# The Power of Rerandomization in Obfustopia

## Collision-Resistant Hash, Somewhere-Extractable BARGs, and More

Shafik Nassar
UT Austin
shafik@cs.utexas.edu

**Abstract**

Indistinguishability obfuscation (iO) combined with one-way functions (OWFs) serves as a powerful foundation for constructing a vast array of cryptographic primitives. However, this combination faces a known black-box barrier established by Asharov and Segev (FOCS '15), which proves the impossibility of constructing collision-resistant hash (CRH) functions. The Asharov-Segev barrier naturally extends to stronger primitives that imply CRH, such as fully homomorphic encryption (FHE) and somewhere-extractable non-interactive batch arguments (seBARGs). This work investigates the power of iO and *rerandomizable* primitives in constructing such "CRH-hard" primitives.

First, we demonstrate the first direct and simple approach to building CRH from iO and rerandomizable OWFs. The only previously known construction was due to Arnon, Ben-David and Yogev (CRYPTO '25), and needed to go through the construction of the adaptively sound SNARG of Waters and Wu (STOC 24'). Using the same approach, we additionally construct a strictly stronger primitive than CRH, which we call perfectly partitionable hash (PPH), from iO and rerandomizable commitments.

Second, we demonstrate the power of rerandomizability for building advanced non-interactive proof systems. Using iO and rerandomizable commitments, we provide a construction of seBARGs with statistical extraction, a security property not achieved by most existing seBARG schemes. By additionally relying on rate-1 fully-homomorphic encryption, we construct the first rate-1 seBARG with statistical extraction. Along the way, we introduce a SNARG that is "sometimes statistically sound", and construct it from iO and rerandomizable commitments.

# Contents


**1 Introduction** **3**
- 1.1 Our Results . . . 3
- 1.2 Discussion . . . 6

**2 Technical Overview** **7**
- 2.1 Partitionable Hash and Collision-Resistance . . . 7
  - 2.1.1 PPH from iO and Rerandomizable Commitments . . . 8
  - 2.1.2 CRH from iO and Rerandomizable OWFs . . . 10
  - 2.1.3 Extractable Partitionable Hash from iO and PPH . . . 11
- 2.2 Somewhere-Extractable BARGs . . . 12

**3 Preliminaries** **14**
- 3.1 Standard Primitives . . . 14
- 3.2 Rerandomizable Primitives . . . 15
- 3.3 Batch Arguments for NP . . . 17

**4 Partitionable Hash** **18**

**5 Extraction from Perfect Partitioning and iO** **20**

**6 Constructing Perfectly Partitionable Hash** **25**
- 6.1 Mode Indistinguishability of Construction 6.1 . . . 27

**7 Constructing Computationally Partitionable Hash** **33**
- 7.1 Mode Indistinguishability of Construction 7.1 . . . 35

**8 Sometimes-Statistically Sound SNARGs** **39**
- 8.1 Construction . . . 40
- 8.2 Mode Indistinguishability of Construction 8.2 . . . 42

**9 Constructing Somewhere-Extractable BARGs** **46**

# 1 Introduction

Ever since the introduction of the punctured programming paradigm [SW14], indistinguishability obfuscation (iO) [BGI+12, GGH+13] paired with one-way functions (OWFs) has emerged as a "central hub" for cryptography, from which many advanced cryptographic primitives can be built.

Surprisingly, one basic primitive still cannot be built from this combination: *collision-resistant hash functions* (CRHFs). Asharov and Segev [AS16] extended the seminal work of Simon [Sim98] and proved that CRHFs are black-box separated from iO and OWFs. This separation naturally extends to all "CRH-hard" primitives from which CRHF can be built in a black-box way, such as fully homomorphic encryption (FHE) [Gen09] and somewhere-extractable non-interactive batch arguments (seBARGs) [CJJ21a].

Adding structure to the underlying primitives is a natural way to bypass barriers. Canetti et al. [CLTV15] showed that rerandomizable encryption and iO can be used to construct FHE. More recently, Waters and Wu [WW24] used rerandomizable OWFs and iO to construct adaptively sound SNARGs with preprocessing, and such SNARGs were later proved sufficient to construct CRHFs (in combination with OWFs) by Arnon et al. [ABY25], thus essentially proving that the combination of iO and rerandomizable OWFs bypasses the CRH barrier. At their core, these works utilize the "perfect rerandomization" of the underlying encryption or OWF to construct CRH-hard primitives, yet the primitives they build are very advanced, and therefore require involved constructions with layers of techniques.

As such, it is natural to seek out the minimal techniques that allow for simple and direct constructions of CRHFs from rerandomizable primitives. We believe that finding a simple construction of CRHFs from iO and rerandomizable primitives serves multiple goals:

1. Isolating the techniques that enable CRH constructions from rerandomizable primitives. By distilling these techniques, we may eventually learn how to "mimic" them using plain OWFs, in order to construct CRHFs from iO and OWFs. Indeed, it might be impossible to eliminate the need for rerandomization or other structure in prior constructions [CLTV15, WW24], even if (non-black box) constructions of CRH from iO and OWFs exist.

2. CRH constructions that go through more complicated primitives inherit the limitations and barriers of those primitives. For example, adaptive SNARGs need to bypass the Gentry-Wichs barrier [GW11], and as such, these constructions rely on a security reduction that is able to solve an NP-complete problem, resulting in a sub-exponential loss in security. There is no reason why this should be inherent to CRH constructions. Therefore a direct CRH construction, even from sub-exponentially secure iO, may serve as a stepping stone towards building CRH from polynomially-secure iO.

3. We believe that getting the simplest construction possible for the sake of simplicity is a valid goal, and determining the minimal assumptions for constructing CRHFs is a fundamental question that deserves to be studied. A sufficiently simple construction from iO and rerandomizable OWFs could even be used as a classroom example on how to utilize iO and rerandomizable primitives.

Finally, it is evident from prior work that iO and rerandomizable primitives have a very strong synergy. As we show in this work, understanding this synergy enables constructions of advanced primitives with properties not previously known from other standard assumptions.

## 1.1 Our Results

**Simple constructions of CRH and friends.** In this work, we present direct CRH constructions from iO and a variety of rerandomizable primitives. Our first construction proves the following result:

**Theorem 1.1** (Informal, see Theorem 7.2)**.** *If sub-exponentially secure indistinguishability obfuscation, sub-exponentially secure OWFs, and polynomially secure rerandomizable OWFs exist, then collision-resistant hash functions exist.*

We reiterate that Theorem 1.1 is already known from combining the works of Waters and Wu [WW24] and Arnon et al. [ABY25]. But our construction is the first *direct* CRH construction from iO and rerandomizable OWFs, which does not go through the heavy machinery of SNARGs and instance compression [HN10], and it is so simple that we can fully describe it right away: the hash key is an obfuscated program which on input $x$, outputs $f(F(x))$, where $F$ is a puncturable pseudorandom function and $f$ is a rerandomizable OWF. While the construction is simple, the security proof utilizes rerandomizable OWFs in a novel way to partition the domain and make the hash key behave differently on different partitions. Crucially, the behavior allows us to argue that finding a collision between the different partitions is computationally difficult.

Next, we define a stronger notion than collision-resistance, which we call *perfect partitioning*. Roughly speaking, the hash key of a *perfectly partitionable hash* (PPH) divides the domain of the hash into disjoint subdomains, each of which maps to a *disjoint* subset in the range, meaning that a collision between different subdomains simply does not exist. Moreover, the hash key should hide the partitioning. This definition resembles function-binding hash [FWW23], but does not require local openings - which are crucial to typical applications of FBH. By replacing the rerandomizable OWF in the construction of Theorem 1.1 with a rerandomizable encryption, we achieve this stronger notion. In fact, we can even use rerandomizable (perfectly binding) commitments, which seem weaker than rerandomizable encryption.

**Theorem 1.2** (Informal, see Theorem 6.2)**.** *If sub-exponentially secure indistinguishability obfuscation, sub-exponentially secure OWFs, and polynomially secure rerandomizable perfectly binding commitments exist, then perfectly partitionable hash functions exist.*

The implication of CRH from rerandomizable encryption and iO is already known [CLTV15]. However, our construction is simpler and more direct since it does not go through the FHE machinery of Canetti et al. [CLTV15]. [1]

As a final contribution in the realm of hashing, we show that PPH can be generically upgraded - using iO - to an *extractable partitionable hash* (EPH): the key generation algorithm additionally outputs a secret trapdoor that extracts from any digest the partition to which it belongs. This is proved in Section 5.

**Rate-1 somewhere proofs of knowledge.** A non-interactive batch argument for NP (BARG) is a proof system with a *common reference string* (CRS) that allows proving many NP statements $x_1, \ldots, x_k$ using a proof of length that is essentially independent of the number of statements. Beyond their immediate application to amortizing the communication cost of NP verification, BARGs have found numerous other applications [CJJ21b, DGKV22, PP22, CW23, KLVW23, BBK+23, BKP+24, BWW24, NWW24, NWW25, DJJ+25, YYS25]. Those applications typically require the BARG to be a *somewhere argument of knowledge*: the CRS can be programmed to be extractable on an index $i^* \in [k]$, meaning that there is a secret trapdoor that allows us to extract a valid witness for statement $x_{i^*}$ from any *efficiently computable* accepting proof. Furthermore, the extracting CRS should be indistinguishable from a normal CRS. Most BARG constructions [CJJ21a, CJJ21b, WW22, CGJ+23] satisfy this property[2]. Recently, Devadas et al. [DJWW25] used

[1] At first, it may seem that the implication of CRH from rerandomizable commitment and iO is new. However, it is actually possible to construct a perfectly hiding commitment from rerandomizable commitments [Rev26]. Such hiding commitments together with iO can be used to get collision-resistance [BDV17].

[2] This property immediately makes seBARG incomparable to adaptive SNARGs. For instance, seBARGs imply CRH, whereas adaptive SNARGs are known from iO and OWFs [WW24].

a seBARG with witness encryption [GGSW13] to get succinct witness encryption for batch languages. Crucial to their application, the BARG needs to be a somewhere *proof* of knowledge: the trapdoor should extract a valid witness from *any* accepting proof, regardless of how it was computed. Essentially, they use such a proof system with a statistical property in order to *compress* an NP witness, thus enabling the application of succinct witness encryption. In general, somewhere statistical properties are "iO-friendly" and enable a plethora of applications, as demonstrated by the success of somewhere-statistically binding hash [HW15]. Unfortunately, only pairing-based seBARG constructions [WW22] satisfy this statistical property. Building on the techniques we developed for constructing PPH, we present a new avenue for constructing somewhere proof of knowledge BARGs from iO and rerandomizable commitments.

**Theorem 1.3** (Informal, see Theorem 9.2)**.** *If sub-exponentially secure indistinguishability obfuscation, sub-exponentially secure one-way functions, and polynomially secure rerandomizable perfectly binding commitments exist, then somewhere proof of knowledge BARGs for* NP *exist.*

Another desirable property of seBARGs is being rate-1 [DGKV22, PP22], i.e., with proof size $n + \mathsf{poly}(\lambda)$ where $n$ is the witness size and $\lambda$ is the security parameter. This allows recursive composition of seBARGs, which enables incrementally verifiable computations and multi-hop signature schemes [DGKV22, PP22, DJJ+25]. None of the direct seBARG constructions [CJJ21a, CJJ21b, WW22, CGJ+23] are rate-1, and while prior works [DGKV22, PP22, BDSZ24] gave generic rate-1 compilers, those compilers inherently lose the statistical extraction property of pairing-based seBARGs or the seBARG from this work. Prior to this work, there was no seBARG construction that was simultaneously rate-1 and a somewhere proof of knowledge. We show that if we assume rate-1 EPH, then our BARG construction is also rate-1. While we do not know how to construct such a hash function from iO and rerandomizable commitments, it is actually easy to construct it from rate-1 FHE [BDGM19, GH19]. Therefore we get the first rate-1 somewhere proof of knowledge BARGs:

**Corollary 1.4** (Informal, see Lemma 9.4)**.** *If sub-exponentially secure indistinguishability obfuscation, sub-exponentially secure one-way functions, polynomially secure rerandomizable perfectly binding commitments, and polynomially secure rate-*1 *FHE exist, then rate-*1 *somewhere proof of knowledge BARGs for* NP *exist.*

**Sometimes-statistically sound SNARGs.** Along the way to building our BARG, we build a proof system that could be of independent interest, which we call a *sometimes-statistically sound SNARG* (SSS-SNARG). In more detail, the CRS of the SSS-SNARG can be either sampled in normal mode, which does not guarantee statistical soundness, or in binding mode on some predicate $\mathsf{pred} : \{0,1\}^n \to \{0,1\}$. In binding mode, the CRS is statistically guaranteed to reject all $x$ such that $\mathsf{pred}(x) = 1$, meaning that there does not exist an accepting proof for any such $x$. We also require mode indistinguishability when the predicate identifies only false statements, i.e., $\forall x : \mathsf{pred}(x) = 1 \implies x \notin \mathcal{L}$, meaning that a CRS that is binding on such predicates is computationally indistinguishable from the normal CRS. Note that we do *not* require the predicate to identify *all* false statements, i.e., $x \notin \mathcal{L} \implies \mathsf{pred}(x) = 1$. We view this as another iO-friendly primitive which enables compressing NP witnesses similar to somewhere proofs of knowledge, and we are able to construct it from iO and rerandomizable commitments via a simple adaptation of the techniques used in Theorem 1.2:

**Theorem 1.5** (Informal, see Theorem 8.3)**.** *If sub-exponentially secure indistinguishability obfuscation, sub-exponentially secure OWFs, and polynomially secure rerandomizable perfectly binding commitments exist, then sometimes-statistically sound SNARGs for* NP *exist.*

We also point out that sometimes-statistically sound SNARGs even for P seem like a challenging primitive to construct without iO, even though (plain) SNARGs for P are known from a variety of standard assumptions [CJJ21b, WW22].

## 1.2 Discussion

**BARGs, SNARGs, and knowledge extraction.** While BARGs are a special case of SNARGs, seBARGs are technically incomparable to SNARGs. Specifically, SNARGs are known from iO and OWFs [SW14, WW25] (in a black-box way), while seBARGs imply CRH (in a black-box way). In terms of end result, one should contrast Theorem 1.3 with the "folklore" construction of seBARGs from *somewhere-extractable hash* (SEH) [HW15] and adaptive SNARGs [WW24, WZ24, WW25]: the prover simply hashes the witnesses and gives a SNARG proof that the digest was computed from witnesses that satisfy the instances.[3] The distinctive advantage of our BARG is that it satisfies the strong notion of *somewhere proof of knowledge* while the folklore construction satisfies only computational extraction, making it harder to use with primitives like iO and WE. For example, the work of Devadas et al. [DJWW25] utilizes a statistically sound BARG to compress the witness and verification of a batch NP relation, thus constructing a succinct witness encryption for batch languages. That being said, we point out the following drawback of our seBARG (and the folklore seBARG): while the proof size is rate-1, the CRS grows with the number of instances. This is not the case for the pairing-based seBARGs [WW22]. It might be possible to get a succinct CRS by incorporating known chaining techniques [GSWW22]. Sometimes-statistically sound SNARGs are related to SNARGs for trapdoor languages, which were featured in some recent works [DJJ+25, BBPS26]; however, as far as we can tell, statistical soundness was never explicitly considered in these works. In addition, SSS-SNARGs can also work for *non*-trapdoor languages, as we demonstrate in this work. In a concurrent and independent work, Dujmovic et al. [DHJQ26] build a SNARG for trapdoor languages from iO and shiftable PRFs and explicitly state that it satisfies statistical soundness once the verification key is punctured.

**Perfectly partitionable hash.** The notion of PPH is very related to function-binding hash [FWW23] (FBH): the main difference is that FBH supports local openings. Our definition does not require local openings, which is why we opted to go with a different name. In addition, our definition supports any partitioning on the domain, whereas FBH only supports local/block functionalities. We also note that PPH can be easily constructed from fully homomorphic encryption (see Section 2 for more details), and as such is already known from iO and rerandomizable encryption. But the challenge in Theorem 1.2 is actually constructing PPH directly, and from rerandomizable commitments - which were not even previously known to imply CRHFs. Extractable partitionable hash resembles previous notions of predicate extractable hash [BBK+23, YYS25, NW26], although again, those notions support local openings and are limited to local functionalities.

**Building rerandomizable primitives.** Waters and Wu [WW24] prove that rerandomizable OWFs can be built from the hardness of the discrete logarithm problem or factoring.[4] Rerandomizable (perfectly binding) commitments and rerandomizable encryption can be built from DDH [Gam84], QR [GM82], DCR [Pai99] and many more.

**(Lack of) Connection between rerandomizable primitives.** As mentioned before, rerandomizable encryption implies rerandomizable (perfectly binding) commitments. But it is not clear how to go the other way and construct rerandomizable encryption from rerandomizable commitments, even assuming iO. On the same note, what is the connection between rerandomizable commitments and rerandomizable OWFs? It might sound surprising at first, but rerandomizable commitments are not known to imply rerandomizable

[3] Note that local openings are not necessary here, so an extractable partitionable hash suffices.

[4] The OWF based on factoring satisfies *statistical* rerandomization with error $2^{-\Omega(m(\lambda))}$ for any polynomial $m(\lambda)$, which suffices for both our construction as well as the Waters-Wu SNARG. We elect to present our results for perfectly rerandomizable primitives for simplicity.

OWFs (and vice versa). The reason is the different nature of the rerandomization properties of those primitives:

- Rerandomizable commitments, similar to rerandomizable encryption, allow the generation of fresh commitments of a message $m$ from a single commitment of $m$. We note that this notion has been used before, for example, by Ananth et al. [ADKL19] who observed that the commitment scheme of Groth et al. [GOS12] is rerandomizable.
- Rerandomization in the context of OWFs means that given a OWF challenge $y^*$, one can generate a fresh challenge $y$. On its own, this property is meaningless since we can always sample an independent OWF challenge. The rerandomized challenge comes with a state st that allows recovering a solution to the original $y^*$ given a solution to the rerandomized challenge $y$. This notion has been used before by Waters and Wu [WW24].

The natural construction of OWFs from commitments is the following: given a bit commitment algorithm $\mathsf{Com}(b;r)$, define the function $f(r) = \mathsf{Com}(0;r)$, where $r$ is the randomness used by the commitment algorithm. While $f$ is indeed a OWF, it is not clear how to rerandomize the challenge in a way that allows solution recovery. Specifically, what would the variable st be? Even if we throw iO into the mix, it is not clear how to get rerandomizable OWFs from rerandomizable commitments and iO. For that reason, the implication of Theorem 1.2 to getting CRH from iO and rerandomizable commitments is not implied by Theorem 1.1.

**Black-box constructions and separations.** It is easy to see that all of our constructions are black-box in the Asharov-Segev sense [AS16], and thus can be used to separate iO and OWFs from any of the rerandomizable primitives we used to get CRH. We discuss those separations below:

- We suspect that the separation from rerandomizable OWFs may be obtained from [WW24, ABY25] in combination with [AS16, BD19]. In more detail, the computational instance compression framework of [ABY25] would be black-box in the Asharov-Segev sense if given a (black-box) preprocessing SNARG for oracle-aided circuit satisfiability. We believe the SNARG of Waters and Wu [WW24] can be upgraded to satisfy this property. Still, the separation we get from our work is more direct and proves that we do not need to go through instance compression and SNARGs to get CRHFs.
- The separation from rerandomizable encryption is known due to Canetti et al. [CLTV15] and the fact that FHE implies CRHFs, in combination with [AS16, BD19].
- The prior separation from rerandomizable commitments follows a different path. It is known that rerandomizable (perfectly binding) commitments imply hard problems in the complexity class SZK. Bitansky et al. [BDV17] proved that iO and OWFs are black-box separated from the hardness of SZK, therefore they cannot be used to get rerandomizable commitments.

# 2 Technical Overview

## 2.1 Partitionable Hash and Collision-Resistance

We now discuss our collision-resistant hash constructions. We begin by defining *perfectly partitionable hash* (PPH). A PPH consists of 3 algorithms:

- A normal key generation algorithm takes a security parameter $\lambda$, an input length $\ell_{\mathsf{in}}$ and a circuit size $s$, and outputs a hash key $\mathsf{hk}$.

- A partitioned key generation algorithm takes, in addition to the previously mentioned parameters, a Boolean circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and outputs a partitioned hash key $\mathsf{hk}^*$.
- A hashing algorithm takes a hash key $\mathsf{hk}$ and a string $x$ of size $\ell_{\mathsf{in}}$ and outputs a digest $\mathsf{dig}$. The digest is required to be much shorter than $\ell_{\mathsf{in}}$. In this work, we consider fully succinct digests, where $|\mathsf{dig}| = \mathsf{poly}(\lambda)$ is independent of $\ell_{\mathsf{in}}$.

The hash should satisfy the two following security properties:

- Mode indistinguishability states that a normal hash key is computationally indistinguishable from a partitioned hash key (for any Boolean circuit $\mathsf{splt}$).
- Perfect partitioning states that for any $x, x' \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $\mathsf{splt}(x) \neq \mathsf{splt}(x')$, the digests computed using the partitioned key $\mathsf{hk}^*$ would never collide, that is, $\mathsf{Hash}(\mathsf{hk}^*, x) \neq \mathsf{Hash}(\mathsf{hk}^*, x')$.

As discussed earlier, PPH is trivially collision resistant: the CRH challenger just guesses the index on which the two colliding inputs differ. This incurs a polynomial loss. Next, we switch to a hybrid where the PPH key is partitioned on that index. This step is indistinguishable to the adversary by the mode indistinguishability of the PPH. Finally, this new experiment is impossible to win by the perfect partitioning of the PPH.
We now move on to constructing PPH. For simplicity, we focus on constructing partitionable hash functions which support $\mathsf{splt}$ that outputs a single bit. However, it is straightforward to extend any construction to support multi-output split functions by concatenation.

**Warm-up: PPH from FHE.** To give intuition, we start by giving a simple construction of PPH from fully homomorphic encryption:

- The normal key generation algorithm samples an FHE key pair $(\mathsf{pk}, \mathsf{sk})$, produces an encryption $\mathsf{ct}_\perp$ of a dummy function $\mathsf{splt}_\perp$ and outputs $\mathsf{hk} = (\mathsf{pk}, \mathsf{ct}_\perp)$.
- The partitioned key generation algorithm does the same except that it replaces the ciphertext $\mathsf{ct}_\perp$ with an encryption $\mathsf{ct}$ of the input function $\mathsf{splt}$.
- The hashing algorithm takes as input a hash key $(\mathsf{pk}, \mathsf{ct})$ and a string $x$, and outputs the homomorphic evaluation $\mathsf{Eval}(\mathsf{pk}, \mathsf{ct}, U_x)$, where $U_x(\cdot)$ is a universal mapping that takes a function $\mathsf{splt}$ and outputs $\mathsf{splt}(x)$. By the compactness of the FHE, we get digest succinctness.

This construction satisfies mode indistinguishability by the CPA security of the FHE scheme. It satisfies perfect partitioning by the correctness of the FHE scheme: if $\mathsf{splt}(x) \neq \mathsf{splt}(x')$, then the digest of $x$ decrypts to a different value than the digest of $x'$, therefore the digests must be different.

### 2.1.1 PPH from iO and Rerandomizable Commitments

To port the previous construction over to iO, the following natural idea comes to mind. Given a perfectly binding commitment $\mathsf{Com}(b; r)$, where $b$ is the committed bit and $r$ is the randomness (we use a semicolon to separate the randomness from the rest of the inputs throughout this work), the hash key in the partitioned mode is an obfuscated program that outputs a commitment of $\mathsf{splt}(x)$. In the normal mode, the hash key just outputs commitments of 0. The randomness for the commitments can be derived from the input $x$ using a puncturable PRF $F$. Namely, the hash keys are obfuscations of the following programs:

**Constants:** A PRF $F : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_r}$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. Outputs $\mathsf{Com}(0; F(x))$.

Figure 1: Normal hashing program.

**Constants:** A PRF $F : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_r}$ and a Boolean circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. Outputs $\mathsf{Com}(\mathsf{splt}(x); F(x))$.

Figure 2: Partitioned hashing program.

The natural strategy for proving mode indistinguishability is to complexity leverage over all $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$, and change the commitment that the program outputs on each $x$, one at a time. Of course, this does not work since the security of the commitment would have to grow with $\ell_{\mathsf{in}}$ in order to accommodate the exponential number of hybrids, which results in long commitments (and thus, non-succinct digests).

**Rerandomization to the rescue.** To solve this problem, we use rerandomizable commitments. Those schemes are equipped with an algorithm $\mathsf{Rerandomize}(\tilde{c}; \rho)$ that takes a commitment $\tilde{c}$ and a uniform string $\rho \xleftarrow{R} \{0,1\}^{\ell_r}$ (without loss of generality, $\rho$ is the same length as a commitment opening) and outputs a commitment $c$. Perfect rerandomization requires that if $\tilde{c}$ is a commitment of a bit $b$, then $c$ is distributed identically to a fresh commitment of $b$. Following a similar strategy to Waters and Wu [WW24], this allows us to complexity leverage over all possible $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$ without having the security parameter of the commitment grow with $\ell_{\mathsf{in}}$. In more detail, let $\tilde{c}$ be a commitment of 0, for now. For each $x^* \in \left\{0, \ldots, 2^{\ell_{\mathsf{in}}}\right\}$, we define a hybrid $\mathsf{Hyb}_{x^*}$ where the hash key is in some "transitional" state: it behaves like a normal (unpartitioned) key for all $x \geq x^*$ (interpret $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ as an integer in $\left\{0, \ldots, 2^{\ell_{\mathsf{in}}} - 1\right\}$) and in an intermediate way (that we describe in Fig. 3) for all $x < x^*$:

**Constants:** A Boolean circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, a puncturable PRF $F : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_r}$, an integer $x^* \in \left\{0, \ldots, 2^{\ell_{\mathsf{in}}}\right\}$ and a commitment $\tilde{c}$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. If $x \geq x^*$, outputs $\mathsf{Com}(0; F(x))$.
2. If $x < x^*$, outputs

$$\begin{cases} \mathsf{Com}(0; F(x)) & \mathsf{splt}(x) = 0 \\ \mathsf{Rerandomize}(\tilde{c}; F(x)) & \mathsf{splt}(x) = 1 \end{cases}$$

Figure 3: Program for the transitional key.

Clearly, the transitional key in $\mathsf{Hyb}_0$ (i.e., $x^* = 0$) is functionally equivalent to the normal key described in Fig. 1, therefore it is computationally indistinguishable just by iO security. Next, we show that $\mathsf{Hyb}_{x^*}$ is computationally indistinguishable from $\mathsf{Hyb}_{x^*+1}$ for all $x^* < 2^{\ell_{\mathsf{in}}}$. If $\mathsf{splt}(x^*) = 0$, the transitional key in $\mathsf{Hyb}_{x^*}$ is functionally equivalent to the transitional key in $\mathsf{Hyb}_{x^*+1}$ (the program outputs $\mathsf{Com}(0; F(x^*))$ on input $x^*$ in both cases). If $\mathsf{splt}(x^*) = 1$:

1. Puncture the PPRF on $x^*$ and hardcode the commitment $c^* = \mathsf{Com}(0; F(x^*))$ that the program outputs on $x^*$. This is possible by punctured correctness and iO security.
2. Set $c^*$ as a uniform fresh commitment of 0. This is possible by the punctured security of the PPRF $F$.

3. Sample $c^* \leftarrow \mathsf{Rerandomize}(\tilde{c})$ as a rerandomization of $\tilde{c}$. This is possible by the perfect rerandomization and the fact that $\tilde{c}$ is a commitment of 0.

4. Set $c^* = \mathsf{Rerandomize}(\tilde{c}; F(x^*))$. Again, this is possible by the punctured security of the PPRF $F$.

5. Finally, we unpuncture the PPRF and remove the hardcoding, which takes us to $\mathsf{Hyb}_{x^*+1}$.

In $\mathsf{Hyb}_{2\ell_{\mathsf{in}}}$, Step 2. is triggered for all $x \in \{0,1\}^{\ell_{\mathsf{in}}}$. To get perfect partitioning, we switch the commitment $\tilde{c}$ with a commitment of 1. This is possible by the computational hiding of the commitment scheme. We claim that the hash key in this hybrid is perfectly partitioned (despite the fact that it is not exactly the program from Fig. 2): if $\mathsf{splt}(x) = 0$, then the hash key outputs a commitment of 0. If $\mathsf{splt}(x) = 1$, then the hash key outputs a commitment of 1 (by the rerandomization property). Those two ranges are disjoint by the perfect binding of the commitment. Crucially, and similar to Waters-Wu [WW24], perfect rerandomization allows us to invoke the hiding security of the commitment only once, yet it affects the behavior of the program on the entire partition. This ensures that the digest remains succinct. PPRF and iO security are invoked an exponential number of times, but those only affect the size of the hash key.

### 2.1.2 CRH from iO and Rerandomizable OWFs

Our next goal is to get collision-resistant hash from iO and a rerandomizable OWF $f : \{0,1\}^n \to \{0,1\}^m$. Perfect rerandomization means that there exists some algorithm $\mathsf{Rerandomize}(\tilde{z}; \rho)$ that takes a OWF challenge $\tilde{z}$ and random coins $\rho \xleftarrow{R} \{0,1\}^n$ (assume without loss of generality that $\rho$ is as long as the input to $f$), and outputs a new challenge $z$ and a state st, such that $z$ is distributed identically to a fresh OWF challenge, and a solution for $z$ can be "combined" with the state st to recover a solution for $\tilde{z}$. One could try to extend the previous approach to work with rerandomizable OWFs by constructing rerandomizable commitments. Even *computationally* binding commitments can be used, yielding the relaxed notion of computational partitioning - which is equivalent to collision-resistance. However, it is not clear how to construct a rerandomizable commitment from rerandomizable OWFs. Specifically, we do not know how to construct a useful notion of "rerandomizable hardcore bits".

Instead of trying to construct a rerandomizable commitment from rerandomizable OWFs and iO, we take a direct approach and try to reduce the computational partitioning security to the rerandomizable OWF security directly. Our construction works as follows. The normal hash key is an obfuscated program that on input $x$, outputs $f(F(x))$, essentially just replacing the function $\mathsf{Com}(0;\cdot)$ from Fig. 1 with $f$. The partitioned key is an obfuscation of the following program:

**Constants:** A PRF $F : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_r}$, a Boolean circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, a OWF challenge $\tilde{z}$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. Outputs $\begin{cases} f(F(x)) & \mathsf{splt}(x) = 0 \\ \mathsf{Rerandomize}(\tilde{z}; F(x)) & \mathsf{splt}(x) = 1 \end{cases}$

Figure 4: Program Partitioned hashing program.

In other words, on one partition the hash key outputs OWF challenges for which it knows the preimages, whereas on the second partition it outputs rerandomizations of a hard OWF challenge, for which it knows st! A collision from two different partitions could exist, but it allows inverting the OWF on the challenge $\tilde{z}$. In more detail, assume we get such a collision for $x, x'$ where $\mathsf{splt}(x) = 0, \mathsf{splt}(x') = 1$ and the digest is

$z$. Since $\mathsf{splt}(x) = 0$, $y = F(x)$ is a preimage of $z$. On the other hand, since $\mathsf{splt}(x') = 1$, we can compute the state st using $(z, \mathsf{st}) = \mathsf{Rerandomize}(\tilde{z}; F(x'))$. Together, $y$ and st can be used to recover a solution for $\tilde{z}$, thus breaking the one-wayness of $f$. Proving that the normal hash key is computationally indistinguishable from a partitioned key is very similar to the PPH construction, so we skip it in this overview. See Section 7 for the formal construction and complete proofs.

### 2.1.3 Extractable Partitionable Hash from iO and PPH

We extend PPH so that it is equipped with an extraction algorithm that allows extracting the partition from the digest. This property is immediately satisfied by the FHE-based PPH construction we described in the warmup. We are interested in this variant because it will help us build the somewhere proof of knowledge BARG in the next section. Namely, the syntax is changed as follows:

- The normal key generation algorithm remains the same. The partitioned hash key generation takes in the circuit splt and now outputs a hash key $\mathsf{hk}^*$ and a trapdoor td.
- The extraction algorithm takes in a digest dig and a trapdoor td, and outputs a bit $b$.

Mode indistinguishability remains the same. The perfect partitioning property is now replaced with a stronger *perfect extraction* property:

$$\forall x \in \{0,1\}^{\ell_{\mathsf{in}}} \ : \ \mathsf{Extract}(\mathsf{td}, \mathsf{Hash}(\mathsf{hk}^*, x)) = \mathsf{splt}(x)$$

We show that perfect partitioning can be used to get perfect extraction. The idea is to have the hash key encrypt $\mathsf{splt}(x)$ using a standard PRF encryption, where the encryption randomness is derived from a PPH. That is, the hash key is an obfuscation of the following program:

**Constants:** A PRF $F : \{0,1\}^{\ell_{\mathsf{out}}} \to \{0,1\}$, a Boolean circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, a PPH $H : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. Hashes $\rho = H(x)$.
2. Outputs $(\rho, F(\rho) \oplus \mathsf{splt}(x))$.

Figure 5: Program Extractable hashing program.

The trapdoor is the PPRF key, and the extraction algorithm on a digest $(\rho, c)$ simply outputs $F(\rho) \oplus c$. Perfect extraction follows trivially from iO functionality preserving.

We can think of the normal hash key as an extractable hash key on the all-zero function, meaning that it outputs $(\rho, F(\rho))$. To argue that both hash keys are indistinguishable, we first partition the underlying PPH on the same function splt. Next, we define a transitional key and go over every value $\rho^* \in \{0,1\}^{\ell_{\mathsf{out}}}$ and do the following:

1. Puncture the PPRF on $\rho^*$ and hardcode $v^* = F(\rho^*)$ in the program.
2. Set $v^* \xleftarrow{R} \{0,1\}$ uniformly.
3. Find $x^*$ such that $H(x^*) = \rho^*$, compute $\beta = \mathsf{splt}(x^*)$, and hardcode the value $v^* \oplus \beta$ in the program. By the perfect partitioning of $H$, this $x^*$ is a representative of all $x \in H^{-1}(\rho^*)$. Since $v^*$ is uniform, the distributions are identical.

4. Now we go back to $v^* = F(\rho^*)$, and we still hardcode $v^* \oplus \beta$. This step requires a non-uniform reduction to PPRF security since the reduction needs to "invert" $H$ on $\rho^*$ in order to compute $\beta$.

5. Finally, we unpuncture the PPRF and remove the hardcoding.

The exact details are in Section 5.

## 2.2 Somewhere-Extractable BARGs

Now we shift our attention to proof systems. Let us recall the "folklore" approach for constructing seBARGs from SNARGs. Assume we have an adaptive SNARG for NP [WW24, WZ24, WW25] and an extractable partitionable hash. Let $\mathcal{R}$ be an NP relation. The prover, which has instances $\mathbf{x} = (x_1, \ldots, x_k)$ and witnesses $\mathbf{w} = (w_1, \ldots, w_k)$, simply hashes the witnesses $\mathsf{dig} = \mathsf{Hash}(\mathbf{w})$, and sends dig and a SNARG proof $\pi$ of the statement

$$\exists \mathbf{w} = (w_1, \ldots, w_k) \ : \ \mathsf{dig} = \mathsf{Hash}(\mathbf{w}) \text{ and } \forall i \in [k], \mathcal{R}(x_i, w_i).$$

This is clearly an NP statement, with the instance being $(\mathbf{x}, \mathsf{dig})$ and the witness being $\mathbf{w}$, so the prover can use the SNARG to generate the proof. Completeness follows by the completeness of the SNARG. To make the BARG extractable on index $i^* \in [k]$, the hash function is sampled to extract the witness in position $i^*$. The BARG extraction algorithm simply uses the hash trapdoor to extract the witness. Mode indistinguishability follows from index hiding of the hash. To prove that the construction is a somewhere argument of knowledge, we use the perfect extraction of the hash to argue that any vector of witnesses $\mathbf{w}^*$ that maps to the digest dig must have the same $w_{i^*}$ in position $i^*$, and moreover that $w_{i^*}$ can be extracted using the trapdoor. In particular, if the extracted witness does not satisfy $\mathcal{R}(x_{i^*}, w_{i^*})$, then the statement $(\mathbf{x}, \mathsf{dig})$ is false, which allows us to appeal to the SNARG soundness. While this guarantees computational extraction, we cannot guarantee statistical extraction if we just rely on a computationally sound SNARG. In particular, when using any of the iO-based SNARGs [WW24, WZ24, WW25], an unbounded prover can produce accepting proofs for any digest.

**Statistical soundness, sometimes.** Of course, requiring statistical soundness for all false statements is too much to ask for unless the polynomial hierarchy collapses [BHZ87, GVW02]. But in the somewhere proof of knowledge security game, the prover sends $(\mathbf{x}, \mathsf{dig})$ such that the relation $\mathcal{R}(x_{i^*}, \mathsf{Extract}(\mathsf{td}, \mathsf{dig}))$ does not hold. The observation is that since we can efficiently identify these instances given the hash trapdoor, requiring the SNARG to be statistically sound on such instances is no longer far-fetched. We call this a *sometimes-statistically sound SNARG* (SSS-SNARG). Specifically, the SNARG should have two setup modes: a normal setup, mirroring the normal setup of a BARG, and a binding setup which takes a predicate, in this case $\mathsf{pred}_{\mathsf{td},i^*}(\mathbf{x}, \mathsf{dig}) := \neg\mathcal{R}(x_{i^*}, \mathsf{Extract}(\mathsf{td}, \mathsf{dig}))$, and outputs a binding CRS. We require the two following properties:

- **Statistical soundness on the predicate:** the binding CRS guarantees that there are no accepting proofs for any instance that satisfies the predicate.
- **Mode indistinguishability:** if the predicate only accepts false statements, then the binding CRS is computationally indistinguishable from the normal CRS.

We can replace the SNARG from the folklore BARG construction with this SSS-SNARG in order to guarantee statistical extraction. In normal mode, the hash and the SNARG are sampled in normal mode. In somewhere proof of knowledge mode, the hash is extracting on index $i^*$ and the SNARG is statistically binding on $\mathsf{pred}_{\mathsf{td},i^*}(\mathbf{x}, \mathsf{dig})$. The two modes are computationally indistinguishable by the mode indistinguishability

of both underlying primitives, and statistical extraction follows by the statistical soundness of the SNARG. All that remains now is to build the SSS-SNARG.

**Building SSS-SNARGs.** Let $\mathcal{R}$ be an NP relation and $\mathcal{L}$ be the corresponding language. The construction follows the Sahai-Waters SNARG framework [SW14]. The CRS consists of two programs:

- $\mathsf{GenProof}(x, w) \to \pi$: takes in an instance $x$ and a witness $w$, and outputs a "signature" on $x$ if the witness is valid.
- $\mathsf{GenInst}(x) \to z$: takes in an instance $x$, and outputs a "challenge" $z$ whose solution is the "signature" on $x$.

Taking inspiration from the Waters-Wu SNARG [WW24], the challenges in our construction will be rerandomizable. However, it is not clear how to use rerandomizable OWFs in order to achieve a statistical property, so we use rerandomizable commitments. Namely, in the normal mode, the challenge will be a rerandomizable commitment of 0, with the "signature" as the randomness used for the commitment, i.e., $\mathsf{GenInst}(x) \to \mathsf{Com}(0; F(x))$. In fact, GenInst is identical to the normal PPH hash key from Fig. 1. The program $\mathsf{GenProof}(x, w)$ simply outputs $\pi = F(x)$ if $(x, w)$ satisfies the relation. The verifier, given an instance $x$ and a proof $\pi$, checks whether $\mathsf{GenInst}(x) = \mathsf{Com}(0; \pi)$. Completeness is straightforward. In binding mode on a predicate pred, the challenge will be a rerandomizable commitment of 0 if $\mathsf{pred}(x) = 0$, and will be a commitment of 1 if $\mathsf{pred}(x) = 1$, where the commitment of 1 is obtained via rerandomizing some canonical commitment $\tilde{c}$. Again, GenInst is identical to the partitioning PPH hash key from Fig. 2. Statistical soundness follows by the perfect binding of the commitment. Mode indistinguishability follows by the same argument for mode indistinguishability of the PPH, and the observation that whenever we puncture the PRF on any $x^*$ such that $\mathsf{pred}(x^*) = 1$, we are guaranteed that the functionality of GenProof does not change. This is due to the assumption that $\mathsf{pred}(x^*) = 1 \implies x^* \notin \mathcal{L}$, therefore GenProof was never going to evaluate $F(x^*)$ anyway.

**Preprocessing SNARGs and collision resistance.** As in [WW24], our SNARG is preprocessing, and the GenInst program is identical to the PPH hash key program. This is no coincidence. A collision between a valid and an invalid instance would immediately break soundness: a prover could produce a proof for the valid instance and reuse it for the invalid one. Collision resistance of the preprocessing procedure between those partitions is therefore *necessary* for soundness. We view this as an alternative explanation for why preprocessing SNARGs yield CRH [ABY25].

**Achieving rate-1.** The proof in our BARG construction consists of the SNARG proof $\pi$ and the digest dig. The size of $\pi$ only depends on the security parameter. If we use our EPH from iO and rerandomizable commitments, the digest will have size $n \cdot \mathsf{poly}(\lambda)$, which suffices for standard BARG succinctness. But to get rate-1, we need the digest to have length $n + \mathsf{poly}(\lambda)$. Luckily, the warmup construction of PPH from FHE does support extraction and can achieve rate-1 digests if the FHE is rate-1. Therefore, combining our seBARG with known rate-1 FHE constructions [BDGM19, GH19], we get the first construction of rate-1 somewhere proof of knowledge BARGs.

**Remark 2.1.** All of our constructions would work with *statistically rerandomizable primitives* with a sufficiently small rerandomization error, without any additional conceptual modifications. The only difference is that we would have to keep track of multiple additional error parameters. For that reason, we choose to present all of our definitions and proofs using perfect rerandomization.

# 3 Preliminaries

For any positive integer $n \in \mathbb{N}$, we denote $[n] := \{1, \ldots, n\}$. The security parameter will be denoted by $\lambda$. A function $\varepsilon(\lambda)$ is called negligible if for every positive constant $c$, we have $\varepsilon(\lambda) = o(\lambda^{-c})$. We denote this by writing $\varepsilon(\lambda) = \mathsf{negl}(\lambda)$. A function $f(\lambda)$ is polynomial if there exists a positive constant $c$ such that $f(\lambda) = O(\lambda^c)$. We denote this by writing $f(\lambda) = \mathsf{poly}(\lambda)$. For any set $S$, when we write $x \xleftarrow{R} S$ we mean that $x$ is sampled uniformly from $S$. For convenience, we write $U_n$ to represent the uniform distribution over bit strings of length $n$. For any string $x \in \{0,1\}^n$, we view $x$ as both a bit-string and an integer in the set $\{0, \ldots, 2^n - 1\}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input.

## 3.1 Standard Primitives

In this section, we recall the definition of a few standard cryptographic building blocks we use in this work.

**Definition 3.1** (Indistinguishability Obfuscation [BGI+12, GGH+13])**.** Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of polynomial-size circuits. An indistinguishability obfuscator $i\mathcal{O}$ for $\mathcal{C}$ is an efficient algorithm that takes as input the security parameter $\lambda$, a circuit $C \in \mathcal{C}_\lambda$ and outputs a circuit $C'$. An $i\mathcal{O}$ scheme should satisfy the following properties:

- **Functionality-preserving:** For all security parameters $\lambda \in \mathbb{N}$, all $C \in \mathcal{C}_\lambda$, all $C'$ in the support of $i\mathcal{O}(1^\lambda, C)$, and all inputs $x$, we have that $C'(x) = C(x)$.
- **Security:** We say that $i\mathcal{O}$ is $\varepsilon$-secure if for all polynomial time adversaries $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$ and all functionally equivalent $C_0, C_1 \in \mathcal{C}_\lambda$

$$\left| \Pr[\mathcal{A}(\mathsf{st}, i\mathcal{O}(1^\lambda, C_0)) = 1] - \Pr[\mathcal{A}(\mathsf{st}, i\mathcal{O}(1^\lambda, C_1)) = 1] \right| \leq \varepsilon(\lambda),$$

**Definition 3.2** (Puncturable PRF [BW13, KPTZ13, BGI14])**.** A puncturable pseudorandom function consists of a tuple of efficient algorithms $\Pi_{\mathsf{PPRF}} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Puncture})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to K$: On input a security parameter $\lambda$, an input length $\ell_{\mathsf{in}}$, and an output length $\ell_{\mathsf{out}}$, the key generation algorithm outputs a key $K$. We assume the key contains an implicit description of $\ell_{\mathsf{in}}$ and $\ell_{\mathsf{out}}$.
- $\mathsf{Puncture}(K, x^*) \to K\{x^*\}$: On input a key $K$ and a string $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$, the puncture algorithm outputs a punctured key $K\{x^*\}$. We assume the punctured key contains an implicit description of $\ell_{\mathsf{in}}$ and $\ell_{\mathsf{out}}$.
- $\mathsf{Eval}(K, x) \to y$: On input a key $K$ and an input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, the evaluation algorithm outputs a value $y \in \{0,1\}^{\ell_{\mathsf{out}}}$. We sometimes write $F(K, x)$ to denote $\mathsf{Eval}(K, x)$.

In addition, $\Pi_{\mathsf{PPRF}}$ should satisfy the following properties:

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

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

- **Punctured pseudorandomness:** For a bit $b \in \{0,1\}$ and a security parameter $\lambda$, we define the (selective) punctured pseudorandomness game between an adversary $\mathcal{A}$ and a challenger as follows:

– On input a parameter $\lambda$, the adversary outputs an input length $1^{\ell_{\text{in}}}$ and an output length $1^{\ell_{\text{out}}}$, and commits to a string $x^* \in \{0,1\}^{\ell_{\text{in}}}$.

– The challenger samples a key $K \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}})$, punctures $K\{x^*\} = \mathsf{Puncture}(K, x^*)$, and gives the punctured key $K\{x^*\}$ to $\mathcal{A}$.

– If $b = 0$, then the challenger sets $y^* = \mathsf{Eval}(K, x^*)$. Otherwise it samples $y^* \xleftarrow{R} \{0,1\}^{\ell_{\text{out}}}$. In any case, the challenger gives $y^*$ to $\mathcal{A}$.

– The adversary outputs a guess $b' \in \{0,1\}$, which is the output of the experiment.

We say that $\Pi_{\mathsf{PPRF}}$ satisfies $\varepsilon$-punctured pseudorandomness if for all polynomial time adversaries $\mathcal{A}$, there exists $\Lambda \in \mathbb{N}$ such that for $\lambda \geq \Lambda$

$$\left|\Pr[\mathsf{ExptPPRF}_{\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{ExptPPRF}_{\mathcal{A}}(\lambda, 1) = 1]\right| \leq \varepsilon(\lambda)$$

## 3.2 Rerandomizable Primitives

In this section, we define the rerandomizable primitives we use in this work.

**Definition 3.3** (Rerandomizable Perfectly-Binding Commitments)**.** A rerandomizable bit-commitment scheme $\Pi_{\mathsf{Com}}$ is a tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Rerandomize})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input a security parameter $\lambda$, outputs the public parameters pp.
- $\mathsf{Com}(\mathsf{pp}, x; r) \to \mathsf{com} \in \{0,1\}^{\ell_{\mathsf{com}}}$: On input public parameters pp, a bit $x$, and random coins $r$, the commitment algorithm outputs a commitment com.
- $\mathsf{Rerandomize}(\mathsf{pp}, \mathsf{com}; r') \to \mathsf{com}'$: On input public parameters pp, a commitment com, and random coins $r'$, the rerandomization algorithm outputs another commitment com′. Without loss of generality, we assume that Com and Rerandomize take the same number of random coins.

$\Pi_{\mathsf{Com}}$ should satisfy the following properties:

- **Perfect binding:** For each $\lambda \in \mathbb{N}$, it holds

$$\Pr[\exists \mathsf{com}, r_0, r_1 \text{ s.t. } \forall b \in \{0,1\}, \mathsf{Com}(\mathsf{pp}, b; r_b) = \mathsf{com} : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)] = 0.$$

- **Computational hiding:** We say that $\Pi_{\mathsf{Com}}$ is computationally hiding if any efficient algorithm $\mathcal{A}$ has negligible advantage in distinguishing the following distributions

$$\left\{(\mathsf{pp}, \mathsf{com}) : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{com} \leftarrow \mathsf{Com}(\mathsf{pp}, 0; U_{\ell_r}) \end{array}\right\}$$

$$\left\{(\mathsf{pp}, \mathsf{com}) : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{com} \leftarrow \mathsf{Com}(\mathsf{pp}, 1; U_{\ell_r}) \end{array}\right\}$$

- **Perfect rerandomization:** For each $\lambda \in \mathbb{N}$ and each $x \in \{0,1\}$, the following distributions are identical

$$\left\{(\mathsf{pp}, \mathsf{com}, \mathsf{com}') : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{com} \leftarrow \mathsf{Com}(\mathsf{pp}, x; U_{\ell_r}) \\ \mathsf{com}' \leftarrow \mathsf{Com}(\mathsf{pp}, x; U_{\ell_r}) \end{array}\right\}$$

$$\left\{ (\mathsf{pp}, \mathsf{com}, \mathsf{com}') : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{com} \leftarrow \mathsf{Com}(\mathsf{pp}, x; U_{\ell_r}) \\ \mathsf{com}' \leftarrow \mathsf{Rerandomize}(\mathsf{pp}, \mathsf{com}; U_{\ell_r}) \end{array} \right\}$$

In particular, perfect rerandomization in this case implies another property called **rerandomization correctness**, which states that for any commitment com of a bit $b$, the rerandomization com′ is also a commitment of $b$.

**Definition 3.4** (Rerandomizable One-Way Functions, adapted from [WW24])**.** A rerandomizable one-way function $\Pi_{\mathsf{ROWF}}$ is a tuple of efficient algorithms (Setup, Rerandomize, RecoverSolution) with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to f$: On input a security parameter $\lambda$, outputs a Boolean circuit $f : \{0,1\}^n \to \{0,1\}^m$. Note that since Setup is efficient, we can efficiently evaluate $f$ on any input.
- $\mathsf{Rerandomize}(f, z; r) \to (z', \mathsf{st})$: On input a Boolean circuit $f : \{0,1\}^n \to \{0,1\}^m$, an image $z \in \{0,1\}^m$, and random coins $r$, outputs a rerandomized image $z'$ and a state st. Without loss of generality, assume that $r$ is $n$ bits long.
- $\mathsf{RecoverSolution}(f, y', \mathsf{st}) \to y$: On input a Boolean circuit $f : \{0,1\}^n \to \{0,1\}^m$, a preimage $y' \in \{0,1\}^n$, and a state st, outputs a preimage $y$.

$\Pi_{\mathsf{ROWF}}$ should satisfy the following properties:

- **Rerandomization correctness:** For each $\lambda \in \mathbb{N}$ and each $y, y' \in \{0,1\}^n$, we have

$$\Pr\left[ f(y') = z' \Rightarrow f(y^*) = f(y) : \begin{array}{c} f \leftarrow \mathsf{Setup}(1^\lambda) \\ (z', \mathsf{st}) \leftarrow \mathsf{Rerandomize}(f, f(y); U_n) \\ y^* = \mathsf{RecoverSolution}(f, y', \mathsf{st}) \end{array} \right] = 1.$$

- **One-wayness:** For every efficient adversary $\mathcal{A}$, the following holds

$$\Pr\left[ f\big(\mathcal{A}(1^\lambda, f, z)\big) = z : \begin{array}{c} f \leftarrow \mathsf{Setup}(1^\lambda) \\ y \xleftarrow{R} \{0,1\}^n \\ z = f(y) \end{array} \right] = \mathsf{negl}(\lambda).$$

- **Perfect rerandomization:** For each $\lambda \in \mathbb{N}$, the following distributions are identical

$$\left\{ (f, f(y), f(y')) : \begin{array}{c} f \leftarrow \mathsf{Setup}(1^\lambda) \\ y, y' \xleftarrow{R} \{0,1\}^n \end{array} \right\}$$

$$\left\{ (f, f(y), z') : \begin{array}{c} f \leftarrow \mathsf{Setup}(1^\lambda) \\ y \xleftarrow{R} \{0,1\}^n \\ z = f(y) \\ (z', \mathsf{st}) \leftarrow \mathsf{Rerandomize}(f, z; U_n) \end{array} \right\}$$

### 3.3 Batch Arguments for NP

In this section, we define non-interactive batch arguments for NP.

**Definition 3.5** (Somewhere-extractable BARG)**.** A somewhere-extractable non-interactive batch argument (BARG) for any NP relation $\mathcal{R}$ is a tuple of efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{TrapGen}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^k, 1^n) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$, and instance size $n \in \mathbb{N}$, the generator algorithm outputs a common reference string crs. For simplicity, we assume the witness for a statement of size $n$ also has the same size $n$.
- $\mathsf{Prove}(\mathsf{crs}, (x_1, \ldots, x_k), (w_1, \ldots, w_k)) \to \pi$: On input the common reference string crs, statements $x_1, \ldots, x_k \in \{0,1\}^n$, and witnesses $w_1, \ldots, w_k \in \{0,1\}^h$, the prove algorithm outputs a proof $\pi$.
- $\mathsf{Verify}(\mathsf{crs}, (x_1, \ldots, x_k), \pi) \to b$: On input the common reference string crs, statements $x_1, \ldots, x_k \in \{0,1\}^n$ and a proof $\pi$, the verification algorithm outputs a bit $b \in \{0,1\}$.
- $\mathsf{TrapGen}(1^\lambda, 1^k, 1^n, i^*) \to (\mathsf{crs}, \mathsf{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $k \in \mathbb{N}$, instance size $n \in \mathbb{N}$, and an index $i^* \in [k]$, the trapdoor generator algorithm outputs a common reference string crs, and an extraction trapdoor td.
- $\mathsf{Extract}(\mathsf{td}, \pi) \to w$. On input the trapdoor td and a proof $\pi$, the extraction algorithm outputs a witness $w$.

Moreover, $\Pi_{\mathsf{BARG}}$ should satisfy the following properties:

- **Completeness:** For all $\lambda, k, n \in \mathbb{N}$, all statements $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^{kn}$ and witnesses $\mathbf{w} = (w_1, \ldots, w_k) \in \{0,1\}^{kn}$ where $(x_i, w_i) \in \mathcal{R}$ for all $i \in [k]$,

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

- **Succinctness:** There exists a fixed polynomial poly such that for all $\lambda, k, n \in \mathbb{N}$ and all crs in the support of $\mathsf{Gen}(1^\lambda, 1^k, 1^n)$, the length of the proof $\pi$ output by $\mathsf{Prove}(\mathsf{crs}, \cdot, \cdot)$ is bounded by $\mathsf{poly}(\lambda, n, \log k)$.
- **Index hiding:** For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, define the index hiding experiment as follows:
  1. Algorithm $\mathcal{A}(1^\lambda)$ starts by sending the number of instances $1^k$, the instance size $1^n$, and an index $i^* \in [k]$ to the challenger.
  2. If $b = 0$, the challenger gives $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^k, 1^n)$ to $\mathcal{A}$. If $b = 1$, the challenger samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, 1^k, 1^n, i^*)$ and gives crs to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  Then, $\Pi_{\mathsf{BARG}}$ satisfies index hiding if for every efficient adversary $\mathcal{A}$, the following holds

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

- **Somewhere proof of knowledge:** For an adversary $\mathcal{A}$, define the somewhere extractable security game as follows:

1. Algorithm $\mathcal{A}(1^\lambda)$ starts by sending the number of instances $1^k$, the instance size $1^n$, and an index $i^* \in [k]$ to the challenger.
2. The challenger samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda, 1^k, 1^n, i^*)$ and gives $\mathsf{crs}, \mathsf{td}$ to $\mathcal{A}$.
3. Algorithm $\mathcal{A}$ outputs statements $x_1, \ldots, x_k \in \{0,1\}^n$, and a proof $\pi$.
4. The output of the game is $b = 1$ if $\mathsf{Verify}(\mathsf{crs}, (x_1, \ldots, x_k), \pi) = 1$ and $(x_{i^*}, w) \notin \mathcal{R}$ where $w \leftarrow \mathsf{Extract}(\mathsf{td}, \pi)$. Otherwise, the output is $b = 0$.

Then $\Pi_{\mathsf{BARG}}$ is a somewhere proof of knowledge if for any (unbounded) adversary $\mathcal{A}$, we have $\Pr[b = 1] = 0$ in the somewhere extractable game.

# 4 Partitionable Hash

In this section, we introduce partitionable hash functions. In Definition 4.1, we give two flavors: perfect partitioning and computational partitioning. Either of the two flavors suffices to prove collision-resistance by a standard guessing argument.

**Definition 4.1** (Partitionable Hash)**.** A partitionable hash $\Pi_{\mathsf{PH}}$ is a tuple of efficient algorithms with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s) \to \mathsf{hk}$ : On input a security parameter $\lambda$, a length parameter $\ell_{\mathsf{in}}$, and circuit size bound $s$, outputs a hash key $\mathsf{hk}$.
- $\mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}) \to \mathsf{hk}$ : On input a security parameter $\lambda$, a length parameter $\ell_{\mathsf{in}}$, and a Boolean function $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, outputs a hash key $\mathsf{hk}$.
- $\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{dig}$: On input a hash key $\mathsf{hk}$ and a string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, outputs a digest $\mathsf{dig}$. This algorithm is deterministic.

The scheme should satisfy the following properties:

- **Digest succinctness:** There exists a polynomial poly such that for each $\lambda, \ell_{\mathsf{in}} \in \mathbb{N}$, each $\mathsf{hk}$ in the range of $\mathsf{Gen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s)$, and each $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, it holds that

$$|\mathsf{Hash}(\mathsf{hk}, x)| = \mathsf{poly}(\lambda).$$

  In particular, the digest length is independent of $\ell_{\mathsf{in}}$.

- **Partitioning security:** For any $\lambda \in \mathbb{N}$, and any algorithm $\mathcal{A}$, we define the following experiment $\mathsf{ExptPS}(\lambda, \mathcal{A})$:

  1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}$ to the challenger.
  2. The challenger samples a hash key $\mathsf{hk} \leftarrow \mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt})$ and sends $\mathsf{hk}$ to $\mathcal{A}$.
  3. $\mathcal{A}$ outputs $x, x' \in \{0,1\}^{\ell_{\mathsf{in}}}$.
  4. The output of the experiment is 1 if and only if $\mathsf{splt}(x) \neq \mathsf{splt}(x')$ and $\mathsf{Hash}(\mathsf{hk}, x) = \mathsf{Hash}(\mathsf{hk}, x')$.

**Perfect Partitioning:** We say that $\Pi_{\mathsf{PH}}$ satisfies *perfect partitioning* if for any unbounded adversary $\mathcal{A}$, we have $\Pr[\mathsf{ExptPS}(\lambda, \mathcal{A}) = 1] = 0$.

**Computational Partitioning:** We say that $\Pi_{\mathsf{PH}}$ satisfies *computational partitioning* if for any bounded adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that $\Pr[\mathsf{ExptPS}(\lambda, \mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

- **Mode indistinguishability:** For any $\lambda \in \mathbb{N}$ and any algorithm $\mathcal{A}$, we define the following experiment which is parameterized by a bit $b \in \{0,1\}$:

  1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, \mathsf{splt}$ to the challenger.
  2. The challenger samples $\mathsf{hk}_0 \leftarrow \mathsf{Gen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s)$ and $\mathsf{hk}_1 \leftarrow \mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt})$.
  3. The challenger sends $\mathsf{hk}_b$ to $\mathcal{A}$.
  4. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

  We say that $\Pi_{\mathsf{PH}}$ satisfies $\varepsilon$-mode indistinguishability if for any efficient $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that

  $$|\Pr[b' = 0 : b = 0] - \Pr[b' = 0 : b = 1]| \leq \varepsilon(\lambda).$$

  We say that $\Pi_{\mathsf{PH}}$ satisfies polynomial mode indistinguishability if for every efficient $\mathcal{A}$, the above probability is negligible.

We now extend perfect partitioning to have an extraction algorithm which outputs the partition to which the digest belongs. This will be useful for constructing somewhere-extractable BARGs in Section 9.

**Definition 4.2** (Extractable Partitionable Hash)**.** An *extractable partitionable hash* (EPH) is a tuple of algorithms $(\mathsf{Gen}, \mathsf{GenPart}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s) \to \mathsf{hk}$ : On input a security parameter $\lambda$, a length parameter $\ell_{\mathsf{in}}$, and circuit size bound $s$, outputs a hash key $\mathsf{hk}$.
- $\mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}) \to (\mathsf{hk}, \mathsf{td})$: On input a security parameter $\lambda$, a length parameter $\ell_{\mathsf{in}}$, a circuit size bound $s$, and a Boolean function $\mathsf{splt}$, outputs a hash key $\mathsf{hk}$ and a trapdoor $\mathsf{td}$.
- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) \to b \in \{0,1\}$: On input a trapdoor $\mathsf{td}$ and a digest $\mathsf{dig}$, outputs a bit $b$. This algorithm is deterministic.

We first require that $(\mathsf{Gen}, \mathsf{GenPart}_{\mathsf{hk}}, \mathsf{Extract})$ is a perfectly partitionable hash, where $\mathsf{GenPart}_{\mathsf{hk}}$ is the algorithm that runs GenPart and only outputs the hash key $\mathsf{hk}$. In addition to the properties of a partitionable hash, it should satisfy:

- **Perfect extraction:** For any $\lambda, \ell_{\mathsf{in}}, s \in \mathbb{N}$, any function $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, any $(\mathsf{hk}, \mathsf{td})$ in the range of $\mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt})$, and any $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, it holds that

  $$\mathsf{Extract}(\mathsf{td}, \mathsf{Hash}(\mathsf{hk}, x)) = \mathsf{splt}(x).$$

**Remark 4.3.** Observe that perfect extraction implies the perfect partitioning security from Definition 4.1: for any $x, x' \in \{0,1\}^{\ell_{\mathsf{in}}}$, if $\mathsf{splt}(x) \neq \mathsf{splt}(x')$ then $\mathsf{Extract}(\mathsf{td}, \mathsf{Hash}(\mathsf{hk}, x)) \neq \mathsf{Extract}(\mathsf{td}, \mathsf{Hash}(\mathsf{hk}, x'))$ by perfect extraction, which implies that $\mathsf{Hash}(\mathsf{hk}, x) \neq \mathsf{Hash}(\mathsf{hk}, x')$.

**Remark 4.4** (Multi-output Partitioning Functions)**.** Definitions 4.1 and 4.2 can be extended to support multi-output partitioning functions. For partitioning functions with $\ell_{\text{out}}$ output bits, digest succinctness requires the digest to have length $\ell_{\text{out}} \cdot \text{poly}(\lambda)$. Any construction that satisfies Definitions 4.1 and 4.2 can be extended to support multi-output partitioning functions by simple concatenation. A *rate*-1 partitionable hash has digests of length $\ell_{\text{out}} + \text{poly}(\lambda)$.

# 5 Extraction from Perfect Partitioning and iO

In this section, we show how to construct an extractable partitionable hash from indistinguishability obfuscation, puncturable PRFs, and perfectly partitionable hash.

**Construction 5.1.** The construction uses the following ingredients:

- Indistinguishability obfuscator $i\mathcal{O}$ defined in Definition 3.1.
- A puncturable PRF $\Pi_{\mathsf{PPRF}} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Puncture})$ defined in Definition 3.2.
- A perfectly partitionable hash $\Pi_{\mathsf{PH}} = (\mathsf{Gen}, \mathsf{GenPart}, \mathsf{Hash})$ defined in Definition 4.1.

The algorithms are defined as follows:

- $\mathsf{Gen}(1^\lambda, 1^{\ell_{\text{in}}}, 1^s)$:
  1. Samples a standard perfectly partitionable hash key $\mathsf{hk}' \leftarrow \mathsf{PH.Gen}(1^\lambda, 1^{\ell_{\text{in}}}, 1^{s'})$. Let $\ell$ be the digest length of the hash. The parameter $s'$ will be chosen later.
  2. Samples an encryption PPRF key $K_{\mathsf{Enc}} \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^\ell, 1^1)$.
  3. The hash key is an obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}])$ where pHash is described in Fig. 6.
  4. Outputs hk.

**Constants:** A hash key $\mathsf{hk}'$, an encryption PPRF key $K_{\mathsf{Enc}}$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\text{in}}}$.

1. $\rho = \mathsf{PH.Hash}(\mathsf{hk}', x)$.
2. Outputs $\left(\rho, \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho)\right)$.

Figure 6: Program $\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}]$.

- $\mathsf{GenPart}(1^\lambda, 1^{\ell_{\text{in}}}, 1^s, \mathsf{splt})$:
  1. Samples a partitioned hash key $\mathsf{hk}' \leftarrow \mathsf{PH.GenPart}(1^\lambda, 1^{\ell_{\text{in}}}, 1^{s'}, \mathsf{splt})$. Let $\ell$ be the digest length of the hash.
  2. Samples an encryption PPRF key $K_{\mathsf{Enc}} \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^\ell, 1^1)$.
  3. The hash key is an obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}])$ where pHash is described in Fig. 7.

4. Outputs the hash key hk and the trapdoor $\mathsf{td} = K_{\mathsf{Enc}}$.

**Constants:** A hash key $\mathsf{hk}'$, an encryption PPRF key $K_{\mathsf{Enc}}$, a function splt.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. $\rho = \mathsf{PH.Hash}(\mathsf{hk}', x)$.
2. Outputs $\big(\rho, \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho) \oplus \mathsf{splt}(x)\big)$.

Figure 7: Program $\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}]$.

- $\mathsf{Hash}(\mathsf{hk}, x)$:
  1. Interprets hk as a program and outputs $\mathsf{hk}(x)$.
- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$:
  1. Parses $\mathsf{td} = K_{\mathsf{Enc}}$ and $\mathsf{dig} = (\rho, v)$.
  2. Outputs $v \oplus \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho)$.

**Theorem 5.2.** *Assume that $i\mathcal{O}$ is a sub-exponentially secure indistinguishability obfuscation, $\Pi_{\mathsf{PPRF}}$ is a sub-exponentially secure puncturable PRF, and $\Pi_{\mathsf{PH}}$ is a polynomially secure perfectly partitionable hash. Then Construction 5.1 is an extractable partitionable hash.*

Theorem 5.2 will follow from Lemmas 5.3 to 5.5.

**Lemma 5.3** (Digest succinctness)**.** *If $\Pi_{\mathsf{PH}}$ is succinct then Construction 5.1 is succinct.*

*Proof.* The digest dig consists of a $\Pi_{\mathsf{PH}}$ digest $\rho$ and a 1-bit ciphertext $v$. If $\Pi_{\mathsf{PH}}$ satisfies succinctness, then $\rho$ is $\mathsf{poly}(\lambda)$ bits long. The total digest length is $\mathsf{poly}(\lambda)$, which is independent of $\ell_{\mathsf{in}}$. The claim follows. □

**Lemma 5.4** (Perfect Extraction)**.** *Construction 5.1 satisfies perfect extraction.*

*Proof.* Fix any $\lambda, \ell_{\mathsf{in}}, s \in \mathbb{N}$ and any function $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$. Let $(\mathsf{hk}, \mathsf{td})$ be any key-trapdoor pair in the range of $\mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt})$, and let $x \in \{0,1\}^{\ell_{\mathsf{in}}}$. By construction, the program hk computes $\rho = \mathsf{PH.Hash}(\mathsf{hk}', x)$ and outputs $\mathsf{dig} = (\rho, v)$ where $v = \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho) \oplus \mathsf{splt}(x)$. The extraction algorithm parses $\mathsf{td} = K_{\mathsf{Enc}}$ and outputs $v \oplus \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho)$, which simplifies to $\mathsf{splt}(x)$. Perfect extraction holds. □

**Lemma 5.5** (Mode Indistinguishability)**.** *Assume that $i\mathcal{O}$ is $2^{-\lambda^\delta_{i\mathcal{O}}}$-secure, $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda^\delta_{\mathsf{PPRF}}}$-punctured pseudorandomness, and $\Pi_{\mathsf{PH}}$ satisfies polynomial mode indistinguishability. If we set $\lambda_{i\mathcal{O}}(\lambda, \ell) = \lambda_{\mathsf{PPRF}}(\lambda, \ell) = (\lambda + \ell)^{1/\delta}$ then Construction 5.1 satisfies mode indistinguishability.*

*Proof.* Let $\mathcal{A}$ be an adversary for the mode indistinguishability game of Construction 5.1. We define a sequence of hybrids:

- $\mathsf{Real}_0$: This is the mode indistinguishability experiment where $b = 0$. The challenger does the following:
  1. Samples $\mathsf{hk}' \leftarrow \mathsf{PH.Gen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{s'})$.

2. Samples $K_{\mathsf{Enc}} \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell}, 1^{1})$.
3. Samples $\mathsf{hk} \leftarrow i\mathcal{O}(\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}])$ as defined in Fig. 6.
4. Sends $\mathsf{hk}$ to $\mathcal{A}$.

- $\mathsf{Hyb}'$: The same as $\mathsf{Real}_0$, except that the challenger samples a partitioned hash key as follows $\mathsf{hk}' \leftarrow \mathsf{PH.GenPart}(1^{\lambda}, 1^{\ell_{\mathsf{in}}}, 1^{s}, \mathsf{splt})$. Note that the obfuscated program still outputs an encryption of 0.
- $\mathsf{Real}_1$: This is the mode indistinguishability experiment where $b = 1$. The challenger does the following:
  1. Samples $\mathsf{hk}' \leftarrow \mathsf{PH.GenPart}(1^{\lambda}, 1^{\ell_{\mathsf{in}}}, 1^{s}, \mathsf{splt})$.
  2. Samples $K_{\mathsf{Enc}} \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell}, 1^{1})$.
  3. Samples $\mathsf{hk} \leftarrow i\mathcal{O}(\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}])$ as defined in Fig. 7.
  4. Sends $\mathsf{hk}$ to $\mathcal{A}$.

The indistinguishability of $\mathsf{Real}_0$ and $\mathsf{Hyb}'$ follows from the mode indistinguishability of $\Pi_{\mathsf{PH}}$.

**Claim 5.6.** *Assume that $\Pi_{\mathsf{PH}}$ satisfies polynomial mode indistinguishability. Then for any adversary $\mathcal{A}$, we have*

$$|\Pr[\mathsf{Real}_0 = 1] - \Pr[\mathsf{Hyb}' = 1]| = \mathsf{negl}(\lambda).$$

The proof of Claim 5.6 is straightforward. To get from $\mathsf{Hyb}'$ to $\mathsf{Real}_1$, we define an inner sequence of hybrids parameterized by $\rho^* \in \{0, \ldots, 2^{\ell}\}$:

- $\mathsf{Hyb}_{\rho^*}$: The program outputs an encryption of $\mathsf{splt}(x)$ for $\rho < \rho^*$, and an encryption of 0 for $\rho \geq \rho^*$. That is:
  3. Samples $\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, \rho^*]$ as defined in Fig. 8.

**Constants:** A hash key $\mathsf{hk}'$, an encryption PPRF key $K_{\mathsf{Enc}}$, a function splt, an integer $\rho^*$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. $\rho = \mathsf{PH.Hash}(\mathsf{hk}', x)$.
2. $\mathsf{dig} = \begin{cases} (\rho, \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho) \oplus \mathsf{splt}(x)) & \rho < \rho^* \\ (\rho, \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho) \oplus 0) & \rho \geq \rho^* \end{cases}$.
3. Outputs dig.

Figure 8: Program $\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, \rho^*]$.

By construction, we have that program $\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, 0]$ from Fig. 8 is functionally equivalent to the obfuscated program in $\mathsf{Hyb}'$, and that program $\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, 2^{\ell}]$ from Fig. 8 is functionally equivalent to $\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}]$ from Fig. 7. Therefore the following claims follow by iO security:

**Claim 5.7.** *Assume that $i\mathcal{O}$ is polynomially secure. Then for any adversary $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}' = 1] - \Pr[\mathsf{Hyb}_0 = 1]\right| = \mathsf{negl}(\lambda).$$

**Claim 5.8.** *Assume that $i\mathcal{O}$ is polynomially secure. Then for any adversary $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}_{2^\ell} = 1] - \Pr[\mathsf{Real}_1 = 1]\right| = \mathsf{negl}(\lambda).$$

To move from $\mathsf{Hyb}_{\rho^*}$ to $\mathsf{Hyb}_{\rho^*+1}$, we use the following sub-hybrids:

- $\mathsf{Hyb}_{\rho^*,1}$: Same as $\mathsf{Hyb}_{\rho^*}$ except $K_{\mathsf{Enc}}$ is punctured at $\rho^*$, and $v^* = \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho^*)$ is hardcoded. That is:
  1. Samples $\mathsf{hk}' \leftarrow \mathsf{PH.GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt})$.
  2. Samples $K_{\mathsf{Enc}} \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^\ell, 1^1)$ and punctures $K'_{\mathsf{Enc}} = \mathsf{PPRF.Puncture}(K_{\mathsf{Enc}}, \rho^*)$.
  3. Sets $v^* = \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho^*)$ and samples $\mathsf{hk} \leftarrow i\mathcal{O}(\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, \rho^*, v^*])$ as defined in Fig. 9.
  4. Sends $\mathsf{hk}$ to $\mathcal{A}$.

**Constants:** A hash key $\mathsf{hk}'$, a punctured PPRF key $K_{\mathsf{Enc}}\{\rho^*\}$, a function splt, an integer $\rho^*$, a value $v^*$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$.

1. $\rho = \mathsf{PH.Hash}(\mathsf{hk}', x)$.
2. $\mathsf{dig} = \begin{cases} (\rho, \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho) \oplus \mathsf{splt}(x)) & \rho < \rho^* \\ (\rho^*, v^*) & \rho = \rho^* \\ (\rho, \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho) \oplus 0) & \rho > \rho^* \end{cases}$.
3. Outputs dig.

Figure 9: Program $\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}\{\rho^*\}, \mathsf{splt}, \rho^*, v^*]$.

- $\mathsf{Hyb}_{\rho^*,2}$: Same as $\mathsf{Hyb}_{\rho^*,1}$, but $v^*$ is sampled uniformly at random from $\{0,1\}$. That is:
  3. Samples $v^* \xleftarrow{R} \{0,1\}$ and $\mathsf{hk} \leftarrow i\mathcal{O}(\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, \rho^*, v^*])$ as defined in Fig. 9.
- $\mathsf{Hyb}_{\rho^*,3}$: Same as $\mathsf{Hyb}_{\rho^*,2}$, but if there exists $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $\mathsf{PH.Hash}(\mathsf{hk}', x^*) = \rho^*$ and $\mathsf{splt}(x^*) = 1$ then the hardcoded value is changed to $v^* \oplus 1$. Otherwise, $v^*$ remains the same. That is:
  3. Samples $v^* \xleftarrow{R} \{0,1\}$ and $\mathsf{hk} \leftarrow i\mathcal{O}(\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, \rho^*, v^* \oplus \beta])$ as defined in Fig. 9, where

$$\beta = \begin{cases} 1 & \exists x^* \in \{0,1\}^{\ell_{\mathsf{in}}} \text{ such that } \mathsf{PH.Hash}(\mathsf{hk}', x^*) = \rho^* \text{ and } \mathsf{splt}(x^*) = 1 \\ 0 & \text{otherwise} \end{cases}$$

- $\mathsf{Hyb}_{\rho^*,4}$: Same as $\mathsf{Hyb}_{\rho^*,3}$, but $v^*$ is changed back to pseudorandom. That is:
  3. Sets $v^* = \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho^*)$ and samples $\mathsf{hk} \leftarrow i\mathcal{O}(\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, \rho^*, v^* \oplus \beta])$ as defined in Fig. 9, where

$$\beta = \begin{cases} 1 & \exists x^* \in \{0,1\}^{\ell_{\mathsf{in}}} \text{ such that } \mathsf{PH.Hash}(\mathsf{hk}', x^*) = \rho^* \text{ and } \mathsf{splt}(x^*) = 1 \\ 0 & \text{otherwise} \end{cases}$$

We formally state the claims:

**Claim 5.9.** *Assume that $i\mathcal{O}$ is $2^{-\lambda-\ell}$-secure and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{\rho^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\rho^*,1}(\mathcal{A}) = 1]\right| \leq 2^{-\lambda}2^{-\ell}.$$

*Proof.* Functionality of the programs in both hybrids is identical by the punctured correctness of $\Pi_{\mathsf{PPRF}}$. The claim therefore follows by $2^{-\lambda^{\delta}_{i\mathcal{O}}}$-security of $i\mathcal{O}$. □

**Claim 5.10.** *Assume that $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda-\ell}$-punctured pseudorandomness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{\rho^*,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\rho^*,2}(\mathcal{A}) = 1]\right| \leq 2^{-\lambda}2^{-\ell}.$$

*Proof.* We construct a reduction $\mathcal{B}$ that breaks the punctured pseudorandomness of $\Pi_{\mathsf{PPRF}}$. The reduction $\mathcal{B}$ receives $(K\{\rho^*\}, v^*)$ from the PPRF challenger, and uses those values to simulate the mode indistinguishability experiment for $\mathcal{A}$. If $v^*$ is pseudorandom, it simulates $\mathsf{Hyb}_{\rho^*,1}$; if $v^*$ is uniform, it simulates $\mathsf{Hyb}_{\rho^*,2}$. □

**Claim 5.11.** *For any adversary $\mathcal{A}$, we have* $\left|\Pr[\mathsf{Hyb}_{\rho^*,2} = 1] - \Pr[\mathsf{Hyb}_{\rho^*,3} = 1]\right| = 0.$

*Proof.* This statistical claim follows by the fact that $v^*$ is uniformly distributed, and appears only once in the view of $\mathcal{A}$. Therefore no matter how the bit $\beta$ is computed, as long as it is independent of $v^*$, the distribution of $v^*$ is identical to that of $v^* \oplus \beta$. □

**Claim 5.12.** *Assume that $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda-\ell}$-punctured pseudorandomness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{\rho^*,3}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\rho^*,4}(\mathcal{A}) = 1]\right| \leq 2^{-\lambda}2^{-\ell}.$$

*Proof.* We construct a non-uniform reduction $\mathcal{B}$ that breaks the punctured pseudorandomness of $\Pi_{\mathsf{PPRF}}$:

1. In the offline phase, algorithm $\mathcal{B}$ runs the offline phase of $\mathcal{A}$, then samples the hash key as follows $\mathsf{hk}' \leftarrow \mathsf{PH.GenPart}(1^{\lambda}, 1^{\ell_{\mathsf{in}}}, 1^{s}, \mathsf{splt})$, and computes

$$\beta = \begin{cases} 1 & \exists x^* \in \{0,1\}^{\ell_{\mathsf{in}}} \text{ such that } \mathsf{PH.Hash}(\mathsf{hk}', x^*) = \rho^* \text{ and } \mathsf{splt}(x^*) = 1 \\ 0 & \text{otherwise} \end{cases}$$

2. In the online phase, algorithm $\mathcal{B}$ submits $\rho^*$ as the challenge and receives $K'_{\mathsf{Enc}}$ and a challenge $v^*$ from the PPRF challenger.

3. Algorithm $\mathcal{B}$ samples the obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(\mathsf{pHash}[\mathsf{hk}', K_{\mathsf{Enc}}, \mathsf{splt}, \rho^*, v^* \oplus \beta])$ as defined in Fig. 9, and sends hk to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

If $v^*$ is pseudorandom, $\mathcal{B}$ simulates $\mathsf{Hyb}_{\rho^*,4}$. If $v^*$ is uniform, it simulates $\mathsf{Hyb}_{\rho^*,3}$. The online phase of $\mathcal{B}$ is efficient if $\mathcal{A}$ is efficient. The claim follows by the punctured security of $\Pi_{\mathsf{PPRF}}$. □

**Claim 5.13.** *Assume that $i\mathcal{O}$ is $2^{-\lambda-\ell}$-secure, $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness and $\Pi_{\mathsf{PH}}$ satisfies perfect partitioning. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{\rho^*,4}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\rho^*+1}(\mathcal{A}) = 1]\right| \leq 2^{-\lambda}2^{-\ell}.$$

*Proof.* We show that the programs in both hybrids are functionally equivalent. Note that both programs compute $\rho = \mathsf{PH.Hash}(\mathsf{hk}', x)$. For any input $x$ such that $\rho \neq \rho^*$, both programs behave identically to $\mathsf{Hyb}_{\rho^*+1}$ by the definition of the hybrids and the punctured correctness of $\Pi_{\mathsf{PPRF}}$. For any input $x$ such that $\rho = \rho^*$, the program in $\mathsf{Hyb}_{\rho^*,4}$ outputs $(\rho^*, \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho^*) \oplus \beta)$. By definition of $\beta$, if there exists $x^*$ mapping to $\rho^*$ such that $\mathsf{splt}(x^*) = 1$, then $\beta = 1$. Due to the perfect partitioning of $\Pi_{\mathsf{PH}}$, if such $x^*$ exists, then all $x$ mapping to $\rho^*$ must satisfy $\mathsf{splt}(x) = 1$. In this case, $\beta = 1 = \mathsf{splt}(x)$ for all $x \in \mathsf{Hash}^{-1}(\rho^*)$. If no such $x^*$ exists, then $\beta = 0$. Again by perfect partitioning, if there are any $x$ mapping to $\rho^*$, they must all satisfy $\mathsf{splt}(x) = 0$. In this case, $\beta = 0 = \mathsf{splt}(x)$ for all $x \in \mathsf{Hash}^{-1}(\rho^*)$. Finally, if $\mathsf{Hash}^{-1}(\rho^*)$ is empty, the behavior on $\rho = \rho^*$ is unreachable. In all cases, for any $x$ mapping to $\rho^*$, we have $\beta = \mathsf{splt}(x)$. It follows that for $\rho = \rho^*$, the program outputs $(\rho^*, \mathsf{PPRF.Eval}(K_{\mathsf{Enc}}, \rho^*) \oplus \mathsf{splt}(x))$, which matches the behavior of $\mathsf{Hyb}_{\rho^*+1}$. Since the programs are functionally equivalent, the claim follows from the $2^{-\lambda^{\delta}_{i\mathcal{O}}}$-security of $i\mathcal{O}$. □

**Proof of Lemma 5.5.** By combining Claims 5.9 to 5.13 a total of $2^{\ell}$ times, we get that there exists $\Lambda$ such that for all $\lambda \geq \Lambda$:

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2^{\ell}}(\mathcal{A}) = 1]\right| \leq O(2^{-\lambda}).$$

By additionally considering Claims 5.6 to 5.8, we have

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

and the lemma follows. □

# 6 Constructing Perfectly Partitionable Hash

In this section, we present our construction of perfectly partitionable hash from iO and rerandomizable commitments.

**Construction 6.1.** The construction uses the following ingredients:

- Indistinguishability obfuscator $i\mathcal{O}$ defined in Definition 3.1, for all Boolean circuits of size at most $s' := s'(\lambda, \ell_{\mathsf{in}}, s)$, which we specify later.
- A puncturable PRF $\Pi_{\mathsf{PPRF}} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Puncture})$ defined in Definition 3.2.
- A rerandomizable commitment $\Pi_{\mathsf{Com}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Rerandomize})$ defined in Definition 3.3. Let $\ell_{\mathsf{out}} := \ell_{\mathsf{out}}(\lambda)$ be the commitment length and $\ell_r := \ell_r(\lambda)$ be the opening length for security parameter $\lambda$.

The algorithms are defined as follows:

- $\mathsf{Gen}(1^{\lambda}, 1^{\ell_{\mathsf{in}}}, 1^{s})$:
  1. Samples commitment public parameters $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^{\lambda})$.

2. Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_r})$.
3. The hash key is an obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[\mathsf{pp}, K])$ where pHash is described in Fig. 10

**Constants:** Commitment public parameters pp, a PPRF key $K$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$

1. Outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x)\big)$.

Figure 10: Program $\mathsf{pHash}[\mathsf{pp}, K]$.

- $\mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt})$:
  1. Samples commitment public parameters $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$.
  2. Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_r})$.
  3. Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 1; U_{\ell_r})$.
  4. The hash key is an obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, \tilde{c}])$ where pHash is described in Fig. 11

**Constants:** A Boolean function $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, commitment public parameters pp, a PPRF key $K$, a commitment $\tilde{c}$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$

1. If $\mathsf{splt}(x) = 0$, then outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x)\big)$.
2. If $\mathsf{splt}(x) = 1$, then outputs $\mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, \mathsf{PPRF.Eval}(K, x)\big)$.

Figure 11: Program $\mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, \tilde{c}]$.

- $\mathsf{Hash}(\mathsf{hk}, x)$: Interprets hk as a program and outputs $\mathsf{hk}(x)$.

**Theorem 6.2** (PPH from iO and rerandomizable commitments)**.** *Assume that $i\mathcal{O}$ is a sub-exponentially secure indistinguishability obfuscation, $\Pi_{\mathsf{PPRF}}$ is a sub-exponentially secure puncturable PRF, and $\Pi_{\mathsf{Com}}$ is a polynomially secure rerandomizable perfectly-binding commitment. Then Construction 6.1 is a perfectly partitionable hash.*

Theorem 6.2 will follow by Lemmas 6.3 to 6.5.

**Lemma 6.3** (Digest succinctness)**.** *There exists a polynomial $p(\cdot)$ such that for any $\lambda, \ell_{\mathsf{in}} \in \mathbb{N}$, the digest length of Construction 6.1 is always bounded by $p(\lambda)$. Specifically, it is independent of $\ell_{\mathsf{in}}$.*

*Proof.* This claim follows immediately by inspecting the construction: the digest is a commitment, and the commitment parameters are instantiated using the security parameter $\lambda$. Specifically, they are independent of $\ell_{\mathsf{in}}$. Since $\Pi_{\mathsf{Com}}$ consists of efficient algorithms, the commitment (and thus the digest) has length $\mathsf{poly}(\lambda)$ and the claim follows. □

**Lemma 6.4** (Perfect Partitioning). *If $\Pi_{\mathsf{Com}}$ is perfectly binding and satisfies rerandomization correctness, and $i\mathcal{O}$ satisfies functionality-preserving, then Construction 6.1 satisfies perfect partitioning.*

*Proof.* Fix any $\lambda, \ell_{\mathsf{in}}, s \in \mathbb{N}$ and any function $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$. Let $\mathsf{hk}$ be a hash key in the range of $\mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt})$, and let $x, x' \in \{0,1\}^{\ell_{\mathsf{in}}}$ such that $\mathsf{splt}(x) = 0$ and $\mathsf{splt}(x') = 1$. By construction and the functionality-preserving of $i\mathcal{O}$, we have that $\mathsf{Hash}(\mathsf{hk}, x) = \mathsf{hk}(x)$ is a commitment to the bit 0, whereas $\mathsf{Hash}(\mathsf{hk}, x') = \mathsf{hk}(x')$ is a rerandomization of some $\tilde{c}$, where $\tilde{c}$ is a commitment of 1. By rerandomization correctness, we have that $\mathsf{hk}(x')$ is also a commitment of 1. By perfect binding, it follows that $\mathsf{hk}(x) \neq \mathsf{hk}(x')$. □

## 6.1 Mode Indistinguishability of Construction 6.1

In this section, we prove the following lemma:

**Lemma 6.5** (Mode Indistinguishability). *Assume that $i\mathcal{O}$ is $2^{-\lambda_{i\mathcal{O}}^{\delta}}$-secure, $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda_{\mathsf{PPRF}}^{\delta}}$-punctured pseudorandomness, and $\Pi_{\mathsf{Com}}$ is (polynomially) hiding and perfectly rerandomizable. If we set $\lambda_{i\mathcal{O}}(\lambda, \ell_{\mathsf{in}}) = \lambda_{\mathsf{PPRF}}(\lambda, \ell_{\mathsf{in}}) = (\lambda + \ell_{\mathsf{in}})^{1/\delta}$ then Construction 6.1 satisfies polynomial mode indistinguishability. Moreover, for any $\varepsilon(\lambda) = \Omega(2^{-\lambda})$, if $\Pi_{\mathsf{Com}}$ is $\varepsilon$-hiding, then Construction 6.1 satisfies $O(\varepsilon)$-mode indistinguishability.*

Let $\mathcal{A}$ be an adversary for the mode indistinguishability game of $\Pi_{\mathsf{PH}}$. Let $n$ be a bound on the function output length that $\mathcal{A}$ chooses, and without loss of generality assume $\mathcal{A}$ chooses this $n$ (we can also pad the output of the function splt). We start by formally stating the mode indistinguishability experiments for Construction 6.1.

- $\mathsf{Real}_0$: In this game, the challenger samples a non-partitioning hash key.
  1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s \in \mathbb{N}$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}$ to the challenger.
  2. The challenger does the following:
     (a) Samples commitment public parameters $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$.
     (b) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_r})$.
     (c) Samples the obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[\mathsf{pp}, K])$ where pHash is described in Fig. 10

     The challenger sends $\mathsf{hk}$ to $\mathcal{A}$.
  3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.
- $\mathsf{Real}_1$: In this game, the challenger samples a partitioning hash key.
  1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s \in \mathbb{N}$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}$ to the challenger.
  2. The challenger does the following:
     (a) Samples commitment public parameters $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$.
     (b) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_r})$.
     (c) Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 1; U_{\ell_r})$.
     (d) Samples the obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, \tilde{c}])$ where pHash is described in Fig. 11

The challenger sends $\mathsf{hk}$ to $\mathcal{A}$.

3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

Next, we define the following sequence of hybrids for each $x^* \in [0, 2^{\ell_{\mathsf{in}}}]$:

- $\mathsf{Hyb}_{x^*}$: In this experiment, the challenger samples a transitional hash key that behaves like a normal key for all $x \geq x^*$, and "almost" like a partitioned key for all $x < x^*$. We say "almost" because the commitment $\tilde{c}$ at this point is a commitment of 0.

  1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s \in \mathbb{N}$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}$ to the challenger.
  2. The challenger does the following:
     (a) Samples commitment public parameters $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$.
     (b) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_r})$.
     (c) Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 0; U_{\ell_r})$.
     (d) Samples the obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, x^*, \tilde{c}])$ where $\mathsf{pHash}$ is described in Fig. 12.

     The challenger sends $\mathsf{hk}$ to $\mathcal{A}$.
  3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

**Constants:** A Boolean function $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, commitment public parameters $\mathsf{pp}$, a PPRF key $K$, an integer $x^*$, a commitment $\tilde{c}$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$

1. If $x \geq x^*$, outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x)\big)$.
2. If $x < x^*$, then
   (a) If $\mathsf{splt}(x) = 0$, then outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x)\big)$.
   (b) If $\mathsf{splt}(x) = 1$, then outputs $\mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, \mathsf{PPRF.Eval}(K, x)\big)$.

Figure 12: Program $\mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, x^*, \tilde{c}]$.

- $\mathsf{Hyb}_{x^*,1}$: Same as $\mathsf{Hyb}_{x^*}$, except that the PPRF key is punctured at $x^*$.

  1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s \in \mathbb{N}$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}$ to the challenger.
  2. The challenger does the following:
     (a) Samples commitment public parameters $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$.
     (b) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^{\ell_r})$.
     (c) Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 0; U_{\ell_r})$.
     (d) Punctures $K\{x^*\} = \mathsf{PPRF.Puncture}(K, x^*)$.
     (e) Computes $c^* = \mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x^*)\big)$.

(f) Samples the obfuscated program $\mathsf{hk} \leftarrow iO(1^{\lambda_{iO}}, \mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K\{x^*\}, x^*, \tilde{c}, c^*])$ where pHash is described in Fig. 13.

The challenger sends hk to $\mathcal{A}$.

3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

**Constants:** A Boolean function $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, commitment public parameters pp, a punctured PPRF key $K\{x^*\}$, an integer $x^*$, a commitment $\tilde{c}$, an additional commitment $c^*$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$

1. If $x > x^*$, outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K\{x^*\}, x)\big)$.
2. If $x = x^*$, outputs $c^*$.
3. If $x < x^*$, then
   (a) If $\mathsf{splt}(x) = 0$, then outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K\{x^*\}, x)\big)$.
   (b) If $\mathsf{splt}(x) = 1$, then outputs $\mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, \mathsf{PPRF.Eval}(K\{x^*\}, x)\big)$.

Figure 13: Program $\mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K\{x^*\}, x^*, \tilde{c}, c^*]$.

- $\mathsf{Hyb}_{x^*,2}$: Same as $\mathsf{Hyb}_{x^*,1}$, except that if $\mathsf{splt}(x^*) = 1$, then $c^*$ is sampled as a fresh commitment of 0.

  2. The challenger does the following changes to how it samples the hash key:

     (f) Samples $c^* = \begin{cases} \mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x^*)\big) & \mathsf{splt}(x^*) = 0 \\ \mathsf{C.Com}\big(\mathsf{pp}, 0, U_{\ell_r}\big) & \mathsf{splt}(x^*) = 1 \end{cases}$

- $\mathsf{Hyb}_{x^*,3}$: Same as $\mathsf{Hyb}_{x^*,2}$, except that if $\mathsf{splt}(x^*) = 1$, then $c^*$ is sampled as a rerandomization of $\tilde{c}$.

  2. The challenger does the following changes to how it samples the hash key:

     (f) Samples $c^* = \begin{cases} \mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x^*)\big) & \mathsf{splt}(x^*) = 0 \\ \mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, U_{\ell_r}\big) & \mathsf{splt}(x^*) = 1 \end{cases}$

- $\mathsf{Hyb}_{x^*,4}$: Same as $\mathsf{Hyb}_{x^*,3}$, except that if $\mathsf{splt}(x^*) = 1$, then $c^*$ is sampled as a pseudorandom rerandomization of $\tilde{c}$, where the coins are derived using $K'$.

  2. The challenger does the following changes to how it samples the hash key:

     (f) Sets $c^* = \begin{cases} \mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x^*)\big) & \mathsf{splt}(x^*) = 0 \\ \mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, \mathsf{PPRF.Eval}(K, x^*)\big) & \mathsf{splt}(x^*) = 1 \end{cases}$

- $\mathsf{Hyb}_{x^*,5}$: Same as $\mathsf{Hyb}_{x^*,4}$, except that the PPRF key is no longer punctured. This is identical to $\mathsf{Hyb}_{x^*+1}$ (for $x^* < 2^{\ell_{\mathsf{in}}}$).

Finally, we define the following experiment:

- $\mathsf{Hyb}'_{2^{\ell_{\mathsf{in}}}}$: Same as $\mathsf{Hyb}_{2^{\ell_{\mathsf{in}}},5}$, except that $\tilde{c}$ is sampled as a commitment of 1 instead of 0.

  2. The challenger does the following changes to how it samples the hash key:

(c) Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 1; U_{\ell_r})$.

We start by proving the two following claims about the first and last hybrids in the sequence:

**Claim 6.6.** *Assume that $i\mathcal{O}$ is polynomially secure. Then for any bounded adversary $\mathcal{A}$ we have*

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

*Proof.* By construction, for any function splt, any commitment public parameters pp, any PPRF key $K$ and any commitment $\tilde{c}$, the two following programs are functionally equivalent:

- $\mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, 0, \tilde{c}]$ (with $x^* = 0$) from Fig. 12.
- $\mathsf{pHash}[\mathsf{pp}, K]$ from Fig. 10.

Let $\mathcal{A}$ be an efficient adversary and denote

$$\varepsilon := \left|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]\right|.$$

We construct an efficient adversary $\mathcal{B}$ for the iO security game as follows:

1. $\mathcal{B}$ runs $\mathcal{A}$ and gets $1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}$.
2. $\mathcal{B}$ generates the two programs $C_0 = \mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, 0, \tilde{c}]$ and $C_1 = \mathsf{pHash}[\mathsf{pp}, K]$, and sends them as the iO challenge.
3. $\mathcal{B}$ maintains the state of $\mathcal{A}$ as its state st.
4. $\mathcal{B}$ receives an obfuscated program $\tilde{C}$ from the iO challenger and sends $\mathsf{hk} = \tilde{C}$ to $\mathcal{A}$ as the mode indistinguishability challenge key.
5. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. By construction, we have that

$$\varepsilon = \left|\Pr[\mathcal{B}(\mathsf{st}, i\mathcal{O}(1^\lambda, C_0)) = 1] - \Pr[\mathcal{B}(\mathsf{st}, i\mathcal{O}(1^\lambda, C_1)) = 1]\right|.$$

If $i\mathcal{O}$ is $2^{-\lambda^\delta_{iO}}$-secure, then there exists $\Lambda$ such that for all $\lambda \geq \Lambda$ we have $\varepsilon \leq 2^{-\lambda^\delta_{iO}} = 2^{-\lambda - \ell_{\mathsf{in}}}$ and the claim follows. □

**Claim 6.7.** *Assume that $i\mathcal{O}$ is polynomially secure. Then for any bounded adversary $\mathcal{A}$ we have*

$$\left|\Pr[\mathsf{Hyb}'_{2^{\ell_{\mathsf{in}}}}(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

*Proof.* This claim is completely analogous to Claim 6.6. □

Next, we prove the following claims about the inner hybrids for all $x^* \in [2^{\ell_{\mathsf{in}}}]$:

**Claim 6.8.** *Assume that $i\mathcal{O}$ is $2^{-\lambda - \ell_{\mathsf{in}}}$-secure and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{x^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{x^*,1}(\mathcal{A}) = 1]\right| \leq 2^{-\lambda} 2^{-\ell_{\mathsf{in}}}.$$

*Proof.* If $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness, then for any $x^*$, any function splt, any commitment public parameters pp, any PPRF key $K$ and any commitment $\tilde{c}$, and $c^* = \mathsf{C.Com}(\mathsf{pp}, 0; \mathsf{PPRF.Eval}(K, x^*))$ the two following programs are functionally equivalent:

- $\mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, x^*, \tilde{c}, c^*]$ from Fig. 13.
- $\mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K, x^*, \tilde{c}]$ from Fig. 12.

Similar to Claim 6.6, we can use any efficient adversary $\mathcal{A}$ with $\varepsilon := \left|\Pr[\mathsf{Hyb}_{x^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{x^*,1}(\mathcal{A}) = 1]\right|$ to break the security of $i\mathcal{O}$ with the same advantage $\varepsilon$. Therefore by the $2^{-\lambda^{\delta}_{i\mathcal{O}}}$-security of $i\mathcal{O}$, there exists $\Lambda$ such that for all $\lambda \geq \Lambda$ we have $\varepsilon \leq 2^{-\lambda-\ell_{\mathsf{in}}}$ and the claim follows. □

**Claim 6.9.** *Assume that $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda-\ell_{\mathsf{in}}}$-punctured pseudorandomness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{x^*,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A}) = 1]\right| \leq 2^{-\lambda}2^{-\ell_{\mathsf{in}}}.$$

*Proof.* Let $\mathcal{A}$ be an efficient adversary and denote $\varepsilon := \left|\Pr[\mathsf{Hyb}_{x^*,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A}) = 1]\right|$. We construct an adversary $\mathcal{B}$ for the punctured pseudorandomness game of $\Pi_{\mathsf{PPRF}}$ as follows:

1. $\mathcal{B}$ runs $\mathcal{A}$ and gets $1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}$.
2. If $\mathsf{splt}(x^*) = 0$, then $\mathcal{B}$ outputs a random $b \xleftarrow{\mathrm{R}} \{0,1\}$ and stops the experiment. Otherwise, it continues.
3. $\mathcal{B}$ sends the input length parameter $1^{\ell_{\mathsf{in}}}$, the output length parameter $1^{\ell_r}$ and the string $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$ to the PPRF challenger.
4. The challenger responds with the punctured key $K\{x^*\}$ and a string $y$.
5. $\mathcal{B}$ samples the commitment public parameters pp and a commitment $\tilde{c}$ of 0, and computes $c^* = \mathsf{C.Com}(\mathsf{pp}, 0; y)$.
6. $\mathcal{B}$ obfuscates $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[\mathsf{splt}, \mathsf{pp}, K\{x^*\}, x^*, \tilde{c}, c^*])$ where pHash is described in Fig. 13 and sends hk to $\mathcal{A}$.
7. $\mathcal{B}$ forwards whatever $\mathcal{A}$ outputs.

First, we observe that if $\mathsf{splt}(x^*) = 0$, then $\mathsf{Hyb}_{x^*,1}$ and $\mathsf{Hyb}_{x^*,2}$ are identical. This means that conditioned on this event, we have $\Pr[\mathsf{Hyb}_{x^*,1}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A}) = 1]$, therefore $\mathcal{B}$ has the same advantage as $\mathcal{A}$ in this case. Second, if $\mathsf{splt}(x^*) = 1$, then

- $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{x^*,1}(\mathcal{A})$ when the PPRF challenger sends $y = \mathsf{PPRF.Eval}(K, x^*)$.
- $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{x^*,2}(\mathcal{A})$ when the PPRF challenger sends $y \xleftarrow{\mathrm{R}} \{0,1\}^{\ell_r}$.

Therefore we have that $\mathcal{B}$ distinguishes a pseudorandom $y$ from a truly random $y$ with advantage exactly $\varepsilon$. By the $2^{-\lambda^{\delta}_{\mathsf{PPRF}}}$-punctured pseudorandomness of $\Pi_{\mathsf{PPRF}}$, there exists $\Lambda$ such that for all $\lambda \geq \Lambda$ we have $\varepsilon \leq 2^{-\lambda^{\delta}_{\mathsf{PPRF}}} = 2^{-\lambda-\ell_{\mathsf{in}}}$ and the claim follows. □

**Claim 6.10.** *Assume that $\Pi_{\mathsf{Com}}$ satisfies perfect rerandomization. Then for any (unbounded) adversary $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{x^*,3}(\mathcal{A}) = 1]\right| = 0.$$

*Proof.* The only difference between $\mathsf{Hyb}_{x^*,2}$ and $\mathsf{Hyb}_{x^*,3}$ is in how $c^*$ is computed if $\mathsf{splt}(x^*) = 1$:

- In $\mathsf{Hyb}_{x^*,2}$, we have $c^* \leftarrow \mathsf{C.Com}(\mathsf{pp}, 0; U_{\ell_r})$.
- In $\mathsf{Hyb}_{x^*,3}$, we have $c^* \leftarrow \mathsf{C.Rerandomize}(\mathsf{pp}, \tilde{c}; U_{\ell_r})$, where $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 0; U_{\ell_r})$.

By the perfect rerandomization of $\Pi_{\mathsf{Com}}$, the two distributions are identical, therefore for any unbounded $\mathcal{A}$ we have $\Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_{x^*,3}(\mathcal{A}) = 1]$ and the claim follows. □

**Claim 6.11.** *Assume that $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda-\ell_{\mathsf{in}}}$-punctured pseudorandomness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

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

*Proof.* This claim is completely analogous to Claim 6.9. □

**Claim 6.12.** *Assume that $i\mathcal{O}$ is $2^{-\lambda-\ell_{\mathsf{in}}}$-secure and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{x^*,4}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{x^*,5}(\mathcal{A}) = 1]\right| \leq 2^{-\lambda}2^{-\ell_{\mathsf{in}}}.$$

*Proof.* This claim is completely analogous to Claim 6.8. □

**Claim 6.13.** *Assume that $\Pi_{\mathsf{Com}}$ satisfies $\varepsilon(\lambda)$-computational hiding. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{2^{\ell_{\mathsf{in}}}}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}'_{2^{\ell_{\mathsf{in}}}}(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda).$$

*Proof.* The only difference between $\mathsf{Hyb}_{2^{\ell_{\mathsf{in}}}}$ and $\mathsf{Hyb}'_{2^{\ell_{\mathsf{in}}}}$ is in how $\tilde{c}$ is computed:

- In $\mathsf{Hyb}_{2^{\ell_{\mathsf{in}}}}$, we have $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 0; U_{\ell_r})$.
- In $\mathsf{Hyb}'_{2^{\ell_{\mathsf{in}}}}$, we have $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 1; U_{\ell_r})$.

Any efficient adversary $\mathcal{A}$ that distinguishes the two hybrids can be used to distinguish the two commitments with the same advantage. If $\Pi_{\mathsf{Com}}$ is $\varepsilon(\lambda)$-computational hiding, then there exists $\Lambda$ such that for $\lambda \geq \Lambda$ we have

$$\left|\Pr[\mathsf{Hyb}_{2^{\ell_{\mathsf{in}}}}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}'_{2^{\ell_{\mathsf{in}}}}(\mathcal{A}) = 1]\right| \leq \varepsilon$$

and the claim follows. □

**Proof of Lemma 6.5.** We set $s'(\lambda, \ell_{\mathsf{in}}, s)$ to be the size of the largest program among Figs. 10 to 13, which allows us to invoke $i\mathcal{O}$ security when obfuscating those programs. By combining Claims 6.8 to 6.12 a total of $2^{\ell_{\mathsf{in}}}$ times, we get that there exists $\Lambda$ such that for all $\lambda \geq \Lambda$:

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2^{\ell_{\mathsf{in}}}}(\mathcal{A}) = 1]\right| \leq O(2^{-\lambda}).$$

Now in addition, if $\Pi_{\mathsf{Com}}$ is $\varepsilon(\lambda)$-hiding for some $\varepsilon(\lambda) = O(2^{-\lambda})$ then by applying Claim 6.13 there exists $\Lambda'$ such that for all $\lambda \geq \Lambda'$:

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}'_{2^{\ell_{\mathsf{in}}}}(\mathcal{A}) = 1]\right| \leq O(\varepsilon(\lambda)).$$

By considering Claims 6.6 and 6.7 as well, we get

$$|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]| \leq O(\varepsilon(\lambda))$$

and the lemma follows.

# 7 Constructing Computationally Partitionable Hash

In this section, we present our construction of computationally partitionable hash from iO and rerandomizable OWFs.

**Construction 7.1.** The construction uses the following ingredients:

- Indistinguishability obfuscator $i\mathcal{O}$ defined in Definition 3.1, for all Boolean circuits of size at most $s' := s'(\lambda, \ell_{\mathsf{in}}, s)$, which we specify later.
- A puncturable PRF $\Pi_{\mathsf{PPRF}} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Puncture})$ defined in Definition 3.2.
- A rerandomizable OWF $\Pi_{\mathsf{ROWF}} = (\mathsf{Setup}, \mathsf{Rerandomize}, \mathsf{RecoverSolution})$ defined in Definition 3.4.

The algorithms are defined as follows:

- $\mathsf{Gen}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s)$:
  1. Samples rerandomizable OWF $f \leftarrow \mathsf{ROWF.Setup}(1^\lambda)$. Let $n$ be the input length of $f$.
  2. Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^n)$.
  3. The hash key is an obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[f, K])$ where pHash is described in Fig. 14

**Constants:** A rerandomizable OWF $f$, a PPRF key $K$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$

1. Computes $y = \mathsf{PPRF.Eval}(K, x)$.
2. Outputs $f(y)$.

Figure 14: Program $\mathsf{pHash}[f, K]$.

- $\mathsf{GenPart}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt})$:
  1. Samples rerandomizable OWF $f \leftarrow \mathsf{ROWF.Setup}(1^\lambda)$. Let $n$ be the input length of $f$, and also the number of random bits needed to rerandomize a challenge.
  2. Samples a challenge $\tilde{y} \xleftarrow{R} \{0,1\}^n$, $\tilde{z} = f(\tilde{y})$.
  3. Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^n)$.
  4. The hash key is an obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[f, K, \tilde{z}, \mathsf{splt}])$ where pHash is described in Fig. 15

**Constants:** A rerandomizable OWF $f$, a PPRF key $K$, a challenge $\tilde{z}$, and a predicate splt.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$

1. If $\mathsf{splt}(x) = 0$ then outputs $f(\mathsf{PPRF.Eval}(K, x))$.
2. If $\mathsf{splt}(x) = 1$ then
   (a) Rerandomizes $(z, \mathsf{st}) \leftarrow \mathsf{ROWF.Rerandomize}(f, \tilde{z}; \mathsf{PPRF.Eval}(K, x))$.
   (b) Outputs $z$.

Figure 15: Program $\mathsf{pHash}[f, K, \tilde{z}, \mathsf{splt}]$.

- $\mathsf{Hash}(\mathsf{hk}, x)$: Interprets hk as a program and outputs $\mathsf{hk}(x)$.

**Theorem 7.2.** *Assume that $i\mathcal{O}$ is a sub-exponentially secure indistinguishability obfuscation, $\Pi_{\mathsf{PPRF}}$ is a sub-exponentially secure puncturable PRF, and $\Pi_{\mathsf{ROWF}}$ is a polynomially secure rerandomizable one-way function. Then Construction 7.1 is a computationally partitionable hash.*

Theorem 7.2 will follow by Lemmas 7.3 to 7.5.

**Lemma 7.3** (Digest succinctness)**.** *There exists a polynomial $p(\cdot)$ such that for any $\lambda, \ell_{\mathsf{in}} \in \mathbb{N}$, the digest length of Construction 7.1 is always bounded by $p(\lambda)$. Specifically, it is independent of $\ell_{\mathsf{in}}$.*

*Proof.* This claim follows immediately by inspecting the construction: the digest is a OWF challenge, and the OWF is instantiated using the security parameter $\lambda$. Specifically, they are independent of $\ell_{\mathsf{in}}$. Since $\Pi_{\mathsf{ROWF}}$ consists of efficient algorithms, the OWF challenge (and thus the digest) has length $\mathrm{poly}(\lambda)$ and the claim follows. □

**Lemma 7.4** (Computational Partitioning)**.** *If $\Pi_{\mathsf{ROWF}}$ is polynomially one-way and satisfies rerandomization correctness, and $i\mathcal{O}$ is functionality-preserving then $\Pi_{\mathsf{PH}}$ satisfies computational partitioning.*

*Proof.* Let $\mathcal{A}$ be a stateful adversary for the partitioning game. We construct an algorithm $\mathcal{B}$ that inverts the OWF function $\Pi_{\mathsf{ROWF}}$ as follows:

1. $\mathcal{B}$ gets as input a OWF $f$ and a challenge $\tilde{z}$. Let $n$ be the input length of $f$.
2. $\mathcal{B}$ computes $(1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}) \leftarrow \mathcal{A}(1^\lambda)$ where $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ is of size at most $s$.
3. $\mathcal{B}$ samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{2^{-\lambda^{\delta}_{\mathsf{PPRF}}}}, 1^{\ell_{\mathsf{in}}}, 1^n)$.
4. $\mathcal{B}$ samples an obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[f, K, \tilde{z}, \mathsf{splt}])$ where pHash is described in Fig. 15.
5. $\mathcal{B}$ computes $(x, x') \leftarrow \mathcal{A}(\mathsf{hk})$.
6. If $\mathsf{splt}(x) = \mathsf{splt}(x')$ or $\mathsf{hk}(x) \neq \mathsf{hk}(x')$, then $\mathcal{B}$ outputs $\perp$.
7. Otherwise, assume without loss of generality that $\mathsf{splt}(x) = 0$ and $\mathsf{splt}(x') = 1$. Then $\mathcal{B}$ computes its output as follows:

- $y = \mathsf{PPRF.Eval}(K, x)$.
- $(z, \mathsf{st}) = \mathsf{ROWF.Rerandomize}(f, \tilde{z}; \mathsf{PPRF.Eval}(K, x'))$.
- Outputs $\tilde{y} = \mathsf{ROWF.RecoverSolution}(f, y, \mathsf{st})$.

Fix any $\lambda \in \mathbb{N}$, any OWF $f$ and any OWF challenge $\tilde{z}$. We prove that with probability 1 if $\mathsf{splt}(x) = 0, \mathsf{splt}(x') = 1$ and $\mathsf{hk}(x) = \mathsf{hk}(x')$, then $f(\tilde{y}) = \tilde{z}$. By the functionality-preserving of $i\mathcal{O}$, we have that:

- $\mathsf{hk}(x) = f(y)$ where $y = \mathsf{PPRF.Eval}(K, x)$.
- $\mathsf{hk}(x') = z$ where $(z, \mathsf{st}) = \mathsf{ROWF.Rerandomize}(f, \tilde{z}; \mathsf{PPRF.Eval}(K, x'))$.

Since $\mathsf{hk}(x) = \mathsf{hk}(x')$, we have $f(y) = z$. By the rerandomization correctness of $\Pi_{\mathsf{ROWF}}$, we have $f(\tilde{y}) = \tilde{z}$ where $\tilde{y} = \mathsf{ROWF.RecoverSolution}(f, y, \mathsf{st})$. Putting it all together, if $\mathcal{A}$ wins the partitioning game with probability $\varepsilon$, then $\mathcal{B}$ inverts the OWF with the same probability $\varepsilon$. Moreover, if $\mathcal{A}$ is efficient then so is $\mathcal{B}$. Therefore, the claim follows. □

## 7.1 Mode Indistinguishability of Construction 7.1

We now move on to proving mode indistinguishability. This is very similar to the proof of Lemma 6.5 and follows an almost identical structure.

**Lemma 7.5** (Mode Indistinguishability). *Assume that $i\mathcal{O}$ is $2^{-\lambda_{i\mathcal{O}}^{\delta}}$-secure, $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda_{\mathsf{PPRF}}^{\delta}}$-punctured pseudorandomness, and $\Pi_{\mathsf{ROWF}}$ is (polynomially) secure and perfectly rerandomizable. If we set $\lambda_{i\mathcal{O}}(\lambda, \ell_{\mathsf{in}}) = \lambda_{\mathsf{PPRF}}(\lambda, \ell_{\mathsf{in}}) = (\lambda + \ell_{\mathsf{in}})^{1/\delta}$ then $\Pi_{\mathsf{PH}}$ satisfies mode indistinguishability.*

Let $\mathcal{A}$ be an adversary for the mode indistinguishability game of $\Pi_{\mathsf{PH}}$. We start by formally stating the mode indistinguishability experiments for Construction 7.1.

- $\mathsf{Real}_0$: In this game, the challenger samples a non-partitioning hash key.
  1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s \in \mathbb{N}$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, 1^{s}, \mathsf{splt}$ to the challenger.
  2. The challenger does the following:
     (a) Samples rerandomizable OWF $f \leftarrow \mathsf{ROWF.Setup}(1^{\lambda})$. Let $n$ be the input length of $f$.
     (b) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^{n})$.
     (c) Samples the obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[f, K])$ where pHash is described in Fig. 14.

     The challenger sends hk to $\mathcal{A}$.
  3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.
- $\mathsf{Real}_1$: In this game, the challenger samples a partitioning hash key.
  1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s \in \mathbb{N}$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, 1^{s}, \mathsf{splt}$ to the challenger.
  2. The challenger does the following:
     (a) Samples rerandomizable OWF $f \leftarrow \mathsf{ROWF.Setup}(1^{\lambda})$. Let $n$ be the input length of $f$, and also the number of random bits needed to rerandomize a challenge.

(b) Samples a challenge $\tilde{y} \xleftarrow{R} \{0,1\}^n$, $\tilde{z} = f(\tilde{y})$.

(c) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^n)$.

(d) Samples the obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[f, K, \tilde{z}, \mathsf{splt}])$ where pHash is described in Fig. 15

The challenger sends hk to $\mathcal{A}$.

3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

Next, we define the following sequence of hybrids for each $x^* \in [0, 2^{\ell_{\mathsf{in}}}]$:

- $\mathsf{Hyb}_{x^*}$: In this experiment, the challenger samples a transitional hash key that behaves like a normal key for all $x \geq x^*$, and like a partitioned key for all $x < x^*$.

1. $\mathcal{A}$ chooses $\ell_{\mathsf{in}}, s \in \mathbb{N}$ and a circuit $\mathsf{splt} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ of size at most $s$, and sends $1^{\ell_{\mathsf{in}}}, 1^s, \mathsf{splt}$ to the challenger.
2. The challenger does the following:
   (a) Samples rerandomizable OWF $f \leftarrow \mathsf{ROWF.Setup}(1^\lambda)$. Let $n$ be the input length of $f$, and also the number of random bits needed to rerandomize a challenge.
   (b) Samples a challenge $\tilde{y} \xleftarrow{R} \{0,1\}^n$, $\tilde{z} = f(\tilde{y})$.
   (c) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^{\ell_{\mathsf{in}}}, 1^n)$.
   (d) Samples the obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[f, K, \tilde{z}, \mathsf{splt}])$ where pHash is described in Fig. 16

   The challenger sends hk to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

**Constants:** A rerandomizable OWF $f$, a PPRF key $K$, a challenge $\tilde{z}$, a predicate splt, an integer $x^*$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$

1. If $\mathsf{splt}(x) = 0$ or $x \geq x^*$ then outputs $f(\mathsf{PPRF.Eval}(K, x))$.
2. If $\mathsf{splt}(x) = 1$ and $x < x^*$ then
   (a) Rerandomizes $(z, \mathsf{st}) \leftarrow \mathsf{ROWF.Rerandomize}(f, \tilde{z}; \mathsf{PPRF.Eval}(K, x))$.
   (b) Outputs $z$.

Figure 16: Program $\mathsf{pHash}[f, K, \tilde{z}, \mathsf{splt}, x^*]$.

- $\mathsf{Hyb}_{x^*,1}$: Same as $\mathsf{Hyb}_{x^*}$, except that the PPRF key is punctured at $x^*$.

2. The challenger does the following changes to how it samples the hash key:
   (d) Punctures $K\{x^*\} = \mathsf{PPRF.Puncture}(K, x^*)$.
   (e) Computes $z^* = f(\mathsf{PPRF.Eval}(K, x^*))$.
   (f) Samples the obfuscated program $\mathsf{hk} \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{pHash}[f, K, \tilde{z}, \mathsf{splt}, z^*])$ where pHash is described in Fig. 17.

   The challenger sends hk to $\mathcal{A}$.

3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

**Constants:** A rerandomizable OWF $f$, a PPRF key $K$, a challenge $\tilde{z}$, a predicate splt, an integer $x^*$, an additional challenge $z^*$.
**Inputs:** A string $x \in \{0,1\}^{\ell_{\text{in}}}$.

1. If $\text{splt}(x) = 0$ or $x > x^*$ then outputs $f(\text{PPRF.Eval}(K, x))$.
2. If $x = x^*$ then outputs $z^*$.
3. If $\text{splt}(x) = 1$ and $x < x^*$ then
   (a) Rerandomizes $(z, \text{st}) \leftarrow \text{ROWF.Rerandomize}(f, \tilde{z}; \text{PPRF.Eval}(K, x))$.
   (b) Outputs $z$.

Figure 17: Program $\text{pHash}[f, K, \tilde{z}, \text{splt}, x^*, z^*]$.

- $\text{Hyb}_{x^*,2}$: Same as $\text{Hyb}_{x^*,1}$, except that if $\text{splt}(x^*) = 1$, then $z^*$ is sampled as a fresh challenge.

  2. The challenger does the following changes to how it samples the hash key:
     (e) Samples $z^* = \begin{cases} f(\text{PPRF.Eval}(K, x^*)) & \text{splt}(x^*) = 0 \\ f(U_n) & \text{splt}(x^*) = 1 \end{cases}$

- $\text{Hyb}_{x^*,3}$: Same as $\text{Hyb}_{x^*,2}$, except that if $\text{splt}(x^*) = 1$, then $z^*$ is sampled as a rerandomization of $\tilde{z}$.

  2. The challenger does the following changes to how it samples the hash key:
     (e) Samples $(z, \text{st}) \leftarrow \text{ROWF.Rerandomize}(f, \tilde{z}; U_n)$ and sets

$$z^* = \begin{cases} f(\text{PPRF.Eval}(K, x^*)) & \text{splt}(x^*) = 0 \\ z & \text{splt}(x^*) = 1 \end{cases}$$

- $\text{Hyb}_{x^*,4}$: Same as $\text{Hyb}_{x^*,3}$, except that if $\text{splt}(x^*) = 1$, then $z^*$ is computed as a pseudorandom rerandomization of $\tilde{z}$, where the coins are derived from $x^*$.

  2. The challenger does the following changes to how it samples the hash key:
     (e) Computes $(z, \text{st}) \leftarrow \text{ROWF.Rerandomize}(f, \tilde{z}; \text{PPRF.Eval}(K, x^*))$ and sets

$$z^* = \begin{cases} f(\text{PPRF.Eval}(K, x^*)) & \text{splt}(x^*) = 0 \\ z & \text{splt}(x^*) = 1 \end{cases}$$

- $\text{Hyb}_{x^*,5}$: Same as $\text{Hyb}_{x^*,4}$, except that the PPRF key is no longer punctured. This is identical to $\text{Hyb}_{x^*+1}$ (for $x^* < 2^{\ell_{\text{in}}}$).

We start by proving the two following claims about the first and last hybrids in the sequence:

**Claim 7.6.** *Assume that $i\mathcal{O}$ is polynomially secure. Then for any bounded adversary $\mathcal{A}$ we have*

$$\left|\Pr[\text{Real}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_0(\mathcal{A}) = 1]\right| = \text{negl}(\lambda).$$

*Proof.* This claim is completely analogous to Claim 6.6. □

**Claim 7.7.** *Assume that $i\mathcal{O}$ is polynomially secure. Then for any bounded adversary $\mathcal{A}$ we have*

$$\left|\Pr[\mathsf{Hyb}_{2^{\ell_{\mathsf{in}}},5}(\mathcal{A})=1]-\Pr[\mathsf{Real}_1(\mathcal{A})=1]\right|=\mathsf{negl}(\lambda).$$

*Proof.* This claim is completely analogous to Claim 6.7. □

**Claim 7.8.** *Assume that $i\mathcal{O}$ is $2^{-\lambda-\ell_{\mathsf{in}}}$-secure and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda\in\mathbb{N}$ such that for all $\lambda\geq\Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{x^*}(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_{x^*,1}(\mathcal{A})=1]\right|\leq 2^{-\lambda}2^{-\ell_{\mathsf{in}}}.$$

*Proof.* This claim is completely analogous to Claim 6.8. □

**Claim 7.9.** *Assume that $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda-\ell_{\mathsf{in}}}$-punctured pseudorandomness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda\in\mathbb{N}$ such that for all $\lambda\geq\Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{x^*,1}(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A})=1]\right|\leq 2^{-\lambda}2^{-\ell_{\mathsf{in}}}.$$

*Proof.* This claim is completely analogous to Claim 6.9. □

**Claim 7.10.** *If $\Pi_{\mathsf{ROWF}}$ satisfies perfect rerandomization then for any (unbounded) adversary $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_{x^*,3}(\mathcal{A})=1]\right|=0.$$

*Proof.* The only difference between $\mathsf{Hyb}_{x^*,2}$ and $\mathsf{Hyb}_{x^*,3}$ is in how $z^*$ is computed if $\mathsf{splt}(x^*)=1$:

- In $\mathsf{Hyb}_{x^*,2}$, we have $z^*\leftarrow f(U_n)$.
- In $\mathsf{Hyb}_{x^*,3}$, we have $z^*=z$, where $(z,\mathsf{st})\leftarrow\mathsf{ROWF.Rerandomize}(f,\tilde{z};U_n)$.

By the perfect rerandomization of $\Pi_{\mathsf{ROWF}}$, the two distributions are identical, therefore for any unbounded $\mathcal{A}$ we have $\Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A})=1]=\Pr[\mathsf{Hyb}_{x^*,3}(\mathcal{A})=1]$ and the claim follows. □

**Claim 7.11.** *Assume that $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda-\ell_{\mathsf{in}}}$-punctured pseudorandomness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda\in\mathbb{N}$ such that for all $\lambda\geq\Lambda$*

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

*Proof.* This claim is completely analogous to Claim 6.11. □

**Claim 7.12.** *Assume that $i\mathcal{O}$ is $2^{-\lambda-\ell_{\mathsf{in}}}$-secure and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda\in\mathbb{N}$ such that for all $\lambda\geq\Lambda$*

$$\left|\Pr[\mathsf{Hyb}_{x^*,4}(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_{x^*,5}(\mathcal{A})=1]\right|\leq 2^{-\lambda}2^{-\ell_{\mathsf{in}}}.$$

*Proof.* This claim is completely analogous to Claim 6.12. □

**Proof of Lemma 7.5.** By combining Claims 7.8 to 7.12 a total of $2^{\ell_{\text{in}}}$ times, we get that there exists $\Lambda$ such that for all $\lambda \geq \Lambda$:

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2^{\ell_{\text{in}}}}(\mathcal{A}) = 1]\right| \leq O(2^{-\lambda}).$$

By considering Claims 7.6 and 7.7 as well, we get

$$|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]| \leq O(2^{-\lambda}) = \mathsf{negl}(\lambda)$$

and the lemma follows.

# 8 Sometimes-Statistically Sound SNARGs

In this section, we define and construct sometimes-statistically sound SNARGs (SSS-SNARGs).

**Definition 8.1** (Sometimes-Statistically Sound SNARG)**.** A sometimes statistically sound SNARG for a relation $\mathcal{R}$ consists of four PPT algorithms $(\mathsf{Setup}, \mathsf{BindingSetup}, \mathsf{Prove}, \mathsf{Verify})$:

- $\mathsf{Setup}(1^\lambda, 1^n, 1^s) \to \mathsf{crs}$: On input a security parameter $1^\lambda$, an instance size $1^n$, and a predicate size bound $1^s$, outputs a common reference string crs.
- $\mathsf{BindingSetup}(1^\lambda, 1^n, 1^s, \mathsf{pred}) \to \mathsf{crs}$: On input a security parameter $1^\lambda$, an instance size $1^n$, a predicate size bound $1^s$, and a predicate $\mathsf{pred} : \{0,1\}^n \to \{0,1\}$, outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: On input a common reference string crs, an instance $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^n$, outputs a proof $\pi$.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: On input a common reference string crs, an instance $x \in \{0,1\}^n$, and a proof $\pi$, outputs a bit $b \in \{0,1\}$.

The scheme should satisfy the following properties:

- **Perfect Completeness**: For all $\lambda, n, s \in \mathbb{N}$ and all $x \in \{0,1\}^n$ and $w \in \{0,1\}^n$ such that $(x, w) \in \mathcal{R}$, the following holds

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

- **Succinctness**: The size of the proof $\pi$ is $\mathsf{poly}(\lambda)$, independent of the size of the witness or statement.
- **Statistical Soundness in Binding Mode**: For all security parameters $\lambda \in \mathbb{N}$, we define the following security game:
  1. The adversary $\mathcal{A}$ sends the parameters $1^n, 1^s$ and a predicate $\mathsf{pred} : \{0,1\}^n \to \{0,1\}$ of size at most $s$ to the challenger.
  2. The challenger sends $\mathsf{crs} \leftarrow \mathsf{BindingSetup}(1^\lambda, 1^n, 1^s, \mathsf{pred})$ to the adversary.
  3. The adversary sends $x \in \{0,1\}^n$ and a proof $\pi$.
  4. The output of the experiment is 1 if and only if $\mathsf{pred}(x) = 1$ and $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$.

We say that the SSS-SNARG satisfies statistical soundness in binding mode if for any unbounded adversary, the probability of the experiment outputting 1 is $\mathsf{negl}(\lambda)$.

- **Mode Indistinguishability**: For all security parameters $\lambda \in \mathbb{N}$, we define the following security game parameterized by a bit $b \in \{0,1\}$:

  1. The adversary $\mathcal{A}$ sends the parameters $1^n, 1^s$ and a predicate $\mathsf{pred} : \{0,1\}^n \to \{0,1\}$ of size at most $s$ to the challenger.
  2. If there exists an instance $x \in \{0,1\}^n$ and a witness $w \in \{0,1\}^n$ such that $\mathsf{pred}(x) = 1$ and $(x, w) \in \mathcal{R}$, then the experiment aborts and outputs a random bit.
  3. The challenger samples $\mathsf{crs}_0 \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{crs}_1 \leftarrow \mathsf{BindingSetup}(1^\lambda, 1^n, 1^s, \mathsf{pred})$, and sends $\mathsf{crs}_b$ to the adversary.
  4. The adversary sends $x \in \{0,1\}^n$ and a proof $\pi$.
  5. The adversary outputs a bit $b'$, which is the output of the experiment.

  We say that the SSS-SNARG satisfies mode indistinguishability if for any bounded adversary there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

## 8.1 Construction

The initial observation is that for any NP language $\mathcal{L}$, the Sahai-Waters SNARG can be made statistically sound for any fixed invalid instance $x \notin \mathcal{L}$ if the OWF is replaced with a (statistically binding) commitment scheme. To get statistical soundness for general predicates, we use a rerandomizable commitment and employ the same partitioning strategy used in Section 6 to get a perfectly partitionable hash.

**Construction 8.2.** The construction uses the following ingredients:

- Indistinguishability obfuscator $i\mathcal{O}$ defined in Definition 3.1.
- A puncturable PRF $\Pi_{\mathsf{PPRF}} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Puncture})$ defined in Definition 3.2.
- A rerandomizable commitment $\Pi_{\mathsf{Com}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Rerandomize})$ defined in Definition 3.3.

Fix any NP relation $\mathcal{R}$. The algorithms are defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^n, 1^s)$:

  1. Samples the public parameters of a rerandomizable commitment $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$. Let $r$ be the number of random coins needed by C.Com and C.Rerandomize.
  2. Samples $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^n, 1^r)$.
  3. $P \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenProof}[K])$ where GenProof is described in Fig. 18.
  4. $G \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenInst}[K, \mathsf{pp}])$ where GenInst is described in Fig. 19.
  5. Outputs $\mathsf{crs} = (P, G, \mathsf{pp})$.

**Constants:** A PPRF key $K$.
**Inputs:** An instance $x$ and a witness $w$.

1. Checks $\mathcal{R}(x, w) = 1$.
2. Outputs $\pi = \mathsf{PPRF.Eval}(K, x)$.

Figure 18: Program $\mathsf{GenProof}[K]$.

**Constants:** A PPRF key $K$, commitment public parameters pp.
**Inputs:** An instance $x$.

1. Outputs $\mathsf{C.Com}(\mathsf{pp}, 0; \mathsf{PPRF.Eval}(K, x))$.

Figure 19: Program $\mathsf{GenInst}[K, \mathsf{pp}]$.

- $\mathsf{BindingSetup}(1^\lambda, 1^n, 1^s, \mathsf{pred})$:
  1. Samples the public parameters of a rerandomizable commitment $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$.
  2. Samples $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^n, 1^r)$.
  3. Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 1; U_{\ell_r})$.
  4. $P \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenProof}[K])$ where GenProof is described in Fig. 18.
  5. $G \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenInst}'[K, \mathsf{pp}, \tilde{c}])$ where $\mathsf{GenInst}'$ is described in Fig. 20.
  6. Outputs $\mathsf{crs} = (P, G, \mathsf{pp})$.

**Constants:** A PPRF key $K$, commitment public parameters pp, a commitment $\tilde{c}$, a predicate pred.
**Inputs:** An instance $x$.

1. If $\mathsf{pred}(x) = 1$, outputs $\mathsf{C.Rerandomize}(\mathsf{pp}, \tilde{c}; \mathsf{PPRF.Eval}(K, x))$.
2. Else, outputs $\mathsf{C.Com}(\mathsf{pp}, 0; \mathsf{PPRF.Eval}(K, x))$.

Figure 20: Program $\mathsf{GenInst}'[K, \mathsf{pp}, \tilde{c}, \mathsf{pred}]$.

- $\mathsf{Prove}(\mathsf{crs}, x, w)$:
  1. Parses $\mathsf{crs} = (P, G, \mathsf{pp})$.
  2. Outputs $P(x, w)$.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi)$:
  1. Parses $\mathsf{crs} = (P, G, \mathsf{pp})$.
  2. Outputs 1 if and only if $\mathsf{C.Com}(\mathsf{pp}, 0; \pi) = G(x)$.

**Theorem 8.3.** *Assume that $i\mathcal{O}$ is a sub-exponentially secure indistinguishability obfuscation, $\Pi_{\mathsf{PPRF}}$ is a sub-exponentially secure puncturable PRF, and $\Pi_{\mathsf{Com}}$ is a polynomially secure rerandomizable perfectly-binding commitment. Then Construction 8.2 is a Sometimes-Statistically Sound SNARG.*

Theorem 8.3 follows from Lemmas 8.4 to 8.7.

**Lemma 8.4** (Perfect Completeness)**.** *If $i\mathcal{O}$ is functionality-preserving, then Construction 8.2 satisfies perfect completeness.*

*Proof.* Fix any $\lambda, n, s \in \mathbb{N}$ and let $x \in \{0,1\}^n, w \in \{0,1\}^n$ such that $(x, w) \in \mathcal{R}$. Let $K$ be a PPRF key and pp be commitment public parameters. By construction, the program $\mathsf{GenProof}(K)$ on input $(x, w)$ outputs $\pi = \mathsf{PPRF.Eval}(K, x)$. By construction, the program $\mathsf{GenInst}(K, \mathsf{pp})$ on input $x$ outputs $c = \mathsf{C.Com}(\mathsf{pp}, 0; \mathsf{PPRF.Eval}(K, x))$, and the verifier accepts since $\mathsf{C.Com}(\mathsf{pp}, 0; \pi) = c$. Since $i\mathcal{O}$ is functionality-preserving, the obfuscated programs preserve this behavior. Thus, perfect completeness holds. □

**Lemma 8.5** (Succinctness)**.** *If $\Pi_{\mathsf{PPRF}}$ has output length $\mathsf{poly}(\lambda)$, then Construction 8.2 satisfies succinctness.*

*Proof.* The proof $\pi$ is exactly the output of the PPRF, which is $\ell_r$ bits long, where $\ell_r$ is the number of random coins needed for C.Com, which is $\mathsf{poly}(\lambda)$. The claim follows. □

**Lemma 8.6** (Statistical Soundness in Binding Mode)**.** *If $\Pi_{\mathsf{Com}}$ is perfectly binding and satisfies rerandomization correctness, and $i\mathcal{O}$ is functionality-preserving, then Construction 8.2 satisfies statistical soundness in binding mode.*

*Proof.* Fix any $\lambda, n, s \in \mathbb{N}$ and any predicate $\mathsf{pred} : \{0,1\}^n \to \{0,1\}$ of size at most $s$. Let crs be any common reference string in the support of $\mathsf{BindingSetup}(1^\lambda, 1^n, 1^s, \mathsf{pred})$. By construction, the algorithm BindingSetup samples some commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 1; U_{\ell_r})$. Importantly, $\tilde{c}$ is a commitment of 1. For any instance $x$ such that $\mathsf{pred}(x) = 1$, by the construction of GenInst′ and functionality-preserving of $i\mathcal{O}$, $G(x)$ outputs a rerandomization of $\tilde{c}$, denote it by $c$. By rerandomization correctness of $\Pi_{\mathsf{Com}}$, $c$ is a commitment of 1. The verifier accepts if and only if $c = \mathsf{C.Com}(\mathsf{pp}, 0; \pi^*)$. But by perfect binding of $\Pi_{\mathsf{Com}}$, a commitment of 0 can never equal a commitment of 1. Thus, no such $\pi^*$ exists, and the verifier always rejects such $x$. The claim follows. □

## 8.2 Mode Indistinguishability of Construction 8.2

In this section, we prove that Construction 8.2 satisfies mode indistinguishability. The proof strategy is very similar to the one that appears in Section 6.1, therefore the proofs of our inner hybrids will be more succinct.

**Lemma 8.7** (Mode Indistinguishability)**.** *Assume that $i\mathcal{O}$ is $2^{-\lambda_{i\mathcal{O}}^{\delta}}$-secure, $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda_{\mathsf{PPRF}}^{\delta}}$-punctured pseudorandomness, and $\Pi_{\mathsf{Com}}$ is (polynomially) hiding and perfectly rerandomizable. If we set $\lambda_{i\mathcal{O}}(\lambda, n) = \lambda_{\mathsf{PPRF}}(\lambda, n) = (\lambda + n)^{1/\delta}$ then Construction 8.2 satisfies mode indistinguishability.*

Let $\mathcal{A}$ be an adversary for the mode indistinguishability game. We define the following mode indistinguishability experiments for Construction 8.2.

- $\mathsf{Real}_0$: In this game, the challenger samples a normal CRS.

    1. $\mathcal{A}$ chooses $n, s \in \mathbb{N}$ and a predicate $\mathsf{pred} : \{0,1\}^n \to \{0,1\}$ of size at most $s$, and sends $1^n, 1^s, \mathsf{pred}$ to the challenger. If there exists $x, w \in \{0,1\}^n$ such that $\mathsf{pred}(x) = 1$ and $(x, w) \in \mathcal{R}$, the experiment aborts.

2. The challenger does the following:
   (a) Samples commitment public parameters $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$.
   (b) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^n, 1^{\ell_r})$.
   (c) Samples $P \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenProof}[K])$ where GenProof is described in Fig. 18.
   (d) Samples $G \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenInst}[K, \mathsf{pp}])$ where GenInst is described in Fig. 19.

   The challenger sends $\mathsf{crs} = (P, G, \mathsf{pp})$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

- $\mathsf{Real}_1$: In this game, the challenger samples a binding CRS.

1. $\mathcal{A}$ chooses $n, s \in \mathbb{N}$ and a predicate $\mathsf{pred} : \{0,1\}^n \to \{0,1\}$ of size at most $s$, and sends $1^n, 1^s, \mathsf{pred}$ to the challenger. If there exists $x, w \in \{0,1\}^n$ such that $\mathsf{pred}(x) = 1$ and $(x, w) \in \mathcal{R}$, the experiment aborts.
2. The challenger does the following:
   (a) Samples commitment public parameters $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$.
   (b) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Gen}(1^{\lambda_{\mathsf{PPRF}}}, 1^n, 1^{\ell_r})$.
   (c) Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 1; U_{\ell_r})$.
   (d) Samples $P \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenProof}[K])$ where GenProof is described in Fig. 18.
   (e) Samples $G \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenInst}'[K, \mathsf{pp}, \tilde{c}, \mathsf{pred}])$ where $\mathsf{GenInst}'$ is described in Fig. 20.

   The challenger sends $\mathsf{crs} = (P, G, \mathsf{pp})$ to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit $b'$ which is the output of the experiment.

Next, we define the following sequence of hybrids for each $x^* \in [0, 2^n]$:

- $\mathsf{Hyb}_{x^*}$: In this experiment, the challenger samples a transitional CRS where $G$ behaves like a normal key for $x \geq x^*$, and "almost" like a binding key for $x < x^*$. We say "almost" because the commitment $\tilde{c}$ at this point is a commitment of 0.

2. The challenger does the following changes to how it samples the CRS:
   (c) Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 0; U_{\ell_r})$.
   (d) Samples $G \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenInst}[K, \mathsf{pp}, x^*, \tilde{c}, \mathsf{pred}])$ where GenInst is described in Fig. 21.

**Constants:** A PPRF key $K$, commitment public parameters pp, an integer $x^*$, a commitment $\tilde{c}$, a predicate pred.
**Inputs:** An instance $x \in \{0,1\}^n$

1. If $x \geq x^*$, outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x)\big)$.
2. If $x < x^*$, then
   (a) If $\mathsf{pred}(x) = 0$, outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x)\big)$.
   (b) If $\mathsf{pred}(x) = 1$, outputs $\mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, \mathsf{PPRF.Eval}(K, x)\big)$.

Figure 21: Program $\mathsf{GenInst}[K, \mathsf{pp}, x^*, \tilde{c}, \mathsf{pred}]$.

- $\mathsf{Hyb}_{x^*,1}$: Same as $\mathsf{Hyb}_{x^*}$, except that if $\mathsf{pred}(x^*) = 1$, the PPRF key is punctured at $x^*$. The program $P$ is otherwise unchanged. For $G$, we hardcode the commitment corresponding to $x^*$.

  2. If $\mathsf{pred}(x^*) = 1$, the challenger does the following changes to how it samples the CRS:
     (d) Punctures $K\{x^*\} = \mathsf{PPRF.Puncture}(K, x^*)$.
     (e) Computes $c^* = \mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K, x^*)\big)$.
     (f) Samples $P \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenProof}[K\{x^*\}])$ where GenProof is described in Fig. 18.
     (g) Samples $G \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenInst}[K\{x^*\}, \mathsf{pp}, x^*, \tilde{c}, c^*, \mathsf{pred}])$ where GenInst is described in Fig. 22.

  If $\mathsf{pred}(x^*) = 0$, the challenger behaves exactly like in $\mathsf{Hyb}_{x^*}$.

**Constants:** A punctured PPRF key $K\{x^*\}$, commitment public parameters pp, an integer $x^*$, a commitment $\tilde{c}$, an additional commitment $c^*$, a predicate pred.
**Inputs:** An instance $x \in \{0,1\}^n$

1. If $x > x^*$, outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K\{x^*\}, x)\big)$.
2. If $x = x^*$, outputs $c^*$.
3. If $x < x^*$, then
   (a) If $\mathsf{pred}(x) = 0$, outputs $\mathsf{C.Com}\big(\mathsf{pp}, 0, \mathsf{PPRF.Eval}(K\{x^*\}, x)\big)$.
   (b) If $\mathsf{pred}(x) = 1$, outputs $\mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, \mathsf{PPRF.Eval}(K\{x^*\}, x)\big)$.

Figure 22: Program $\mathsf{GenInst}[K\{x^*\}, \mathsf{pp}, x^*, \tilde{c}, c^*, \mathsf{pred}]$.

- $\mathsf{Hyb}_{x^*,2}$: Same as $\mathsf{Hyb}_{x^*,1}$, except that if $\mathsf{pred}(x^*) = 1$, then $c^*$ is sampled as a fresh commitment of 0.

  2. If $\mathsf{pred}(x^*) = 1$, the challenger does the following changes to how it samples the CRS:
     (e) Samples $c^* = \mathsf{C.Com}\big(\mathsf{pp}, 0, U_{\ell_r}\big)$.

- $\mathsf{Hyb}_{x^*,3}$: Same as $\mathsf{Hyb}_{x^*,2}$, except that if $\mathsf{pred}(x^*) = 1$, then $c^*$ is sampled as a rerandomization of $\tilde{c}$.

  2. If $\mathsf{pred}(x^*) = 1$, the challenger does the following changes to how it samples the CRS:
     (e) Samples $c^* = \mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, U_{\ell_r}\big)$.

- $\mathsf{Hyb}_{x^*,4}$: Same as $\mathsf{Hyb}_{x^*,3}$, except that if $\mathsf{pred}(x^*) = 1$, then $c^*$ is sampled as a pseudorandom rerandomization of $\tilde{c}$, where the coins are derived using $K$.

  2. If $\mathsf{pred}(x^*) = 1$, the challenger does the following changes to how it samples the CRS:
     (e) Sets $c^* = \mathsf{C.Rerandomize}\big(\mathsf{pp}, \tilde{c}, \mathsf{PPRF.Eval}(K, x^*)\big)$.

- $\mathsf{Hyb}_{x^*,5}$: Same as $\mathsf{Hyb}_{x^*,4}$, except that the PPRF key is no longer punctured, and we go back to using the unpunctured programs. This is identical to $\mathsf{Hyb}_{x^*+1}$ (for $x^* < 2^n$).

  1. Samples $P \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenProof}[K])$ where GenProof is described in Fig. 18.
  2. Samples $G \leftarrow i\mathcal{O}(1^{\lambda_{i\mathcal{O}}}, \mathsf{GenInst}[K, \mathsf{pp}, x^* + 1, \tilde{c}, \mathsf{pred}])$ where GenInst is described in Fig. 21.

Finally, we define the following experiment:

- $\mathsf{Hyb}'_{2^n}$: Same as $\mathsf{Hyb}_{2^n,5}$, except that $\tilde{c}$ is sampled as a commitment of 1 instead of 0.

2. The challenger does the following changes to how it samples the CRS:
   (c) Samples a commitment $\tilde{c} \leftarrow \mathsf{C.Com}(\mathsf{pp}, 1; U_{\ell_r})$.

We start by proving the two following claims about the first and last hybrids in the sequence:

**Claim 8.8.** *Assume that $i\mathcal{O}$ is polynomially secure. Then for any bounded adversary $\mathcal{A}$ we have*

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

*Proof.* By construction, the program $G$ in $\mathsf{Real}_0$ and $\mathsf{Hyb}_0$ are functionally equivalent, as $x^* = 0$ implies $x \geq x^*$ is always true. Thus, an adversary distinguishing the two hybrids can be used to break the security of $i\mathcal{O}$. □

**Claim 8.9.** *Assume that $i\mathcal{O}$ is polynomially secure. Then for any bounded adversary $\mathcal{A}$ we have*

$$\left|\Pr[\mathsf{Hyb}'_{2^n}(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

*Proof.* This claim is completely analogous to Claim 8.8. □

Next, we prove the following claims about the inner hybrids for all $x^* \in [2^n]$.

**Claim 8.10.** *Assume that $i\mathcal{O}$ is $2^{-\lambda-n}$-secure and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

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

*Proof.* If $\mathsf{pred}(x^*) = 0$, then the two hybrids are identical, so assume $\mathsf{pred}(x^*) = 1$. The programs in $\mathsf{Hyb}_{x^*}$ and $\mathsf{Hyb}_{x^*,1}$ are functionally equivalent. For $P$, notice that if $\mathsf{pred}(x^*) = 1$, the mode indistinguishability game guarantees there is no $w$ such that $(x^*, w) \in \mathcal{R}$. Thus $P$ in $\mathsf{Hyb}_{x^*}$ would never evaluate the PRF at $x^*$ because it checks $\mathcal{R}(x^*, w) = 1$ first. If $\mathsf{pred}(x^*) = 0$, the PRF evaluation is identical. Thus the punctured program is functionally equivalent to the original one. For $G$, the functional equivalence follows directly by correctness of the PPRF at the punctured point. The claim follows by the $2^{-\lambda^{\delta}_{i\mathcal{O}}}$-security of $i\mathcal{O}$. □

**Claim 8.11.** *Assume that $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda-n}$-punctured pseudorandomness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

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

*Proof.* If $\mathsf{pred}(x^*) = 0$, the two hybrids are identical. If $\mathsf{pred}(x^*) = 1$, the value $\mathsf{PPRF.Eval}(K, x^*)$ is only used to compute $c^* = \mathsf{C.Com}(\mathsf{pp}, 0; \mathsf{PPRF.Eval}(K, x^*))$ in $\mathsf{Hyb}_{x^*,1}$. This value can be replaced by a truly random string, matching $\mathsf{Hyb}_{x^*,2}$, by the punctured pseudorandomness of $\Pi_{\mathsf{PPRF}}$. The claim follows. □

**Claim 8.12.** *Assume that $\Pi_{\mathsf{Com}}$ satisfies perfect rerandomization. Then for any (unbounded) adversary $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{Hyb}_{x^*,2}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{x^*,3}(\mathcal{A}) = 1]\right| = 0.$$

*Proof.* The only difference is how $c^*$ is generated when $\mathsf{pred}(x^*) = 1$. In $\mathsf{Hyb}_{x^*,2}$, it is a fresh commitment of 0. In $\mathsf{Hyb}_{x^*,3}$, it is a rerandomization of $\tilde{c}$, which is also a commitment of 0. By perfect rerandomization, these distributions are identical. □

**Claim 8.13.** *Assume that $\Pi_{\mathsf{PPRF}}$ satisfies $2^{-\lambda-n}$-punctured pseudorandomness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

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

*Proof.* This claim is completely analogous to Claim 8.11. □

**Claim 8.14.** *Assume that $i\mathcal{O}$ is $2^{-\lambda-n}$-secure and $\Pi_{\mathsf{PPRF}}$ satisfies punctured correctness. Then for any polynomial time adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$*

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

*Proof.* This claim is completely analogous to Claim 8.10. □

**Lemma 8.15.** *Assume that $\Pi_{\mathsf{Com}}$ satisfies computational hiding. Then for any bounded adversary $\mathcal{A}$ we have*

$$\left|\Pr[\mathsf{Hyb}_{2^n}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}'_{2^n}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

*Proof.* The only difference is whether $\tilde{c}$ is a commitment of 0 or 1. This reduces directly to the hiding property of $\Pi_{\mathsf{Com}}$. □

**Proof of Lemma 8.7.** By combining Claims 8.10 to 8.14 a total of $2^n$ times, we get that there exists $\Lambda$ such that for all $\lambda \geq \Lambda$:

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2^n}(\mathcal{A}) = 1]\right| \leq O(2^{-\lambda}).$$

If $\Pi_{\mathsf{Com}}$ is computationally hiding we can apply Lemma 8.15 to get

$$\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}'_{2^n}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

By considering Claims 8.8 and 8.9 as well, we get

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

and the lemma follows.

# 9 Constructing Somewhere-Extractable BARGs

In this section, we combine SSS-SNARGs and EPH to construct a somewhere proof of knowledge BARG. For simplicity, assume the NP relation $\mathcal{R}$ is hardcoded, and assume that the instances and witnesses have the same length $n$. The batch size is $k$.

**Construction 9.1.** The construction uses the following ingredients:

- A multi-output extractable partitionable hash $\Pi_{\mathsf{PH}} = (\mathsf{Gen}, \mathsf{GenPart}, \mathsf{Hash}, \mathsf{Extract})$ defined in Definition 4.2 and Remark 4.4. We will use EPH for functions with $n$ output bits. Let $\ell := \ell(\lambda, n)$ be the length of the digest.
- A sometimes-statistically sound SNARG $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{BindingSetup}, \mathsf{Prove}, \mathsf{Verify})$ for the NP relation
$$\mathcal{R}^* = \left\{ \left( (\mathbf{x}, \mathsf{hk}, \mathsf{dig}), \mathbf{w} \right) : \mathsf{Hash}(\mathsf{hk}, \mathbf{w}) = \mathsf{dig} \text{ and } \forall i \in [k], \mathcal{R}(x_i, w_i) = 1 \right\}$$
and denote the corresponding language by $\mathcal{L}^*$.

Let $s := s(\lambda, n, k) = \Omega(|\mathcal{R}| + |\mathsf{PH.Extract}|)$. Specifically, $s$ upper bounds the size of $\mathsf{pred}_{i^*}$ defined below. Let $m := kn + \ell$. The algorithms are defined as follows:

- $\mathsf{Gen}(1^\lambda, 1^k, 1^n)$:
  1. Samples $\mathsf{hk} \leftarrow \mathsf{PH.Gen}(1^\lambda, 1^{kn}, 1^s)$.
  2. Samples $\mathsf{crs}_{\mathsf{SNARG}} \leftarrow \mathsf{SNARG.Setup}(1^\lambda, 1^m, 1^s)$.
  3. Outputs $\mathsf{crs} = (\mathsf{crs}_{\mathsf{SNARG}}, \mathsf{hk})$.
- $\mathsf{TrapGen}(1^\lambda, 1^k, 1^n, i^*)$:
  1. Samples $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{PH.GenPart}(1^\lambda, 1^{kn}, 1^s, \mathsf{splt}_{i^*})$, where $\mathsf{splt}_{i^*}(\mathbf{w}) = w_{i^*}$.
  2. Samples $\mathsf{crs}_{\mathsf{SNARG}} \leftarrow \mathsf{SNARG.BindingSetup}(1^\lambda, 1^m, 1^s, \mathsf{pred}_{i^*})$, where
$$\mathsf{pred}_{i^*}(\mathbf{x}, \mathsf{dig}) = \neg \mathcal{R}(x_{i^*}, \mathsf{PH.Extract}(\mathsf{td}, \mathsf{dig})).$$
  3. Outputs $\mathsf{crs} = (\mathsf{crs}_{\mathsf{SNARG}}, \mathsf{hk})$ and $\mathsf{td}$.
- $\mathsf{Prove}(\mathsf{crs}, \mathbf{x}, \mathbf{w})$:
  1. Parses $\mathsf{crs} = (\mathsf{crs}_{\mathsf{SNARG}}, \mathsf{hk})$.
  2. Hashes $\mathsf{dig} = \mathsf{PH.Hash}(\mathsf{hk}, \mathbf{w})$.
  3. Outputs $\pi = \left(\mathsf{dig}, \mathsf{SNARG.Prove}(\mathsf{crs}_{\mathsf{SNARG}}, \mathbf{x} \circ \mathsf{dig}, \mathbf{w})\right)$.
- $\mathsf{Verify}(\mathsf{crs}, \mathbf{x}, \pi)$:
  1. Parses $\mathsf{crs} = (\mathsf{crs}_{\mathsf{SNARG}}, \mathsf{hk})$.
  2. Parses $\pi = (\mathsf{dig}, \sigma)$.
  3. Outputs $\mathsf{SNARG.Verify}(\mathsf{crs}_{\mathsf{SNARG}}, \mathbf{x} \circ \mathsf{dig}, \sigma)$.
- $\mathsf{Extract}(\mathsf{td}, \pi)$:
  1. Parses $\pi = (\mathsf{dig}, \sigma)$.
  2. Outputs $\mathsf{PH.Extract}(\mathsf{td}, \mathsf{dig})$.

**Theorem 9.2.** *Assume that $\Pi_{\mathsf{SNARG}}$ is a secure sometimes-statistically sound SNARG and that $\Pi_{\mathsf{PH}}$ is a secure extractable partitionable hash. Then Construction 9.1 is a somewhere proof of knowledge BARG. Moreover, if $\Pi_{\mathsf{PH}}$ is rate-1 then Construction 9.1 is rate-1.*

Theorem 9.2 will follow by Lemmas 9.3 to 9.6.

**Lemma 9.3.** *If $\Pi_{\mathsf{SNARG}}$ is complete, then Construction 9.1 is complete.*

*Proof.* Fix $\lambda, n, k \in \mathbb{N}$, and let $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^{kn}$ and $\mathbf{w} = (w_1, \ldots, w_k) \in \{0,1\}^{kn}$ such that $(x_i, w_i) \in \mathcal{R}$ for all $i \in [k]$. By construction, the prover hashes $\mathsf{dig} = \mathsf{PH.Hash}(\mathsf{hk}, \mathbf{w})$ and computes a SNARG proof that $\big((\mathbf{x}, \mathsf{hk}, \mathsf{dig}), \mathbf{w}\big) \in \mathcal{R}^*$. By definition, the tuple $\big((\mathbf{x}, \mathsf{hk}, \mathsf{dig}), \mathbf{w}\big)$ indeed satisfies the relation, so completeness follows by the completeness of $\Pi_{\mathsf{SNARG}}$. □

**Lemma 9.4.** *If $\Pi_{\mathsf{PH}}$ has digests of length $\ell$ and $\Pi_{\mathsf{SNARG}}$ is succinct, then Construction 9.1 has proofs of size $\ell + \mathsf{poly}(\lambda)$. Specifically, if $\Pi_{\mathsf{PH}}$ satisfies standard digest succinctness then Construction 9.1 is succinct. If $\Pi_{\mathsf{PH}}$ is rate-1, then Construction 9.1 is rate-1 and has proofs of size $n + \mathsf{poly}(\lambda)$.*

*Proof.* The proof $\pi$ consists of a digest dig and a SNARG proof $\sigma$. By the assumption, dig is $\ell$ bits long. By the succinctness of $\Pi_{\mathsf{SNARG}}$, we have $|\sigma| = \mathsf{poly}(\lambda)$. Therefore the total length of the proof is $\ell + \mathsf{poly}(\lambda)$ and the claim follows. □

**Lemma 9.5.** *If $\Pi_{\mathsf{SNARG}}$ is statistically sound in binding mode and $\Pi_{\mathsf{PH}}$ is perfectly extractable, then Construction 9.1 is a somewhere proof of knowledge.*

*Proof.* Fix $\lambda, n, k \in \mathbb{N}$ and $i^* \in [k]$. Let $\mathsf{crs}_{\mathsf{SNARG}} \leftarrow \mathsf{SNARG.BindingSetup}(1^\lambda, 1^m, 1^s, \mathsf{pred}_{i^*})$ and $\mathsf{hk} \leftarrow \mathsf{PH.GenPart}(1^\lambda, 1^{kn}, 1^s, \mathsf{splt}_{i^*})$. Recall that $\mathsf{pred}_{i^*}(\mathbf{x}, \mathsf{dig}) = 1$ if and only if $\neg\mathcal{R}(x_{i^*}, \mathsf{PH.Extract}(\mathsf{td}, \mathsf{dig}))$.

Fix any $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^{kn}$ and any proof $\pi = (\mathsf{dig}, \sigma)$. Suppose that $\mathsf{Verify}(\mathsf{crs}_{\mathsf{SNARG}}, \mathsf{hk}, \mathbf{x}, \pi) = 1$. By construction of the verifier, this implies that $\mathsf{SNARG.Verify}(\mathsf{crs}_{\mathsf{SNARG}}, \mathbf{x} \circ \mathsf{dig}, \sigma) = 1$. By the statistical soundness in binding mode of $\Pi_{\mathsf{SNARG}}$, for any statement $st = \mathbf{x} \circ \mathsf{dig}$ such that $\mathsf{pred}_{i^*}(st) = 1$, the verifier accepts with negligible probability. Thus, with all but negligible probability, we must have $\mathsf{pred}_{i^*}(\mathbf{x} \circ \mathsf{dig}) = 0$. By definition of $\mathsf{pred}_{i^*}$, this means that $\mathcal{R}(x_{i^*}, \mathsf{PH.Extract}(\mathsf{td}, \mathsf{dig})) = 1$. Since $\mathsf{Extract}(\mathsf{td}, \pi)$ outputs $w^* = \mathsf{PH.Extract}(\mathsf{td}, \mathsf{dig})$, it follows that $\mathcal{R}(x_{i^*}, w^*) = 1$, as required. □

**Lemma 9.6.** *If $\Pi_{\mathsf{SNARG}}$ satisfies mode indistinguishability and $\Pi_{\mathsf{PH}}$ satisfies mode indistinguishability and perfect extraction, then Construction 9.1 satisfies index hiding.*

*Proof.* Fix any $\lambda, n, k \in \mathbb{N}$ and $i^* \in [k]$. We show that crs sampled from $\mathsf{Gen}(1^\lambda, 1^k, 1^n)$ is computationally indistinguishable from crs sampled from $\mathsf{TrapGen}(1^\lambda, 1^k, 1^n, i^*)$ using a hybrid argument. Recall that in Gen, the hash key hk is sampled from PH.Gen, while in TrapGen, it is sampled from $\mathsf{PH.GenPart}(\mathsf{splt}_{i^*})$. By the mode indistinguishability of $\Pi_{\mathsf{PH}}$, these hash keys are computationally indistinguishable. Thus, Gen is indistinguishable from a hybrid experiment where hk is sampled from PH.GenPart but $\mathsf{crs}_{\mathsf{SNARG}}$ is still sampled from SNARG.Setup.

Next, we switch $\mathsf{crs}_{\mathsf{SNARG}}$ from SNARG.Setup to $\mathsf{SNARG.BindingSetup}(\mathsf{pred}_{i^*})$. To use mode indistinguishability of the $\Pi_{\mathsf{SNARG}}$, we need to prove that $\mathsf{pred}_{i^*}(\mathbf{x}, \mathsf{dig}) = 1 \Rightarrow (\mathbf{x}, \mathsf{dig}) \notin \mathcal{L}^*$. Let $(\mathbf{x}, \mathsf{dig}) \in \mathcal{L}^*$ and denote $w^* = \mathsf{PH.Extract}(\mathsf{td}, \mathsf{dig})$. By the perfect extraction of $\Pi_{\mathsf{PH}}$ on $\mathsf{splt}_{i^*}$, for any $\mathbf{w} \in \{0,1\}^{kn}$ such that $\mathsf{PH.Hash}(\mathsf{hk}, \mathbf{w}) = \mathsf{dig}$, we have $\mathsf{PH.Extract}(\mathsf{td}, \mathsf{dig}) = w^*$. By the fact that $(\mathbf{x}, \mathsf{dig}) \in \mathcal{L}^*$, we have in particular that $\mathcal{R}(x_{i^*}, w^*) = 1$. Therefore $\mathsf{pred}_{i^*}(\mathbf{x}, \mathsf{dig}) = 0$. This allows us to invoke the mode indistinguishability of $\Pi_{\mathsf{SNARG}}$ to switch $\mathsf{crs}_{\mathsf{SNARG}}$ to be statistically sound on $\mathsf{pred}_{i^*}$. This hybrid experiment corresponds exactly to $\mathsf{TrapGen}(i^*)$. This concludes the proof. □

## Acknowledgments

The author would like to thank George Lu for endless discussions and feedback, and Brent Waters and David Wu for feedback and comments on a preliminary version of the results in this work. This work was done in part while the author was visiting the Simons Institute for the Theory of Computing in Summer 2025.

## References

[ABY25] Gal Arnon, Ben-David, Shany, and Eylon Yogev. Instance compression, revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 244–273. Springer, 2025.

[ADKL19] Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. Fully homomorphic nizk and niwi proofs. In *Theory of Cryptography Conference*, pages 356–385. Springer, 2019.

[AS16] Gilad Asharov and Gil Segev. Limits on the power of indistinguishability obfuscation and functional encryption. *SIAM Journal on Computing*, 45(6):2117–2176, 2016.

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

[BBPS26] Zvika Brakerski, Maya Farber Brodsky, Omer Paneth, and Tomer Solomon. Mergeable snargs for trapdoor languages and their applications. *Cryptology ePrint Archive*, 2026.

[BD19] Nir Bitansky and Akshay Degwekar. On the complexity of collision resistant hash functions: New and old black-box separations. In *Theory of Cryptography Conference*, pages 422–450. Springer, 2019.

[BDGM19] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In *Theory of Cryptography Conference*, pages 407–437. Springer, 2019.

[BDSZ24] Pedro Branco, Nico Döttling, Akshayaram Srinivasan, and Riccardo Zanotto. Rate-1 fully local somewhere extractable hashing from ddh. In *IACR International Conference on Public-Key Cryptography*, pages 356–386. Springer, 2024.

[BDV17] Nir Bitansky, Akshay Degwekar, and Vinod Vaikuntanathan. Structure vs. hardness through the obfuscation lens. In *Annual International Cryptology Conference*, pages 696–723. Springer, 2017.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), 2012.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *International workshop on public key cryptography*, pages 501–519. Springer, 2014.

[BHZ87] Ravi B Boppana, Johan Hastad, and Stathis Zachos. Does co-np have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.

[BKP+24] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron D. Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. In *STOC*, pages 435–443, 2024.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology-ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II 19*, pages 280–300. Springer, 2013.

[BWW24] Eli Bradley, Brent Waters, and David J. Wu. Batch arguments to nizks from one-way functions. In *TCC*, 2024.

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

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

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

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *TCC*, pages 468–497, 2015.

[CW23] Jeffrey Champion and David J. Wu. Non-interactive zero-knowledge from non-interactive batch arguments. In *CRYPTO*, pages 38–71, 2023.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *FOCS*, pages 1057–1068, 2022.

[DHJQ26] Jesko Dujmovic, Yao-Ching Hsieh, Abhishek Jain, and Willy Quach. How to use polynomially-hard io: Turing machine obfuscation and more. In *Annual International Cryptology Conference*, pages 317–349. Springer, 2026.

[DJJ+25] Pratish Datta, Abhishek Jain, Zhengzhong Jin, Alexis Korb, Surya Mathialagan, and Amit Sahai. Incrementally verifiable computation for np from standard assumptions. In *Annual International Cryptology Conference*, pages 611–642. Springer, 2025.

[DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J Wu. Succinct witness encryption for batch languages and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 130–162. Springer, 2025.

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

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 467–476, 2013.

[GH19] Craig Gentry and Shai Halevi. Compressible fhe with applications to pir. In *Theory of cryptography conference*, pages 438–464. Springer, 2019.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377, 1982.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):1–35, 2012.

[GSWW22] Rachit Garg, Kristin Sheridan, Brent Waters, and David J Wu. Fully succinct batch arguments for np from indistinguishability obfuscation. In *Theory of Cryptography Conference*, pages 526–555. Springer, 2022.

[GVW02] Oded Goldreich, Salil Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *computational complexity*, 11(1):1–53, 2002.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108, 2011.

[HN10] Danny Harnik and Moni Naor. On the compressibility of np instances and cryptographic applications. *SIAM Journal on Computing*, 39(5):1667–1713, 2010.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, pages 163–172, 2015.

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

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 669–684, 2013.

[NW26] Shafik Nassar and Brent Waters. Adaptive nike for unbounded parties. *Cryptology ePrint Archive*, 2026.

[NWW24] Shafik Nassar, Brent Waters, and David J Wu. Monotone policy bargs from bargs and additively homomorphic encryption. In *Theory of Cryptography Conference*, pages 399–430. Springer, 2024.

[NWW25] Shafik Nassar, Brent Waters, and David J Wu. Monotone-policy bargs and more from bargs and quadratic residuosity. In *IACR International Conference on Public-Key Cryptography*, pages 283–313. Springer, 2025.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *FOCS*, pages 1045–1056, 2022.

[Rev26] Anonymous Reviewer. Anonymous review by a TCC 2026 program committee member. 2026.

[Sim98] Daniel R Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 334–345. Springer, 1998.

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

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

[WW24] Brent Waters and David J Wu. Adaptively-sound succinct arguments for np from indistinguishability obfuscation. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 387–398, 2024.

[WW25] Brent Waters and David J Wu. A pure indistinguishability obfuscation approach to adaptively-sound snargs for np. Annual International Cryptology Conference (CRYPTO), 2025.

[WZ24] Brent Waters and Mark Zhandry. Adaptive security in snargs via io and lossy functions. In *Annual International Cryptology Conference*, pages 72–104. Springer, 2024.

[YYS25] Rupeng Yang, Zuoxia Yu, and Willy Susilo. Fully homomorphic encryption with chosen-ciphertext security from lwe. In *Annual International Cryptology Conference*, pages 371–405. Springer, 2025.