Adaptive Security in SNARGs via iO and Lossy Functions

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

Mark Zhandry
NTT Research
mzhandry@gmail.com

Abstract

We construct an adaptively sound SNARGs in the plain model with CRS relying on the assumptions of (subexponential) indistinguishability obfuscation (iO), subexponential one-way functions and a notion of lossy functions we call length parameterized lossy functions. Length parameterized lossy functions take in separate security and input length parameters and have the property that the function image size in lossy mode depends only on the security parameter. We then show a novel way of constructing such functions from the Learning with Errors (LWE) assumption.

Our work provides an alternative path towards achieving adaptively secure SNARGs from the recent work of Waters and Wu [WW24]. Their work required the use of (essentially) perfectly re-randomizable one way functions (in addition to obfuscation). Such functions are only currently known to be realizable from assumptions such as discrete log or factoring that are known to not hold in a quantum setting.

1 Introduction

A Succinct Non-interactive Argument (SNARG) for NP are an important foundational object in cryptography. A SNARG allows a computationally bounded prover to convince a verifier of some NP statement by sending only a single short message, much shorter than the statement and witness size. Succinct arguments were first constructed in the random oracle model (ROM) [Kil92, Mic94]. This was followed by numerous constructions in the plain model assuming a CRS; such a CRS is likely inherent to the plain model [BP04, Wee05]. In the CRS model, the CRS is allowed to be longer than the instance and witness size, but the prover's message must still remain short.

In the plain model with a CRS, an important security goal is that of *adaptive soundness*, which allows the bounded prover to choose the instance being proved adaptively after seeing the CRS. Adaptive soundness in the plain model has been very difficult to achieve.

Our Work. In this work, we construct adaptively sound SNARGs in the plain model with CRS, assuming (subexponential) indistinguishability obfuscation (iO) plus sufficiently lossy functions (as well as subexponential one-way functions). We can instantiate the necessary lossy functions using LWE, which in turn can be based on the worst-case hardness of lattice assumptions. Previous lossy functions from LWE were not sufficiently lossy for our needs, so we designed a new lossy function from LWE which may be useful in other contexts.

Our work complements a very recent result of [WW24], which constructs the first adaptively sound SNARGs, using (subexponential) iO plus sufficiently strong rerandomizeable one-way functions (OWFs). They instantiated the needed rerandomizeable OWFs from the discrete logarithm or factoring assumptions. Importantly, even though LWE is rerandomizeable, the statistical error in the rerandomization process is too great for the reduction in [WW24] to go through, so their result cannot be instantiated using LWE. In particular, their claimed security proof requires pre-quantum assumptions, leaving open the existance of post-quantum secure SNARGs from falsifiable assumptions.

As a technical contribution crucial to obtaining our result, we construct lossy trapdoor functions (LTDFs) from LWE. While such functions were previously known, ours achieve a very strong form of lossiness in the lossy mode. In more detail, we allow the input length to be an arbitrarily large polynomial in the security parameter. But we require that the number of bits of information about the input that is present in the output of the lossy mode is a fixed polynomial in the security parameter, independent of the input length. Previous lossy functions based on LWE did not achieve this level of lossiness.

1.1 Motivation and Challenges

Challenges in proving SNARGs in the standard model. When reducing to falsifiable assumptions, [GW11, CGKS23] argue that adaptive SNARGs require the reduction to run in time larger than the statement size. One can try to compensate for such an inefficient reduction by incurring a loss of $2^{|x|}$ in the security reduction, where x is the statement, and making subexponential hardness assumptions. However, in order to accommodate a $2^{|x|}$ security loss, the security parameter needs to be set somewhat larger than $|x|$. Consequently, the proof size grows with the security parameter, meaning the proof size ends up growing with the instance size, making the protocol no longer succinct. As a result, it may seem that provable adaptive soundness for SNARGs under falsifiable assumptions is not possible. Indeed, many constructions in the literature use non-falsifiable assumptions instead (e.g. [Gro10, BCCT12, DFH12, Lip13, GGPR13, BCI+13, BCC+17, ACL+22, CLM23]).

Very recently, [WW24] overcome this challenge. Their key idea is to actually have two security parameters. One security parameter will absorb the $2^{|x|}$ loss of the inefficient reduction and will therefore be large, while the other security parameter will remain small; the arguments of [GW11, CGKS23] only imply that one of the two security parameters needs to be large. Importantly, the proof size only depends on the small security parameter, meaning it can remain small. Meanwhile, the CRS will depend on both security parameters and will therefore grow with the instance size, as expected. Even with this idea, executing a security proof that prevents the small security parameter from incurring the $2^{|x|}$ loss is tricky and requires a careful proof. The main issue is that typical hybrid arguments end up requiring the subexponential security of *all* primitives involved, in which case all security parameters would need to be made large.

What about LWE? The construction in [WW24] uses iO, as well as rerandomizeable one-way functions. Importantly for their proof, rerandomization must produce a distribution that is identical to the original distribution, or a slight generalization allows for the rerandomized distribution to be made $2^{-\ell}$ close, where ℓ needs to be much larger than the image size of the one-way function. The authors show how to instantiate such a function using either discrete logarithms or factoring.

A natural question is whether the existing adaptive SNARGs can be instantiated using LWE (in addition to iO). For example, using an alternative to discrete logarithms or factoring would be necessary if one wants post-quantum security. Along similar lines, perhaps some day it will be discovered that iO can be built from LWE; following [WW24] would require making assumptions in addition to LWE, whereas our work would then only require LWE.

Unfortunately, while LWE is rerandomizeable, rerandomization introduces noise, so the rerandomized distribution is only statistically close to the original distribution. Importantly, for LWE the closeness $2^{-\ell}$ must always have ℓ be much smaller than the instance size, meaning LWE cannot be used in the proof of [WW24].

Assumptions in Obfustopia. Indistinguishability obfuscation can be used to achieve many cryptographic applications. However, iO alone is almost never enough: indeed, if $\mathsf{P} = \mathsf{NP}$, then iO exists unconditionally, but most of the cryptosystems we would like to build from iO cannot exist. As a result, some extra computational assumption is typically required. The minimal extra assumption is often a one-way function, which, in a world where iO exists, can in turn can be replaced with a worst-case complexity assumption [KMN+14].

However, it turns out that even iO plus one-way functions are often not enough to achieve various structured primitives. For example, iO and one-way functions are not enough for collision resistance [AS15] (and therefore anything that implies collision resistance such as homomorphic encryption), one-way permutations [AS18], or even any hard problem in $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ [BDV17]. A natural theoretical question is then to explore what structured assumptions are needed in addition to iO to achieve these applications.

The work of [WW24] shows that sufficiently strong rerandomizeable one-way functions plus (subexponential) iO are enough to achieve SNARGs. Rerandomizeablility is a kind of structure that is typically not achievable with iO and one-way functions alone. For example, (subexponential) rerandomizeable encryption and iO gives fully homomorphic encryption [CLTV15], which as discussed above likely cannot be achieved from iO and one-way functions.

Our work therefore complements [WW24], by showing that a different structure, namely lossiness, can be used in conjunction with iO to achieve SNARGs. Given the state-of-the-art, lossy functions and rerandomizeable one-way functions seem incomparable, though we note that other rerandomizeable primitives imply types of lossiness[1], suggesting that losiness may typically be a milder structure.

1.2 Technical Overview

The [SW14] SNARG. We start by briefly recalling the obfuscation-based SNARG of [SW14], which was the first SNARG with provable security under falsifiable assumptions.

The prover's message in [SW14] is simply a signature on the statement being proved, using an appropriate signature scheme. The verifier can easily verify these signatues, and signatures can be made very short, much shorter than the instace size. In order to allow the honest prover to actually compute the signature, the CRS contains an obfuscated program P which has the signing key hard-coded. P takes as input the statement x and witness w, and signs x if and only if the witness w is a valid witness for x. To prove security, [SW14] argue that for false statements x^*, one can move to a hybrid game where the program P contains a "punctured" signing key that is incapable of signing x^*. Generating a proof for x^* then means generating a signature on x^*, despite not having the ability to sign x^*, which violates the security of the punctured signature scheme.

Puncturing the signing key at x^* requires fully specifying x^* in the program P, meaning the adversary needs to commit to x^* before seeing the CRS. This is why [SW14] are limited to selective security. One could attempt to guess x^* and incur a loss of $2^{-|x^*|}$, and then set the security parameter to be much larger than $|x^*|$ to compensate (assuming subexponential security). The problem is that the signature scheme incurs this loss as well, meaning that signatures need to be made larger than the instance length, violating succinctness.

The [WW24] SNARG. To get around this challenge, [WW24] use a trick where they actually have two valid signatures for each statement, but the program P only outputs one of them, the choice of which

[1]For example, [CLTV15] explain that rerandomizeable encryption gives lossy encryption. Rerandomizeable pseudorandom generators with sufficient stretch are also give simple constructions of lossy functions.

signature being pseudorandomly determined based on the instance. The authors first argue that, for false statements, the signature produced by any computationally bounded prover must, with non-negligible probability, be the signature that P does not produce. This step requires an exponential number of hybrids, one per false statement. This in turn means making subexponential assumptions and setting certain security parameters to be large. But importantly, the security parameter that effects the signature length is not used in this step of the proof, meaning the signature length can still be small.

We now need a way to argue that *any* signature not produced by P is hard to compute. [WW24] accomplish this by embedding a single one-way function challenge into every possible signature not produced by P. It is important that a single challenge is embedded into all the signatures, since we want any possible signature to allow us to break the one-way function. The primary obstacle is that embedding the same challenge into many different signatures would seem to make the signatures correlated. In contrast, the signatures in the [WW24] construction are essentially uncorrelated[2]. To get around this issue, the authors assume a rerandomizeable one-way function. They then, for each signature, rerandomize the original one-way function and embed the now-independent challenge into the signature, maintaining the uncorrelated structure of the signatures. This embedding step also requires an exponential number of hybrid steps. However, as long as the rerandomization is perfect (or very near perfect), it does not affect the security parameter of the one-way function, allowing signatures to remain small. After this embedding is achieved, any attacker that signs a false statement leads directly to an inversion of the one-way function.

Our Idea. Abstractly, rerandomization is used in order to embed a single challenge into an exponential number of signatures without introducing correlations. This is required since the original construction has uncorrelated signatures.

Our insight is that we can, instead of preserving uncorrelated signatures, try to move to a hybrid where the signatures are very correlated. Once the signatures are corerlated, it may be possible to embed one or several challenges into all the signatures without needing the challenges to be rerandomizeable.

Concretely, we will assume a lossy function. Recall that a lossy function [PW08] is a function that comes in two modes: an injective mode and a lossy mode. The injective mode is injective, while the lossy mode has a number of outputs that is much smaller than the number of inputs. Despite being very different functions, the two modes are required to be computationally indistinguishable. Note that lossy functions are typically assumed to have a trapdoor in the inejctive mode to allow inversion; we will not need such a trapdoor.

Suppose we first sample an injective mode function f, and then have the signature on an instance x be derived from $f(x)$. In the injective mode, each x maps to a unique $f(x)$, so the signatures remain uncorrelated. However, we now switch to f being lossy. Now many x map to the same value $f(x)$. Since signatures are derived from $f(x)$, the signatures are now correlated.

Remark 1.1. Note that we only derived signatures from $f(x)$ in the proof: the construction remains unchanged, but we perform a hybrid argument where we switch from uncorrelated signatures being derived from x to uncorrelated signatures being derived from $f(x)$ when f is injective. This step requires making subexponential hardness assumptions, but importantly this does not affect the security parameter of the lossy function or the length of the signatures.

Let 2^p be the number of possible values of $f(x)$. We will now embed an independent challenge into each of the 2^p possible signatures derived from $f(x)$. We then know that an adversary which breaks the

[2]They are the outputs a pseudorandom function (PRF). They are certainly correlated through having a common PRF key. However, aside from being generated pseudorandomly from a common PRF key, there are no correlations. Embedding a common challenge into all the signatures would seem to require *even more* correlations.

SNARG must break *one* of the 2^p challenges, but we don't know which until the end of the experiment. We therefore simply guess which one will be broken, incurring a 2^p loss. This requires subexponential assusmptions and the security showing up in the signature to grow with p. But importantly, p is a fixed polynomial independent of the instance size, meaning we maintain succinctness.

Executing the above blueprint requires a careful hybrid argument, which we show how to perform in the body. In particular we wish to avoid any additional assumptions beyond a bound on the image size of the lossy function, such as regularity or ability to sample from the images. At some hybrid we will actually run the forgeability game on the attacker twice where in the first run the attacker wins on some statement x'. Then on the next run we use the same lossy function and make the guess on the output $f(x')$.

Remark 1.2. One technical issue we encounter is the following. We need to embed our challenge into a random choice among the 2^p challenges that the adversary may break. This requires the ability to sample uniformly from among the 2^p possible outputs of the lossy-mode lossy function. While we can certainly sample from the image space by just evaluating on a random input, this distribution may be far from uniform. The adversary could in principle always solve instances that we "miss" by this simple sampling procedure, which would break the reduction. We show that it is possible to get around this issue by running the adversary *twice*, once to learn the relevant image, and once again to actually embed the challenge in that image. With this process, the reduction loss is equal to the collision probability of the adversary's distribution, which is always bounded by 2^{-p}. We note that similar issues have come up in applications of Extremely Lossy Functions (ELFs) [Zha16], leading works in this space (e.g. [Zha16, Zha19, ACH20, AWZ23a]) to assume a strong regularity condition on the ELFs, which in particular implies that the image of uniform inputs is uniform over the images. Our technique also gets around this issue, meaning that we can remove the regularity requirement from prior works. While the only known ELFs are regular, our technique may be used if other ELFs are discovered that are not guaranteed to be regular.

Lossy Functions from LWE. We now turn to instantiating the needed lossy functions. DDH-based lossy functions can readily be adapted to give the level of lossiness we need, though this will not result in an improvement to [WW24] which could be based on discrete logarithms. Instead, here we construct lossy functions from LWE. Lossy functions from LWE are already known [PW08, AKPW13, DGI+19, HHK+24], but they do not have the needed lossiness, as we will now explain.

We start from the construction of lossy functions from LWE due to [AKPW13]. The injective mode is described by a uniformly chosen tall skinny matrix $A \in \mathbb{Z}_q^{m\times\ell}$ where ℓ is the desired input length. To evaluate the function on an input $x \in \{0,1\}^\ell$, we compute $A \cdot x \bmod q$, and then round the output to an appropriately coarse rounding.

In the lossy mode, we switch to sampling A as $A = A_0 \cdot A_1 + E$ where A_0, A_1 are random matrices except that A_0 only has λ columns while A_1 has λ rows. Here, E is a matrix with small entries sampled from a discrete Gaussian.

Indisitnguishability of modes follows from the LWE assumption. Injectivity (with high probability) follows from simple statistical arguments and the fact that A is a tall skinny matrix.

Lossiness follows from the following argument. Since $x \in \{0,1\}^\ell$ is short (its entries are only in 0/1), we have that $A \cdot x = A_0 \cdot A_1 \cdot x + E \cdot x$. Here, $E \cdot x$ is short, and hopefully gets rounded away by the rounding, meaning the function is only a function of $A_0 \cdot A_1 \cdot x$. But as there are only q^λ values for $A_1 \cdot x$, the total number of images is small. In particular, we will typically have q bounded by 2^λ, in which case the number of images is bounded by 2^{λ^2}, independent of the input length.

The above works, except that for some x, $A_0 \cdot A_1 \cdot x$ will be close to a rounding boundary, meaning the error term $E \cdot x$ can actually change the outcome of the rounded value. We can ensure that "most" x

end up far from a rounding boundary by setting q to be very large. But even if the fraction of x that have components close to the rounding boundary is small, the number of them is very large. The issue is that we can no longer say that there are only q^λ possible outputs. Even with a negligibly fraction of x being close to a rounding boundary, since there are 2^ℓ different x, we will still have $2^\ell \times \mathsf{negl} \gg q^\lambda$ different outputs. While this amount of lossiness is enough for many of the applications considered in [PW08], this will not be lossy enough for our proof. This is because, as discussed above, our proof size grows with the (logarithm of the) number of possible outputs in the lossy mode. With all known LWE-based lossy functions, the proof size will end up being at least $\Theta(\ell)$, which is not succinct. We therefore need a vastly more lossy function.

Our insight is to observe that we can actually tell (with some false positives) when $A_0 \cdot A_1 \cdot x$ will be close to a rounding boundary: namely, when $A \cdot x$ is close to a rounding boundary. Thus, if we exclude all x such that $A \cdot x$ is close to a rounding boundary, then the lossy mode function actually will have the desired image size q^λ which is vastly less than 2^ℓ.

In order to accommodate all x as inputs, we simply sample many A. To evaluate, iterate through the different A, finding the first one where $A \cdot x$ is far from a rounding boundary. For that function, output the rounding of $A \cdot x$. A union-bound argument shows that, with high probability, there will exist *some* A for each x, meaning the function is well-defined. This slightly blows up the image size, but since the number of A is polynomial, the image size stays very small.

There are a handful of subtle issues that we need to take care of to get the lossy function to be sufficiently lossy for our proof. In particular, the injective mode actually needs to be injective with certainty. This is because in the steps where we rely on injectivity, we step through 2^ℓ hybrids, and each one (at least if done straightforwardly) would incur a statistical loss if there was some probability of being non-inejective. Thus, the statistical loss gets blown up by 2^ℓ.

We make sure the inejctive mode is actually injective by ensuring we sample A in a way that the lossy function can actually be inverted. This uses techniques from the literature for generating lattice trapdoors. These trapdoors actually turn our lossy function into a lossy *trapdoor* functions with a large lossiness. While we do not actually need the trapdoor functionality in this work (beyond using it to justify injectivity), lossy trapdoor functions in general have many applications such as CCA-security [PW08].

1.3 Other Related Work

Other SNARG Constructions. Aside from the random-oracle constructions of SNARGs [Kil92, Mic94], there are a number of constructions based on non-falsifiable knowledge assumptions, e.g. [Gro10, BCCT12, DFH12, Lip13, GGPR13, BCI+13, BCC+17, ACL+22, CLM23]. There are also SNARGs for subsets of NP, e.g. [KR09, KP16, BHK17, JKKZ21]. The only known SNARGs for all of NP from falsifiable assumptions are those based on iO, namely [SW14] in the static setting, and the very recent work of [WW24] in the adaptive setting.

Lossiness and Obfuscation. Lossy functions were originally introduced by [PW08]. The variant usually considered in the literature additionally has a trapdoor int he injective mode. This variant has many applications, such as CCA-secure encryption. More recently, lossy functions without a trapdoor have been used in conjunction with iO in several contexts [Zha16, ACH20, AWZ23b].

2 Preliminaries

In this section we give the preliminaries for our work. All of our components besides our notion of length parameterized lossy functions are the same as that of Waters and Wu[WW24]. In order to facilitate consistency between the works we take from their material[WW24] the definitions provided in this section.

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

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

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

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

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

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

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

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

  in the program indistinguishability game defined above.

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

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

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

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

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

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

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

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

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

  in the punctured pseudorandomness security game.

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

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

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

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

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

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

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

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

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

- **Completeness:** For all security parameters $\lambda \in \mathbb{N}$, all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all instances (x, w) where $C(x, w) = 1$,
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, C) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array}\right] = 1.$$

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

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

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

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

We next define one way functions. Intuitively our definition is simply a one way function where there is first a setup algorithm producing a CRS. In order for our construction to notationally match [WW24] we retain their notation. This includes the somewhat atypical notion of a verify algorithm.

Definition 2.7 (One-Way Functions with Setup)**.** A rerandomizable one-way function is a tuple of efficient algorithms $\Pi_{\mathsf{OWF}} = (\mathsf{Setup}, \mathsf{GenInstance}, \mathsf{Verify})$ with the following syntax:

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

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

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

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

- **Correctness:** For all $\lambda, m \in \mathbb{N}$, it holds that

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, y, z) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^m) \\ (y, z) \leftarrow \mathsf{GenInstance}(\mathsf{crs}) \end{array}\right] = 1.$$

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

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

$$\mathsf{OWFAdv}_{\mathcal{A}}(\lambda) := \Pr[b = 1] \leq \varepsilon(\lambda)$$

  in the one-wayness game.

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

3 Length Parameterized Lossy Functions

In the section we give the definition of length parameterized lossy functions and follow it with a construction from the LWE assumption. Length parameterized lossy functions follow the concept of lossy functions introduced in [PW08] with the distinction that the setup algorithm takes as input a security parameter λ along with an explicit input length parameter ℓ_{in}. As in [PW08] there is both an injective and a lossy mode of setup and these should be computationally indistinguishable. However, image size of the function in lossy mode should be a polynomial that depends *only* on the security parameter λ and is independent of ℓ_{in}.

Definition 3.1 (Length Parameterized Lossy Functions)**.** A Length Parameterized Lossy Function a tuple of efficient algorithms $\Pi_{\mathsf{LossyF}} = (\mathsf{SetupInj}, \mathsf{SetupLossy}, \mathsf{Eval})$ with the following syntax:

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

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

- $\mathsf{Eval}(k, x) \to y$: On input a key k and an input $x \in \{0, 1\}^{\ell_{\mathsf{in}}}$, the evaluation algorithm outputs a value $y \in \{0, 1\}^{\ell_{\mathsf{out}}}$, for some output length ℓ_{out} that is determined during function setup.

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

- **Injectivity in Injective Mode:** For all $\lambda, \ell_{\mathsf{in}} \in \mathbb{N}$, every pair of inputs $x_0, x_1 \in \{0,1\}^{\ell_{\mathsf{in}}}$ if $k \leftarrow \mathsf{SetupInj}(1^\lambda, 1^{\ell_{\mathsf{in}}})$ and $\mathsf{Eval}(k, x_0) = \mathsf{Eval}(k, x_1)$, then $x_0 = x_1$.

- **Lossiness in Lossy Mode:** There exists a (single variate) polynomial p such that for all $\lambda, \ell_{\mathsf{in}} \in \mathbb{N}$ the following holds. Let $k \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^{\ell_{\mathsf{in}}})$ then define S_k to be the set where $y \in \{0,1\}^{\ell_{\mathsf{out}}} \in S_k$ if and only if there exists an $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ where $\mathsf{Eval}(k, x) = y$. (I.e. S_k is the image of the function $\mathsf{Eval}(k, \cdot)$.) Then $|S_k| \le 2^{p(\lambda)}$. Thus, the image size is bounded by a polynomial in λ which is independent of ℓ_{in}.

  Remark 3.2. As a technicality the image size is not necessarily independent of ℓ_{in}, it is there exists a *bound* on it that is independent of ℓ_{in}. In addition, it might be the case that for certain combinations of $\lambda, \ell_{\mathsf{in}}$ that in lossy mode the function does not loose information on the input. The definition only guarantees lossiness when $\ell_{\mathsf{in}} > p(\lambda)$. Nonetheless, this is what we will need for our applications.

- **Mode indistinguishability:** For a bit $b \in \{0,1\}$ and a security parameter λ, we define the mode indistinguishability game between an adversary $\mathcal{A}$ and a challenger as follows:

  – On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the input length $1^{\ell_{\mathsf{in}}}$.
  – If $b = 0$, the challenger gives $k \leftarrow \mathsf{SetupInj}(1^\lambda, 1^{\ell_{\mathsf{in}}})$ to $\mathcal{A}$. If $b = 1$, then it gives $k \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^{\ell_{\mathsf{in}}})$ to $\mathcal{A}$.
  – At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{LossyF} satisfies mode indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, it holds that

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

  in the mode indistinguishability security game.

We will also consider a slightly weaker version of lossiness, which we call **Almost Perfect Lossiness**. This states that the lossy condition only holds with probability at least $1 - \mathsf{negl}(\lambda)$ over the choice of lossy key $k \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^{\ell_{\mathsf{in}}})$.

Remark 3.3. Suppose we have a length parameterized lossy function system where $|S_k|$ is bounded by $2^{q(\lambda, \lg(\ell_{\mathsf{in}}))}$ for some bivariate polynomial q. Then we have a length parameterized system that meets our single variate condition above. Consider the transformation (which is considered folklore in many other settings). If $\ell_{\mathsf{in}} > 2^\lambda$, then go into a degenerative setup where $\mathsf{SetupInj}(1^\lambda, 1^{\ell_{\mathsf{in}}})$ produces a key where evaluation will be the identity function. And $\mathsf{SetupLossy}(1^\lambda, 1^{\ell_{\mathsf{in}}})$ produces a key where evaluation will always output an all 0's string. Whenever the degenerative condition is triggered we will clearly have the lossiness and injective conditions. Whenever it isn't we can bound $q(\lambda, \lg(\ell_{\mathsf{in}}))$ by $q(\lambda, \lambda)$ which gives us a single variate polynomial in λ. Moreover, adding this condition does not impact mode indistinguishability security as any polynomial time adversary will only be capable of triggering it for a finite number of security parameters.

In the next subsections, we will construct lossy functions satisfying Definition 3.1, namely the variant with *almost perfect lossiness*. Our construction will work in two parts. First, we will explain how to realize Definition 3.1 from a weaker notion of lossiness. Then we will show a simple way to achieve this weaker lossiness directly from LWE.

3.1 Boosting Lossiness

Approximate Lossiness. Consider the following *approximate lossiness* condition, which relaxes "Lossiness in Lossy Mode" in Definition 3.1:

- There is a deterministic procedure $\mathsf{VerLossy}(k, x)$ which outputs either 1 or 0.
- For any $\lambda, \ell_{\mathsf{in}}$ and $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, $\Pr[\mathsf{VerLossy}(\mathsf{SetupLossy}(1^\lambda, 1^{\ell_{\mathsf{in}}}), x) = 1] \geq 1/2$. Here, the constant $1/2$ is arbitrary.
- Let S_k be the set of images of $\mathsf{Eval}(k, \cdot)$ *restricted to the points* x *such that* $\mathsf{VerLossy}(k, x) = 1$. Then we require that $|S_k| \leq 2^{p(\lambda)}$.

In other words, there is a reasonably dense recognizable set (those x such that $\mathsf{VerLossy}(k, x) = 1$) such that on this recognizable set, $\mathsf{Eval}(k, \cdot)$ achieves the desired lossiness. We allow for the overall function, however, to be not lossy once x such that $\mathsf{VerLossy}(k, x) = 0$ are taken into account. Even if strengthened the conditions so that x constituted a negligible fraction of the domain, such functions would only be lossy in a very weak sense, losing only a few bits of information about x. But we desire very strong lossiness where the all *but* a few bits of information are lost, and specifically the amount of information left is essentially independent of the input size. Note that standard lossiness simply has VerLossy accept all inputs.

Lemma 3.4. *If there exists* $\Pi_{\mathsf{LossyF}} = (\mathsf{SetupInj}, \mathsf{SetupLossy}, \mathsf{Eval}, \mathsf{VerLossy})$ *satisfying Definitino 3.1 except with approximate lossiness, then there exists a protocol* $\Pi'_{\mathsf{LossyF}} = (\mathsf{SetupInj}', \mathsf{SetupLossy}', \mathsf{Eval}')$ *satisfying Definition 3.1 with almost perfect lossiness.*

Proof. $\Pi'_{\mathsf{LossyF}} = (\mathsf{SetupInj}', \mathsf{SetupLossy}', \mathsf{Eval}')$ works as follows:

- $\mathsf{SetupInj}'(1^\lambda, 1^{\ell_{\mathsf{in}}})$: Let $r = \ell_{\mathsf{in}} + \lambda$. For $i \in [r]$, sample $k_i \leftarrow \mathsf{SetupInj}(1^\lambda, 1^{\ell_{\mathsf{in}}})$. Output $k = (k_i)_{i \in [r]}$.
- $\mathsf{SetupLossy}'(1^\lambda, 1^{\ell_{\mathsf{in}}})$: Let $r = \ell_{\mathsf{in}} + \lambda$. For $i \in [r]$, sample $k_i \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^{\ell_{\mathsf{in}}})$. Output $k = (k_i)_{i \in [r]}$.
- $\mathsf{Eval}'(k, x)$: parse $k = (k_1, \cdots, k_r)$. Let i be the smallest value in $[r]$ such that $\mathsf{VerLossy}(k_i, x) = 1$; if no such value exists, let $i = 0$. If $i \neq 0$, let $y_i \leftarrow \mathsf{Eval}(k, x)$; if $i = 0$, let $y_0 = x$, appropriately padded to match the output length of Eval. Output (i, y_i).

Indistinguishability of injective and lossy modes for Π'_{LossyF} follows from the indistinguishability for Π_{LossyF} by a simple hybrid over the keys.

In the injective mode suppose there was a collision $\mathsf{Eval}'(k, x) = \mathsf{Eval}'(k, x') = (i, y_i)$ for $x \neq x'$. This in particular means that $\mathsf{Eval}(k_i, x) = \mathsf{Eval}(k_i, x') = y_i$, contradicting the (perfect) injectiveness of the key k_i. This we see that the injective mode for Π'_{LossyF} is indeed injective. We note that in the case that there is no i such that $\mathsf{VerLossy}(k_i, x) = 1$ the output of setting $y_0 = x$ is trivially injective.

For the lossy mode, let $p' = p + \log_2 r$, which is a polynomial in λ under the assumption that $r \leq 2^\lambda + \lambda$, or equivalently, that $\ell_{\mathsf{in}} \leq 2^\lambda$.

Let S_k denote the set of images (i, y_i) of Eval' with $i \neq 0$. Observe that since such values are only outputted on inputs x such that $\mathsf{VerLossy}(k_i, x) = 1$, we must have that S_k is the union of $\{i\} \times S_{k_i}$. Each S_{k_i} has size at most 2^p, meaning that S_k has size at most $2^p \times r = 2^{p + \log_2(r)} = 2^{p'}$. It therefore suffices to show that the image of Eval' is exactly S_k (or equivalently that there is no image of Eval' with $i = 0$) with overwhelming probability.

Toward that end, fix an x, and consider choosing a random key $k \leftarrow \mathsf{SetupLossy}'(1^\lambda, 1^{\ell_{\mathsf{in}}})$ and letting $\mathsf{Eval}'(k, x) = (i, y_i)$. If $i = 0$, we say that x is "bad." We are interested in the probability that x is bad. Having $i = 0$ means that $\mathsf{VerLossy}(k_j, x) = 0$ for all $j \in [r]$. Since the k_j are independent, and for each j we have that $\Pr[\mathsf{VerLossy}(k_j, x) = 0] \leq 1/2$, we therefore have that $\Pr[i = 0] \leq 2^{-r}$.

A union bound over all $2^{\ell_{\mathsf{in}}}$ inputs gives an overall probability of there existing *some* bad x is at most $2^{\ell_{\mathsf{in}}-r} = 2^{-\lambda}$, which is negligible, as desired. Thus we see that the lossy mode for Π'_{LossyF} is indeed almost perfectly lossy. □

3.2 Approximate Lossiness from LWE

Here, we describe how to achieve approximate lossiness from Learning With Errors (LWE).

Lattice Background. Here we recall some basic lattice background. Let χ_σ be the distribution over $\mathbb{Z}$ where $\Pr[x \leftarrow \chi_\sigma] \propto e^{2\pi x^2/\sigma^2}$. Let $\chi_{\sigma,B}$ be the bounded version of χ_σ, namely the distribution over $[-B, B]$ where $\Pr[x \leftarrow \chi_{\sigma,B}] \propto e^{2\pi x^2/\sigma^2}$.

Fact 3.5. For $\sigma \geq \omega(\log \lambda)$ and $B \geq \sigma \times \omega(\log \lambda)$, the distributions χ_σ and $\chi_{\sigma,B}$ are negligibly close in λ.

Definition 3.6. Let m, q, σ be functions in λ where m, $\log(q)$, and $\log(\sigma)$ are bounded by polynomials in λ. The (m, q, σ)-LWE assumption holds if, for any polynomial-time (in λ) adversary $\mathcal{A}$, there exists a negligible function $\epsilon = \epsilon(\lambda)$ such that $\left|\Pr[\mathcal{A}(A, v) = 1 : {\scriptstyle A \leftarrow \mathbb{Z}_q^{m\times\lambda} \atop v \leftarrow \mathbb{Z}_q^m}] - \Pr[\mathcal{A}(A, v) = 1 : \substack{A \leftarrow \mathbb{Z}_q^{m\times\lambda} \\ s \leftarrow \mathbb{Z}_q^\lambda, e \leftarrow \chi_\sigma^m \\ v \leftarrow A\cdot s+e \bmod q}]\right| \leq \epsilon(\lambda)$

We can also consider a matrix variant of LWE where instead of a vector V, the adversary is given $V \in \mathbb{Z}_q^{m\times\ell}$, where either V is uniform in $\mathbb{Z}_q^{m\times\ell}$, or $V \leftarrow A \cdot S + E \bmod q$, where S is uniform in $\mathbb{Z}_q^{\lambda\times\ell}$, and $E \leftarrow \chi_\sigma^{m\times\ell}$. This matrix version follows from the plain version by a simple hybrid argument.

Theorem 3.7 ([AP11]). *There exists a PPT algorithm* $\mathsf{TrapGen}(q, \ell, m)$, $m \geq \Omega(\ell \log q)$, *that samples a pair* (A, S) *with* $A \in \mathbb{Z}_q^{m\times\ell}$ *and* $S \in \mathbb{Z}^{m\times m}$ *such that:*

- $S \cdot A \bmod q = 0$
- *A is full rank (rank ℓ) over $\mathbb{Z}_q$ and S has rank m over $\mathbb{Z}$*
- *There is a polynomial $M(\ell, m, \log q)$ which bounds all entries in S.*
- *A is statistically close (in ℓ) to uniform over $\mathbb{Z}_q^{m\times\ell}$.*

Let q, t be integers with $t \leq q$. We associate $\mathbb{Z}_q$ with the interval $[-\lfloor(q-1)/2\rfloor, \lceil(q-1)/2\rceil]$ in the natural way. We then partition $\mathbb{Z}_q$ into t intervals $I_0 = [u_0, u_1), I_1 = [u_1, u_2), \cdots, I_{t-1} = [u_{t-1}, u_t]$ where $u_0 = -\lfloor(q-1)/2\rfloor$ and $u_1 = \lceil(q-1)/2\rceil$. We will choose the intervals so that each has size $\lfloor q/t\rfloor$ or $\lceil q/t\rceil$. We also specify a point c_i for each interval I_i, where $c_i = \lfloor(u_{i+1} + u_i)/2\rfloor$. Then define $\lfloor x\rceil_t$ to be c_i where i is such that $x \in I_i$. Observe that $|\lfloor x\rceil_t - x| \leq (q/t) + 1$

LWE implies Approximate Lossiness. We now show how to construct lossy functions from LWE by adapting existing techniques.

Lemma 3.8. *Assuming (m, q, σ)-LWE is hard for $\sigma = \omega(\log \lambda)$ and for all polynomials m, q, there exists a lossy function* $\Pi_{\mathsf{LossyF}} = (\mathsf{SetupInj}, \mathsf{SetupLossy}, \mathsf{Eval}, \mathsf{VerLossy})$ *that is approximately lossy.*

As an immediate corollary by combining with Lemma 3.4, we obtain that:

Corollary 3.9. *Under the same assumption as Lemma 3.8, there exists a length parameterized lossy function with almost perfect lossiness.*

We now prove Lemma 3.8:

Proof. Let m, t, σ, B, q be polynomials in $\lambda, \ell_{\mathsf{in}}$ satisfying:

$$\begin{aligned} m &\geq \Omega(\ell_{\mathsf{in}} \log q) \\ t &> M(\ell_{\mathsf{in}}, m, \log q) \times m \\ \sigma &\geq \omega(\log \lambda) \\ B &\geq \sigma \times \omega(\log \lambda) \\ q &\geq 3t \times (2\ell_{\mathsf{in}} B + 3) \end{aligned}$$

For example, set $m = \Theta(\ell_{\mathsf{in}} \times \lambda)$, $t = \Theta(M(\ell_{\mathsf{in}}, m, \lambda) \times m)$, $\sigma = \lambda$, $B = \sigma \times \lambda$, and $q = \Theta(t\ell_{\mathsf{in}} B)$. We then observe that $\log q \leq \lambda$, and so all the desired inequalities hold.

Now Π_{LossyF} will work as follows:

- $\mathsf{SetupInj}(1^\lambda, 1^{\ell_{\mathsf{in}}})$: Sample a pair $(A, S) \leftarrow \mathsf{TrapGen}(q, \ell_{\mathsf{in}}, m)$. Output $k = A$.
- $\mathsf{SetupLossy}(1^\lambda, 1^{\ell_{\mathsf{in}}})$: Sample uniform matrices $A_0 \in \mathbb{Z}_q^{m \times \lambda}$ and $A_1 \in \mathbb{Z}_q^{\lambda \times \ell_{\mathsf{in}}}$. Also sample $E \leftarrow \chi_{\sigma,B}^{m \times \ell_{\mathsf{in}}}$. Let $A = A_0 \cdot A_1 + E \in \mathbb{Z}_q^{m \times \ell_{\mathsf{in}}}$. Output $k = A$
- $\mathsf{Eval}(k, x)$: interpret k as A. Let t be a polynomial to be discussed later. Output $y = \lfloor A.x \rceil_t$.
- $\mathsf{VerLossy}(k, x)$: interpret k as A. Output 0 if there exists a i, j such that $|(A \cdot x)_j - u_i \bmod q|_\infty \leq B \times \ell_{\mathsf{in}} + 1$. If no such i, j exists, output 1.

We now prove that Π_{LossyF} is actually an approximately lossy function.

Indistinguishability of Modes. We consider four hybrid distributions over q, m, A: Hyb_0 is the case where A is generated from $\mathsf{TrapGen}(q, \ell_{\mathsf{in}}, m)$. Hyb_1 is the case where A is uniform in $\mathbb{Z}_q^{m \times \ell_{\mathsf{in}}}$. Hyb_2 is the case where $A = A_0 \cdot A_1 + E$ where A_0, A_1 are uniform in $\mathbb{Z}_q^{m \times p}$ and $\mathbb{Z}_q^{p \times \ell_in}$, respectively, and $E \leftarrow \chi_\sigma^{m \times \ell_{\mathsf{in}}}$, the unbounded discrete Gaussian. Finally Hyb_3 is the case where $A = A_0 \cdot A_1 + E$ and $E \leftarrow \chi_{\sigma,B}^{m \times \ell_{\mathsf{in}}}$, the bounded discrete Gaussian. Our goal is to prove the indistinguishability of Hyb_0 (the injective mode) and Hyb_3 (the lossy mode).

We first see that the distinguishing advantage between Hyb_0 and Hyb_1 bounded by Theorem 3.7 to be negligible in ℓ_{in}, which is also negligible in λ. We then see that Hyb_1 and Hyb_2 are exactly the cases in the matrix version of LWE, with λ for the LWE assumption matching λ for Π_{LossyF}. Thus by the LWE assumption, the distinguishing advantage between Hyb_1 and Hyb_2 is negligible in λ. Next, the difference between Hyb_2 and Hyb_3 is simply whether or not E is sampled from χ_σ or $\chi_{\sigma,B}$. By our assumption that σ and B/σ are $\omega(\log \lambda)$, these distributions are negligibly close. This shows the indistinguishability of Hyb_0 and Hyb_3, as desired.

Injectivity. We explain that, information-theoretically, it is possible to recover x from $\mathsf{Eval}(k, x)$ when x is sampled in the injective mode. This implies injectivity.

To recover x, we assume we have knowledge of S that was sampled with A by $\mathsf{TrapGen}$. Then given $y = \mathsf{Eval}(k, x)$, we compute $z = y - S^{-1} \cdot (S \cdot y \bmod q)$. Here, S^{-1} is the inverse over the integers. We now prove that $z = A \cdot x$.

First, we observe that $y = \lfloor A \cdot x \rceil_t = A \cdot x + e$ where $e \in [-(q/t + 1), q/t + 1]$. Then $S \cdot y \bmod q = S \cdot A \cdot x + S \cdot e \bmod q = S \cdot e \bmod q$. The entries in S are bounded by $M = M(\ell_{\mathsf{in}}, m, \log q)$. Therefore, the entries in $S \cdot e$ are bounded by $M \times m \times (q/t + 1) < q/2$. As such, $S \cdot e \bmod q = S \cdot e$, where the right-hand side is not reduced mod q. We thus see that $z = y - S^{-1} \cdot (S \cdot y \bmod q) = y - S^{-1} \cdot S \cdot e = y - e = A \cdot x$. Since A is full rank, we know that given $z = A \cdot x$ we can recover x. Thus, injectivity holds.

Lossiness. Consider a lossy mode key k containing matrix $A = A_0 \cdot A_1 + E$ where $E \leftarrow \chi_{\sigma,B}^{m \times \ell_{\mathsf{in}}}$. Let S_k be the set of images of $\mathsf{Eval}(k, \cdot)$ restricted to x such that $\mathsf{VerLossy}(k, x) = 1$. In other words, $|(A \cdot x)_j - u_i \bmod q| > B \times \ell_{\mathsf{in}} + 1$ for all i, j. Let $\mathsf{Eval}'(x)$ denote $\lfloor A_0 \cdot A_1 \cdot x \rceil_t$.

Observe that since $A_1 \in \mathbb{Z}_q^{\lambda \times \ell_{\mathsf{in}}}$ and $\mathsf{Eval}'(x)$ is a function of $A_1 \cdot x$, its image size is only $q^\lambda = 2^{\lambda \log_2 q} \leq 2^{\lambda^2}$, where we use that $\log_2 q \leq \lambda$. Thus setting $p(\lambda) = \lambda^2$ a fixed polynomial in λ independent of ℓ_{in}, we have that the image of Eval' has size at most $2^{p(\lambda)}$.

We now claim that $\mathsf{Eval}'(x) = \mathsf{Eval}(k, x)$ for all $x \in S_k$, and thus that the images of $x \in S_k$ have size at most $2^{p(\lambda)}$. Indeed, since $x \in S_k$, we have that all coordinates of $A \cdot x$ are at least $B \times \ell_{\mathsf{in}} + 1$ far from any u_i. Meanwhile, $A_0 \cdot A_1 \cdot x = A \cdot x - E \cdot x$ and since the entries of E are bounded by B, the entries of $E \cdot x$ are bounded by $\ell_{\mathsf{in}} \times B$. This means that $A_0 \cdot A_1 \cdot x$ lies in the same interval I_i as $A \cdot x$ for every coordinate, and thus the rounding function $\lfloor \cdot \rceil_t$ gives the same result on both vectors.

Finally, we want that for any given x, the probability over k that $\mathsf{VerLossy}(k, x) = 1$ is at least $1/2$. Fix E. We first observe that $A_1 \cdot x = 0$ with probability $q^{-\lambda}$, which is negligible in λ. We therefore fix an A_1 such that $A_1 \cdot x \neq 0$. Let $r = A_1 \cdot x$. Now consider sampling a uniform A_0. For our fixed x, A_1, E, each entry of $A \cdot x = A_0 \cdot r + E \cdot x$ is therefore a uniform random value in $\mathbb{Z}_q$. The number of points in $\mathbb{Z}_p$ that are within $\ell_{\mathsf{in}} \times B + 1$ of one of the u_i is $t \times (2\ell_{\mathsf{in}} B + 3)$ [3]. Thus, for this fixed x, A_1, E, the probability over A_0 that $\mathsf{VerLossy}(k, x) = 1$ is at least $1 - t \times (2\ell_{\mathsf{in}} B + 3)/q \geq 2/3$. Then the overall probability $\mathsf{VerLossy}(k, x) = 1$ is at least $2/3 - \mathsf{negl} \geq 1/2$. □

3.3 Extension to Lossy Trapdoor Functions

The lossy functions originally proposed by [PW08] were lossy *trapdoor* functions (LTDFs), where the injective mode comes with a secret trapdoor that allows for inverting the function. Such LTDFs have numerous additional applications, such as CCS-secure public key encryption.

Here, we briefly explain how to extend the definition of length parameterized lossy functions and our construct to have a trapdoor.

Definition 3.10 (Length Parameterized LTDFs)**.** A Length Parameterized Lossy Trapdoor Function (LTDF) is an ordinary Length Parameterized lossy function, except that:

- $\mathsf{SetupInj}(1^\lambda, 1^{\ell_{\mathsf{in}}})$ outputs a pair (t, k) instead of just k
- There is an additional function $\mathsf{Invert}(t, y) \rightarrow x$

[3]Observe that even though there are $t + 1$ of the u_i, we only care about half of the points near u_0 and u_t, since they are on the edges of $\mathbb{Z}_q = [-\lfloor (q-1)/2 \rfloor, \lceil (q-1)/2 \rceil]$.

- **Correct Inversion:** For any $\lambda, \ell_{\mathsf{in}}$, for any $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ and any (t, k) produced by $\mathsf{SetupInj}(1^\lambda, 1^{\ell_{\mathsf{in}}})$, $\mathsf{Invert}(t, \mathsf{Eval}(k, x)) = x$.

We can anlogously define the notion of an approximately lossy length parameterized LTDF.

Adding a trapdoor to Lemma 3.4. We see that Lemma 3.4 will also lift an approximately lossy LTDF to an LTDF with almost perfect lossiness. The construction is the same, except that $\mathsf{SetupInj}$ samples $(t_i, k_i) \leftarrow \mathsf{SetupInj}(1^\lambda, 1^{\ell_{\mathsf{in}}})$ and outputs $t = (t_i)_{i\in[r]}$ in addition to $k = (k_i)_{i\in[r]}$. Let Invert be the inversion algorithm for Π_{LossyF}. Then $\mathsf{Invert}'(t, y)$ works as follows:

- Parse t as $(t_i)_{i\in[r]}$. Parse y as (i, y_i).
- If $i = 0$, output y_0.
- Otherwise, output $x \leftarrow \mathsf{Invert}(t_i, y_i)$

Correctness of inversion follows immediately from the correctness of inversion of Π_{LossyF}. Security is identical.

Approximately Lossy LTDFs from LWE. We now explain that our construction of an approximately lossy LTDF in Lemma 3.8 actually already has a trapdoor. In particular, we set $t = S$. In order to prove injectivity in Lemma 3.8, we actually proved that knowledge of S allows for efficient inversion. This efficient inversion now becomes the algorithm Invert. As a result, we obtain the following:

Theorem 3.11. *Assuming (m, q, σ)-LWE is hard for $\sigma = \omega(\log\lambda)$ and for all polynomials m, q, there exists a LTDF $\Pi_{\mathsf{LossyF}} = (\mathsf{SetupInj}, \mathsf{SetupLossy}, \mathsf{Eval}, \mathsf{Invert})$ that is almost perfectly lossy.*

4 The Waters-Wu [WW24] SNARG

The novelty in our work is to present a novel security analysis of the Waters and Wu [WW24] SNARG construction. Our analysis follows a lossy function paradigm that does not require a one way function that is perfectly rerandomizable; thus untethering us from the discrete log or other non post quantum assumptions. Since the novelty of our work is centered in the analysis of the construction we will provide the description of the Waters and Wu construction below and intentionally take care to follow their presentation style and description as closely as possible. The construction as presented will have the following minor modifications:

- Unlike Waters and Wu [WW24] we do *not* require the one way function will to be re-randomizable.
- We will require the one way function to have sub-exponential hardness. We note that [WW24] (as well as this work) require puncturable PRFs subexponential hardness. Since these imply one way functions with subexponential hardness, this won't add additional assumptions to our theorem.
- The size padding for obfuscation will depend on the programs which are obfuscated as part of our security analysis.
- While we do employ length parameterized lossy functions for our analysis, these are not used in the construction itself other than impacting the aforementioned size padding in one way functions. However, we list Lossy Functions as one of the primitives so it can be referenced later in the analysis.

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

- Let iO be a indistinguishability obfuscator for Boolean circuits.
- Let $\Pi_{\mathsf{OWF}} = (\mathsf{OWF.Setup}, \mathsf{OWF.GenInstance}, \mathsf{OWF.Verify},)$ be a one-way function with setup.
- Let $\Pi_{\mathsf{PPRF}} = (\mathsf{F.KeyGen}, \mathsf{F.Eval}, \mathsf{F.Puncture})$ be a puncturable PRF. For a key k and an input x, we will write $\mathsf{F}(k, x)$ to denote $\mathsf{F.Eval}(k, x)$.
- Let $\Pi_{\mathsf{LossyF}} = (\mathsf{SetupInj}, \mathsf{SetupLossy}, \mathsf{Eval})$ be a length parameterized lossy function with function output length $\ell_{\mathsf{out}}(1^{\lambda}, 1^{\ell_{\mathsf{in}}})$ for security parameter λ and input length ℓ_{in}.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Theorem 4.2 (Completeness [WW24])**.** *If $i\mathcal{O}$ and Π_{OWF} are correct, then Section 4 is complete.*

Correctness follows exactly as the argument of correctness from [WW24]. As stated above the only differences are the amount of $i\mathcal{O}$ padding which will not impact correctness along with the fact that we employ a OWF that is not necessarily re-randomizable. However, the re-randomization property is not employed in the completeness proof of [WW24].

5 Adaptive Proof of Security

In this section we provide our security proof which proceeds via a sequence of hybrid games. The first two hybrid transitions establish that if an attacker wins in the game with non-negligible probability, then it will win with an "off-path" proof with non-negligible probability. By off path we mean that it produces a valid proof for statement $\pi = (b, z)$ for statement x where $b \neq \mathsf{F}(k_{\mathsf{sel}}, x)$. This argument follows identically to [WW22] and we are able to cite their lemmas.

Next, we wish to plant challenges at all 2^n off path locations. We do this in Hyb_3 by first sampling a lossy function key in injective mode as $\mathsf{SetupInj}(1^\lambda, 1^n) \to k_{\mathsf{lf}}$. Then in a modified $\mathsf{GenInst}_1$ we add the computation: compute $(y_{1-b}, z_{1-b}) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x)))$ where k_{replace} is a new puncturable PRF key which is introduced for this planting step. We insert this change on each of the 2^n inputs one at a time using punctured programs techniques. Our ability to execute the substitutions relies critically on the fact that the lossy function is in injective mode at this juncture.

After this planting is complete we can switch the lossy function to lossy mode by sampling $k_{\mathsf{lf}} \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^n)$. At this point there are at most $2^{p(\lambda)}$ output values of $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x)$ and correspondingly at most $2^{p(\lambda)}$ output of y_{1-b} from $\mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x)))$.

Ideally, we could take a one way function challenge y^* and embed it randomly at one of these possible outputs in order to get a one way function attacker with $2^{p(\lambda)}$ loss which we could absorb via subexponential hardness. However, this argument implicitly imposes an additional property on our hash function that we can sample uniformly among the possible outputs in lossy mode.

We avoid introducing such an assumption with the following technique which is captured in Hyb_6. Here we first run the security experiment and obtain a proof on statement x' from the attacker. Then we re-run the experiment, but with the same lossy key k_{lf}. In this second time we consider the attacker to only win if it proves on statement x where $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x') = \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x)$. We show this incurs a loss proportional to $2^{p(\lambda)}$.

In this way we will let the attacker itself guide us to where to embed the one way function challenge y^*. In subsequent experiments we embed this one way function challenge in the second run of the game.

Theorem 5.1 (Adaptive Soundness). *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure, Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security, Π_{OWF} is $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{owf}}}})$-secure and that Π_{LossyF} is mode indistinguishability secure for constants $\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}}, \varepsilon_{\mathsf{owf}} \in (0, 1)$.*

In addition, suppose $i\mathcal{O}$ is correct, Π_{PPRF} satisfies punctured correctness, and Π_{LossyF} satisfies the injectivity in injective mode and with all but negligible probability image sizes are bound by $2^{p(\lambda)}$ for some polynomial $p(\cdot)$ in lossy mode. Let $\lambda_{\mathsf{obf}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{obf}}}$, $\lambda_{\mathsf{PRF}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{PRF}}}$, and $\lambda_{\mathsf{owf}} = (\lambda + p(\lambda))^{\varepsilon_{\mathsf{owf}}}$. Then, Section 4 is adaptively sound.

We begin our proof by defining a sequence of hybrid experiments beginning with the actual security game. Let $\mathcal{A}$ be an efficient adversary for the adaptive soundness game for Section 4. [4] We define a sequence of hybrid games:

- Hyb_0: This is the real adaptive soundness experiment. Namely, the adversary starts by outputting a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$. The challenger then constructs the CRS as follows:
  - Sample $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$.
  - Sample PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$ and $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$.
  - The challenger then constructs $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}[C, \mathsf{crs}_{\mathsf{OWF}}, k_0, k_1])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}$ on the programs from Figs. 1 and 2, and s is the same size parameter from Section 4.

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

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

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

- Hyb_3: Same as Hyb_2, except the challenger changes how it constructs $\mathsf{ObfVerify}$. During setup, the challenger now does the following:
  - Sample a lossy function key in injective mode as $\mathsf{SetupInj}(1^\lambda, 1^n) \to k_{\mathsf{lf}}$. And let $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(1^\lambda, 1^n)$ denote the output length associated with the keyed function.

[4] As in [WW24] we will assume without loss of generality that for each security parameter λ, algorithm $\mathcal{A}$ always outputs a Boolean circuit C with statements of length $n = n(\lambda)$, for some fixed (and known) polynomial n.

– Sample a PRF key $k_{\text{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\text{PRF}}}, 1^{\ell_{\text{out}}}, 1^{\rho})$. Define the following program $\mathsf{GenInst}_1$:

> **Input:** statement x
> **Hard-coded:** Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and common reference string $\mathsf{crs}_{\mathsf{OWF}}$ for the rerandomizable one-way function, puncturable PRF keys $k_{\text{sel}}, k_0, k_1, k_{\text{replace}}$ and lossy key k_{lf}
>
> On input a statement $x \in \{0,1\}^n$:
>
> * Compute $b = \mathsf{F}(k_{\text{sel}}, x)$.
> * Compute $(y_b, z_b) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_b, x))$.
> * Compute $(y_{1-b}, z_{1-b}) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\text{replace}}, \mathsf{LossyF.Eval}(k_{\text{lf}}, x)))$.
> * Output (y_0, y_1).

Figure 3: The instance-generation program $\mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\text{sel}}, k_0, k_1, k_{\text{replace}}, k_{\text{lf}}]$.

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

- Hyb_4: Same as Hyb_3, except the challenger samples the lossy function in lossy mode intead of injective mode. In particular it samples $k_{\text{lf}} \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^n)$.

- Hyb_5: Same as Hyb_4, except we let $S_{k_{\text{lf}}}$ be the image size of $\mathsf{LossyF.Eval}(k_{\text{lf}}, \cdot)$. The game aborts, outputs 0 and the attacker looses if $|S_{k_{\text{lf}}}| > 2^{p(\lambda)}$. Otherwise it proceeds as in Hyb_4.

- Hyb_6: This game intuitively runs the experiment of Hyb_5 twice, but with the same lossy function key k_{lf} across both experiments. The adversary will output a statement and proof x', π' from the first run and then a second statement and proof x, π from the second run. We declare the attacker to have won if and only if both proofs verify *and* $\mathsf{LossyF.Eval}(k_{\text{lf}}, x) = \mathsf{LossyF.Eval}(k_{\text{lf}}, x)$. We include a complete description below to ensure precision in communicating our game.

  – The challenger samples $k_{\text{lf}} \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^n)$. Let $S_{k_{\text{lf}}}$ be the image size of $\mathsf{LossyF.Eval}(k_{\text{lf}}, \cdot)$. The game aborts, outputs 0 and the attacker looses if $|S_{k_{\text{lf}}}| > 2^{p(\lambda)}$.
  – The first run begins with the adversary starts by outputting a Boolean circuit $C'\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
  – Sample $\mathsf{crs}'_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\text{owf}}})$.
  – Sample PRF keys $k'_{\text{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\text{PRF}}}, 1^n, 1^1)$, $k'_0, k'_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\text{PRF}}}, 1^n, 1^\rho)$ and $k'_{\text{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\text{PRF}}}, 1^{\ell_{\text{out}}}, 1^\rho)$.
  – The challenger then constructs $\mathsf{ObfProve}' \leftarrow i\mathcal{O}(1^{\lambda_{\text{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\text{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify}' \leftarrow i\mathcal{O}(1^{\lambda_{\text{obf}}}, 1^s, \mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\text{sel}}, k_0, k_1, k_{\text{replace}}, k_{\text{lf}}])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}_1$ are the programs from Figs. 1 and 3, and s is the same size parameter from Section 4.
  – The challenger gives the $\mathsf{crs}' = (\mathsf{crs}'_{\mathsf{OWF}}, \mathsf{ObfProve}', \mathsf{ObfVerify}')$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ then outputs a statement x' and a proof $\pi' = (b', z')$.
  – The second run begins with the adversary starts by outputting a Boolean circuit $C'\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

  - Sample $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$.
  - Sample PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$ and $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^\rho)$.
  - The challenger then constructs $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}])$ where $\mathsf{GenProof}, \mathsf{GenInst}_1$ are the programs from Figs. 1 and 3, and s is the same size parameter from Section 4.
  - The challenger gives the $\mathsf{crs} = (\mathsf{crs}_{\mathsf{OWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ then outputs a statement x' and a proof $\pi = (b, z)$.

The challenger then computes $(y_0', y_1') = \mathsf{ObfVerify}(x')$ and $(y_0, y_1) = \mathsf{ObfVerify}(x)$. The output is 1 if

$$b' = 1 - \mathsf{F}(k_{\mathsf{sel}}', x') \quad \text{and} \quad \mathsf{OWF.Verify}(\mathsf{crs}_{\mathsf{OWF}}', y_{b'}', z') = 1$$
$$\text{and} b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x) \quad \text{and} \quad \mathsf{OWF.Verify}(\mathsf{crs}_{\mathsf{OWF}}, y_b, z) = 1$$
$$\text{and} \quad \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x') = \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x).$$

- Hyb_7: Same as Hyb_6, except we remove the winning restriction on the image size of $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, \cdot)$ where k_{lf} is the sampled lossy function key. Thus the attacker can now win even if $|S_{k_{\mathsf{lf}}}| > 2^{p(\lambda)}$,
- Hyb_8: Follows the same as Hyb_7 with the following changes.
  - After the first run concludes sets $w^* = \mathsf{LossyF}, \mathsf{Eval}(k_{\mathsf{lf}}, x')$ where x' is the statement from the first run.
  - On the second run after $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^\rho)$ is sampled do the following. Set $y^* = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}, w^*))$. And compute $k_{\mathsf{replace}}^{(w^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{replace}}, w^*)$.
  - Define the following program $\mathsf{GenInst}_3$:[5]

Input: statement x
Hard-coded: Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ and common reference string $\mathsf{crs}_{\mathsf{OWF}}$ for the rerandomizable one-way function, puncturable PRF keys $k_{\mathsf{sel}}, k_0, k_1$, punctured key $k_{\mathsf{replace}}^{(w^*)}$, lossy key k_{lf} and values y^*, w^*

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

- Compute $b = \mathsf{F}(k_{\mathsf{sel}}, x)$.
- Compute $(y_b, z_b) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_b, x))$.
- If $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) \neq w^*$
  compute $(y_{1-b}, z_{1-b}) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}^{(w^*)}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x)))$.
- If $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) = w^*$ set $y_{1-b} = y^*$.
- Output (y_0, y_1).

Figure 4: The instance-generation program $\mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}^{(w^*)}, k_{\mathsf{lf}}, y^*, w^*]$.

[5]We note that the program $\mathsf{GenInst}_2$ is defined in the proofs of Appendix A.1. So we are actually not skipping over an index.

- On the second run constructs
  $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and
  $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k^{(w^*)}_{\mathsf{replace}}, k_{\mathsf{lf}}], y^*, w^*)$ where GenProof and $\mathsf{GenInst}_1$ are the programs from Figs. 1 and 4, and s is the same size parameter from Section 4.

- Hyb_9: Follows the same as Hyb_8 except that on the second run it chooses $r^* \leftarrow \{0,1\}^\rho$ and sets $y^* = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; r^*)$. (I.e. y^* is an fresh and independently sampled image of a one way function.)

5.1 Proofs of Closeness of Hybrids

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

Lemma 5.2. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure and suppose Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security for constants $\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}} \in (0,1)$. In addition, suppose that $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$ and $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. Finally, suppose Π_{PPRF} satisfies punctured correctness. Then,*

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

The proof of this lemma is the same as the corresponding one in [WW24] as the proof does not depend on in any way the re-randomization property of the one way function.

Lemma 5.3. *It holds that $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.*

Proof. As in [WW24] this holds trivially for the reason that the conditions for outputting one inHyb_1 are a subset of those for outputting 1 in Hyb_2. □

Lemma 5.4. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$-secure, Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security. Let $\lambda_{\mathsf{obf}} = (\lambda + n)^{1/\varepsilon_{\mathsf{obf}}}$, $\lambda_{\mathsf{PRF}} = (\lambda + n)^{1/\varepsilon_{\mathsf{PRF}}}$. Finally, suppose Π_{PPRF} satisfies punctured correctness and and Π_{LossyF} satisfies the injectivity property. Then,*

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

The proof of this lemma sequences through each possible statement $x \in \{0,1\}^n$ and replaces the line $\mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{b-1}, x))$ with $\mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x)))$. The execution of this proof follows from standard punctured programming techniques originated in [SW14], but with the adaptations of [WW24] in the analysis to handle stepping through all 2^n inputs. However, our proof for the key k_{replace} does not puncture at each index i. Instead it punctures at $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i)$. During these steps it is critical that k_{lf} is sampled to be an injective key so that $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) \neq \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i)$ when $x \neq i$. Otherwise, we would not be able to use indistinguishability obfuscation as programs would not be functionally equivalent. We defer the proof to Appendix A.1.

Lemma 5.5. *Suppose Π_{LossyF} is mode indistinguishability secure. Then for all PPT $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that,*

$$|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

The proof is a basic application of mode changing. We defer it to Appendix A.2.

Lemma 5.6. *Suppose* Π_{LossyF} *is has lossiness in lossy mode with all but negligible probability. Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that,*

$$|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. The only difference between Hyb_5 from Hyb_4 is that when we let $S_{k_{\mathsf{lf}}}$ be the image size of $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, \cdot)$ the game aborts and outputs 0 if $|S_{k_{\mathsf{lf}}}| > 2^{p(\lambda)}$. However, since the scheme has almost perfect lossiness, this event occurs with at most negligible probability and the difference in advantages between the two games must be negligibly close. □

Lemma 5.7. *Suppose* Π_{LossyF} *is a lossy function with lossiness function* $p(\cdot)$*, then*

$$\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]^2/2^{p(\lambda)}.$$

The game in Hyb_5 begins with the challenger sampling a value $t \in \{0,1\}^\tau$ and then sampling $k_{\mathsf{lf}} \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^n; t)$ where τ is the number of random bits used by calling $\mathsf{SetupLossy}$ with parameters 1^λ and 1^n. For this proof we call out the randomness to this function explicitly.

Proof. Let $\varepsilon = \varepsilon(\lambda)$ denote the probability of the game outputting 1 in Hyb_5. We then let μ_t the probability (over the attacker's randomness and rest of the challengers coins) that the game outputs 1 *given* that t was chosen as the randomness for $\mathsf{SetupLossy}$. It follows that

$$\varepsilon = \sum_{t \in \{0,1\}^\tau} 2^{-\tau} \mu_t$$

We next define $\mu_{t,i}$ for $t \in \{0,1\}^\tau, i \in [1, 2^{p(\lambda)}]$. Let $k_{\mathsf{lf}t} \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^n; t)$ and let $S_{k_{\mathsf{lf}t}}$ be the image size of $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, \cdot)$. We tackle two cases.
Case 1: We have $|S_{k_{\mathsf{lf}t}}| > 2^{p(\lambda)}$. In this case let $\mu_{t,i} = 0 \ \forall i \in [1, 2^{p(\lambda)}]$. Since the game aborts and outputs 0 when $|S_{k_{\mathsf{lf}t}}| > 2^{p(\lambda)}$ we have that $\mu_t = 0$. It follows by our setting that $\mu_t = \sum_{i \in [1, 2^{p(\lambda)}]} \mu_{t,i}$.
Case 2: We have $|S_{k_{\mathsf{lf}t}}| \leq 2^{p(\lambda)}$. For $i \in [1, |S_{k_{\mathsf{lf}t}}|]$ set $\mu_{t,i}$ to be the probability the challenger outputs 1 in Hyb_5 and the attacker outputs a statement x where $\mathsf{LossyF.Eval}(k_{\mathsf{lf}t}, x)$ is the i-th image in the set $S_{k_{\mathsf{lf}t}}$ (relative to some ordering) when the string t is used as the randomness to $\mathsf{SetupLossy}$. For $i \in [|S_{k_{\mathsf{lf}t}}| + 1, 2^{p(\lambda)}]$ set $\mu_{t,i} = 0$. Here it also follows by our setting that $\mu_t = \sum_{i \in [1, 2^{p(\lambda)}]} \mu_{t,i}$.

We can now begin to analyze Hyb_6. Each string t will be selected with probability exactly $2^{-\tau}$. Once a particular string t is selected the attacker will need to win the Hyb_5 experiment twice in a row and with $\mathsf{LossyF.Eval}(k_{\mathsf{lf}t}, x') = \mathsf{LossyF.Eval}(k_{\mathsf{lf}t}, x)$. Thus the probability the attacker wins for a particular string t is $\sum_{i \in [1, 2^{p(\lambda)}]} \mu_{t,i}^2$. And its total advantage is

$$2^{-\tau} \sum_{t \in \{0,1\}^\tau, i \in [1, 2^{p(\lambda)}]} \mu_{t,i}^2.$$

We next state a basic claim that will help us prove our lemma.

Claim 5.8. *Let* $a_1, \ldots, a_L \in \mathrm{R}^L$ *be a sequence of* L *real numbers then*

$$\sum_{j \in L} a_j^2 \geq \frac{1}{L} \Big(\sum_{j \in L} a_j \Big)^2.$$

Proof. Recall the L_1-L_2 norm inequality, which is that for any vector of dimension L, $|\mathbf{a}|_1 \leq |\mathbf{a}|_2 \times \sqrt{L}$ where $|\cdot|_p$ is the L_p norm. Rearranging gives $|\mathbf{a}|_2^2 \geq |\mathbf{a}|_1^2/L$. If we write $\mathbf{a} = (a_1, \cdots, a_L)$, we see that the L_1-L_2 norm inequality is equivalent to the claim. □

For any t we can apply Claim 5.8 with setting $L = 2^{p(\lambda)}$ and $a_i = \mu_{t,i}$. We then have that $\sum_{i\in[1,2^{p(\lambda)}]} \mu_{t,i}^2 \geq 2^{-p(\lambda)}\mu_t^2$. It follows that the total advantage is at least

$$2^{-p(\lambda)}2^{-\tau}\sum_{t\in\{0,1\}^\tau}\mu_t^2.$$

We next recall that from our settings that $\epsilon = \sum_{t\in\{0,1\}^\tau} 2^{-\tau}\mu_t$. We then apply Claim 5.8 once again. This time setting $L = 2^\tau$ and $a_i = 2^{-\tau}\mu_t$. Plugging things in we get

$$\sum_{t\in\{0,1\}^\tau} 2^{-2\tau}\mu_t^2 \geq 2^{-\tau}\Big(\sum_{t\in\{0,1\}^\tau}\mu_t\Big)^2.$$

If we multiply both sides of the equation by 2^τ and substitute in $\varepsilon = \sum_{t\in\{0,1\}^\tau} 2^{-\tau}\mu_t$ then we have

$$2^{-\tau}\sum_{t\in\{0,1\}^\tau}\mu_t^2 \geq \varepsilon^2.$$

Finally, we can plug this into our bound for the total advantage and now see that the total advantage is at least

$$2^{-p(\lambda)}\varepsilon^2$$

which concludes the proof.

□

Lemma 5.9. *It holds that* $\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]$.

Proof. As in Lemma 5.3 this holds trivially for the reason that the conditions for outputting one in Hyb_6 are a subset of those for outputting 1 in Hyb_7. □

Lemma 5.10. *Suppose iO is* $(1, 2^{-\lambda^{\varepsilon_{\mathrm{obf}}}})$*-secure for some constant* $\varepsilon_{\mathrm{obf}} \in (0,1)$ *and suppose* $\lambda_{\mathrm{obf}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathrm{obf}}}$. *Suppose* Π_{PPRF} *satisfies punctured correctness. Then, there exists* $\lambda_{\mathcal{A}} \in \mathbb{N}$ *such that for all* $\lambda \geq \lambda_{\mathcal{A}}$,

$$|\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]| \leq 1/2^{\lambda+p(\lambda)}.$$

This is another example of an indistinguishablity obfuscation proof where a value is hardwired into a program. The proof will proceed along similar lines to Claim A.1. One difference is that this case we aim to show that the difference in advantage is $2^{-p(\lambda)}$ times a negligible function instead of 2^{-n} times a negligible function. This is because we want to match the fact that a successful attacker that wins with non-negligible probability in Hyb_0 will only be guaranteed to win with $2^{-p(\lambda)}$ times a non-negligible value by Hyb_7. Thus we need that bound on the invariant going at this stage. A second difference is that in order to figure out what to hardwire we will first need to run the first half of Hyb_7 to determine $w^* = \mathsf{LossyF}, \mathsf{Eval}(k_{\mathsf{lf}}, x')$ where x'. This value w^* will then determine where to puncture the next k_{replace} and how to modify the next obfuscated program.

The proof is given in Appendix A.3

Lemma 5.11. *Suppose Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}{}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security for some constant $\varepsilon_{\mathsf{PRF}} \in (0, 1)$ and $\lambda_{\mathsf{PRF}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{PRF}}}$. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, it holds that*

$$|\Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1] \leq 1/2^{\lambda+p(\lambda}$$

This proof will use the security of the punctured PRF and proceed along similar lines to that of Claim A.3. Like the previous lemma there will be differences centered on needing to first learn w^* from the first half of the experiment which the reduction uses to learn where to puncture.

The proof is given in Appendix A.4.

Lemma 5.12. *Suppose there is a probabilistic polynomial time attacker $\mathcal{A}$ that wins with non-negligible probability in Hyb_0. And suppose $i\mathcal{O}$ is $(1, 2^{-\lambda_{\mathsf{obf}}{}^{\varepsilon_{\mathsf{obf}}}})$-secure, Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}{}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security, Π_{OWF} is $(1, 2^{-\lambda_{\mathsf{PRF}}{}^{\varepsilon_{\mathsf{owf}}}})$-secure and that Π_{LossyF} is mode indistinguishability secure for constants $\varepsilon_{\mathsf{obf}}, \varepsilon_{\mathsf{PRF}}, \varepsilon_{\mathsf{owf}} \in (0, 1)$. In addition, suppose $i\mathcal{O}$ is correct, Π_{PPRF} satisfies punctured correctness, and Π_{LossyF} satisfies the injectivity in injective mode and with all but negligible probability image sizes are bound by $2^{p(\lambda)}$ for some polynomial $p(\cdot)$ in lossy mode. Let $\lambda_{\mathsf{obf}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{obf}}}$, $\lambda_{\mathsf{PRF}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{PRF}}}$, and $\lambda_{\mathsf{owf}} = (\lambda + p(\lambda))^{\varepsilon_{\mathsf{owf}}}$.*

Then for some non-negligible function $\varepsilon(\lambda)$

$$\Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1] \geq 2^{-p(\lambda)}\varepsilon(\lambda).$$

Proof. By Lemmas 5.2 to 5.6 we have that if $\mathcal{A}$ has non-negligible advantage in Hyb_0 it will also have non-negligible advantage in Hyb_5. By Lemmas Lemmas 5.7 and 5.9 if $\mathcal{A}$ has non-negligible advantage in Hyb_5 then it will have 2^{-p} times a non-negligible advantage in Hyb_7. And by Lemmas 5.10 and 5.11 if $\mathcal{A}$ has 2^{-p} times a non-negligible advantage in Hyb_7 it will also have 2^{-p} times a non-negligible advantage in Hyb_9. This follows from the fact that these lemmas show the difference in advantage is at most negligible time $2^{-p(\lambda)}$. □

Lemma 5.13. *Suppose Π_{OWF} satisfies $(1, 2^{-\lambda_{\mathsf{PRF}}{}^{\varepsilon_{\mathsf{owf}}}})$-onewayness security for some constant $\varepsilon_{\mathsf{owf}} \in (0, 1)$ and $\lambda_{\mathsf{owf}} = (\lambda + p(\lambda))^{1/\varepsilon_{\mathsf{owf}}}$ and $i\mathcal{O}$ satisfies correctness. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1] \leq 2^{-p(\lambda)}\mathsf{negl}(\lambda).$$

We prove the lemma in Section A.5.

We finally conclude by observing that Theorem 5.1 follows directly from Lemmas 5.12 and 5.13. Otherwise if there existed a poly-time attacker in Hyb_0 these two lemmas would contradict each other.

5.2 Succinctness and Zero Knowledge

Theorem 5.14 (Succinctness)**.** *If Π_{OWF} is succinct, then Section 4 is succinct.*

Proof. A proof π in Section 4 consists of a bit $b \in \{0, 1\}$ and an element z output by the algorithm $\mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}})$. The output length of Π_{OWF} depends only a polynomial $p(\lambda_{\mathsf{owf}})$. Since λ_{owf} is itself a polynomial in λ, this means there exists a polynomial p' where the length is $p'(\lambda)$ which meets the requirements for succinctness.

□

Theorem 5.15 (Perfect Zero-Knowledge)**.** *If $i\mathcal{O}$ is correct, then Section 4 satisfies perfect zero-knowledge.*

This proof follows from the corresponding one in [WW24] as it did not depend on the re-randomization of the one way function.

References

[ACH20] Thomas Agrikola, Geoffroy Couteau, and Dennis Hofheinz. The usefulness of sparsifiable inputs: How to avoid subexponential io. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 187–219, Cham, 2020. Springer International Publishing.

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

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 57–74, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[AP11] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, 2011.

[AS15] Gilad Asharov and Gil Segev. Limits on the power of indistinguishability obfuscation and functional encryption. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 191–209, 2015.

[AS18] Gilad Asharov and Gil Segev. On constructing one-way permutations from indistinguishability obfuscation. *Journal of Cryptology*, 31(3):698–736, 2018.

[AWZ23a] Damiano Abram, Brent Waters, and Mark Zhandry. Security-preserving distributed samplers: How to generate any crs in one round without random oracles. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 489–514. Springer Nature Switzerland, 2023.

[AWZ23b] Damiano Abram, Brent Waters, and Mark Zhandry. Security-preserving distributed samplers: How to generate any crs in one round without random oracles. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 489–514, Cham, 2023. Springer Nature Switzerland.

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

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

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

[BDV17] Nir Bitansky, Akshay Degwekar, and Vinod Vaikuntanathan. Structure vs. hardness through the obfuscation lens. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 696–723, Cham, 2017. Springer International Publishing.

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

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

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

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

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

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

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

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography*, pages 468–497, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

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

[DGI$^+$19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 3–32, Cham, 2019. Springer International Publishing.

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

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

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

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

[HHK$^+$24] Dennis Hofheinz, Kristina Hostáková, Kastner, Karen Klein, and Akin Ünal. Compact selective opening security from lwe. In *PKC 2024*, 2024.

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

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

[KMN+14] Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 374–383, 2014.

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

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

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

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

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

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC*, pages 187–196, 2008.

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

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

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

[WW24] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for np from indistinguishability obfuscation. STOC 2024 (To Appear), 2024. https://eprint.iacr.org/2024/165.

[Zha16] Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 479–508. Springer, Heidelberg, August 2016.

[Zha19] Mark Zhandry. On ELFs, deterministic encryption, and correlated-input security. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 3–32. Springer, Heidelberg, May 2019.

A Proofs of Missing Lemmas

A.1 Proof of Lemma 5.4

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

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

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

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

- Compute $b = \mathsf{F}(k_{\mathsf{sel}}, x)$.
- Compute $(y_b, z_b) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_b, x))$.
- Compute y_{1-b} as follows:
  - If $x < i$, let $(y_{1-b}, \mathsf{st}) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x)))$.
  - If $x = i$, let $y_{1-b} = y^*$.
  - If $x > i$, let $(y_{1-b}, z_{1-b}) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{1-b}, x))$.
- Output (y_0, y_1).

Figure 5: The instance-generation program $\mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}, y^*, i]$.

Then, the challenger samples $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}}, 1^n)$ and PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$, $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^\rho)$. The challenger also samples the following additional components:

- Sample $k_{\mathsf{lf}} \leftarrow \mathsf{SetupInj}(1^\lambda, 1^n)$.
- Let $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$. Compute $r^* = \mathsf{F}(k_{b^*}, i)$ and $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; r^*)$.

The challenger computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}, y^*, i])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}_2$ are the programs from Figs. 1 and 5 and s is the bound on the program size from Section 4. Algorithm $\mathcal{B}$ gives $\mathsf{crs} = (\mathsf{crs}_{\mathsf{OWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$. After $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$, the challenger computes $(y_0, y_1) = \mathsf{ObfVerify}(x)$ and outputs 1 if

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

- $\mathsf{Hyb}_{2,i}^{(1)}$: Same as $\mathsf{Hyb}_{2,i}^{(0)}$, except after computing $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$, the challenger punctures k_{b^*} at input i and k_{replace} at input $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i)$. Namely, it computes $k_{b^*}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$ and $k_{\mathsf{replace}}^{(\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))$ It still sets $r^* = \mathsf{F}(k_{b^*}, i)$ and $(y^*, z^*) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; r^*)$. Then, it uses the punctured keys $k_{b^*}^{(\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))}$ and $k_{\mathsf{replace}}^{(i)}$ in place of k_{b^*} and k_{replace} in $\mathsf{ObfProve}$ and $\mathsf{ObfVerify}$. Specifically, $\mathsf{ObfProve}$ and $\mathsf{ObfVerify}$ are now defined as follows:
  - If $b^* = 0$, then the challenger sets $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1, k_{\mathsf{replace}}^{(\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))}, k_{\mathsf{lf}}, y^*, i])$.

  - If $b^* = 1$, then the challenger sets $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_{b^*}^{(i)}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_{b^*}^{(i)}, k_{\mathsf{replace}}^{(\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))}, k_{\mathsf{lf}}, y^*, i])$.

- $\mathsf{Hyb}_{2,i}^{(2)}$: Same as $\mathsf{Hyb}_{2,i}^{(1)}$, except the challenger samples $r^* \xleftarrow{\mathrm{R}} \{0,1\}^{\rho}$.

- $\mathsf{Hyb}_{2,i}^{(3)}$: Same as $\mathsf{Hyb}_{2,i}^{(2)}$, except the challenger sets $r^* = \mathsf{F}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))$.

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

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

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

The proof of this claim follows almost identically to [WW24] with just the syntactic changes of what hardwired values the program $\mathsf{GenInst}$ *is given. However, we include it here for completeness.*

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

- Let $b = \mathsf{F}(k_{\mathsf{sel}}, x)$. Then $\mathsf{GenInst}_2$ computes $(y_b, z_b) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_b, x))$, which is exactly how the program $\mathsf{GenInst}$ computes (y_b, z_b).

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

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

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

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

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

2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and then samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$, $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^{\rho})$. It also samples $k_{\mathsf{lf}} \leftarrow \mathsf{SetupInj}(1^{\lambda}, 1^n)$.

3. Algorithm $\mathcal{B}$ then computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, 0)$, $r^* = \mathsf{F}(k_{b^*}, 0)$ and $(y^*, z^*) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; r^*)$.

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

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

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

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

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

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

The proof of this claim is also similar to [WW24], but with one critical difference. We need to argue that the program $\mathsf{GenInst}$ *when given a punctured key $k_{\mathsf{replace}}^{(\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))}$ at input* $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i)$ *is functionally equivalent to when it is given an unpunctured version. The key k_{replace} is used at inputs $x < i$. This would be problematic if* $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) = \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i)$ *for any $x < i$. However, this is guaranteed not to happen due to the fact that the function is injective. It is for this reason that we introduce the lossy function first in injective mode and will only later on move it to lossy mode.*

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

- First, the key $k_{b^*}^{(i)}$ is punctured on input i, so it follows that $\mathsf{F}(k_{b^*}^{(i)}, x) = \mathsf{F}(k_{b^*}, x)$ for all $x \neq i$. Thus, on all inputs (x, w) where $x \neq i$, the two programs behave identically.

- Consider an input (x, w) where $x = i$. In this case, both programs first computes $b = \mathsf{F}(k_{\mathsf{sel}}, i)$ and then evaluate $\mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_b, i))$. However, by definition, $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i) = 1 - b \neq b$. In this case, both programs derive the randomness using $\mathsf{F}(k_{1-b^*}, x) = \mathsf{F}(k_1, x)$ if $b^* = 0$ and $\mathsf{F}(k_0, x)$ if $b^* = 1$. Once again, the two programs have identical functionality.

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

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

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

  For the latter to hold it is important to observe that there are no collisions when the function is in injective mode. Since k_{lf} was sampled to be an injective key (and the function has the injectivity in injective mode) *for all $x \neq i$ we have* $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) \neq \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i)$. *Thus the punctured key never has to evaluate on the punctured input for any $x \neq i$.*

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

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

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

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

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

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

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

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

2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$, and $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^{\rho})$. It also samples $k_{\mathsf{lf}} \leftarrow \mathsf{SetupInj}(1^{\lambda}, 1^n)$.

3. Algorithm $\mathcal{B}$ then computes $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$, $r^* = \mathsf{F}(k_{b^*}, i)$ and $(y^*, z^*) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; r^*)$.

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

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

   The challenger replies with an obfuscated program ObfProve.

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

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

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

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

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

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

Combining Eqs. (A.1) and (A.2), we conclude that for all sufficiently-large $\lambda \in \mathbb{N}$,

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

Claim A.3. *Suppose Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security for some constant $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and $\lambda_{\mathsf{PRF}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{PRF}}}$. Then, for all $i \in \{0, \dots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \ge \lambda_{\mathcal{A}}$, it holds that*

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

This proof follows very closely to [WW24] with the exception of syntax changes to accommodate the lossy function. We include it for completeness.

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

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

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

1. On input the security parameter $1^{\lambda_{\mathsf{PRF}}}$ (and advice string 1^{λ}), algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input 1^{λ} to obtain a circuit $C: \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$ and $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$. It then evaluates $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$. It samples $k_{1-b^*} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$ and $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^{\rho})$. Algorithm $\mathcal{B}$ also computes $k_{\mathsf{replace}}^{(i)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{replace}}, i)$. It also samples $k_{\mathsf{lf}} \leftarrow \mathsf{SetupInj}(1^{\lambda}, 1^n)$.
3. Algorithm $\mathcal{B}$ submits the input length 1^n, the output length 1^{ρ}, and a point $i \in \{0,1\}^n$ to the punctured PRF challenger. It receives the punctured key $k_{b^*}^{(i)}$ as well as the challenge value $r^* \in \{0,1\}^{\rho}$.
4. Algorithm $\mathcal{B}$ now samples $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; r^*)$. Then, algorithm $\mathcal{B}$ sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$ and constructs the programs ObfProve and ObfVerify as follows:
    - If $b^* = 0$, then it computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_{1-b^*}])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_2[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_{1-b^*}, k_{\mathsf{replace}}^{(i)}, k_{\mathsf{lf}}, y^*, i])$.

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

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

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

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

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

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

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

Claim A.4. *Suppose Π_{PPRF} satisfies selective $(1, 2^{-\lambda_{\mathsf{PRF}}^{\varepsilon_{\mathsf{PRF}}}})$-punctured security for some constant $\varepsilon_{\mathsf{PRF}} \in (0,1)$ and $\lambda_{\mathsf{PRF}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{PRF}}}$. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$, it holds that*

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

Proof. This follows by a similar argument as in the proof of Claim A.3, except the reduction algorithm provides a punctured PRF with input length $1^{\ell_{\mathsf{out}}}$ where ℓ_{out} is the output length of the lossy function. This reflects the fact that the output of the lossy function serves as the input to the punctured PRF. The reduction then programs $k_{\mathsf{replace}}^{(\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))}$ to be the punctured key (and samples k_0, k_1 itself). We note this difference is a departure from [WW24]. The rest of the argument proceeds analogously. □

Claim A.5. *Suppose $i\mathcal{O}$ is $(1, 2^{-\lambda^{\varepsilon_{\mathsf{obf}}}})$-secure for some constant $\varepsilon_{\mathsf{obf}} \in (0,1)$ and suppose $\lambda_{\mathsf{obf}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{obf}}}$. Suppose Π_{PPRF} satisfies punctured correctness. Then, for all $i \in \{0, \ldots, 2^n - 1\}$, there exists $\lambda_{\mathcal{A}} \in \mathbb{N}$ such that for all $\lambda \geq \lambda_{\mathcal{A}}$,*

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

Proof. This follows by a similar argument as the proof of Claim A.2. For completeness, we show that the programs associated with $\mathsf{ObfProve}$ and $\mathsf{ObfVerify}$ have identical behavior in the two experiments. The claim then follows by security of $i\mathcal{O}$ (as in the proof of Claim A.2). Take any $i \in \{0, \ldots, 2^n - 1\}$ and consider an execution of $\mathsf{Hyb}_{2,i}^{(3)}$ and $\mathsf{Hyb}_{2,i+1}^{(0)}$. Let $b^* = 1 - \mathsf{F}(k_{\mathsf{sel}}, i)$. First, consider the case where $b^* = 0$.

The GenProof programs. When $b^* = 0$, by the identical analysis as in the proof of Claim A.2, the programs $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_{b^*}^{(i)}, k_1]$ in $\mathsf{Hyb}_{2,i}^{(3)}$ computes the same functionality as the program $\mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_{b^*}, k_1]$ in $\mathsf{Hyb}_{2,i+1}^{(0)}$.

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

- By punctured correctness, for all $x \neq i$, it follows that
$$\mathsf{F}\big(k_{b^*}^{(i)}, x\big) = \mathsf{F}(k_{b^*}, x) \quad \text{and} \quad \mathsf{F}\big(k_{\mathsf{replace}}^{(\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))}, x\big) = \mathsf{F}(k_{\mathsf{replace}}, x).$$
Again this requires that when k_{lf} *is sampled to be in injective mode* $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) \neq \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i)$ *when* $x \neq i$. Thus, for all inputs $x \notin \{i, i+1\}$, the two programs have identical behavior.

- Suppose $x = i$. In this case, the $\mathsf{GenInstance}_2$ program in $\mathsf{Hyb}_{2,i}^{(3)}$ sets $y_{1-b} = y^*$ where $y^* = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; r^*)$ and $r^* = \mathsf{F}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, i))$. This coincides with the behavior of the program in $\mathsf{Hyb}_{2,i+1}^{(0)}$.

- Suppose $x = i + 1$. Let $b = \mathsf{F}(k_{\mathsf{sel}}, i+1)$. Then, the program in $\mathsf{Hyb}_{2,i}^{(3)}$ sets y_{1-b} as follows:
  - If $1 - b = b^* = 0$, it computes $(y_{1-b}, z_{1-b}) = \mathsf{OWF.GenInstance}\big(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}\big(k_{b^*}^{(i)}, i+1\big)\big)$. By punctured correctness, $\mathsf{F}\big(k_{b^*}^{(i)}, i+1\big) = \mathsf{F}(k_{b^*}, i+1) = \mathsf{F}(k_0, i+1)$.
  - If $1 - b = 1 - b^* = 1$, it computes $(y_{1-b}, z_{1-b}) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_1, i+1))$.

  In particular, the program in $\mathsf{Hyb}_{2,i}^{(3)}$ sets $y_{1-b} = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{1-b}, i+1))$. In $\mathsf{Hyb}_{2,i+1}^{(0)}$, the challenger sets $y_{1-b} = y^*$, where $y^* = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{1-b}, i+1))$. Once more, the two programs have identical behavior.

Completing the proof of Claim A.5. The above analysis shows that when $b^* = 0$, the $\mathsf{GenProof}$ and $\mathsf{GenInst}$ programs in $\mathsf{Hyb}_{2,i}^{(3)}$ and $\mathsf{Hyb}_{2,i+1}^{(0)}$ compute identical functionality. An analogous argument applies when $b^* = 1$. The claim now follows by security of $i\mathcal{O}$ (following the exact same structure as in the proof of Claim A.2). ☐

Claim A.6. *Suppose* $i\mathcal{O}$ *is* $(1, 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}})$*-secure for some constant* $\varepsilon_{\mathsf{obf}} \in (0,1)$ *and suppose* $\lambda_{\mathsf{obf}} = (\lambda + n + p(\lambda))^{1/\varepsilon_{\mathsf{obf}}}$. *Suppose* Π_{PPRF} *satisfies punctured correctness. Then, there exists* $\lambda_{\mathcal{A}} \in \mathbb{N}$ *such that for all* $\lambda \geq \lambda_{\mathcal{A}}$,
$$|\Pr[\mathsf{Hyb}_{2,2^n}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq 1/2^{\lambda+n}.$$

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

- Consider the behavior of $\mathsf{GenInst}_2$. Since $x \in \{0,1\}^n$, it follows that $x < 2^n$. In this case, $\mathsf{GenInst}_2$ computes
$$\begin{aligned}(y_b, z_b) &= \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_b, x)) \\ (y_{1-b}, \mathsf{st}) &= \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x))).\end{aligned}$$

- Consider the behavior of $\mathsf{GenInst}_1$. By definition, $\mathsf{GenInst}_1$ sets
$$\begin{aligned}(y_b, z_b) &= \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_b, x)) \\ (y_{1-b}, \mathsf{st}) &= \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x))).\end{aligned}$$

Both experiments sample the quantities $\mathsf{crs}_{\mathsf{OWF}}$, k_{sel}, k_0, k_1, k_{replace}, and k_{lf} using identical procedures. We conclude that the two programs compute identical functionality. The claim now follows via $i\mathcal{O}$ security (as in the proof of Claim A.1). $\square$

We now return to the proof of Lemma 5.4. By Claims A.2 to A.5, for all $i \in \{0, \dots, 2^n - 1\}$, and all sufficiently-large $\lambda \in \mathbb{N}$, it follows that

$$|\Pr[\mathsf{Hyb}_{2,i}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,i+1}^{(0)}(\mathcal{A}) = 1]| \le 6/2^{\lambda + n(\lambda)}.$$

By the triangle inequality, this means that

$$|\Pr[\mathsf{Hyb}_{2,0}^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,2^n}^{(0)}(\mathcal{A}) = 1]| \le 2^{n(\lambda)} \cdot \frac{6}{2^{\lambda+n(\lambda)}} = \frac{6}{2^\lambda}.$$

Combined with Claims A.1 and A.6, we conclude that

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

A.2 Proof of Lemma 5.5

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

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

We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$, $\mathsf{LossyAdv}_{\mathcal{B}}(\lambda) > 1/q(\lambda)$. Algorithm $\mathcal{B}$ works as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input 1^λ to obtain a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

2. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$ and $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$. It samples $k_0 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$, $k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$ and $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^\rho)$.

3. Algorithm $\mathcal{B}$ submits the input length 1^n, to the lossy function mode indistinguishability challenger. It receives the lossy key k_{lf}.

4. Algorithm $\mathcal{B}$ sets $\lambda_{\mathsf{obf}} = \lambda_{\mathsf{obf}}(\lambda, n)$ and constructs the programs $\mathsf{ObfProve}$ and $\mathsf{ObfVerify}$ as follows: It computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}])$.

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

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

Consider now the distribution of the challenge value k_{lf}:

- Suppose $k_{\mathsf{lf}} \leftarrow \mathsf{SetupInj}(1^\lambda, 1^n)$. Then, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_3 and outputs 1 with probability $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

- Suppose $k_{\mathsf{lf}} \leftarrow \mathsf{SetupLossy}(1^\lambda, 1^n)$. Then, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_3 and outputs 1 with probability $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

Then $\mathsf{LossyAdv}_{\mathcal{B}}(\lambda) \ge 1/q(\lambda)$, and the claim follows. $\square$

A.3 Proof of Lemma Lemma 5.10

Proof. We start by showing that the program $\mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}]$ in Hyb_7 and the corresponding program $\mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}^{(w^*)}, y^*, w^*]$ in Hyb_8 compute identical functionalities. Take any input $x \in \{0,1\}^n$, and consider the program $\mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}^{(w^*)}, k_{\mathsf{lf}}, y^*, w^*]$ in Hyb_8:

- When $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) \neq w^*$ it computes $(y_{1-b}, z_{1-b}) = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}^{(w^*)}, \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x)))$. This is exactly as before except the punctured key $k_{\mathsf{replace}}^{(w^*)}$ is used in $\mathsf{GenInst}_3$ instead of the unpunctured key k_{replace}. However, by punctured key correctness, this will compute the same value due to the premise that the function is *not* evaluated on w^*.

- When $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) = w^*$ it lets $y_{1-b} = y^*$ where y^* was set as $y^* = \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; \mathsf{F}(k_{\mathsf{replace}}, w^*))$. However, this is exactly the way the output would have been computed in $\mathsf{GenInst}_1$. Therefore the outputs are the same.

We conclude that on all inputs x, the verification programs $\mathsf{GenInst}_1$ and $\mathsf{GenInst}_3$ in Hyb_7 and Hyb_8 have identical input/output behavior. The lemma now holds by security of $i\mathcal{O}$. Formally, suppose there exists an infinite set $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda_{\mathcal{A}}$,

$$|\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]| > 1/2^{\lambda + p(\lambda)}.$$

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

1. On input the security parameter $1^{\lambda_{\mathsf{obf}}}$ (and advice string 1^{λ}), algorithm $\mathcal{B}$ runs $\mathcal{A}$ on security parameter λ. It begins by sampling $k_{\mathsf{lf}} \leftarrow \mathsf{SetupLossy}(1^{\lambda}, 1^n)$.

2. Next it starts the first run of $\mathcal{A}$ to get a circuit $C' \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

3. Algorithm $\mathcal{B}$ samples $\mathsf{crs}'_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and then samples PRF keys $k'_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k'_0, k'_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$, $k'_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^{\rho})$.

4. Algorithm $\mathcal{B}$ then constructs $\mathsf{ObfProve}' \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C', \mathsf{crs}_{\mathsf{OWF}}, k'_{\mathsf{sel}}, k'_0, k'_1])$ and $\mathsf{ObfVerify}' \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_1[C', \mathsf{crs}_{\mathsf{OWF}}, k'_{\mathsf{sel}}, k'_0, k'_1, k'_{\mathsf{replace}}, k_{\mathsf{lf}}])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}_1$ are the programs from Figs. 1 and 3, and s is the same size parameter from Section 4.

5. Algorithm $\mathcal{B}$ gives the $\mathsf{crs}' = (\mathsf{crs}'_{\mathsf{OWF}}, \mathsf{ObfProve}', \mathsf{ObfVerify}')$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ then outputs a statement x' and a proof $\pi' = (b', z')$.

6. Algorithm $\mathcal{B}$ then sets $w^* = \mathsf{LossyF}, \mathsf{Eval}(k_{\mathsf{lf}}, x')$ where x' is the statement from the first run.

7. Next it starts the second run of $\mathcal{A}$ to get a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

8. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and then samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$, $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^{\rho})$.

9. Algorithm $\mathcal{B}$ computes $k_{\mathsf{replace}}^{(w^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{replace}}, w^*)$.

10. Algorithm $\mathcal{B}$ computes the parameter s as in Section 4 and gives 1^s, $\mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}]$, and $\mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}^{(w^*)}, y^*, w^*]$ to the challenger. The challenger replies with an obfuscated program $\mathsf{ObfVerify}$.

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

12. After $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$ algorithm $\mathcal{B}$ computes $(y_0', y_1') = \mathsf{ObfVerify}(x')$ and $(y_0, y_1) = \mathsf{ObfVerify}(x)$. Algorithm $\mathcal{B}$ outputs 1 if:

$$b' = 1 - \mathsf{F}(k_{\mathsf{sel}}', x') \quad \text{and} \quad \mathsf{OWF.Verify}(\mathsf{crs}_{\mathsf{OWF}}', y_{b'}', z') = 1$$

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

$$\text{and} \quad \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x') = \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x).$$

If the challenger obfuscates the program $\mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}]$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_7. If the challenger obfuscates the program $\mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}^{(w^*)}, y^*, w^*]$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_8. Correspondingly, $\mathsf{iOAdv}_{\mathcal{B}}(\lambda_{\mathsf{obf}}) > 2^{-(\lambda + p(\lambda))} > 2^{-\lambda_{\mathsf{obf}}^{\varepsilon_{\mathsf{obf}}}}$. □

A.4 Proof of Lemma Lemma 5.11

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

$$|\Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1]| > 1/2^{\lambda + p(\lambda)}.$$

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

1. On input the security parameter $1^{\lambda_{\mathsf{PRF}}}$ (and advice string 1^{λ}), algorithm $\mathcal{B}$ runs $\mathcal{A}$ on security parameter λ. It begins by sampling $k_{\mathsf{lf}} \leftarrow \mathsf{SetupLossy}(1^{\lambda}, 1^n)$.

2. Next it starts the first run of $\mathcal{A}$ to get a circuit $C' \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

3. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{OWF}}' \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and then samples PRF keys $k_{\mathsf{sel}}' \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0', k_1' \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^{\rho})$, $k_{\mathsf{replace}}' \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^{\rho})$.

4. Algorithm $\mathcal{B}$ then constructs $\mathsf{ObfProve}' \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C', \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}', k_0', k_1'])$ and $\mathsf{ObfVerify}' \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_1[C', \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}', k_0', k_1', k_{\mathsf{replace}}', k_{\mathsf{lf}}])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}_1$ are the programs from Figs. 1 and 3, and s is the same size parameter from Section 4.

5. Algorithm $\mathcal{B}$ gives the $\mathsf{crs}' = (\mathsf{crs}_{\mathsf{OWF}}', \mathsf{ObfProve}', \mathsf{ObfVerify}')$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ then outputs a statement x' and a proof $\pi' = (b', z')$.

6. Algorithm $\mathcal{B}$ then sets $w^* = \mathsf{LossyF}, \mathsf{Eval}(k_{\mathsf{lf}}, x')$ where x' is the statement from the first run.

7. Next it starts the second run of $\mathcal{A}$ to get a circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.

8. Algorithm $\mathcal{B}$ samples $\mathsf{crs}_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and then samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^1)$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^n, 1^\rho)$.

9. Algorithm $\mathcal{B}$ submits the input length $1^{\ell_{\mathsf{out}}}$, the output length 1^ρ, and the point $w^* \in \{0,1\}^n$ to the punctured PRF challenger. It receives the punctured key $k_{\mathsf{replace}}^{(w^*)}$ as well as the challenge value $r^* \in \{0,1\}^\rho$.

10. Algorithm $\mathcal{B}$ now samples $(y^*, z^*) \leftarrow \mathsf{OWF.GenInstance}(\mathsf{crs}_{\mathsf{OWF}}; r^*)$.

11. Algorithm $\mathcal{B}$ then constructs $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^s, \mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}^{(k^*)}, k_{\mathsf{lf}}, y^*, w^*])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}_1$ are the programs from Figs. 1 and 4, and s is the same size parameter from Section 4.

12. Algorithm $\mathcal{B}$ computes the parameter s as in Section 4 and gives 1^s, $\mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}]$, and $\mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}^{(w^*)}, y^*, w^*]$ to the challenger. The challenger replies with an obfuscated program $\mathsf{ObfVerify}$.

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

14. After $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$ algorithm $\mathcal{B}$ computes $(y_0', y_1') = \mathsf{ObfVerify}(x')$ and $(y_0, y_1) = \mathsf{ObfVerify}(x)$. Algorithm $\mathcal{B}$ outputs 1 if:

$$b' = 1 - \mathsf{F}(k_{\mathsf{sel}}', x') \quad \text{and} \quad \mathsf{OWF.Verify}(\mathsf{crs}_{\mathsf{OWF}}', y_{b'}', z') = 1$$
$$\text{and} \quad b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x) \quad \text{and} \quad \mathsf{OWF.Verify}(\mathsf{crs}_{\mathsf{OWF}}, y_b, z) = 1$$
$$\text{and} \quad \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x') = \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x).$$

By definition, the punctured PRF challenger constructs key $k_{\mathsf{replace}}^{(w^*)}$ by first sampling $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^\rho)$ and setting $k_{\mathsf{replace}}^{(w^*)} \leftarrow \mathsf{F.Puncture}(k_{b^*}, i)$. This matches the specification both in Hyb_8 and Hyb_9. Consider now the distribution of the challenge value r^*:

- Suppose $r^* = \mathsf{F}(k_{\mathsf{replace}}, w^*)$. Then, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_8 and outputs 1 with probability $\Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]$.
- Suppose $r^* \xleftarrow{\mathrm{R}} \{0,1\}^\rho$. Then, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_9 and outputs 1 with probability $\Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1]$.

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

□

A.5 Proof of Lemma Lemma 5.13

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

$$\Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1] > 1/2^{\lambda + p(\lambda)}.$$

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

1. On input the security parameter $1^{\lambda_{\mathsf{owf}}}$ (and advice string 1^{λ}), algorithm $\mathcal{B}$ receives a one way function challenge $(\mathsf{crs}_{\mathsf{OWF}}, y^*)$.
2. Next the algorithm $\mathcal{B}$ runs $\mathcal{A}$ on security parameter λ. It begins by sampling $k_{\mathsf{lf}} \leftarrow \mathsf{SetupLossy}(1^{\lambda}, 1^{n})$.
3. Next it starts the first run of $\mathcal{A}$ to get a circuit $C' \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
4. Algorithm $\mathcal{B}$ samples $\mathsf{crs}'_{\mathsf{OWF}} \leftarrow \mathsf{OWF.Setup}(1^{\lambda_{\mathsf{owf}}})$. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and then samples PRF keys $k'_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n}, 1^{1})$, $k'_0, k'_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n}, 1^{\rho})$, $k'_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^{\rho})$.
5. Algorithm $\mathcal{B}$ then constructs $\mathsf{ObfProve}' \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{s}, \mathsf{GenProof}[C', \mathsf{crs}_{\mathsf{OWF}}, k'_{\mathsf{sel}}, k'_0, k'_1])$ and $\mathsf{ObfVerify}' \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{s}, \mathsf{GenInst}_1[C', \mathsf{crs}_{\mathsf{OWF}}, k'_{\mathsf{sel}}, k'_0, k'_1, k'_{\mathsf{replace}}, k_{\mathsf{lf}}])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}_1$ are the programs from Figs. 1 and 3, and s is the same size parameter from Section 4.
6. Algorithm $\mathcal{B}$ gives the $\mathsf{crs}' = (\mathsf{crs}'_{\mathsf{OWF}}, \mathsf{ObfProve}', \mathsf{ObfVerify}')$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ then outputs a statement x' and a proof $\pi' = (b', z')$.
7. Algorithm $\mathcal{B}$ then sets $w^* = \mathsf{LossyF}, \mathsf{Eval}(k_{\mathsf{lf}}, x')$ where x' is the statement from the first run.
8. Next it starts the second run of $\mathcal{A}$ to get a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$.
9. It sets $\lambda_{\mathsf{PRF}} = \lambda_{\mathsf{PRF}}(\lambda, n)$ and then samples PRF keys $k_{\mathsf{sel}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n}, 1^{1})$, $k_0, k_1 \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{n}, 1^{\rho})$, $k_{\mathsf{replace}} \leftarrow \mathsf{F.Setup}(1^{\lambda_{\mathsf{PRF}}}, 1^{\ell_{\mathsf{out}}}, 1^{\rho})$. *Note it does not sample* $\mathsf{OWF.Setup}$ *in this stage as it already has* $\mathsf{crs}_{\mathsf{OWF}}$ *and y^* from the OWF challenger at the beginning of the reduction.*
10. Algorithm $\mathcal{B}$ computes $k_{\mathsf{replace}}^{(w^*)} \leftarrow \mathsf{F.Puncture}(k_{\mathsf{replace}}, w^*)$.
11. Algorithm $\mathcal{B}$ then constructs $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{s}, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and $\mathsf{ObfVerify} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{s}, \mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}^{(k^*)}, k_{\mathsf{lf}}, y^*, w^*])$ where $\mathsf{GenProof}$ and $\mathsf{GenInst}_1$ are the programs from Figs. 1 and 4, and s is the same size parameter from Section 4.
12. Algorithm $\mathcal{B}$ computes the parameter s as in Section 4 and gives 1^{s}, $\mathsf{GenInst}_1[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}]$, and $\mathsf{GenInst}_3[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1, k_{\mathsf{replace}}, k_{\mathsf{lf}}^{(w^*)}, y^*, w^*]$ to the challenger. The challenger replies with an obfuscated program $\mathsf{ObfVerify}$.
13. Algorithm $\mathcal{B}$ computes $\mathsf{ObfProve} \leftarrow i\mathcal{O}(1^{\lambda_{\mathsf{obf}}}, 1^{s}, \mathsf{GenProof}[C, \mathsf{crs}_{\mathsf{OWF}}, k_{\mathsf{sel}}, k_0, k_1])$ and gives the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{OWF}}, \mathsf{ObfProve}, \mathsf{ObfVerify})$ to $\mathcal{A}$.

14. After $\mathcal{A}$ outputs the statement x and the proof $\pi = (b, z)$ algorithm $\mathcal{B}$ computes $(y'_0, y'_1) = \mathsf{ObfVerify}(x')$ and $(y_0, y_1) = \mathsf{ObfVerify}(x)$. Algorithm $\mathcal{B}$ outputs 1 if:

$$b' = 1 - \mathsf{F}(k'_{\mathsf{sel}}, x') \quad \text{and} \quad \mathsf{OWF.Verify}(\mathsf{crs}'_{\mathsf{OWF}}, y'_{b'}, z') = 1$$

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

$$\text{and} \quad \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x') = \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x).$$

In order for the game to output 1 it must be the case that $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) = \mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x') = w^*$. And that its second proof $\pi = (b, z)$ must have that $b = 1 - \mathsf{F}(k_{\mathsf{sel}}, x)$. However, on any input x where $\mathsf{LossyF.Eval}(k_{\mathsf{lf}}, x) = w^*$ the obfuscated program $\mathsf{ObfVerify}$ outputs $y_{1-\mathsf{F}(k_{\mathsf{sel}}, x)} = y^*$. This means that $\mathsf{OWF.Verify}(\mathsf{crs}_{\mathsf{OWF}}, y_b, z) = 1$ and z is a solution to the one way function.

Then $\mathsf{PPRFAdv}_{\mathcal{B}}(\lambda_{\mathsf{owf}}) > 2^{-(\lambda + p(\lambda))} > 2^{-\lambda_{\mathsf{owf}}{}^{\epsilon_{\mathsf{owf}}}}$, and the claim follows.

□