Fast and Adaptively Secure Signatures in the Random Oracle Model from Indistinguishability Obfuscation

Bei Liang[1] and Aikaterini Mitrokotsa[1]

[1] Chalmers University of Technology, Gothenburg, Sweden
{lbei,aikmitr}@chalmers.se

Abstract. Indistinguishability obfuscation ($i\mathcal{O}$) is a powerful cryptographic tool often employed to construct a variety of core cryptographic primitives such as public key encryption and signatures. In this paper, we focus on the employment of $i\mathcal{O}$ in order to construct short signatures with strong security guarantees (*i.e.,* adaptive security) that provide a very efficient signing process for resource-constrained devices. Sahai and Waters (SW) (STOC 2014) initially explored the construction of $i\mathcal{O}$-based short signature schemes but their proposal provides selective security. Ramchen and Waters (RW) (CCS 2014) attempted to provide stronger security guarantees (*i.e.,* adaptive security) but their proposal is much more computationally expensive than the SW proposal.
In this work, we propose an $i\mathcal{O}$-based short signature scheme that provides adaptive security, fast signing for resource-constrained devices and is much more cost-efficient than the RW signature scheme. More precisely, we employ a puncturable PRF with a fixed length input to get a fast and adaptively secure signature scheme without any additional hardness assumption as in the SW signature scheme. To achieve this goal, we employ the technique of Hofheinz *et al.* called *"delayed backdoor programming"* using a random oracle, which allows to embed an execution thread that will only be invoked by special inputs generated using secret key information. Furthermore, we compare the cost of our signature scheme in terms of the cost of the underlying PRG used by the puncturable PRF. Our scheme has a much lower cost than the RW scheme, while providing strong security guarantees (*i.e.,* adaptive security).

Keywords: Signature scheme, indistinguishability obfuscation, puncturable pseudo-random functions.

1 Introduction

The notion of indistinguishability obfuscation ($i\mathcal{O}$), initially introduced by Barak *et al.* [2], requires that the obfuscation of any two distinct (equal-size) programs that implement identical functionalities, renders them computationally indistinguishable from each other. However, the problem of whether or not indistinguishability obfuscation exists and how useful it is, has been unclear until

the breakthrough result of Garg *et al.* [8] when they proposed the first candidate construction of an efficient indistinguishability obfuscator for general programs [9]. This initial breakthrough by Garg *et al.* has motivated a new line of research focusing on re-exploring the construction of existing cryptographic primitives through the lens of obfuscation. For instance, Sahai and Waters [14] performed a systematic study of employing indistinguishability obfuscation to public-key encryption, short signatures, non-interactive zero-knowledge proofs, injective trapdoor functions, and oblivious transfer. This line of research is of great importance since it may lead to unexpected results and qualitatively different ways of settling cryptographic problems.

In this paper, we explore the employment of $i\mathcal{O}$ to build new signature schemes with two main properties: *(i)* they are short signatures with strong security guarantees (*i.e.,* adaptive security), and *(ii)* they provide a fast signing process suitable for resource-constrained devices (*e.g.,* sensors). The latter objective naturally leads to an *imbalanced* scheme, where the signing process is fast, while the verification process is longer; this guarantees that resource-constrained devices can sign, while computationally powerful devices will be employed for the verification. Such imbalanced schemes have been explored before *e.g.,* the research area of *delegation of computation* schemes focus on saving resources in computationally weak devices.

Although current obfuscation candidates may lead to very slow verification process, current work on obfuscation techniques (esp. on implementing specific functionalities) is under development, rendering plausible the realisation of systems with reasonable performance in the near future.

SW short signature. We begin by reviewing the selectively-secure signature scheme of Sahai-Waters (SW) based on $i\mathcal{O}$ and puncturable pseudorandom functions (PRFs) as well as one-way functions [14]. In this approach, the secret signing key is simply a key k for a puncturable PRF $F_k(\cdot)$, and a message m is signed by simply evaluating $\sigma = F_k(m)$. The public verification key is an indistinguishability obfuscation $\hat{C} \leftarrow i\mathcal{O}(C_k)$ of a circuit C_k that on input a message/signature pair (m, σ), verifies that the value $f(\sigma)$ is equal to the value $f(F_k(m))$. Verifiying any σ for m is simply done by executing the program $\hat{C}$ on input (m, σ). One significant limitation of this scheme is that it only satisfies unforgeability against a *selective* attacker. In this notion of security, the attacker is forced to preselect the message m^*, he will attempt to forge, before seeing the verification key and before querying for signatures on other messages.

RW short signature. In CCS'14, Ramchen and Waters (RW) [13] explored methods for achieving *adaptively secure* obfuscation-derived signatures in the standard model. More precisely, they employed the prefix-guessing technique of Hohenberger-Waters [12]. Their signature scheme consists of two main pieces. The first piece is a one-time signature for a tag t, which is the value of a puncturable PRF on the tag t. The second signature piece is the ability to sign the tag t according to the prefix-guessing technique [12]. A signature on the message is the tag along with the xor of these two parts. To generate the first piece, they choose a tag t of λ bits and compute $s_1 = \oplus_{i=1}^{\ell} F_1(K_1, t\|i\|m(i))$, where $F_1(K_1, \cdot)$

is a puncturable PRF with appropriate input length and $m(i)$ is the i-th bit of an ℓ-bit message m. To generate the second piece they choose λ puncturable PRFs $F_{2,i}(K_{2,i}, \cdot)$ for $i \in [1, \lambda]$ which takes inputs of i bits, and they compute $s_2 = \oplus_{i=1}^{\lambda} F_{2,i}(K_{2,i}, t^{(i)})$ where $t^{(i)}$ denotes the first i bits of t. A signature for the message m is $(t, s = s_1 \oplus s_2)$.

To improve the signing process (*i.e.*, fast sign) of their scheme, they also give a slightly modified second construction. The primary change is that instead of using λ different punctured PRF systems, each with a different domain size, a punctured PRF with a variable length domain is used in the second piece of the signature. Ramchen and Waters [13] have shown that the variable-input-length punctured PRF can be created by a length tripling PRG. We note that in the generation of the first piece of the signature, ℓ values of one fixed-input-length punctured PRF must be evaluated, and in the generation of the second piece of signature, either values of λ different fixed-input-length punctured PRFs or λ values of one variable-input-length punctured PRF must be evaluated. All these require many more computations than the SW signature scheme.

Our contribution. This state of affairs has motivated us to explore the following ambitious question: *Is it possible to construct an efficient (i.e., fast signing) and adaptively secure short signature scheme, in which the signature for a message m is a value of a puncturable PRF on m?* More precisely, in this paper we consider the problem of modifying the SW signature scheme [14] to accommodate adaptive security, where the attacker can adaptively choose which message he will forge on, and provide a positive answer to the above question. Instead of resorting to the tag-based technique of the RW scheme, which requires using either λ different fixed-input-length punctured PRFs or one variable-input-length punctured PRF, we explore to simply use one puncturable PRF with a fixed length input to get a fast signature as the SW signature scheme does, while at the same time providing strong security guarantees[1]. In particular, we present a fast signing, short signature scheme that is adaptively secure in the random oracle model relying on $i\mathcal{O}$, puncturable pseudorandom functions (PRFs) and one-way functions.

In the random oracle (RO) model, a trivial generic way to transform the selective security of the SW signature scheme to adaptive security is by hashing the message prior to signing. That is the signature for a message m is the value $\sigma = F_k(H(m))$. Now the public verification key is an indistinguishability obfuscation of a new circuit C'_k that on input a hash-value/signature pair $(H(m), \sigma)$, verifies that $f(\sigma) = f(F_k(H(m)))$. Let q_H be the number of hash queries during the game. Since with probability $1/q_H$ the simulator correctly guesses the i-th hash query *i.e.,* the query for m^*, it can then use the punctured key $k\{h^*\}$ to answer the signing queries (let h^* is the value of i-th hash query).

One could consider that the above hash-then-sign method is very trivial by employing the hash function on the message to obtain a value $h = H(m)$ with uniform distribution, thus resulting in the pseudorandomness of PRF $\sigma = F_k(h)$.

[1] Contrary to our scheme the SW signature scheme provides weaker security guarantees (*i.e.,* selective security).

However, we are motivated to seek another non-trivial method that can lead to the pseudorandomness of σ in the SW signature scheme in the random oracle model. Namely, we are taking advantage of a hash function in order to produce a new PRF key k' and thus to obtain the signature $\sigma = F_{k'}(m)$ on the message m. To achieve this goal, we employ Hofheinz *et al.*'s technique [11], called *"delayed backdoor programming"* using a programmable random oracle.

At a high level, in our construction the secret signing key is still a key $\tilde{K}$ for a puncturable PRF $F_{\tilde{K}}$, where $\tilde{K}$ is computed by a puncturable PRF F_K on input $w = H(m)$ and the signature σ on the message m is still $\sigma = F_{\tilde{K}}(m)$. The public verification key VK consists of an obfuscated program as well as a hash function H modelled as a random oracle. Let us see how to create a program Verify, that will be obfuscated to create VK. The program will actually follow a similar structure as the program of Hofheinz *et al.*'s [11] universal parameters scheme, which allows to embed an execution thread that will only be invoked by special inputs generated by using secret key information. Informally speaking, the program Verify takes as input a random string w and consists of two main stages. In the first stage, it checks to see if the random string is an encoding, which is produced by a "hidden trigger" using secret key information. This step will either output a string $\alpha \in \{0,1\}^n$ or it will output $\perp$ to indicate failure. For a uniformly randomly chosen string w, this step will output $\perp$ with very high probability, since the trigger values are encoded sparsely. Moreover, without the secret information it should be difficult to distinguish an encoding from a uniformly sampled string. If the first step fails, it will move into its second stage. At this point the program will compute the signing key $\tilde{K} = F_K(w)$ from a puncturable PRF F_K. Now instead of directly comparing the value of $f(\sigma)$ with the value $f(F_{\tilde{K}}(m))$, we add a level of indirection as Hofheinz *et al.* does. The program Verify will output the obfuscated program of the $C_{\tilde{K}}$ as in the SW signature scheme.

Our proof of adaptive security proceeds by a sequence of hybrids. Let any PPT adversary $\mathcal{A}$ make at most a polynomial number $Q = Q(\lambda)$ (unique) queries $m_1, \ldots, m_Q$ to the RO with outputs $w_1, \ldots, w_Q$. We will perform a hybrid of Q outer steps, where at outer step i, we move from using the puncturable PRF key K to output the obfuscated program $i\mathcal{O}(C_{\tilde{K}_i})$ (which is the output of the program Verify on input w_i), to having $i\mathcal{O}(C_{\tilde{K}_i})$ being encoded in w_i itself for $\tilde{K}_i = F_K(w_i)$. More precisely, following the pseudorandomness of the puncturable PRF F_K, we can let a program g to be $i\mathcal{O}(C_{\tilde{K}_i})$, where $\tilde{K}_i$ is chosen independently and then set w_i to be a "hidden trigger" encoding of g. Next, we use punctured programming techniques to replace the normal computation of the program $C_{\tilde{K}_i}$ with a hardwired and randomly sampled value for $\hat{p}_i = f(F_{\tilde{K}_i}(m_i))$. At this point on computing $i\mathcal{O}(\mathsf{Verify})(w_i)$ the output will be the program $g = i\mathcal{O}(C_{\tilde{K}_i, \hat{p}_i, m_i})$. In the final hybrid any poly-time attacker $\mathcal{A}$ that succeeds in outputting a forgery (m^*, σ^*) with non-negligible probability can be used to find a preimage of $\hat{p}_i$ for the one-way function f–that is σ^*, which breaks the security of one way functions.

Comparison of Cost. We compare the cost of the SW [14], the RW [13] schemes and our proposed signature in terms of the cost of the underlying PRG used by the puncturable PRF and the provided security.

Scheme	Security	Model	Employed Primitives	Cost
SW14 [14]	selective	standard	$i\mathcal{O}$ & fixed-length input PRF	$g_D \cdot \ell$
RW14 [13]	adaptive	standard	$i\mathcal{O}$ & fixed-length input PRF variable-length input PRF	$g_D \cdot (\lambda + 2\ell - 1) + g_T \cdot \lambda$
Ours	adaptive	random oracle	$i\mathcal{O}$ & fixed-length input PRF	less than $g_D \cdot (2\ell)$

Table 1: Comparison of our short signature scheme to the SW and RW schemes.

We note (as seen in Table 1) that although the RW scheme is proven to be adaptively secure in the standard model, their proposal is quite heavy computationally. We have chosen to provide a more efficient (fast signing), adaptively secure solution suitable for resource-constrained devices at the cost of employing the random oracle model.

Organization. The paper is organized as follows. In Section 2, we introduce the basic primitives employed in our construction. In Section 3, we present our construction of a fast signing, adaptively secure, obfuscation-derived short signature scheme and give the proof of its security, while in Section 4, we compare the cost of our construction to the SW and RW schemes. In the Appendix, we provide some details for the proof of security of the proposed short signature scheme.

2 Preliminaries

2.1 Signature Schemes

Definition 1. *A signature scheme with message space $\mathcal{M}(\lambda)$, signature key space $\mathcal{SK}(\lambda)$ and verification key space $\mathcal{VK}(\lambda)$ consists of the PPT algorithms $\mathsf{SIG} = (\mathsf{SIG.Setup}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$:*

*– **Key generation.** $\mathsf{SIG.Setup}$ is a randomized algorithm that takes as input the security parameter 1^λ and outputs the signing key $\mathsf{sk} \in \mathcal{SK}$ and the verification key $\mathsf{vk} \in \mathcal{VK}$.*

*– **Signature generation.** $\mathsf{SIG.Sign}$ takes as input the signing key $\mathsf{sk} \in \mathcal{SK}$ and a message $m \in \mathcal{M}$ and outputs a signature σ.*

*– **Verification.** $\mathsf{SIG.Verify}$ takes as input a verification key $\mathsf{vk} \in \mathcal{VK}$, a message $m \in \mathcal{M}$ and a signature σ and outputs either 0 or 1.*

Correctness. *For all $\lambda \in \mathbb{N}$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Setup}(1^\lambda)$, messages $m \in \mathcal{M}(\lambda)$, we require that $\mathsf{SIG.Verify}(\mathsf{vk}, m, \mathsf{SIG.Sign}(\mathsf{sk}, m)) = 1$.*

We say that a signature scheme $\mathsf{SIG} = (\mathsf{SIG.Setup}, \mathsf{SIG.Sign}, \mathsf{SIG.Verify})$ *is existentially unforgeable under adaptively chosen message attacks if*

$$\Pr[\mathit{Exp}^{\mathsf{uf\text{-}cma}}_{\mathsf{SIG},\mathcal{A}}(\lambda) = 1] \leq negl(\lambda)$$

for some negligible function $negl$ *and for all PPT attackers* $\mathcal{A}$*, where* $\mathit{Exp}^{\mathsf{uf\text{-}cma}}_{\mathsf{SIG},\mathcal{A}}(\lambda)$ *is the following experiment with the scheme* SIG *and an attacker* $\mathcal{A}$*:*
1. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Setup}(1^\lambda)$*.*
2. $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(1^\lambda, \mathsf{vk})$*.*
If $\mathsf{SIG.Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$ *and* m^* *was not queried to the* $\mathsf{Sign}(\mathsf{sk}, \cdot)$ *oracle, then return 1, else return 0.*

2.2 Indistinguishability Obfuscation

Definition 2 (Indistinguishability obfuscation [8]). *A probabilistic polynomial time (PPT) algorithm* $i\mathcal{O}$ *is said to be an indistinguishability obfuscator for a circuits class* $\{\mathcal{C}_\lambda\}$*, if the following conditions are satisfied:*

- *For all security parameters* $\lambda \in \mathbb{N}$*, for all* $C \in C_\lambda$*, for all inputs* x*, we have that:*
$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1.$$
- *For any (not necessarily uniform) PPT adversaries* (Samp, D)*, there exists a negligible function* $negl(\cdot)$ *such that the following holds: if* $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)] > 1 - negl(\lambda)$*, then we have:*
$$\big|\Pr[D(\sigma, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)] - \Pr[D(\sigma, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)]\big| \leq negl(\lambda).$$

2.3 Puncturable PRFs

Definition 3. *A puncturable family of PRFs* F *mapping is given by a triple of Turing Machines* $(\mathsf{Key}_F, \mathsf{Puncture}_F, \mathsf{Eval}_F)$*, and a pair of computable functions* $\tau_1(\cdot)$ *and* $\tau_2(\cdot)$*, satisfying the following conditions:*

- ***[Functionality preserved under puncturing]*** *For every PPT adversary* $\mathcal{A}$ *such that* $\mathcal{A}(1^\lambda)$ *outputs a point* $x^* \in \{0,1\}^{\tau_1(\lambda)}$*, then for all* $x \in \{0,1\}^{\tau_1(\lambda)}$ *where* $x \neq x^*$*, we have that:*
$$\Pr[\mathsf{Eval}_F(K, x) = \mathsf{Eval}_F(K_{x^*}, x) : K \leftarrow \mathsf{Key}_F(1^\lambda), K_{x^*} \leftarrow \mathsf{Puncture}_F(K, x^*)] = 1.$$
- ***[Pseudorandom at punctured point]*** *For every PPT adversary* $(\mathcal{A}_1, \mathcal{A}_2)$ *such that* $\mathcal{A}_1(1^\lambda)$ *outputs a point* $x^* \in \{0,1\}^{\tau_1(\lambda)}$ *and a state* σ*, consider an experiment where* $K \leftarrow \mathsf{Key}_F(1^\lambda)$ *and* $K_{x^*} \leftarrow \mathsf{Puncture}_F(K, x^*)$*. Then, we have:*
$$\big|\Pr[\mathcal{A}_2(\sigma, K_{x^*}, x^*, \mathsf{Eval}_F(K, x^*)) = 1] - \Pr[\mathcal{A}_2(\sigma, K_{x^*}, x^*, U_{\tau_2(\lambda)}) = 1]\big| = negl(\lambda),$$

where $negl(\cdot)$ is a negligible function and $U_{\tau_2(\lambda)}$ denotes the uniform distribution over $\tau_2(\lambda)$ bits.

Theorem 1. *[14] If one-way functions exist, then for all efficiently computable functions $\tau_1(\lambda)$ and $\tau_2(\lambda)$, there exists a puncturable family of PRFs that maps $\tau_1(\lambda)$ bits to $\tau_2(\lambda)$ bits.*

3 Adaptively Secure Short Signatures in the RO Model

The proposed construction is parameterized over a security parameter λ and has message space $\mathcal{M} = \mathcal{M}(\lambda) = \{0,1\}^{\ell(\lambda)}$ for some polynomial function $\ell(\cdot)$. We use a random oracle $H : \{0,1\}^{\ell} \to \{0,1\}^{n^2+n}$, a PRG mapping n-bit inputs to $2n$-bit outputs, a one way function $f(\cdot)$ mapping ℓ'-bit inputs to $\hat{\ell}$-bit outputs, and a hash function $H : \{0,1\}^{\ell} \to \{0,1\}^{n^2+n}$. We also make use of four different puncturable PRFs in our construction:

- $F_1^{(n)}$ is a sequence of $2n$ puncturable PRFs $\{F_1^{1,0}, F_1^{1,1}, \ldots, F_1^{n,0}, F_1^{n,1}\}$ that each maps n-bit inputs to n-bit outputs. The corresponding key sequence is denoted by $K_1^{(n)} = \{K_1^{1,0}, K_1^{1,1}, \ldots, K_1^{n,0}, K_1^{n,1}\}$. Then, on an n-bit input v, the output of the function $F_1^{(n)}$ is denoted by $F_1^{(n)}(K_1^{(n)}, v)$.
- $F_2(K_2, \cdot)$ is a puncturable PRF mapping (n^2+n)-bit inputs to n_1-bit outputs, where n_1 is the size of K_3 for the puncturable PRF $F_3(K_3, \cdot)$.
- $F_2'(K_2', \cdot)$ is a puncturable PRF mapping (n^2+n)-bit inputs to n_2-bit outputs, where n_2 is the size of the randomness r used by the $i\mathcal{O}$.
- $F_3(K_3, \cdot)$ is a puncturable PRF mapping ℓ-bit inputs to ℓ'-bit outputs.

Setup(1^λ)**:** On input 1^λ, the Setup algorithm firstly samples the PRF keys $K_1^{(n)}, K_2, K_2'$. Next, it creates an obfuscation of the program Verify as depicted in Figure 1(a). The size of the program is padded to be the maximum of the size of itself and the corresponding programs Verify in various hybrids, as described in section 3.1. The verification key, VK, is the obfuscated program $i\mathcal{O}([\mathsf{Verify}])$. The secret key SK is $(K_1^{(n)}, K_2, K_2')$.

Sign$(SK, m \in \mathcal{M})$: To sign a message m, the Sign algorithm queries the random oracle H to obtain $H(m) = u\|v$ and computes $K_3 = F_2(K_2, u\|v)$. It outputs $\sigma = F_3(K_3, m)$.

Verify(VK, m, σ)**:** To verify a signature σ on message m, the Verify algorithm queries the random oracle H to get $H(m) = u\|v$ and then evaluates the obfuscated program $i\mathcal{O}(\mathsf{Verify})$ with inputs $H(m) = u\|v$ to obtain the obfuscated program $i\mathcal{O}(P_{K_3}; r')$. Then, it runs the program $i\mathcal{O}(P_{K_3}; r')$ on inputs (m, σ) and returns its output.

Theorem 2. *If $i\mathcal{O}$ is a secure indistinguishability obfuscator, $F_1^{(n)}$, F_2, F_2', F_3 are secure puncturable PRFs, $f(\cdot)$ is a one way function, and PRG is a secure pseudo-random generator, then our signature scheme given above is existentially unforgeable under chosen message attacks in the random oracle model.*

Verify

Hardwired into the circuit: $K_1^{(n)}, K_2, K_2'$.

Input to the circuit: $u = u[1] \| \ldots \| u[n] \in \{0,1\}^{n^2}$, $v \in \{0,1\}^n$.

Algorithm:

1. Compute $F_1^{(n)}(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
2. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $\alpha_i = 0$ else if $u[i] = y_{i,1}$ set $\alpha_i = 1$ else set $\alpha_i = \perp$.
3. If $\alpha \in \{0,1\}^n$ contains no $\perp$s, output α.
4. Else set $K_3 = F_2(K_2, u\|v)$, $r' = F_2'(K_2', u\|v)$. Output $i\mathcal{O}(P_{K_3}; r')$ of the program a P_{K_3} of Figure 1(b).

(a) The program **Verify**

P_{K_3}

Hardwired into the circuit: K_3.

Input to the circuit: $m \in \{0,1\}^\ell$, $\sigma \in \{0,1\}^{\ell'}$.

Algorithm:

1. Test if $f(\sigma) = f(F_3(K_3, m))$. Output accept if true, reject if false.

(b) The program P_{K_3}

Fig. 1: The description of the programs Verify and P_{K_3}

Proof. We prove this theorem by a sequence of polynomial hybrid arguments. We begin with Hyb_0 corresponding to the original signature security game. Suppose the adversary makes $q_{\mathsf{RO}}(\lambda)$ queries to the random oracle H, for some polynomial $q_{\mathsf{RO}}(\cdot)$. The argument proceeds via the sequence of hybrids Hyb_0, $\mathsf{Hyb}_{1,1}$, $\mathsf{Hyb}_{1,2}$,..., $\mathsf{Hyb}_{1,12}$, $\mathsf{Hyb}_{2,1}$, $\mathsf{Hyb}_{2,2}$,..., $\mathsf{Hyb}_{2,12}$,..., $\mathsf{Hyb}_{q_{\mathsf{RO}},12}$, each of which we prove to be indistinguishable from the previous one. We define $\mathsf{Hyb}_{0,12}$ for convenience in our proof. Then, we show that any poly-time attacker in the final hybrid $\mathsf{Hyb}_{q_{\mathsf{RO}},12}$ that succeeds in forging with non-negligible probability can be used to break one way functions.

We start by describing Hyb_0 and then describe $\mathsf{Hyb}_{s,1}$, $\mathsf{Hyb}_{s,2}$,..., $\mathsf{Hyb}_{s,12}$ for $s \in [q_{\mathsf{RO}}]$. We denote changes between subsequent hybrids using underlined font with a red line. We also add an intuition behind why the hybrids should be indistinguishable. The detailed proofs can be found in Appendix A.1.

Here we refer to all signatures generated by the Signing oracles and the Random Oracle as intermediate steps in order to answer $\mathcal{A}$'s respective queries.

3.1 Sequence of Games

Hyb_0: This is the original security game instantiated by our construction.
– **Setup phase** The challenger samples the PRF keys $K_1^{(n)}, K_2, K_2'$ and creates the obfuscated program $i\mathcal{O}([\mathsf{Verify}])$, where the program Verify is defined as in

Figure 1(a) and padded to be of appropriate size. The challenger sets $VK = i\mathcal{O}([\mathsf{Verify}])$ and passes it to the attacker $\mathcal{A}$.

– **Query Phase**

- Random Oracle Queries: For each random oracle query m_j, the challenger checks if m_j has already been queried.
  If yes, it lets $(m_j, u_j \| v_j)$ be the tuple corresponding to m_j and sends $u_j \| v_j$ to $\mathcal{A}$.
  Else it chooses $u_j \leftarrow \{0,1\}^{n^2}$, $v_j \leftarrow \{0,1\}^n$, sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to a table.
- Signing Queries: For each signature query m_k, the challenger first looks up the table to obtain $H(m_k) = u_k \| v_k$ and then, computes $K_3 = F_2(K_2, u_k \| v_k)$. It returns $\sigma = F_3(K_3, m_k)$.

– **Forgery Phase** $\mathcal{A}$ finally outputs a forgery (m^*, σ^*) and wins if it holds $\mathsf{Verify}(VK, m^*, \sigma^*) = 1$.

We will now describe the hybrids $\mathsf{Hyb}_{s-1,12}$, $\mathsf{Hyb}_{s,1}, \ldots, \mathsf{Hyb}_{s,12}$ for $s \leq q_{\mathsf{RO}}$.

$\mathsf{Hyb}_{s-1,12}$: It is defined as following:

– **Setup phase** The challenger samples the PRF keys $K_1^{(n)}, K_2, K_2'$ and creates the obfuscated program $i\mathcal{O}([\mathsf{Verify}])$, where the program Verify is defined as in Figure 1(a) and padded to be of appropriate size. The challenger sets $VK = i\mathcal{O}([\mathsf{Verify}])$ and passes it to the attacker $\mathcal{A}$.

– **Query Phase**

- Random Oracle Queries: For each random oracle query m_j, the challenger checks if m_j has already been queried.
  If yes, let $(m_j, u_j \| v_j)$ be the tuple corresponding to m_j. The challenger sends $u_j \| v_j$ to $\mathcal{A}$.
  Else if $j < s$, the challenger chooses $v_j \leftarrow \{0,1\}^n$. It also chooses $K_3 \leftarrow \{0,1\}^{n_1}$, $e' \leftarrow \{0,1\}^{n_2}$, $p_j \leftarrow \{0,1\}^{\ell'}$ and computes $\hat{p}_j = f(p_j)$. It computes $K_3\{m_j\} \leftarrow \mathsf{Puncture}(K_3, m_j)$ and sets $g = i\mathcal{O}(P'_{K_3\{m_j\}, \hat{p}_j, m_j}; e')$ as defined in Figure 2(a). The challenger sets $(y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1}) := F_1^{(n)}(K_1^{(n)}, v_j)$, $u_j[i] := y_{i,g_i}$ for all $i \in [1,n]$, where g_i is the i-th bit of g. It sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to a table.
  If $j \geq s$, the challenger chooses $u_j \leftarrow \{0,1\}^{n^2}$, $v_j \leftarrow \{0,1\}^n$, sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to the table.
- Signing Queries: For each signature query m_k, the challenger first looks up the table to obtain $H(m_k) = u_k \| v_k$.
  If $\exists j$ s.t. $m_k = m_j \ \wedge \ j < s$, it returns $\sigma_k = p_j$;
  If $m_k = m_s$, it computes $K_3 = F_2(K_2, u_s \| v_s)$ and returns $\sigma_k = F_3(K_3, m_k)$;
  Else if $\exists j$ s.t. $m_k = m_j \ \wedge \ j > s$, it computes $K_3 = F_2(K_2, u_k \| v_k)$ and returns $\sigma_k = F_3(K_3, m_k)$.

– **Forgery Phase** $\mathcal{A}$ finally outputs a forgery (m^*, σ^*) and wins if it holds $\mathsf{Verify}(VK, m^*, \sigma^*) = 1$.

$P'_{K_3\{m_j\},\hat{p}_j,m_j}$

Hardwired into the circuit: $K_3\{m_j\}, m_j, \hat{p}_j$.

Input to the circuit: $m \in \{0,1\}^\ell$, $\sigma \in \{0,1\}^{\ell'}$.

Algorithm:

1. If $m = m_j$, test if $f(\sigma) = \hat{p}_j$. Output accept if true, reject if false.
2. Else test if $f(\sigma) = f(F_3(K_3\{m_j\}, m))$. Output accept if true, else reject.

(a) The program $P'_{K_3\{m_j\},\hat{p}_j,m_j}$

Verify 2

Hardwired into the circuit: $K_1^{(n)}\{v_s^*\}$, K_2, K_2', v_s^*, $\{z_{i,b}^*\}_{b\in\{0,1\},i\in[n]}$.

Input to the circuit: $u = u[1]\| \ldots \|u[n] \in \{0,1\}^{n^2}$, $v \in \{0,1\}^n$.

Algorithm:

1. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
   If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $\alpha_i = 0$; if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $\alpha_i = 1$; else $\alpha_i = \perp$.
   Go to step 4.
2. Compute $F_1^{(n)}(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
3. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $\alpha_i = 0$ else if $u[i] = y_{i,1}$ set $\alpha_i = 1$ else set $\alpha_i = \perp$.
4. If $\alpha \in \{0,1\}^n$ contains no $\perp$s, output α.
5. Else set $K_3 = F_2(K_2, u\|v)$, $r' = F_2'(K_2', u\|v)$. Output $i\mathcal{O}(P_{K_3}; r')$ of the program P_{K_3} of Figure 1(b).

(b) The program **Verify 2**

Fig. 2: The description of the programs $P'_{K_3\{m_j\},\hat{p}_j,m_j}$ and Verify 2

$\mathsf{Hyb}_{s,1}$: This hybrid is identical to the $\mathsf{Hyb}_{s-1,12}$, except that the verification key VK is created as an obfuscation of the program Verify 2 which is defined in Figure 2(b). That is:

– **Setup phase** The challenger samples the PRF keys $K_1^{(n)}, K_2, K_2'$. It also chooses $v_s^* \leftarrow \{0,1\}^n$, $u_s^* \leftarrow \{0,1\}^{n^2}$ and sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1^{(n)}(K_1^{(n)}, v_s^*)$. Then, for all $b \in \{0,1\}$ and $i \in [1,n]$, the challenger sets $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$. It computes $K_2\{u_s^*\|v_s^*\} \leftarrow \mathsf{Puncture}(K_2, u_s^*\|v_s^*)$. It creates the obfuscated program $i\mathcal{O}([\mathsf{Verify\ 2}])$ where the program Verify 2 is defined as in Figure 2(b) and padded to be of appropriate size. The challenger sets $VK = i\mathcal{O}([\mathsf{Verify\ 2}])$ and passes it to the attacker $\mathcal{A}$.

– **Query Phase**

- Random Oracle Queries: For each random oracle query m_j, the challenger checks if m_j has already been queried.
  If yes, it lets $(m_j, u_j\|v_j)$ to be the tuple corresponding to m_j and sends $u_j\|v_j$ to $\mathcal{A}$.
  Else if $j < s$, the challenger chooses $v_j \leftarrow \{0,1\}^n$. It also chooses $K_3 \leftarrow$

$\{0,1\}^{n_1}$, $e' \leftarrow \{0,1\}^{n_2}$, $p_j \leftarrow \{0,1\}^{\ell'}$ and computes $\hat{p}_j = f(p_j)$. It computes $K_3\{m_j\} \leftarrow \mathsf{Puncture}(K_3, m_j)$ and sets $g = i\mathcal{O}(P'_{K_3\{m_j\},\hat{p}_j,m_j}; e')$ as defined in Figure 2(a). The challenger sets $(y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1}) := F_1^{(n)}(K_1^{(n)}, v_j)$, $u_j[i] := y_{i,g_i}$ for all $i \in [1,n]$, where g_i is the i-th bit of g. It sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to a table.
If $j = s$, the challenger sets $u_j = u_s^*$ and $v_j = v_s^*$. It sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to the table.
If $j > s$, the challenger chooses $u_j \leftarrow \{0,1\}^{n^2}$, $v_j \leftarrow \{0,1\}^n$, it sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to the table.

- Signing Queries: For each signature query m_k, the challenger first looks up the table to obtain $H(m_k) = u_k \| v_k$.
If $\exists j$ s.t. $m_k = m_j \ \wedge \ j < s$, it returns $\sigma_k = p_j$;
If $m_k = m_s$, the challenger computes $K_3 = F_2(K_2, u_s^* \| v_s^*)$ and returns $\sigma_k = F_3(K_3, m_k)$;
Else if $\exists j$ s.t. $m_k = m_j \ \wedge \ j > s$, it computes $K_3 = F_2(K_2\{u_s^* \| v_s^*\}, u_k \| v_k)$ and returns $\sigma_k = F_3(K_3, m_k)$.

– **Forgery Phase** $\mathcal{A}$ finally outputs a forgery (m^*, σ^*) and wins if it holds $\mathsf{Verify}(VK, m^*, \sigma^*) = 1$.

Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s-1,12}$ by $i\mathcal{O}$ between the programs Verify and Verify 2. The proof is given by Lemma 2 (Appendix).

$\mathsf{Hyb}_{s,2}$: This hybrid is identical to $\mathsf{Hyb}_{s,1}$, except that in the Setup phase the challenger randomly chooses $y_{i,b}^* \leftarrow \{0,1\}^n$ for all $b \in \{0,1\}$ and $i \in [1,n]$ instead of computing by $F_1^{(n)}(K_1^{(n)}, v_s^*)$. Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s,1}$ by the security of the puncturable PRFs $F_1^{(n)}(K_1^{(n)}, \cdot)$. The proof is given by Lemma 3 (Appendix).

$\mathsf{Hyb}_{s,3}$: This hybrid is identical to $\mathsf{Hyb}_{s,2}$, except that in the Setup phase the challenger randomly chooses $z_{i,b}^* \leftarrow \{0,1\}^{2n}$ for all $b \in \{0,1\}$ and $i \in [1,n]$ instead of computing $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$. Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s,2}$ by the security of the PRG. The proof is given by Lemma 4 (Appendix).

$\mathsf{Hyb}_{s,4}$: This hybrid is identical to $\mathsf{Hyb}_{s,3}$, except that VK is created as an obfuscation of the program Verify 3 (defined in Figure 3). That is:

– **Setup phase** The challenger samples the PRF keys $K_1^{(n)}, K_2, K_2'$. It also chooses $v_s^* \leftarrow \{0,1\}^n$, $u_s^* \leftarrow \{0,1\}^{n^2}$ and $z_{i,b}^* \leftarrow \{0,1\}^{2n}$ for all $b \in \{0,1\}$ and $i \in [1,n]$. It computes $K_2\{u_s^* \| v_s^*\} \leftarrow \mathsf{Puncture}(K_2, u_s^* \| v_s^*)$, $e = F_2(K_2, u_s^* \| v_s^*)$, $K_2'\{u_s^* \| v_s^*\} \leftarrow \mathsf{Puncture}(K_2', u_s^* \| v_s^*)$, $e' = F_2'(K_2', u_s^* \| v_s^*)$ and sets $g = i\mathcal{O}(P_e; e')$. Then, it creates the obfuscated program $i\mathcal{O}([\mathsf{Verify\ 3}])$ (Verify 3 is defined in Figure 3) which is padded to be of appropriate size. The challenger sets $VK = i\mathcal{O}([\mathsf{Verify\ 3}])$ and passes it to $\mathcal{A}$.

– **Query Phase**

- Random Oracle Queries: as in $\mathsf{Hyb}_{s,3}$.

- Signing Queries: For each signature query m_k, the challenger first looks up the table to obtain $H(m_k) = u_k \| v_k$.
  If $\exists j$ s.t. $m_k = m_j \ \wedge \ j < s$, it returns $\sigma_k = p_j$;
  If $m_k = m_s$, it returns $\sigma_k = F_3(e, m_k)$;
  Else if $\exists j$ s.t. $m_k = m_j \ \wedge \ j > s$, it computes $K_3 = F_2(K_2\{u_s^* \| v_s^*\}, u_k \| v_k)$ and returns $\sigma_k = F_3(K_3, m_k)$.

– **Forgery Phase** $\mathcal{A}$ finally outputs a forgery (m^*, σ^*) and wins if it holds $\mathsf{Verify}(VK, m^*, \sigma^*) = 1$.

Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s,3}$ by $i\mathcal{O}$ between the programs Verify 2 and Verify 3. The proof is given by Lemma 5 (Appendix).

Verify 3

Hardwired into the circuit: $K_1^{(n)}\{v_s^*\}$, $K_2\{u_s^* \| v_s^*\}, K_2'\{u_s^* \| v_s^*\}$, v_s^*, u_s^*, g, $\{z_{i,b}^*\}_{b\in\{0,1\}, i\in[n]}$.

Input to the circuit: $u = u[1] \| \ldots \| u[n] \in \{0,1\}^{n^2}$, $v \in \{0,1\}^n$.

Algorithm:

1. If $u = u_s^*$ and $v = v_s^*$ output g and stop.
2. If $v = v_s^*$ then for $i = 1, \ldots, n$ do
   If $\mathsf{PRG}(u[i]) = z_{i,0}^*$ let $\alpha_i = 0$; if $\mathsf{PRG}(u[i]) = z_{i,1}^*$ let $\alpha_i = 1$; else $\alpha_i = \perp$.
   Go to step 5.
3. Compute $F_1^{(n)}(K_1^{(n)}, v) = (y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1})$.
4. For $i = 1, \ldots, n$, if $u[i] = y_{i,0}$ set $\alpha_i = 0$ else if $u[i] = y_{i,1}$ set $\alpha_i = 1$ else set $\alpha_i = \perp$.
5. If $\alpha \in \{0,1\}^n$ contains no $\perp$s, output α.
6. Else set $K_3 = F_2(K_2\{u_s^* \| v_s^*\}, u \| v)$, $r' = F_2'(K_2'\{u_s^* \| v_s^*\}, u \| v)$. Output $i\mathcal{O}(P_{K_3}; r')$ of the program P_{K_3} of Figure 1(b).

Fig. 3: Description of the program **Verify 3**.

$\mathsf{Hyb}_{s,5}$: This hybrid is identical to $\mathsf{Hyb}_{s,4}$, except that in the Setup phase the challenger randomly chooses $e \leftarrow \{0,1\}^{n_1}$ instead of computing $e = F_2(K_2, u_s^* \| v_s^*)$. Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s,4}$ by the security of the puncturable PRFs $F_2(K_2, \cdot)$. The proof is given by Lemma 6 (Appendix).

$\mathsf{Hyb}_{s,6}$: This hybrid is identical to $\mathsf{Hyb}_{s,5}$, except that the challenger randomly chooses $e' \leftarrow \{0,1\}^{n_2}$ instead of computing $e' = F_2'(K_2', u_s^* \| v_s^*)$. Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s,5}$ by the security of the puncturable PRFs $F_2'(K_2', \cdot)$. The proof is given by Lemma 7 (Appendix).

$\mathsf{Hyb}_{s,7}$: This hybrid is identical to $\mathsf{Hyb}_{s,6}$, except that the challenger sets the value of $z_{i,g_i}^* = \mathsf{PRG}(u_s^*[i])$ for $u_s^*[i] \leftarrow \{0,1\}^n$ instead of randomly choosing z_{i,g_i}^* from $\{0,1\}^{2n}$. More precisely, this game has the following phases:

– **Setup phase** The challenger samples the PRF keys $K_1^{(n)}, K_2, K_2'$. It chooses $v_s^* \leftarrow \{0,1\}^n$, $e \leftarrow \{0,1\}^{n_1}$ and $e' \leftarrow \{0,1\}^{n_2}$. It sets $g = i\mathcal{O}(P_e; e')$. For all $i \in [1,n]$, it chooses $y_{i,g_i}^* \leftarrow \{0,1\}^n$, $z_{i,\bar{g}_i}^* \leftarrow \{0,1\}^{2n}$, and sets $u_s^*[i] := y_{i,g_i}^*$, $z_{i,g_i}^* := \mathsf{PRG}(y_{i,g_i}^*)$, where g_i is the i-th bit of g and $\bar{g}_i = 1 - g_i$. The challenger computes $u_s^* = u_s^*[1] \| \cdots \| u_s^*[n]$ and $K_2\{u_s^* \| v_s^*\} \leftarrow \mathsf{Puncture}(K_2, u_s^* \| v_s^*)$. Then, it creates the obfuscated program $i\mathcal{O}([\mathsf{Verify\ 2}])$, where the program Verify 2 is defined as in Figure 2(b) and padded to be of appropriate size. The challenger sets $VK = i\mathcal{O}([\mathsf{Verify\ 2}])$ and passes it to the attacker $\mathcal{A}$.

Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s,6}$ by $i\mathcal{O}$ between the programs Verify 3 and Verify 2. The proof is given by Lemma 8 (Appendix).

$\mathsf{Hyb}_{s,8}$: This hybrid is identical to the $\mathsf{Hyb}_{s,7}$, except that the challenger sets $z_{i,\bar{g}_i}^* = \mathsf{PRG}(y_{i,\bar{g}_i}^*)$, where $y_{i,\bar{g}_i}^* \leftarrow \{0,1\}^n$, instead of randomly choosing $z_{i,\bar{g}_i}^*$ from $\{0,1\}^{2n}$. That is:

– **Setup phase** The challenger samples the PRF keys $K_1^{(n)}, K_2, K_2'$. It also chooses $v_s^* \leftarrow \{0,1\}^n$, $e \leftarrow \{0,1\}^{n_1}$ and $e' \leftarrow \{0,1\}^{n_2}$. It chooses $g = i\mathcal{O}(P_e; e')$. For all $i \in [1,n]$, it sets $y_{i,b}^* \leftarrow \{0,1\}^n$, $u_s^*[i] = y_{i,g_i}^*$, and sets $z_{i,g_i}^* := \mathsf{PRG}(y_{i,g_i}^*)$ and $z_{i,\bar{g}_i}^* := \mathsf{PRG}(y_{i,\bar{g}_i}^*)$, where g_i is the i-th bit of g and $\bar{g}_i = 1 - g_i$. The challenger computes $u_s^* = u_s^*[1] \| \cdots \| u_s^*[n]$ and $K_2\{u_s^* \| v_s^*\} \leftarrow \mathsf{Puncture}(K_2, u_s^* \| v_s^*)$. Then, it creates the obfuscated program $i\mathcal{O}([\mathsf{Verify\ 2}])$ (Verify 2 is defined as in Figure 2(b)) which is padded to be of appropriate size. The challenger sets $VK = i\mathcal{O}([\mathsf{Verify\ 2}])$ and passes it to $\mathcal{A}$.

Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s,7}$ by the security of the PRG. The proof is given by Lemma 9 (Appendix).

$\mathsf{Hyb}_{s,9}$: This hybrid is identical to $\mathsf{Hyb}_{s,8}$, except that the challenger sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) := F_1^{(n)}(K_1^{(n)}, v_s^*)$ instead of randomly choosing $y_{i,b}^*$ from $\{0,1\}^n$. Intuitively, it is indistinguishable from $\mathsf{Hyb}_{s,8}$ by the security of the puncturable PRFs $F_1^{(n)}(K_1^{(n)}, \cdot)$. The proof is given by Lemma 10 (Appendix).

$\mathsf{Hyb}_{s,10}$: This hybrid is identical to $\mathsf{Hyb}_{s,9}$, except that the verification key VK is created as an obfuscation of the program Verify as in Figure 1(a)). That is:

– **Setup phase** The challenger samples the PRF keys $K_1^{(n)}, K_2, K_2'$ and creates the obfuscated program $i\mathcal{O}([\mathsf{Verify}])$, where the program Verify is defined as in Figure 1(a). The challenger sets $VK = i\mathcal{O}([\mathsf{Verify}])$ and passes it to $\mathcal{A}$.

– **Query Phase**

- Random Oracle Queries: For each random oracle query m_j, the challenger checks if m_j has already been queried.
  If yes, let $(m_j, u_j \| v_j)$ be the tuple corresponding to m_j. It sends $u_j \| v_j$ to $\mathcal{A}$.
  Else if $j < s$, the challenger chooses $v_j \leftarrow \{0,1\}^n$. It also chooses $K_3 \leftarrow \{0,1\}^{n_1}$, $e' \leftarrow \{0,1\}^{n_2}$, $p_j \leftarrow \{0,1\}^{\ell'}$ and computes $\hat{p}_j = f(p_j)$. It computes $K_3\{m_j\} \leftarrow \mathsf{Puncture}(K_3, m_j)$ and sets $g = i\mathcal{O}(P'_{K_3\{m_j\}, \hat{p}_j, m_j}; e')$ as defined in Figure 2(a). The challenger sets $(y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1}) := F_1^{(n)}(K_1^{(n)}, v_j)$ and $u_j[i] := y_{i,g_i}$ for all $i \in [1,n]$, where g_i is the i-th bit

of g. It sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to a table.
If $j = s$, the challenger chooses $v_j^* \leftarrow \{0,1\}^{n_1}$, $e \leftarrow \{0,1\}^{n_1}$ and $e' \leftarrow \{0,1\}^{n_2}$. It sets $g := i\mathcal{O}(P_e; e')$, $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) := F_1^{(n)}(K_1^{(n)}, v_j^*)$ and $u_j^*[i] = y_{i,g_i}^*$ for all $i \in [1,n]$, where g_i is the i-th bit of g. The challenger computes $u_s^* = u_s^*[1] \| \cdots \| u_s^*[n]$ and $K_2\{u_s^* \| v_s^*\} \leftarrow \mathsf{Puncture}(K_2, u_s^* \| v_s^*)$. It sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to the table.
If $j > s$, the challenger chooses $u_j \leftarrow \{0,1\}^{n^2}$, $v_j \leftarrow \{0,1\}^n$, sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to the table.

- Signing Queries: For each signature query m_k, the challenger first looks up the table to obtain $H(m_k) = u_k \| v_k$.
If $\exists j$ s.t. $m_k = m_j \ \wedge \ j < s$, it returns $\sigma_k = p_j$;
If $m_k = m_s$, it returns $\sigma_k = F_3(e, m_k)$;
Else if $\exists j$ s.t. $m_k = m_j \ \wedge \ j > s$, it computes $K_3 = F_2(K_2\{u_s^* \| v_s^*\}, u_k \| v_k)$ and returns $\sigma_k = F_3(K_3, m_k)$.

– **Forgery Phase** $\mathcal{A}$ finally outputs a forgery (m^*, σ^*) and wins if it holds $\mathsf{Verify}(VK, m^*, \sigma^*) = 1$.

Intuitively, $\mathsf{Hyb}_{s,10}$ is indistinguishable from $\mathsf{Hyb}_{s,9}$ by $i\mathcal{O}$ between the programs Verify 2 and Verify. The proof is given by Lemma 11 (Appendix).

$\mathsf{Hyb}_{s,11}$: This hybrid is identical to $\mathsf{Hyb}_{s,10}$, except that for the Random Oracle Query m_j *s.t.* $j = s$, the challenger chooses $v_j^* \leftarrow \{0,1\}^{n_1}$, $e \leftarrow \{0,1\}^{n_1}$ and $e' \leftarrow \{0,1\}^{n_2}$. It computes $e\{m_j\} \leftarrow \mathsf{Puncture}(e, m_j)$ and sets $g = i\mathcal{O}(P'_{e\{m_j\}, m_j, \hat{p}_j}; e')$ as described in Figure 2(a) where $\hat{p}_j = f(F_3(e, m_j))$ instead of setting $g = i\mathcal{O}(P_e; e')$. It also sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) := F_1^{(n)}(K_1^{(n)}, v_j^*)$ and $u_j^*[i] = y_{i,g_i}^*$ for all $i \in [1,n]$, where g_i is the i-th bit of g. The challenger computes $u_j^* = u_j^*[1] \| \cdots \| u_j^*[n]$ where $j = s$ and $K_2\{u_s^* \| v_s^*\} \leftarrow \mathsf{Puncture}(K_2, u_s^* \| v_s^*)$. It sends $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, u_j \| v_j)$ to the table. Intuitively, $\mathsf{Hyb}_{s,11}$ is indistinguishable from $\mathsf{Hyb}_{s,10}$ by $i\mathcal{O}$ between the programs $P'_{e\{m_j\}, m_j, \hat{p}_j}$ and P_e. The proof is given Lemma 12 (Appendix).

$\mathsf{Hyb}_{s,12}$: This hybrid is identical to $\mathsf{Hyb}_{s,11}$, except that for the random oracle query m_j *s.t.* $j = s$, the challenger sets $g = i\mathcal{O}(P'_{e\{m_j\}, m_j, \hat{p}_j}; e')$ as described in Figure 2(a), where $\hat{p}_j = f(p_j)$ for $p_j \leftarrow \{0,1\}^{\ell'}$ instead of setting $\hat{p}_j = f(F_3(e, m_j))$ and for the Signing Query m_k *s.t.* $m_k = m_s$, the challenger returns $\sigma_k = p_j$. Intuitively, $\mathsf{Hyb}_{s,12}$ is indistinguishable from $\mathsf{Hyb}_{s,11}$ by the security of the puncturable PRFs $F_3(e, \cdot)$. The proof is given by Lemma 13 (Appendix).

Lemma 1. *If f is a one-way function, then for all PPT $\mathcal{A}$ we have that*

$$\mathsf{Adv}_{\mathcal{A}}^{q_{RO},12} = \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}_{\mathcal{A}}^{q_{RO},12} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that, using $\mathcal{A}$, inverts the one-way function f with probability ϵ.

$\mathcal{B}$ receives $\hat{p}^*$ from the one-way function challenger. It samples the PRF keys $K_1^{(n)}, K_2, K_2'$ and creates the obfuscated program $i\mathcal{O}([\mathsf{Verify}])$ as in Figure 1(a) and padded to be of appropriate size. $\mathcal{B}$ sets $VK = i\mathcal{O}([\mathsf{Verify}])$ and passes it to the attacker $\mathcal{A}$. For a $\mathsf{Random\ Oracle}$ query m_j, where $j \in [q_{\mathsf{RO}}]$, $\mathcal{B}$ chooses $v_j \leftarrow \{0,1\}^n$, $K_3 \leftarrow \{0,1\}^{n_1}$, $e' \leftarrow \{0,1\}^{n_2}$, $p_j \leftarrow \{0,1\}^{\ell'}$ and computes $\hat{p}_j = f(p_j)$. $\mathcal{B}$ sets $g = i\mathcal{O}(P'_{K_3\{m_j\},m_j,\hat{p}_j}; e')$ as defined in Figure 2(a). $\mathcal{B}$ also sets $(y_{1,0}, y_{1,1}), \ldots, (y_{n,0}, y_{n,1}) := F_1^{(n)}(K_1^{(n)}, v_j)$ and $u_j[i] = y_{i,g_i}$ for all $i \in [1,n]$, where g_i is the i-th bit of g. $\mathcal{B}$ returns $u_j \| v_j$ to $\mathcal{A}$ and adds $(m_j, p_j, u_j \| v_j)$ to a table. For a $\mathsf{Signing}$ query m_k, $\mathcal{B}$ first looks up the table to obtain $(m_k, p_k, u_k \| v_k)$. We refer to all signatures generated by the $\mathsf{Signing}$ oracles, $\mathsf{Random\ Oracle}$ as intermediate steps in order to answer $\mathcal{A}$'s respective queries. Since $\exists j$ s.t. $m_k = m_j$, $\mathcal{B}$ returns $\sigma_k = p_j$. Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*). If $\mathsf{Verify}(VK, m^*, \sigma^*) = 1$, which means that, for $m = m^*$ it holds that $f(\sigma^*) = \hat{p}^*$, then $\mathcal{B}$ can successfully find an inverse σ^* for $\hat{p}^*$. Clearly, if $\mathcal{A}$ wins in $\mathsf{Hyb}_{q_{\mathsf{RO}},12}$, then $\mathcal{B}$ inverts the one-way function.

To conclude, Theorem 2 follows from the Lemmas 1- 13 (Appendix).

4 Analysis of Costs

In this section, we evaluate the cost of the Sahai-Waters signature [14] (selectively secure), Ramchen and Waters signature [13] (adaptively secure in the standard model) and our proposed signature (adaptively secure in the random oracle model) in terms of the computation of the puncturable PRFs involved in the signing algorithm, which can be constructed by a pseudorandom generator based on GGM [10] trees. We express the cost of the computation of puncturable PRFs involved in the signing algorithm of each scheme in terms of the underlying length-doubling and length-tripling PRGs.

Let g_D be the cost of the length-doubling PRG and g_T be the cost of the length-tripling PRG. We assume that the messages to be signed are ℓ-bits and the size of the image range of the hash function is $|H(\cdot)|$.

Sahai-Waters signature [14] This scheme makes a single call to the fixed-input-length puncturable PRF on an ℓ-bit message. This call traverses the GGM tree according to the message bits, requiring ℓ invocations of the length-doubling PRG. The cost is therefore $g_D \cdot \ell$.

Ramchen and Waters signature [13] This scheme calls the fixed-length puncturable PRF once on each of $\lambda + \lg \ell + 1$ inputs. Since each input has the same λ-bit suffix, the GGM tree can be first traversed to a depth of λ, and then a depth-first search is performed to an additional $\lg \ell + 1$ depth. Thus, $\lambda + 2(2^{\lg \ell} - 1) + 1 = \lambda + 2\ell - 1$ calls are made to the length-doubling PRG. In addition the scheme evaluates the variable-length puncturable PRF once on a λ-bit input, which requires λ calls to the length-tripling PRG. Therefore the total cost is $g_D \cdot (\lambda + 2\ell - 1) + g_T \cdot \lambda$.

Our signature scheme Our adaptively secure scheme makes a call to the puncturable PRF on an $|H(\cdot)|$-bits input and a call to the puncturable PRF on an ℓ-bit message. This call traverses the GGM tree according to the message bits, requiring $|H(\cdot)|$ invocations of the length-doubling PRG. The cost is therefore $g_D \cdot (|H(\cdot)| + \ell)$. Since the hash function is a one-way compression function, then it holds that $|H(\cdot)| < \ell$. Therefore, the total cost of our scheme is less than $g_D \cdot (2\ell)$, which is slightly more than the cost of the SW scheme and a lot less than the cost of RW scheme.

Table 1 (Section 1) summarises the comparison between our proposed scheme and the SW and RW schemes. We note that although the RW scheme is proven to be adaptively secure in the standard model, their proposal is quite heavy computationally. We have chosen to provide a more efficient (fast signing), adaptively secure solution suitable for resource-constrained devices at the cost of employing the random oracle model.

We note that, regarding the fact that RW scheme is proven to be adaptively secure in the standard model while our scheme is secure in the random oracle, the efficiency gain made by our scheme is outweighed by the loss in security.

5 Conclusion

In this paper, We explore the methods for achieving adaptively secure obfuscation-derived signatures. In particular, relying on iO and puncturable pseudorandom functions (PRFs) as well as one-way functions we present a signature scheme that is adaptively secure in the random oracle model.

References

1. Michel Abdalla, Dario Catalano, and Dario Fiore. Verifiable random functions from identity-based key encapsulation. In Advances in Cryptology-EUROCRYPT'09, volume 5479, pages 554-571, 2009.
2. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. Proceedings of CRYPTO 2001, Springer LNCS 2139:1-18, 2001.
3. Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, and Christoph Striecks. Confined guessing: New signatures from standard assumptions. Journal of Cryptology, pages 1-33, 2014.
4. Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. In Kenneth G. Paterson, editor, Advances in Cryptology EUROCRYPT 2011, volume 6632 of Lecture Notes in Computer Science, pages 89-108. Springer Berlin Heidelberg, 2011.
5. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, ASIACRYPT 2013, Part II, volume 8270 of LNCS, pages 280-300. Springer, December 2013.
6. Melissa Chase and Markulf Kohlweiss. A new hash-and-sign approach and structure-preserving signatures from DLIN. In Proceedings of the 8th International Conference on Security and Cryptography for Networks, SCN'12, pages 131-148, Berlin, Heidelberg, 2012. Springer-Verlag.

7. Dario Fiore and Dominique Schröder. Uniqueness is a different story: Impossibility of verifiable random functions from trapdoor permutations. In Ronald Cramer, editor, TCC 2012, volume 7194 of LNCS, pages 636-653, Taormina, Sicily, Italy, March 19-21, 2012. Springer, Berlin, Germany.
8. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In FOCS, 2013.
9. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, EUROCRYPT 2013, volume 7881 of LNCS, pages 1-17. Springer, May 2013.
10. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. Journal of the ACM, 33(4):792-807, October 1986.
11. Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, BrentWaters, and Mark Zhandry. How to generate and use universal parameters. Cryptology ePrint Archive, Report 2014/507, 2014. http://eprint.iacr.org/.
12. Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, Advances in Cryptology-CRYPTO 2009, volume 5677 of Lecture Notes in Computer Science, pages 654-670. Springer Berlin Heidelberg, 2009.
13. Ramchen, K., and Waters, B.. Fully secure and fast signing from obfuscation. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2014: pages 659-673.
14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In STOC, pages 475-484, 2014.

A Appendix

A.1 Indistinguishability Proofs between Hybrids

Let $\mathsf{Adv}^{i}_{\mathcal{A}}$ denote the advantage of an adversary $\mathcal{A}$ in Hyb_i. We now establish via a sequence of Lemmas that the difference of the attacker $\mathcal{A}$'s advantage between each adjacent hybrid is negligible.

Observation 1 *For any PPT adversary $\mathcal{A}$,*

$$Adv^{0}_{\mathcal{A}} = Adv^{0,12}_{\mathcal{A}}.$$

Proof. Hyb_0 and $\mathsf{Hyb}_{0,12}$ are identical by inspection.

Lemma 2. *If $i\mathcal{O}$ is a secure indistinguishability obfuscator, then for $s \in [q_{RO}]$, all PPT $\mathcal{A}$ we have that*

$$Adv^{s-1,12}_{\mathcal{A}} - Adv^{s,1}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscation. Suppose there exists a PPT adversary $\mathcal{A}$ such that

$\mathsf{Adv}_{\mathcal{A}}^{s-1,12} - \mathsf{Adv}_{\mathcal{A}}^{s,1} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ using $\mathcal{A}$.

We describe and analyze a PPT reduction algorithm $\mathcal{B}$ that plays the indistinguishability obfuscation security game with $\mathcal{A}$. First, $\mathcal{B}$ samples the PRF keys $K_1^{(n)}, K_2, K_2'$. It also chooses $v_s^* \leftarrow \{0,1\}^n$, $u_s^* \leftarrow \{0,1\}^{n^2}$ and sets $(y_{1,0}^*, y_{1,1}^*), \ldots, (y_{n,0}^*, y_{n,1}^*) = F_1^{(n)}(K_1^{(n)}, v_s^*)$. for all $b \in \{0,1\}$ and $i \in [1,n]$, $\mathcal{B}$ sets $z_{i,b}^* = \mathsf{PRG}(y_{i,b}^*)$. Next, $\mathcal{B}$ creates the program $C_0 = \mathsf{Verify}$ as described in Figure 1(a) and the program $C_1 = \mathsf{Verify\ 2}$ as described in Figure 2(b). It submits both of these programs to the $i\mathcal{O}$ challenger and receives back a program C which is passed to the attacker $\mathcal{A}$ as the verification key VK. For the $\mathsf{Random\ Oracle}$ and the $\mathsf{Signing}$ queries, $\mathcal{B}$ can simulate perfectly as in $\mathsf{Hyb}_{s-1,12}$ (which is the same in $\mathsf{Hyb}_{s-1,12}$ and $\mathsf{Hyb}_{s,1}$). Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

It is easy to observe that the programs $C_0 = \mathsf{Verify}$ and $C_1 = \mathsf{Verify\ 2}$ are functionally equivalent for inputs $v \neq v_s^*$. Moreover, even on input $v = v_s^*$ such that $(z_{1,0}^*, z_{1,1}^*), \ldots, (z_{n,0}^*, z_{n,1}^*) = \mathsf{PRG}(F_1^{(n)}(K_1^{(n)}, v_s^*))$, the functionality of both programs is identical if the PRG is injective.

We observe that when C is generated as an obfuscation of C_0, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s-1,12}$ to $\mathcal{A}$. Otherwise if C is generated as an obfuscation of C_1 the view is of $\mathsf{Hyb}_{s,1}$. Therefore, if $\mathsf{Adv}_{\mathcal{A}}^{s-1,12} - \mathsf{Adv}_{\mathcal{A}}^{s,1}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the indistinguishability obfuscation game.

Lemma 3. *If $F_1^{(n)}(K_1^{(n)}, \cdot)$ is a secure puncturable PRF, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$\mathsf{Adv}_{\mathcal{A}}^{s,1} - \mathsf{Adv}_{\mathcal{A}}^{s,2} = \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}_{\mathcal{A}}^{s,1} - \mathsf{Adv}_{\mathcal{A}}^{s,2} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the selective security of the punctured PRFs $F_1^{(n)}(K_1^{(n)}, \cdot)$ (at least one of the punctured PRFs in the sequence $F_1^{(n)}(K_1^{(n)}, \cdot)$) using $\mathcal{A}$.

Consider a sequence of $2n+1$ sub-hybrids, where the i-th sub-hybrid $\mathsf{Hyb}_{s,1,i}$, is the same as $\mathsf{Hyb}_{s,1}$ except that:

- For $i \leq n$, $\forall j \leq i$, $y_{j,0}^* \leftarrow \{0,1\}^n$. Also $\forall n \geq j > i$, $y_{j,0}^* = F_1(K_1^{j,0}, v_s^*)$ and $\forall j \leq n$, $y_{j,1}^* = F_1(K_1^{j,1}, v_s^*)$.
- For $i > n$, $\forall j \leq n$, $y_{j,0}^* \leftarrow \{0,1\}^n$. Also $\forall n < j \leq i$, $y_{j-n,1}^* \leftarrow \{0,1\}^n$ and $\forall j > i$, $y_{j-n,1}^* = F_1(K_1^{(j-n),1}, v_s^*)$.

Note that $\mathsf{Hyb}_{s,1,0} \equiv \mathsf{Hyb}_{s,1}$ and $\mathsf{Hyb}_{s,1,2n} \equiv \mathsf{Hyb}_{s,2}$.

Then, there exists some $i \in [0, 2n-1]$ such that $\mathsf{Adv}_{\mathcal{A}}^{s,1,i} - \mathsf{Adv}_{\mathcal{A}}^{s,1,i+1} = \frac{1}{2n} \cdot \epsilon$.

Assume without loss of generality that $i \leq n$ (arguments for $i > n$ will follow similarly), then we can construct a PPT algorithm $\mathcal{B}$ that breaks the selective security of the punctured $F_1(K_1^{i+1,0}, \cdot)$ by using $\mathcal{A}$. $\mathcal{B}$ chooses $v_s^* \leftarrow \{0,1\}^n$,

submits it to the PRF challenger and receives the punctured PRF key $K_1^{i+1,0}\{v_s^*\}$ and the challenge y, which is either chosen uniformly at random or is the output of the $F_1(K_1^{i+1,0}, \cdot)$ at v_s^*. $\mathcal{B}$ samples the PRF keys $\{(K_1^{1,0}, K_1^{1,1}), \ldots, (K_1^{i,0}, K_1^{i,1}), (K_1^{i+2,0}, K_1^{i+2,1}), \ldots, (K_1^{n,0}, K_1^{n,1})\}, K_2, K_2'$. $\mathcal{B}$ also chooses $u_s^* \leftarrow \{0,1\}^{n^2}$. For $j \in [1,i]$, $\mathcal{B}$ chooses $y_{j,0}^* \leftarrow \{0,1\}^n$ and sets $y_{i+1,0}^* := y$; for $j \in [i+2,n]$, it sets $y_{j,0}^* := F_1(K_1^{j,0}, v_s^*)$; and for $j \in [1,n]$, it sets $y_{j,1}^* := F_1(K_1^{j,1}, v_s^*)$. For all $b \in \{0,1\}$, $\mathcal{B}$ sets $z_{j,b}^* = \mathsf{PRG}(y_{j,b}^*)$ for $j \in [1,n]$. Next, $\mathcal{B}$ creates the obfuscated program $i\mathcal{O}(\mathsf{Verify\ 2})$ as described in Figure 2(b) and passes it to the attacker $\mathcal{A}$ as verification key VK. For the Random Oracle and Signing queries, $\mathcal{B}$ can simulate perfectly as in $\mathsf{Hyb}_{s,1,i}$ (which is the same in $\mathsf{Hyb}_{s,1,i}$ and $\mathsf{Hyb}_{s,1,i+1}$). Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

We observe that when y is generated as $F_1(K_1^{i+1,0}, v_s^*)$, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s,1,i}$ to $\mathcal{A}$. Otherwise if y is chosen randomly, then $\mathcal{B}$ gives the view of $\mathsf{Hyb}_{s,1,i+1}$. Therefore, if $\mathsf{Adv}_{\mathcal{A}}^{s,1} - \mathsf{Adv}_{\mathcal{A}}^{s,2}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the security of the punctured PRF in the sequence $F_1^{(n)}(K_1^{(n)}, \cdot)$.

Lemma 4. *If PRG is a secure PRG, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$ we have that:*

$$\mathsf{Adv}_{\mathcal{A}}^{s,2} - \mathsf{Adv}_{\mathcal{A}}^{s,3} = \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}_{\mathcal{A}}^{s,2} - \mathsf{Adv}_{\mathcal{A}}^{s,3} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of the PRG by using $\mathcal{A}$.

Consider a sequence of $2n+1$ sub-hybrids, where the i-th sub-hybrid $\mathsf{Hyb}_{s,2,i}$, is the same as $\mathsf{Hyb}_{s,2}$ except that:

- For $i \leq n$, $\forall j \leq i$, $z_{j,0}^* \leftarrow \{0,1\}^{2n}$. Also $\forall n \geq j > i$, $z_{j,0}^* = \mathsf{PRG}(y_{j,0}^*)$ for $y_{j,0}^* \leftarrow \{0,1\}^n$ and $\forall j \leq n$, $z_{j,1}^* = \mathsf{PRG}(y_{j,1}^*)$ for $y_{j,1}^* \leftarrow \{0,1\}^n$.
- For $i > n$, $\forall j \leq n$, $z_{j,0}^* \leftarrow \{0,1\}^{2n}$. Also $\forall n < j \leq i$, $z_{j-n,1}^* \leftarrow \{0,1\}^{2n}$ and $\forall j > i$, $z_{j-n,1}^* = \mathsf{PRG}(y_{j-n,1}^*)$ for $y_{j-n,1}^* \leftarrow \{0,1\}^n$.

Note that $\mathsf{Hyb}_{s,2,0} \equiv \mathsf{Hyb}_{s,2}$ and $\mathsf{Hyb}_{s,2,2n} \equiv \mathsf{Hyb}_{s,3}$.

Then, there exists some $i \in [0, 2n-1]$ such that $\mathsf{Adv}_{\mathcal{A}}^{s,2,i} - \mathsf{Adv}_{\mathcal{A}}^{s,2,i+1} = \frac{1}{2n} \cdot \epsilon$.

Assume without loss of generality that $i \leq n$ (arguments for $i > n$ will follow similarly), then we can construct a PPT algorithm $\mathcal{B}$ that breaks the security of the PRG by using $\mathcal{A}$. On receiving a PRG challenge z which is either chosen uniformly at random from $\{0,1\}^{2n}$ or is the output of $\mathsf{PRG}(y_{i+1,0}^*)$ for $y_{i+1,0}^* \leftarrow \{0,1\}^n$, $\mathcal{B}$ samples the PRF keys $K_1^{(n)}, K_2, K_2'$. It also chooses $v_s^* \leftarrow \{0,1\}^n$, $u_s^* \leftarrow \{0,1\}^{n^2}$. For $\forall j \in [1,i]$, set $z_{j,0}^* \leftarrow \{0,1\}^{2n}$, $z_{i+1,0}^* = z$ and $\forall j \in [i+2,n]$, set $z_{j,0}^* = \mathsf{PRG}(y_{j,0}^*)$ for $y_{j,0}^* \leftarrow \{0,1\}^n$. For $\forall j \in [1,n]$, set $z_{j,1}^* = \mathsf{PRG}(y_{j,1}^*)$ where $y_{j,1}^* \leftarrow \{0,1\}^n$. Next, $\mathcal{B}$ creates the obfuscated program $i\mathcal{O}(\mathsf{Verify\ 2})$ as described in Figure 2(b) and passes it to the attacker $\mathcal{A}$ as verification key VK. For the Random Oracle and Signing queries, $\mathcal{B}$ can simulate perfectly as

in $\mathsf{Hyb}_{s,2,i}$ (which is the same in $\mathsf{Hyb}_{s,2,i}$ and $\mathsf{Hyb}_{s,2,i+1}$). Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

We observe that when z is generated as $\mathsf{PRG}(y^*_{i+1,0})$ for $y^*_{i+1,0} \leftarrow \{0,1\}^n$, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s,2,i}$ to $\mathcal{A}$. Otherwise if z is chosen randomly from $\{0,1\}^{2n}$, then $\mathcal{B}$ gives gives the view of $\mathsf{Hyb}_{s,2,i+1}$. Therefore, if $\mathsf{Adv}^{s,2}_{\mathcal{A}} - \mathsf{Adv}^{s,3}_{\mathcal{A}}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the security of PRG.

Lemma 5. *If $i\mathcal{O}$ is a secure indistinguishability obfuscator, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$ we have that:*

$$Adv^{s,3}_{\mathcal{A}} - Adv^{s,4}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscation. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^{s,3}_{\mathcal{A}} - \mathsf{Adv}^{s,4}_{\mathcal{A}} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ using $\mathcal{A}$.

We describe and analyze a PPT reduction algorithm $\mathcal{B}$ that plays the indistinguishability obfuscation security game with $\mathcal{A}$. First, $\mathcal{B}$ samples the PRF keys $K^{(n)}_1, K_2, K'_2$. It also chooses $v^*_s \leftarrow \{0,1\}^n$, $u^*_s \leftarrow \{0,1\}^{n^2}$ and $z^*_{i,b} \leftarrow \{0,1\}^{2n}$ for all $b \in \{0,1\}$, $i \in [1,n]$. It computes $e = F_2(K_2, u^*_s \| v^*_s)$ and $e' = F'_2(K'_2, u^*_s \| v^*_s)$ and sets $g = i\mathcal{O}(P_e; e')$. Next, $\mathcal{B}$ creates a program $C_0 = \mathsf{Verify\ 2}$ as described in Figure 2(b) and a program $C_1 = \mathsf{Verify\ 3}$ as described in Figure 3. It submits both of these programs to the $i\mathcal{O}$ challenger and receives back a program C which is passed to the attacker $\mathcal{A}$ as verification key VK. For the Random Oracle and Signing queries, $\mathcal{B}$ can simulate perfectly as in $\mathsf{Hyb}_{s,3}$ (which is the same in $\mathsf{Hyb}_{s,3}$ and $\mathsf{Hyb}_{s,4}$). Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

It is easy to observe that the programs $C_0 = \mathsf{Verify\ 2}$ and $C_1 = \mathsf{Verify\ 3}$ are functionally equivalent on all inputs other than $u^*_s \| v^*_s$. Moreover, on input v^*_s, note that the condition in Step 1 is never satisfied in $\mathsf{Verify\ 2}$ except with probability 2^{-n}, since $z^*_{i,b}$ is chosen at random. Therefore, the output of $\mathsf{Verify\ 2}$ on input $u^*_s \| v^*_s$ is $P = i\mathcal{O}(P_{K_3}; r')$ of the program P_{K_3} (described in Figure 1(b)) using randomness r', where $K_3 = F_2(K_2, u^*_s \| v^*_s)$, $r' = F'_2(K'_2, u^*_s \| v^*_s)$. On input $u^*_s \| v^*_s$, the output of $\mathsf{Verify\ 3}$ (which is g) is the same as that of $\mathsf{Verify\ 2}$.

We observe that when C is generated as an obfuscation of C_0, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s,3}$ to $\mathcal{A}$. Otherwise if C is chosen as an obfuscation of C_1, then $\mathcal{B}$ gives the view of $\mathsf{Hyb}_{s,4}$. Therefore if $\mathsf{Adv}^{s,3}_{\mathcal{A}} - \mathsf{Adv}^{s,4}_{\mathcal{A}}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the indistinguishability obfuscation game.

Lemma 6. *If $F_2(K_2, \cdot)$ is a secure puncturable PRF, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$Adv^{s,4}_{\mathcal{A}} - Adv^{s,5}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^{s,4}_{\mathcal{A}} - \mathsf{Adv}^{s,5}_{\mathcal{A}} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the selective security of the punctured PRFs $F_2(K_2, \cdot)$ using $\mathcal{A}$.

First, $\mathcal{B}$ chooses $v^*_s \leftarrow \{0,1\}^n$, $u^*_s \leftarrow \{0,1\}^{n^2}$ and $z^*_{i,b} \leftarrow \{0,1\}^{2n}$ for all $b \in \{0,1\}$, $i \in [1,n]$. $\mathcal{B}$ submits $u^*_s \| v^*_s$ to the PRF challenger and receives the punctured PRF key $K_2\{u^*_s \| v^*_s\}$ and the challenge K, which is either chosen uniformly at random or is the output of the $F_2(K_2, \cdot)$ at $u^*_s \| v^*_s$. $\mathcal{B}$ samples the PRF keys $K^{(n)}_1, K'_2$. It sets $e = K$, $e' = F'_2(K'_2, u^*_s \| v^*_s)$ and $g = i\mathcal{O}(P_e; e')$. Next, $\mathcal{B}$ creates the obfuscated program $i\mathcal{O}(\mathsf{Verify\ 3})$ as described in Figure 3 and passes it to the attacker $\mathcal{A}$ as verification key VK. For the Random Oracle and Signing queries, $\mathcal{B}$ can simulate perfectly as in $\mathsf{Hyb}_{s,4}$ (which is the same in $\mathsf{Hyb}_{s,4}$ and $\mathsf{Hyb}_{s,5}$). Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

We observe that when K is generated as $F_2(K_2, u^*_s \| v^*_s)$, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s,4}$ to $\mathcal{A}$. Otherwise if K is chosen randomly, the view is of $\mathsf{Hyb}_{s,5}$. Therefore, if $\mathsf{Adv}^{s,4}_{\mathcal{A}} - \mathsf{Adv}^{s,5}_{\mathcal{A}}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the security of the punctured PRF F_2.

Lemma 7. *If $F'_2(K'_2, \cdot)$ is a secure puncturable PRF, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$Adv^{s,5}_{\mathcal{A}} - Adv^{s,6}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

The proof of this lemma is analogous to that of Lemma 5.

Lemma 8. *If $i\mathcal{O}$ is a secure indistinguishability obfuscator, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$Adv^{s,6}_{\mathcal{A}} - Adv^{s,7}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscation. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^{s,6}_{\mathcal{A}} - \mathsf{Adv}^{s,7}_{\mathcal{A}} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ using $\mathcal{A}$.

We describe and analyze a PPT reduction algorithm $\mathcal{B}$ that plays the indistinguishability obfuscation security game with $\mathcal{A}$. First, $\mathcal{B}$ samples the PRF keys $K^{(n)}_1, K_2, K'_2$. It also chooses $v^*_s \leftarrow \{0,1\}^n$, $e \leftarrow \{0,1\}^{n_1}$ and $e' \leftarrow \{0,1\}^{n_2}$ and sets $g = i\mathcal{O}(P_e; e')$. For all $i \in [1,n]$, it sets $y^*_{i,g_i} \leftarrow \{0,1\}^n$, $u^*_s[i] = y^*_{i,g_i}$, $z^*_{i,g_i} = \mathsf{PRG}(y^*_{i,g_i})$ and $z^*_{i,\bar{g}_i} \leftarrow \{0,1\}^{2n}$, where g_i is the i-th bit of g and $\bar{g}_i = 1 - g_i$. Next, $\mathcal{B}$ creates a program $C_0 = \mathsf{Verify\ 3}$ as described in Figure 3 and program $C_1 = \mathsf{Verify\ 2}$ as described in Figure 2(b). It submits both of these to the $i\mathcal{O}$ challenger and receives back a program C, which is passed to the attacker $\mathcal{A}$ as verification key VK. For the Random Oracle and Signing queries, $\mathcal{B}$ can simulate perfectly as in $\mathsf{Hyb}_{s,6}$ (which is the same in $\mathsf{Hyb}_{s,6}$ and $\mathsf{Hyb}_{s,7}$). Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

It is easy to observe that the programs $C_0 = \mathsf{Verify\ 3}$ and $C_1 = \mathsf{Verify\ 2}$ are functionally equivalent for $v \neq v_s^*$ and for $(u = u_s^*, v = v_s^*)$. Moreover, we note that, for input $(u \neq u_s^*, v = v_s^*)$, since $z_{i,b}^*$ are chosen uniformly at random, the condition in step 2 is possibly satisfied in the program $\mathsf{Verify\ 3}$ (that is C_0) with probability only 2^{-n} by security of the length-doubling PRG. Therefore, the output of $\mathsf{Verify\ 3}$ on input $(u \neq u_s^*, v = v_s^*)$ is $P = i\mathcal{O}(P_{K_3}; r')$ of the program P_{K_3} of Figure 2 using randomness r', where $K_3 = F_2(K_2\{u_s^* \| v_s^*\}, u \| v_s^*)$, $r' = F_2'(K_2'\{u_s^* \| v_s^*\}, u \| v *_s)$. And in program $\mathsf{Verify\ 2}$ (that is C_1), for input $(u \neq u_s^*, v = v_s^*)$, the condition in step 1 will only be satisfied with probability 2^{-n} by the security of the length-doubling PRG and by injectivity of the PRG. The output of $\mathsf{Verify\ 2}$ on input $(u \neq u_s^*, v = v_s^*)$ is therefore the same as that of $\mathsf{Verify\ 3}$.

We observe that when C is generated as an obfuscation of C_0, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s,6}$ to $\mathcal{A}$. Otherwise if C is chosen as an obfuscation of C_1 the view is of $\mathsf{Hyb}_{s,7}$. Therefore if $\mathsf{Adv}_{\mathcal{A}}^{s,6} - \mathsf{Adv}_{\mathcal{A}}^{s,7}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the indistinguishability obfuscation game.

Lemma 9. *If PRG is a secure PRG, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$Adv_{\mathcal{A}}^{s,7} - Adv_{\mathcal{A}}^{s,8} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}_{\mathcal{A}}^{s,7} - \mathsf{Adv}_{\mathcal{A}}^{s,8} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of the PRG by using $\mathcal{A}$.

Consider a sequence of $n+1$ sub-hybrids, such that for $i \in [0, n]$ the i-th sub-hybrid $\mathsf{Hyb}_{s,7,i}$, is the same as $\mathsf{Hyb}_{s,7}$ except that:

- For $\forall j \leq i$, $z_{j,\bar{g}_j}^* = \mathsf{PRG}(y_{j,\bar{g}_j}^*)$, where $y_{j,\bar{g}_j}^* \leftarrow \{0,1\}^n$.
- For $\forall j > i$, $z_{j,\bar{g}_j}^* \leftarrow \{0,1\}^{2n}$.

Note that $\mathsf{Hyb}_{s,7,0} \equiv \mathsf{Hyb}_{s,7}$ and $\mathsf{Hyb}_{s,7,n} \equiv \mathsf{Hyb}_{s,8}$.

Then, there exists some $i \in [0, n-1]$ such that $\mathsf{Adv}_{\mathcal{A}}^{s,7,i} - \mathsf{Adv}_{\mathcal{A}}^{s,7,i+1} = \frac{1}{n} \cdot \epsilon$.

We can construct a PPT algorithm $\mathcal{B}$ that breaks the security of the PRG by using $\mathcal{A}$. On receiving a PRG challenge z which is either chosen uniformly at random from $\{0,1\}^{2n}$ or is the output of $\mathsf{PRG}(y_{i+1,\bar{g}_{i+1}}^*)$ for $y_{i+1,\bar{g}_{i+1}}^* \leftarrow \{0,1\}^n$, $\mathcal{B}$ samples the PRF keys $K_1^{(n)}, K_2, K_2'$. It also chooses $v_s^* \leftarrow \{0,1\}^n$, $e \leftarrow \{0,1\}^{n_1}$ and $e' \leftarrow \{0,1\}^{n_2}$. It sets $g = i\mathcal{O}(P_e; e')$. Let g_j is the j-th bit of g and $\bar{g}_j = 1 - g_j$. For all $j \in [1, n]$, it sets $y_{j,g_j}^* \leftarrow \{0,1\}^n$, $u_s^*[j] = y_{j,g_j}^*$, $z_{j,g_j}^* = \mathsf{PRG}(y_{j,g_j}^*)$. For all $j \neq i+1$, it sets $z_{j,\bar{g}_j}^* = \mathsf{PRG}(y_{j,\bar{g}_j}^*)$ where $y_{j,\bar{g}_j}^* \leftarrow \{0,1\}^n$ and $z_{i+1,\bar{g}_{i+1}}^* = z$. Next, $\mathcal{B}$ creates the obfuscated program $i\mathcal{O}(\mathsf{Verify\ 2})$ as described in Figure 2(b) and passes it to the attacker $\mathcal{A}$ as verification key VK. For the Random Oracle and Signing queries, $\mathcal{B}$ can simulate perfectly as in $\mathsf{Hyb}_{s,7,i}$ (which is the same in $\mathsf{Hyb}_{s,7,i}$ and $\mathsf{Hyb}_{s,7,i+1}$). Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

We observe that when z is generated as $\mathsf{PRG}(y^*_{i+1,\bar{g}_{i+1}})$ for $y^*_{i+1,\bar{g}_{i+1}} \leftarrow \{0,1\}^n$, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s,7,i+1}$ to $\mathcal{A}$. Otherwise if z is chosen randomly from $\{0,1\}^{2n}$, the view is of $\mathsf{Hyb}_{s,7,i}$. Therefore, if $\mathsf{Adv}^{s,7}_{\mathcal{A}} - \mathsf{Adv}^{s,8}_{\mathcal{A}}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the security of the PRG.

Lemma 10. *If $F_1^{(n)}(K_1^{(n)}, \cdot)$ is a secure puncturable PRF, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$Adv^{s,8}_{\mathcal{A}} - Adv^{s,9}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

The proof of this lemma is analogous to that of Lemma 2.

Lemma 11. *If $i\mathcal{O}$ is a secure indistinguishability obfuscator, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$Adv^{s,9}_{\mathcal{A}} - Adv^{s,10}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

The proof of this lemma is analogous to that of Lemma 1.

Lemma 12. *If $i\mathcal{O}$ is a secure indistinguishability obfuscator, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$Adv^{s,10}_{\mathcal{A}} - Adv^{s,11}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

Proof. We prove this lemma by giving a reduction to the indistinguishability security of the obfuscation. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^{s,10}_{\mathcal{A}} - \mathsf{Adv}^{s,11}_{\mathcal{A}} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the security of $i\mathcal{O}$ using $\mathcal{A}$.

We describe and analyze a PPT reduction algorithm $\mathcal{B}$ that plays the indistinguishability obfuscation security game with $\mathcal{A}$. First, $\mathcal{B}$ samples the PRF keys $K_1^{(n)}, K_2, K_2'$ and creates the obfuscated program $i\mathcal{O}([\mathsf{Verify}])$, where the program Verify is described in Figure 1(a) and padded to be of appropriate size. The challenger sets $VK = i\mathcal{O}([\mathsf{Verify}])$ and passes it to the attacker $\mathcal{A}$. For the Random Oracle Query m_j, if $j = s$, $\mathcal{B}$ chooses $v_j^* \leftarrow \{0,1\}^n$, $e \leftarrow \{0,1\}^{n_1}$ and $e' \leftarrow \{0,1\}^{n_2}$ and computes $\hat{p}_j = f(F_3(e, m_j))$. $\mathcal{B}$ creates a program $C_0 = P_e$ as described in Figure 1(b) and a program $C_1 = P'_{e\{m_j\},m_j,\hat{p}_j}$ as described in Figure 2(a). It submits both of these to the $i\mathcal{O}$ challenger and receives back a program C. $\mathcal{B}$ sets $g = C$. For all $i \in [1,n]$, it sets $(y^*_{1,0}, y^*_{1,1}), \ldots, (y^*_{n,0}, y^*_{n,1}) = F_1^{(n)}(K_1^{(n)}, v_j^*)$, $u_j^*[i] = y^*_{i,g_i}$, where g_i is the i-th bit of g. For the Random Oracle Query m_j, *s.t.* $j \neq s$, $\mathcal{B}$ can simulate perfectly as in $\mathsf{Hyb}_{s,10}$ (which is the same in $\mathsf{Hyb}_{s,10}$ and $\mathsf{Hyb}_{s,11}$). For Signing queries, $\mathcal{B}$ can simulate perfectly as in

$\mathsf{Hyb}_{s,10}$ (which is the same in $\mathsf{Hyb}_{s,10}$ and $\mathsf{Hyb}_{s,11}$). Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

It is easy to observe that the programs $C_0 = P_e$ (as described in Figure 1(b)) and program $C_1 = P'_{e\{m_j\},m_j,\hat{p}_j}$ (as described in Figure 2(a)) are functionally equivalent. We note that when C is generated as an obfuscation of C_0, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s,10}$ to $\mathcal{A}$. Otherwise, if C is chosen as an obfuscation of C_1 the view is of $\mathsf{Hyb}_{s,11}$. Therefore, if $\mathsf{Adv}^{s,10}_{\mathcal{A}} - \mathsf{Adv}^{s,11}_{\mathcal{A}}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the indistinguishability obfuscation game.

Lemma 13. *If $F_3(e, \cdot)$ is a secure puncturable PRF, then for $s \in [q_{RO}]$ and all PPT $\mathcal{A}$, we have that:*

$$Adv^{s,11}_{\mathcal{A}} - Adv^{s,12}_{\mathcal{A}} = negl(\lambda)$$

for some negligible function $negl(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}^{s,11}_{\mathcal{A}} - \mathsf{Adv}^{s,12}_{\mathcal{A}} = \epsilon$. We will construct a PPT algorithm $\mathcal{B}$ that breaks the selective security of the punctured PRFs $F_3(e, \cdot)$ using $\mathcal{A}$.

First, $\mathcal{B}$ samples the PRF keys $K^{(n)}_1, K_2, K'_2$ and creates the obfuscated program $i\mathcal{O}([\mathsf{Verify}])$ as described in Figure 1(a) and padded to be of appropriate size. $\mathcal{B}$ sets $VK = i\mathcal{O}([\mathsf{Verify}])$ and passes it to the attacker $\mathcal{A}$. For the Random Oracle query m_j, if $j = s$, $\mathcal{B}$ submits m_j to the PRF challenger and receives the punctured PRF key $e\{m_j\}$ and the challenge p_j, which is either chosen uniformly at random or is the output of the $F_3(e, \cdot)$ at m_j. $\mathcal{B}$ chooses $v^*_j \leftarrow \{0,1\}^n$ and $e' \leftarrow \{0,1\}^{n_2}$ and sets $\hat{p}_j = f(p_j)$. $\mathcal{B}$ sets $g = i\mathcal{O}(P'_{e\{m_j\},m_j,\hat{p}_j})$. For all $i \in [1, n]$, it sets $(y^*_{1,0}, y^*_{1,1}), \ldots, (y^*_{n,0}, y^*_{n,1}) = F^{(n)}_1(K^{(n)}_1, v^*_j)$, $u^*_j[i] = y^*_{i,g_i}$, where g_i is the i-th bit of g. For the Random Oracle query m_j, $s.t.\ j \neq s$, $\mathcal{B}$ can simulate perfectly as in $\mathsf{Hyb}_{s,11}$ (which is the same in $\mathsf{Hyb}_{s,11}$ and $\mathsf{Hyb}_{s,12}$). For the signing queries, $\mathcal{B}$ can simulate perfectly. Finally, $\mathcal{A}$ outputs a forgery (m^*, σ^*).

We observe that when p_j is generated as $F_3(e, m_j)$, then $\mathcal{B}$ gives exactly the view of $\mathsf{Hyb}_{s,11}$ to $\mathcal{A}$. Otherwise if p_j is chosen randomly, the view is of $\mathsf{Hyb}_{s,12}$. Therefore, if $\mathsf{Adv}^{s,11}_{\mathcal{A}} - \mathsf{Adv}^{s,12}_{\mathcal{A}}$ is non-negligible, $\mathcal{B}$ must also have non-negligible advantage against the security of the punctured PRF $F_3(e, \cdot)$.