing constructions, the size of the proof grows with the size of the circuit that decides a *single* statement, and the goal is to amortize the proof size across the number of statements. Allowing the proof size to grow with the size of the NP relation avoids black-box separations that pertain to SNARGs for NP [GW11]. In this work, we are interested in batching SNARG proofs, where the size of the proof is sublinear in both the number of statements *and* size of the circuit computing the NP relation; such arguments are said to be *fully* succinct [GSWW22]. The previous work of [GSWW22] showed how to construct fully succinct BARGs for NP using $i\mathcal{O}$ and one-way functions, but the construction only achieved non-adaptive soundness.

- **Using SNARGs for NP:** Another approach to constructing a fully succinct SNARG for a batch language is to view the batch statement $(x_1, \ldots, x_T)$ as a *single* NP statement for a product language (i.e., the statement $(x_1, \ldots, x_T)$ is in the language if for each $i \in [T]$, there exists a valid witness $w_i$ for $x_i$), and then use a SNARG for NP to prove the product language. This approach achieves adaptive soundness if we instantiate the underlying SNARG with an adaptively-sound SNARG for NP [WW24a, WZ24]. However, the size of the CRS in existing adaptively-sound SNARGs [WW24a, WZ24] grows polynomially with the size of the NP relation circuit. Thus, if we directly apply an existing adaptively-sound SNARG for NP to a batch language, the NP relation circuit would take all $T$ statements as input, and the size of the CRS scales polynomially with $T$. This means the CRS is large and moreover, there is an *a priori* bound on the number of statements that can be batched. In this work, our goal is to support aggregating an *arbitrary* polynomial number of (adaptively-sound) proofs on NP statements.

[1] A subsequent work [WW24b] also shows how to construct an adaptively-sound SNARG using $i\mathcal{O}$ and sub-exponentially-secure one-way functions without any additional algebraic assumptions.

**Why not compose?** If we settle for non-adaptive soundness, the work of [GSWW22] shows that we can construct a fully succinct SNARG for batch languages by composing a standard (somewhere-extractable) BARG for NP with a SNARG for NP. Namely, a proof on statements $(x_1, \ldots, x_T)$ is a BARG proof that there exists SNARG proofs $\pi_1, \ldots, \pi_T$ for the statements $x_1, \ldots, x_T$. In this case, the NP relation associated with the BARG is the SNARG verification circuit, which is small by construction. Moreover, if the BARG is *somewhere extractable* [CJJ22][2] and the SNARG is non-adaptively sound, then it is straightforward to show that the composed scheme satisfies non-adaptive soundness. While we can replace the underlying SNARG in this composition with an adaptively-sound construction, we are not able to prove adaptive soundness for the composition. The issue is that if we rely on somewhere extractability for the BARG, then the reduction needs to "know" the index of the false statement and program it into the CRS; this is not possible when the statements are adaptively chosen.

Alternatively, we could consider a reduction algorithm that guesses the index of the false statement. Since the index is computationally hidden from the malicious prover, the hope would be that a prover that consistently chooses statements $(x_1, \ldots, x_T)$ that evades the guess (i.e., where the index of the false statement is different from the guessed index) must be breaking index hiding of the somewhere extractable BARG. The problem is that checking whether the adversary successfully evaded the guess (and thus, broke index hiding) is not an efficient procedure (it requires deciding the underlying NP statement). We could handle this by complexity leveraging and relying on a super-polynomial time reduction that is able to decide the underlying NP relation. However, if we do so, then the size of the resulting BARG starts scaling with the size of the NP relation, and the resulting construction is no longer succinct.

**This work.** In this work, we show how to construct adaptively-sound SNARGs for batch languages with almost no overhead compared to the single-statement setting. Specifically, we show how to leverage the adaptively-sound SNARG for NP from [WW24a] to obtain an adaptively-sound SNARG for batch languages with only polylogarithmic additive overhead in the number of statements $T$. We summarize our instantiation in the following (informal) theorem:

**Theorem 1.1** (Informal)**.** *Let $\lambda$ be a security parameter. Assuming (1) the polynomial hardness of computing discrete logs in a prime-order group, (2) the existence of a sub-exponentially-secure indistinguishability obfuscation scheme for Boolean circuits, and (3) the existence of a sub-exponentially-secure one-way function, there exists an adaptively-sound SNARG for batch NP with the following properties:*

- **Preprocessing SNARG:** *Let $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$ be the circuit that computes the NP relation (where $n$ is the statement size and $v$ is the witness size). The size of the common reference string for proving up to $T \le 2^\lambda$ statements is* $\mathsf{poly}(\lambda + |C|)$.
- **Proof size:** *A proof on a batch of $T \le 2^\lambda$ statements $(x_1, \ldots, x_T)$ has size* $\mathsf{poly}(\lambda)$.

*Additionally, the SNARG is perfect zero-knowledge.*

**The Gentry-Wichs separation.** The classic result of Gentry and Wichs [GW11] gives a barrier for constructing adaptively-sound SNARGs for NP from falsifiable assumptions where the running time of the reduction is insufficient to decide the underlying NP language. Consequently, existing constructions of

[2] A BARG is somewhere extractable if the CRS can be programmed on a (hidden) index $i \in [T]$. Then, given a valid BARG proof $\pi$ on a batch of statements $(x_1, \ldots, x_T)$, there is an efficient extraction algorithm that recovers a witness $w_i$ for $x_i$. The special index $i$ is computationally hidden by the CRS. Somewhere extractable BARGs can be constructed from most number-theoretic assumptions [CJJ21, KVZ21, CJJ22, WW22, HJKS22, DGKV22, PP22, CGJ+23, KLV23, KLVW23].

adaptively-sound SNARGs for NP [WW24a, WZ24, MPV24] all rely on complexity leveraging and super-polynomial-time security reductions. In these constructions, the cost of the complexity leveraging is incurred in the size of the CRS. In the setting of batch NP, the time it takes to decide a batch of $T$ statements $(x_1, \ldots, x_T)$ is only a factor of $T$ greater than the time it takes to decide a single statement. As such, obtaining an adaptively-sound SNARG for batch NP would only increase the running time of the reduction algorithm by a factor of $T$. In this case, the size of the CRS (or the proof) would only need to increase by a factor of $\log T$. In contrast, for a general NP relation where the statements and witnesses are a factor of $T$ longer, the reduction may have to run in time that is greater by a factor $2^T$ to decide the larger language, which would lead to a CRS that is larger by a factor of $\mathsf{poly}(T)$ rather than $\mathsf{poly}(\log T)$.

### 1.1 Technical Overview

We begin by describing the Waters-Wu [WW24a] adaptively-sound SNARG for NP based on indistinguishability obfuscation ($i\mathcal{O}$) and re-randomizable one-way functions. Throughout, we consider the language of Boolean circuit satisfiability, where the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$ is fixed ahead of time (i.e., part of the CRS). A statement $x \in \{0,1\}^n$ is true if there exists a witness $w \in \{0,1\}^v$ such that $C(x,w) = 1$.

**Building blocks.** In addition to $i\mathcal{O}$, the [WW24a] construction requires a puncturable pseudorandom function (PRF) [BW13, KPTZ13, BGI14], and a re-randomizable one-way function:

- In a puncturable PRF $\mathsf{F}(k,\cdot)$, the holder of the secret key $k$ can "puncture" the key at an input point $x^*$ to create a punctured key $k^{(x^*)}$. The punctured key $k^{(x^*)}$ can be used to evaluate $\mathsf{F}(k,x)$ on all points $x \neq x^*$. However, the value $\mathsf{F}(k,x^*)$ at the punctured point remains pseudorandom even given the punctured key $k^{(x^*)}$.
- The second ingredient they require is a re-randomizable one-way function (OWF) $f$. This is a OWF equipped with a *statistical* re-randomization algorithm that takes as input a OWF challenge $y_{\mathsf{base}}$ and produces a fresh challenge $y$ (sampled uniformly at random from the challenge space of the OWF). Moreover, given the re-randomization randomness together with a solution to the re-randomized statement, there is an efficient algorithm for recovering a solution to the original OWF challenge $y_{\mathsf{base}}$. In other words, the re-randomization can be viewed as a (perfect) random self-reducibility property on the OWF.

**The Waters-Wu construction.** In the Waters-Wu construction, the CRS consists of two obfuscated programs: (1) a "solution-generator" program GenSol used to construct proofs; and (2) a "challenge-generator" program GenChall used to verify proofs. The solution-generator GenSol has the circuit $C$ (for the NP relation) together with three puncturable PRF keys $k_{\mathsf{sel}}, k_0, k_1$ hard-wired inside.

The solution-generator program takes as input a bit $b \in \{0,1\}$, a statement $x$, and a witness $w$. It checks that $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$ and $C(x,w) = 1$. If so, it outputs the solution $\mathsf{F}(k_b, x)$; the proof is the pair $(b, \mathsf{F}(k_b,x))$. Next, the challenge-generator program takes as input a bit $b$ and a statement $x$ and outputs the challenge $y_b = f(\mathsf{F}(k_b,x))$. To verify a proof $\pi = (b,z)$ on a statement $x$, the verification algorithm first runs the challenge-generator program on input $(b,x)$ to obtain a challenge $y$. Then it checks that $f(z) = y$.

The idea is that the solution-generator program only outputs *one* of the two possible solutions associated with each statement $x$. Moreover, which one it chooses is determined pseudorandomly by evaluating the selector PRF $\mathsf{F}(k_{\mathsf{sel}}, x)$. We will refer to the challenge $y_b$ associated with $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ as the "on-path" challenge for $x$ and the challenge $y_b$ associated with $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$ as the "off-path" challenge for $x$.

In the Waters-Wu construction, the GenSol program is constructed so it only provides solutions to the off-path challenge and never generates a solution to an on-path challenge. Then, in the proof of adaptive soundness, [WW24a] show how to replace the on-path challenge for *every* statement with a re-randomized challenge of a one-way function. The hope is that if the malicious prover ever produces a proof for a false statement $x$ that corresponds to the on-path challenge, then it successfully breaks the one-way function. Finally, the [WW24a] analysis appeals to the fact that for a false statement $x$, the value of the selector PRF $\mathsf{F}(k_{\mathsf{sel}}, x)$ is computationally unpredictable to the adversary; as such, with probability close to $1/2$, the prover provides a solution to the on-path challenge, which completes the adaptive soundness analysis. We now give the formal description of the GenSol and GenChall programs:[3]

| $\mathsf{GenSol}(b, x, w)$ | $\mathsf{GenChall}(b, x)$ |
|---|---|
| • If $C(x, w) = 0$, output $\perp$.<br>• If $b = \mathsf{F}(k_{\mathsf{sel}}, x)$, output $\perp$.<br>• Output $z = \mathsf{F}(k_b, x)$. | • Output $y = f(\mathsf{F}(k_b, x))$. |

To construct a proof for a statement $x$ and witness $w$, the prover simply runs the (obfuscated) GenSol program on input $(0, x, w)$ and input $(1, x, w)$. GenSol will output $\perp$ on one of these inputs, and an OWF preimage $z = \mathsf{F}(k_b, x)$ on the other. The proof $\pi = (b, z)$ consists of the bit $b$ and the preimage $z$. To check the proof $\pi$, the verifier simply runs the (obfuscated) GenChall program on input $(b, x)$. GenChall will output a OWF challenge $y = f(\mathsf{F}(k_b, x))$, and the verifier checks that $f(z) = y$.

We now sketch the proof of soundness from [WW24a]. As mentioned above, the proof proceeds in a sequence of hybrid experiments. First, they argue that with probability $1/2$, the malicious prover will output an on-path solution as its proof; this is because for a false statement $x$, it is unable to predict the value of $\mathsf{F}(k_{\mathsf{sel}}, x)$. Next, they gradually replace the on-path challenge for every statement program with a re-randomized one-way function challenge. This way, a solution to *any* on-path challenge implies a solution to the original one-way function challenge. Since the GenSol program never outputs an on-path solution, this does not affect completeness. However, if the prover ever produces an on-path solution, then it successfully inverts the one-way function and adaptive soundness follows. We now sketch the sequence of hybrids from [WW24a]:

- $\mathsf{Hyb}_0$: This is the real adaptive soundness game. The challenger outputs 1 only if the adversary $\mathcal{A}$ produces an accepting proof $\pi = (b, z)$ for a false statement $x$: namely, $f(z) = y = \mathsf{GenChall}(b, x)$.
- $\mathsf{Hyb}_1$: After the adversary $\mathcal{A}$ outputs its proof $\pi = (b, z)$, the challenger additionally checks that $b = \mathsf{F}(k_{\mathsf{sel}}, x)$, or in other words, that $\mathcal{A}$ output a solution to the on-path challenge. This can only reduce $\mathcal{A}$'s success probability by a factor of 2, since the value of $\mathsf{F}(k_{\mathsf{sel}}, x)$ is computationally hidden from the adversary for every false statement $x$ (by puncturing security). Formally, [WW24a] show this by considering an exponential sequence of hybrids, one for each false statement $x^*$. In $\mathsf{Hyb}_1^{(x^*)}$, the challenger punctures $k_{\mathsf{sel}}$ at $x^*$ and hard-wires the punctured key $k_{\mathsf{sel}}^{(x^*)}$ in GenSol instead of $k_{\mathsf{sel}}$:

[3] Note that the original Waters-Wu construction did not require GenSol and GenChall to take the bit $b \in \{0, 1\}$ as input. Instead, GenSol computed $b = \mathsf{F}(k_{\mathsf{sel}}, x)$ and outputted $z = \mathsf{F}(k_b, x)$ while GenChall outputted $f(\mathsf{F}(k_0, x))$ and $f(\mathsf{F}(k_1, x))$. The adaptation here is equivalent to the original Waters-Wu construction and the updated syntax will be conducive when extending to batch NP.

| $\mathsf{GenSol}^{(x^*)}(b, x, w)$ | $\mathsf{GenChall}(b, x)$ |
|---|---|
| – If $C(x, w) = 0$, output $\perp$.<br>– If $b = \mathsf{F}(k_{\mathsf{sel}}^{(x^*)}, x)$, output $\perp$.<br>– Output $z = \mathsf{F}(k_b, x)$. | – Output $y = f(\mathsf{F}(k_b, x))$. |

When $x^*$ is a false statement, $\mathsf{GenSol}^{(x^*)}$ still computes the same functionality as GenSol: both immediately reject, since there does not exist a $w$ such that $C(x^*, w) = 1$. Thus, $\mathsf{GenSol}^{(x^*)}$ does not need to evaluate $\mathsf{F}(k_{\mathsf{sel}}, x^*)$. Now, by puncturing security, the value of $\mathsf{F}(k_{\mathsf{sel}}, x^*)$ is pseudorandom even given $k_{\mathsf{sel}}^{(x^*)}$. Thus, if the adversary outputs a proof $\pi = (b, z)$ for $x^*$, with probability $1/2 - \mathsf{negl}(\lambda)$, it will be the case that $\mathsf{F}(k_{\mathsf{sel}}, x^*) = b$.

- $\mathsf{Hyb}_2$: In this experiment, the challenger stops checking whether or not $x$ is false; observe that this can only increase the adversary's success probability. In addition, the challenger samples a random OWF challenge $y_{\mathsf{base}} \leftarrow f(r)$ for uniform $r$ along with a puncturable PRF key $k_{\mathsf{rerand}}$ that will be used to re-randomize $y_{\mathsf{base}}$. The challenger now modifies GenChall to output a re-randomization of $y_{\mathsf{base}}$ on $(b, x)$ whenever $b = \mathsf{F}(k_{\mathsf{sel}}, x)$. In other words, the on-path challenges are now replaced by a re-randomized instance of $y_{\mathsf{base}}$. To argue that this is computationally indistinguishable from the previous hybrid, the [WW24a] reduction again steps through an exponential number of hybrids, one for each statement $x^*$. Planting the re-randomized challenge is then an exercise in punctured programming [SW14]. The key observation is that the GenSol program never evaluates $\mathsf{F}(k_b, x^*)$ for $b = \mathsf{F}(k_{\mathsf{sel}}, x^*)$. We can then appeal to punctured pseudorandomness of $\mathsf{F}(k_b, x^*)$ to conclude that the challenge $y_b$ is computationally indistinguishable from a fresh one-way function challenge, which is in turn statistically indistinguishable from a re-randomized instance.

In $\mathsf{Hyb}_2$, algorithm $\mathcal{A}$ can only succeed if it provides a solution to a re-randomized one-way function instance. But this means that $\mathcal{A}$ also inverts the original one-way function challenge, which completes the proof of adaptive security. Observe that here, polynomial security of the one-way function already suffices. Importantly, this final step is the only step in the analysis that relies on one-wayness. Thus, the proof $\pi$ remains succinct despite the use of an exponential number of hybrids in the previous steps. The exponential sequence of hybrids require blowing up the security parameters for the $i\mathcal{O}$ and puncturable PRF schemes, but this only affects the length of the CRS and *not* the proof.

### 1.1.1 Batching SNARGs Using Homomorphic One-Way Functions

We now show how to extend the Waters-Wu scheme to the batch setting. Recall that in this setting, the prover has a collection of $T$ statements $x_1, \ldots, x_T$ and its goal is to prove that all $T$ statements are true. If we directly modify the GenSol and GenChall programs above to take in all $T$ statements, then the resulting CRS would have size that scales linearly with $T$, and moreover, the scheme would only support an a priori bounded number of statements. Our goal is to obtain a construction without this limitation. Our first approach relies on a homomorphic re-randomizable one-way function while our second approach (see Section 1.1.2) uses a chaining-based approach that does not rely on any homomorphic properties on the re-randomizable one-way function.

**Homomorphic re-randomizable one-way functions.** As described above, the Waters-Wu construction [WW24a] uses a re-randomizable one-way function. Specifically, they show two instantiations of the

re-randomizable one-way function: the first is based on the hardness of discrete log while the second is based on factoring. In this work, we will consider the construction based on discrete log. To recall, let $\mathbb{G}$ be a group of prime-order $p$ and let $g$ be a generator of $\mathbb{G}$. The one-way function $f : \mathbb{Z}_p \to \mathbb{G}$ is then defined to be the mapping $z \mapsto g^z$. The re-randomizable algorithm takes an instance $y = g^z$ and samples a random $r \xleftarrow{R} \mathbb{Z}_p$ and outputs $y \cdot g^r = g^{z+r}$. Our first observation is that this one-way function is homomorphic:

$$f(z_1 + z_2) = g^{z_1+z_2} = g^{z_1} \cdot g^{z_2} = f(z_1) \cdot f(z_2).$$

In the context of the Waters-Wu SNARG, the values $z$ would correspond to the preimages in the proof $\pi$. Suppose now that we have $T$ proofs $(b_1, z_1), \ldots, (b_T, z_T)$ on $T$ different statements $x_1, \ldots, x_T$. Then a natural approach to obtain a batch proof on all $T$ statements is to compute $z = \sum_{i \in [T]} z_i \in \mathbb{Z}_p$. Then,

$$f(z) = f\left(\sum_{i \in [T]} z_i\right) = \prod_{i \in [T]} f(z_i) = \prod_{i \in [T]} y_{i,b_i},$$

where $y_{i,b_i} = \mathsf{GenChall}(b_i, x_i)$ is the challenge bit associated with statement $i$. Now, if the verifier knew the bits $b_1, \ldots, b_T$, it can compute $y_{i,b_i} = \mathsf{GenChall}(b_i, x_i)$ and then $y = \prod_{i \in [T]} y_{i,b_i} \in \mathsf{G}$. Then, the verification algorithm would simply boil down to checking that $y = f(z)$. In this case, the prover just needs to provide the *aggregated* preimage $z$ rather than the individual preimages $(z_1, \ldots, z_T)$. The problem with this basic approach is that the verifier does *not* know the individual bits $b_i \in \{0, 1\}$. While the prover can certainly include the bits $b_i$ for each statement as part of the proof, this means the size of the proof is now $T + \mathsf{poly}(\lambda)$, which no longer meets our succinctness requirement. Note that if $T = O(\log \lambda)$, the verifier can try all the possible values for $b_1, \ldots, b_T$, but this approach does not work for general $T$.

**Using a large alphabet.** We solve this problem by increasing the alphabet size. Namely, instead of having two challenges, suppose instead we had $T + 1$ challenges (i.e., the selector PRF $\mathsf{F}(k_{\mathsf{sel}}, \cdot)$ now outputs an element of the set $\{1, 2, \ldots, T + 1\}$) and correspondingly, $T + 1$ PRF keys $k_1, \ldots, k_{T+1}$ used to generate the challenges. In the batch setting, the on-path challenge is a function of *both* the statement $x_i$ and the index $i \in [T]$ (i.e., the $j^{\text{th}}$ challenge is $z = \mathsf{PRF}(k_j, (x_i, i))$). For each statement-index pair $(x_i, i)$, there is a single "on-path" challenge index $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i)) \in [T + 1]$ for which the GenSol program will not provide a preimage and $T$ off-path challenges for which the GenSol program will provide preimages (given a valid witness for $x_i$). This means that for any batch of $T$ statements $\vec{x} = (x_1, \ldots, x_T)$, there always exists some index $j \in [T + 1]$ for which $j \neq \mathsf{F}(k_{\mathsf{sel}}, (i, x_i))$ for all $i \in [T]$. Since the same index $j$ can now be *shared* across all $T$ statements, the prover only needs to communicate the single index (of length $O(\log T)$) as part of its proof. Concretely, the programs in the CRS (where the re-randomizable one-way function is instantiated with the discrete log construction) are now defined as follows:

| $\mathsf{GenSol}(i, j, x_i, w_i)$ | $\mathsf{GenChall}(i, j, x_i)$ |
|---|---|
| • If $C(x_i, w_i) = 0$, output $\perp$.<br>• If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.<br>• Output $z = \mathsf{F}(k_j, (x_i, i))$. | • Output $y = g^{\mathsf{F}(k_j, (x_i, i))}$. |

Our scheme now operates as follows:

- **Proof generation:** To construct a proof on $x_1, \ldots, x_T$ (using witnesses $w_1, \ldots, w_T$), the prover first finds an index $j \in [T + 1]$ where $z_i = \mathsf{GenSol}(i, j, x_i, w_i) \neq \perp$ for all $i \in [T]$. Then it computes the aggregated proof $z = \sum_{i \in [T]} z_i$ and outputs the proof $\pi = (j, z)$.

- **Proof verification:** To verify the proof, the verifier computes the challenge $y_i = \mathsf{GenChall}(i, j, x_i)$ for each $i \in [T]$ and then computes the aggregated challenge $y = \prod_{i \in [T]} y_i$. Finally, the verifier checks that $g^z = y$.

As written, the GenSol and GenChall programs would require us to hard-wire all $T+1$ PRF keys $k_1, \ldots, k_{T+1}$ into the GenSol and GenChall programs. Consequently, the size of the CRS now grows with $T$, which is no better than directly applying [WW24a] to the batch language. To get around this, we derive the keys $k_j$ for $j \in [T+1]$ from another (puncturable) PRF. The modified programs are defined as follows:

| $\mathsf{GenSol}(i, j, x_i, w_i)$ | $\mathsf{GenChall}(i, j, x_i)$ |
|---|---|
| • If $C(x_i, w_i) = 0$, output $\perp$.<br>• If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.<br>• Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>• Output $z = \mathsf{F}(k_j, (x_i, i))$. | • Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>• Output $y = g^{\mathsf{F}(k_j, (x_i, i))}$. |

To argue adaptive soundness, we adopt a strategy similar to that used in [WW24a]:

- We start by arguing that with non-negligible probability, the adaptive soundness adversary outputs a tuple of statements $\vec{x} = (x_1, \ldots, x_T)$ and an accepting proof $\pi = (j, z)$ where $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$ for some $i^* \in [T]$. In other words, the adversary gives a proof for an on-path challenge. This step is the analog of the transition between $\mathsf{Hyb}_0$ and $\mathsf{Hyb}_1$ in the above sketch of the [WW24a] reduction.

  This argument relies on the fact that if $x_{i^*}$ is a false instance, the value of $\mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$ is computationally indistinguishable from a random index in $[T+1]$. Thus, whenever the adversary outputs a tuple $\vec{x} = (x_1, \ldots, x_T)$ that contains a false instance $x_{i^*}$ together with a proof $\pi = (j, z)$, then with probability roughly $1/(T+1)$, the adversary's index $j$ satisfies $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$. The formal argument relies on $i\mathcal{O}$ security and security of the puncturable PRF.

  If the adversary breaks adaptive soundness with advantage $\varepsilon$, then the above argument shows that with probability roughly $\varepsilon/(T+1)$, the adversary outputs $\vec{x}$ and $\pi = (j, z)$ where $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$ for some $i^* \in [T]$. We can now move to an experiment where the challenger guesses the index $i^* \xleftarrow{R} [T]$ and the adversary is only considered successful if it breaks adaptive soundness *and* moreover, the index $j$ it outputs satisfies $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$. Over the randomness of $i^*$, we conclude that the adversary will win this modified experiment with probability $\varepsilon/(T(T+1))$, which remains non-negligible since the number of instances $T$ is always polynomially-bounded.

- At this point, the adversary wins only if it outputs a proof $\pi = (j, z)$ where $j = \mathsf{PRF}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$. Similar to the analysis in [WW24a], the reduction algorithm now modifies the GenChall program to output a re-randomized one-way function challenge (derived from a single base instance $y_{\mathsf{base}}$) as the on-path challenge for *every* statement $x^*$ at index $i^*$:

| $\mathsf{GenSol}(i, j, x_i, w_i)$ | $\mathsf{GenChall}(i, j, x_i)$ |
|---|---|
| – If $C(x_i, w_i) = 0$, output $\perp$.<br>– If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.<br>– Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>– Output $z = \mathsf{F}(k_j, (x_i, i))$. | – Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>– If $i = i^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $y = y_{\mathsf{base}} \cdot g^{\mathsf{F}(k_j, (x_i, i))}$.<br>– Otherwise, output $y = g^{\mathsf{F}(k_j, (x_i, i))}$. |

This is the analog of the transition between $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_2$ in the above sketch of the [WW24a] reduction and relies on the fact that for all instances $x^*$, the GenSol program never needs to evaluate $\mathsf{F}(k_j, (x^*, i^*))$ where $j = \mathsf{F}(k_{\mathsf{sel}}, (x^*, i^*))$. Thus, we can rely on $i\mathcal{O}$ security and punctured pseudorandomness to replace the challenge $g^{\mathsf{PRF}(k_j,(x^*,i^*))}$ with a re-randomized instance $y_{\mathsf{base}} \cdot g^{\mathsf{PRF}(k_j,(x^*,i^*))}$.

Now, suppose the adversary is successful in the final experiment: namely, the adversary constructs a valid proof $\pi = (j, z)$ on $\vec{x} = (x_1, \ldots, x_T)$ and $j = \mathsf{PRF}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$ is the on-path challenge index. We claim that we can use any such adversary to solve the discrete log problem. Let $(g, h)$ where $h = g^\alpha$ be the discrete log challenge. The reduction algorithm samples all of the PRF keys itself and uses the discrete log challenge $h$ as the base instance $y_{\mathsf{base}} = h$ that it embeds into GenChall. Suppose the adversary constructs a valid proof $\pi = (j, z)$ on $\vec{x} = (x_1, \ldots, x_T)$ and $j = \mathsf{PRF}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$. Then, the following hold:

- For each $i \in [T] \setminus \{i^*\}$, $\mathsf{GenChall}(i, j, x_i)$ outputs $y_i = g^{\mathsf{F}(k_j,(x_i,i))}$.
- Since $j = \mathsf{PRF}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$, $\mathsf{GenChall}(i^*, j, x_i)$ outputs the re-randomized instance $y_{i^*} = h \cdot g^{\mathsf{F}(k_j,(x_{i^*},i^*))}$.
- If $z \in \mathbb{Z}_p$ is a valid proof, then it must be the case that

$$g^z = \prod_{i\in[T]} y_i = h \cdot g^{\sum_{i\in[T]} \mathsf{F}(k_j,(x_i,i))}.$$

Writing $h = g^\alpha$, this means

$$\alpha = z - \sum_{i\in[T]} F(k_j, (x_i, i)). \tag{1.1}$$

If the adversary outputs a valid proof $z$, then the reduction algorithm is able to recover the discrete log $\alpha$ of the challenge instance $h$, which complete the proof. Note that the reduction algorithm chose all of the PRF keys, so it is able to compute all terms in Eq. (1.1). We give the formal description and analysis of this scheme in Section 4. There, we describe our construction with respect to an arbitrary homomorphic re-randomizable one-way function (as opposed to just the discrete log version illustrated above).

**Parameter sizes.** The size of the programs in the CRS in the above construction is $\mathsf{poly}(\lambda + |C| + \log T)$. Thus, setting $T = 2^\lambda$ allows us to support any a priori unbounded polynomial number of statements. This gives the first adaptively-sound SNARG for batch NP (that supports an unbounded number of statements) with full succinctness from standard falsifiable assumptions.

### 1.1.2 Batching SNARGs via a Chaining Approach and Re-randomizable PRGs

Thus far, we have demonstrated how to extend the Waters-Wu SNARG to support batching by relying on the homomorphic structure of the one-way function. In this work, we also give a second approach to support batching that does not assume any homomorphic properties on the output SNARG. Instead, our construction relies on a re-randomizable pseudorandom generator (PRG).

**The chaining template from [GSWW22].** We follow a similar template as the general aggregation approach from [GSWW22]. The work of [GSWW22] constructs a non-adaptively-sound SNARG for batch NP by adapting the non-adaptively-sound SNARG for NP by Sahai and Waters [SW14]. Specifically, they describe a "chaining" approach where the prover program (in the CRS) takes as input a hash digest dig of the statements $(x_1, \ldots, x_T)$, a proof $\pi_{i-1}$ on the first $i-1$ statements, the next statement $x_i$, and an associated

witness $w_i$, together with an opening of $x_i$ with respect to the digest dig. The prover checks that $\pi_{i-1}$ is a valid proof on the digest dig, that dig opens to $x_i$ at position $i$, and that $w_i$ is a valid witness for $x_i$. If all of these properties hold, then the program outputs a proof for the first $i$ statements (with respect to dig).

To prove non-adaptive soundness, the idea in [GSWW22] is to first identify the index $i \in [T]$ of a *false* statement, and use punctured programming to argue that there does not *exist* any accepting proofs on the first $i$ statements. This step relies on the fact that there are no witnesses for the false statement $x_i$. Then, they show that if there does not exist an accepting proof for index $i$, there also does not exist an accepting proof for index $i+1$. This proceeds until the final hybrid where they argue that there does not exist any proof for index $T$, at which point non-adaptive soundness holds.

**The challenge with adaptive soundness.** Unlike the single-statement setting, in the chaining approach, it is no longer sufficient to argue that an accepting proof of a false statement is computationally hard to find. This is because the obfuscated prover program (i.e., the analog of GenSol) is first checking the proof on the first $i-1$ statements when deciding whether to generate a proof for the first $i$ statements or not. If there exists a valid proof on the first $i-1$ statements, then this program does not output $\perp$ on all inputs with index $i$ (e.g., consider the setting where statement $x_i$ is true). As a result, we cannot argue that there does not exist a proof on the first $i$ statements. In contrast, if we can argue that there are no accepting proofs on the first $i-1$ statements, then we can leverage $i\mathcal{O}$ security to argue that there are also no accepting proofs on the first $i$ statements (since the obfuscated prover program never accepts a proof on the first $i-1$ statements, it would never output a proof for the first $i$ statements).

In the Waters-Wu approach, they showed that if an adversary could construct a proof of a false statement, then the adversary can also invert the one-way function. Notably, this is a *computational* property, and the previous analysis can only rule out an adversary finding an accepting proof efficiently. Consequently, this is insufficient to implement the chaining approach from [GSWW22] as proofs of *false* statements do exist (but are hard to find). The work of [GSWW22] leverages an (expanding) pseudorandom generator to check the proofs instead of using a one-way function precisely to move to a hybrid where proofs on false statements no longer exist.

**Re-randomizable PRGs.** In Section 6, we show how to use a similar chaining strategy together with the Waters-Wu approach to obtain an adaptively-sound SNARG for batch NP. For the reasons outlined above, our approach requires a way to rule out the *existence* of proofs on false statements. To do so, we rely on the stronger notion of a re-randomizable PRG instead of a re-randomizable OWF. In a re-randomizable PRG $\mathsf{G}\colon \{0,1\}^\lambda \to \{0,1\}^t$, there is an algorithm that takes a string $y_{\mathsf{base}} \in \{0,1\}^t$ and re-randomizes it to a new string $y \in \{0,1\}^t$ with the following properties:

- If $y_{\mathsf{base}}$ is the in the image of the PRG (i.e., $y_{\mathsf{base}} = \mathsf{PRG}(s)$ for some $s \in \{0,1\}^\lambda$), then the re-randomized value $y$ is distributed according to $\mathsf{G}(s)$ for a fresh seed $s \xleftarrow{R} \{0,1\}^\lambda$.
- If $y_{\mathsf{base}}$ is not in the image of the PRG, then the re-randomized value $y$ is distributed according to a random value $y \xleftarrow{R} \{0,1\}^t \setminus \{\mathsf{PRG}(s) : s \in \{0,1\}^\lambda\}$.

We can construct a re-randomizable PRF from the decisional Diffie-Hellman (DDH) assumption. In particular, we work over a group $\mathbb{G}$ of prime order $p$ and generator $g$, and define the public parameters to be $(g, h)$ where $h \xleftarrow{R} \mathbb{G}$. Then, we define the generator $\mathsf{G}\colon \mathbb{Z}_p^* \to \mathbb{G} \times \mathbb{G}$ as the mapping $x \mapsto (g^x, h^x)$. Pseudorandomness follows directly from the DDH assumption, and the re-randomization follows via the DDH random self-reduction that maps $(u, v) \mapsto (u^r, v^r)$ where $r \xleftarrow{R} \mathbb{Z}_p^*$.

**The chaining approach using re-randomizable PRGs.** In our chaining-based approach for aggregating adaptively-sound SNARGs, we replace the GenSol and GenChall programs from [WW24a] with a proof aggregation program AggProof and a proof verification program VerProof with the following syntax:

- The proof aggregation program AggProof outputs a proof for a digest dig and index $i$ if it is given a valid proof on dig and index $i-1$, a valid statement-witness pair $(x_i, w_i)$, and a proof that dig opens to statement $x_i$ at index $i$.
- The proof verification program VerProof only accepts a proof $(j, z)$ for the digest dig and index $i$ if $\mathsf{G}(z) = \mathsf{G}(\mathsf{F}(k_j, (\mathsf{dig}, i)))$. Importantly, note that the proof just consists of the index $j$ and the solution $z$; it does *not* contain the digest dig (which is at least as long as a single statement). The verifier computes the digest dig from the statements at verification time.

Importantly, we have replaced the re-randomizable one-way function $f$ with a re-randomizable PRG. If we instantiate this template with the adaptively-sound construction with $T+1$ challenges, the CRS consists of obfuscations of the following programs:

| $\mathsf{AggProof}(i, j, \mathsf{dig}, x, w, \sigma, z_{i-1})$ | $\mathsf{VerProof}(i, j, \mathsf{dig}, z_i)$ |
|---|---|
| • If $C(x, w) = 0$, output $\perp$.<br>• If $\sigma$ does not open dig to $x$ at index $i$, output $\perp$.<br>• If $j = \mathsf{F}(k_{\mathsf{sel}}, (x, i))$, output $\perp$.<br>• If $i \neq 1$ and $\mathsf{VerProof}(j, \mathsf{dig}, i-1, z_{i-1}) = 0$, output $\perp$.<br>• Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>• Output $z = \mathsf{F}(k_j, (\mathsf{dig}, i))$. | • Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>• If $\mathsf{G}(z_i) = \mathsf{G}(\mathsf{F}(k_j, (\mathsf{dig}, i)))$, output 1. Otherwise, output 0. |

To construct a proof on statements $\vec{x} = (x_1, \ldots, x_T)$ with associated witnesses $(w_1, \ldots, w_T)$, the prover proceeds as follows:

- It start by computing a digest dig of the statements $(x_1, \ldots, x_T)$. Let $\sigma_1, \ldots, \sigma_T$ be the associated local openings of dig to the statements $x_1, \ldots, x_T$, respectively.
- For each $j \in [T+1]$, the prover initializes $z_0 = \perp$. Then, for each $i = 1, \ldots, T$, it computes $z_i \leftarrow \mathsf{AggProof}(i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$. If $z_1, \ldots, z_T \neq \perp$, then the prover outputs the proof $(j, z_T)$. Otherwise, it retries with the next value of $j$.

For each statement-index pair $(x_i, i)$, there is exactly one index $j$ where AggProof outputs $\perp$. Thus, there exists at least one index $j \in [T+1]$ where the prover succeeds in constructing a proof $(j, z_T)$. To check a proof $\pi = (j, z)$ for a batch of statements $\vec{x} = (x_1, \ldots, x_T)$, the verifier computes the digest dig of $(x_1, \ldots, x_T)$ and outputs $\mathsf{VerProof}(T, j, \mathsf{dig}, z)$.

**Arguing adaptive soundness.** We now give a sketch of the adaptive soundness proof:

- Suppose we have an adversary $\mathcal{A}$ that can break adaptive soundness with advantage $\varepsilon$. This means that $\mathcal{A}$ outputs a vector of statements $\vec{x} = (x_1, \ldots, x_T)$ and a proof $\pi = (j, z)$ such that $\pi$ is a valid proof and moreover, there exists some index $i^* \in [T]$ where $x_{i^*}$ is a false instance.
- Our proof of adaptive soundness steps through a sequence of hybrid experiments. In the initial sequence, the challenger starts by sampling an index $i^* \xleftarrow{R} [T]$ and we declare the adversary

to be successful only if it outputs a valid proof $\pi = (j, z)$ on a vector of statements $\vec{x}$ where $j = \mathsf{PRF}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$. As in the construction from Section 1.1.1, this corresponds to a proof for an "on-path" challenge. To argue this, we use the fact that if $x_{i^*}$ is a false statement, then the AggProof program never needs to compute $\mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$. Correspondingly, we can rely on puncturing security of $\mathsf{F}(k_{\mathsf{sel}}, \cdot)$ to conclude that the value of $\mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$ is pseudorandom from the perspective of the adversary. Then the following hold:

  - Conditioned on $x_{i^*}$ being a false statement, with probability negligibly close to $\varepsilon/(T+1)$, the adversary wins the adaptive soundness game and outputs an index $j$ where $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$.
  - The reduction samples $i^* \xleftarrow{R} [T]$ and a successful adversary must choose $\vec{x}$ that contains at least one false instance, so $x_{i^*}$ is a false instance with probability at least $1/T$.[4]

  Putting these pieces together, we conclude that the adversary wins the adaptive soundness game and outputs an index $j$ where $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$ with probability that is negligibly close to $\varepsilon/(T(T+1))$.

- We are now ready to begin the chaining argument. Our analysis proceeds in a sequence of hybrids indexed by $t = i^*, i^*+1, \ldots, T$. In $\mathsf{Hyb}_t$, we modify the verification program $\mathsf{VerProof}(j, \mathsf{dig}, i, z_i)$ to always output 0 when $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$ and $i^* \le i \le t$ (*irrespective* of dig or $z_i$). In the final hybrid $\mathsf{Hyb}_T$, the VerProof program always outputs 0 if $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$ and $i = T$. Correspondingly, the challenger in this final hybrid also always outputs 0, which completes the proof.

  The wrinkle with this strategy is that we *cannot* embed the instance $x_{i^*}$ into the VerProof program. This is because the adaptive adversary chooses the instance $x_{i^*}$ *after* seeing the CRS (which contains the obfuscated version of VerProof). To carry out this strategy, we need to augment VerProof with a mechanism to determine $x_{i^*}$ from the inputs. In this case, we leverage the digest dig and assume that the underlying hash function is somewhere *extractable* at $i^*$. Namely, there is an $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$ algorithm that takes as input a trapdoor td and a digest dig and outputs a value $x_{i^*}$ with the property that the only valid opening for dig at index $i^*$ is to the value $x_{i^*}$. We then inductively show that the following invariant holds for all $t = i^*, i^*+1, \ldots, T$:

  - $\mathsf{VerProof}(j, \mathsf{dig}, i, z_i)$ outputs 0 for all dig and $z_i$ when $j = j^*$ and $i^* \le i \le t$, where $j^* = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{Extract}(\mathsf{td}, \mathsf{dig}), i^*))$.

  The base case corresponds to showing that the invariant holds when $t = i^*$. To argue this, we replace the obfuscation of AggProof in the CRS with an obfuscation of the modified program $\mathsf{AggProof}_{i^*, i^*}$ (and leave VerProof unchanged):

| $\mathsf{AggProof}_{i^*, i^*}(i, j, \mathsf{dig}, x, w, \sigma, z_{i-1})$ | $\mathsf{VerProof}(i, j, \mathsf{dig}, z_i)$ |
|---|---|
| – If $C(x, w) = 0$, output $\bot$.<br>– If $\sigma$ does not open dig to $x$ at index $i$, output $\bot$.<br>– Compute $j^* = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{Extract}(\mathsf{td}, \mathsf{dig}), i^*))$. If $i = i^*$ and $j = j^*$, output $\bot$.<br>– If $j = \mathsf{F}(k_{\mathsf{sel}}, (x, i))$, output $\bot$.<br>– If $i \neq 1$ and $\mathsf{VerProof}(j, \mathsf{dig}, i-1, z_{i-1}) = 0$, output $\bot$.<br>– Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>– Output $z = \mathsf{F}(k_j, (\mathsf{dig}, i))$. | – Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>– If $\mathsf{G}(z_i) = \mathsf{G}(\mathsf{F}(k_j, (\mathsf{dig}, i)))$, output 1. Otherwise, output 0. |

[4] In the proof of Theorem 6.4, we provide the "most-likely" index $i^*$ where the adversary cheats as non-uniform advice. This is analogous to guessing the cheating index.

The somewhere extractability property of the hash function ensures that $\mathsf{AggProof}_{i^*,i^*}$ has the same input/output behavior as AggProof. Namely, the only inputs $(i, j, \mathsf{dig}, x, w, \sigma, z_{i-1})$ where the two programs could differ are those where $i = i^*$ and $\sigma$ opens to $x$ at index $i^*$. Since $\sigma$ can be opened to $x$ at $i^*$, and the hash function is extractable at $i^*$, this means that $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) = x$. The additional condition introduced in $\mathsf{AggProof}_{i^*,i^*}$ is simply checking if $j = \mathsf{F}(k_{\mathsf{sel}}, (x, i^*))$, which is already checked in the original program AggProof. As such, the obfuscations of $\mathsf{AggProof}_{i^*,i^*}$ and AggProof are computationally indistinguishable by $i\mathcal{O}$ security.

Next we modify the verification program to reject on dig and $z_i$ when $i = i^*$ and $j = j^*$. This corresponds to the base case for the chaining analysis:

| $\mathsf{AggProof}_{i^*,i^*}(i, j, \mathsf{dig}, x, w, \sigma, z_{i-1})$ | $\mathsf{VerProof}_{i^*,i^*}(i, j, \mathsf{dig}, z_i)$ |
|---|---|
| – If $C(x, w) = 0$, output $\perp$.<br>– If $\sigma$ does not open dig to $x$ at index $i$, output $\perp$.<br>– Compute $j^* = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{Extract}(\mathsf{td}, \mathsf{dig}), i^*))$. If $i = i^*$ and $j = j^*$, output $\perp$.<br>– If $j = \mathsf{F}(k_{\mathsf{sel}}, (x, i))$, output $\perp$.<br>– If $i \neq 1$ and $\mathsf{VerProof}_{i^*,i^*}(j, \mathsf{dig}, i-1, z_{i-1}) = 0$, output $\perp$.<br>– Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>– Output $z = \mathsf{F}(k_j, (\mathsf{dig}, i))$. | – Let $j^* = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{Extract}(\mathsf{td}, \mathsf{dig}), i^*))$.<br>– If $i = i^*$ and $j = j^*$, output $\perp$.<br>– Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>– If $\mathsf{G}(z_i) = \mathsf{G}(\mathsf{F}(k_j, (\mathsf{dig}, i)))$, output 1. Otherwise, output 0. |

We argue this via the following sequence of steps:

- By construction, $\mathsf{AggProof}_{i^*,i^*}$ does not compute $\mathsf{PRF}(k_{j^*}, (\mathsf{dig}, i^*))$ for all choices of dig. This is because when $i = i^*$ and $j = j^*$, the $\mathsf{AggProof}_{i^*,i^*}$ program always outputs $\perp$.
- We now appeal to (punctured) pseudorandomness of $\mathsf{PRF}(k_{j^*}, \cdot)$ and $i\mathcal{O}$ security to replace *all* of the PRG outputs $\mathsf{G}(\mathsf{PRF}(k_{j^*}, (\mathsf{dig}, i^*)))$ with a re-randomized instance derived from a single challenge $y_{\mathsf{base}}$ (where the re-randomization randomness is obtained by evaluating a PRF with $k_{j^*}$). Formally, we implement this with an exponential number of hybrids (one for each dig) and iteratively replace $\mathsf{G}(\mathsf{PRF}(k_{j^*}, (\mathsf{dig}, i^*)))$ with the re-randomized instance.
- Then, we appeal to pseudorandomness of the PRG to replace $y_{\mathsf{base}}$ with a random instance. If the PRG is (sufficiently) expanding, then with overwhelming probability, the value of $y_{\mathsf{base}}$ is no longer in the image of the PRG. In this case, *all* of the re-randomized challenges (associated with each $(\mathsf{dig}, i^*)$ pair) are no longer in the image of the PRG, and as such, there no longer exists $z$ that satisfies the verification requirement for *any* choice of $(\mathsf{dig}, i^*)$. This means the verification program outputs 0 on all inputs $(i, j, \mathsf{dig}, z_i)$ where $i = i^*$ and $j = j^*$. This is precisely the behavior of $\mathsf{VerProof}_{i^*,i^*}$, and the claim holds by $i\mathcal{O}$ security.

Having established the base case, the inductive step proceeds similarly. We define a sequence of hybrid experiments $\mathsf{Hyb}_t$ where $t = i^*, i^* + 1, \ldots, T$ where in $\mathsf{Hyb}_t$, the CRS contains obfuscations of the programs:

| $\mathsf{AggProof}_{i^*,t}(i, j, \mathsf{dig}, x, w, \sigma, z_{i-1})$ | $\mathsf{VerProof}_{i^*,t}(i, j, \mathsf{dig}, z_i)$ |
|---|---|
| – If $C(x, w) = 0$, output $\perp$.<br>– If $\sigma$ does not open dig to $x$ at index $i$, output $\perp$.<br>– Compute $j^* = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{Extract}(\mathsf{td}, \mathsf{dig}), i^*))$. If $i = i^*$ and $j = j^*$, output $\perp$.<br>– If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.<br>– If $i \neq 1$ and $\mathsf{VerProof}_{i^*,t}(j, \mathsf{dig}, i-1, z_{i-1}) = 0$, output $\perp$.<br>– Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>– Output $z = \mathsf{F}(k_j, (\mathsf{dig}, i))$. | – Let $j^* = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{Extract}(\mathsf{td}, \mathsf{dig}), i^*))$.<br>– If $i^* \leq i \leq t$ and $j = j^*$, output $\perp$.<br>– Compute $k_j \leftarrow \mathsf{F}(k, j)$.<br>– If $\mathsf{G}(z_i) = \mathsf{G}(\mathsf{F}(k_j, (x_i, i)))$, output 1. Otherwise, output 0. |

We can move from $\mathsf{Hyb}_t$ to $\mathsf{Hyb}_{t+1}$ using a similar argument as used to establish the base case:

- By construction, $\mathsf{AggProof}_{i^*,t}$ does not compute $\mathsf{PRF}(k_{j^*}, (\mathsf{dig}, t+1))$ for all choices of dig. This is because $\mathsf{VerProof}_{i^*,t}(i, j, \mathsf{dig}, z_i)$ outputs 0 when $i = t$ and $j = j^*$.
- As in the base case, we next appeal to (punctured) pseudorandomness of $\mathsf{PRF}(k_{j^*}, \cdot)$ and $i\mathcal{O}$ security to replace *all* of the PRG outputs $\mathsf{G}(\mathsf{PRF}(k_{j^*}, (\mathsf{dig}, t+1)))$ with a re-randomized instance derived from a single challenge $y_{\mathsf{base}}$.
- Then, we appeal to pseudorandomness of the PRG to replace $y_{\mathsf{base}}$ with a random instance. This means the verification program outputs 0 on all inputs $(i, j, \mathsf{dig}, z_i)$ where $j = j^*$ and $i^* \leq i \leq t+1$. But this is precisely the behavior of $\mathsf{VerProof}_{i^*,t+1}$, and the claim holds by $i\mathcal{O}$ security.

We provide the formal description in Section 6 (specifically, see the proof of Theorem 6.4).

**Potential extensions and open problems.** While this construction does not achieve better properties than our above approach relying on homomorphic re-randomizable one-way functions, it provides an alternative approach for constructing adaptively-sound SNARGs for batch NP. We believe these techniques are of independent interest, and may be amenable to generalizing beyond batch NP (e.g., to monotone-policy batch NP [BBK+23, NWW24]). We leave this as an intriguing open problem.

The homomorphic aggregation approach critically assumes that the proofs themselves are algebraic objects and satisfy some homomorphism. While initial constructions such as [WW24a, WZ24] have this property, it is not true of all adaptively-sound SNARGs (e.g., the very recent work [WW24b]). The chaining approach does not rely on any assumption about the structure of the proofs themselves, and thus, could plausibly be based on unstructured assumptions (similar to how [WW24b] constructs a SNARG for NP).

## 2 Preliminaries

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

**Sub-exponential hardness.** Our construction will rely on sub-exponential hardness assumptions, so we will formulate some of our security definitions using $(t, \varepsilon)$-notation. Generally, we say that a primitive is $(t, \varepsilon)$-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 $\mathsf{negl}(\cdot)$.

## 2.1 Cryptographic Building Blocks

We now recall the main cryptographic primitives we use in this work.

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

- **Correctness.** For any security parameter $\lambda \in \mathbb{N}$, circuit size parameter $s \in \mathbb{N}$, Boolean circuit $C$ of size at most $s$, and input $x$,

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

- **Security.** For a security parameter $\lambda$ and a bit $b \in \{0, 1\}$, we define the program indistinguishability game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input security parameter $1^\lambda$, $\mathcal{A}$ 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 $i\mathcal{O}(1^\lambda, 1^s, C_b)$.
  - $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  We say that $i\mathcal{O}$ is $(t, \varepsilon)$-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

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

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

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to k$: On input security parameter $1^\lambda$, input length $1^{\ell_{\mathsf{in}}}$, and output length $1^{\ell_{\mathsf{out}}}$, the randomized setup algorithm outputs a key $k$. We assume that the key $k$ contains an implicit description of $\ell_{\mathsf{in}}$ and $\ell_{\mathsf{out}}$.
- $\mathsf{Eval}(k, x) \to y$: On input the key $k$ and a point $x \in \{0, 1\}^{\ell_{\mathsf{in}}}$, the deterministic evaluation algorithm outputs a value $y \in \{0, 1\}^{\ell_{\mathsf{out}}}$.
- $\mathsf{Puncture}(k, x^*) \to k^{(x^*)}$: On input key $k$ and point $x^* \in \{0, 1\}^{\ell_{\mathsf{in}}}$, the puncturing algorithm outputs a punctured key $k^{(x^*)}$. We assume that the punctured key $k^{(x^*)}$ also contains an implicit description of $\ell_{\mathsf{in}}$ and $\ell_{\mathsf{out}}$.

We require that $\Pi_{\mathsf{PPRF}}$ satisfy the following properties:

- **Punctured correctness.** For all $\lambda, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, and all distinct points $x \neq x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$,

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

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

  We say that $\Pi_{\mathsf{PPRF}}$ satisfes $(t, \varepsilon)$-puncturing security if for all adversaries $\mathcal{A}$ running in time at most $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, it holds that

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

**Definition 2.3** (Somewhere Extractable Hash Family [HW15,CJJ22])**.** A somewhere extractable hash family consists of a tuple of efficient algorithms $\Pi_{\mathsf{SEH}} = (\mathsf{Setup}, \mathsf{SetupTD}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to \mathsf{hk}$: On input security parameter $1^\lambda$ and block size $1^\ell$, the setup algorithm outputs a hash key $\mathsf{hk}$.
- $\mathsf{SetupTD}(1^\lambda, 1^\ell, i) \to (\mathsf{hk}, \mathsf{td})$: On input security parameter $1^\lambda$, block size $1^\ell$, and index $i \in [2^\lambda]$, the trapdoor setup algorithm outputs a hash key $\mathsf{hk}$ and an extraction trapdoor $\mathsf{td}$.
- $\mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_t)) \to \mathsf{dig}$: On input hash key $\mathsf{hk}$ and ordered list of inputs $x_1, \ldots, x_t \in \{0,1\}^\ell$, the hash algorithm outputs a hash value $\mathsf{dig}$.
- $\mathsf{Open}(\mathsf{hk}, (x_1, \ldots, x_t), i) \to \sigma$: On input hash key $\mathsf{hk}$, ordered list of inputs $x_1, \ldots, x_t \in \{0,1\}^\ell$, and index $i \in [t]$, the opening algorithm outputs an opening $\sigma$.
- $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, x, \sigma) \to b$: On input hash key $\mathsf{hk}$, hash value $\mathsf{dig}$, index $i$, string $x \in \{0,1\}^\ell$, and opening $\sigma$, the verification algorithm outputs a bit $b \in \{0,1\}$.
- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) \to x$: On input extraction trapdoor $\mathsf{td}$ and hash value $\mathsf{dig}$, the extraction algorithm outputs a value $x \in \{0,1\}^\ell$.

We require that $\Pi_{\mathsf{SEH}}$ satisfy the following properties:

- **Opening completeness.** For any $\lambda, \ell, t \in \mathbb{N}$ with $t \leq 2^\lambda$, any $i \in [t]$, and any $x_1, \ldots, x_t \in \{0,1\}^\ell$,

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, x_i, \sigma) = 1 : \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \mathsf{dig} = \mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_t)) \\ \sigma = \mathsf{Open}(\mathsf{hk}, (x_1, \ldots, x_t), i) \end{array}\right] = 1.$$

- **Succinctness.** There exists a fixed polynomial $p$ such that the lengths of the hash values dig output by Hash and the lengths of the openings $\sigma$ output by Open in the completeness experiment satisfy $|\mathsf{dig}|, |\sigma| = p(\lambda, \ell)$.
- **Index hiding.** For a security parameter $\lambda$ and a bit $b \in \{0,1\}$, we define the index-hiding security game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input security parameter $1^\lambda$, algorithm $\mathcal{A}$ outputs the block length $1^\ell$ and an index $i \in [2^\lambda]$.
  - If $b = 0$, the challenger samples $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$. If $b = 1$, the challenger samples $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SetupTD}(1^\lambda, 1^\ell, i)$ The challenger sends hk to $\mathcal{A}$.
  - $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that $\Pi_{\mathsf{SEH}}$ satisfies $(t, \varepsilon)$-index-hiding security if for all adversaries $\mathcal{A}$ running in time $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,

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

- **Extraction correctness.** For any $\lambda, \ell, t \in \mathbb{N}$ with $t \leq 2^\lambda$, any $i \in [t]$, any $x_1, \ldots, x_t \in \{0,1\}^\ell$,

$$\Pr\left[x_i \neq \mathsf{Extract}(\mathsf{td}, \mathsf{dig}) : \begin{array}{l}(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SetupTD}(1^\lambda, 1^\ell, i) \\ \mathsf{dig} = \mathsf{Hash}(\mathsf{hk}, (x_1, \ldots, x_t))\end{array}\right] = 0.$$

- **Statistically binding.** For any $\lambda, \ell, t \in \mathbb{N}$ with $t \leq 2^\lambda$, any $i \in [t]$,

$$\Pr\left[\begin{array}{l}\exists \mathsf{dig}, x, \sigma : x \neq \mathsf{Extract}(\mathsf{td}, \mathsf{dig}) \\ \wedge\ \mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, x, \sigma) = 1\end{array} : (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SetupTD}(1^\lambda, 1^\ell, i)\right] = 0.$$

## 2.2 Batch Arguments for NP

We now formally define the notion of a batch argument for NP. We start with the definition of the NP-complete language of Boolean circuit satisfiability.

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

$$\mathcal{L}_{\mathsf{SAT}} = \{(C, x) \mid \exists w \in \{0,1\}^v \text{ s.t. } C(x, w) = 1\}$$

where $C$ is a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$ and $x \in \{0,1\}^n$ is a statement.

**Definition 2.5** (Non-interactive Batch Argument for NP)**.** A non-interactive batch argument (BARG) for the Boolean circuit satisfiability language $\mathcal{L}_{\mathsf{SAT}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, T, C) \to \mathsf{crs}$: On input security parameter $1^\lambda$, batch size $T$, and Boolean circuit $C$, the setup algorithm outputs a common reference string crs.
- $\mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T)) \to \pi$: On input common reference string crs, statements $x_1, \ldots, x_T$, and witnesses $w_1, \ldots, w_T$, the prover algorithm outputs a proof $\pi$.
- $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi) \to b$: On input common reference string crs, statements $x_1, \ldots, x_T$, and proof $\pi$, the verifier algorithm outputs a bit $b \in \{0,1\}$.

We require that $\Pi_{\mathsf{BARG}}$ satisfy the following properties:

- **Completeness.** For any security parameter $\lambda \in \mathbb{N}$, polynomials $n = n(\lambda), v = v(\lambda), T = T(\lambda)$, Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$ of $\mathrm{poly}(\lambda)$ size, and statements $x_1, \ldots, x_T \in \{0,1\}^n$ and witnesses $w_1, \ldots, w_T \in \{0,1\}^v$ such that $C(x_i, w_i) = 1$ for all $i \in [T]$, it holds that
$$\Pr\left[\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi) = 1 : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, C) \\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T)) \end{array}\right] = 1.$$
- **Succinctness.** There exists a universal polynomial $p$ such that in the completeness experiment above, we have that $|\pi| = p(\lambda, \log T, |C|)$. We say the proof is *fully succinct* if we have that $|\pi| = p(\lambda, \log T, \log |C|)$.[5]
- **Adaptive soundness.** For a security parameter $\lambda$, we define the adaptive soundness game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input security parameter $1^\lambda$, $\mathcal{A}$ starts by outputting a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$ and a number of instances $T$.
  - The challenger replies with $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, C)$.
  - $\mathcal{A}$ outputs statements $x_1, \ldots, x_T \in \{0,1\}^n$ and a proof $\pi$.
  - The output of the experiment is $b = 1$ if there exists some $i \in [T]$ such that $(C, x_i) \notin \mathcal{L}_{\mathsf{SAT}}$ and $\mathsf{Verify}(\mathsf{crs}, (x_1, \ldots, x_T), \pi) = 1$ and $b = 0$ otherwise.

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

**Remark 2.6** (Supporting Arbitrary Batch Size)**.** In our definition, the Setup algorithm needs to take the batch size $T$ as input (in binary). Note that this restriction can be generically removed using a standard "powers-of-two" construction, where we generate a CRS for every value of $T = 2^i$ for $i \in [\lambda]$. This is still efficient as the size of each CRS depends only polylogarithmically on the batch size, and padding to the next power of two only incurs constant overhead.

**Remark 2.7** (Fast Verification)**.** Definition 2.5 only requires that the size of the proof be short and does not impose any requirements on the running time of the verification algorithm. Since the size of the CRS in an adaptively-sound SNARG can scale with the circuit size $|C|$, this means the verification time may also scale polynomially with $|C|$. By the same approach described in [WW24a, Remark 2.7], we can compose the SNARG with a RAM delegation scheme (e.g., [CJJ22, KVZ21, KLVW23]) to obtain a SNARG for batch NP where the verification time is $\mathrm{poly}(\lambda, T, n, \log |C|)$, where $n$ is the length of a single statement.

**Zero-knowledge.** We also define a zero-knowledge property which essentially requires that the proof $\pi$ for a batch of statements $(x_1, \ldots, x_T)$ leak nothing more about $(x_1, \ldots, x_T)$ other than the fact that all of the statements are true.

**Definition 2.8** (Perfect Zero-Knowledge)**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ for the Boolean circuit satisfiability language $\mathcal{L}_{\mathsf{SAT}}$ satisfies perfect zero-knowledge if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ such that for any Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$ and any tuple of statements $\vec{x} = (x_1, \ldots, x_T)$ and witnesses $\vec{w} = (w_1, \ldots, w_T)$ such that $C(x_i, w_i) = 1$ for all $i \in [T]$, the following distributions are identical:
$$\left\{(\mathsf{crs}, \vec{x}, \pi) : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, C) \\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, \vec{x}, \vec{w}) \end{array}\right\} \equiv \left\{(\mathsf{crs}, \vec{x}, \pi) : \begin{array}{l} (\mathsf{crs}, \mathsf{st}) \leftarrow \mathcal{S}_0(1^\lambda, T, C) \\ \pi \leftarrow \mathcal{S}_1(\mathsf{st}, \vec{x}) \end{array}\right\}.$$

[5] This is the notion of succinctness that our constructions achieve.

### 2.3 Cryptographic Assumptions in Pairing-Free Groups

Our constructions will rely on the hardness of the discrete log and decisional Diffie-Hellman (DDH) assumptions in (pairing-free) groups. We recall the notion of a prime-order group along with the formal description of the assumptions below.

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

**Definition 2.9** (Prime-Order Group Generator)**.** Let $\lambda$ be a security parameter. A prime-order group generator is an efficient algorithm GroupGen that takes as input security parameter $1^\lambda$ and outputs the description $\mathcal{G} = (\mathbb{G}, p, g)$ of a group $\mathbb{G}$ of prime order $p = 2^{\Theta(\lambda)}$ and generated by $g \in \mathbb{G}$. Moreover, we require that the group operation in $\mathbb{G}$ be efficiently computable.

**Definition 2.10** (Discrete Log Assumption)**.** Let GroupGen be a prime-order group generator. We say that the discrete log assumption holds with respect to GroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$,

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

**Definition 2.11** (Decisional Diffie-Hellman Assumption)**.** For a security parameter $\lambda$, a bit $b \in \{0, 1\}$, and a prime-order group generator GroupGen, we define the decisional Diffie-Hellman (DDH) security game between an adversary $\mathcal{A}$ and a challenger as follows:

- The challenger starts by sampling $\mathcal{G} = (\mathbb{G}, g, p) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $x, y \xleftarrow{R} \mathbb{Z}_p^*$.
- If $b = 0$, the challenger computes $z = xy \in \mathbb{Z}_p^*$. If $b = 1$, the challenger samples $z \xleftarrow{R} \mathbb{Z}_p^*$.[6]
- The challenger then sends $(\mathcal{G}, g^x, g^y, g^z)$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit $b'$, which is the output of the experiment.

We say that the DDH assumption holds with respect to GroupGen if for all efficient adversaries $\mathcal{A}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all security parameters $\lambda \geq \lambda_{\mathcal{A}}$, it holds that

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

in the DDH security game.

## 3 Homomorphic Re-randomizable One-Way Functions

In this section, we introduce the notion of a homomorphic re-randomizable OWF, which is one of the main building blocks we use in our construction of an adaptive fully succinct BARG in Section 4. Then, in Section 3.1, we show how to construct a homomorphic re-randomizable OWF from discrete log.

**Definition 3.1** (Homomorphic Re-randomizable OWF)**.** A homomorphic re-randomizable OWF is a tuple of efficient algorithms $\Pi_{\mathsf{OWF}} = (\mathsf{Setup}, \mathsf{GenInstance}, \mathsf{Rerandomize}, \mathsf{Verify}, \mathsf{InHom}, \mathsf{OutHom}, \mathsf{RecoverSolution})$ with the following syntax:

[6] For convenience, we define the DDH assumption as sampling the exponents $x, y, z$ uniformly from $\mathbb{Z}_p^*$ as opposed to $\mathbb{Z}_p$. When $p = 2^{\Omega(\lambda)}$, the uniform distribution over $\mathbb{Z}_p^*$ and $\mathbb{Z}_p$ for prime $p$ is statistically indistinguishable.

- $\mathsf{Setup}(1^\lambda, 1^m) \to \mathsf{crs}$ : On input security parameter $1^\lambda$ and re-randomization parameter $1^m$, the setup algorithm outputs a common reference string crs. We assume that the crs contains an implicit description of the input space $\mathcal{Z}$ and the output space $\mathcal{Y}$.
- $\mathsf{GenInstance}(\mathsf{crs}) \to (y, z)$ : On input common reference string crs, the instance-generation algorithm outputs challenge $y \in \mathcal{Y}$ together with a preimage $z \in \mathcal{Z}$.
- $\mathsf{Rerandomize}(\mathsf{crs}, y) \to y'$ : On input common reference string crs and challenge $y \in \mathcal{Y}$, the randomization algorithm outputs a new challenge $y' \in \mathcal{Y}$.
- $\mathsf{Verify}(\mathsf{crs}, y, z) \to 0/1$. On input common reference string crs, challenge $y \in \mathcal{Y}$, and preimage $z \in \mathcal{Z}$, the verification algorithm outputs 0 or 1.
- $\mathsf{RecoverSolution}(\mathsf{crs}, z', r) \to z$ : On input common reference string crs, preimage $z' \in \mathcal{Z}$, and randomness $r$, the preimage recovery algorithm outputs a new preimage $z \in \mathcal{Z}$.
- $\mathsf{InHom}(\mathsf{crs}, (z_1, \ldots, z_\ell)) \to z$: On input common reference string crs and preimages $z_1, \ldots, z_\ell \in \mathcal{Z}$, the input homomorphism algorithm outputs a new preimage $z \in \mathcal{Z}$.
- $\mathsf{OutHom}(\mathsf{crs}, (y_1, \ldots, y_\ell)) \to y$: On input common reference string crs and challenges $y_1, \ldots, y_\ell \in \mathcal{Y}$, the output homomorphism algorithm outputs a new challenge $y \in \mathcal{Y}$.

We require that $\Pi_{\mathsf{OWF}}$ satisfy the following properties:

- **Correctness.** For all $\lambda, m \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, and all $(y, z)$ in the support of $\mathsf{GenInstance}(\mathsf{crs})$, we have that $\mathsf{Verify}(\mathsf{crs}, y, z) = 1$.
- **Homomorphism.** For all $\lambda, m \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, all $z_1, \ldots, z_\ell \in \mathcal{Z}$, and all $y_1, \ldots, y_\ell \in \mathcal{Y}$ such that $\mathsf{Verify}(\mathsf{crs}, y_i, z_i) = 1$ for all $i \in [\ell]$, we have that

  $$\mathsf{Verify}(\mathsf{crs}, \mathsf{OutHom}(\mathsf{crs}, (y_1, \ldots, y_\ell)), \mathsf{InHom}(\mathsf{crs}, (z_1, \ldots, z_\ell))) = 1.$$

  Further, InHom has a corresponding inversion algorithm $\mathsf{InHom}^{-1}$ such that for all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, for all preimages $z, z' \in \mathcal{Z}$ and challenges $y, y' \in \mathcal{Y}$, if

  $$\mathsf{Verify}(\mathsf{crs}, \mathsf{OutHom}(\mathsf{crs}, (y, y')), z) = 1 \text{ and } \mathsf{Verify}(\mathsf{crs}, y', z') = 1,$$

  then

  $$\mathsf{Verify}(\mathsf{crs}, y, \mathsf{InHom}^{-1}(\mathsf{crs}, (z, z'))) = 1.$$

- **One-wayness.** For a security parameter $\lambda$, a re-randomization parameter $m$, and a bit $b \in \{0, 1\}$, we define the one-wayness security game between an adversary $\mathcal{A}$ and a challenger as follows:
  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$ and sends $(\mathsf{crs}, y)$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ sends a preimage $z'$ to the challenger.
  - The challenger outputs a bit $b' \leftarrow \mathsf{Verify}(\mathsf{crs}, y, z')$.

We say that $\Pi_{\mathsf{OWF}}$ is $(t, \varepsilon)$-one-way if for all polynomials $m = m(\lambda)$ and 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 security parameters $\lambda \geq \lambda_{\mathcal{A}}$, it holds that

$$\mathsf{PRGAdv}_{\mathcal{A}}(\lambda) \coloneqq \Pr[b' = 1] \leq \varepsilon(\lambda)$$

in the one-wayness security game.

- **Re-randomization correctness.** For all $\lambda \in \mathbb{N}$, all polynomials $m = m(\lambda)$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, all preimages $z' \in \mathcal{Z}$, all $y \in \mathcal{Y}$, and all randomness $r$ where

$$\mathsf{Verify}(\mathsf{crs}, \mathsf{Rerandomize}(\mathsf{crs}, y; r), z') = 1$$

we have that

$$\mathsf{Verify}(\mathsf{crs}, y, \mathsf{RecoverSolution}(\mathsf{crs}, z', r)) = 1.$$

- **Re-randomization security.** For a security parameter $\lambda$, a re-randomization parameter $m$, and a bit $b \in \{0, 1\}$, we define the re-randomization security game between an adversary $\mathcal{A}$ and a challenger as follows:
  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ and $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$.
  - If $b = 0$, the challenger samples $(y^*, z^*) \leftarrow \mathsf{GenInstance}(\mathsf{crs})$. If $b = 1$, the challenger samples $y^* \leftarrow \mathsf{Rerandomize}(\mathsf{crs}, y_{\mathsf{base}})$.
  - The challenger then sends $(\mathsf{crs}, y_{\mathsf{base}}, y^*)$ to $\mathcal{A}$.
  - $\mathcal{A}$ outputs a bit $b'$, which is the output of the experiment.

  We say that $\Pi_{\mathsf{OWF}}$ satisfies $(t, \varepsilon)$-re-randomization security if for all polynomials $m = m(\lambda)$ and 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 security parameters $\lambda \geq \lambda_{\mathcal{A}}$, it holds that

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

  in the re-randomization security game.

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

### 3.1 Constructing Homomorphic Re-randomizable OWFs

In this section, we show that the construction of a re-randomizable OWF from discrete log [WW24a, Construction 5.3] is a homomorphic re-randomizable OWF. Though we do not formalize it in this work, the second construction of a re-randomizable OWF in [WW24a] based on computing modular square roots (i.e., the function $f(z) = z^2 \mod N$ where $N = pq$ is an RSA modulus) is also homomorphic, as $f(z_0 z_1) = (z_0 z_1)^2 = z_0^2 z_1^2 = f(z_0) f(z_1) \mod N$. We start by recalling the discrete-log-based construction from [WW24a], with the addition of the InHom and OutHom algorithms for the homomorphism property. For simplicity, we describe the scheme with *additive* blinding rather than multiplicative blinding:

**Construction 3.2** (Homomorphic Re-randomizable OWF)**.** Let GroupGen be a prime-order group generator. We construct a homomorphic re-randomizable OWF $\Pi_{\mathsf{OWF}} = (\mathsf{Setup}, \mathsf{GenInstance}, \mathsf{Rerandomize}, \mathsf{InHom}, \mathsf{OutHom}, \mathsf{RecoverSolution})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^m)$: On input security parameter $1^\lambda$ and re-randomization parameter $1^m$, the setup algorithm samples $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and outputs $\mathsf{crs} = (\mathbb{G}, p, g)$. The domain of the OWF is $\mathcal{Z} = \mathbb{Z}_p^*$ and the range is $\mathcal{Y} = \mathbb{G}$.
- $\mathsf{GenInstance}(\mathsf{crs}) \to (y, z)$: On input common reference string $\mathsf{crs} = (\mathbb{G}, p, g)$, the instance generation algorithm samples $z \xleftarrow{R} \mathbb{Z}_p^*$, computes $y = g^z$, and outputs $(y, z)$.
- $\mathsf{Rerandomize}(\mathsf{crs}, y) \to y'$: On input common reference string $\mathsf{crs} = (\mathbb{G}, p, g)$ and challenge $y \in \mathbb{G}$, the randomization algorithm samples $r \xleftarrow{R} \mathbb{Z}_p^*$ and outputs $y \cdot g^r \in \mathbb{G}$.
- $\mathsf{Verify}(\mathsf{crs}, y, z) \to 0/1$: On input common reference string $\mathsf{crs} = (\mathbb{G}, p, g)$, challenge $y \in \mathbb{G}$, and preimage $z \in \mathbb{Z}_p^*$, the verification algorithm outputs 1 if $y = g^z$ and 0 otherwise.
- $\mathsf{RecoverSolution}(\mathsf{crs}, z', r) \to z$: On input common reference string crs, preimage $z' \in \mathbb{Z}_p^*$, and randomness $r \in \mathbb{Z}_p^*$, the preimage recovery algorithm outputs $z' - r \in \mathbb{Z}_p^*$.
- $\mathsf{InHom}(\mathsf{crs}, (z_1, \ldots, z_\ell)) \to z$: On input common reference string crs and preimages $z_1, \ldots, z_\ell \in \mathbb{Z}_p^*$, the input homomorphism outputs $\sum_{i \in [\ell]} z_i \in \mathbb{Z}_p^*$.
- $\mathsf{OutHom}(\mathsf{crs}, (y_1, \ldots, y_\ell)) \to y$: On input common reference string crs and challenges $y_1, \ldots, y_\ell \in \mathbb{G}$, the output homomorphism outputs $\prod_{i \in [\ell]} y_i \in \mathbb{G}$.

We refer to [WW24a, §5.1] for the proofs of the correctness, one-wayness, re-randomization correctness, re-randomization security, and succinctness properties. Here, we show the homomorphism property.

**Theorem 3.3** (Homomorphism)**.** *Construction 3.2 satisfies homomorphism.*

*Proof.* Take any $\mathsf{crs} = (\mathbb{G}, p, g)$ in the support of $\mathsf{GroupGen}(1^\lambda)$, and any $z_1, \ldots, z_\ell \in \mathbb{Z}_p$, $y_1, \ldots, y_\ell \in \mathrm{G}$ where $\mathsf{Verify}(\mathsf{crs}, y_i, z_i) = 1$ for all $i \in [\ell]$. By construction of Verify, this means that $y_i = g^{z_i}$ for all $i \in [\ell]$. Then

$$g^{\mathsf{InHom}(\mathsf{crs}, (z_1, \ldots, z_\ell))} = g^{\sum_{i \in [\ell]} z_i} = \prod_{i \in [\ell]} g^{z_i} = \prod_{i \in [\ell]} y_i = \mathsf{OutHom}(\mathsf{crs}, (y_1, \ldots, y_\ell)).$$

Thus

$$\mathsf{Verify}(\mathsf{crs}, \mathsf{OutHom}(\mathsf{crs}, (y_1, \ldots, y_\ell)), \mathsf{InHom}(\mathsf{crs}, (z_1, \ldots, z_\ell))) = 1,$$

as required. Next, define the inversion algorithm $\mathsf{InHom}^{-1}(\mathsf{crs}, z, z') = z - z'$. Take any $z, z' \in \mathbb{Z}_p^*$ and $y, y' \in \mathbb{G}$, where $\mathsf{Verify}(\mathsf{crs}, \mathsf{OutHom}(\mathsf{crs}, (y, y')), z) = 1$ and $\mathsf{Verify}(\mathsf{crs}, y', z') = 1$. This means

$$g^z = \mathsf{OutHom}(\mathsf{crs}, y, y') = yy' \quad \text{and} \quad g^{z'} = y'.$$

Then,

$$g^{\mathsf{InHom}^{-1}(\mathsf{crs}, z, z')} = g^{z - z'} = \frac{yy'}{y'} = y,$$

and $\mathsf{Verify}(\mathsf{crs}, y, \mathsf{InHom}^{-1}(\mathsf{crs}, (z, z'))) = 1$ as desired. □

# 4 SNARG for Batch NP from Homomorphic Re-randomizable OWFs

In this section, we show how to construct a fully succinct SNARG for batch NP using indistinguishability obfuscation together with a homomorphic re-randomizable OWF. Our construction follows a similar template as the construction from [WW24a].

**Construction 4.1** (Adaptive Batch Argument for NP)**.** Let $\lambda$ be a security parameter. We construct a BARG scheme that supports NP languages with an arbitrary polynomial number $T = T(\lambda) < 2^\lambda$ of instances of length $n = n(\lambda)$. Our construction will leverage sub-exponential hardness of the below primitives (except for one-wayness of $\Pi_{\mathsf{OWF}}$). Our construction relies on the following primitives:

- Let $i\mathcal{O}$ be an indistinguishability obfuscation scheme for Boolean circuits.
- Let $\Pi_{\mathsf{PPRF}} = (\mathsf{F.Setup}, \mathsf{F.Eval}, \mathsf{F.Puncture})$ be a puncturable PRF. For a key $k$ and an input $x$, we will write $\mathsf{F}(k, x)$ to denote $\mathsf{F.Eval}(k, x)$.
- Let $\Pi_{\mathsf{OWF}} = (\mathsf{OWF.Setup}, \mathsf{OWF.GenInstance}, \mathsf{OWF.Rerandomize}, \mathsf{OWF.Verify}, \mathsf{OWF.RecoverSolution}, \mathsf{OWF.InHom}, \mathsf{OWF.OutHom})$ be a homomorphic re-randomizable one-way function.

We will describe how to define the polynomials $\lambda_{\mathsf{obf}}, \lambda_{\mathsf{PRF}}$, and $m$ in the security analysis. We construct a fully succinct non-interactive batch argument $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ for NP as follows:

- $\mathsf{Setup}(1^\lambda, T, C)$: On input security parameter $1^\lambda$, batch size $T$, and Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, the setup algorithm does the following:
  - Sample OWF parameters $\mathsf{crs}_f \leftarrow \mathsf{OWF.Setup}(1^\lambda, 1^m)$.
  - Let $t = \log(T+1)$. Let $\rho$ be a bound on the number of bits of randomness the sampling algorithm $\mathsf{OWF.GenInstance}(\mathsf{crs}_f)$ takes. Let $\tau$ be the number of bits of randomness that the setup algorithm $\mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^\rho)$ takes.
  - Sample a "selector" PPRF key $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^t)$.
  - Sample a "key generator" PPRF key $k \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^t, 1^\tau)$.
  - Define the GenSol program with the OWF parameters $\mathsf{crs}_f$, circuit $C$, and PPRF keys $k, k_{\mathsf{sel}}$ hard-coded:

    $\mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}](i, j, x_i, w_i)$

    **Inputs:** index $i$, selection symbol $j$, statement $x_i$, witness $w_i$

    1: If $C(x_i, w_i) = 0$, output $\perp$.
    2: If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.
    3: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^\rho; \mathsf{F}(k, j))$.
    4: Compute $(y, z) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ and output $z$.
  - Define the GenChall program with the OWF parameters $\mathsf{crs}_f$ and PPRF key $k$ hard-coded:

| $\mathsf{GenChall}[\mathsf{crs}_f, k](i, j, x_i)$ |
|---|
| **Inputs:** index $i$, selection symbol $j$, statement $x_i$<br>1: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j))$.<br>2: Compute $(y, z) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ and output $y$. |

  - Let $s = s(\lambda, n, |C|)$ be the maximum size of the GenChall and GenSol programs as well as those appearing in the security analysis.
  - Construct the obfuscated programs

$$\mathsf{ObfGenChall} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenChall}[\mathsf{crs}_f, k])$$

    and

$$\mathsf{ObfGenSol} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}]).$$

  - Output $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenChall}, \mathsf{ObfGenSol})$.

- $\mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T))$: On input $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenChall}, \mathsf{ObfGenSol})$, the statements $x_1, \ldots, x_T$, and the witnesses $w_1, \ldots, w_T$, the prover algorithm proceeds as follows.
  - Initialize $i = 1, j = 1$. Then, while $i \leq T$:
    - Compute $z_i \leftarrow \mathsf{ObfGenSol}(i, j, x_i, w_i)$.
    - If $z_i = \perp$, set $i = 1, j = j + 1$. Otherwise, set $i = i + 1$.
  - Compute $z = \mathsf{OWF.InHom}(\mathsf{crs}_f, (z_1, \ldots, z_T))$ and output $(j, z)$.
- $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi)$: On input $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenChall}, \mathsf{ObfGenSol})$, the statements $x_1, \ldots, x_T$, and the proof $\pi = (j, z)$, the verification algorithm proceeds as follows:
  - For each $i \in [T]$, compute $y_i \leftarrow \mathsf{ObfGenChall}(i, j, x_i)$.
  - Compute $y = \mathsf{OWF.OutHom}(\mathsf{crs}_f, (y_1, \ldots, y_T))$ and output $\mathsf{OWF.Verify}(\mathsf{crs}_f, y, z)$.

**Theorem 4.2** (Completeness)**.** *Suppose $i\mathcal{O}$ is correct and $\Pi_{\mathsf{OWF}}$ satisfies homomorphism. Then Construction 4.1 is complete.*

*Proof.* Take any security parameter $\lambda \in \mathbb{N}$, any Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, any $T \leq 2^\lambda$, and any statements $(x_1, \ldots, x_T)$ and witnesses $(w_1, \ldots, w_T)$ such that $C(x_i, w_i) = 1$ for all $i \in [T]$. Let $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenSol}, \mathsf{ObfGenChall}) \leftarrow \mathsf{Setup}(1^\lambda, C, T)$ and $\pi = (j, z) \leftarrow \mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T))$. Consider the output of $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi)$:

- By construction, ObfGenSol is an obfuscation of the program $\mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}]$, where

$$k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^t) \quad \text{and} \quad k \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^t, 1^\tau).$$

  Algorithm P obtains $(j, z_1), \ldots, (j, z_T)$ by evaluating ObfGenSol on inputs $(i, j, x_i, w_i)$. By correctness of $i\mathcal{O}$ and the definition of GenSol, this means that $z_i$ was generated by computing $(y_i, z_i) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ for all $i \in [T]$. Note that an index $j \in [T + 1]$ always exists, because for each index $i$, there is just a single index $j_i = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$ where the GenSol program outputs $\perp$. Since there are at most $T$ instances, there are at most $T$ indices $j \in [T + 1]$ that fail, or equivalently, there must exist at least one index $j \in [T + 1]$ where $j \neq \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$ for all $i \in [T]$.

- By construction, ObfGenChall is an obfuscation of the program $\mathsf{GenChall}[\mathsf{crs}_f, k]$ where $\mathsf{crs}_f \leftarrow \mathsf{OWF.Setup}(1^\lambda, 1^m)$. Algorithm V computes the instance $y_i \leftarrow \mathsf{ObfGenChall}(i, j, x_i)$, which was generated by computing $(y_i, z_i) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ for all $i \in [T]$. By correctness of $i\mathcal{O}$ and correctness of OWF, this means that $\mathsf{OWF.Verify}(\mathsf{crs}_f, y_i, z_i) = 1$ for all $i \in [T]$.

- Finally, algorithm P computes $z = \mathsf{OWF.InHom}(\mathsf{crs}_f, (z_1, \ldots, z_T))$ and algorithm V computes $y = \mathsf{OWF.OutHom}(\mathsf{crs}_f, (y_1, \ldots, y_T))$. By homomorphism of $\Pi_{\mathsf{OWF}}$, we have $\mathsf{OWF.Verify}(\mathsf{crs}_f, y, z) = 1$.

Thus V outputs 1 with probability 1. □

**Theorem 4.3** (Succinctness). *Suppose $\Pi_{\mathsf{OWF}}$ is succinct. Then Construction 4.1 is succinct.*

*Proof.* A proof $(j, z)$ in Construction 4.1 consists of a selection symbol $j \in [T+1]$ and a OWF preimage $z$. Since $\Pi_{\mathsf{OWF}}$ is succinct, there is a fixed polynomial $p$ such that $|z| \le p(\lambda + \log m)$. Since $m(\lambda, n)$ in Construction 4.1 is a fixed polynomial in the security parameter $\lambda$ and the statement length $n$ and the statement length is always upper-bounded by the circuit size, it follows that $|\pi| \le \mathsf{poly}(\lambda + \log |C|) + \log T$. □

**Theorem 4.4** (Adaptive Soundness). *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure, $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness and $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security, and $\Pi_{\mathsf{OWF}}$ satisfies re-randomization correctness, $(1, \mathsf{negl}(\lambda))$-one-wayness, and $(1, 2^{-m^{\varepsilon_m}})$-re-randomization security for constants $\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}}, \varepsilon_m \in (0, 1)$ and security parameters $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}, \lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}, m = (\lambda + n)^{1/\varepsilon_m}$. Then Construction 4.1 is adaptively sound.*

*Proof.* Our proof follows a similar structure as the proof of [WW24a, Theorem 4.3]. Let $\mathcal{A}$ be an efficient adversary that succeeds in the adaptive soundness game against Construction 4.1 with (non-negligible) probability $\varepsilon(\lambda)$. We first claim that without loss of generality, we can assume that for every security parameter $\lambda$, $\mathcal{A}$ always outputs a circuit $C$ with statements of a fixed length $n = n(\lambda)$ and witnesses of a fixed length $v = v(\lambda)$ and a fixed batch size $T = T(\lambda)$. Formally, since $\mathcal{A}$ is a polynomial-time algorithm, $\mathcal{A}(1^\lambda)$ outputs a Boolean circuit of size at most $s_{\max}(\lambda) = \mathsf{poly}(\lambda)$ and a maximum batch size $T_{\max}(\lambda) = \mathsf{poly}(\lambda)$. This in turn defines maximum statement and witness lengths $n_{\max}(\lambda), v_{\max}(\lambda) \le s_{\max}(\lambda)$. In an execution of the adaptive soundness game, let $\mathsf{E}_{n',v',T'}$ be the event that $\mathcal{A}$ outputs a circuit $C$ with statements of length $n'$ and witnesses of length $v'$ and batch size $T'$. Then

$$\Pr[\mathcal{A} \text{ wins the soundness game}] = \sum_{\substack{n' \in [n_{\max}] \\ v' \in [v_{\max}] \\ T' \in [T_{\max}]}} \Pr[\mathcal{A} \text{ wins the soundness game} \wedge \mathsf{E}_{n',v',T'}].$$

Thus there must exist some $(n, v, T) \in [n_{\max}] \times [v_{\max}] \times [T_{\max}]$ such that such that

$$\Pr[\mathcal{A}(1^\lambda) \text{ wins the soundness game} \wedge \mathsf{E}_{n,v,T}] \ge \frac{\varepsilon(\lambda)}{n_{\max} \cdot v_{\max} \cdot T_{\max}}.$$

For each security parameter $\lambda$, define $n = n(\lambda)$, $v = v(\lambda)$, and $T = T(\lambda)$ to be the smallest values such that the above equation holds. We now construct a new (non-uniform) adversary $\mathcal{A}'$ that functions as a wrapper around $\mathcal{A}$, but only outputs circuits with fixed statement and witness lengths and a fixed batch size. Namely, $\mathcal{A}'$ takes as input the security parameter $1^\lambda$ and the non-uniform advice $n = n(\lambda), v = v(\lambda), T = T(\lambda)$. $\mathcal{A}'$ runs $(C', T') \leftarrow \mathcal{A}(1^\lambda)$. If $C'$ does not have statements of length $n$

and witnesses of length $v$ or $T' \neq T$ , then $\mathcal{A}'$ aborts. Otherwise, $\mathcal{A}'$ simply follows the behavior of $\mathcal{A}$ (and outputs whatever $\mathcal{A}$ outputs). By construction,

$$\begin{aligned}\Pr[\mathcal{A}'(1^\lambda) \text{ wins the soundness game}] &= \Pr[\mathcal{A}(1^\lambda) \text{ wins the soundness game} \wedge \mathsf{E}_{n,v,T}] \\ &\geq \frac{\varepsilon(\lambda)}{n_{\max} \cdot v_{\max} \cdot T_{\max}}.\end{aligned}$$

Thus $\mathcal{A}'$ still has a non-negligible success probability in the soundness game. For the remainder of this proof, we will assume the adversary always outputs a circuit $C$ with statements of length $n$ and witnesses of length $v$ and batch size $T$. We now define a sequence of hybrid experiments:

$\mathsf{Hyb}_0$: This is the real adaptive soundness experiment.

- Adversary $\mathcal{A}$, on input $1^\lambda$, starts by outputting a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, and the batch size $T$.
- The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, C)$ and gives crs to $\mathcal{A}$.
- Adversary $\mathcal{A}$ outputs a batch of statements $\vec{x} = (x_1, \ldots, x_T)$ and a proof $\pi = (j, z)$.
- The challenger outputs 1 if and only if for some $i \in [T]$, $(C, x_i) \notin \mathcal{L}_{\mathsf{SAT}}$ and $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$.

$\mathsf{Hyb}_1$: Same as $\mathsf{Hyb}_0$ except the challenger samples $i^* \xleftarrow{R} [T]$ and outputs 1 if and only if $(C, x_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$ and $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi)) = 1$.

$\mathsf{Hyb}_2$: Same as $\mathsf{Hyb}_1$ except the challenger additionally checks that $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$

$\mathsf{Hyb}_3$: Same as $\mathsf{Hyb}_2$ except the challenger stops checking that $(C, x_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$.

$\mathsf{Hyb}_4$: Same as $\mathsf{Hyb}_3$ except the challenger changes how it defines the GenChall program. During setup, the challenger additionally samples

- $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f)$
- $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^\mu)$

where $\mu$ is a bound on the number of bits of randomness the OWF.Rerandomize algorithm takes. It defines GenChall′ as follows:

| $\mathsf{GenChall}'[\mathsf{crs}_f, k, k_{\mathsf{sel}}, i^*, k_{\mathsf{rerand}}, y_{\mathsf{base}}](i, j, x_i)$ |
|---|
| **Inputs:** index $i$, selection symbol $j$, statement $x_i$<br>1: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^\rho; \mathsf{F}(k, j))$.<br>2: If $i = i^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (x_i, i)))$.<br>3: Else: compute $(y, z) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ and output $y$. |

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

**Lemma 4.5.** $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \geq \frac{1}{T} \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$.

*Proof.* Suppose the output in $\mathsf{Hyb}_0$ is 1. This means there exists some index $i \in [T]$ where $(C, x_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$. Since the challenger samples $i^* \xleftarrow{R} [T]$ independently of the common reference string (and thus, the view of the adversary), with probability at least $1/T$, it will also be the case that $(C, x_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$, in which case the output in $\mathsf{Hyb}_1$ is also 1. □

**Lemma 4.6.** *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness and $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security for constants $\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}} \in (0,1)$ and security parameters $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}, \lambda_{\mathsf{PRF}} = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then*

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \frac{1}{T+1} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - 2^{-\Omega(\lambda)}.$$

*Proof.* Consider an execution of $\mathsf{Hyb}_1$ or $\mathsf{Hyb}_2$. For a fixed $x^* \in \{0,1\}^n$, let $\mathsf{E}_{x^*}$ be the event that $\mathcal{A}$ outputs $\vec{x} = (x_1, \ldots, x_T)$ such that $x_{i^*} = x^*$. By definition, we can now write

$$\begin{aligned} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] &= \sum_{x^* \in \{0,1\}^n} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] \\ \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] &= \sum_{x^* \in \{0,1\}^n} \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}]. \end{aligned} \tag{4.1}$$

To prove the lemma, we show that for all $x^* \in \{0,1\}^n$,

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] \geq \frac{1}{T+1} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] - \frac{O(T)}{2^{\lambda+n}}. \tag{4.2}$$

If Eq. (4.2) holds, then

$$\begin{aligned} \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] &= \sum_{x^* \in \{0,1\}^n} \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] \\ &\geq \sum_{x^* \in \{0,1\}^n} \left( \frac{1}{T+1} \Pr[\mathsf{Hyb}_1(\mathcal{A}) \wedge \mathsf{E}_{x^*}] - \frac{O(T)}{2^{\lambda+n}} \right) \\ &\geq \frac{1}{T+1} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \frac{O(T)}{2^{\lambda}}, \end{aligned}$$

which proves the claim since $T = \mathsf{poly}(\lambda)$. We now show Eq. (4.2) holds. Fix any $x^* \in \{0,1\}^n$. If $(C, x^*) \in \mathcal{L}_{\mathsf{SAT}}$, then

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] = 0 = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}],$$

so Eq. (4.2) holds. Thus we only need to consider the case where $(C, x^*) \notin \mathcal{L}_{\mathsf{SAT}}$. We proceed by defining a sequence of intermediate hybrids.

$\mathsf{Hyb}_{1,0}^{(x^*)}$ : Same as $\mathsf{Hyb}_1$ except the challenger additionally checks that $x_{i^*} = x^*$ (i.e., that $\mathsf{E}_{x^*}$ occurred).

$\mathsf{Hyb}_{1,1}^{(x^*)}$ : Same as $\mathsf{Hyb}_{1,0}^{(x^*)}$ except the challenger does the following. It first computes a punctured key $k_{\mathsf{sel}}^{(x^*,i^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{sel}}, (x^*, i^*))$ and defines a modified version of GenSol which additionally has $i^*, x^*$ hard-coded as follows:

| $\mathsf{GenSol}'[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}^{(x^*,i^*)}, i^*, x^*](i, j, x_i, w_i)$ |
|---|
| **Inputs:** index $i$, selection symbol $j$, statement $x_i$, witness $w_i$ |
| 1: If $i = i^*$ and $x_i = x^*$, output $\perp$. |

2: If $C(x_i, w_i) = 0$, output $\bot$.

3: If $j = \mathsf{F}(k_{\mathsf{sel}}^{(x^*,i^*)}, (x_i, i))$, output $\bot$.

4: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j))$.

5: Compute $(y, z) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ and output $z$.

$\mathsf{Hyb}_{1,2}^{(x^*)}$ : Same as $\mathsf{Hyb}_{1,1}^{(x^*)}$ except that after $\mathcal{A}$ outputs $(\vec{x}, \pi)$ where $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$, the challenger samples $j' \xleftarrow{R} [T+1]$ and additionally checks that $j = j'$.

$\mathsf{Hyb}_{1,3}^{(x^*)}$ : Same as $\mathsf{Hyb}_{1,2}^{(x^*)}$ except the challenger replaces the check $j = j'$ with an updated check $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$.

$\mathsf{Hyb}_{1,4}^{(x^*)}$ : Same as $\mathsf{Hyb}_{1,3}^{(x^*)}$ except the challenger reverts to obfuscating $\mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}]$ instead of $\mathsf{GenSol}'[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}^{(x^*,i^*)}, i^*, x^*]$.

By definition,

$$\begin{aligned} \Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A}) = 1] &= \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] \\ \Pr[\mathsf{Hyb}_{1,4}^{(x^*)}(\mathcal{A}) = 1] &= \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}]. \end{aligned} \tag{4.3}$$

We now consider each pair of adjacent distributions.

**Claim 4.7.** *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then*

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

*Proof.* We first show that $\mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}]$ in $\mathsf{Hyb}_{1,0}^{(x^*)}$ and $\mathsf{GenSol}'[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}^{(x^*,i^*)}, i^*, x^*]$ in $\mathsf{Hyb}_{1,1}^{(x^*)}$ compute identical functionalities. For a particular input $(i, j, x_i, w_i)$ consider the following cases:

**Case 1.** If $i \neq i^*$ or $x_i \neq x^*$, the two programs behave identically except that the latter is using $k_{\mathsf{sel}}^{(x^*,i^*)}$, so by punctured correctness, they have the same output.

**Case 2.** If $i = i^*$ and $x_i = x^*$, $\mathsf{GenSol}'$ immediately rejects. Since $(C, x^*) \notin \mathcal{L}_{\mathsf{SAT}}$, it follows that $C(x^*, w_i) = C(x_i, w_i) = 0$, so GenSol also rejects.

The claim now follows from $i\mathcal{O}$ security. Formally, suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$, we have that $|\Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A})]| > 1/2^{\lambda+n}$. Let $\Lambda_{\mathcal{B}} = \{(\lambda + n)^{1/\varepsilon_{\mathsf{obf}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $n$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set.

We define an efficient algorithm $\mathcal{B}$ which plays the $i\mathcal{O}$ security game with $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{obf}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{obf}} < 1$ and $n > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{obf}}$).

**Algorithm $\mathcal{B}[x^*]$**

**Inputs:** $1^{\lambda_{\mathsf{obf}}}$ from $i\mathcal{O}$ challenger, $1^{\lambda}$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Sample $i^* \leftarrow [T]$ and $\mathsf{crs}_f, k_{\mathsf{sel}}, k$. Then compute ObfGenChall as in Setup.

3: Compute $k_{\mathsf{sel}}^{(x^*,i^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{sel}}, (x^*, i^*))$.

4: Construct challenge programs $\mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}]$ and $\mathsf{GenSol}'[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}^{(x^*,i^*)}, i^*, x^*]$ and send to the $i\mathcal{O}$ challenger. The challenger replies with an obfuscated program ObfGenSol.

5: Let $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenChall}, \mathsf{ObfGenSol})$.

6: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

7: Output 1 if and only if $x^* = x_{i^*}$ and $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi) = 1$.

If the $i\mathcal{O}$ challenger obfuscates $\mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}]$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,0}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A}) = 1]$. If the $i\mathcal{O}$ challenger obfuscates $\mathsf{GenSol}'[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}^{(x^*,i^*)}, i^*, x^*]$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,1}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1]$. Thus by $i\mathcal{O}$ security we have that

$$|\Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A}) = 1]| = \mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) \leq 1/2^{\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}} = 1/2^{\lambda+n} \leq 1/2^{\lambda+n}. \qquad \square$$

**Claim 4.8.** $\Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1] \geq \frac{1}{T+1} \Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1]$.

*Proof.* The challenger samples $j' \xleftarrow{R} [T+1]$ after $\mathcal{A}$ outputs $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$). $\square$

**Claim 4.9.** *Suppose* $\Pi_{\mathsf{PPRF}}$ *satisfies* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-puncturing security for constant* $\varepsilon_{\mathsf{PRF}} \in (0, 1)$ *and security parameter* $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. *Then*

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

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

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

Let $\Lambda_{\mathcal{B}} = \{(\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $n$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set. We define an efficient algorithm $\mathcal{B}$ which plays the puncturing security game with $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{PRF}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{PRF}} < 1$ and $n > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{PRF}}$).

**Algorithm** $\mathcal{B}[x^*]$

**Inputs:** $1^{\lambda_{\mathsf{PRF}}}$ from PPRF challenger, $1^{\lambda}$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Sample $i^* \xleftarrow{R} [T]$, and $\mathsf{crs}_f, k_{\mathsf{sel}}$ as in Setup. Compute ObfGenChall as in Setup.

3: Send input length $1^n$, output length $1^t$, and punctured point $(x^*, i^*)$ to the PPRF challenger. The PPRF challenger replies with the punctured key $k_{\mathsf{sel}}^{(x^*,i^*)}$ and challenge value $j' \in \{0,1\}^t$.

4: Compute $\mathsf{ObfGenSol} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenSol}'[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}^{(x^*,i^*)}, i^*, x^*])$.

5: Let $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenSol}, \mathsf{ObfGenChall})$.

6: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

7: Output 1 if and only if $x^* = x_{i^*}$, $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi) = 1$, and $j = j'$.

If the PPRF challenger samples $j' \xleftarrow{R} \{0,1\}^t$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,2}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1]$. If the PPRF challenger computes $j' \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x^*, i^*))$ then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,3}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{1,3}^{(x^*)}(\mathcal{A}) = 1]$. Thus by PPRF security we have that

$$|\Pr[\mathsf{Hyb}_{1,3}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1]| = \mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) \leq 1/2^{\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}} = 1/2^{\lambda+n} \leq 1/2^{\lambda+n}. \quad \square$$

**Claim 4.10.** *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then*

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

*Proof.* This follows by an analogous argument as Claim 4.7. $\square$

Combining Claims 4.7 to 4.10, we have that

$$\begin{aligned}
\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] &= \Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A}) = 1] && \text{by Eq. (4.3)} \\
&\leq \Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1] + \frac{1}{2^{\lambda+n}} && \text{by Claim 4.7} \\
&\leq (T+1) \cdot \Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1] + \frac{1}{2^{\lambda+n}} && \text{by Claim 4.8} \\
&\leq (T+1) \cdot \left(\Pr[\mathsf{Hyb}_{1,3}^{(x^*)}(\mathcal{A}) = 1] + \frac{1}{2^{\lambda+n}}\right) + \frac{1}{2^{\lambda+n}} && \text{by Claim 4.9} \\
&\leq (T+1) \cdot \left(\Pr[\mathsf{Hyb}_{1,4}^{(x^*)}(\mathcal{A}) = 1] + \frac{2}{2^{\lambda+n}}\right) + \frac{1}{2^{\lambda+n}} && \text{by Claim 4.10} \\
&= (T+1) \cdot \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] + \frac{2T+3}{2^{\lambda+n}} && \text{by Eq. (4.3).}
\end{aligned}$$

Thus, Eq. (4.2) holds for the case where $(C, x^*) \notin \mathcal{L}_{\mathsf{SAT}}$. This proves Lemma 4.6. $\square$

**Lemma 4.11.** $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

*Proof.* The conditions for outputting 1 in $\mathsf{Hyb}_3$ are a strict subset of those for outputting 1 in $\mathsf{Hyb}_2$. $\square$

**Lemma 4.12.** *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure, the punctured PRF $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness and $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security, and $\Pi_{\mathsf{OWF}}$ satisfies re-randomization correctness and $(1, \mathsf{negl}(\lambda))$-one-wayness and $(1, 2^{-m^{\varepsilon_m}})$-re-randomization security for constants $\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}}, \varepsilon_m \in (0,1)$ and security parameters $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}, \lambda_{\mathsf{PRF}} = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}, m = (\lambda+n)^{1/\varepsilon_m}$. Then*

$$\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - 2^{-\Omega(\lambda)}.$$

*Proof.* We proceed by defining a sequence of intermediate hybrids for each value of $x^* \in \{0,1\}^n$.

$\mathsf{Hyb}_{3,1}^{(x^*)}$ : Same as $\mathsf{Hyb}_3$ except the challenger computes

- $j^* \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x^*, i^*))$;
- $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j^*))$;
- $k^{(j^*)} \leftarrow \mathsf{F.Puncture}(k, j^*)$;
- $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_{j^*}, (x^*, i^*)))$;
- $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f)$;
- $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\mu})$.

Here, $\mu$ is a bound on the number of bits of randomness the OWF.Rerandomize algorithm takes. The challenger then defines a modified version of GenChall as follows:

$\mathsf{GenChall}''[\mathsf{crs}_f, k_{\mathsf{sel}}, i^*, j^*, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, x^*, y^*, y_{\mathsf{base}}](i, j, x_i)$

**Inputs:** index $i$, selection symbol $j$, statement $x_i$

1: If $j = j^*$ : let $k_j = k_{j^*}$. Otherwise, compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k^{(j^*)}, j))$.

2: If $i = i^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$:

- If $x_i < x^*$: output $\mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (x_i, i)))$.
- If $x_i = x^*$: output $y^*$.
- If $x_i > x^*$: compute $(y, z) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ and output $y$.

Otherwise, compute $(y, z) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ and output $y$.

$\mathsf{Hyb}_{3,2}^{(x^*)}$ : Same as $\mathsf{Hyb}_{3,1}^{(x^*)}$ except the challenger samples $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho})$ instead of computing $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j^*))$.

$\mathsf{Hyb}_{3,3}^{(x^*)}$ : Same as $\mathsf{Hyb}_{3,2}^{(x^*)}$ except the challenger additionally computes

- $k_{j^*}^{(x^*, i^*)} \leftarrow \mathsf{F.Puncture}(k_{j^*}, (x^*, i^*))$
- $k_{\mathsf{rerand}}^{(x^*, i^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{rerand}}, (x^*, i^*))$

and uses the punctured keys in place of $k_{j^*}, k_{\mathsf{rerand}}$.

$\mathsf{Hyb}_{3,4}^{(x^*)}$ : Same as $\mathsf{Hyb}_{3,3}^{(x^*)}$ except the challenger samples $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f)$ instead of computing $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_{j^*}, (x^*, i^*)))$.

$\mathsf{Hyb}_{3,5}^{(x^*)}$ : Same as $\mathsf{Hyb}_{3,4}^{(x^*)}$ except the challenger samples $y^* \leftarrow \mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}})$.

$\mathsf{Hyb}_{3,6}^{(x^*)}$ : Same as $\mathsf{Hyb}_{3,5}^{(x^*)}$ except the challenger computes

$$y^* \leftarrow \mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (x^*, i^*))).$$

$\mathsf{Hyb}_{3,7}^{(x^*)}$ : Same as $\mathsf{Hyb}_{3,6}^{(x^*)}$ except the challenger reverts to using unpunctured keys $k_{j^*}, k_{\mathsf{rerand}}$ instead of punctured keys $k_{j^*}^{(x^*, i^*)}, k_{\mathsf{rerand}}^{(x^*, i^*)}$.

$\mathsf{Hyb}_{3,8}^{(x^*)}$ : Same as $\mathsf{Hyb}_{3,7}^{(x^*)}$ except the challenger reverts to computing $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j^*))$ instead of sampling $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho})$.

We now consider each pair of adjacent distributions.

**Claim 4.13.** *Fix any $x^* \in \{0,1\}^n \setminus \{0^n\}$. Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_{3,1}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,8}^{(x^*-1)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

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

$$|\Pr[\mathsf{Hyb}_{3,1}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,8}^{(x^*-1)}(\mathcal{A}) = 1]| > 1/2^{\lambda+n}.$$

Let $\Lambda_{\mathcal{B}} = \{(\lambda+n)^{1/\varepsilon_{\mathsf{obf}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $n$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set. We define an efficient algorithm $\mathcal{B}$ which plays the $i\mathcal{O}$ security game with $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{obf}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{obf}} < 1$ and $n > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{obf}}$).

**Algorithm $\mathcal{B}[x^*]$**

**Inputs:** $1^{\lambda_{\mathsf{obf}}}$ from $i\mathcal{O}$ challenger, $1^{\lambda}$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^\lambda)$.

2: Sample $i^* \leftarrow [T]$, and $\mathsf{crs}_f, k_{\mathsf{sel}}, k$ as in Setup.

3: Compute $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f)$ and $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\mu})$.

4: Compute $j' \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x^* - 1, i^*))$.

5: Compute $k^{(j')} \leftarrow \mathsf{F.Puncture}(k, j')$, and $k_{j'} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j'))$.

6: Compute $y' \leftarrow \mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (x^* - 1, i^*)))$.

7: Compute $j^* \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x^*, i^*))$.

8: Compute $k^{(j^*)} \leftarrow \mathsf{F.Puncture}(k, j^*)$, and $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j^*))$.

9: Compute $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_{j^*}, (x^*, i^*)))$.

10: Compute $\mathsf{ObfGenSol} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenSol}[\mathsf{crs}_f, C, k_{\mathsf{sel}}, k])$.

11: Construct challenge programs $\mathsf{GenChall}''[\mathsf{crs}_f, k_{\mathsf{sel}}, i^*, j', k^{(j')}, k_{j'}, k_{\mathsf{rerand}}, x^* - 1, y', y_{\mathsf{base}}]$ and $\mathsf{GenChall}''[\mathsf{crs}_f, k_{\mathsf{sel}}, i^*, j^*, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, x^*, y^*, y_{\mathsf{base}}]$ and send to the $i\mathcal{O}$ challenger. The $i\mathcal{O}$ challenger replies with an obfuscated program $\mathsf{ObfGenChall}$.

12: Let $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenSol}, \mathsf{ObfGenChall})$.

13: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

14: Output 1 if and only if $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$.

We first show that

$$V \coloneqq \mathsf{GenChall}''[\mathsf{crs}_f, k_{\mathsf{sel}}, i^*, j', k^{(j')}, k_{j'}, k_{\mathsf{rerand}}, x^* - 1, y', y_{\mathsf{base}}]$$

which is computed as in $\mathsf{Hyb}_{3,8}^{(x^*-1)}$ and

$$V' := \mathsf{GenChall}''[\mathsf{crs}_f, k_{\mathsf{sel}}, i^*, j^*, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, x^*, y^*, y_{\mathsf{base}}]$$

which is computed as in $\mathsf{Hyb}_{3,1}^{(x^*)}$ compute identical functionalities. For a particular input $(i, j, x_i)$ consider the following cases:

**Case 1.** If $i \neq i^*$ or $x_i > x^*$ or $j \neq \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, the two programs behave identically except $V$ may be using hard-coded key $k_{j'} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j'))$ and $V'$ may be using hard-coded key $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; \mathsf{F}(k, j^*))$. Both programs compute

$$(y_i, z_i) = \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$$

and output $y_i$.

**Case 2.** If $i = i^*$ and $x_i < x^* - 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, both programs output

$$y_i = \mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (x_i, i))).$$

**Case 3.** If $i = i^*$ and $x_i = x^* - 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i)) = j'$, the two programs behave identically except $V$ uses the hard-coded value $y' = \mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (x^* - 1, i^*)))$.

**Case 4.** If $i = i^*$ and $x_i = x^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i)) = j^*$, the two programs behave identically except $V'$ uses the hard-coded value $y^*$ where $(y^*, z^*) = \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_{j^*}, (x^*, i^*)))$.

We conclude that the two programs output identical functionality. If the $i\mathcal{O}$ challenger obfuscates $V$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,8}^{(x^*-1)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{3,8}^{(x^*-1)}(\mathcal{A}) = 1]$. If the $i\mathcal{O}$ challenger obfuscates $V'$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,1}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{3,1}^{(x^*)}(\mathcal{A}) = 1]$. Thus by $i\mathcal{O}$ security we have that

$$|\Pr[\mathsf{Hyb}_{3,1}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}'_{3,8}{}^{(x^*-1)}(\mathcal{A}) = 1]| = \mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) \leq 1/2^{\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}} = 1/2^{\lambda+n}. \qquad \square$$

**Claim 4.14.** *Fix $x^* = 0^n$. Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0, 1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_{3,1}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

*Proof.* This follows by an analogous argument as Claim 4.13. $\square$

**Claim 4.15.** *Fix any $x^* \in \{0,1\}^n$. Suppose $\Pi_{\mathsf{PPRF}}$ satisfies $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security for constants $\varepsilon_{\mathsf{PRF}} \in (0, 1)$ and security parameter $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then*

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

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

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

Let $\Lambda_{\mathcal{B}} = \{(\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $n$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set. We define an efficient algorithm $\mathcal{B}$ which plays the puncturing security game with $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{PRF}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{PRF}} < 1$ and $n > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{PRF}}$).

**Algorithm $\mathcal{B}[x^*]$**

**Inputs:** $1^{\lambda_{\mathsf{PRF}}}$ from PPRF challenger, $1^{\lambda}$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Sample $i^* \leftarrow [T]$, and $\mathsf{crs}_f, k_{\mathsf{sel}}$ as in Setup.

3: Compute $j^* \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x^*, i^*))$.

4: Send input length $1^t$, output length $1^\tau$, and punctured point $j^*$ to the PPRF challenger. The PPRF challenger replies with the punctured key $k^{(j^*)}$ and challenge value $r \in \{0,1\}^t$.

5: Compute $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^{\rho}; r)$.

6: Compute $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_{j^*}, (\mathsf{dig}^*, d)))$ and $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f)$.

7: Sample the re-randommization key $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+\lambda}, 1^{\mu})$.

8: Compute $\mathsf{ObfGenSol} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}])$.

9: Compute $\mathsf{ObfGenChall} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenChall}''[\mathsf{crs}_f, k_{\mathsf{sel}}, i^*, j^*, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, x^*, y^*, y_{\mathsf{base}}])$.

10: Let $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenSol}, \mathsf{ObfGenChall})$.

11: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

12: Output 1 if and only if $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$.

If the PPRF challenger samples $r \xleftarrow{R} \{0,1\}^{\rho}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,2}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{3,2}^{(x^*)}(\mathcal{A}) = 1]$. If the PPRF challenger computes $r \leftarrow \mathsf{F}(k, (j^*))$ then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,1}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{3,1}^{(x^*)}(\mathcal{A}) = 1]$. Thus by PPRF security we have that

$$|\Pr[\mathsf{Hyb}_{3,1}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,2}^{(x^*)}(\mathcal{A}) = 1]| = \mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) \leq 1/2^{\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}} = 1/2^{\lambda+n}. \qquad \square$$

**Claim 4.16.** *Fix any $x^* \in \{0,1\}^n$. Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then*

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

*Proof.* This follows by an analogous argument as Claim 4.13. $\square$

**Claim 4.17.** *Fix any $x^* \in \{0,1\}^n$. Suppose $\Pi_{\mathsf{PPRF}}$ satisfies $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security for constants $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and security parameter $\lambda_{\mathsf{PRF}} = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then*

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

*Proof.* This follows by an analogous argument as Claim 4.15. $\square$

**Claim 4.18.** *Fix any $x^* \in \{0,1\}^n$. Suppose $\Pi_{\mathsf{OWF}}$ satisfies $(1, 2^{-m^{\varepsilon_m}})$-re-randomization security for constant $\varepsilon_m \in (0,1)$ and re-randomization parameter $m = (\lambda+n)^{1/\varepsilon_m}$. Then*

$$|\Pr[\mathsf{Hyb}_{3,5}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,4}^{(x^*)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

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

$$|\Pr[\mathsf{Hyb}_{3,5}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,4}^{(x^*)}(\mathcal{A})]| > 1/2^{\lambda+n}.$$

Let $m(\lambda) = (\lambda+n)^{1/\varepsilon_m}$. We define an efficient algorithm $\mathcal{B}$ which plays the re-randomization security game with $m = (\lambda+n)^{1/\varepsilon_m}$ by running $\mathcal{A}$ with security parameter $\lambda$.

**Algorithm $\mathcal{B}[x^*]$**

**Inputs:** $\mathsf{crs}_f \leftarrow \mathsf{OWF.Setup}(1^\lambda, 1^m)$, $y_{\mathsf{base}}, y^*$ from re-randomization challenger

1: Run $(C, T) \leftarrow \mathcal{A}(1^\lambda)$.

2: Sample $i^* \xleftarrow{R} [T]$, and $k, k_{\mathsf{sel}}$ as in Setup.

3: Compute $j^* \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x^*, i^*))$ and $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^\rho)$.

4: Compute $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+\lambda}, 1^\mu)$.

5: Compute the punctured keys $k^{(j^*)} \leftarrow \mathsf{F.Puncture}(k, j^*)$, $k_{j^*}^{(x^*,i^*)} \leftarrow \mathsf{F.Puncture}(k_{j^*}, (x^*, i^*))$ as well as the re-randomization key $k_{\mathsf{rerand}}^{(x^*,i^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{rerand}}, (x^*, i^*))$.

6: Compute $\mathsf{ObfGenSol} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}])$.

7: Compute $\mathsf{ObfGenChall} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenChall}''[\mathsf{crs}_f, k_{\mathsf{sel}}, i^*, j^*, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, x^*, y^*, y_{\mathsf{base}}])$.

8: Let $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenSol}, \mathsf{ObfGenChall})$.

9: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

10: Output 1 if and only if $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$.

If the re-randomization challenger samples $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,4}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{3,4}^{(x^*)}(\mathcal{A}) = 1]$. If the re-randomization challenger computes $(y^*, z^*) \leftarrow \mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}})$ then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{3,5}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{3,5}^{(x^*)}(\mathcal{A}) = 1]$. Thus by re-randomization security we have that

$$|\Pr[\mathsf{Hyb}_{3,5}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,4}^{(x^*)}(\mathcal{A}) = 1]| = \mathsf{RerandAdv}_{\mathcal{B}}(m) \leq 1/2^{m^{\varepsilon}_m} = 1/2^{\lambda+n}. \qquad \square$$

**Claim 4.19.** *Fix any $x^* \in \{0,1\}^n$. Suppose $\Pi_{\mathsf{PPRF}}$ satisfies $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security for constants $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and security parameter $\lambda_{\mathsf{PRF}} = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then*

$$|\Pr[\mathsf{Hyb}_{3,6}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,5}^{(x^*)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

*Proof.* This follows by an analogous argument as Claim 4.15. $\square$

**Claim 4.20.** *Fix any $x^* \in \{0,1\}^n$. Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_{3,7}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,6}^{(x^*)}(\mathcal{A}) = 1]| \le 1/2^{\lambda+n}.$$

*Proof.* This follows by an analogous argument as Claim 4.13. □

**Claim 4.21.** *Fix any $x^* \in \{0,1\}^n$. Suppose $\Pi_{\mathsf{PPRF}}$ satisfies $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security for constants $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and security parameter $\lambda_{\mathsf{PRF}} = (\lambda+n)^{1/\varepsilon_{\mathsf{PRF}}}$. Then*

$$|\Pr[\mathsf{Hyb}_{3,8}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,7}^{(x^*)}(\mathcal{A}) = 1]| \le 1/2^{\lambda+n}.$$

*Proof.* This follows by an analogous argument as Claim 4.15. □

**Claim 4.22.** *Fix $x^* = 1^n$. Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda+n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,8}^{(x^*)}(\mathcal{A}) = 1]| \le 1/2^{\lambda+n}.$$

*Proof.* This follows by an analogous argument as Claim 4.13. □

**Proof of Lemma 4.12.** We now return to the proof of Lemma 4.12. By Claims 4.7 to 4.10, and the triangle inequality, we can now write

$$\begin{aligned}
&\left|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \\
&\quad\le \left|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,8}^{(1^n)}(\mathcal{A}) = 1]\right| \\
&\qquad + \sum_{x\in\{0,1\}^n}\sum_{\ell=2}^{8}\left|\Pr[\mathsf{Hyb}_{3,\ell}^{(x)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,\ell-1}^{(x)}(\mathcal{A}) = 1]\right| \\
&\qquad + \sum_{x\in\{0,1\}^n\setminus\{0^n\}}\left|\Pr[\mathsf{Hyb}_{3,1}^{(x)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,8}^{(x-1)}(\mathcal{A}) = 1]\right| \\
&\qquad + \left|\Pr[\mathsf{Hyb}_{3,1}^{(0^n)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \\
&\quad\le \underbrace{\frac{1}{2^{\Omega(\lambda)}}}_{\text{Claim 4.22}} + \underbrace{2^n\cdot\frac{O(1)}{2^{\lambda+n}}}_{\text{Claims 4.15 to 4.21}} + \underbrace{2^n\cdot\frac{1}{2^{\lambda+n}}}_{\text{Claim 4.13}} + \underbrace{\frac{1}{2^{\lambda+n}}}_{\text{Claim 4.14}},
\end{aligned}$$

which is bounded by a negligible function. Lemma 4.12 holds. □

**Lemma 4.23.** *Suppose that $\Pi_{\mathsf{OWF}}$ satisfies re-randomization correctness, homomorphism, and $(1, \mathsf{negl}(\lambda))$-one-wayness. Then $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] \le \mathsf{negl}(\lambda)$.*

*Proof.* We define an efficient algorithm $\mathcal{B}$ which plays the one-wayness security game with security parameter $\lambda$ and re-randomization parameter $m = m(\lambda, n)$:

| **Algorithm $\mathcal{B}$** |
|---|
| **Inputs:** $\mathsf{crs}_f \leftarrow \mathsf{OWF.Setup}(1^\lambda, 1^m)$ and $y_{\mathsf{base}}$ from the challenger where the challenger samples $(y_{\mathsf{base}}, z_{\mathsf{base}}) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f)$ |

1: Run $(C, T) \leftarrow \mathcal{A}(1^\lambda)$.

2: Sample $i^* \xleftarrow{R} [T]$, and $k, k_{\mathsf{sel}}$ as in Setup.

3: Compute $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^\mu)$.

4: Compute $\mathsf{ObfGenSol} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenSol}[\mathsf{crs}_f, C, k, k_{\mathsf{sel}}])$.

5: Compute $\mathsf{ObfGenChall} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenChall}'[\mathsf{crs}_f, k, k_{\mathsf{sel}}, i^*, k_{\mathsf{rerand}}, y_{\mathsf{base}}])$.

6: Let $\mathsf{crs} = (\mathsf{crs}_f, \mathsf{ObfGenSol}, \mathsf{ObfGenChall})$.

7: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

8: For each $i \in [T]$, compute $(y_i, z_i) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$.

9: Compute $z^* \leftarrow \mathsf{OWF.InHom}(\mathsf{crs}_f, \{z_i\}_{i \in [T] \setminus \{i^*\}})$.

10: Compute $z_{i^*} \leftarrow \mathsf{OWF.InHom}^{-1}(\mathsf{crs}_f, z, z^*)$.

11: Send $z_{\mathsf{base}} \leftarrow \mathsf{OWF.RecoverSolution}(\mathsf{crs}_f, z_{i^*}, \mathsf{F}(k_{\mathsf{rerand}}, (x_{i^*}, i^*)))$ to challenger.

Let $y_{i^*} = \mathsf{OWF.Rerandomize}(\mathsf{crs}_f, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (x_{i^*}, i^*)))$. Similarly, let

$$y^* = \mathsf{OWF.OutHom}(\mathsf{crs}_f, \{\mathsf{ObfGenChall}(i, j, x_i)\}_{i \in [T] \setminus \{i^*\}}).$$

Recall that $\mathsf{Hyb}_4(\mathcal{A}) = 1$ only if $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$. This means

$$\mathsf{OWF.Verify}(\mathsf{crs}_f, \mathsf{OWF.OutHom}(y^*, y_{i^*}), z) = 1.$$

Next note that for all $i \neq i^*$, we have

$$\mathsf{OWF.Verify}(\mathsf{crs}_f, \mathsf{ObfGenChall}(i, j, x_i), z_i) = 1,$$

so by homomorphism of $\Pi_{\mathsf{OWF}}$, we have $\mathsf{OWF.Verify}(\mathsf{crs}_f, y^*, z^*) = 1$. Since $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$, we have $\mathsf{ObfGenChall}(i^*, j, x_{i^*}) = y_{i^*}$. Then by (reverse) homomorphism of $\Pi_{\mathsf{OWF}}$, we have

$$\mathsf{OWF.Verify}(\mathsf{crs}_f, y^*, z^*) = 1.$$

Lastly, by re-randomization correctness of $\Pi_{\mathsf{OWF}}$, we have that $\mathsf{OWF.Verify}(\mathsf{crs}_f, y_{\mathsf{base}}, z_{\mathsf{base}}) = 1$. Combining the above, we conclude

$$\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] \leq \Pr[\mathsf{OWF.Verify}(\mathsf{crs}_f, y_{\mathsf{base}}, z_{\mathsf{base}}) = 1] = \mathsf{OWFAdv}_{\mathcal{B}}(\lambda) \leq \mathsf{negl}(\lambda). \qquad \square$$

**Proof of Theorem 4.4.** Combining Lemmas 4.5, 4.6, 4.11, and 4.12, we have for all sufficiently-large $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] \geq \frac{1}{T(T+1)} \cdot \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - 2^{-\Omega(\lambda)}.$$

By Lemma 4.23, we have $\Pr[\mathsf{Hyb}_4 = 1] = \mathsf{negl}(\lambda)$. We conclude that

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

which remains negligible since $T = \mathsf{poly}(\lambda)$. Finally $\mathsf{Hyb}_0$ corresponds to the real adaptive soundness security game, so Theorem 4.4 holds. $\square$

**Theorem 4.24** (Perfect Zero-Knowledge)**.** *Suppose $i\mathcal{O}$ is correct. Then Construction 4.1 satisfies perfect zero-knowledge.*

*Proof.* We construct the simulator as follows:

- $\mathcal{S}_0(1^\lambda, T, C)$: On input the security parameter $\lambda$, a batch size $T$, and a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, the simulator samples the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, C)$ exactly as in the real scheme. Let $\mathsf{crs}_f, k_{\mathsf{sel}}, k$ be the underlying OWF parameters and PPRF keys sampled in Setup. The simulator outputs the crs along with the state $\mathsf{st} = (\mathsf{crs}_f, k_{\mathsf{sel}}, k)$.
- $\mathcal{S}_1(\mathsf{st}, (x_1, \ldots, x_T))$: On input the state $\mathsf{st} = (\mathsf{crs}_f, k_{\mathsf{sel}}, k)$ and statements $(x_1, \ldots, x_T)$, the simulator computes $j_i \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$ and selects the smallest $j \in [T{+}1]$ such that $j \neq j_i$ for all $i \in [T]$. It then computes $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^\rho; \mathsf{F}(k, j))$ and $(y_i, z_i) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f, \mathsf{F}(k_j, (x_i, i)))$ for all $i$. The simulator outputs $\pi = (j, \mathsf{OWF.InHom}(\mathsf{crs}_f, (z_1, \ldots, z_T)))$.

Take any Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, batch size $T$, and statements $x_1, \ldots, x_T$ and witnesses $w_1, \ldots, w_T$ such that $C(x_i, w_i) = 1$ for all $i \in [T]$. First, observe that the common reference string crs output by $\mathcal{S}_0(1^\lambda, T, C)$ is distributed identically to $\mathsf{Setup}(1^\lambda, T, C)$. It now suffices to consider the proof. By construction, the proof $\pi = (j, z)$ output by $\mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T))$ is obtained by evaluating ObfGenSol on inputs $(i, j, x_i, w_i)$. By correctness of $i\mathcal{O}$ and the definition of GenSol and P, this means that $j$ is the smallest value in $[T+1]$ such that $j \neq \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$ for all $i \in [T]$, and that $z_i$ was generated by computing $(y_i, z_i) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_f; \mathsf{F}(k_j, (x_i, i)))$ for all $i$. Finally, P computes $z = \mathsf{OWF.InHom}(\mathsf{crs}_f, (z_1, \ldots, z_T))$. Thus the proof output by $\mathcal{S}_1(\mathsf{st}, (x_1, \ldots, x_T))$ is distributed identically to $\pi$. □

# 5 Re-randomizable Pseudorandom Generators

In this section, we introduce the notion of a re-randomizable pseudorandom generator (PRG), which is one of the main building blocks we use in our alternative construction of an adaptively-sound fully succinct BARG in Section 6. Then, in Section 5.1, we show how to construct a re-randomizable PRG from DDH.

**Definition 5.1** (Re-randomizable PRG)**.** A re-randomizable pseudorandom generator (PRG) is a tuple of efficient algorithms $\Pi_{\mathsf{RPRG}} = (\mathsf{Setup}, \mathsf{GenSeed}, \mathsf{Eval}, \mathsf{Rerandomize})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^m) \to \mathsf{crs}$ : On input security parameter $1^\lambda$ and re-randomization parameter $1^m$, the setup algorithm outputs a common reference string crs. We assume that the crs contains an implicit description of the seed space $\mathcal{Z}$ and the output space $\mathcal{Y}$, and that elements of $\mathcal{Z}$ can be represented by bit-strings of length $\ell_z$ and elements of $\mathcal{Y}$ can be represented by bit-strings of length $\ell_y$.
- $\mathsf{GenSeed}(\mathsf{crs}) \to z$ : On input common reference string crs, the seed-generation algorithm outputs a seed $z \in \mathcal{Z}$.
- $\mathsf{Eval}(\mathsf{crs}, z) \to y$. On input common reference string crs and seed $z \in \mathcal{Z}$, the deterministic evaluation algorithm outputs $y \in \mathcal{Y}$.
- $\mathsf{Rerandomize}(\mathsf{crs}, y) \to y'$ : On input common reference string crs and instance $y \in \mathcal{Y}$, the re-randomization algorithm outputs a new instance $y' \in \mathcal{Y}$.

We require that $\Pi_{\mathsf{RPRG}}$ satisfy the following properties:

- **Succinctness and expansion.** There exists a fixed polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, m \in \mathbb{N}$, and all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, it holds that the seed length satisfies $\ell_z \leq \mathsf{poly}(\lambda + \log m)$. In addition, the size of the output space satisfies $|\mathcal{Y}| \geq 2^{\Omega(\lambda)} \cdot |\mathcal{Z}|$.

- **Pseudorandomness.** For a security parameter $\lambda$, a re-randomization parameter $m$, and a bit $b \in \{0,1\}$, we define the pseudorandomness security game between an adversary $\mathcal{A}$ and a challenger as follows:

  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$.
  - If $b = 0$, the challenger samples $z \leftarrow \mathsf{GenSeed}(\mathsf{crs})$ and computes $y \leftarrow \mathsf{Eval}(\mathsf{crs}, z)$. If $b = 1$, the challenger samples $y \xleftarrow{R} \mathcal{Y}$.
  - The challenger then sends $(\mathsf{crs}, y)$ to $\mathcal{A}$.
  - $\mathcal{A}$ outputs a bit $b'$, which is the output of the experiment.

  We say that $\Pi_{\mathsf{RPRG}}$ is $(t, \varepsilon)$-pseudorandom if for all polynomials $m = m(\lambda)$ and 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 security parameters $\lambda \geq \lambda_{\mathcal{A}}$, it holds that

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

  in the pseudorandomness security game.

- **Re-randomization correctness.** For all security parameters $\lambda$, all $m = m(\lambda)$ all crs in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, all $y \in \mathcal{Y}$, and all $y'$ in the support of $\mathsf{Rerandomize}(\mathsf{crs}, y)$, either

  - $y, y'$ are both in the image of $\mathsf{Eval}(\mathsf{crs}, \cdot)$; or
  - $y, y'$ are both not in the image of $\mathsf{Eval}(\mathsf{crs}, \cdot)$.

- **Re-randomization security.** For a security parameter $\lambda$, a re-randomization parameter $m$, and a bit $b \in \{0,1\}$, we define the re-randomization security game between an adversary $\mathcal{A}$ and a challenger as follows:

  - The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$, $z_{\mathsf{base}} \leftarrow \mathsf{GenSeed}(\mathsf{crs})$, and computes $y_{\mathsf{base}} \leftarrow \mathsf{Eval}(\mathsf{crs}, z_{\mathsf{base}})$.
  - If $b = 0$, the challenger samples $z^* \leftarrow \mathsf{GenSeed}(\mathsf{crs})$ and computes $y^* \leftarrow \mathsf{Eval}(\mathsf{crs}, z^*)$. If $b = 1$, the challenger samples $y^* \leftarrow \mathsf{Rerandomize}(\mathsf{crs}, y_{\mathsf{base}})$.
  - The challenger then sends $(\mathsf{crs}, y_{\mathsf{base}}, y^*)$ to $\mathcal{A}$.
  - $\mathcal{A}$ outputs a bit $b'$, which is the output of the experiment.

  We say that $\Pi_{\mathsf{RPRG}}$ satisfies $(t, \varepsilon)$-re-randomization security if for all polynomials $m = m(\lambda)$ and 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 security parameters $\lambda \geq \lambda_{\mathcal{A}}$, it holds that

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

  in the re-randomization security game.

## 5.1 Constructing Re-randomizable PRGs

In this section, we show how to construct a re-randomizable PRG from the decisional Diffie-Hellman assumption.

**Construction 5.2** (Re-randomizable PRG)**.** Let GroupGen be a prime-order group generator. We construct a re-randomizable PRG $\Pi_{\mathsf{RPRG}} = (\mathsf{Setup}, \mathsf{GenSeed}, \mathsf{Eval}, \mathsf{Rerandomize})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^m)$: On input security parameter $1^\lambda$ and re-randomization parameter $1^m$, the setup algorithm samples $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$, $x \xleftarrow{R} \mathbb{Z}_p^*$, and outputs $\mathsf{crs} = (\mathbb{G}, p, g, h)$ where $h = g^x$. The seed space is $\mathcal{Z} = \mathbb{Z}_p^*$ and the output space is $\mathcal{Y} = (\mathbb{G} \setminus \{1\})^2$.
- $\mathsf{GenSeed}(\mathsf{crs})$: On input common reference string $\mathsf{crs} = (\mathbb{G}, p, g, h)$, the seed generation algorithm samples and outputs $z \xleftarrow{R} \mathbb{Z}_p^*$.
- $\mathsf{Eval}(\mathsf{crs}, z)$: On input common reference string $\mathsf{crs} = (\mathbb{G}, p, g, h)$ and seed $z \in \mathbb{Z}_p^*$, the evaluation algorithm outputs $(g^z, h^z) \in \mathcal{Y}$.
- $\mathsf{Rerandomize}(\mathsf{crs}, y)$: On input common reference string crs and instance $y = (y_1, y_2)$, the re-randomization algorithm samples $r \xleftarrow{R} \mathbb{Z}_p^*$ and outputs $(y_1^r, y_2^r) \in \mathcal{Y}$.

**Theorem 5.3** (Succinctness and Expansion)**.** *Construction 5.2 satisfies succinctness and expansion.*

*Proof.* For $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$, we have that $p$ is a $\lambda$-bit prime. Thus a seed $z \in \mathbb{Z}_p^*$ can be described by a string of length at most $\ell_z = \lambda$. Next, an instance $y = (g^z, h^z)$ consists of two group elements and thus $|\mathcal{Y}| = (p-1)^2 > 2^{\Omega(\lambda)} \cdot |\mathbb{Z}_p^*|$, since $p = 2^{\Omega(\lambda)}$. □

**Theorem 5.4** (Pseudorandomness)**.** *Suppose the decisional Diffie-Hellman assumption holds with respect to* GroupGen*. Then Construction 5.2 satisfies pseudorandomness.*

*Proof.* Let $\mathcal{A}$ be an efficient adversary for the pseudorandomness game against Construction 5.2. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the DDH problem:

**Algorithm $\mathcal{B}$**

**Inputs:** $((\mathbb{G}, p, g), g^\alpha, g^\beta, g^\gamma)$ from the DDH challenger

1: Let $\mathsf{crs} = (\mathbb{G}, p, g, h)$ where $h = g^\alpha$ and $y = (g^\beta, g^\gamma)$.

2: Run $b' \leftarrow \mathcal{A}(\mathsf{crs}, y)$.

3: Send $b'$ to challenger.

Note that if $b = 0$ (i.e., $\gamma = \alpha\beta$ for uniform $\alpha, \beta \xleftarrow{R} \mathbb{Z}_p^*$), then $g^\gamma = g^{\alpha\beta} = h^\beta$ so $y = (g^\beta, h^\beta)$. If $b = 1$ (i.e., $\beta, \gamma \xleftarrow{R} \mathbb{Z}_p^*$), then $y = (g^\beta, g^\gamma)$ is uniform over $(\mathbb{G} \setminus \{1\})^2$. Thus,

$$\mathsf{PRGAdv}_{\mathcal{A}}(\lambda) \le \mathsf{DDHAdv}_{\mathcal{B}}(\lambda) \le \mathsf{negl}(\lambda).$$ □

**Theorem 5.5** (Re-randomization Correctness)**.** *Construction 5.2 satisfies re-randomization correctness.*

*Proof.* Take any $\lambda, m \in \mathbb{N}$, any $\mathsf{crs} = (\mathbb{G}, p, g, h = g^x)$ in the support of $\mathsf{Setup}(1^\lambda, 1^m)$, any $y \in \mathcal{Y}$, and any $y'$ in the support of $\mathsf{Rerandomize}(\mathsf{crs}, y)$. We show that $y$ is in the support of $\mathsf{Eval}(\mathsf{crs}, \cdot)$ if and only if $y'$ is in the support of $\mathsf{Eval}(\mathsf{crs}, \cdot)$.

- For the forward direction, suppose $y = (y_1, y_2) = \mathsf{Eval}(\mathsf{crs}, z)$ for some seed $z \in \mathbb{Z}_p^*$. Then $y_1 = g^z$ and $y_2 = h^z$. We have that for some $r \in \mathbb{Z}_p^*$, $y' = (y_1^r, y_2^r) = (g^{rz}, h^{rz})$, so $y' = \mathsf{Eval}(\mathsf{crs}, rz)$.
- For the reverse direction, suppose $y' = (y'_1, y'_2) = \mathsf{Eval}(\mathsf{crs}, z)$ for some $z \in \mathbb{Z}_p^*$. Then $y'_1 = g^z$ and $y'_2 = h^z$. Since $y' = \mathsf{Rerandomize}(\mathsf{crs}, y)$, and by construction of Rerandomize, there exists $r \in \mathbb{Z}_p^*$ such that
$$y = \left((y'_1)^{r^{-1}}, (y'_2)^{r^{-1}}\right) = \left(g^{r^{-1}z}, h^{r^{-1}z}\right).$$
Thus $y = \mathsf{Eval}(\mathsf{crs}, r^{-1}z)$, as required.

We conclude that either $y, y'$ are both in the image of $\mathsf{Eval}(\mathsf{crs}, \cdot)$ or they are both not in the image of $\mathsf{Eval}(\mathsf{crs}, \cdot)$. □

**Theorem 5.6** (Re-randomization Security)**.** *Construction 5.2 satisfies perfect re-randomizable security. For all polynomials $m = m(\lambda)$ and all adversaries $\mathcal{A}$, $\mathsf{RerandAdv}_{\mathcal{A}}(\lambda) = 0$.*

*Proof.* Take any polynomial $m = m(\lambda)$. Let $\mathsf{crs} = (\mathbb{G}, p, g, h) \leftarrow \mathsf{Setup}(1^\lambda, 1^m)$ where $h = g^x$. By construction of GenSeed, we have that a fresh instance $\mathsf{Eval}(\mathsf{crs}, z^*) = (g^{z^*}, h^{z^*})$ is uniformly distributed over the set $\{(g^z, h^z) \mid z \in \mathbb{Z}_p^*\}$. By construction of Rerandomize, we have that a re-randomized instance $y' = (y_1^r, y_2^r) = (g^{rz}, h^{rz})$ is still uniformly distributed over the set $\{(g^z, h^z) \mid z \in \mathbb{Z}_p^*\}$, since $rz$ is uniformly distributed over $\mathbb{Z}_p^*$. Note that this proof uses the fact that the exponents are sampled from $\mathbb{Z}_p^*$ (rather than $\mathbb{Z}_p$). □

# 6 SNARG for Batch NP from Re-randomizable PRGs

In this section, we show how to construct a fully succinct SNARG for batch NP using indistinguishability obfuscation together with a re-randomizable PRG. As described in Section 1.1, this construction builds on the chaining approach from [GSWW22]. We give the construction below:

**Construction 6.1** (Adaptive Batch Argument for NP)**.** Let $\lambda$ be a security parameter. We construct a BARG scheme that supports NP languages with an arbitrary polynomial number $T = T(\lambda) < 2^\lambda$ of instances of length $n = n(\lambda)$. Our construction will leverage *sub-exponential* hardness of the following primitives (except for pseudorandomness of the re-randomizable PRG $\Pi_{\mathsf{RPRG}}$). Our construction relies on the following primitives:

- Let $i\mathcal{O}$ be an indistinguishability obfuscator for Boolean circuits.
- Let $\Pi_{\mathsf{SEH}} = (\mathsf{H.Setup}, \mathsf{H.Hash}, \mathsf{H.Open}, \mathsf{H.Verify}, \mathsf{H.Extract})$ be a somewhere-extractable hash family.
- Let $\Pi_{\mathsf{PPRF}} = (\mathsf{F.Setup}, \mathsf{F.Eval}, \mathsf{F.Puncture})$ be a puncturable PRF. For a key $k$ and an input $x$, we will write $\mathsf{F}(k, x)$ to denote $\mathsf{F.Eval}(k, x)$.
- Let $\Pi_{\mathsf{RPRG}} = (\mathsf{PRG.Setup}, \mathsf{PRG.GenSeed}, \mathsf{PRG.Eval}, \mathsf{PRG.Rerandomize})$ be a re-randomizable PRG.

We will describe how to define the polynomials $\lambda_{\mathsf{SEH}}$, $\lambda_{\mathsf{obf}}$, $\lambda_{\mathsf{PRF}}$, and $m$ in the security analysis. We construct a fully succinct non-interactive batch argument $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ for NP as follows:

- $\mathsf{Setup}(1^\lambda, T, C)$: On input security parameter $1^\lambda$, batch size $T$, and Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, the setup algorithm does the following:
  - Sample PRG parameters $\mathsf{crs}_{\mathsf{G}} \leftarrow \mathsf{PRG.Setup}(1^\lambda, 1^m)$.

- Sample an SEH hash key $\mathsf{hk} \leftarrow \mathsf{H.Setup}(1^{\lambda_{\mathsf{SEH}}}, 1^n)$.
- Let $t = \log(T+1)$. Let $n'$ be the output length of $\mathsf{H.Hash}(\mathsf{hk}, \cdot)$. Let $\rho$ be a bound on the number of bits of randomness the $\mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}})$ algorithm takes. Let $\tau$ be the number of bits of randomness the $\mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho})$ algorithm takes.
- Sample a "selector" PPRF key $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^t)$.
- Sample a "key generator" PPRF key $k \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^t, 1^{\tau})$.
- Define the VerProof program with the PRG parameters $\mathsf{crs}_{\mathsf{G}}$ and PPRF key $k$ hard-coded:

$\mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k](i, j, \mathsf{dig}, z_i)$

**Inputs:** index $i$, selection symbol $j$, hash value dig, proof $z_i$

1: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.
2: Compute $z \leftarrow \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i)))$.
3: Output 1 if $\mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, z_i) = \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, z)$ and 0 otherwise.

- Define the AggProof program (which has the code for VerProof replicated inside) with the circuit $C$, PRG parameters $\mathsf{crs}_{\mathsf{G}}$, SEH hash key hk, and PPRF keys $k_{\mathsf{sel}}, k$ hard-coded:

$\mathsf{AggProof}[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k](i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$

**Inputs:** index $i$, selection symbol $j$, hash value dig, statement $x_i$, witness $w_i$, opening $\sigma_i$, prior proof $z_{i-1}$

1: If $C(x_i, w_i) = 0$, output $\perp$.
2: If $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i, \sigma_i) = 0$, output $\perp$.
3: If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.
4: If $i \neq 1$ and $\mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k](i-1, j, \mathsf{dig}, z_{i-1}) = 0$, output $\perp$.
5: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.
6: Output $z_i = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i)))$.

- Let $s = s(\lambda, n, |C|)$ be the maximum size of the AggProof and VerProof programs as well as those appearing in the security analysis.
- Construct the obfuscated programs

$$\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{AggProof}[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k])$$

and

$$\mathsf{ObfVerProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k]).$$

- Output $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

- $\mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T))$: On input $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$, statements $x_1, \ldots, x_T \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_T \in \{0,1\}^v$, the prover algorithm proceeds as follows:
  - Compute $\mathsf{dig} \leftarrow \mathsf{H.Hash}(\mathsf{hk}, (x_1, \ldots, x_T))$.
  - Initialize $i = 1$, $j = 1$ and $z_0 = \emptyset$.

- While $i \le T$ :
  - Compute $\sigma_i \leftarrow \mathsf{H.Open}(\mathsf{hk}, (x_1, \ldots, x_T), i)$.
  - Compute $z_i \leftarrow \mathsf{ObfAggProof}(i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$.
  - If $z_i = \bot$, set $i = 1$ and $j = j + 1$. Otherwise, set $i = i + 1$.
- Output $\pi = (j, z_T)$.

- $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi)$: On input $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$, statements $x_1, \ldots, x_T \in \{0,1\}^n$, and the proof $\pi = (j, z_T)$, the verification algorithm proceeds as follows:
  - If $j \notin [T+1]$, then output 0.
  - Otherwise, compute $\mathsf{dig} \leftarrow \mathsf{H.Hash}(\mathsf{hk}, (x_1, \ldots, x_T))$ and output $\mathsf{ObfVerProof}(T, j, \mathsf{dig}, z_T)$.

**Theorem 6.2** (Completeness)**.** *Suppose $i\mathcal{O}$ is correct and $\Pi_{\mathsf{SEH}}$ satisfies opening completeness. Then Construction 6.1 is complete.*

*Proof.* Take any security parameter $\lambda \in \mathbb{N}$, any Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, any $T \le 2^\lambda$, and any collection of statements $(x_1, \ldots, x_T)$ and witnesses $(w_1, \ldots, w_T)$ where $C(x_i, w_i) = 1$ for all $i \in [T]$. Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof}) \leftarrow \mathsf{Setup}(1^\lambda, C, T)$ and $\pi = (j, z_T) \leftarrow \mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T))$. Consider the output of $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi)$:

- By construction, ObfAggProof is an obfuscation of the program $\mathsf{AggProof}[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k]$, where

$$\begin{aligned} \mathsf{crs}_{\mathsf{G}} &\leftarrow \mathsf{G.Setup}(1^\lambda, 1^m) \\ \mathsf{hk} &\leftarrow \mathsf{H.Setup}(1^{\lambda_{\mathsf{SEH}}}, 1^n) \\ k_{\mathsf{sel}} &\leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^t) \\ k &\leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^t, 1^\tau). \end{aligned}$$

- Let $\mathsf{dig} = \mathsf{H.Hash}(\mathsf{hk}, (x_1, \ldots, x_T))$ and $\sigma_i \leftarrow \mathsf{H.Open}(\mathsf{hk}, (x_1, \ldots, x_T), i)$ for all $i \in [T]$. By completeness of $\Pi_{\mathsf{SEH}}$, we have that for all $i \in [T]$,

$$\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, i, x_i, \sigma_i) = 1.$$

- Let $j^* \in [T+1]$ be the smallest index where $\mathsf{F}(k_{\mathsf{sel}}, (x_i, i)) \neq j^*$ for all $i \in [T]$. By correctness of $i\mathcal{O}$ and the definition of AggProof, it must be the case that $j \ge j^*$.

- Let $z_1^*, \ldots, z_T^*$ be the intermediate proofs obtained by P through evaluating ObfAggProof on inputs $(i, j^*, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$. By construction, it follows that $C(x_i, w_i) = 1$ and $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i, \sigma_i) = 1$ and $j^* \neq \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$.

- We now claim that by correctness of $i\mathcal{O}$ and the definition of AggProof, this means that for all $i \in [T]$, it holds that

$$\mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k](i, j^*, \mathsf{dig}, z_i^*) = 1. \tag{6.1}$$

  Consider the case where $i = 1$. In this case, AggProof outputs

$$z_1 = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; F(k_{j^*}, (\mathsf{dig}, 1))).$$

Correspondingly, this means

$$\mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, z_1) = \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; F(k_{j^*}, (\mathsf{dig}, 1)))),$$

which precisely coincides with the verification condition $\mathsf{VerProof}(1, j^*, \mathsf{dig}, z_1^*)$. Thus, Eq. (6.1) holds when $i = 1$. For the inductive step, take $i > 1$ and suppose $\mathsf{VerProof}(i-1, j, \mathsf{dig}, z_{i-1}^*) = 1$. Then, AggProof outputs

$$z_i^* = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_{j^*}, (\mathsf{dig}, i))).$$

As in the base case, this means $\mathsf{VerProof}(i, j^*, \mathsf{dig}, z_i^*) = 1$ and so by induction on $i$, we have that Eq. (6.1) holds for all $i \in [T]$. In this case, algorithm P outputs the proof $\pi = (j^*, z_T^*) = (j, z_T)$.

- By construction, $\mathsf{ObfVerProof}$ is an obfuscation of the program $\mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k]$. The verification algorithm V computes $\mathsf{dig} = \mathsf{H.Hash}(\mathsf{hk}, (x_1, \ldots, x_T))$ and outputs $b \leftarrow \mathsf{ObfVerProof}(T, j, \mathsf{dig}, z_T)$. By correctness of $i\mathcal{O}$, the definition of $\mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k]$, and Eq. (6.1), $b = 1$ and completeness holds. □

**Theorem 6.3** (Succinctness). *Suppose $\Pi_{\mathsf{SEH}}$ and $\Pi_{\mathsf{RPRG}}$ are succinct. Then Construction 6.1 is succinct.*

*Proof.* A proof $(j, z_T)$ in Construction 6.1 consists of a selection symbol $j \in [T+1]$ and a PRG seed $z_T$. By construction, there is a fixed polynomial $\mathsf{poly}(\cdot)$ such that $|z| \le \mathsf{poly}(\lambda + \log m)$. In Construction 6.1, $m(\lambda, n')$ is a fixed polynomial in the security parameter $\lambda$ and $n'$, which is the output length of $\mathsf{H.Hash}(\mathsf{hk}, \cdot)$ for $\mathsf{hk} \leftarrow \mathsf{H.Setup}(1^{\lambda_{\mathsf{SEH}}}, 1^n)$ where $\lambda_{\mathsf{SEH}}$ is a fixed polynomial in the witness length $v$ and $\lambda$. By succinctness of $\Pi_{\mathsf{SEH}}$, we have that $n'$ is a fixed polynomial in $\lambda$, $n$, and $v$. The statement length and witness length are always upper-bounded by the circuit size, so it follows that $|\pi| \le \mathsf{poly}(\lambda + \log |C|) + \log T$. □

**Theorem 6.4** (Adaptive Soundness). *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure, $\Pi_{\mathsf{SEH}}$ satisfies statistical binding and $(2^{\lambda_{\mathsf{SEH}}^{\varepsilon_{\mathsf{SEH}}}}, \mathsf{negl}(\lambda_{\mathsf{SEH}}))$-index hiding security, $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness and $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security, $\Pi_{\mathsf{RPRG}}$ is expanding and $(1, \mathsf{negl}(\lambda))$-pseudorandomness and $(1, 2^{-m^{\varepsilon_m}})$-re-randomization security for constants $(\varepsilon_{\mathsf{SEH}}, \varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}}, \varepsilon_m) \in (0,1)$ and security parameters $\lambda_{\mathsf{SEH}} = (v + \omega(\log\lambda))^{1/\varepsilon_{\mathsf{SEH}}}$, $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$, $\lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$, $m = (\lambda + n')^{1/\varepsilon_m}$ where $n'$ is the length of $\mathsf{H.Hash}(\mathsf{H.Setup}(1^{\lambda_{\mathsf{SEH}}}, 1^n), \cdot)$. Then Construction 6.1 satisfies adaptive soundness.*

*Proof.* Let $\mathcal{A}$ be an efficient adversary that succeeds in the adaptive soundness game against Construction 6.1 with (non-negligible) probability $\varepsilon(\lambda)$. We first claim that without loss of generality, we can assume that for every security parameter $\lambda$, $\mathcal{A}$ always outputs a circuit $C$ with statements of a fixed length $n = n(\lambda)$ and witnesses of a fixed length $v = v(\lambda)$ and a fixed batch size $T = T(\lambda)$. Formally, since $\mathcal{A}$ is a polynomial-time algorithm, $\mathcal{A}(1^\lambda)$ outputs a Boolean circuit of size at most $s_{\max}(\lambda) = \mathsf{poly}(\lambda)$ and a maximum batch size $T_{\max}(\lambda) = \mathsf{poly}(\lambda)$. This in turn defines maximum statement and witness lengths $n_{\max}(\lambda), v_{\max}(\lambda) \le s_{\max}(\lambda)$. In an execution of the adaptive soundness game, let $\mathsf{E}_{n',v',T'}$ be the event that $\mathcal{A}$ outputs a circuit $C$ with statements of length $n'$ and witnesses of length $v'$ and batch size $T'$. Then

$$\Pr[\mathcal{A} \text{ wins the soundness game}] = \sum_{\substack{n' \in [n_{\max}] \\ v' \in [v_{\max}] \\ T' \in [T_{\max}]}} \Pr[\mathcal{A} \text{ wins the soundness game} \wedge \mathsf{E}_{n',v',T'}].$$

Thus there must exist some $(n, v, T) \in [n_{\max}] \times [v_{\max}] \times [T_{\max}]$ such that such that

$$\Pr[\mathcal{A}(1^\lambda) \text{ wins the soundness game} \wedge \mathsf{E}_{n,v,T}] \ge \frac{\varepsilon(\lambda)}{n_{\max} \cdot v_{\max} \cdot T_{\max}}.$$

For each security parameter $\lambda$, define $n = n(\lambda)$, $v = v(\lambda)$, and $T = T(\lambda)$ to be the smallest values such that the above equation holds. We now construct a new (non-uniform) adversary $\mathcal{A}'$ that functions as a wrapper around $\mathcal{A}$, but only outputs circuits with fixed statement and witness lengths and a fixed batch size. Namely, $\mathcal{A}'$ takes as input the security parameter $1^\lambda$ and the non-uniform advice $n = n(\lambda), v = v(\lambda), T = T(\lambda)$. $\mathcal{A}'$ runs $(C', T') \leftarrow \mathcal{A}(1^\lambda)$. If $C'$ does not have statements of length $n$ and witnesses of length $v$ or $T' \neq T$ , then $\mathcal{A}'$ aborts. Otherwise, $\mathcal{A}'$ simply follows the behavior of $\mathcal{A}$ (and outputs whatever $\mathcal{A}$ outputs). By construction,

$$\begin{aligned}\varepsilon' &= \Pr[\mathcal{A}'(1^\lambda) \text{ wins the soundness game}] \\ &= \Pr[\mathcal{A}(1^\lambda) \text{ wins the soundness game} \wedge \mathsf{E}_{n,v,T}] \geq \frac{\varepsilon(\lambda)}{n_{\max} \cdot v_{\max} \cdot T_{\max}}.\end{aligned}$$

Thus $\mathcal{A}'$ still has a non-negligible success probability $\varepsilon'$ in the soundness game. Furthermore, we note that without loss of generality there exists some index $i^* = i^*(\lambda) \in [T]$ such that $\mathcal{A}'$ cheats on index $i^*$, with probability at least $\varepsilon'/T$. In other words, in the adaptive soundness game, algorithm $\mathcal{A}'$ outputs a batch of statements $\vec{x} = (x_1, \ldots, x_T)$ and an accepting proof $\pi$, and moreover, instance $x_{i^*}$ is false. Since $T = \mathsf{poly}(\lambda)$, and $\varepsilon'$ is non-negligible, we have that $\varepsilon'/T$ remains non-negligible. Thus, for the remainder of this proof, we will declare the adversary successful if it wins the adaptive soundness game by outputting an accepting proof on $\vec{x} = (x_1, \ldots, x_T)$ where $x_{i^*}$ is a false instance for a *fixed* index $i^*$. As argued here, every adversary that breaks adaptive soundness implies an adversary that succeeds in this "fixed-index" variant for some index $i^*$. The index $i^*$ will be provided as non-uniform advice to all of our reduction algorithms. We now define our sequence of hybrid experiments.

$\mathsf{Hyb}_0$ : This is the real adaptive soundness experiment with a fixed index $i^*$.

- Adversary $\mathcal{A}$, on input $1^\lambda$, starts by outputting a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, and the batch size $T$.
- The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, C)$ and gives crs to $\mathcal{A}$.
- Adversary $\mathcal{A}$ outputs statements $\vec{x} = (x_1, \ldots, x_T)$ and a proof $\pi = (j, z)$.
- The challenger outputs 1 if and only if $(C, x_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$ and $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$.

$\mathsf{Hyb}_1$ : Same as $\mathsf{Hyb}_0$ except the challenger samples $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{H.SetupTD}(1^\lambda, 1^n, i^*)$.

$\mathsf{Hyb}_2$ : Same as $\mathsf{Hyb}_1$ except the challenger additionally checks that $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$. Specifically, if $j \neq \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$, then the challenger outputs 0.

$\mathsf{Hyb}_3$ : Same as $\mathsf{Hyb}_2$ except the challenger stops checking that $(C, x_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$.

$\mathsf{Hyb}_4$ : Same as $\mathsf{Hyb}_3$ except the challenger defines a modified version of AggProof which additionally has td, $i^*$ hard-coded as follows:

$\mathsf{AggProof}'[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, \mathsf{td}, i^*](i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$

**Inputs:** index $i$, selection symbol $j$, hash value dig, statement $x_i$, witness $w_i$, opening $\sigma_i$, prior proof $z_{i-1}$

1: If $C(x_i, w_i) = 0$, output $\perp$.

2: If $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i, \sigma_i) = 0$, output $\perp$.

3: If $i = i^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$, output $\perp$.

4: If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.

5: If $i \neq 1$ and $\mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k](i-1, j, \mathsf{dig}, z_{i-1}) = 0$, output $\perp$.

6: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.

7: Output $z_i = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i)))$.

When constructing the CRS, the challenger now computes

$$\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{AggProof}'[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, \mathsf{td}, i^*]).$$

$\mathsf{Hyb}_{5,d}$ : For $d \in [T]$ : $\mathsf{Hyb}_{5,d}$ is the same as $\mathsf{Hyb}_4$ except the challenger defines a modified version of VerProof which additionally has $k_{\mathsf{sel}}$, td, $i^*$, and $d$ hard-coded as follows:

$\mathsf{VerProof}'[\mathsf{crs}_{\mathsf{G}}, k, k_{\mathsf{sel}}, \mathsf{td}, i^*, d](i, j, \mathsf{dig}, z_i)$

**Inputs:** index $i$, selection symbol $j$, hash value dig, proof $z_i$

1: If $i^* \leq i \leq d$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$, output 0.

2: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.

3: Compute $z = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i)))$.

4: Output 1 if $\mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, z_i) = \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, z)$ and 0 otherwise.

The challenger also uses $\mathsf{VerProof}'[\mathsf{crs}_{\mathsf{G}}, k, k_{\mathsf{sel}}, \mathsf{td}, i^*, d]$ in place of $\mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k]$ in the proof aggregation program. Specifically, in this experiment, the challenger defines a modified version of AggProof as follows:

$\mathsf{AggProof}'[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, \mathsf{td}, i^*, d](i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$

**Inputs:** index $i$, selection symbol $j$, hash value dig, statement $x_i$, witness $w_i$, opening $\sigma_i$, prior proof $z_{i-1}$

1: If $C(x_i, w_i) = 0$, output $\perp$.

2: If $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i, \sigma_i) = 0$, output $\perp$.

3: If $i = i^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$, output $\perp$.

4: If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.

5: If $i \neq 1$ and $\mathsf{VerProof}'[\mathsf{crs}_{\mathsf{G}}, k, k_{\mathsf{sel}}, \mathsf{td}, i^*, d](i-1, j, \mathsf{dig}, z_{i-1}) = 0$, output $\perp$.

6: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.

7: Output $z_i = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i)))$.

When constructing the CRS, the challenger now computes

$$\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{AggProof}'[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, \mathsf{td}, i^*, d])$$

and

$$\mathsf{ObfVerProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}'[\mathsf{crs}_{\mathsf{G}}, k, k_{\mathsf{sel}}, \mathsf{td}, i^*, d]).$$

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

**Lemma 6.5.** *Suppose* $\Pi_{\mathsf{SEH}}$ *satisfies* $(2^{\lambda_{\mathsf{SEH}}^{\varepsilon_{\mathsf{SEH}}}}, \mathsf{negl}(\lambda_{\mathsf{SEH}}))$*-index hiding security for constant* $\varepsilon_{\mathsf{SEH}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{SEH}} = (v(\lambda) + \omega(\log\lambda))^{1/\varepsilon_{\mathsf{SEH}}}$*. Then*

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

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

$$|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A})]| > \delta(\lambda).$$

Let $\Lambda_{\mathcal{B}} = \{(v(\lambda) + \omega(\log\lambda))^{1/\varepsilon_{\mathsf{SEH}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $v$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set. We define a $2^{\lambda_{\mathsf{SEH}}^{\varepsilon_{\mathsf{SEH}}}}$-time algorithm $\mathcal{B}$ which plays the index-hiding security game with $\lambda_{\mathsf{SEH}} = (v + \omega(\log\lambda))^{1/\varepsilon_{\mathsf{SEH}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{SEH}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{SEH}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{SEH}} < 1$ and $v > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{SEH}}$).

**Algorithm $\mathcal{B}$**

**Inputs:** $1^{\lambda_{\mathsf{SEH}}}$ from index-hiding challenger, $1^{\lambda}$ and $i^*$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Send the block size $1^n$ and the index $i^*$ to the index-hiding challenger. The index-hiding challenger replies with the hash key $\mathsf{hk}$.

3: Sample $\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, k$ and compute $\mathsf{ObfAggProof}, \mathsf{ObfVerProof}$ as in Setup.

4: Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

5: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

6: Output 1 if and only if $(C, x_{i^*}) \notin \mathcal{L}_{\mathsf{SAT}}$ and $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$.

Algorithm $\mathcal{B}$ has to check all possible witnesses for $x_{i^*}$, so it runs in time $2^v \cdot \mathsf{poly}(\lambda) \leq 2^{v+\omega(\log\lambda)} = 2^{\lambda_{\mathsf{SEH}}^{\varepsilon_{\mathsf{SEH}}}}$. If the index-hiding challenger sampled $\mathsf{hk} \leftarrow \mathsf{H.Setup}(1^{\lambda_{\mathsf{SEH}}}, 1^n)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_0$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$. If the index-hiding challenger sampled $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{H.SetupTD}(1^{\lambda_{\mathsf{SEH}}}, 1^n, i^*)$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_1$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$. Thus by index-hiding security we have that

$$|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]| = \mathsf{SEHAdv}_{\mathcal{B}}(\lambda_{\mathsf{SEH}}) \leq \mathsf{negl}(\lambda_{\mathsf{SEH}}) = \mathsf{negl}(\lambda)$$

for sufficiently large $\lambda_{\mathsf{SEH}}$. □

**Lemma 6.6.** *Suppose* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$*-secure,* $\Pi_{\mathsf{SEH}}$ *satisfies statistical binding, and* $\Pi_{\mathsf{PPRF}}$ *satisfies punctured correctness and* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-puncturing security for constants* $(\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}}) \in (0,1)$ *and security parameters* $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}, \lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$*. Then*

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \frac{1}{T+1}\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - 2^{-\Omega(\lambda)}.$$

*Proof.* Consider an execution of $\mathsf{Hyb}_1$ or $\mathsf{Hyb}_2$. For a fixed $x^* \in \{0,1\}^n$, let $\mathsf{E}_{x^*}$ be the event that $\mathcal{A}$ outputs $(x_1, \ldots, x_T)$ such that $x_{i^*} = x^*$. By definition, we can now write

$$\begin{aligned} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] &= \sum_{x^* \in \{0,1\}^n} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] \\ \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] &= \sum_{x^* \in \{0,1\}^n} \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}]. \end{aligned} \tag{6.2}$$

To prove the lemma, we show that for all $x^* \in \{0,1\}^n$,

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] \geq \frac{1}{T+1} \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] - \frac{O(T)}{2^{\lambda+n}}. \tag{6.3}$$

By a similar argument as in the proof of Lemma 4.6, this suffices to prove the claim. Fix any $x^* \in \{0,1\}^n$. If $(C, x^*) \in \mathcal{L}_{\mathsf{SAT}}$, then

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] = 0 = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}]. \tag{6.4}$$

In this case, Eq. (6.3) holds. Thus, we only need to consider the case where $(C, x^*) \notin \mathcal{L}_{\mathsf{SAT}}$. We proceed by defining a sequence of intermediate hybrids.

$\mathsf{Hyb}_{1,0}^{(x^*)}$ : Same as $\mathsf{Hyb}_1$ except the challenger additionally checks that $x_{i^*} = x^*$ (i.e., that $\mathsf{E}_{x^*}$ occurred).

$\mathsf{Hyb}_{1,1}^{(x^*)}$ : Same as $\mathsf{Hyb}_{1,0}^{(x^*)}$ except the challenger does the following. It computes

$$k_{\mathsf{sel}}^{(x^*,i^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{sel}}, (x^*, i^*))$$

and defines a modified version of AggProof which additionally has $x^*$ hard-coded as follows:

$\mathsf{AggProof}_1[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k, \mathsf{td}, i^*, k_{\mathsf{sel}}^{(x^*,i^*)}, x^*](i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$

**Inputs:** index $i$, selection symbol $j$, hash value dig, statement $x_i$, witness $w_i$, opening $\sigma_i$, proof $z_{i-1}$

1: If $i = i^*$ and $\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}) = x^*$, output $\perp$.
2: If $C(x_i, w_i) = 0$, output $\perp$.
3: If $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i, \sigma_i) = 0$, output $\perp$.
4: If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.
5: If $i \neq 1$ and $\mathsf{VerProof}[\mathsf{crs}_{\mathsf{G}}, k](i-1, j, \mathsf{dig}, z_{i-1}) = 0$, output $\perp$.
6: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.
7: Output $z_i = \mathsf{F}(k_j, (\mathsf{dig}, i))$.

$\mathsf{Hyb}_{1,2}^{(x^*)}$ : Same as $\mathsf{Hyb}_{1,1}^{(x^*)}$ except that after $\mathcal{A}$ outputs $(\vec{x}, \pi)$ where $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$), the challenger samples $j' \xleftarrow{R} [T+1]$ and additionally checks that $j = j'$.

$\mathsf{Hyb}_{1,3}^{(x^*)}$ : Same as $\mathsf{Hyb}_{1,2}^{(x^*)}$ except the challenger instead checks that $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$.

$\mathsf{Hyb}_{1,4}^{(x^*)}$ : Same as $\mathsf{Hyb}_{1,3}^{(x^*)}$ except the challenger reverts to obfuscating AggProof instead of $\mathsf{AggProof}_1$.

By definition,

$$\begin{aligned}\Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A}) = 1] &= \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] \\ \Pr[\mathsf{Hyb}_{1,4}^{(x^*)}(\mathcal{A}) = 1] &= \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}].\end{aligned} \tag{6.5}$$

We now consider each pair of adjacent distributions.

**Claim 6.7.** *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then*

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

*Proof.* We first show that $\mathsf{AggProof}[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k]$ in $\mathsf{Hyb}_{1,0}^{(x^*)}$ and $\mathsf{AggProof}_1[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}^{(x^*,i^*)}, k]$ in $\mathsf{Hyb}_{1,1}^{(x^*)}$ compute identical functionalities. For a particular input $(i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$ consider the following cases:

**Case 1.** Suppose $i \neq i^*$. In this case, the two programs behave identically except that the latter is using $k_{\mathsf{sel}}^{(x^*,i^*)}$, so by punctured correctness, the two programs compute identical functionality.

**Case 2.** Suppose $x_i = x^*$. By assumption, $(C, x^*) \notin \mathcal{L}_{\mathsf{SAT}}$, so there does not exist a witness $w_i$ such that $C(x_i, w_i) = 0$, so both programs reject.

**Case 3.** Suppose $i = i^*$, $x_i \neq x^*$, and $\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}) \neq x^*$. Like the first case, the two programs behave identically except that the latter is using $k_{\mathsf{sel}}^{(x^*,i^*)}$, so by punctured correctness, the two programs compute identical functionality.

**Case 4.** Suppose $i = i^*$, $x_i \neq x^*$, and $\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}) = x^*$. In this case $\mathsf{AggProof}_1$ immediately outputs $\perp$. By statistical binding of $\Pi_{\mathsf{SEH}}$, since dig extracts to $x^*$ at position $i$, there does not exist an opening $\sigma_i$ such that $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i, \sigma_i) = 1$ whenever $x_i \neq x^*$. As such, AggProof also rejects.

The claim now follows from $i\mathcal{O}$ security. Formally, suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$,

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

Let $\Lambda_{\mathcal{B}} = \{(\lambda + n')^{1/\varepsilon_{\mathsf{obf}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $n'$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set. We define an efficient algorithm $\mathcal{B}$ which plays the $i\mathcal{O}$ security game with $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{obf}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{obf}} < 1$ and $n' > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{obf}}$).

**Algorithm $\mathcal{B}[x^*]$**

**Inputs:** $1^{\lambda_{\mathsf{obf}}}$ from $i\mathcal{O}$ challenger, $1^{\lambda}$ and $i^*$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Sample $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{H.SetupTD}(1^{\lambda}, 1^{n}, i^*)$.

3: Sample $\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, k$ and compute ObfVerProof as in Setup.

4: Compute $k_{\mathsf{sel}}^{(x^*,i^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{sel}}, (x^*, i^*))$.

5: Construct the challenge programs

$$\mathsf{AggProof}[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k] \text{ and } \mathsf{AggProof}_1[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k, \mathsf{td}, i^*, k_{\mathsf{sel}}^{(x^*,i^*)}, x^*]$$

and send them to the $i\mathcal{O}$ challenger. The $i\mathcal{O}$ challenger replies with an obfuscated program ObfAggProof.

6: Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

7: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

8: Output 1 if and only if $x^* = x_{i^*}$ and $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi) = 1$.

If the $i\mathcal{O}$ challenger obfuscates $\mathsf{AggProof}[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k]$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,0}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A}) = 1]$. Alternatively, if the $i\mathcal{O}$ challenger obfuscates the program $\mathsf{AggProof}_1[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k, \mathsf{td}, i^*, k_{\mathsf{sel}}^{(x^*,i^*)}, x^*]$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,1}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1]$. Thus by $i\mathcal{O}$ security we have that

$$|\Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A}) = 1]| = \mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) \leq 1/2^{\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}} = 1/2^{\lambda+n'} \leq 1/2^{\lambda+n}. \qquad \square$$

**Claim 6.8.** $\Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1] \geq \frac{1}{T+1} \Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1]$.

*Proof.* The challenger samples the index $j' \xleftarrow{\mathrm{R}} [T+1]$ *after* $\mathcal{A}$ outputs $(\vec{x}, \pi)$, where $\pi = (j, z)$. The output in $\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A})$ is 1 only if $j \in [T+1]$. Thus, with probability at least $\frac{1}{T+1}$, it will be the case that $j' = j$. In this case, the output in $\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A})$ is also 1 and the claim holds. $\square$

**Claim 6.9.** *Suppose* $\Pi_{\mathsf{PPRF}}$ *satisfies* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-puncturing security for constants* $\varepsilon_{\mathsf{PRF}} \in (0, 1)$ *and security parameter* $\lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$. *Then* $|\Pr[\mathsf{Hyb}_{1,3}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}$.

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

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

Let $\Lambda_{\mathcal{B}} = \{(\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $n'$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set. We define an efficient algorithm $\mathcal{B}$ which plays the puncturing security game with $\lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{PRF}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{PRF}} < 1$ and $n' > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{PRF}}$).

**Algorithm** $\mathcal{B}[x^*]$

**Inputs:** $1^{\lambda_{\mathsf{PRF}}}$ from PPRF challenger, $1^{\lambda}$ and $i^*$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Sample $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{H.SetupTD}(1^{\lambda}, 1^{n}, i^*)$, and $\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}$ as in Setup.

3: Compute ObfVerProof as in Setup.

4: Send input length $1^n$, output length $1^t$, and punctured point $(x^*, i^*)$ to the PPRF challenger. The PPRF challenger replies with the punctured key $k_{\mathsf{sel}}^{(x^*,i^*)}$ and a challenge value $j' \in \{0,1\}^t$.

5: Compute $\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{AggProof}_1[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k, \mathsf{td}, i^*, k_{\mathsf{sel}}^{(x^*,i^*)}, x^*])$.

6: Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

7: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

8: Output 1 if and only if $x^* = x_{i^*}$, $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$, and $j = j'$.

If the PPRF challenger samples $j' \xleftarrow{\mathrm{R}} \{0,1\}^t$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,2}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1]$. If the PPRF challenger computes $j' \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x^*, i^*))$ then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,3}^{(x^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{1,3}^{(x^*)}(\mathcal{A}) = 1]$. Thus by PPRF security we have that

$$|\Pr[\mathsf{Hyb}_{1,3}^{(x^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1]| = \mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) \le 1/2^{\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}} = 1/2^{\lambda+n'} \le 1/2^{\lambda+n}. \qquad \square$$

**Claim 6.10.** *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then,*

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

*Proof.* This follows by an analogous argument as Claim 6.7. $\square$

Combining Claims 6.7 to 6.10, we conclude that

$$\begin{aligned}
\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] &= \Pr[\mathsf{Hyb}_{1,0}^{(x^*)}(\mathcal{A}) = 1] && \text{by Eq. (6.5)} \\
&\le \Pr[\mathsf{Hyb}_{1,1}^{(x^*)}(\mathcal{A}) = 1] + \frac{1}{2^{\lambda+n}} && \text{by Claim 6.7} \\
&\le (T+1) \cdot \Pr[\mathsf{Hyb}_{1,2}^{(x^*)}(\mathcal{A}) = 1] + \frac{1}{2^{\lambda+n}} && \text{by Claim 6.8} \\
&\le (T+1) \cdot \left(\Pr[\mathsf{Hyb}_{1,3}^{(x^*)}(\mathcal{A}) = 1] + \frac{1}{2^{\lambda+n}}\right) + \frac{1}{2^{\lambda+n}} && \text{by Claim 6.9} \\
&\le (T+1) \cdot \left(\Pr[\mathsf{Hyb}_{1,4}^{(x^*)}(\mathcal{A}) = 1] + \frac{2}{2^{\lambda+n}}\right) + \frac{1}{2^{\lambda+n}} && \text{by Claim 6.10} \\
&= (T+1) \cdot \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \wedge \mathsf{E}_{x^*}] + \frac{2T+3}{2^{\lambda+n}} && \text{by Eq. (6.5).}
\end{aligned}$$

Thus Eq. (6.3) holds for all $x^*$ where $(C, x^*) \notin \mathcal{L}_{\mathsf{SAT}}$. Combined with Eq. (6.4), this means Eq. (6.3) holds for all $x^* \in \{0,1\}^n$. This proves Lemma 6.6. $\square$

**Lemma 6.11.** $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \ge \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

*Proof.* The conditions for outputting 1 in $\mathsf{Hyb}_3$ are a strict subset of those for outputting 1 in $\mathsf{Hyb}_2$. $\square$

**Lemma 6.12.** *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure for constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and security parameter $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ and $\Pi_{\mathsf{SEH}}$ satisfies statistical binding. Then*

$$\left|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| \le 1/2^{\lambda}.$$

*Proof.* We first show that $\mathsf{AggProof}[C, \mathsf{crs}_\mathsf{G}, \mathsf{hk}, k_\mathsf{sel}, k]$ in $\mathsf{Hyb}_2$ and $\mathsf{AggProof}'[C, \mathsf{crs}_\mathsf{G}, \mathsf{hk}, k_\mathsf{sel}, k, \mathsf{td}, i^*]$ in $\mathsf{Hyb}_4$ compute identical functionalities. For a particular input $(i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$, consider the following cases:

**Case 1.** If $i \neq i^*$, then the two programs behave identically.

**Case 2.** If $i = i^*$ and $\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}) = x_i$, then the condition $j = \mathsf{F}(k_\mathsf{sel}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$ in $\mathsf{AggProof}'$ is equivalent to checking $j = \mathsf{F}(k_\mathsf{sel}, (x_i, i))$ and outputting $\perp$ if this condition holds. This is the same condition in AggProof, so the output of both programs is $\perp$ in this case.

**Case 3.** If $i = i^*$ and $\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}) \neq x_i$, then by statistical binding of $\Pi_\mathsf{SEH}$, there does not exist an opening $\sigma_i$ where $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i^*, \sigma_i) = 1$. In this case, both programs reject.

The claim now follows from $i\mathcal{O}$ security. Formally, suppose there exists an infinite set $\Lambda_\mathcal{A} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$, we have that

$$|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A})]| > 1/2^\lambda.$$

Let $\Lambda_\mathcal{B} = \{(\lambda + n')^{1/\varepsilon_\mathsf{obf}} \mid \lambda \in \Lambda_\mathcal{A}\}$. $\Lambda_\mathcal{B}$ is also an infinite set. We define an efficient algorithm $\mathcal{B}$ which plays the $i\mathcal{O}$ security game with $\lambda_\mathsf{obf} = (\lambda + n')^{1/\varepsilon_\mathsf{obf}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_\mathsf{obf} \in \Lambda_\mathcal{B}$, we provide the associated value of $\lambda \in \Lambda_\mathcal{A}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_\mathcal{A}$ associated with a particular $\lambda_\mathsf{obf}$, we pick the largest such $\lambda$; note that since $\varepsilon_\mathsf{obf} < 1$ and $n' > 0$, it will always be the case that $\lambda < \lambda_\mathsf{obf}$).

**Algorithm $\mathcal{B}$**

**Inputs:** $1^{\lambda_\mathsf{obf}}$ from $i\mathcal{O}$ challenger, $1^\lambda$ and $i^*$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^\lambda)$.

2: Sample $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{H.SetupTD}(1^\lambda, 1^n, i^*)$.

3: Sample $\mathsf{crs}_\mathsf{G}, k_\mathsf{sel}, k$ and compute ObfVerProof as in Setup.

4: Construct the challenge programs

$$\mathsf{AggProof}[C, \mathsf{crs}_\mathsf{G}, \mathsf{hk}, k_\mathsf{sel}, k] \text{ and } \mathsf{AggProof}'[C, \mathsf{crs}_\mathsf{G}, \mathsf{hk}, k_\mathsf{sel}, k, \mathsf{td}, i^*]$$

and send them to the $i\mathcal{O}$ challenger. The $i\mathcal{O}$ challenger replies with the obfuscated program ObfAggProof.

5: Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_\mathsf{G}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

6: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

7: Output 1 if and only if $j = \mathsf{F}(k_\mathsf{sel}, (x_{i^*}, i^*))$ and $\mathsf{V}(\mathsf{crs}, (x_1, \ldots, x_T), \pi) = 1$.

If the $i\mathcal{O}$ challenger obfuscates $\mathsf{AggProof}[C, \mathsf{crs}_\mathsf{G}, \mathsf{hk}, k_\mathsf{sel}, k]$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_3$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$. If the $i\mathcal{O}$ challenger obfuscates $\mathsf{AggProof}'[C, \mathsf{crs}_\mathsf{G}, \mathsf{hk}, k_\mathsf{sel}, k, \mathsf{td}, i^*]$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_4$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]$. Thus by $i\mathcal{O}$ security we have that

$$|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathsf{iOAdv}_\mathcal{B}(\lambda_\mathsf{obf}) \leq 1/2^{\lambda_\mathsf{obf}^{\varepsilon_\mathsf{obf}}} = 1/2^{\lambda + n'} \leq 1/2^\lambda. \qquad \square$$

**Lemma 6.13.** *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure, $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness and $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-puncturing security, $\Pi_{\mathsf{RPRG}}$ is expanding, satisfies $(1, \mathsf{negl}(\lambda))$-pseudorandomness, and $(1, 2^{-m^{\varepsilon_m}})$-re-randomization security for constants $(\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}}, \varepsilon_m) \in (0,1)$ and security parameters $\lambda_{\mathsf{obf}} = (\lambda+n')^{1/\varepsilon_{\mathsf{obf}}}, \lambda_{\mathsf{PRF}} = (\lambda+n')^{1/\varepsilon_{\mathsf{PRF}}}, m = (\lambda+n')^{1/\varepsilon_m}$. Then for all $d \in [T]$,*

$$\Pr[\mathsf{Hyb}_{5,d}(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_{5,d-1}(\mathcal{A}) = 1] - \mathsf{negl}(\lambda).$$

*where for notational convenience we define $\mathsf{Hyb}_{5,0} := \mathsf{Hyb}_4$.*

*Proof.* We proceed by defining a sequence of intermediate hybrids for each value of $\mathsf{dig}^* \in \{0,1\}^{n'}$.

$\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*-1)}$ (or $\mathsf{Hyb}_{5,d-1}$ if $\mathsf{dig}^* = 0^{n'}$) except the challenger computes

- $j^* \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}^*), i^*))$
- $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda}; \mathsf{F}(k, j^*))$
- $k^{(j^*)} \leftarrow \mathsf{F.Puncture}(k, j^*)$
- $y^* \leftarrow \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_{j^*}, (\mathsf{dig}^*, d))))$
- $y_{\mathsf{base}} \leftarrow \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}))$
- $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+\lambda}, 1^{\mu})$

where $\mu$ is a bound on the number of bits of randomness the PRG.Rerandomize algorithm takes. Then, the challenger defines the following modified version of VerProof′ as follows:

$\mathsf{VerProof}_2[\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, \mathsf{td}, i^*, d, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}](i, j, \mathsf{dig}, z_i)$

**Inputs:** index $i$, selection symbol $j$, hash value dig, proof $z_i$

1: If $i^* \leq i < d$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$, output 0.

2: If $j = j^*$ : let $k_j = k_{j^*}$.

3: Else: compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k^{(j^*)}, j))$.

4: If $i^* \leq i = d$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$:

- If $\mathsf{dig} < \mathsf{dig}^*$ : compute $y_i \leftarrow \mathsf{PRG.Rerandomize}(\mathsf{crs}_{\mathsf{G}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (\mathsf{dig}, i)))$
- If $\mathsf{dig} = \mathsf{dig}^*$ : let $y_i = y^*$.
- If $\mathsf{dig} > \mathsf{dig}^*$ : compute $y_i \leftarrow \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i))))$.

5: Else: Compute $y_i = \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i))))$.

6: Output 1 if $\mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, z_i) = y_i$ and 0 otherwise.

The challenger also uses $\mathsf{VerProof}_2[\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, \mathsf{td}, i^*, d, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}]$ in place of $\mathsf{VerProof}'[\mathsf{crs}_{\mathsf{G}}, k, k_{\mathsf{sel}}, \mathsf{td}, i^*, d]$ in the proof aggregation program. Specifically, in this experiment, the challenger defines a modified version of AggProof′ as follows:

$\mathsf{AggProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, \mathsf{td}, i^*, d, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}](i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$

**Inputs:** index $i$, selection symbol $j$, hash value dig, statement $x_i$, witness $w_i$, opening $\sigma_i$, prior proof $z_{i-1}$

1: If $C(x_i, w_i) = 0$, output $\bot$.
2: If $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i, \sigma_i) = 0$, output $\bot$.
3: If $i = i^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$, output $\bot$.
4: If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\bot$.
5: If $\mathsf{VerProof}_2[\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, \mathsf{td}, i^*, d, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}](i-1, j, \mathsf{dig}, z_{i-1}) = 0$ and $i \neq 1$, output $\bot$.
6: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.
7: Output $z_i = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i)))$.

When constructing the CRS, the challenger now computes

$$\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{AggProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, \mathsf{td}, i^*, d, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}])$$

and

$$\mathsf{ObfVerProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_2[\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, \mathsf{td}, i^*, d, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}]).$$

$\mathsf{Hyb}_{5,d,2}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}$ except the challenger samples $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda})$ instead of computing $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda}; \mathsf{F}(k, j^*))$.

$\mathsf{Hyb}_{5,d,3}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,2}^{(\mathsf{dig}^*)}$ except the challenger additionally computes

- $k_{j^*}^{(\mathsf{dig}^*, d)} \leftarrow \mathsf{F.Puncture}(k_{j^*}, (\mathsf{dig}^*, d))$
- $k_{\mathsf{rerand}}^{(\mathsf{dig}^*, d)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{rerand}}, (\mathsf{dig}^*, d))$

and uses the punctured keys in place of $k_{j^*}, k_{\mathsf{rerand}}$.

$\mathsf{Hyb}_{5,d,4}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,3}^{(\mathsf{dig}^*)}$ except the challenger samples $y^* \leftarrow \mathsf{PRG.Eval}(\mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}))$ instead of computing $y^* = \mathsf{PRG.Eval}(\mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_{j^*}, (\mathsf{dig}^*, d))))$.

$\mathsf{Hyb}_{5,d,5}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,4}^{(\mathsf{dig}^*)}$ except the challenger samples $y^* \leftarrow \mathsf{PRG.Rerandomize}(\mathsf{crs}_{\mathsf{G}}, y_{\mathsf{base}})$.

$\mathsf{Hyb}_{5,d,6}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,5}^{(\mathsf{dig}^*)}$ except the challenger computes

$$y^* \leftarrow \mathsf{PRG.Rerandomize}(\mathsf{crs}_{\mathsf{G}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (\mathsf{dig}^*, d))).$$

$\mathsf{Hyb}_{5,d,7}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,6}^{(\mathsf{dig}^*)}$ except the challenger reverts to using unpunctured keys $k_{j^*}, k_{\mathsf{rerand}}$ in place of the punctured keys $k_{j^*}^{(\mathsf{dig}^*, d)}, k_{\mathsf{rerand}}^{(\mathsf{dig}^*, d)}$.

$\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,7}^{(\mathsf{dig}^*)}$ except the challenger reverts to computing

$$k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda}; \mathsf{F}(k, j^*))$$

instead of sampling $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda})$.

We also define two more intermediate hybrids:

$\mathsf{Hyb}_{5,d,9}$ : Same as $\mathsf{Hyb}_{5,d,8}^{(1^{n'})}$ except the challenger reverts to using the unpunctured key $k$ in place of the punctured key $k^{(j^*)}$ and defines a modified version of $\mathsf{VerProof}_2$ as follows:

$\mathsf{VerProof}_3[\mathsf{crs}_G, \mathsf{td}, i^*, d, k, k_{\mathsf{sel}}, k_{\mathsf{rerand}}, y_{\mathsf{base}}](i, j, \mathsf{dig}, z_i)$

**Inputs:** index $i$, selection symbol $j$, hash value dig, proof $z_i$

1: Compute $x^*_{i^*} \leftarrow \mathsf{H.Extract}(\mathsf{td}, \mathsf{dig})$.
2: If $i^* \le i < d$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x^*_{i^*}, i^*))$, output 0.
3: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.
4: If $i^* \le i = d$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x^*_{i^*}, i^*))$, compute $y_i \leftarrow \mathsf{PRG.Rerandomize}(\mathsf{crs}_G, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (\mathsf{dig}, i)))$.
5: Else: Compute $y_i = \mathsf{PRG.Eval}(\mathsf{crs}_G, \mathsf{PRG.GenSeed}(\mathsf{crs}_G; \mathsf{F}(k_j, (\mathsf{dig}, i))))$.
6: Output 1 if $\mathsf{PRG.Eval}(\mathsf{crs}_G, z_i) = y_i$ and 0 otherwise.

The challenger also uses $\mathsf{VerProof}_3[\mathsf{crs}_G, \mathsf{td}, i^*, d, k, k_{\mathsf{sel}}, k_{\mathsf{rerand}}, y_{\mathsf{base}}]$ in place of the verification program $\mathsf{VerProof}_2[\mathsf{crs}_G, k_{\mathsf{sel}}, \mathsf{td}, i^*, d, k^{(j^*)}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}]$ in the proof aggregation program. Specifically, in this experiment, the challenger defines a modified version of $\mathsf{AggProof}_2$ as follows:

$\mathsf{AggProof}_3[C, \mathsf{crs}_G, \mathsf{td}, i^*, d, k, k_{\mathsf{sel}}, k_{\mathsf{rerand}}, y_{\mathsf{base}}](i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i-1})$

**Inputs:** index $i$, selection symbol $j$, hash value dig, statement $x_i$, witness $w_i$, opening $\sigma_i$, prior proof $z_{i-1}$

1: If $C(x_i, w_i) = 0$, output $\perp$.
2: If $\mathsf{H.Verify}(\mathsf{hk}, \mathsf{dig}, x_i, i, \sigma_i) = 0$, output $\perp$.
3: If $i = i^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$, output $\perp$.
4: If $j = \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$, output $\perp$.
5: If $i \neq 1$ and $\mathsf{VerProof}_3[\mathsf{crs}_G, \mathsf{td}, i^*, d, k, k_{\mathsf{sel}}, k_{\mathsf{rerand}}, y_{\mathsf{base}}](i-1, j, \mathsf{dig}, z_{i-1}) = 0$, output $\perp$.
6: Compute $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\rho}; \mathsf{F}(k, j))$.
7: Output $z_i = \mathsf{PRG.GenSeed}(\mathsf{crs}_G; \mathsf{F}(k_j, (\mathsf{dig}, i)))$.

When constructing the CRS, the challenger now computes

$$\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{AggProof}_3[C, \mathsf{crs}_G, \mathsf{td}, i^*, d, k, k_{\mathsf{sel}}, k_{\mathsf{rerand}}, y_{\mathsf{base}}])$$

and

$$\mathsf{ObfVerProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{VerProof}_3[\mathsf{crs}_G, \mathsf{td}, i^*, d, k, k_{\mathsf{sel}}, k_{\mathsf{rerand}}, y_{\mathsf{base}}]).$$

$\mathsf{Hyb}_{5,d,10}$ : Same as $\mathsf{Hyb}_{5,d,9}$ except the challenger samples $y_{\mathsf{base}} \xleftarrow{R} \mathcal{Y}$ (where $\mathcal{Y}$ is the output space implicitly defined by $\mathsf{crs}_G$).

We now consider each pair of adjacent distributions.

**Claim 6.14.** *Fix any* $\mathsf{dig}^* \in \{0,1\}^{n'} \setminus \{0^{n'}\}$. *Suppose* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$*-secure for constant* $\varepsilon_{\mathsf{obf}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ *and* $\Pi_{\mathsf{PPRF}}$ *satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*-1)}(\mathcal{A}) = 1]| \leq 2/2^{\lambda+n'}.$$

*Proof.* To complete the proof, we first introduce an intermediate hybrid:

$\mathsf{iHyb}_{5,d,1}^{(\mathsf{dig}^*)}$ : Same as $\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}$ except the challenger defines AggProof as in $\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*-1)}$.

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

$$|\Pr[\mathsf{iHyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A})]| > 1/2^{\lambda+n'}.$$

Let $\Lambda_{\mathcal{B}} = \{(\lambda + n')^{1/\varepsilon_{\mathsf{obf}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $n'$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set. We define an efficient algorithm $\mathcal{B}$ which plays the $i\mathcal{O}$ security game with $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{obf}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{obf}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{obf}} < 1$ and $n' > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{obf}}$).

**Algorithm** $\mathcal{B}[d, \mathsf{dig}^*]$

**Inputs:** $1^{\lambda_{\mathsf{obf}}}$ from $i\mathcal{O}$ challenger, $1^{\lambda}$ and $i^*$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Sample $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{H.SetupTD}(1^{\lambda}, 1^{n}, i^*)$, and $\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}$ as in Setup.

3: Sample $y_{\mathsf{base}} \leftarrow \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}))$, and $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+\lambda}, 1^{\mu})$.

4: Compute $j' \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}^* - 1), i^*))$.

5: Compute $k^{(j')} \leftarrow \mathsf{F.Puncture}(k, j')$, and $k_{j'} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda}; \mathsf{F}(k, j'))$.

6: Compute $y' \leftarrow \mathsf{PRG.Rerandomize}(\mathsf{crs}_{\mathsf{G}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (\mathsf{dig}^* - 1, d)))$.

7: Compute $j^* \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}^*), i^*))$.

8: Compute $k^{(j^*)} \leftarrow \mathsf{F.Puncture}(k, j^*)$, and $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda}; \mathsf{F}(k, j^*))$.

9: Compute $y^* \leftarrow \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_{j^*}, (\mathsf{dig}^*, d))))$.

10: Construct the obfuscated program $\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{s}, P)$ where

$$P := \mathsf{AggProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, \mathsf{td}, i^*, d, k^{(j')}, k_{\mathsf{sel}}, k_{j'}, k_{\mathsf{rerand}}, \mathsf{dig}^* - 1, j', y', y_{\mathsf{base}}].$$

11: Construct the following two challenge programs:

- $V := \mathsf{VerProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{td}, i^*, d, k^{(j')}, k_{\mathsf{sel}}, k_{j'}, k_{\mathsf{rerand}}, \mathsf{dig}^* - 1, j', y', y_{\mathsf{base}}]$
- $V' := \mathsf{VerProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{td}, i^*, d, k^{(j^*)}, k_{\mathsf{sel}}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}]$

and send $(V, V')$ to the $i\mathcal{O}$ challenger. The $i\mathcal{O}$ challenger replies with the obfuscated program ObfVerProof.

12: Let crs = $(\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

13: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

14: Output 1 if and only if $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$.

We first show that the program $V$ computed as in $\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*-1)}$ and the program $V'$ computed as in $\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}$ compute identical functionalities. For a particular input $(i, j, \mathsf{dig}, z_i)$ consider the following cases:

**Case 1.** If $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$ and $i^* \leq i < d$, both programs output 0.

**Case 2.** If $\mathsf{dig} < \mathsf{dig}^* - 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$ and $i^* \leq i = d$, then both programs compute $y_i$ as

$$y_i = \mathsf{PRG.Rerandomize}(\mathsf{crs}_{\mathsf{G}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (\mathsf{dig}, i))).$$

The remaining logic in the two programs is identical.

**Case 3.** If $\mathsf{dig} = \mathsf{dig}^* - 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*)) = j'$ and $i^* \leq i = d$, the two programs behave identically except $V$ uses the hard-coded value

$$y' = \mathsf{PRG.Rerandomize}(\mathsf{crs}_{\mathsf{G}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (\mathsf{dig}^* - 1, d))).$$

**Case 4.** If $\mathsf{dig} = \mathsf{dig}^*$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*)) = j^*$ and $i^* \leq i = d$, the two programs behave identically except $V'$ uses the hard-coded value

$$y^* = \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_{j^*}, (\mathsf{dig}^*, d)))).$$

**Case 5.** If $\mathsf{dig} > \mathsf{dig}^*$ or $j \neq \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Extract}(\mathsf{td}, \mathsf{dig}), i^*))$ or $i^* > i$ or $i > d$, the two programs behave identically except $V$ may be using the hard-coded key $k_{j'} = \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda}; \mathsf{F}(k, j'))$ and $V'$ may be using the hard-coded key $k_{j^*} = \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda}; \mathsf{F}(k, j^*))$. Both compute $y_i = \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, i))))$ in an identical manner.

We conclude that the two programs output identical functionality. If the $i\mathcal{O}$ challenger obfuscates $V$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*-1)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*-1)}(\mathcal{A}) = 1]$. If the $i\mathcal{O}$ challenger obfuscates $V'$, then $\mathcal{B}$ perfectly simulates $\mathsf{iHyb}_{5,d,1}^{(\mathsf{dig}^*)}$ and outputs 1 with probability $\Pr[\mathsf{iHyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]$. Thus by $i\mathcal{O}$ security we have that

$$|\Pr[\mathsf{iHyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*-1)}(\mathcal{A}) = 1]| = \mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) \leq 1/2^{\lambda^{\varepsilon_{\mathsf{obf}}}_{\mathsf{obf}}} = 1/2^{\lambda+n'}.$$

By an analogous argument (where the reduction algorithm obtains ObfAggProof from the $i\mathcal{O}$ challenger), we can show that for all sufficiently large $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A})]| \leq 1/2^{\lambda+n'}.$$

Thus by combining the above two relations, we conclude that

$$|\Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*-1)}(\mathcal{A})]| \leq 2/2^{\lambda+n'}.$$ □

**Claim 6.15.** *Let* $\mathsf{dig}^* = 0^{n'}$. *Suppose* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$*-secure for constant* $\varepsilon_{\mathsf{obf}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ *and* $\Pi_{\mathsf{PPRF}}$ *satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d-1}(\mathcal{A}) = 1]| \leq 2/2^{\lambda+n'}.$$

*Proof.* This follows by an analogous argument as Claim 6.14. □

**Claim 6.16.** *Fix any* $\mathsf{dig}^* \in \{0,1\}^{n'}$. *Suppose* $\Pi_{\mathsf{PPRF}}$ *satisfies* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-puncturing security for constants* $\varepsilon_{\mathsf{PRF}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$. *Then*

$$|\Pr[\mathsf{Hyb}_{5,d,2}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n'}.$$

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

$$|\Pr[\mathsf{Hyb}_{5,d,2}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A})]| > 1/2^{\lambda+n'}.$$

Let $\Lambda_{\mathcal{B}} = \{(\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}} \mid \lambda \in \Lambda_{\mathcal{A}}\}$. Since $n'$ is non-negative, $\Lambda_{\mathcal{B}}$ is also an infinite set. We define an efficient algorithm $\mathcal{B}$ which plays the puncturing security game with $\lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$ by running $\mathcal{A}$ with security parameter $\lambda$. For each value of $\lambda_{\mathsf{PRF}} \in \Lambda_{\mathcal{B}}$, we provide the associated value of $\lambda \in \Lambda_{\mathcal{A}}$ to $\mathcal{B}$ as non-uniform advice (if there are multiple such $\lambda \in \Lambda_{\mathcal{A}}$ associated with a particular $\lambda_{\mathsf{PRF}}$, we pick the largest such $\lambda$; note that since $\varepsilon_{\mathsf{PRF}} < 1$ and $n' > 0$, it will always be the case that $\lambda < \lambda_{\mathsf{PRF}}$).

**Algorithm** $\mathcal{B}[d, \mathsf{dig}^*]$

**Inputs:** $1^{\lambda_{\mathsf{PRF}}}$ from PPRF challenger, $1^{\lambda}$ and $i^*$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Sample $\mathsf{hk} \leftarrow \mathsf{H.SetupTD}(1^{\lambda}, 1^{n}, i^*)$, and $\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}$ as in Setup.

3: Compute $j^* \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{dig}^*, i^*))$

4: Send input length $1^t$, output length $1^{\tau}$, and punctured point $j^*$ to the PPRF challenger. The PPRF challenger replies with the punctured key $k^{(j^*)}$ and the challenge value $r \in \{0,1\}^t$ from PPRF challenger.

5: Compute $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda}; r)$.

6: Compute $y^* = \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_{j^*}, (\mathsf{dig}^*, d))))$.

7: Sample $y_{\mathsf{base}} \leftarrow \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}))$.

8: Compute $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+\lambda}, 1^{\mu})$.

9: Compute $\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{s}, P)$ where

$$P := \mathsf{AggProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, \mathsf{td}, i^*, k_{\mathsf{sel}}, k^{(j^*)}, d, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}].$$

10: Compute $\mathsf{ObfVerProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{s}, V)$ where

$$V := \mathsf{VerProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{td}, i^*, k^{(j^*)}, d, k_{\mathsf{sel}}, k_{j^*}, k_{\mathsf{rerand}}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}].$$

11: Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

12: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

13: Output 1 if and only if $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$.

We consider the two possibilities for the behavior for the PPRF challenger:

- If the PPRF challenger samples $r \xleftarrow{R} \{0,1\}^{\rho}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{5,d,2}^{(\mathsf{dig}^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{5,d,2}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]$.
- If the PPRF challenger computes $r \leftarrow \mathsf{F}(k, (j^*))$ then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]$.

Thus by PPRF security, we have that

$$|\Pr[\mathsf{Hyb}_{5,d,1}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,2}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| = \mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{PRF}}) \leq 1/2^{\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}} = 1/2^{\lambda+n'}. \qquad \square$$

**Claim 6.17.** *Fix any* $\mathsf{dig}^* \in \{0,1\}^{n'}$. *Suppose* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$*-secure for constant* $\varepsilon_{\mathsf{obf}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ *and* $\Pi_{\mathsf{PPRF}}$ *satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_{5,d,3}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,2}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| \leq 2/2^{\lambda+n'}.$$

*Proof.* This follows by an analogous argument as Claim 6.14. $\square$

**Claim 6.18.** *Fix any* $\mathsf{dig}^* \in \{0,1\}^{n'}$. *Suppose* $\Pi_{\mathsf{PPRF}}$ *satisfies* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-puncturing security for constants* $\varepsilon_{\mathsf{PRF}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$. *Then*

$$|\Pr[\mathsf{Hyb}_{5,d,4}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,3}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n'}.$$

*Proof.* This follows by an analogous argument as Claim 6.16. $\square$

**Claim 6.19.** *Fix any* $\mathsf{dig}^* \in \{0,1\}^{n'}$. *Suppose* $\Pi_{\mathsf{RPRG}}$ *satisfies* $(1, 2^{-m^{\varepsilon_m}})$*-re-randomization security for constant* $\varepsilon_m \in (0,1)$ *and re-randomization parameter* $m = (\lambda + n')^{1/\varepsilon_m}$. *Then*

$$|\Pr[\mathsf{Hyb}_{5,d,5}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,4}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n'}.$$

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

$$|\Pr[\mathsf{Hyb}_{5,d,5}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,4}^{(\mathsf{dig}^*)}(\mathcal{A})]| > 1/2^{\lambda+n'}.$$

Let $m(\lambda) = (\lambda + n)^{1/\varepsilon_m}$. We define an efficient algorithm $\mathcal{B}$ which plays the re-randomization security game with $m = (\lambda + n')^{1/\varepsilon_m}$ by running $\mathcal{A}$ with security parameter $\lambda$.

**Algorithm** $\mathcal{B}[d, \mathsf{dig}^*]$

**Inputs:** $\mathsf{crs}_{\mathsf{G}} \leftarrow \mathsf{G.Setup}(1^{\lambda}, 1^m)$, $y_{\mathsf{base}}$, $y^*$ from re-randomization challenger, $i^*$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^{\lambda})$.

2: Sample $\mathsf{hk} \leftarrow \mathsf{H.SetupTD}(1^{\lambda}, 1^n, i^*)$, and $k_{\mathsf{sel}}$ as in Setup.

3: Compute $j^* \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{dig}^*, i^*))$ and $k_{j^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+t}, 1^{\lambda})$.

4: Compute $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+\lambda}, 1^{\mu})$.

5: Compute $k^{(j^*)} \leftarrow \mathsf{F.Puncture}(k, j^*)$, $k_{j^*}^{(\mathsf{dig}^*,d)} \leftarrow \mathsf{F.Puncture}(k_{j^*}, (\mathsf{dig}^*, d))$, and $k_{\mathsf{rerand}}^{(\mathsf{dig}^*,d)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{rerand}}, (\mathsf{dig}^*, d))$.

6: Compute $\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, P)$ where

$$P := \mathsf{AggProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k^{(j^*)}, i^*, d, k_{j^*}^{(\mathsf{dig}^*,d)}, k_{\mathsf{rerand}}^{(\mathsf{dig}^*,d)}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}]).$$

7: Compute $\mathsf{ObfVerProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, V$ where

$$V := \mathsf{VerProof}_2[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k^{(j^*)}, i^*, d, k_{\mathsf{sel}}, k_{j^*}^{(\mathsf{dig}^*,d)}, k_{\mathsf{rerand}}^{(\mathsf{dig}^*,d)}, \mathsf{dig}^*, j^*, y^*, y_{\mathsf{base}}]).$$

8: Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

9: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

10: Output 1 if and only if $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Hash}(\mathsf{hk}, \vec{x}), i^*))$.

If the re-randomization challenger samples $y^* \leftarrow \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{5,d,4}^{(\mathsf{dig}^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{5,d,4}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]$. If the re-randomization challenger samples $y^* \leftarrow \mathsf{PRG.Rerandomize}(\mathsf{crs}_{\mathsf{G}}, y_{\mathsf{base}})$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{5,d,5}^{(\mathsf{dig}^*)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{5,d,5}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]$. Thus by re-randomization security we have that

$$|\Pr[\mathsf{Hyb}_{5,d,5}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,4}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| = \mathsf{RerandAdv}_{\mathcal{B}}(m) \leq 1/2^{m\varepsilon_m} = 1/2^{\lambda+n'}. \qquad \square$$

**Claim 6.20.** *Fix any* $\mathsf{dig}^* \in \{0,1\}^{n'}$. *Suppose* $\Pi_{\mathsf{PPRF}}$ *satisfies* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-puncturing security for constants* $\varepsilon_{\mathsf{PRF}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$. *Then*

$$|\Pr[\mathsf{Hyb}_{5,d,6}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,5}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n'}.$$

*Proof.* This follows by an analogous argument as Claim 6.16. $\square$

**Claim 6.21.** *Fix any* $\mathsf{dig}^* \in \{0,1\}^{n'}$. *Suppose* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$*-secure for constant* $\varepsilon_{\mathsf{obf}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{obf}} = (\lambda + n')^{1/\varepsilon_{\mathsf{obf}}}$ *and* $\Pi_{\mathsf{PPRF}}$ *satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_{5,d,7}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,6}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| \leq 2/2^{\lambda+n'}.$$

*Proof.* This follows by an analogous argument as Claim 6.14. $\square$

**Claim 6.22.** *Fix any* $\mathsf{dig}^* \in \{0,1\}^{n'}$. *Suppose* $\Pi_{\mathsf{PPRF}}$ *satisfies* $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$*-puncturing security for constants* $\varepsilon_{\mathsf{PRF}} \in (0,1)$ *and security parameter* $\lambda_{\mathsf{PRF}} = (\lambda + n')^{1/\varepsilon_{\mathsf{PRF}}}$. *Then*

$$|\Pr[\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,7}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n'}.$$

*Proof.* This follows by an analogous argument as Claim 6.16. $\square$

**Claim 6.23.** *Fix* $\mathsf{dig}^* = 1^{n'}$. *Suppose* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathrm{obf}}^{\varepsilon_{\mathrm{obf}}}})$*-secure for constant* $\varepsilon_{\mathrm{obf}} \in (0,1)$ *and security parameter* $\lambda_{\mathrm{obf}} = (\lambda + n')^{1/\varepsilon_{\mathrm{obf}}}$ *and* $\Pi_{\mathsf{PPRF}}$ *satisfies punctured correctness. Then*

$$|\Pr[\mathsf{Hyb}_{5,d,9}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,8}^{(\mathsf{dig}^*)}(\mathcal{A}) = 1]| \le 2/2^{\lambda+n'}.$$

*Proof.* This follows by an analogous argument as Claim 6.14. □

**Claim 6.24.** *Suppose that* $\Pi_{\mathsf{RPRG}}$ *is* $(1, \mathsf{negl}(\lambda))$*-pseudorandom. Then*

$$|\Pr[\mathsf{Hyb}_{5,d,10}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,9}(\mathcal{A}) = 1]| \le \mathsf{negl}(\lambda).$$

*Proof.* We define an efficient algorithm $\mathcal{B}$ which plays the pseudorandomness security game with security parameter $\lambda$ by running $\mathcal{A}$ with security parameter $\lambda$.

**Algorithm** $\mathcal{B}[d]$

**Inputs:** $\mathsf{crs}_{\mathsf{G}} \leftarrow \mathsf{G.Setup}(1^\lambda, 1^{m(\lambda)})$, $y_{\mathsf{base}}$ from challenger, $i^*$ as non-uniform advice

1: Run $(C, T) \leftarrow \mathcal{A}(1^\lambda)$.

2: Sample $\mathsf{hk} \leftarrow \mathsf{H.SetupTD}(1^\lambda, 1^n, i^*)$, and $k_{\mathsf{sel}}$ as in Setup.

3: Compute $k_{\mathsf{rerand}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n'+\lambda}, 1^\mu)$.

4: Compute $\mathsf{ObfAggProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathrm{obf}}}, 1^s, \mathsf{AggProof}_3[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, i^*, d, k_{\mathsf{rerand}}, y_{\mathsf{base}}])$.

5: Compute $\mathsf{ObfVerProof} \leftarrow i\mathcal{O}(1^{\lambda_{\mathrm{obf}}}, 1^s, \mathsf{VerProof}_3[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, i^*, d, k_{\mathsf{rerand}}, y_{\mathsf{base}}])$.

6: Let $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, \mathsf{ObfAggProof}, \mathsf{ObfVerProof})$.

7: Run $(\vec{x}, \pi) \leftarrow \mathcal{A}(\mathsf{crs})$ and parse $\vec{x} = (x_1, \ldots, x_T)$ and $\pi = (j, z)$.

8: Output 1 if and only if $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{H.Hash}(\mathsf{hk}, \vec{x}), i^*))$.

If the challenger samples $y_{\mathsf{base}} \leftarrow \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}))$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{5,d,9}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{5,d,9}(\mathcal{A}) = 1]$. If the challenger samples $y_{\mathsf{base}}$ uniformly from $\mathcal{Y}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{5,d,10}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_{5,d,10}(\mathcal{A}) = 1]$. Thus we have that

$$|\Pr[\mathsf{Hyb}_{5,d,10}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,9}(\mathcal{A}) = 1]| \le \mathsf{negl}(\lambda).$$ □

**Claim 6.25.** *Suppose that* $\Pi_{\mathsf{RPRG}}$ *satisfies correctness and re-randomization correctness and is expanding. Then*

$$|\Pr[\mathsf{Hyb}_{5,d}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,10}(\mathcal{A}) = 1]| \le 1/2^{\Omega(\lambda)}.$$

*Proof.* We show that with overwhelming probability over the choice of $y_{\mathsf{base}}$, the programs

$$\mathsf{AggProof}_3[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k_{\mathsf{sel}}, k, i^*, d, k_{\mathsf{rerand}}, y_{\mathsf{base}}]) \quad \text{and} \quad \mathsf{VerProof}_3[\mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, k, i^*, d, k_{\mathsf{rerand}}, y_{\mathsf{base}}])$$

which the challenger obfuscates in $\mathsf{Hyb}_{5,d,10}$ and

$$\mathsf{AggProof}'[C, \mathsf{crs}_{\mathsf{G}}, \mathsf{hk}, k, i^*, d] \quad \text{and} \quad \mathsf{VerProof}'[\mathsf{crs}_{\mathsf{G}}, k, i^*, d]$$

which the challenger obfuscates in $\mathsf{Hyb}_{5,d}$ compute identical functionalities, respectively.

**The verification programs.** We first consider $\mathsf{VerProof}_3$ and $\mathsf{VerProof}'$. For a particular input $(i, j, \mathsf{dig}, z_i)$, consider the following cases:

**Case 1.** If $i \neq d$ or $j \neq \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{dig}, i))$, then $\mathsf{VerProof}_3$ and $\mathsf{VerProof}'$ behave identically.

**Case 2.** If $i = d$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (\mathsf{dig}, i))$, then $\mathsf{VerProof}'$ always outputs 0. Consider the behavior in $\mathsf{VerProof}_3$. By construction, $\mathsf{VerProof}_3$ first computes

$$y_i = \mathsf{PRG.Rerandomize}(\mathsf{crs}_{\mathsf{G}}, y_{\mathsf{base}}; \mathsf{F}(k_{\mathsf{rerand}}, (\mathsf{dig}, i)))$$

and outputs 1 if and only if $\mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, z_i) = y_i$. Note that by re-randomization correctness, this never occurs if $y_{\mathsf{base}}$ is not in the image of $\mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, \cdot)$.

Since $|\mathcal{Y}| \geq 2^{\Omega(\lambda)} \cdot |\mathcal{Z}|$ and $y_{\mathsf{base}} \xleftarrow{\mathrm{R}} \mathcal{Y}$, we have that

$$\Pr[\exists z \in \mathcal{Z} : \mathsf{PRG.Eval}(\mathsf{crs}_{\mathsf{G}}, z) = y_{\mathsf{base}}] \leq 1/2^{\Omega(\lambda)}.$$

so with probability $1 - 1/2^{\Omega(\lambda)}$, $\mathsf{VerProof}_3$ and $\mathsf{VerProof}'$ also behave identically for all inputs which fall into case 2. In other words, with overwhelming probability over the choice of $y_{\mathsf{base}}$, $\mathsf{VerProof}_3$ and $\mathsf{VerProof}'$ compute identical functionality.

**The proof-aggregation programs.** Next, we consider the proof-aggregation programs $\mathsf{AggProof}_3$ and $\mathsf{AggProof}'$. The only difference between these is that $\mathsf{AggProof}_3$ calls $\mathsf{VerProof}_3$ while $\mathsf{AggProof}'$ calls $\mathsf{VerProof}'$. By our above argument, the verification programs $\mathsf{VerProof}_3$ and $\mathsf{VerProof}'$ compute identical functionality, so the same extends to $\mathsf{AggProof}_3$ and $\mathsf{AggProof}'$. The claim now follows by $i\mathcal{O}$ security, using an analogous argument as in 6.14. □

**Proof of Lemma 6.13.** We now return to the proof of Lemma 6.13. By Claims 6.14 to 6.25 and the triangle inequality, we can now write

$$\begin{aligned}
&\left|\Pr[\mathsf{Hyb}_{5,d}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d-1}(\mathcal{A}) = 1]\right| \\
&\quad\leq \left|\Pr[\mathsf{Hyb}_{5,d}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,10}(\mathcal{A}) = 1]\right| \\
&\qquad + \left|\Pr[\mathsf{Hyb}_{5,d,10}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d,9}(\mathcal{A}) = 1]\right| \\
&\qquad + \left|\Pr[\mathsf{Hyb}_{5,d,9}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(1^{n'})}_{5,d,8}(\mathcal{A}) = 1]\right| \\
&\qquad + \sum_{\mathsf{dig}^* \in \{0,1\}^{n'}} \sum_{\ell=2}^{8} \left|\Pr[\mathsf{Hyb}^{(\mathsf{dig}^*)}_{5,d,\ell}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(\mathsf{dig}^*)}_{5,d,\ell-1}(\mathcal{A}) = 1]\right| \\
&\qquad + \sum_{\mathsf{dig}^* \in \{0,1\}^{n'} \setminus \{0^{n'}\}} \left|\Pr[\mathsf{Hyb}^{(\mathsf{dig}^*)}_{5,d,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(\mathsf{dig}^*-1)}_{5,d,8}(\mathcal{A}) = 1]\right| \\
&\qquad + \left|\Pr[\mathsf{Hyb}^{(0^{n'})}_{5,d,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{5,d-1}(\mathcal{A}) = 1]\right| \\
&\quad\leq \underbrace{\frac{1}{2^{\Omega(\lambda)}}}_{\text{Claim 6.25}} + \underbrace{\mathsf{negl}(\lambda)}_{\text{Claim 6.24}} + \underbrace{\frac{2}{2^{\lambda+n'}}}_{\text{Claim 6.23}} + \underbrace{2^{n'} \cdot \frac{O(1)}{2^{\lambda+n'}}}_{\text{Claims 6.16 to 6.22}} + \underbrace{2^{n'} \cdot \frac{2}{2^{\lambda+n'}}}_{\text{Claim 6.14}} + \underbrace{\frac{2}{2^{\lambda+n'}}}_{\text{Claim 6.15}},
\end{aligned}$$

which is bounded by a negligible function. Lemma 6.13 holds. □

**Lemma 6.26.** $\Pr[\mathsf{Hyb}_{5,T}(\mathcal{A}) = 1] = 0$.

*Proof.* In $\mathsf{Hyb}_{5,T}$, the program VerProof, and thus ObfVerProof, and thus V, outputs 0 on all inputs where $i = T$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x^*_{i^*}, i^*))$, where $x^*_{i^*} \leftarrow \mathsf{H.Extract}(\mathsf{td}, \mathsf{H.Hash}(\vec{x}))$. By extraction correctness of $\Pi_{\mathsf{SEH}}$, it cannot be the case that $\mathsf{V}(\mathsf{crs}, \vec{x}, \pi) = 1$ and $j = \mathsf{F}(k_{\mathsf{sel}}, (x_{i^*}, i^*))$. Therefore the challenger in this experiment always outputs 0. □

**Proof of Theorem 6.4.** Theorem 6.4 now follows from Lemmas 6.5, 6.6, and 6.11 to 6.13. Specifically, there exist negligible functions $\delta_1, \delta_2, \delta_3$ such that

$$
\begin{aligned}
\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] &\le \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] + \delta_1(\lambda) && \text{by Lemma 6.5} \\
&\le (T+1) \cdot \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] + \delta_1(\lambda) + \delta_2(\lambda) && \text{by Lemma 6.6} \\
&\le (T+1) \cdot \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] + \delta_1(\lambda) + \delta_2(\lambda) && \text{by Lemma 6.11} \\
&\le (T+1) \left( \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] + \frac{1}{2^\lambda} \right) + \delta_1(\lambda) + \delta_2(\lambda) && \text{by Lemma 6.12} \\
&\le (T+1) \left( \Pr[\mathsf{Hyb}_{5,T}(\mathcal{A}) = 1] + T \cdot \delta_3(\lambda) \right) + \frac{T+1}{2^\lambda} + \delta_1(\lambda) + \delta_2(\lambda) && \text{by Lemma 6.13,}
\end{aligned}
$$

where we have used the fact that $\mathsf{Hyb}_{5,0} \equiv \mathsf{Hyb}_4$. By Lemma 6.26, we have that $\Pr[\mathsf{Hyb}_{5,T}(\mathcal{A}) = 1] = 0$. Since $T = \mathsf{poly}(\lambda)$ and $\delta_1, \delta_2, \delta_3 = \mathsf{negl}(\lambda)$, we conclude that $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$, which completes the proof of adaptive soundness. □

**Theorem 6.27** (Perfect Zero-Knowledge)**.** *Suppose $i\mathcal{O}$ is correct. Then Construction 6.1 satisfies perfect zero-knowledge.*

*Proof.* We construct the simulator as follows:

- $\mathcal{S}_0(1^\lambda, T, C)$: On input the security parameter $1^\lambda$, the batch size $T$, and the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, the simulator samples the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, T, C)$ exactly as in the real scheme. Let $\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, k$ be the underlying hash key, PRG parameters and PPRF keys sampled in Setup. The simulator outputs the crs along with the state $\mathsf{st} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, k)$.
- $\mathcal{S}_1(\mathsf{st}, (x_1, \ldots, x_T))$: On input the state $\mathsf{st} = (\mathsf{hk}, \mathsf{crs}_{\mathsf{G}}, k_{\mathsf{sel}}, k)$ and statements $(x_1, \ldots, x_T)$, the simulator computes $j_i \leftarrow \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$ and selects the smallest $j \in [T+1]$ such that $j \neq j_i$ for all $i \in [T]$. It then computes $k_j \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n+t}, 1^\lambda; \mathsf{F}(k, j))$ and $z_T = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, T)))$. The simulator outputs $\pi = (j, z_T)$.

Take any Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^v \to \{0,1\}$, batch size $T$, and statements $x_1, \ldots, x_T$ and witnesses $w_1, \ldots, w_T$ such that $C(x_i, w_i) = 1$ for all $i \in [T]$. First, observe that the common reference string crs output by $\mathcal{S}_0(1^\lambda, T, C)$ is distributed identically to $\mathsf{Setup}(1^\lambda, T, C)$. It now suffices to consider the proof. By construction, the proof $\pi = (j, z_T)$ output by $\mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T))$ is obtained by evaluating ObfAggProof on inputs $(i, j, \mathsf{dig}, x_i, w_i, \sigma_i, z_{i+1})$. By correctness of $i\mathcal{O}$ and the definition of AggProof and P, this means that $j$ is the smallest value in $[T+1]$ such that $j \neq \mathsf{F}(k_{\mathsf{sel}}, (x_i, i))$ for all $i \in [T]$ and that $z_T = \mathsf{PRG.GenSeed}(\mathsf{crs}_{\mathsf{G}}; \mathsf{F}(k_j, (\mathsf{dig}, T)))$. Thus the proof $\pi = (j, z_T)$ output by $\mathcal{S}_1(\mathsf{st}, (x_1, \ldots, x_T))$ is distributed identically to $\mathsf{P}(\mathsf{crs}, (x_1, \ldots, x_T), (w_1, \ldots, w_T))$. □

## 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 Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 102–132. Springer, Heidelberg, August 2022.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 252–283. Springer, Heidelberg, August 2023.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 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 Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th ACM STOC*, pages 505–514. ACM Press, May / June 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 Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 247–277. Springer, Heidelberg, April / May 2017.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[CGJ+23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 635–668. Springer, Heidelberg, August 2023.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Heidelberg.

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022.

[CLM23] Valerio Cini, Russell W. F. Lai, and Giulio Malavolta. Lattice-based succinct arguments from vanishing polynomials - (extended abstract). In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 72–105. Springer, Heidelberg, August 2023.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS 2010*, pages 310–331. Tsinghua University Press, January 2010.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 54–74. Springer, Heidelberg, March 2012.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *63rd FOCS*, pages 1057–1068. IEEE Computer Society Press, October / November 2022.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[GSWW22] Rachit Garg, Kristin Sheridan, Brent Waters, and David J. Wu. Fully succinct batch arguments for NP from indistinguishability obfuscation. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 526–555. Springer, Heidelberg, November 2022.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 520–549. Springer, Heidelberg, May / June 2022.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in $NC^0$. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[KLV23] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs and PPAD hardness from the decisional Diffie-Hellman assumption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 470–498. Springer, Heidelberg, April 2023.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1545–1552. ACM Press, June 2023.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019.

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Heidelberg, November 2021.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Heidelberg, December 2013.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

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

[NWW24] Shafik Nassar, Brent Waters, and David J. Wu. Monotone policy BARGs from BARGs and additively homomorphic encryption. In *TCC*, 2024.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *63rd FOCS*, pages 1045–1056. IEEE Computer Society Press, October / November 2022.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Heidelberg, August 2022.

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

[WW24b] Brent Waters and David J. Wu. A pure indistinguishability obfuscation approach to adaptively-sound SNARGs for NP. Cryptology ePrint Archive, Paper 2024/933, 2024.

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