Leakage-Resilient Cryptography from Puncturable Primitives and Obfuscation

Yu Chen * Yuyu Wang † Hong-sheng Zhou ‡

Abstract

In this work, we develop a framework for building leakage-resilient cryptosystems in the bounded leakage model from puncturable primitives and indistinguishability obfuscation ($i\mathcal{O}$). Our main results are as follows:

- We build leakage-resilient weak pseudorandom functions (PRFs) from weak puncturable PRFs and $i\mathcal{O}$, which readily imply leakage-resilient secret-key encryption.
- We build leakage-resilient publicly evaluable pseudorandom functions (PEPRFs) from puncturable PEPRFs and $i\mathcal{O}$, which readily imply leakage-resilient key encapsulation mechanism and thus public-key encryption. As a building block of independent interest, we realize puncturable PEPRFs from either new puncturable objects including puncturable trapdoor functions and puncturable extractable hash proof systems or existing puncturable PRFs with $i\mathcal{O}$.
- We construct the first leakage-resilient public-coin signature from selective puncturable PRFs, leakage-resilient one-way functions and $i\mathcal{O}$. This settles the open problem posed by Boyle, Segev and Wichs (Eurocrypt 2011).

By further assuming the existence of lossy functions, all the above constructions can achieve optimal leakage rate of $1 - o(1)$. Such a leakage rate is not known to be achievable for weak PRFs, PEPRFs and public-coin signatures before.

*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences. State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, China. School of Cyber Security, University of Chinese Academy of Sciences. Email: chenyu@iie.ac.cn

†Tokyo Institute of Technology, IOHK, AIST. Email: wang.y.ar@m.titech.ac.jp

‡Virginia Commonwealth University. Email: hongsheng.zhou@gmail.com

Contents

1 Introduction

A main line in cryptography is to design cryptosystems in security models that capture a wide range of possible attacks. Based on the idealized assumption that software/hardware implementations of cryptosystems perfectly hide the internal secrets, traditional security models (following the seminal work of Goldwasser and Micali [GM84]) only give an adversary "black-box" access to cryptosystems. However, advancements of cryptanalysis indicate that such an idealized assumption is false in real world: an adversary can launch a variety of *key leakage attacks* (such as [Koc96, BDL97, BS97, KJJ99, HSH+08]) to get some partial information about secret keys.

To thwart key leakage attacks in a systematic manner, the research community has paid extensive efforts on the design of provably secure leakage-resilient cryptosystems in the last decade, spreading from basic primitives (including one-way functions, pseudorandom functions, message authentication codes, encryptions, and signatures) to advanced protocols (including identifications, authenticated key agreements, and zero-knowledge proof systems).

Leakage models. Briefly speaking, leakage models are defined by strengthening standard models with a leakage oracle $\mathcal{O}_{\mathsf{leak}}(\cdot)$, from which an adversary can (adaptively) specify a series of *leakage functions* $f_i : \{0,1\}^* \to \{0,1\}^{\ell_i}$ and learn the result of f_i applied to the internal secret state. Over the years, several leakage models have been proposed in the literature, differing in the specifications of f_i. In this work we focus on a simple yet general model called *bounded leakage* model, introduced by Akavia et al. [AGV09]. In the bounded leakage model, all secrets in memory are subject to leakage, i.e., the input of f_i could be entire secret key sk, while f_i could be arbitrary subjected to the natural restriction that $\sum_i \ell_i$ is bounded by some parameter ℓ, called the leakage bound. The leakage rate is defined as the ratio of ℓ to the secret key size $|sk|$, i.e., $\ell/|sk|$. Obviously, the optimal leakage rate is $1 - o(1)$ since otherwise the adversary can trivially learn the entire secret via querying $\mathcal{O}_{\mathsf{leak}}(\cdot)$.

To date, the bounded leakage model has been widely adopted in many works [NS09, KV09, CDRW10, BG10, GKPV10, HL11, BK12, BCH12]. The results from the bounded leakage model are usually used as building blocks for leakage-resilient schemes in more complex leakage models.

Approach towards leakage resilience. From the perspective of provable security, the main technical hurdle to achieve leakage-resilience is that the reduction must be able to handle leakage queries w.r.t. arbitrary functions chosen from $\mathcal{L}$, where $\mathcal{L}$ is the ensemble of admissible leakage functions. This seemingly stipulates that the reduction should know the secret key while typically this is not the case because the underlying intractable problems is usually embedded in the secret key. This intuition has been formalized as "useless attacker paradox" in [Wic13]. Prior works overcome this paradox by taking the following two approaches.

One approach is directly resorting to *leakage-resilient* assumptions (which might be well packed as advanced assumptions). Following this approach, the reduction can easily handle leakage queries by simply forwarding them to its own challenger. Goldwasser et al. [GKPV10] proved that the LWE assumption itself is leakage-resilient and then built a leakage-resilient secret-key encryption from it. Akavia et al. [AGV09] proved that meaningful and meaningless public keys are computationally indistinguishable even in the presence of secret key leakage based on the LWE assumption, and then utilized this leakage-resilient "assumption" to show that Regev's PKE [Reg05] is actually leakage-resilient. Katz and Vaikuntanathan [KV09] built a leakage-resilient signature from universal one-way hash functions (UOWHFs)[1] together with PKE and simulation-sound non-interactive zero knowledge (NIZK) proof system, where the UOWHFs

[1] This is sometimes called second pre-image resistant functions.

are actually used as leakage-resilient one-way functions. Similar strategy is also adopted for constructing other leakage-resilient signature schemes [DHLW10, BSW11, MTVY11].

Another approach is combining *key detached* strategy and *leakage-resilient* facts/assumptions, which is mainly used in the constructions of leakage-resilient PKE. Informally, the key detached strategy means the underlying intractable problems are not embedded to the secret keys, but to the ciphertexts. Following this approach, the reduction can easily handle key leakage queries by either owning the secret key or relying on leakage-resilient assumptions. Naor and Segev [NS09] utilized hash proof system (HPS) as a powerful tool to construct leakage-resilient PKE. In the security proof, valid ciphertexts are first switched to invalid ones (such switching is computationally indistinguishable even given the whole secret key because the underlying subset membership problem and secret keys are detached) to ensure that the hash proof π has high min-entropy, then the leftover hash lemma is used to prove the session key of the form $\mathsf{ext}(\pi, s)$ is random even in the presence of bounded key leakage.[2] Subsequently, Alwen et al. [ADN+10] and Hazay et al. [HLWW13] extended HPS to the identity-based and symmetric-key setting respectively, and used them to construct leakage-resilient identity-based encryption and secret-key encryption. Dodis et al. [DGK+10] constructed leakage-resilient PKE in the auxiliary input model via a similar method. In the security proof, valid ciphertexts are also first switched to invalid ones, then the generalized Goldreich-Levin theorem is used to argue that the session key of the form $\mathsf{hc}(sk)$ is pseudorandom even given auxiliary-input of the secret key sk.[3]

1.1 Motivation

So far, a broad range of leakage-resilient cryptographic schemes under various leakage models have been proposed in the literature. Nevertheless, several interesting problems are still left open around lower-level, "workhorse" primitives like SKE, PKE, and signature under the basic bounded leakage model.

For leakage-resilient SKE, the task can be reduced to constructing leakage-resilient weak PRFs (wPRFs) in the bounded leakage model. However, the literature on this topic is sparse. [Pie09, DY13] showed that any wPRF is already leakage-resilient for a logarithmic leakage bound $\ell = O(\log \lambda)$. Hazay et al. [HLWW13] built leakage-resilient wPRF from any one-way functions. Their construction only requires minimal assumption, but its leakage rate is $O(\log(\lambda)/|sk|)$, which is rather poor. To date, essentially nothing better was known for generic construction of leakage-resilient SKE with optimal leakage rate, beyond simply using leakage-resilient PKE in the symmetric-key setting.

For leakage-resilient PKE, existing constructions [AGV09, BG10, DGK+10, NS09, ADN+10] are based on either specific assumptions such as LWE, DDH, DCR, QR, or somewhat more generally the hash proof systems[4]. It is intriguing to know if there is a generic construction. In particular, whether the generic constructions of PKE based on trapdoor functions/relations [PW08, RS09, KMO10, Wee10] can be made leakage-resilient is still unclear. On the other hand, semantic security against chosen-ciphertext attacks (CCA) is the strongest notion for PKE in the traditional security model [GM84]. Several previous works [NS09, LWZ13, QL13, QL14, CQX18] studied how to achieve leakage-resilience and CCA security simultaneously via dedicated composition of separate techniques. Nevertheless, no prior work considered the orthogonal problem:

[2] Leftover hash lemma could be interpreted as a leakage-resilient fact, which stipulates $\mathsf{ext}(x, s)$ is close to uniform even given a correlated value z, as long as s is a random seed chosen independently and x still has high min-entropy given leakage z.

[3] Goldreich-Levin theorem can be interpreted as a leakage-resilient assumption, which states that if h is one-way then $\mathsf{hc}(x)$ is pseudorandom even in the presence of $h(x)$. Here hc serves as a computational randomness extractor and $h(x)$ could be viewed as leakage on x.

[4] Following current conventions, we do not regard hash proof systems [CS02] as a general assumption.

whether we can acquire leakage-resilience from CCA security. We observe that in the CCA security experiment, responses to decryption queries can be viewed as a certain form of key leakage (the leakage function f is tied to decryption algorithm but with unbounded output length). It is interesting to know whether there is a general connection between the two important security notions for PKE.

For leakage-resilient signature, achieving *fully leakage-resilience* is of particular interest since it better captures real attacks [KV09]. This notion requires a signature to remain existentially unforgeable under chosen-message attacks even when an adversary obtains bounded leakage information on all intermediate states, including the secret keys and internal random coins. Clearly, if the signing procedure is deterministic or public-coin[5], standard leakage resilience automatically implies fully leakage resilience. To date, all the known fully leakage-resilient signature schemes [BSW11, MTVY11, LLW11, GJS11] in the standard model are randomized and secret-coin. The existence of leakage-resilient deterministic or public-coin signature is unclear and was left as an open problem by Boyle et al. [BSW11]. Earlier, the leakage-resilient signature scheme by Katz and Vaikuntanathan [KV09] is deterministic but only "one-time" secure. Recently, Wang et al. [WMHT16] proposed a leakage-resilient public-coin signature scheme. However, their construction is only secure against selective leakage attacks, i.e., an adversary has to declare the leakage function before seeing the verification key. Besides, their construction requires differing-input obfuscation [BGI+12], whose existence is seriously cast in doubt [GGHW14, BSW16]. From this perspective, the problem posed by Boyle et al. is still largely open.

1.2 Our Contributions

With the preceding discussion in mind, in this work we focus on generic constructions of leakage-resilient encryption and signature in the bounded leakage model. We summarize our main results (depicted in Figure 1) as below.

Leakage-resilient SKE. As shown in [HLWW13], the classic construction of CPA-secure SKE from wPRF is leakage-resilience-preserving. So, we restrict our attention to constructing leakage-resilient wPRFs. Towards this goal, in Section 3.2 we first put forward a new notion called weak puncturable PRFs (wPPRFs), which could be thought of as the puncturable version of wPRFs. We then show wPPRFs and selective puncturable PRFs (sPPRFs) [SW14] imply each other, while the latter is implied by the GGM-tree based PRFs [GGM86]. Finally, in Section 3.3 we build leakage-resilient wPRFs from wPPRFs and $i\mathcal{O}$.

Leakage-resilient KEM. The KEM-DEM paradigm (here KEM stands for key encapsulation mechanism, DEM stands for data encapsulation mechanism) is a modular and efficient approach for building PKE. In the leakage setting, one can build a leakage-resilient PKE by combining a leakage-resilient KEM with a standard DEM. In the rest of this work, we only focus on the construction of leakage-resilient KEM. Chen and Zhang [CZ14] put forward the notion of publicly evaluable PRFs (PEPRFs), which encompasses almost all the known constructions of KEM. We observe that leakage-resilient PEPRFs naturally imply leakage-resilient KEM. So, the task is reduced to acquiring leakage resilience for PEPRFs.

To this end, in Section 4.2 we first put forward the notion of puncturable PEPRFs, then build leakage-resilient PEPRFs from puncturable PEPRFs and $i\mathcal{O}$ in Section 4.3. Moreover, we

[5] A signature is *secret-coin* if its security breaks down when the randomness used in the signing procedure is revealed. On the contrary, a signature is *public-coin* if it stays secure even when the random coins used in the signing procedure are revealed (i.e., provided in-the-clear by the signature). In other words, public-coin signature is secure even when the *entire* random coins used for signing are leaked.

instantiate puncturable PEPRFs from either newly introduced primitives such as puncturable trapdoor functions and puncturable extractable hash proof systems, or existing puncturable PRFs with $i\mathcal{O}$.

This result provides a unified framework for constructing leakage-resilient KEM, which not only clarifies and encompasses the construction by Dachman-Soled et al. [DGL+16, Section 5.1], but also indicates that the PKE constructions based on "puncturable" trapdoor functions/relations (which in turn implied by correlated-product trapdoor functions [RS09] or extractable hash proof systems [Wee10] with puncturable property) can be made leakage resilient! Recently, Matsuda and Hanaoka [MH15] introduced a new primitive called puncturable KEM (PKEM), which captures a common pattern towards CCA security underlying many constructions of CCA-secure PKE. We remark that PPEPRFs imply PKEM with perfect strong punctured decapsulation soundness. This result establishes a somewhat surprising connection between CCA security and leakage resilience, that is, CCA security obtained along the puncturable road can be converted to leakage-resilience in a non-black-box manner via obfuscation.

Leakage-resilient signature. In Section 5, we show how to build leakage-resilient signature from selective puncturable PRFs, $i\mathcal{O}$, and leakage-resilient one-way functions. Our basic scheme is deterministic but only achieves selective security[6]. To attain adaptive security, several bootstrapping techniques can be used without compromising leakage resilience. More precisely, as we elaborate in Section D, one can either use the magic method enabled by extremely lossy function [Zha16], obtaining the first deterministic leakage-resilient signature scheme, or apply the "prefix-guessing technique" [HW09, RW14], yielding the first public-coin leakage-resilient signature scheme.

We highlight that in our construction the signature size is exactly the output size of a puncturable PRF[7], which is very close to the leakage bound. Clearly, signature size cannot be shorter than leakage bound, since otherwise an adversary can directly obtain a forged signature from leakage. In this sense, our constructions also enjoy the almost optimal signature size.

All the basic constructions described above can tolerate L bits of leakage for any polynomial L of security parameter λ. However, the leakage rate is low due to the fact that secret keys are obfuscated programs, which could be very huge. By further assuming the existence of lossy functions [PW08], we can remarkably shrink the size of secret keys and achieve optimal leakage rate $1-o(1)$, as we demonstrate in Section 4.4 and Section 5.2. Such a leakage rate is not known to be achievable for weak PRFs, PEPRFs and deterministic/public-coin signatures before.

1.3 Overview of Our Techniques

As we summarized before, a common theme of the two main approaches towards leakage resilience in the literature is that the reduction always try to simulate leakage oracle *perfectly*, i.e., answering leakage queries with *real* leakage. To do so, we have to either rely on leakage-resilient assumptions or resort to sophisticated design with specific structure. It is interesting to investigate the possibility of simulating leakage oracle *computationally*, namely answering leakage queries with *simulated* leakage, as long as it is computationally indistinguishable from *real* leakage. This would possibly lend new techniques to address the unsolved problems in leakage-resilient cryptography.

[6] In selective security model, the adversary must declare the message m^* on which it will make a forgery before seeing the verification key, but then can adaptively make signing queries on messages distinct from m^*.

[7] In the case of our adaptively secure construction, a signature additionally contains a public coin of size λ^c for any constant $c < 1$.

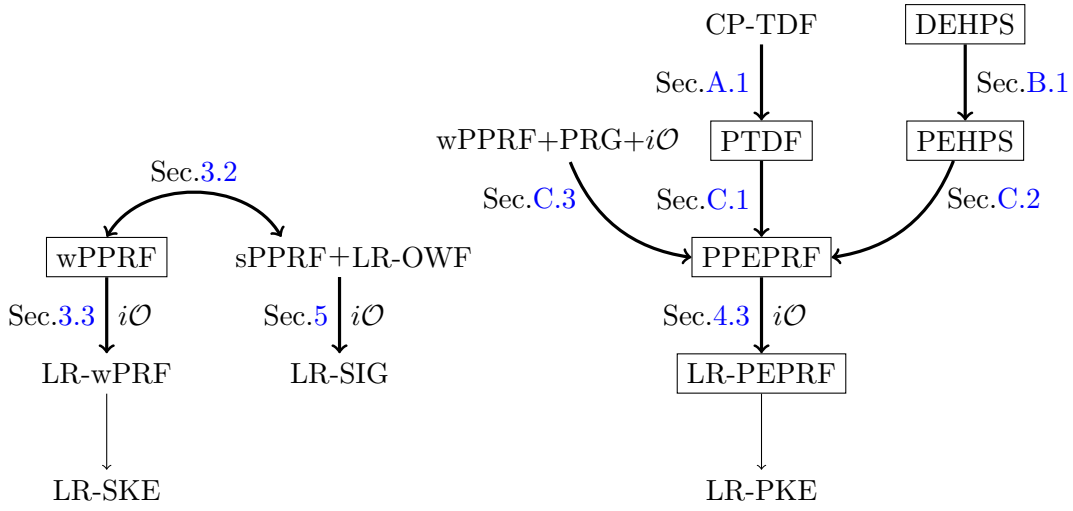

Figure 1: The bold lines and rectangles denote our contributions (the thin lines denote those that are straightforward or follow readily from previous work).

Very recently, Dachman-Soled et al. [DGL+16] discovered powerful applications of $i\mathcal{O}$ to leakage-resilient cryptography. In the continual leakage model, they presented an $i\mathcal{O}$-based compiler that transforms any public-key encryption or signature scheme with *consecutive* continual leakage-resilience to continual leakage resilience allowing leakage on key updates. In the bounded leakage model, they showed how to modify the Sahai-Waters PKE to be leakage-resilient. We observe that their work essentially embodies the idea of simulating leakage oracle computationally.

Simulate leakage via obfuscation. At the heart of our leakage-resilient encryptions and signatures is a general approach of simulating leakages enabled by puncturable primitives and obfuscation, which is largely inspired by the leakage-resilient variant of Sahai-Waters PKE due to Dachman-Soled et al. [DGL+16]. Next, we first distill and extend the idea underlying the work of [DGL+16], then carry out a systematic study of its applicability to leakage-resilient cryptography.

Recall that the common technical hurdle towards leakage resilience is to handle leakage queries. As opposed to the naive strategy of answering leakage queries with real secret keys, another promising strategy is simulating leakage with "faked" secret keys. By the composition lemma, as long as the faked secret keys are indistinguishable from the real ones, the simulated leakages are also indistinguishable from the real leakages because all leakage functions are efficiently computable.

Our approach adopts the second strategy. First, a secret key sk of any cryptographic scheme can always be expressed as a program Eval with sk hardwired. If a cryptographic scheme is puncturable (e.g., puncturable PRFs), then the reduction may build a functional-equivalent program Eval$'$ with sk_{x^*} and y^* hardwired, where sk_{x^*} is the punctured secret key at input x^* and $y^* = \text{Eval}(x^*)$. Secondly, note that indistinguishability obfuscation preserves functionality and guarantees that the obfuscations of any two functional-equivalent programs are computationally indistinguishable. Therefore, by setting the new secret key as $i\mathcal{O}(\text{Eval})$, the reduction is able to simulate leakage queries with $i\mathcal{O}(\text{Eval}')$. This approach abstracts the high-level idea of how to acquire leakage-resilience when puncturable primitives meet $i\mathcal{O}$.

Obfuscate key and extract randomness. Our leakage-resilient encryptions exactly follow this approach. In a nutshell, we use $i\mathcal{O}$ to compile weak (resp. publicly evaluable) puncturable

PRFs into leakage-resilient weak (resp. publicly evaluable) PRFs, which immediately yield leakage-resilient secret-key (resp. public-key) encryption.

To best illustrate our approach, we focus here on the secret-key setting, as it already emphasizes the main ideas underlying our approach. Starting from a weak puncturable PRF $F : K \times X \to \{0,1\}^n$, we use $i\mathcal{O}$ to compile it into a leakage-resilient weak PRF with a randomness extractor $\mathsf{ext} : \{0,1\}^n \times S \to Z$. The construction is instructive: (1) generate a secret key k for F, then create a program Eval with k hardwired, which on input x and s outputs $\mathsf{ext}(F_k(x), s)$; (2) set the secret key as $i\mathcal{O}(\text{Eval})$. This defines a weak PRF $\hat{F} : \{0,1\}^n \times S \to Z$. To establish security, the hybrid argument starts with the real game for leakage-resilient wPRF, where leakage and evaluation queries are handled with real secret key $i\mathcal{O}(\text{Eval})$. In the next game, the challenger picks the challenge input x^* and s^* at the very beginning, create a program Eval$'$ with the same input-output behavior as Eval, where k_{x^*} is the punctured key for k w.r.t. x^* and $y^* = F_k(x^*)$. The leakage and evaluation queries are thus handled with $i\mathcal{O}(\text{Eval}')$. Such modifications are undetectable by the security of $i\mathcal{O}$. In the final game, the challenger switches y^* from $F_k(x^*)$ to a random value. This transition is undetectable by the weak pseudorandomness of the starting puncturable PRF. An important fact is that the responses to evaluation queries are determined by k_{x^*}, and thus do not leak any information about y^*. Now, we can argue the desired security in purely information-theoretic way. By appropriate parameter choice, y^* still retains high min-entropy in the presence of leakage, and thus the value $\mathsf{ext}(y^*, s^*)$ is statistically close to uniform distribution.

In the public-key setting, our construction essentially follows the same approach. We use $i\mathcal{O}$ to compile puncturable PEPRF into leakage-resilient PEPRF, which readily yield leakage-resilient CPA-secure KEM. The main technical novelty lies in realizing puncturable PEPRFs from a variety of puncturable primitives. More precisely, we build puncturable PEPRFs from: (1) newly introduced notion of puncturable TDFs, which is in turn implied by correlated-product TDFs [RS09]; (2) newly introduced notion of puncturable EHPS, which is implied by EHPS [Wee10] satisfying derivable property; (3) selective puncturable PRFs, pseudorandom generator, and $i\mathcal{O}$ (adapted from the Sahai-Waters PKE [SW14]). This provides us a unified method to build leakage-resilient KEM from various puncturable primitives and $i\mathcal{O}$.

Obfuscate key and translate leakage. Along our approach towards leakage resilience, we investigate the possibility of building leakage-resilient signature from puncturable primitives and $i\mathcal{O}$. We choose the short "hash-and-sign" selectively secure signature by Sahai and Waters [SW14] as our starting point, since it inherits the puncturable property from its underlying selective puncturable PRFs. To best illustrate the idea of our adaption, we first briefly review the Sahai-Waters signature scheme.

The Sahai-Waters signature is essentially a PRF-based MAC with public verifiability. The signing key sk is simply a secret key of selective puncturable PRF (sPPRF), and the signature on m is $\sigma \leftarrow F_k(m)$. The verification key vk is set as $i\mathcal{O}(\text{Vefy})$ where Vefy is a program that can check the MAC publicly. To excise out the information about $F_k(m^*)$ (here m^* denotes the target message), Vefy computes $g(F_k(m))$ and compares the result for equality to $g(\sigma)$, where g is a one-way function and σ is the claimed signature on m. To establish security, the hybrid argument starts with the real game for selective signature. The intermediate hybrid game builds an equivalent verification program using a punctured key k_{m^*} and $y^* \leftarrow g(\sigma^*)$ where $\sigma^* = F_k(m^*)$. The final hybrid game replaces σ^* with a random value. The first transition is undetectable by the security of $i\mathcal{O}$, while the second transition is undetectable by the pseudorandomness of sPPRF. In the final game, no PPT adversary is able to output a valid forgery (find the preimage σ^*) with non-negligible advantage by the one-wayness of g.

Following the new approach of simulating leakage, a tempting idea to make the Sahai-Waters signature leakage-resilient is setting the signing key as $i\mathcal{O}(\text{Sign})$, where Sign is a program that on

input m outputs $F_k(m)$. Among the transitions of hybrid games, Sign is replaced by Sign$'$ (with k_{m^*} and σ^* hardwired). In this way, leakage and signing queries can be handled with "faked" signing key. However, we are unable to reduce the leakage-resilient unforgeability to the one-wayness of g. This is because in addition to $y^* = g(\sigma^*)$ revealed in vk, the information of σ^* may also be leaked via leakage queries on signing key $i\mathcal{O}(\text{Sign}')$. Therefore, the security proof breaks down in the final game, i.e., the reduction has to build Sign$'$ while σ^* is unknown.[8] We overcome this obstacle by using *leakage-resilient* OWF to replace standard OWF. Briefly, OWF is *leakage-resilient* if one-wayness remains in the presence of certain leakage on the preimage. Also observe that a leakage function f about the signing key $i\mathcal{O}(\text{Sign}')$ can be efficiently translated to leakage about σ^*, since both f and $i\mathcal{O}$ are efficiently computable. With such enhancement, in the final game the reduction can handle signing queries using k_{m^*} and handle leakage queries on signing key $i\mathcal{O}(\text{Sign}')$ by translating them to leakage queries on preimage σ^* to the underlying leakage-resilient OWF. See Section 5 for technical details.

Improving leakage rate via lossy functions. Applying the above approach in a straightforward manner will incur poor leakage rate, because the secret keys are obfuscated programs, which could be very large.

In [DGL+16], the authors showed how to modify their basic leakage-resilient PKE construction to achieve optimal leakage rate. Next, we briefly revisit their technique in the context of our construction of leakage-resilient wPRF. Now, the key generation algorithm works as follows: (1) pick a random key k_e for a SKE scheme and generate a dummy ciphertext $ct \leftarrow \mathsf{Enc}(k_e, 0^n)$ as the secret key sk; (2) pick a collision-resistant hash h and compute $\eta^* \leftarrow h(ct)$; (3) pick a random key k for the underlying weak PRF, obfuscate a program Eval and store the obfuscated result C_{eval} into public parameters. Here, the program Eval is hardwired with k and t^*, which on input sk and (x, s) outputs $\mathsf{ext}(F_k(x), s)$ if and only if $h(sk) = \eta^*$. Intuitively, ct acts as a trigger of C_{eval}, which only works when $h(ct)$ matches η^*. In this way, the size of secret key is greatly reduced.

In the security proof, the first game is the real game. In the next game, ct is switched to an encryption of the PRF value $y^* \leftarrow F_k(x^*)$. This modification is undetectable by the semantic security of SKE. Then, C_{eval} is switched to C'_{eval}, which is an obfuscation of program Eval$'$. With k_e and a punctured PRF key k_{x^*} hardwired, Eval' works if and only if the hash value of its input ct matches η^*. When $h(ct) = t^*$, it evaluates with k_{x^*} if $x \neq x^*$, otherwise it evaluates after decrypting ct to y^*. In the final game, y^* is switched to a uniformly random value. The rest security analysis is routine. A subtle problem arised is that now Eval and Eval$'$ have differing inputs, because h is compressing and thus a collision ct' (i.e., $h(ct') = \eta^* = h(ct)$) that encrypts a value $y' \neq y^*$ is likely to exist. Therefore, they have to rely on public-coin differing-input obfuscation [IPS15], which is stronger than indistinguishability obfuscation.

As analyzed above, the usage of CRHF leads to the reliance on differing-input obfuscation, while the choice of CRHF seems necessary to ensure that η^* only leaks partial information about y^* (encrypted in ct), which is crucial to achieve high leakage rate. Can we achieve higher leakage rate without resorting to differing-input obfuscation? The answer is affirmative. Our idea is to replace CRHFs with lossy functions [PW08]. In the real construction, h is generated as an injective function. By this choice, η^* uniquely fixes its preimage ct and thus the value y^*, With this setting, Eval and Eval$'$ agree on all inputs, and $i\mathcal{O}$ suffices to guarantee the switching from Eval to Eval$'$ is undetectable. To argue the high leakage rate we can attain, in the last game we switches h to a lossy function that significantly lose the information about y^*. By the security of lossy functions, this change is undetectable. Clearly, in the last game y^* still

[8] Note that this dilemma does not occur in the case of encryption, since the argument in the final game is information-theoretic.

maintains sufficiently large min-entropy even in the presence of η^* and leakage. By appropriate choice of parameter, optimal leakage rate is achievable. The above technique carries over to the constructions of leakage-resilient PEPRF and signature as well, except some subtle issues need to be carefully dealt with for the case of signature. See the discussion at the end of Section 5.2 for details.

We believe that the our technique of improving leakage rate by interplaying $i\mathcal{O}$ with lossy functions will also be instructive for avoiding using differing-input obfuscation in other places.

1.4 Related Work

Leakage models. Several leakage models have been proposed in the literature. In the seminal work, Micali and Reyzin [MR04] initiated the formal study of side-channel attacks by introducing the "only computation leaks information" model. Unfortunately, it fails to capture many practical leakage attacks, such as the cold-boot attack of [HSH+08].

To capture more general side-channel attacks known as memory attacks, Akavia et al. [AGV09] introduced the bounded leakage model, in which the adversary can obtain arbitrary length-bounded leakage. The follow-up works considered various strengthens to accommodate more complex and general leakage scenarios. Naor and Segev [NS09] generalized the bounded leakage model to noisy leakage model (also known as entropy leakage model), where length-bounded leakage is relaxed to entropy-bounded leakage. Alwen et al. [ADW09a, ADN+10] suggested the bounded-retrieval model, which imposes an additional requirement that the tolerated leakage amount can grow by proportionally expanding the secret key without increasing the size of public key, or computation/bandwidth efficiency. Dodis et al. [DHLAW10] and Brakerski et al. [BKKV10] introduced the continual leakage model for public-key schemes, where the secret key can be periodically self-refreshed while the public key remains the same. This model allows bounded leakage between any two successive refreshes without a-priori bound on the overall amount of leakage throughout the lifetime of the system.

The bottomline of the bounded leakage model and its variants interpret the following restriction on the leakage: it is information-theoretically impossible to recover the secret key from the leakage. Dodis et al. [DKL09, DGK+10] introduced the auxiliary input model (AIM), in which the total amount of leakage could be unbounded, as long as the secret key remains hard-to-invert given the leakage (but even if the secret key is fully determined in an information-theoretic sense). As noted in [KV09], a drawback of this model is that given some collection of leakage functions $\{f_i\}$ there is no way to tell, in general, whether they satisfy the stated requirement or not. Furthermore, existing constructions in this model require super-polynomial hardness assumptions.

Leakage-resilient cryptosystems. There is a large body of constructions of leakage-resilient cryptosystems in various models. In the bounded leakage model, there are OWF [KV09, Kom16], MAC and SKE [HLWW13], PKE [AGV09, NS09, LWZ13, QL13, QL14, CQX18], IBE [AGV09, ADN+10, CDRW10], signature [KV09, ADW09a], AKE [ADW09a], and zero-knowledge proofs [GJS11]. In the continual leakage model, there are PKE [DHLAW10, BKKV10], IBE [LRW11, YCZY12, YXZ+15], and signature [BSW11, MTVY11, LLW11]. In the auxiliary input model, there are SKE [DKL09], PKE [DGK+10], and signature [WMHT16].

2 Preliminaries

Notation. For a distribution or random variable X, we write $x \xleftarrow{R} X$ to denote the operation of sampling a random x according to X. For a set X, we use $x \xleftarrow{R} X$ to denote the operation

of sampling x uniformly at random from X, and use $|X|$ to denote its size. We use U_X to denote the uniform distribution over X. For a positive integer n, we use $[n]$ to denote the set $\{1, \ldots, n\}$. Unless described otherwise, all quantities are implicitly functions of a security parameter denoted λ. We say that a quantity is negligible, written $\mathsf{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ. A probabilistic polynomial time (PPT) algorithm is a randomized algorithm that runs in time $\mathsf{poly}(\lambda)$. If $\mathcal{A}$ is a randomized algorithm, we write $z \leftarrow \mathcal{A}(x_1, \ldots, x_n; r)$ to indicate that $\mathcal{A}$ outputs z on inputs $(x_1, \ldots, x_n)$ and random coins r. For notational clarity we usually omit r and write $z \leftarrow \mathcal{A}(x_1, \ldots, x_n)$.

2.1 Entropy and Randomness Extraction

We use capital letters (e.g. X) for random variables, standard letters (e.g. x) for values, and calligraphic letters (e.g. $\mathcal{X}$) for sets. The *min-entropy* of a random variable X over $\mathcal{X}$ is the negative (base-2) logarithm of the *unpredictability* of X: $\mathsf{H}_\infty(X) = -\log\left(\max_{x \in \mathcal{X}} \Pr[X = x]\right)$.

In many natural settings, the variable X is correlated with another variable Y whose value is known to an adversary. In such scenarios, it is most convenient to use the notion of *average min-entropy* as defined by Dodis et al. [DORS08], which captures the *average unpredictability* of X conditioned on Y:

$$\tilde{\mathsf{H}}_\infty(X|Y) = -\log\left(\mathrm{E}_{y \leftarrow Y}\left[\max_{x \in \mathcal{X}} \Pr[X = x|Y = y]\right]\right)$$

The following bound of average min-entropy was proved in [DORS08].

Lemma 2.1 ([DORS08])**.** *Let X, Y, Z be arbitrarily correlated random variables where the support of Y has at most 2^r elements. Then $\tilde{\mathsf{H}}_\infty(X|(Y,Z)) \geq \mathsf{H}_\infty(X|Z) - r$. In particular, $\tilde{\mathsf{H}}_\infty(X|Y) \geq \mathsf{H}_\infty(X) - r$.*

In cryptographic applications, we usually need to derive nearly uniform bits from a weakly random source X that has some average min-entropy. This is accomplished via an appropriate type of *randomness extractor*. We recall the definition of average-case strong extractor from [DORS08].

Definition 2.1 (Randomness Extractor)**.** An efficient function $\mathsf{ext} : \mathcal{X} \times \mathcal{S} \to \mathcal{Y}$ is an average-case (n, ϵ)-strong extractor if for all (correlated) random variables (X, Z) s.t. $\tilde{\mathsf{H}}_\infty(X|Z) \geq n$, it holds that:

$$\Delta((\mathsf{ext}(X, S), S, Z), (Y, S, Z)) \leq \epsilon,$$

where S is uniform over $\mathcal{S}$ and Y is uniform over $\mathcal{Y}$, respectively.

Dodis et al. [DORS08] proved that any strong extractor is in fact an average-case strong extractor for appropriate setting of the parameters. As a specific example, they proved that any family of universal hash functions is an average-case strong extractor.

Lemma 2.2 ([DORS08])**.** *Let X and Z be random variables such that $\tilde{\mathsf{H}}_\infty(X|Z) \geq n$, and $\mathcal{H} = \{h_S : \mathcal{X} \to \mathcal{Y}\}_{S \in \mathcal{S}}$ be a family of universal hash functions. Then $\mathsf{ext}(X, S) := h_S(X)$ is a (n, ϵ)-extractor as long as $n \geq \log|\mathcal{Y}| + 2\log(1/\epsilon)$.*

2.2 Leakage Types

In the standard bounded leakage model [AGV09], the amount of leakage that the adversary learns is measured by the output length of leakage function f, which is refereed to as *length-bounded leakage* [BSW11]. However, this is not the most general way of measuring leakage.

As an alternative suggested by Naor and Segev [NS09], Dodis et al. [DHLAW10], and Boyle et al. [BSW11], we could measure the amount of leakage via the *entropy loss* to the input of f, given the output of f. By relaxing the requirement on leakage functions from length-bounded to entropy-bounded, the standard *bounded leakage model* strengthens to *entropy leakage model*. Next, we formally recall the notion of entropy-bounded function from [BSW11] as below.

Definition 2.2 (ℓ-entropy-bounded functions)**.** A (possibly randomized) efficiently computable function $f : \{0,1\}^* \to \{0,1\}^*$ is ℓ-entropy-bounded if there exists some (possibly inefficiently computable) function f' such that:

- For all $x \in \{0,1\}^*$, $f(x) \approx_s f'(x)$ (over the randomness of f and f').
- For all integers $n \geq 1$, $\tilde{\mathsf{H}}_\infty(U_n | f'(U_n)) \geq n - \ell$, where U_n is the uniform distribution over $\{0,1\}^n$.

Notice that any function $f : \{0,1\}^* \to \{0,1\}^\ell$ is ℓ-entropy-bounded. Clearly, there are functions which are ℓ-entropy-bounded but whose output lengths can be arbitrarily long. Therefore, resilience to ℓ-bit of entropy leakage is a seemingly stronger notion of security than resilience to ℓ-bit of length-bounded leakage.

2.3 Cryptographic Primitives

2.3.1 Lossy Functions

Lossy functions are the trapdoor-free version of lossy trapdoor functions introduced by Peikert and Waters [PW08].

Definition 2.3 (Lossy Functions)**.** A family of (n, τ)-lossy functions G from $X = \{0,1\}^n$ to Y is given by three polynomial time algorithms satisfying the following properties:

- $\mathsf{GenInj}(\lambda)$: on input a security parameter λ, output a function index i such that $G(i, \cdot)$ is an injective function from X to Y.
- $\mathsf{GenLossy}(\lambda)$: on input a security parameter λ, output a function index i such that $G(i, \cdot)$ is a lossy function from X to Y whose image has size at most 2^τ. The *lossiness* is defined as $n - \tau$.
- $\mathsf{Eval}(i, x)$: on input a function index i and an element x, outputs $G(i, x)$.[9]

Hard to distinguish injective from lossy. The outputs of $\mathsf{GenInj}(\lambda)$ and $\mathsf{GenLossy}(\lambda)$ are computationally indistinguishable.

2.3.2 One-way Functions

Roughly, a one-way function $g : \{0,1\}^n \to \{0,1\}^m$ is leakage-resilient if it remains one-way even in the presence of some leakage about preimage. Boyle et al. [BSW11] formalized the notion of leakage-resilient one-wayness w.r.t. either length-bounded leakage or entropy-bounded leakage (which is more general), which we recall as below.

Leakage-resilient one-wayness. Let $\mathcal{A}$ be an adversary against g and define its advantage in the following experiment:

$$\mathrm{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[g(x) = y^* : \begin{array}{l} x^* \xleftarrow{R} \{0,1\}^n, y^* \leftarrow g(x^*); \\ x \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{leak}}(\cdot)}(y^*); \end{array} \right].$$

[9] For simplicity, we write $g_i(x)$ to represent $G(i, x)$ and drop the subscript index when the context is clear.

Here $\mathcal{O}_{\mathsf{leak}}(\cdot)$ is a leakage oracle that on input $f : \{0,1\}^n \to \{0,1\}^*$ returns $f(x^*)$, subjected to the restriction that its total output lengths is at most ℓ. We say g is ℓ-leakage-resilient one-way if for any PPT adversary $\mathcal{A}$ its advantage defined as above is negligible in λ.

Constructions of LR-OWFs. As implicitly or explicitly shown in [KV09, ADW09b, DHLW10, BSW11, Kom16], a universal one-way hash function with n-bit inputs and m-bit outputs automatically constitutes an ℓ-leakage-resilient OWF as long as $n - m - \ell \geq \omega(\log \lambda)$. The leakage amount ℓ is roughly the number of bits by which the UOWHF shrinks its input.

Chen et al. [CQX18] realized that any lossy (non-trapdoor) functions [PW08] yield a family of leakage-resilient injective OWFs. Let LF be a collection of (n, τ)-lossy functions and ℓ be the amount of leakage. They proved that for $n - \tau - \ell \geq \omega(\log \lambda)$, the functions in the injective mode readily constitute a family of ℓ-leakage-resilient injective OWFs. The leakage amount ℓ is roughly the lossiness $(n - \tau)$ and the leakage rate could be $1 - o(1)$ by setting $\tau = o(n)$.

The above leakage-resilient results also hold in the entropy leakage model.

2.3.3 Signatures

Definition 2.4 (Signatures)**.** A signature scheme with message space M and signature space Σ consists of three polynomial time algorithms as follows.

- $\mathsf{Gen}(\lambda)$: on input a security parameter λ, output a verification key vk and a signing key sk.
- $\mathsf{Sign}(sk, m)$: on input sk and a message $m \in M$, output a signature $\sigma \in \Sigma$.
- $\mathsf{Verify}(vk, m, \sigma)$: on input vk, a message m, and a purported signature σ, output 1 indicating acceptance or 0 indicating rejection.

Correctness. For all $(vk, sk) \leftarrow \mathsf{Gen}(\lambda)$ and all $m \in M$, we have $\mathsf{Verify}(vk, m, \mathsf{Sign}(sk, m)) = 1$.

The definition of leakage resilience for signature is the standard notion of existential unforgeability under adaptive chosen-message attacks (EUF-CMA), except that the adversary is additionally given access to a leakage oracle.

Leakage-resilient EUF-CMA. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against signature and define its advantage in the following experiment:

$$\mathrm{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c|l} \mathsf{Verify}(vk, m^*, \sigma^*) = 1 & (vk, sk) \leftarrow \mathsf{Gen}(\lambda); \\ \wedge\ m^* \notin \mathcal{Q} \quad : & (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sign}}(\cdot), \mathcal{O}_{\mathsf{leak}}(\cdot)}(vk); \end{array}\right].$$

Here $\mathcal{O}_{\mathsf{leak}}(\cdot)$ is a leakage oracle that on input $f : SK \to \{0,1\}^*$ returns $f(sk)$, subjected to the restriction that the total output length of all f is at most ℓ. $\mathcal{O}_{\mathsf{sign}}(\cdot)$ is a signing oracle that on input m outputs $\sigma \leftarrow \mathsf{Sign}(sk, m)$. The set $\mathcal{Q}$ records queries to $\mathcal{O}_{\mathsf{sign}}(\cdot)$. A signature is ℓ-leakage-resilient EUF-CMA if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the above security experiment. A weaker notion called leakage-resilient selectively EUF-CMA can be defined in exactly the same way except that the adversary is asked to declare m^* even before seeing vk.

2.3.4 Secret-Key Encryption

Definition 2.5 (Secret-Key Encryption)**.** An SKE scheme with message space M, ciphertext space C and secret key space K consists of three polynomial time algorithms as follows.

- $\mathsf{Gen}(\lambda)$: on input a security parameter λ, output a random secret key $k \xleftarrow{\mathrm{R}} K$.

- $\mathsf{Enc}(k, m)$: on input a secret key k and a message $m \in M$, output a ciphertext $c \in C$.
- $\mathsf{Dec}(k, c)$: on input a secret key k and a ciphertext $c \in C$, output a message $m \in M$.

Correctness. For all $k \leftarrow \mathsf{Gen}(\lambda)$ and all $m \in M$, we have $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m$.

We recall leakage-resilient chosen-plaintext security (LR-CPA) for SKE as follows.

Leakage-resilient CPA. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against SKE and define its advantage in the following experiment:

$$\mathrm{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} k \leftarrow \mathsf{Gen}(\lambda); \\ (state, m_0, m_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{leak}}(\cdot), \mathcal{O}_{\mathsf{enc}}}(\lambda); \\ \beta \xleftarrow{\mathrm{R}} \{0,1\}; \\ c^* \leftarrow \mathsf{Enc}(k, m_\beta); \\ \beta' \leftarrow \mathcal{A}_2(state, c^*); \end{array}\right] - \frac{1}{2}.$$

Here $\mathcal{O}_{\mathsf{leak}}(\cdot)$ is a leakage oracle that on input $f : SK \to \{0,1\}^*$ returns $f(sk)$, subjected to the restriction that the sum of its output lengths is at most ℓ. $\mathcal{O}_{\mathsf{enc}}$ is an encryption oracle that on input $m \in M$ outputs $\mathsf{Enc}(k, m)$. (When Enc is randomized, $\mathcal{O}_{\mathsf{enc}}$ uses fresh randomness each time it answers a query.) A SKE is ℓ-leakage-resilient CPA-secure if no PPT adversary has non-negligible advantage in the above security experiment.

2.3.5 Key Encapsulation Mechanism

Definition 2.6 (Key Encapsulation Mechanism)**.** A KEM with message space M, ciphertext space C and encapsulated key space K consists of three polynomial time algorithms as below.

- $\mathsf{Gen}(\lambda)$: on input a security parameter λ, output a public key pk and a secret key sk.
- $\mathsf{Encaps}(pk)$: on input a public key pk, output a ciphertext $c \in C$ and an encapsulated key $k \in K$.
- $\mathsf{Decaps}(sk, c)$: on input a secret key sk and a ciphertext $c \in C$, output an encapsulated key $k \in K$ or a distinguished symbol $\perp$ indicating that c is invalid.

Correctness. For all $(pk, sk) \leftarrow \mathsf{Gen}(\lambda)$ and all $(c, k) \leftarrow \mathsf{Encaps}(pk)$, we have $\mathsf{Decaps}(sk, c) = k$.

We recall leakage-resilient chosen-plaintext security (LR-CPA) for KEM as follows.

Leakage-resilient CPA. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against KEM and define its advantage in the following experiment:

$$\mathrm{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} (pk, sk) \leftarrow \mathsf{Gen}(\lambda); \\ state \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{leak}}(\cdot)}(pk); \\ (c^*, k_0^*) \leftarrow \mathsf{Encaps}(pk), k_1^* \xleftarrow{\mathrm{R}} K; \\ \beta \xleftarrow{\mathrm{R}} \{0,1\}; \\ \beta' \leftarrow \mathcal{A}_2(state, c^*, k_\beta^*); \end{array}\right] - \frac{1}{2}.$$

Here $\mathcal{O}_{\mathsf{leak}}(\cdot)$ is a leakage oracle that on input $f : SK \to \{0,1\}^*$ returns $f(sk)$, subjected to the restriction that the sum of its output lengths is at most ℓ. A KEM is ℓ-leakage-resilient CPA-secure if no PPT adversary has non-negligible advantage in the above security experiment.

Remark 2.1. The KEM-DEM approach also works in the leakage setting, i.e., one can build leakage-resilient PKE by combining a leakage-resilient KEM and a standard DEM. The resulting PKE inherits the same leakage-resilience from the underlying KEM.

2.3.6 Puncturable Pseudorandom Functions

Puncturable PRFs (PPRFs) [SW14] is the simplest type of constrained PRFs [KPTZ13, BW13, BGI14]. In a PPRF, the constrained key is associated with an element $x^* \in X$, which allows evaluation on all elements $x \neq x^*$. Next, we recall the definition and security notion of PPRFs as below.

Definition 2.7 (PPRFs)**.** A PPRF $F : K \times X \to Y$ consists of four polynomial time algorithms:

- $\mathsf{Gen}(\lambda)$: on input λ, output public parameter pp and a secret key $k \xleftarrow{R} K$. pp will be used as an implicit input of $\mathsf{PrivEval}$, $\mathsf{Puncture}$ and $\mathsf{PuncEval}$.
- $\mathsf{PrivEval}(k, x)$: on input a secret key k and $x \in X$, output $F(k, x)$.
- $\mathsf{Puncture}(k, x^*)$: on input a secret key k and $x^* \in X$, output a punctured key $k(\{x^*\})$.[10]
- $\mathsf{PuncEval}(k_{x^*}, x)$: on input a punctured key k_{x^*} and an element $x \in X$, output $F(k, x)$ if $x \neq x^*$ and a special reject symbol $\perp$ otherwise.

For ease of notation, we write k_{x^*} to represent $k(\{x^*\})$, write $F_k(x)$ and $F(k, x)$ interchangeably and write $F_{k_{x^*}}(x)$ or $F(k_{x^*}, x)$ to represent $\mathsf{PuncEval}(k_{x^*}, x)$.

Sahai and Waters [SW14] defined selective pseudorandomness for PPRFs, which is weaker than full pseudorandomness in that the adversary must commit to the target input x^* even before seeing the public parameter.

Selective pseudorandomness. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against PPRFs and define its advantage in the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} (state, x^*) \leftarrow \mathcal{A}_1(\lambda); \\ (pp, k) \leftarrow \mathsf{Gen}(\lambda); \\ k_{x^*} \leftarrow \mathsf{Puncture}(k, x^*); \\ y_0^* \leftarrow F_k(x^*), y_1^* \xleftarrow{R} Y; \\ \beta \xleftarrow{R} \{0,1\}; \\ \beta' \leftarrow \mathcal{A}_2(state, pp, k_{x^*}, y_\beta^*); \end{array}\right] - \frac{1}{2}.$$

A PPRF is said to be selectively pseudorandom if for any PPT adversary $\mathcal{A}$ its advantage defined as above is negligible in λ. For simplicity, we refer to selectively pseudorandom PPRFs as sPPRFs. sPPRFs with fixed-length domain are easily obtained from the GGM tree-based PRFs [GGM86], as observed in [BW13, BGI14, KPTZ13]. Ramchen and Waters [RW14] also showed the existence of sPPRFs with variable-length domain.

2.3.7 Indistinguishability Obfuscation for Circuits

We recall the definition and security notion of indistinguishability obfuscator from [GGH+13] as below.

Definition 2.8 (Indistinguishability Obfuscator ($i\mathcal{O}$))**.** A uniform PPT machine $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- (Preserving Functionality) For all security parameter $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, and for all inputs $x \in \{0,1\}^*$, we have:

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

[10] Without loss of generality, we assume that $k(\{x^*\})$ includes the information of x^* in plain.

- (Indistinguishability of Obfuscation) For any PPT adversaries $(\mathcal{S}, \mathcal{D})$, there exists a negligible function α such that the following holds: if $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, aux) \leftarrow \mathcal{S}(\lambda)] \geq 1 - \alpha(\lambda)$, then we have:

$$|\Pr[\mathcal{D}(aux, i\mathcal{O}(\lambda, C_0)) = 1] - \Pr[\mathcal{D}(aux, i\mathcal{O}(\lambda, C_1)) = 1]| \leq \alpha(\lambda)$$

3 Leakage-Resilient SKE

We begin this section by recalling the notion of leakage-resilient wPRFs and their application in building leakage-resilient CPA-secure SKE from [HLWW13]. We then introduce a new notion called weak puncturable PRFs (weak PPRFs), and show how to compile weak PPRFs to leakage-resilient wPRFs via $i\mathcal{O}$.

3.1 Leakage-Resilient Weak PRFs

Standard PRFs require full pseudorandomness: given polynomially many *arbitrarily* inputs $x_1, \ldots, x_q$, the outputs $F_k(x_1), \ldots, F_k(x_q)$ look pseudorandom. Sometimes, the full power of PRFs is not needed and it is sufficient to have weak PRFs which only claim weak pseudorandomness, where pseudorandomness holds for *uniformly random* choice of inputs $\{x_i\}$. The corresponding leakage-resilient notion requires that weak pseudorandomness holds even if the adversary can learn some leakage about the secret key k. Now, we recall the formal definition of leakage-resilient weak pseudorandomness from [HLWW13].

Leakage-resilient weak pseudorandomness. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT adversary against PRFs and define its advantage in the following experiment.

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} (pp, k) \leftarrow \mathsf{Gen}(\lambda); \\ state \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{leak}}(\cdot), \mathcal{O}_{\mathsf{eval}}(\$)}(pp); \\ x^* \xleftarrow{R} X; \\ y_0^* \leftarrow F_k(x^*), y_1^* \xleftarrow{R} Y; \\ \beta \xleftarrow{R} \{0,1\}; \\ \beta' \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{eval}}(\$)}(state, x^*, y_\beta^*); \end{array}\right] - \frac{1}{2}.$$

Here $\mathcal{O}_{\mathsf{leak}}(\cdot)$ is a leakage oracle that on input leakage function $f : K \to \{0,1\}^*$ returns $f(k)$, subjected to the restriction that the sum of its output lengths is at most ℓ. $\mathcal{O}_{\mathsf{eval}}(\$)$ is an evaluation oracle that does not take any input and on each invocation, chooses a freshly random $x \in X$ and outputs $(x, F_k(x))$. A PRF is ℓ-leakage-resilient weakly pseudorandom if no PPT adversary has non-negligible advantage in the above experiment.

Remark 3.1. As pointed out in [HLWW13], since the adversary can always learn a few bits of $F_k(x)$ for some x of its choice (via leakage query), we cannot hope to achieve full pseudorandomness in the presence of leakage, and hence setting for weak pseudorandomness is a natural choice.

Leakage-resilient SKE. The construction of LR CPA-secure SKE from LR wPRF is obvious. We sketch the construction from [HLWW13] for completeness. Assume $F : K \times X \to Y$ is a leakage-resilient wPRF, whose range Y is an additive group (e.g., bit-strings under XOR). The secret key is exactly the key of the underlying wPRF. To encrypt a message $m \in Y$, one samples $x \xleftarrow{R} X$ and outputs the ciphertext $(x, F_k(x) + m)$. The decryption process is obvious. The desired LR CPA security of SKE follows readily from the LR weak pseudorandomness of the wPRF.

3.2 Weak Puncturable PRFs

Towards the construction of leakage-resilient wPRFs, we put forward a new notion called weak PPRFs by introducing weak pseudorandomness for PPRFs. We show that weak PPRFs and selective PPRFs imply each other, while the latter is directly implied by the GGM-tree based PRFs [GGM86].

Next, we formally introduce weak pseudorandomness for PPRFs, which differs from selective pseudorandomness (cf. definition in Section 2.3.6) in that the target input x^* is uniformly chosen by the challenger, rather than being arbitrarily chosen by the adversary before seeing the public parameter.

Weak pseudorandomness. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against PPRFs and define its advantage in the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} (pp, k) \leftarrow \mathsf{Gen}(\lambda); \\ x^* \xleftarrow{\mathrm{R}} X; \\ k_{x^*} \leftarrow \mathsf{Puncture}(k, x^*); \\ y_0^* \leftarrow F_k(x^*), y_1^* \xleftarrow{\mathrm{R}} Y; \\ \beta \xleftarrow{\mathrm{R}} \{0,1\}; \\ \beta' \leftarrow \mathcal{A}(pp, x^*, k_{x^*}, y_\beta^*); \end{array}\right] - \frac{1}{2}.$$

A PPRF is weakly pseudorandom if no PPT adversary has non-negligible advantage in the above experiment. For simplicity, we refer to weakly pseudorandom PPRFs as wPPRFs.

Interestingly, we show that wPPRFs and sPPRFs imply each other.

Theorem 3.1. *wPPRFs and sPPRFs imply each other.*

Proof. We first show that "wPPRFs imply sPPRFs" by building sPPRFs from wPPRFs. Let $F : K \times X \to Y$ be a wPPRF, we build a sPPRF $\hat{F} : K \times X \to Y$ from F as below.

- $\mathsf{Gen}(\lambda)$: run $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, pick $r^* \xleftarrow{\mathrm{R}} X$, set $\hat{pp} = (pp, r^*)$ and k as the secret key.
- $\mathsf{PrivEval}(k, x)$: on input k and x, output $\hat{y} \leftarrow F_k(x+r^*)$ via computing $F.\mathsf{PrivEval}(k, x+r^*)$. This algorithm defines $\hat{F}_k(x) := F_k(x + r^*)$.
- $\mathsf{Puncture}(k, x^*)$: compute $k_{x^*+r^*} \leftarrow F.\mathsf{Puncture}(k, x^* + r^*)$, output $\hat{k}_{x^*} = k_{x^*+r^*}$.
- $\mathsf{PuncEval}(\hat{k}_{x^*}, x)$: parse $\hat{k}_{x^*}$ as $k_{x^*+r^*}$, if $x \neq x^*$ output $y \leftarrow F_{k_{x^*+r^*}}(x+r^*)$ via computing $F.\mathsf{PuncEval}(k_{x^*+r^*}, x + r^*)$, else output $\perp$.

We now reduce the selective pseudorandomness of the above construction to the weak pseudorandomness of the underlying wPPRF. Let $\mathcal{A}$ be an adversary against sPPRF with advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$, we build an adversary $\mathcal{B}$ that breaks wPPRF with the same advantage. $\mathcal{B}$ interacts with $\mathcal{A}$ in the selective pseudorandomness experiment of sPPRF as below:

1. Commit: $\mathcal{A}$ submits its target input $\hat{x^*}$.
2. Setup and Challenge: $\mathcal{B}$ invokes its wPPRF challenger and receives back the wPPRF challenge instance $(pp, k_{x^*}, x^*, y_\beta^*)$ where x^* is randomly chosen from X, y_β^* is either $F_k(x^*)$ if $\beta = 0$ or randomly chosen from Y if $\beta = 1$. $\mathcal{B}$ then sets $r^* = x^* - \hat{x^*}$, $\hat{pp} = (pp, r^*)$, $\hat{k}_{\hat{x^*}} = k_{x^*}$, sends $(\hat{pp}, \hat{k}_{\hat{x^*}}, y_\beta^*)$ to $\mathcal{A}$ as the sPPRF challenge.
3. Guess: $\mathcal{A}$ outputs its guess β' for β and $\mathcal{B}$ forwards β' to its own challenger.

Note that x^* is distributed uniformly at random over X, thereby so is r^*. According to the construction, the punctured key $\hat{k}_{\hat{x^*}}$ at point $\hat{x^*}$ in sPPRF $\hat{F}$ equals the punctured key

$k_{\hat{x}^*+r^*} = k_{x^*}$ at point x^* in wPPRF F. Therefore, $\mathcal{B}$'s simulation is perfect and has the same advantage as $\mathcal{A}$. This proves the forward implication.

The reverse direction that "sPPRFs imply wPPRFs" follows by a simple reduction of weak pseudorandomness to selective pseudorandomness. Let $\mathcal{A}$ be an adversary against wPPRF with advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$, we build an adversary $\mathcal{B}$ that breaks sPPRF with the same advantage. $\mathcal{B}$ interacts with $\mathcal{A}$ in the weak pseudorandomness experiment of wPPRF as below:

1. Setup and Challenge: $\mathcal{B}$ picks $x^* \xleftarrow{R} X$ and submits x^* to its own sPPRF challenger. Upon receiving back (pp, k_{x^*}, y^*_β) where y_β is either $F_k(x^*)$ if $\beta = 0$ or randomly chosen from Y if $\beta = 1$, $\mathcal{B}$ sends $(pp, x^*, k_{x^*}, y^*_\beta)$ to $\mathcal{A}$ as the wPPRF challenge.
2. Guess: $\mathcal{A}$ outputs its guess β' for β and $\mathcal{B}$ forwards β' to its own challenger.

Note that x^* is distributed uniformly over X. Therefore, $\mathcal{B}$'s simulation is perfect and has the same advantage as $\mathcal{A}$. This proves the inverse implication.

The theorem immediately follows. □

3.3 Leakage-Resilient wPRFs from wPPRFs and $i\mathcal{O}$

Now, we show how to construct leakage-resilient wPRFs from wPPRFs and $i\mathcal{O}$. Let $F : K \times X \to Y$ be a wPPRF, $i\mathcal{O}$ be an indistinguishability obfuscator, and $\mathsf{ext} : Y \times S \to Z$ be an average-case (n, ϵ)-strong extractor. In what follows, we build a LR wPRF $\hat{F} : \hat{K} \times \hat{X} \to Z$, where $\hat{X} = X \times S$.

- $\mathsf{Gen}(\lambda)$: run $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, output pp and $\hat{k} \leftarrow i\mathcal{O}(\text{PrivEval})$, where PrivEval is the program defined in Figure 2.
- $\mathsf{PrivEval}(\hat{k}, \hat{x})$: on input $\hat{k}$ and $\hat{x} = (x, s) \in X \times S$, output $y \leftarrow \hat{k}(x, s)$. This algorithm implicitly defines $\hat{F}_{\hat{k}}(\hat{x}) := \mathsf{ext}(F_k(x), s)$.

PrivEval

Constants: wPPRF key k

Input: $\hat{x} = (x, s)$

1. Output $z \leftarrow \mathsf{ext}(F_k(x), s)$.

Figure 2: Program PrivEval. This program is appropriately padded to the maximum of the size of itself and program PrivEval* defined in Figure 3.

PrivEval*

Constants: wPPRF punctured key k_{x^*}, x^*, y^*

Input: $\hat{x} = (x, s)$

1. If $x = x^*$, output $z \leftarrow \mathsf{ext}(y^*, s)$.
2. Else, output $z \leftarrow \mathsf{ext}(F_{k_{x^*}}(x), s)$.

Figure 3: Program PrivEval*

Theorem 3.2. *If F is a secure wPPRF, $i\mathcal{O}$ is indistinguishably secure,* ext *is an average-case (n, ϵ)-strong extractor, the above construction is a ℓ-LR wPRF as long as $\ell \leq \log |Y| - n$.*

Proof. We proceed via a sequence of games. Let S_i be the event that $\mathcal{A}$ wins in Game i.

Game 0. This game is the standard leakage-resilient weak pseudorandomness game for wPRFs. $\mathcal{CH}$ interacts with $\mathcal{A}$ as below:

Setup: $\mathcal{CH}$ runs $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, creates $\hat{k} \leftarrow i\mathcal{O}(\text{PrivEval})$, where the program PrivEval is defined in Figure 2. $\mathcal{CH}$ then sends pp to $\mathcal{A}$.
Phase 1: $\mathcal{A}$ can make evaluation queries and leakage queries. For each evaluation query, $\mathcal{CH}$ chooses $x \xleftarrow{R} X$ and $s \xleftarrow{R} S$ and returns $(x, s, \hat{k}(x, s))$. For each leakage query $\langle f \rangle$, $\mathcal{CH}$ responds with $f(\hat{k})$.
Challenge: $\mathcal{CH}$ chooses $x^* \xleftarrow{R} X$, $s^* \xleftarrow{R} S$ and computes $y^* \leftarrow F_k(x^*)$, then computes $z_0^* \leftarrow \mathsf{ext}(y^*, s^*)$, picks $z_1^* \xleftarrow{R} Z$ and $\beta \xleftarrow{R} \{0, 1\}$, sends z_β^* to $\mathcal{A}$.
Phase 2: $\mathcal{A}$ continues to make evaluation queries. $\mathcal{CH}$ responds the same way as in Phase 1.
Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

According to the definition, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[S_0] - 1/2|$$

Game 1. Same as Game 0 except that $\mathcal{CH}$ chooses $x^* \xleftarrow{R} X$, $s^* \xleftarrow{R} S$ and computes $y^* \leftarrow F_k(x^*)$ in the Setup stage. This change is only conceptual and thus we have:

$$\Pr[S_1] = \Pr[S_0]$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ directly aborts when handling evaluation queries for $x = x^*$.

Let E be the event that there exists one random sample x that equals x^* when $\mathcal{CH}$ emulates evaluation oracle. Clearly, if E never happens, then Game 1 and Game 2 are identical. Suppose $\mathcal{A}$ makes at most q_e evaluation queries. Since $\mathcal{A}$ is a PPT adversary, q_e is bounded by a polynomial in λ. Therefore, $\Pr[E] \leq q_e/|X| \leq \mathsf{negl}(\lambda)$, we have:

$$|\Pr[S_2] - \Pr[S_1]| \leq \Pr[E] \leq \mathsf{negl}(\lambda)$$

Game 3. Same as Game 2 except that $\mathcal{CH}$ computes $k_{x^*} \leftarrow F.\mathsf{Puncture}(k, x^*)$, $y^* \leftarrow F_k(x^*)$, and creates $\hat{k} \leftarrow i\mathcal{O}(\text{PrivEval}^*)$ in the Setup stage. Here, the program PrivEval* (defined in Figure 3) is built from constants k_{x^*}, x^*, y^*.

By the correctness of wPPRFs, the two programs PrivEval and PrivEval* agree on all inputs. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{Adv}_{\mathcal{A}}^{i\mathcal{O}}$$

Game 4. Same as Game 3 except that $\mathcal{CH}$ picks $y^* \xleftarrow{R} Y$ rather than setting $y^* \leftarrow F_k(x^*)$ in the Setup stage.

By a simple reduction to the weak pseudorandomness of wPPRFs, this modification is undetectable for all PPT adversaries. Thus, we have:

$$|\Pr[S_4] - \Pr[S_3]| \leq \mathsf{Adv}_{\mathcal{A}}^{\text{wPPRF}}$$

Game 5. Same as Game 4 except that $\mathcal{CH}$ picks $z_0^* \xleftarrow{R} Z$ rather than setting $z_0^* \leftarrow \mathsf{ext}(y^*, s^*)$ in the Challenge stage.

We denote by V the set of public parameter pp, (x^*, s^*), the responses to all evaluation queries (determined by k_{x^*}), z_1^* and β. In both Game 4 and Game 5, y^* is uniformly chosen from Y (independent of V), thus $\mathsf{H}_\infty(y^*|V) = \log|Y|$. Observe that $\mathcal{A}$ also obtains at most ℓ bits leakage on $\hat{k}$ (denote by $leak$) which is correlated to y^*, it follows by Lemma 2.1 that $\tilde{\mathsf{H}}_\infty(y^*|(V, leak)) \geq \mathsf{H}_\infty(y^*|V) - \ell = \log|Y| - \ell$. Since ext is an average-case (n, ϵ)-strong extractor, we conclude that $\mathsf{ext}(y^*, s^*)$ is ϵ-close to a uniformly random $z_0^* \xleftarrow{R} Z$, even given V and leakage. Note that $\mathcal{A}$'s view in Game 4 and Game 5 is fully determined by z_0^*, V and $leak$, while V and $leak$ are distributed identically in Game 4 and Game 5. Thereby, $\mathcal{A}$'s view in Game 4 and Game 5 are $\epsilon/2$-close. Thus, we have:

$$|\Pr[S_5] - \Pr[S_4]| \leq \epsilon/2 \leq \mathsf{negl}(\lambda)$$

In Game 5, both z_0^* and z_1^* are randomly chosen from Z. Therefore, we have:

$$\Pr[S_5] = 1/2$$

Putting all the above together, the theorem immediately follows. □

We have sketched how to achieve optimal leakage rate in Section 1.3. To avoid repetition, we omit the details here.

Comparison with prior constructions. [Pie09, DY13] showed that any wPRF is already leakage-resilient for a logarithmic leakage bound $\ell = O(\log \lambda)$. Hazay et al. [HLWW13] showed a black-box construction of LR wPRF from any wPRF $F : K \times X \to Y$. Their construction is somewhat involved: they first constructed symmetric-key weak HPS from wPRF, then built LR wPRF from parallel repetition of symmetric-key weak HPS. The consequence is that it is not flexible and efficient. To make the output size larger than $n \log|Y|$, they have to invoke n independent copies of the basic wPRF, and the domain size must be larger than $n(|X| + \log|Y|)$. Besides, its leakage rate is rather poor, say, $O(\log(\lambda)/|k|)$. In contrast, our construction enjoys flexible parameter choice and optimal leakage rate, which is benefited from the non-black-box use of underlying wPPRF via $i\mathcal{O}$.

We also mention a few conceptually different approaches to leakage-resilient symmetric-key cryptosystems such as leakage-resilient SKE based on the minimal use of leak-free components [PSV15] or leakage-resilient PRF in the random oracle or generic group model [BMOS17], which are incomparable to ours.

4 Leakage-Resilient KEM

We begin this section by formally defining leakage-resilient PEPRFs. We then show that leakage-resilient PEPRFs naturally yield leakage-resilient KEM. Towards achieving leakage-resilience for PEPRFs, we first introduce a new notion called puncturable PEPRFs, and construct them from various puncturable primitives, which we believe is of independent interest. Finally, we show how to compile puncturable PEPRFs to leakage-resilient PEPRFs via $i\mathcal{O}$.

4.1 Leakage-Resilient PEPRFs

Chen and Zhang [CZ14] put forwarded the notion of PEPRFs, which is best viewed as a counterpart of weak PRFs in the public-key setting. In PEPRFs, each secret key is associated with a public key, and there is a collection of $\mathcal{NP}$ languages (indexed by public key) defined over

domain. For any element in the language, in addition to evaluating its PRF value using secret key, one can also evaluate it publicly with public key and the associated witness.

PEPRFs neatly capture the essence of KEM, and they can be instantiated from either specific assumptions or more general assumptions such as (extractable) hash proof systems and trapdoor functions. In what follows, we recall the standard definition of PEPRFs from [CZ14] and proceed to introduce leakage resilience for them.

Definition 4.1 (PEPRFs)**.** Let $L = \{L_{pk}\}_{pk \in PK}$ be a collection of $\mathcal{NP}$ languages defined over X. A PEPRF $F : SK \times X \to Y \cup \perp$[11] for L consists of three polynomial time algorithms as below:

- $\mathsf{Gen}(\lambda)$: on input λ, output a public key pk and a secret key sk.
- $\mathsf{PrivEval}(sk, x)$: on input sk and $x \in X$, output $y \leftarrow F_{sk}(x) \in Y \cup \perp$.
- $\mathsf{PubEval}(pk, x, w)$: on input pk and $x \in L_{pk}$ together with a witness w, output $y \leftarrow F_{sk}(x) \in Y$.

To be applicable, L is required to be efficiently samplable, i.e., for each $pk \in PK$, there exists an efficient sampling algorithm $\mathsf{SampRel}$ that on input pk outputs a random element $x \in L_{pk}$ together with a witness w.

Leakage-resilient weak pseudorandomness. Let $\mathcal{A}$ be an adversary against PEPRFs and define its advantage as below:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta' = \beta : \begin{array}{l} (pk, sk) \leftarrow \mathsf{Gen}(\lambda); \\ state \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{leak}}(\cdot)}(pk); \\ (x^*, w^*) \leftarrow \mathsf{SampRel}(pk); \\ y_0^* \leftarrow F_{sk}(x^*), y_1^* \xleftarrow{\mathrm{R}} Y; \\ \beta \xleftarrow{\mathrm{R}} \{0,1\}; \\ \beta' \leftarrow \mathcal{A}(pk, x^*, y_\beta^*); \end{array}\right] - \frac{1}{2}.$$

Here $\mathcal{O}_{\mathsf{leak}}(\cdot)$ is a leakage oracle that on input $f : SK \to \{0,1\}^*$ returns $f(sk)$, subjected to the restriction that the sum of its output lengths is at most ℓ. A PEPRF is ℓ-leakage-resilient weakly pseudorandom if no PPT adversary has non-negligible advantage in the above experiment. As pointed out in [CZ14], full pseudorandomness is impossible due to the publicly evaluable property.

Leakage-Resilient KEM. [CZ14] showed that weakly pseudorandom PEPRF naturally imply CPA-secure KEM. We observe that this implication applies in the leakage setting as well. We sketch the construction here for completeness. Assume $F : SK \times X \to Y$ is a leakage-resilient PEPRF for $L = \{L_{pk}\}_{pk \in PK}$, where the range Y is an additive group. The key pair is exactly the key pair of the underlying PEPRF. To encrypt a message $m \in Y$, one picks $x \xleftarrow{\mathrm{R}} L_{pk}$ with a witness w, computes $k \leftarrow \mathsf{PubEval}(pk, x, w)$ and outputs ciphertext $(x, k + m)$. The decryption process re-computes k via $\mathsf{PrivEval}(k, x)$. The LR CPA security of KEM readily follows from the LR weak pseudorandomness of the underlying PEPRF. The resulting LR CPA-secure KEM can be boosted to LR CPA-secure PKE by combining data encapsulation mechanism (DEM) with appropriate security properties [CS02].

[11] In a PEPRF, when the input x is not in L_{pk}, its PRF value $F_{sk}(x)$ may not be well defined and will be denoted by a distinguished symbol $\perp$.

4.2 Puncturable PEPRFs

To construct leakage-resilient PEPRFs, we first introduce the puncturable version of PEPRFs, called puncturable PEPRFs (PPEPRFs), which could also be viewed as an extension of PPRFs in the public-key setting. We formally define PPEPRFs as below and defer their realizations to Section C.

Definition 4.2 (PPEPRFs)**.** Let $L = \{L_{pk}\}$ be a collection of $\mathcal{NP}$ languages defined over X. A PPEPRF $F : SK \times X \to Y \cup \perp$ for L consists of the following polynomial time algorithms:

- $\mathsf{Gen}(\lambda)$: on input λ, output a public key pk and a secret key sk.
- $\mathsf{PrivEval}(sk, x)$: on input sk and $x \in X$, output $y \leftarrow F_{sk}(x) \in Y \cup \perp$.
- $\mathsf{Puncture}(sk, x^*)$: on input sk and $x^* \in L_{pk}$, output a punctured key sk_{x^*}.
- $\mathsf{PuncEval}(sk_{x^*}, x)$: on input a punctured key sk_{x^*} and $x \neq x^*$, output $y \leftarrow F_{sk}(x) \in Y \cup \perp$.
- $\mathsf{PubEval}(pk, x, w)$: on input pk and $x \in L_{pk}$ together with a witness w, output $y \leftarrow F_{sk}(x) \in Y$.

For security, we require that weak pseudorandomness remains even when the adversary is given a punctured secret key.

Weak pseudorandomness. Let $\mathcal{A}$ be an adversary against PPEPRFs and define its advantage as below:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta' = \beta : \begin{array}{l} (pk, sk) \leftarrow \mathsf{Gen}(\lambda); \\ (x^*, w^*) \leftarrow \mathsf{SampRel}(pk); \\ sk_{x^*} \leftarrow \mathsf{Puncture}(sk, x^*); \\ y_0^* \leftarrow F_{sk}(x^*), y_1^* \xleftarrow{\mathrm{R}} Y; \\ \beta \xleftarrow{\mathrm{R}} \{0,1\}; \\ \beta' \leftarrow \mathcal{A}(pk, sk_{x^*}, x^*, y_\beta^*); \end{array}\right] - \frac{1}{2}.$$

A PPEPRF is weakly pseudorandom if for any PPT adversary $\mathcal{A}$ its advantage in the above experiment is negligible in λ.

Remark 4.1. We note that our notion of PPEPRFs implies puncturable KEM with perfect strong punctured decapsulation soundness [MH15]. As we will see shortly in Section C, PPEPRFs can be constructed from various puncturable primitives. This greatly enriches the constructions of puncturable KEM, which we believe is of independent interest.

4.3 Leakage-Resilient PEPRFs from PPEPRFs and $i\mathcal{O}$

Let $F : SK \times X \to Y \cup \perp$ be a PPEPRF for $L = \{L_{pk}\}_{pk \in PK}$, $i\mathcal{O}$ be an indistinguishability obfuscation, and $\mathsf{ext} : Y \times S \to Z$ be an average-case (n, ϵ)-strong extractor. Without loss of generality, we assume that $Y = \{0,1\}^\rho$. In what follows, we build a leakage-resilient PEPRF $\hat{F} : \hat{SK} \times \hat{X} \to Z \cup \perp$ for $\hat{L} = \{\hat{L}_{pk}\}_{pk \in PK}$, where $\hat{X} = X \times S$ and $\hat{L}_{pk} = \{\hat{x} = (x, s) : x \in L_{pk} \wedge s \in S\}$. According to the definition of $\hat{L}$, a witness w for $x \in L_{pk}$ is also a witness for $\hat{x} = (x, s) \in \hat{L}_{pk}$, where s could be any seed from S.

- $\mathsf{Gen}(\lambda)$: run $F.\mathsf{Gen}(\lambda)$ to obtain (pk, sk), create $\hat{sk} \leftarrow i\mathcal{O}(\mathrm{PrivEval})$, where the program PrivEval is defined in Figure 4; output $(pk, \hat{sk})$.
- $\mathsf{PrivEval}(\hat{sk}, \hat{x})$: on input $\hat{sk}$ and $\hat{x} = (x, s) \in \hat{X}$, output $\hat{y} \leftarrow \hat{sk}(\hat{x})$. This actually defines $\hat{F}_{\hat{sk}}(\hat{x}) := \mathsf{ext}(F_{sk}(x), s)$, where $\hat{x} = (x, s)$.
- $\mathsf{PubEval}(pk, \hat{x}, w)$: on input pk, $\hat{x} = (x, s) \in \hat{L}_{pk}$ and a witness w for $\hat{x}$, compute $y \leftarrow F_{sk}(x)$ via $F.\mathsf{PubEval}(pk, x, w)$, output $\hat{y} \leftarrow \mathsf{ext}(y, s)$.

PrivEval

Constants: PPEPRF secret key sk

Input: $\hat{x} = (x, s)$

1. Output $\mathsf{ext}(F_{sk}(x), s)$.

Figure 4: Program PrivEval. The program is appropriately padded to the maximum of the size of itself and program PrivEval* described in Figure 5.

PrivEval*

Constants: PPEPRF punctured secret key sk_{x^*}, x^* and y^*

Input: $\hat{x} = (x, s)$

1. If $x = x^*$, output $\mathsf{ext}(y^*, s)$.
2. Else, output $\mathsf{ext}(F_{sk_{x^*}}(x), s)$.

Figure 5: Program PrivEval*

Theorem 4.1. *If F is a secure PPEPRF, $i\mathcal{O}$ is indistinguishably secure, and* ext *is an average-case (n, ϵ)-strong extractor, the above PEPRF construction is ℓ-leakage-resilient weakly pseudorandom as long as $\ell \leq \rho - n$.*

Proof. We proceed via a sequence of games. Let S_i be the event that $\mathcal{A}$ succeeds in Game i.

Game 0. This is the standard leakage-resilient weak pseudorandomness game for PEPRFs. $\mathcal{CH}$ interacts with $\mathcal{A}$ as below.

1. Setup: $\mathcal{CH}$ runs $(pk, sk) \leftarrow F.\mathsf{Gen}(\lambda)$, creates $\hat{sk} \leftarrow i\mathcal{O}(\text{PrivEval})$, then sends pk to $\mathcal{A}$.
2. Leakage Query: Upon receiving leakage query $\langle f \rangle$, $\mathcal{CH}$ responds with $f(\hat{sk})$ as long as the total length of leakage is less than ℓ.
3. Challenge: $\mathcal{CH}$ samples $(x^*, w^*) \leftarrow \mathsf{SampRel}(pk)$, picks $s^* \xleftarrow{R} S$, computes $y^* \leftarrow F_{sk}(x^*)$ via $F.\mathsf{PubEval}(pk, x^*, w^*)$, $z_0^* \leftarrow \mathsf{ext}(y^*, s^*)$, samples $z_1^* \xleftarrow{R} Z$, $\beta \xleftarrow{R} \{0, 1\}$. Finally, $\mathcal{CH}$ sends $\hat{x^*} = (x^*, s^*)$ and z_β^* to $\mathcal{A}$.
4. Guess: $\mathcal{A}$ outputs a guess β' for β and wins if $\beta' = \beta$.

According to the definition, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[S_0] - 1/2|$$

Game 1. Same as Game 0 except that $\mathcal{CH}$ samples x^*, w^* and computes $y^* \leftarrow F_{sk}(x^*)$ in the Setup stage. This change is purely conceptual and thus we have:

$$\Pr[S_1] = \Pr[S_0]$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ also computes $sk_{x^*} \leftarrow F.\mathsf{Puncture}(sk, x^*)$ and creates $\hat{sk} \leftarrow i\mathcal{O}(\text{PuncPriv})$ in the Setup stage. Here, the program PrivEval* (defined in Figure 5) is built from constants sk_{x^*}, x^*, and y^*.

It is easy to verify that the two programs PrivEval and PrivEval* agree on all inputs. By a direct reduction to the security of $i\mathcal{O}$, we conclude that:

$$|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}_{\mathcal{A}}^{i\mathcal{O}} \leq \mathsf{negl}(\lambda)$$

Game 3. Same as Game 2 except that $\mathcal{CH}$ picks $y^* \xleftarrow{R} Y$ rather than setting $y^* \leftarrow F_{sk}(x^*)$ in the Setup stage.

Assuming the weak pseudorandomness of the underlying PPEPRF, this modification is undetectable by any PPT adversaries. Thus, we have:

$$|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{Adv}^{\mathrm{PPEPRF}}_{\mathcal{A}} \leq \mathsf{negl}(\lambda)$$

Game 4. Same as Game 3 except that $\mathcal{CH}$ picks $z_0^* \xleftarrow{R} Z$ rather than setting $z_0^* \leftarrow \mathsf{ext}(y^*, s^*)$ in the Challenge stage.

We denote by V the set of public key pk, x^* and s^*. In both Game 3 and Game 4, y^* is uniformly chosen from Y (independent of V), thus $\mathsf{H}_\infty(y^*|V) = \rho$. Observe that $\mathcal{A}$ also obtains at most ℓ bits leakage on $\hat{k}$ (denote by $leak$) which is correlated to y^*, it follows by Lemma 2.1 that $\tilde{\mathsf{H}}_\infty(y^*|(V, leak)) \geq \mathsf{H}_\infty(y^*|V) - \ell = \rho - \ell$, which is greater than n by the parameter choice. Since ext is an average-case (n, ϵ)-strong extractor, we conclude that $\mathsf{ext}(y^*, s^*)$ is ϵ-close to a uniformly random $z_0^* \in Z$, even given V and leakage. Observe that $\mathcal{A}$'s views in Game 3 and Game 4 are fully determined by z_0^*, z_1^*, β^*, V and $leak$, while z_1^*, β^*, V and $leak$ are distributed identically in Game 3 and Game 4. Thereby, $\mathcal{A}$'s views in Game 3 and Game 4 are $\epsilon/2$-close. Thus, we have:

$$|\Pr[S_4] - \Pr[S_3]| \leq \epsilon/2 \leq \mathsf{negl}(\lambda)$$

In Game 4, both z_0^* and z_1^* are randomly chosen from Z. Therefore, we have:

$$\Pr[S_4] = 1/2$$

Putting all the above together, the theorem immediately follows. □

4.4 Construction with Improved Leakage Rate

The leakage rate of the above basic construction is low. Next, we show how to modify it to achieve optimal leakage rate. We need two extra primitives: (1) an IND-CPA secure SKE with message space $\{0,1\}^\rho$ and ciphertext space $\{0,1\}^v$; (2) a family of (v, τ)-lossy functions. The construction is as below.

- $\mathsf{Gen}(\lambda)$: run $(pk, sk) \leftarrow F.\mathsf{Gen}(\lambda)$, $h \leftarrow \mathrm{LF}.\mathsf{GenInj}(\lambda)$, $k_e \leftarrow \mathrm{SKE}.\mathsf{Gen}(\lambda)$, generate a dummy ciphertext $ct \leftarrow \mathrm{SKE}.\mathsf{Enc}(k_e, 0^\rho)$ as $\hat{sk}$, compute $\eta^* \leftarrow h(ct)$, create $C_{\mathrm{eval}} \leftarrow i\mathcal{O}(\mathrm{PrivEval})$ (here the program PrivEval is defined in Figure 6 and η^* acts as its trigger), set $\hat{pk} = (pk, C_{\mathrm{eval}})$, output $(\hat{pk}, \hat{sk})$.
- $\mathsf{PrivEval}(\hat{sk}, \hat{x})$: on input $\hat{sk}$ and $\hat{x} = (x, s) \in \hat{X}$, output $\hat{y} \leftarrow C_{\mathrm{eval}}(\hat{sk}, \hat{x})$. This actually defines $\hat{F}_{\hat{sk}}(\hat{x}) := \mathsf{ext}(F_{sk}(x), s)$, where $\hat{x} = (x, s)$.
- $\mathsf{PubEval}(\hat{pk}, \hat{x}, w)$: on input $\hat{pk} = (pk, C_{\mathrm{eval}}, t)$, $\hat{x} = (x, s) \in \hat{L}_{pk}$ and a witness w for $\hat{x}$, compute $y \leftarrow F_{sk}(x)$ via $F.\mathsf{PubEval}(pk, x, w)$, output $\hat{y} \leftarrow \mathsf{ext}(y, s)$.

Theorem 4.2. *If F is a secure PPEPRF, $i\mathcal{O}$ is indistinguishably secure, SKE is an IND-CPA secure secret-key encryption, LF is a family of (v, τ)-lossy functions, ext is an average-case (n, ϵ)-strong extractor. the above PEPRF construction is ℓ-leakage-resilient weakly pseudorandom as long as $\ell \leq \rho - n - \tau$.*

Proof. By appropriate parameter choice (e.g. setting $v = \rho + o(\rho)$, $n = o(\rho)$, $\tau = o(v)$), we have $|\hat{sk}| = v = \rho + o(\rho)$ and $\ell = \rho - o(\rho)$ and thus the leakage rate is optimal.

We proceed via a sequence of games. Let S_i be the event that $\mathcal{A}$ succeeds in Game i.

PrivEval

Constants: PPEPRF secret key sk, η^*

Input: $\hat{sk}$, $\hat{x} = (x, s)$

1. If $h(\hat{sk}) \neq \eta^*$, output $\perp$.
2. Else, output $\mathsf{ext}(F_{sk}(x), s)$.

Figure 6: Program PrivEval. This program is appropriately padded to the maximum of the size of itself and the program PrivEval* described in Figure 7.

PrivEval*

Constants: PPEPRF punctured secret key sk_{x^*}, k_e, x^* and η^*

Input: $\hat{sk}$, $\hat{x} = (x, s)$

1. If $h(\hat{sk}) \neq \eta^*$, output $\perp$.
2. If $x = x^*$, set $y^* \leftarrow \mathsf{SKE.Dec}(k_e, \hat{sk})$, output $\mathsf{ext}(y^*, s)$.
3. Else, output $\mathsf{ext}(F_{sk_{x^*}}(x), s)$.

Figure 7: Program PuncEval

Game 0. This is the standard leakage-resilient weak pseudorandomness game for PEPRFs. $\mathcal{CH}$ interacts with $\mathcal{A}$ as below.

1. Setup: $\mathcal{CH}$ runs $(pk, sk) \leftarrow F.\mathsf{Gen}(\lambda)$, $h \leftarrow \mathrm{LF}.\mathsf{GenInj}(\lambda)$, samples $k_e \leftarrow \mathrm{SKE}.\mathsf{Gen}(\lambda)$, generates a dummy ciphertext $ct \leftarrow \mathrm{SKE}.\mathsf{Enc}(k_e, 0^\rho)$ as $\hat{sk}$, computes $\eta^* \leftarrow h(ct)$, creates $C_{\mathrm{eval}} \leftarrow i\mathcal{O}(\mathrm{PrivEval})$. $\mathcal{CH}$ sets $\hat{pk} = (pk, C_{\mathrm{eval}})$ and sends it to $\mathcal{A}$.
2. Leakage Query: Upon receiving leakage query $\langle f \rangle$, $\mathcal{CH}$ responds with $f(\hat{sk})$ as long as the total leakage is less than ℓ.
3. Challenge: $\mathcal{CH}$ samples $(x^*, w^*) \leftarrow \mathsf{SampRel}(pk)$, picks $s^* \xleftarrow{\mathrm{R}} S$, computes $y^* \leftarrow F_{sk}(x^*)$ via $F.\mathsf{PubEval}(pk, x^*, w^*)$, $z_0^* \leftarrow \mathsf{ext}(y^*, s^*)$, samples $z_1^* \xleftarrow{\mathrm{R}} Z$, $\beta \xleftarrow{\mathrm{R}} \{0, 1\}$. Finally, $\mathcal{CH}$ sends $\hat{x^*} = (x^*, s^*)$ and z_β^* to $\mathcal{A}$.
4. Guess: $\mathcal{A}$ outputs a guess β' for β and wins if $\beta' = \beta$.

According to the definition, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[S_0] - 1/2|$$

Game 1. Same as Game 0 except that $\mathcal{CH}$ samples x^*, w^* and computes $y^* \leftarrow F_{sk}(x^*)$ in the Setup stage. This change is purely conceptual and thus we have:

$$\Pr[S_1] = \Pr[S_0]$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ computes $ct \leftarrow \mathrm{SKE}.\mathsf{Enc}(k_e, y^*)$ rather than $ct \leftarrow \mathrm{SKE}.\mathsf{Enc}(k_e, 0^\rho)$ in the Setup stage. By a direct reduction to the IND-CPA security of SKE, we have:

$$|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathrm{SKE}}$$

Game 3. Same as Game 2 except that $\mathcal{CH}$ also computes $sk_{x^*} \leftarrow F.\mathsf{Puncture}(sk, x^*)$ and creates $C_{\text{eval}} \leftarrow i\mathcal{O}(\text{PrivEval})$ in the Setup stage. Here, the program PrivEval* (defined in Figure 7) is built from constants (sk_{x^*}, x^*, y^*).

By the injectivity of h and the correctness of SKE and PPEPRF, the two programs PrivEval and PuncPriv agree on all inputs. By a direct reduction to the security of $i\mathcal{O}$, we conclude that:

$$|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{Adv}^{i\mathcal{O}}_{\mathcal{A}}$$

Game 4. Same as Game 3 except that in the Setup stage $\mathcal{CH}$ picks $y^* \xleftarrow{R} Y$ rather than setting $y^* \leftarrow F_{sk}(x^*)$.

Assuming the weak pseudorandomness of the underlying PPEPRF, this modification is undetectable by all PPT adversaries. Thus, we have:

$$|\Pr[S_4] - \Pr[S_3]| \leq \mathsf{Adv}^{\text{PPEPRF}}_{\mathcal{A}}$$

Game 5. Same as Game 4 except that $\mathcal{CH}$ samples a lossy function h via $\mathsf{LF.GenLossy}(\lambda)$ rather than sampling an injective function in the Setup stage. By a direct reduction to the security of lossy functions, we conclude that:

$$|\Pr[S_5] - \Pr[S_4]| \leq \mathsf{Adv}^{\text{LF}}_{\mathcal{A}}$$

Game 6. Same as Game 5 except that $\mathcal{CH}$ picks $z_0^* \xleftarrow{R} Z$ rather than setting $z_0^* \leftarrow \mathsf{ext}(y^*, s^*)$ in the Challenge stage.

We denote by V the set of public key $\hat{pk} = (pk, C_{\text{eval}})$, x^* and s^*. In both Game 5 and Game 6, y^* is uniformly chosen from Y (independent of sk_{x^*}, x^* and s^*) and but is correlated to η^* which has at most 2^τ values, we have $\mathsf{H}_\infty(y^*|V) \geq \rho - \tau$ by Lemma 2.1. Observe that $\mathcal{A}$ also obtains at most ℓ bits leakage on $\hat{sk}$ (denote by $leak$) which is correlated to y^*, it follows by Lemma 2.1 that $\tilde{\mathsf{H}}_\infty(y^*|(V, leak)) \geq \mathsf{H}_\infty(y^*|V) - \ell = \rho - \tau - \ell$, which is greater than n by the parameter choice. Since ext is an average-case (n, ϵ)-strong extractor, we conclude that $\mathsf{ext}(y^*, s^*)$ is ϵ-close to a uniformly random $z_0^* \in Z$, even given V and leakage. Observe that $\mathcal{A}$'s view in Game 5 and Game 6 are fully determined by z_0^*, z_1^*, β^*, V and $leak$, while z_1^*, β^*, V and $leak$ are distributed identically in Game 5 and Game 6. Thereby, $\mathcal{A}$'s view in Game 5 and Game 6 are $\epsilon/2$-close. Thus, we have:

$$|\Pr[S_6] - \Pr[S_5]| \leq \epsilon/2 \leq \mathsf{negl}(\lambda)$$

In Game 6, both z_0^* and z_1^* are randomly chosen from Z. Therefore, we have:

$$\Pr[S_6] = 1/2$$

Putting all the above together, the theorem immediately follows. □

5 Leakage-Resilient Signature

To best illustrate our idea, in the section we only present the construction with selective security. The construction with adaptive security is deferred to Section D.

5.1 Selective Construction from sPPRFs, Leakage-Resilient OWFs and $i\mathcal{O}$

Let $F : K \times M \to \{0,1\}^n$ be a sPPRF, $i\mathcal{O}$ be an indistinguishability obfuscator, $g : \{0,1\}^n \to \{0,1\}^\mu$ be a leakage-resilient OWF. We build a leakage-resilient signature as below.

- $\mathsf{Gen}(\lambda)$: run $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, create $sk \leftarrow i\mathcal{O}(\text{Sign})$ and $vk \leftarrow i\mathcal{O}(\text{Verify})$. The programs Sign and Verify are defined in Figure 8 and Figure 10 respectively.
- $\mathsf{Sign}(sk, m)$: output $\sigma \leftarrow sk(m)$.
- $\mathsf{Verify}(vk, m, \sigma)$: output $vk(m, \sigma)$.

Sign

Constants: sPPRF key k

Input: message m

1. Compute $\sigma \leftarrow F(k, m)$.

Figure 8: Program Sign. This program is appropriately padded to the maximum of the size of itself and program Sign* defined in Figure 9.

Sign*

Constants: sPPRF punctured key k_{m^*}, m^*, σ^*

Input: message m

1. If $m = m^*$, output σ^*.
2. Else, output $\sigma \leftarrow F(k_{m^*}, m)$.

Figure 9: Program Sign*

Verify

Constants: sPPRF key k

Input: message m and signature σ

1. Test if $g(\sigma) = g(F(k, m))$, output 1 if true and 0 if false.

Figure 10: Program Verify. This program is appropriately padded to the maximum of the size of itself and the program Verify* defined in Figure 11.

Verify*

Constants: sPPRF punctured key k_{m^*}, m^* and y^*

Input: message m and signature σ

1. If $m = m^*$, test whether $g(\sigma) = y^*$. Output 1 if true and 0 if false.
2. Else, test if $g(\sigma) = g(F(k_{m^*}, m))$. Output 1 if true and 0 if false.

Figure 11: Program Verify*

Theorem 5.1. *If F is a secure sPPRF, $i\mathcal{O}$ is indistinguishably secure, g is ℓ-leakage-resilient one-way, the above construction is ℓ-leakage-resilient EUF-CMA in the selective sense.*

Proof. We proceed via a sequence of games. Let S_i be the probability that $\mathcal{A}$ wins in Game i.

Game 0. This is the standard leakage-resilient selective EUF-CMA game for signature. $\mathcal{CH}$ interacts with $\mathcal{A}$ as follows:

1. Commit: $\mathcal{A}$ submits the target message m^* to $\mathcal{CH}$.
2. Setup: $\mathcal{CH}$ runs $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, creates $sk \leftarrow i\mathcal{O}(\text{Sign})$, $vk \leftarrow i\mathcal{O}(\text{Verify})$. $\mathcal{CH}$ sends vk to $\mathcal{A}$.
3. Signing Query: Upon receiving signing query $\langle m \rangle \neq \langle m^* \rangle$, $\mathcal{CH}$ responds with $\sigma \leftarrow sk(m)$.
4. Leakage Query: Upon receiving leakage query $\langle f \rangle$, $\mathcal{CH}$ responds with $f(sk)$.
5. Forge: $\mathcal{A}$ outputs a forgery σ' and wins if $\mathsf{Verify}(vk, m^*, \sigma') = 1$.

According to the definition of $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_0]$$

Game 1. Same as Game 0 except that in the Setup stage $\mathcal{CH}$ computes $\sigma^* \leftarrow F(k, m^*)$, $y^* \leftarrow g(\sigma^*)$, and $k_{m^*} \leftarrow F.\mathsf{Puncture}(k, m^*)$, creates $vk \leftarrow i\mathcal{O}(\text{Verify}^*)$, where the program Verify* is defined in Figure 11.

It is easy to check that the programs Verify and Verify* agree on all inputs. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}^{i\mathcal{O}}_{\mathcal{A}}$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ uses k_{m^*} to handle signing queries, i.e., returning $\sigma \leftarrow F(k_{m^*}, m)$ for $m \neq m^*$. By the correctness of sPPRF, Game 1 and Game 2 are identical in $\mathcal{A}$'s view. Thus, we have:

$$\Pr[S_2] = \Pr[S_1]$$

Game 3. Same as Game 2 except that $\mathcal{CH}$ creates $sk \leftarrow i\mathcal{O}(\text{Sign}^*)$ in the Setup stage. Here the program Sign* (defined in Figure 9) is built from constants k_{m^*}, m^* and σ^*.

It is easy to check that the two programs Sign and Sign* agree on all inputs. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{Adv}^{i\mathcal{O}}_{\mathcal{A}}$$

Game 4. Same as Game 3 except that in Setup stage $\mathcal{CH}$ picks $\sigma^* \xleftarrow{R} \{0,1\}^n$ rather than setting $\sigma^* \leftarrow F(k, m^*)$.

By the selective pseudorandomness of sPPRF, we have:

$$|\Pr[S_4] - \Pr[S_3]| \leq \mathsf{Adv}^{\text{sPPRF}}_{\mathcal{A}} \tag{1}$$

It remains to analyze $\Pr[S_4]$. We have the following claim.

Claim 5.2. *If g is an ℓ-leakage-resilient OWF, then the advantage of any PPT adversary in Game 4 is negligible in λ.*

Proof. Let $\mathcal{A}$ be a PPT adversary wins Game 4 with advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$. We construct an adversary $\mathcal{B}$ that breaks the assumed leakage-resilient one-wayness of g with the same advantage, implying that $\Pr[S_4]$ must be negligible.

Given (g, y^*) where $y^* \leftarrow g(\sigma^*)$ for some $\sigma^* \xleftarrow{R} \{0,1\}^n$, $\mathcal{B}$ interacts with $\mathcal{A}$ in Game 4 with the aim to output σ' such that $g(\sigma') = y^*$.

1. Commit: $\mathcal{A}$ submits the target message m^* to $\mathcal{CH}$.

2. Setup: $\mathcal{B}$ runs $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, computes $k_{m^*} \leftarrow F.\mathsf{Puncture}(k, m^*)$, creates $vk \leftarrow i\mathcal{O}(\text{Verify}^*)$, and sends vk to $\mathcal{A}$. $\mathcal{B}$ also picks random coins r used for obfuscating the program Sign^* (with constants k_{m^*}, m^*, σ^* hardwired) for later simulation. Note that the constant σ^* is unknown to $\mathcal{B}$.
3. Signing Query: Upon receiving signing query $\langle m \rangle \neq \langle m^* \rangle$, $\mathcal{B}$ responds with $\sigma \leftarrow F(k_{m^*}, m)$ using k_{m^*}.
4. Leakage Query: Note that the signing key $sk \leftarrow i\mathcal{O}(\mathsf{Sign}^*_{k_{m^*}, m^*, \sigma^*}; r)$ could be viewed as the value of some function $\psi(\cdot)$ at point σ^*, where $\psi(\cdot)$ on input σ outputs $i\mathcal{O}(\mathsf{Sign}^*_{k_{m^*}, m^*, \sigma}; r)$. Since $i\mathcal{O}$ is efficiently computable, so is $\psi(\cdot)$. Based on this observation, $\mathcal{B}$ can transform any leakage queries on sk to leakage queries on σ^*. Upon receiving leakage query $\langle f \rangle$, $\mathcal{B}$ makes leakage query $\langle f \circ \psi \rangle$ to its own challenger and forwards the reply to $\mathcal{A}$.
5. Forge: $\mathcal{A}$ outputs a forgery σ' and wins if $\mathsf{Verify}(vk, m^*, \sigma') = 1$.

Finally, $\mathcal{B}$ forwards σ' to its challenger. It is straightforward to verify that $\mathcal{B}$'s simulation for Game 4 is perfect. If $\mathcal{A}$ succeeds, according to the definition of algorithm Verify in Game 4, σ' is indeed a preimage of y^* under g, thus $\mathcal{B}$ also succeeds. This proves the claim. □

Putting all the above together, the theorem immediately follows. □

5.2 Construction with Improved Leakage Rate

The basic construction presented in Section 5.1 inherits the same leakage amount from the underlying LR-OWF. However, the leakage rate is low. Next, we show how to modify it to achieve optimal leakage rate. We need two extra primitives: (1) an IND-CPA secure SKE with message space $\{0,1\}^n$ and ciphertext space $\{0,1\}^v$; (2) a family of (v, τ)-lossy functions. Besides, we require the underlying one-way function g to be leakage-resilient in the entropy leakage model (cf. definition in Section 2.2). The construction is as below.

- $\mathsf{Gen}(\lambda)$: run $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, $h \leftarrow \text{LF}.\mathsf{GenInj}(\lambda)$, $k_e \leftarrow \text{SKE}.\mathsf{Gen}(\lambda)$, generate a dummy ciphertext $ct \leftarrow \text{SKE}.\mathsf{Enc}(k_e, 0^n)$, compute $\eta^* \leftarrow h(ct)$, create $C_{\text{sign}} \leftarrow i\mathcal{O}(\text{Sign})$ and $C_{\text{vefy}} \leftarrow i\mathcal{O}(\text{Verify})$. The programs Sign and Verify are defined in Figure 12 and Figure 14 respectively. Finally, output $vk = (C_{\text{vefy}}, C_{\text{sign}})$ and $sk = ct$.
- $\mathsf{Sign}(sk, m)$: output $\sigma \leftarrow C_{\text{sign}}(ct, m)$.
- $\mathsf{Verify}(vk, m, \sigma)$: output $C_{\text{vefy}}(m, \sigma)$.

Sign

Constants: sPPRF key k and η^*

Input: ct, message m

1. If $h(ct) \neq \eta^*$, output $\perp$.
2. Else, compute $\sigma \leftarrow F(k, m)$.

Figure 12: Program Sign. This program is appropriately padded to the maximum of the size of itself and the program Sign^* defined in Figure 13.

Theorem 5.3. *If F is a secure sPPRF, $i\mathcal{O}$ is indistinguishably secure, SKE is IND-CPA secure, LF is a family of (v, τ)-lossy functions, g is ℓ-entropy-leakage-resilient one-way, the above construction is ℓ'-leakage-resilient EUF-CMA in the selective sense as long as $\ell' \leq \ell - \tau$.*

Sign*

Constants: sPPRF punctured key k_{m^*}, k_e, m^* and η^*.

Input: ct, message m

1. If $h(ct) \neq \eta^*$, output $\perp$.
2. If $m = m^*$, output $\sigma^* \leftarrow \mathsf{SKE.Dec}(k_e, ct)$.
3. Else, output $\sigma \leftarrow F(k_{m^*}, m)$.

Figure 13: Program Sign*

Verify

Constants: sPPRF key k

Input: message m and signature σ

1. Test if $g(\sigma) = g(F(k, m))$, output 1 if true and 0 if false.

Figure 14: Program Verify. This program is appropriately padded to the maximum of the size of itself and the program Verify* defined in Figure 15.

Verify*

Constants: sPPRF punctured key k_{m^*}, m^* and y^*

Input: message m and signature σ

1. If $m = m^*$, test whether $g(\sigma) = y^*$. Output 1 if true and 0 if false.
2. Else, test if $g(\sigma) = g(F(k_{m^*}, m))$. Output 1 if true and 0 if false.

Figure 15: Program Verify*

Proof. By appropriate parameter choice (e.g. setting $v = n + o(n)$, $\ell = n - o(n)$, $\tau = o(v)$), we have $|sk| = v = n + o(n)$ and $\ell' = n - o(n)$ and thus the leakage rate is optimal.

We then prove the security via a sequence of games. Let S_i be the probability that $\mathcal{A}$ wins in Game i.

Game 0. This is the standard leakage-resilient selective EUF-CMA game for signature. $\mathcal{CH}$ interacts with $\mathcal{A}$ as follows:

1. Commit: $\mathcal{A}$ submits the target message m^* to $\mathcal{CH}$.
2. Setup: $\mathcal{CH}$ runs $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, $h \leftarrow \mathrm{LF}.\mathsf{GenInj}(\lambda)$, samples a fresh key $k_e \leftarrow \mathrm{SKE}.\mathsf{Gen}(\lambda)$, computes $ct \leftarrow \mathrm{SKE}.\mathsf{Enc}(k_e, 0^n)$ and $\eta^* \leftarrow h(ct)$, creates $C_{\mathrm{sign}} \leftarrow i\mathcal{O}(\mathrm{Sign})$ and $C_{\mathrm{vefy}} \leftarrow i\mathcal{O}(\mathrm{Verify})$. $\mathcal{CH}$ sets $sk = ct$ and sends $vk = (C_{\mathrm{sign}}, C_{\mathrm{vefy}})$ to $\mathcal{A}$.
3. Signing Query: Upon receiving signing query $\langle m \rangle \neq \langle m^* \rangle$, $\mathcal{CH}$ responds with $\sigma \leftarrow C_{\mathrm{sign}}(m)$.
4. Leakage Query: Upon receiving leakage query $\langle f \rangle$, $\mathcal{CH}$ responds with $f(sk)$.
5. Forge: $\mathcal{A}$ outputs a forgery σ' and wins if $C_{\mathrm{vefy}}(m^*, \sigma') = 1$.

According to the definition of $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_0]$$

Game 1. Same as Game 0 except that $\mathcal{CH}$ computes $\sigma^* \leftarrow F(k, m^*)$, $y^* \leftarrow g(\sigma^*)$, $k_{m^*} \leftarrow F.\mathsf{Puncture}(k, m^*)$, then creates $vk \leftarrow i\mathcal{O}(\text{Verify}^*)$ in the Setup stage. Here, the program Verify* (defined in Figure 15) is built from constants k_{m^*}, m^* and y^*.

It is easy to check that the two programs Verify and Verify* agree on all inputs. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_1] - \Pr[S_0]| \leq \mathsf{Adv}^{i\mathcal{O}}_{\mathcal{A}}$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ uses k_{m^*} to handle signing queries, i.e., returning $\sigma \leftarrow F(k_{m^*}, m)$ for $m \neq m^*$. By the correctness of sPPRFs, Game 1 and Game 2 are identical in $\mathcal{A}$'s view. Thus, we have:

$$\Pr[S_2] = \Pr[S_1]$$

After switching C_{vefy} from $i\mathcal{O}(\text{Verify})$ to $i\mathcal{O}(\text{Verify}^*)$, we are going to switch C_{sign} from $i\mathcal{O}(\text{Sign})$ to $i\mathcal{O}(\text{Sign}^*)$. To ensure that such change is undetectable using $i\mathcal{O}$, we have to switch the secret key ct from a dummy encryption of 0^n to an encryption of σ^*.

Game 3. Same as Game 2 except that in the Setup stage $\mathcal{CH}$ generates $ct \leftarrow \text{SKE.}\mathsf{Enc}(k_e, \sigma^*)$ rather than $ct \leftarrow \text{SKE.}\mathsf{Enc}(k_e, 0^n)$. Thus, we have:

$$|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{Adv}^{\text{SKE}}_{\mathcal{A}}$$

Game 4. Same as Game 3 except that in the Setup stage $\mathcal{CH}$ creates $C_{\text{sign}} \leftarrow i\mathcal{O}(\text{Sign}^*)$, where the program Sign* is defined in Figure 13.

By the correctness of sPPRF, SKE, and the injectivity of h, the two programs Sign and Sign* agree on all inputs. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_4] - \Pr[S_3]| \leq \mathsf{Adv}^{i\mathcal{O}}_{\mathcal{A}}$$

Game 5. Same as Game 4 except that in Setup stage $\mathcal{CH}$ picks $\sigma^* \xleftarrow{\text{R}} \{0,1\}^n$ rather than setting $\sigma^* \leftarrow F(k, m^*)$.

By the selective pseudorandomness of sPPRF, we have:

$$|\Pr[S_5] - \Pr[S_4]| \leq \mathsf{Adv}^{\text{sPPRF}}_{\mathcal{A}} \tag{2}$$

Game 6. Same as Game 5 except that in the Setup stage $\mathcal{CH}$ samples a lossy function via $h \leftarrow \text{LF.}\mathsf{GenLossy}(\lambda)$ rather than an injective function. By a direct reduction to the security of LF, we conclude that:

$$|\Pr[S_6] - \Pr[S_5]| \leq \mathsf{Adv}^{\text{LF}}_{\mathcal{A}} \tag{3}$$

It remains to analyze $\Pr[S_6]$. We have the following claim.

Claim 5.4. *If g is an ℓ-entropy-leakage-resilient OWF, then the advantage of any PPT adversary in Game 6 is negligible in λ.*

Proof. Let $\mathcal{A}$ be a PPT adversary that wins Game 6 with advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$. We construct an adversary $\mathcal{B}$ breaking the assumed entropy-leakage-resilient one-wayness of g with the same advantage, implying that $\Pr[S_6]$ must be negligible.

Given (g, y^*) where $y^* \leftarrow g(\sigma^*)$ for some $\sigma^* \xleftarrow{\text{R}} \{0,1\}^n$, $\mathcal{B}$ interacts with $\mathcal{A}$ in Game 6 with the aim to output σ' such that $g(\sigma') = y^*$.

1. Commit: $\mathcal{A}$ submits the target message m^* to $\mathcal{B}$.

2. Setup: $\mathcal{B}$ runs $(pp, k) \leftarrow F.\mathsf{Gen}(\lambda)$, then computes $k_{m^*} \leftarrow F.\mathsf{Puncture}(k, m^*)$, picks $k_e \leftarrow \mathrm{SKE}.\mathsf{Gen}(\lambda)$, $h \leftarrow \mathrm{LF.GenLossy}(\lambda)$; creates $C_{\mathrm{vefy}} \leftarrow i\mathcal{O}(\mathrm{Verify}^*)$ and $C_{\mathrm{sign}} \leftarrow i\mathcal{O}(\mathrm{Sign}^*)$. We remark that there is a subtlety when building the program Sign^*. Note that the program Sign^* has constants k_{m^*}, k_e, m^* and η^* hardwired. According to the definition, $\eta^* = h(ct)$, where ct is an encryption of σ^* under k_e. However, σ^* is unknown to $\mathcal{B}$. Observe that η^* reveals at most τ bits information about ct and thus σ^*, because h is a (v, τ)-lossy function. Again, note that both h and $\mathrm{SKE}.\mathsf{Enc}$ are efficiently computable, thus η^* can be expressed as the value of some entropy-bounded function at point σ^*. More precisely, $\mathcal{B}$ picks a fresh randomness r for encryption, defines function $\psi(\cdot) := \mathrm{SKE}.\mathsf{Enc}(k_e, \cdot; r)$. In this way, $\mathcal{B}$ can obtain η^* by making a leakage query $\langle h \circ \psi \rangle$ about σ^* to its own challenger.
3. Signing Query: Upon receiving signing query $\langle m \rangle \neq \langle m^* \rangle$, $\mathcal{B}$ responds with $\sigma \leftarrow F(k_{m^*}, m)$ using k_{m^*}.
4. Leakage Query: Note that for any leakage query on $sk = ct$, $\mathcal{B}$ can transform it to leakage queries on σ^*. Upon receiving leakage query $\langle f \rangle$, $\mathcal{B}$ makes leakage query $\langle f \circ \psi \rangle$ about σ^* to its own challenger and forwards the reply to $\mathcal{A}$.
5. Forge: $\mathcal{A}$ outputs a forgery σ' and wins if $\mathsf{Verify}(vk, m^*, \sigma') = 1$.

Finally, $\mathcal{B}$ forwards σ' to its challenger. By the parameter choice of $\ell' + \tau \leq \ell$, $\mathcal{B}$'s simulation for Game 6 is perfect. If $\mathcal{A}$ succeeds, according to the definition of algorithm Verify in Game 6, σ' is indeed a preimage of y^* under g, thus $\mathcal{B}$ also succeeds. This proves the claim. □

Putting all the above together, the theorem immediately follows. □

Remark 5.1. Note that the length of η^* in the verification key is potentially larger than n and thus ℓ due to the ciphertext expansion of SKE and the fact that LF is a family of lossy functions. Therefore, standard length-bounded LR-OWF is not sufficient to make the simulation go through. Luckily, in Game 6 the value η^* is an output of lossy function, which reveals at most τ bits entropy about σ^*. This explains why we have to require that the underlying OWF is entropy leakage-resilient (cf. definition and construction in Section 2.3.2).

Acknowledgment

We thank the anonymous reviewers of Asiacrypt 2018 for their helpful comments. The first author is supported by the National Natural Science Foundation of China (Grant No. 61772522), Youth Innovation Promotion Association CAS, Key Research Program of Frontier Sciences, CAS (Grant No. QYZDB-SSW-SYS035).

References

[ADN+10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In *EUROCRYPT*, pages 113–134, 2010.

[ADW09a] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[ADW09b] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In *ICITS*, pages 1–18, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[BCH12] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage-tolerant interactive protocols. In *TCC*, pages 266–284, 2012.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT*, pages 37–51, 1997.

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In *CRYPTO*, pages 1–20, 2010.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

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

[BK12] Zvika Brakerski and Yael Tauman Kalai. A parallel repetition theorem for leakage resilience. In *TCC*, pages 248–265, 2012.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510, 2010.

[BMOS17] Guy Barwell, Daniel P. Martin, Elisabeth Oswald, and Martijn Stam. Authenticated encryption in the face of protocol and side channel leakage. In *ASIACRYPT*, pages 693–723, 2017.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[BSW11] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. In *EUROCRYPT*, pages 89–108, 2011.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In *EUROCRYPT*, pages 792–821, 2016.

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

[CDRW10] Sherman S. M. Chow, Yevgeniy Dodis, Yannis Rouselakis, and Brent Waters. Practical leakage-resilient identity-based encryption from simple assumptions. In *ACM CCS*, pages 152–161, 2010.

[CQX18] Yu Chen, Baodong Qin, and Haiyang Xue. Regularly lossy functions and their applications. In *CT-RSA*, 2018.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002.

[CZ14] Yu Chen and Zongyang Zhang. Publicly evaluable pseudorandom functions and their applications. In *SCN*, pages 115–134, 2014.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

[DGK+10] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[DGL+16] Dana Dachman-Soled, S. Dov Gordon, Feng-Hao Liu, Adam O'Neill, and Hong-Sheng Zhou. Leakage-resilient public-key encryption from obfuscation. In *PKC*, pages 101–128, 2016.

[DHLAW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DHLW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[DY13] Yevgeniy Dodis and Yu Yu. Overcoming weak expectations. In *TCC*, pages 1–22, 2013.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *CRYPTO*, pages 518–535, 2014.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GJS11] Sanjam Garg, Abhishek Jain, and Amit Sahai. Leakage-resilient zero knowledge. In *CRYPTO*, pages 297–315, 2011.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *ICS*, pages 230–240, 2010.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[HL11] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *TCC*, pages 107–124, 2011.

[HLWW13] Carmit Hazay, Adriana López-Alt, Hoeteck Wee, and Daniel Wichs. Leakage-resilient cryptography from minimal assumptions. In *EUROCRYPT*, pages 160–176, 2013.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[HW09] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In *CRYPTO*, pages 654–670, 2009.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *TCC*, pages 668–697, 2015.

[JLL16] Wenpan Jing, Xianhui Lu, and Bao Li. Leakage-resilient IND-CCA KEM from the extractable hash proofs with indistinguishability obfuscation. In *Inscrypt*, pages 291–308, 2016.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[KMO10] Eike Kiltz, Payman Mohassel, and Adam O'Neill. Adaptive trapdoor functions and chosen-ciphertext security. In *EUROCRYPT*, pages 673–692, 2010.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1996.

[Kom16] Ilan Komargodski. Leakage resilient one-way functions: The auxiliary-input setting. In *TCC*, pages 139–158, 2016.

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

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[LLW11] Allison B. Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In *STOC*, pages 725–734, 2011.

[LRW11] Allison B. Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, pages 70–88, 2011.

[LWZ13] Shengli Liu, Jian Weng, and Yunlei Zhao. Efficient public key cryptosystem resilient to key leakage chosen ciphertext attacks. In *CT-RSA*, pages 84–100, 2013.

[MH15] Takahiro Matsuda and Goichiro Hanaoka. Constructing and understanding chosen ciphertext security via puncturable key encapsulation mechanisms. In *TCC*, pages 561–590, 2015.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[MTVY11] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In *TCC*, pages 89–106, 2011.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *Advances in Cryptology - EUROCRYPT 2009*, pages 462–482, 2009.

[PSV15] Olivier Pereira, François-Xavier Standaert, and Srinivas Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *ACM SIGSAC*, pages 96–108, 2015.

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

[QL13] Baodong Qin and Shengli Liu. Leakage-resilient chosen-ciphertext secure public-key encryption from hash proof system and one-time lossy filter. In *ASIACRYPT*, pages 381–400, 2013.

[QL14] Baodong Qin and Shengli Liu. Leakage-flexible cca-secure public-key encryption: Simple construction and free of pairing. In *PKC*, pages 19–36, 2014.

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

[RS09] Alon Rosen and Gil Segev. Chosen-ciphertext security via correlated products. In *TCC*, pages 419–436, 2009.

[RW14] Kim Ramchen and Brent Waters. Fully secure and fast signing from obfuscation. In *ACM CCS*, pages 659–673, 2014.

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

[Wee10] Hoeteck Wee. Efficient chosen-ciphertext security via extractable hash proofs. In *CRYPTO*, pages 314–332, 2010.

[Wic13] Daniel Wichs. Barriers in cryptography with weak, correlated and leaky sources. In *Innovations in Theoretical Computer Science, ITCS*, pages 111–126, 2013.

[WMHT16] Yuyu Wang, Takahiro Matsuda, Goichiro Hanaoka, and Keisuke Tanaka. Signatures resilient to uninvertible leakage. In *SCN*, pages 372–390, 2016.

[YCZY12] Tsz Hon Yuen, Sherman S. M. Chow, Ye Zhang, and Siu-Ming Yiu. Identity-based encryption resilient to continual auxiliary leakage. In *EUROCRYPT*, pages 117–134, 2012.

[YXZ+15] Rupeng Yang, Qiuliang Xu, Yongbin Zhou, Rui Zhang, Chengyu Hu, and Zuoxia Yu. Updatable hash proof system and its applications. In *ESORICS*, pages 266–285, 2015.

[Zha16] Mark Zhandry. The Magic of ELFs. In *CRYPTO*, pages 479–508, 2016.

A Puncturable TDFs

Standard trapdoor functions (TDFs) do not support puncturable property, in that the functionality of its trapdoor is of *all-or-nothing* flavor.

In this section, we introduce a new notion named puncturable TDFs (PTDFs) and show how to construct them from correlated-product TDFs [RS09]. Briefly, PTDF allows one to derive a punctured trapdoor from the master trapdoor w.r.t. a particular element in the range, and such punctured trapdoor can invert all the elements but the punctured element. The security of PTDFs requires that one-wayness remains even in the presence of a punctured key.

Definition A.1 (PTDFs)**.** A PTDF $G : EK \times X \to Y$ consists of the following polynomial time algorithms:

- $\mathsf{Gen}(\lambda)$: on input λ, output an evaluation key ek and a master trapdoor td. Each ek defines a deterministic function $G_{ek} : X \to Y$.
- $\mathsf{Eval}(ek, x)$: on input ek and x, output $y \leftarrow G_{ek}(x)$.
- $\mathsf{Puncture}(td, y^*)$: on input a master trapdoor td and an element $y^* \in \mathrm{Img}(G_{ek})$, output a punctured trapdoor td_{y^*}.
- $\mathsf{Inv}(td, y)$: on input a trapdoor td and an element $y \in Y$, output $x \leftarrow G_{ek}^{-1}(y)$.[12]
- $\mathsf{PuncInv}(td_{y^*}, y)$: on input a punctured trapdoor td_{y^*} and an element $y \in Y$, output $x \leftarrow G_{ek}^{-1}(y)$ if $y \neq y^*$ and $\perp$ otherwise.

One-wayness. Let $\mathcal{A}$ be a PPT adversary against PTDFs and define its advantage in the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[G_{ek}(x) = y^* : \begin{array}{l} (ek, td) \leftarrow \mathsf{Gen}(\lambda); \\ x^* \xleftarrow{\mathrm{R}} X, y^* \leftarrow G_{ek}(x^*); \\ td_{y^*} \leftarrow \mathsf{Puncture}(td, y^*); \\ x \leftarrow \mathcal{A}(ek, td_{y^*}, y^*) \end{array} \right]$$

A PTDF is one-way if no PPT adversary has non-negligible advantage in the above experiment.

Remark A.1. Kiltz et al. [KMO10] introduced the notion of adaptive TDFs, which stipulates that TDFs remain one-way even when the adversary is given access to an inversion oracle. It is easy to see that PTDFs imply ATDFs. In fact, PTDF are best viewed as a special kind of ATDF whose inversion oracle can be instantiated succinctly.

A.1 Construction from Correlate-Product TDFs

Rosen and Segev [RS09] introduced the notion of correlated-product TDFs (CP-TDFs). Briefly, CP-TDFs remain one-way even when the adversary sees many independent instances of the TDF evaluated on correlated inputs. Inspired by the Dolev-Dwork-Naor construction [DDN00] of CCA-secure PKE from CPA-secure PKE, Kiltz et al. [KMO10] showed a black-box construction of ATDFs from CP-TDFs. Our insight is that their construction enjoys puncturable property and thus naturally yields PTDFs.

Let $G : EK \times X \to Y$ be a CP-TDF. Assume that each element in X can be uniquely encoded as a binary string in $\{0,1\}^n$, we build a PTDF $\hat{G} : EK^{2n+1} \times X \to Y^{n+1}$ as below.

- $\mathsf{Gen}(\lambda)$: run $G.\mathsf{Gen}(\lambda)$ independently $2n+1$ times to generate (ek_0, td_0) and $(ek_{i,b}, td_{i,b})$ for $b \in \{0,1\}$ and $i \in [n]$, output the evaluation key $ek = (ek_0, (ek_{1,0}, ek_{1,1}), \ldots, (ek_{n,0}, ek_{n,1}))$ and the corresponding master trapdoor $td = (td_0, (td_{1,0}, td_{1,1}), \ldots, (td_{n,0}, td_{n,1}))$.

[12] If y is not in the image of G_{ek}, $G_{ek}^{-1}(y)$ returns $\perp$.

- $\mathsf{Eval}(ek, x)$: on input $ek = (ek_0, (ek_{1,0}, ek_{1,1}), \ldots, (ek_{n,0}, ek_{n,1}))$ and an element $x \in X$, output $y \leftarrow (G(ek_0, x), G(ek_{1,b_1}, x), \ldots, G(ek_{n,b_n}, x))$ where b_i denotes the ith bit of $G(ek_0, x)$. This algorithm defines $\hat{G}_{ek} : X \to Y^{n+1}$.
- $\mathsf{Inv}(td, y)$: on input $td = (td_0, (td_{1,0}, td_{1,1}, \ldots, (td_{n,0}, td_{n,1}))$ and $y = (y_0, y_1, \ldots, y_n)$, compute $x \leftarrow G^{-1}_{ek_0}(td_0, y_0)$, output x if $y_i = G(ek_{i,b_i}, x)$ for all $i \in [n]$ (here b_i denotes the ith bit of y_0), and return $\perp$ otherwise.
- $\mathsf{Puncture}(td, y^*)$: on input $td = (td_0, (td_{1,0}, td_{1,1}), \ldots, (td_{n,0}, td_{n,1}))$ and an image $y^* = (y_0^*, y_1^*, \ldots, y_n^*) \in \mathrm{Img}(G_{ek})$, output punctured trapdoor $td_{y^*} = (y^*, td_{1,1-b_1^*}, \ldots, td_{n,1-b_n^*})$, where b_i^* denotes the ith bit of y_0^*.
- $\mathsf{PuncInv}(td_{y^*}, y)$: on input td_{y^*} and $y = (y_0, y_1, \ldots, y_n) \neq y^*$, choose an index j such that $b_j \neq b_j^*$ (b_j and b_j^* denotes the jth bit of y_0 and y_0^* respectively, such j must exist as we argue below), output $x \leftarrow G^{-1}_{ek_{j,b_j}}(td_{j,b_j}, y_j)$ if $y_0 = G(ek_0, x)$ and $y_i = G(ek_{i,b_i}, x)$ for all $i \in [n] \backslash j$.

Remark A.2. For a well-formed image $y = (y_0, y_1, \ldots, y_n) \in \mathrm{Img}(G_{ek})$, the injectivity of $G(ek_0, \cdot)$ and $G(ek_{i,b}, \cdot)$ for all $b \in \{0, 1\}$ and $i \in [n]$ guarantees the value of y_0 uniquely determines the rest parts, which means if $y \neq y^*$ we must have $y_0 \neq y_0^*$. This guarantees the existence of such $j \in [n]$.

Theorem A.1. *If G is a $(n+1)$-CP-TDF, then the above construction $\hat{G}$ is a PTDF.*

Proof. Let $\mathcal{A}$ be an adversary against PTDFs with advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$, we build an adversary $\mathcal{B}$ against CP-TDFs with the same advantage.

Given $ek_0, ek_1, \ldots, ek_n$ and $y^* = (G(ek_0, x^*), G(ek_1, x^*), \ldots, G(ek_n, x^*))$ for some unknown $x^* \xleftarrow{R} X$, $\mathcal{B}$ sets $ek_{i,b_i^*} = ek_i$ for all $i \in [n]$ (here b_i^* is the ith bit of $y_0^* = G(ek_0, x^*)$), generates $(ek_{i,1-b_i^*}, td_{i,1-b_i^*}) \leftarrow G.\mathsf{Gen}(\lambda)$ for all $i \in [n]$, sets $ek = (ek_0, (ek_{1,0}, ek_{1,1}), \ldots, (ek_{n,0}, ek_{n,1}))$, the punctured trapdoor $td_{y^*} = (y^*, td_{1,1-b_1^*}, \ldots, td_{n,1-b_n^*})$. $\mathcal{B}$ sends (ek, td_{y^*}, y^*) to $\mathcal{A}$ as the challenge. When $\mathcal{A}$ outputs its solution, $\mathcal{B}$ forwards it as the solution to its own challenger. Clearly, $\mathcal{B}$'s simulation is perfect. If $\mathcal{A}$ succeeds, so does $\mathcal{B}$. This proves the theorem. □

B Puncturable Extractable Hash Proof System

Wee [Wee10] introduced the notion of extractable hash proof system (EHPS), which unifies most existing constructions of CCA-secure PKE based on search problems. In this section, we first consider an extension of EHPS which we call derivable EHPS (DEHPS), then introduce the notion called puncturable EHPS (PEHPS). In what follows, we first recall the notion of binary relation.

Binary Relations. Let $\mathsf{R}_{pp} : X \times W$ be a collection of binary relations indexed by public parameter pp, where X is an $\mathcal{NP}$ language and W is the corresponding set of witnesses. To be applicable, R_{pp} is efficiently verifiable (possibly given some trapdoor for pp) and samplable. We denote the sampling algorithm by $\mathsf{SampRel}$, which on input pp and random coins r outputs a random tuple $(x, w) \in \mathsf{R}_{pp}$. For notation convenience of this work, we further decompose $\mathsf{SampRel}$ to $\mathsf{SampLan}$ and $\mathsf{SampWit}$, where $\mathsf{SampLan}$ (resp. $\mathsf{SampWit}$) outputs the first (resp. second) output of $\mathsf{SampRel}$.

R_{pp} is hard if given a random $x \in X$, it is computationally impossible to find w such $(x, w) \in \mathsf{R}_{pp}$. More formally, we say that R_{pp} is *one-way* if:

- there is an efficiently computable function hc such that $\mathsf{hc}(w)$ is pseudorandom against a PPT adversary that gets pp and x, where pp is randomly chosen and $(x, w) \leftarrow \mathsf{SampRel}(pp)$. In other words, we say hc extracts hardcore bits from w.

For any one-way relation, Goldreich-Levin hardcore predicate $\mathsf{GL}(\cdot)$ [GL89] constitutes such hardcore function with one-bit output. For concrete relations, we may obtain hc with linear number of hardcore bits by either iterating a one-way permutation or relying on decisional assumptions.

Now, we are ready to define derivable EHPS.

Definition B.1 (Derivable EHPS)**.** A DEHPS for $\mathsf{R}_{pp} : X \times W$ is central around a hash function $\mathsf{H} : PK \times X \to \Pi$ with the following polynomial time algorithms:

- $\mathsf{Setup}(\lambda)$: on input λ, output public parameter pp and secret parameter sp. We require that pp uniquely determines sp.
- $\mathsf{Pub}(pk, r)$: on input a public key pk and random coins r, output $\pi \leftarrow \mathsf{H}_{pk}(x)$ where $(x, w) \leftarrow \mathsf{SampRel}(pp; r)$.
- $\mathsf{GenExt}(pp)$: on input pp, output a public key pk and a secret key sk. We require that pp and pk uniquely determines sk.
- $\mathsf{Ext}(sk, x, \pi)$: on input sk, x and π, output $w \in W$ such that $(x, w) \in \mathsf{R}_{pp}$ if $\pi = \mathsf{H}_{pk}(x)$ and $\perp$ otherwise.
- $\mathsf{Derive}(sp, sk)$: on input sp and sk, output sk'. We refer to sk' as the secret hashing key of sk. This algorithm is deterministic, which ensures that sk' is well-defined w.r.t. sp and sk.
- $\mathsf{Priv}(sk', x)$: on input sk' and x, output $\pi \leftarrow \mathsf{H}_{pk}(x)$.
- $\mathsf{GenHash}(pp)$: on input pp, output pk and the secret hashing key sk' determined by Derive.

In DEHPS, Setup generates global parameters, while Pub admits public evaluation of $\mathsf{H}_{pk}(x)$ with the randomness used to sample x. GenExt generates a public key pk and a secret key sk, while Ext can extract witness w from x together with its hash proof π using sk. In addition, Derive allows one to derive a secret hashing key sk' from sk (possibly with the help of sp), while Priv admits private evaluation of $\mathsf{H}_{pk}(x)$ with sk'. $\mathsf{GenHash}$ generates a public key pk together with its secret hashing key sk'. For the relation between the two modes of key generation, we require the following property:

Indistinguishability. For all $(pp, sp) \leftarrow \mathsf{Setup}(\lambda)$, the first outputs (namely pk) of $\mathsf{GenExt}(pp)$ and $\mathsf{GenHash}(pp)$ are statistically indistinguishable.

Relation to EHPS. In EHPS, one can only sample a secret hashing key sk' along with a public key via $\mathsf{GenHash}$. In DEHPS, we require pp (resp. pk) uniquely determines sp (resp. sk), and define the secret hashing key w.r.t. sp and sk via algorithm Derive. This requirement is mild since it is met by all the known instantiations of EHPS, and the reason of such settlement will be clear soon.

We then extend the notion of DEHPS to PEHPS, which is analogous to the extension from EHPS to ABO-EHPS in [Wee10].

Definition B.2 (PEHPS)**.** A PEHPS for R_{pp} is central around a hash function $\mathsf{H} : PK \times X \to \Pi$ with the following polynomial time algorithms.

- $\mathsf{Setup}(\lambda)$: on input λ, output public parameter pp and secret parameter sp. Same as DEHPS, pp uniquely determines sp.

- $\mathsf{Pub}(pk, r)$: on input a public key pk and random coins r, output $\pi \leftarrow \mathsf{H}_{pk}(x)$ where $(x, w) \leftarrow \mathsf{SampRel}(pp; r)$.
- $\mathsf{GenExt}(pp)$: on input pp, output a public key pk and a secret key sk. Same as DEHPS, pk uniquely determines sk.
- $\mathsf{Ext}(sk, x, \pi)$: on input sk, x and π, output $w \in W$ such that $(x, w) \in \mathsf{R}_{pp}$ if $x = \mathsf{H}_{pk}(x)$ and $\perp$ otherwise.
- $\mathsf{Puncture}(sp, sk, x^*)$: on input sp, sk and $x^* \in X$, output sk_{x^*}. We refer to sk_{x^*} as the punctured secret key w.r.t. x^*. This algorithm is deterministic, which ensures that sk_{x^*} is well-defined w.r.t. sp, sk and x^*.
- $\mathsf{PuncPriv}(sk_{x^*}, x^*)$: on input sk_{x^*} and x^*, output $\pi \leftarrow \mathsf{H}_{pk}(x^*)$.
- $\mathsf{PuncExt}(sk_{x^*}, x, \pi)$: on input sk_{x^*}, $x \neq x^*$ and π, output $w \in W$ such that $(x, w) \in \mathsf{R}_{pp}$ if $\pi = \mathsf{H}_{pk}(x)$ and $\perp$ otherwise.
- $\mathsf{GenPunc}(pp, x^*)$: on input pp and x^*, output pk and the punctured secret key sk_{x^*} determined by $\mathsf{Puncture}$.

In a PEHPS, the algorithms Setup, GenExt, Pub and Ext are the same as that of DEHPS. Additionally, the algorithm $\mathsf{Puncture}$ allows one to derive a punctured secret key sk_{x^*} from sk (possibly with sp). Such punctured secret key simultaneously allows for *one-out-all* hash evaluation and *all-but-one* witness extraction. The algorithm $\mathsf{GenPunc}$ admits efficient generation of a punctured secret key sk_{x^*} with a public key pk. For the relation between the two modes of key generation, we require the following property.

Indistinguishability. For all $(pp, sp) \leftarrow \mathsf{Setup}(\lambda)$ and all $x^* \in X$, the first outputs (namely pk) of $\mathsf{GenExt}(pp)$ and $\mathsf{GenPunc}(pp, x^*)$ are statistically indistinguishable.

Relation to ABO-EHPS. Our notion of PEHPS is reminiscent to the notion of ABO-EHPS [Wee10], but has three important differences: (1) Towards efficient and flexible instantiations, ABO-EHPS is tag-based, in which H_{pk} also takes a tag as an auxiliary input. In contrast, PEHPS is tag-free, which is conceptually simple and clean. (2) In PEHPS, pp (resp. pk) uniquely determines sp (resp, sk), and the punctured secret key is defined w.r.t. sp and sk by algorithm $\mathsf{Puncture}$. This settlement guarantees that the punctured secret key is also well-defined w.r.t. pp and pk. (3) The algorithm $\mathsf{GenPunc}$ works in line of the punctured secret key determined by $\mathsf{Puncture}$.[13] As shown in Section C.2, this is crucial for constructing PPEPRFs from PEHPS via $i\mathcal{O}$, since we have to ensure the hybrid game simulated by operating PEHPS in the extraction mode and the hybrid game simulated by operating PEHPS in the punctured mode are still statistically close even in the presence of a punctured secret key.

Remark B.1. In a concurrent work of [DGL+16], Jing et al. [JLL16] showed how to construct leakage-resilient CCA-secure PKE from $i\mathcal{O}$ and ABO-EHPS with secret key deduce algorithm. Their construction fails to be CCA-secure since the ciphertext is easily malleable due to the appearance of seed. Besides, their security proof breaks down when arguing adversary's advantage in hybrid 1 and hybrid 2 are negligibly close based on the statistical indistinguishability of public keys between extraction and ABO modes. This is because $\mathcal{A}$'s view also includes sk_{x^*}, which may differ in two modes.

[13] Similar treatments are informally sketched in [MH15] when discussing how to capture puncturable KEM via ABO-EHPS.

B.1 Construction from DEHPS

Starting from a DEHPS for binary relation $\mathsf{R}_{pp} : X \times W$[14] central around $\mathsf{H} : PK \times X \to \Pi$, we can construct a PEHPS for the same relation central around $\hat{\mathsf{H}} : PK^{2n} \times X \to \Pi^n$ via a DDN-like approach [DDN00, Wee10].

- $\mathsf{Setup}(\lambda)$: same as DEHPS.$\mathsf{Setup}(\lambda)$.
- $\mathsf{GenExt}(pp)$: run DEHPS.$\mathsf{GenExt}(pp)$ to obtain $(pk_{i,b}, sk_{i,b})$ for $1 \le i \le n$ and $0 \le b \le 1$, output $pk = (pk_{i,0}, pk_{i,1})_{i\in[n]}$ and $sk = (sk_{i,0}, sk_{i,1})_{i\in[n]}$.
- $\mathsf{Pub}(pk, r)$: on input $pk = (pk_{i,0}, pk_{i,1})_{i\in[n]} \in PK^{2n}$ and random coins r, run $(x, w) \leftarrow \mathsf{SampRel}(r)$, compute $\pi_i \leftarrow$ DEHPS.$\mathsf{Pub}(pk_{i,x_i}, r)$ for $1 \le i \le n$, output $\pi = (\pi_1, \ldots, \pi_n)$.
- $\mathsf{Ext}(sk, x, \pi)$: on input $sk = (sk_{i,0}, sk_{i,1})_{i\in[n]} \in SK^{2n}$, x and $\pi = (\pi_1, \ldots, \pi_n) \in \Pi^n$, compute $w_i \leftarrow$ DEHPS.$\mathsf{Ext}(sk_{i,x_i}, x, \pi_i)$, output the common value if all n values agree, and $\perp$ otherwise.
- $\mathsf{Puncture}(sp, sk, x^*)$: on input sp, $sk = (sk_{i,0}, sk_{i,1})_{i\in[n]}$ and $x^* \in X$, compute secret hashing keys $sk'_{i,x_i^*} \leftarrow$ DEHPS.$\mathsf{Derive}(sp, sk_{i,x_i^*})$ for each $i \in [n]$, where x_i^* represents the ith bit of x^*, output the punctured secret key $sk_{x^*} = (sk_{i,1-x_i^*}, sk'_{i,x_i^*})_{i\in[n]}$.
- $\mathsf{PuncPriv}(sk_{x^*}, x^*)$: on input $sk_{x^*} = (sk_{i,1-x_i^*}, sk'_{i,x_i^*})_{i\in[\ell]}$ and $x^* \in X$, compute $\pi_i \leftarrow$ DEHPS.$\mathsf{Priv}(sk'_{i,x_i^*}, x^*)$ for $i \in [n]$, output $\pi = (\pi_1, \ldots, \pi_n)$.
- $\mathsf{PuncExt}(sk_{x^*}, x, \pi)$: on input $sk_{x^*} = (sk_{i,1-x_i^*}, sk'_{i,x_i^*})_{i\in[n]}$, $x \ne x^*$ and $\pi = (\pi_1, \ldots, \pi_n) \in \Pi^n$, first check that if $\pi_i =$ DEHPS.$\mathsf{Priv}(sk'_{i,x_i^*}, x^*)$ for all i such that $x_i = x_i^*$. If not, output $\perp$. Else, for all i such that $x_i \ne x_i^*$ compute $w_i \leftarrow$ DEHPS.$\mathsf{Ext}(sk_{i,1-x_i^*}, x^*, \pi_i)$, output the common value if all these values agree and $\perp$ otherwise.
- $\mathsf{GenPunc}(pp, x^*)$: on input pp and x^*, run DEHPS.$\mathsf{GenExt}(pp)$ independently n times to obtain $(pk_{i,1-x_i^*}, sk_{i,1-x_i^*})_{i\in[n]}$, run DEHPS.$\mathsf{GenHash}(pp)$ independently n times to obtain $(pk_{i,x_i^*}, sk'_{i,x_i^*})_{i\in[n]}$; output $pk = (pk_{i,0}, pk_{i,1})_{i\in[n]}$ and $sk' = (sk_{i,1-x_i^*}, sk'_{i,x_i^*})_{i\in[n]}$.

Correctness and indistinguishability of the above construction follows immediately from the starting DEHPS.

In addition to the above generic construction, PEHPS can also be directly built from concrete assumptions. Particularly, several existing realizations of ABO-EHPS already satisfy the definition of PEHPS, such as the one from the bilinear Diffie-Hellman assumption [Wee10, Section 5.1].

C Constructions of PPEPRFs

In this section, we demonstrate the existence of PPEPRFs by realizing them from a variety of puncturable primitives.

C.1 Construction from PTDFs

For the purpose of constructing PPEPRFs, we introduce a new primitive called puncturable trapdoor functions (PTDFs). To avoid departing from the main theme, we defer the definition and construction of PTDFs to Section A.

Let $G : EK \times X \to Y$ be an injective PTDF, $\mathsf{hc}(\cdot)$ be its hardcore function from X to Z.[15] We build a PPEPRF $F : SK \times Y \to Z \cup \perp$ as below.

[14] For simplicity, we assume that each element in X can be uniquely encoded as binary string of length n.

[15] For simplicity, we require hc outputs $\perp$ if and only if its input is $\perp$.

- $\mathsf{Gen}(\lambda)$: run $(ek, td) \leftarrow G.\mathsf{Gen}(\lambda)$, output $pk = ek$ and $sk = td$. For each pk, $L_{pk} = \{y : \exists x \in X \text{ s.t. } y = G_{pk}(x)\}$, where x serves as the witness for $y \in L_{pk}$.
- $\mathsf{PrivEval}(sk, y)$: on input sk and y, compute $x \leftarrow G_{pk}^{-1}(y)$ via $G.\mathsf{Inv}(sk, y)$, output $z \leftarrow \mathsf{hc}(x)$. This algorithm defines $F_{sk}(y) := \mathsf{hc}(G.\mathsf{Inv}(sk, y))$.
- $\mathsf{PubEval}(pk, y, x)$: on input pk, $y \in L_{pk}$ and a witness x, output $z \leftarrow \mathsf{hc}(x)$.
- $\mathsf{Puncture}(sk, y^*)$: on input sk and $y^* \in L_{pk}$, output $sk_{y^*} \leftarrow G.\mathsf{Puncture}(sk, y^*)$.
- $\mathsf{PuncEval}(sk_{y^*}, y)$: on input sk_{y^*} and y, if $y \neq y^*$ compute $x \leftarrow G_{pk}^{-1}(y)$ via $G.\mathsf{PuncInv}(sk_{y^*}, y)$ and output $z \leftarrow \mathsf{hc}(x)$, else output $\perp$.

The correctness of the above construction follows from that of PTDF.

Theorem C.1. *F is a weakly pseudorandom PPEPRF if G is a one-way PTDF.*

Proof. We proceed via a sequence of games. Let S_i be the event that $\mathcal{A}$ succeeds in Game i.

Game 0. This is the standard weak pseudorandomness game for PPEPRFs. $\mathcal{CH}$ interacts with $\mathcal{A}$ as below.

1. Setup: $\mathcal{CH}$ runs $G.\mathsf{Gen}(\lambda)$ to generate pk and sk.
2. Challenge: $\mathcal{CH}$ picks $x^* \xleftarrow{R} X$, computes $y^* \leftarrow G_{pk}(x^*)$, derives $sk_{y^*} \leftarrow G.\mathsf{Puncture}(sk, y^*)$, sets $z_0^* \leftarrow \mathsf{hc}(x^*)$, picks $z_1^* \xleftarrow{R} Z$, $\beta \xleftarrow{R} \{0,1\}$, sends $(pk, sk_{y^*}, y^*, z_\beta^*)$ to $\mathcal{A}$.
3. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

According to the definition of $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[S_0] - 1/2|$$

Game 1. Same as Game 0 except that in the Challenge stage $\mathcal{CH}$ picks $z_0^* \xleftarrow{R} Z$ rather than computing $z_0^* \leftarrow \mathsf{hc}(x^*)$.

Lemma C.2. *If G is a one-way PTDF, then the advantages of any PPT adversary in Game 0 and Game 1 are negligibly close.*

Proof. We prove this lemma by giving a reduction to the one-wayness of the underlying PTDFs. Let $\mathcal{B}$ be an adversary against PTDFs. Given (ek, td_{y^*}, y^*, z^*), $\mathcal{B}$ interacts with $\mathcal{A}$ as below with the aim to determine if $z^* = \mathsf{hc}(x^*)$ or $z^* \xleftarrow{R} Z$. $\mathcal{B}$ sets $pk = ek$, $sk_{y^*} = td_{y^*}$, $z_0^* = z^*$, picks $z_1^* \xleftarrow{R} Z$, $\beta \xleftarrow{R} \{0,1\}$, sends $(pk, sk_{y^*}, y^*, z_\beta^*)$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$. If $\mathcal{A}$ wins, $\mathcal{B}$ outputs 1.

By the definitions of Game 0 and Game 1, if $\mathcal{B}$ receives $z^* = \mathsf{hc}(x^*)$, then the probability that $\mathcal{B}$ outputs 1 is exactly the probability of $\mathcal{A}$ winning in Game 0. If $\mathcal{B}$ receives $z^* \xleftarrow{R} Z$, then the probability that $\mathcal{B}$ outputs 1 is the probability of $\mathcal{A}$ winning in Game 1. The lemma follows. □

Lemma C.3. *For any (even unbounded) adversary, its advantage in Game 1 is 0.*

Proof. In Game 1, both z_0^* and z_1^* are picked uniformly at random from Z. Therefore, β is perfectly hidden. The lemma follows. □

Putting all the above together, the theorem immediately follows. □

C.2 Construction from PEHPS

For the purpose of constructing PPEPRFs, we introduce a new notion called puncturable extractable hash proof systems (PEHPS). To avoid departing from the main theme, we defer the definition and construction of PEHPS to Section B.

Let Γ be a PEHPS for one-way relation $\mathsf{R}_{pp} : X \times W$ central around a hash function $\mathsf{H} : PK \times L \to \Pi$, R be the randomness space used by $\mathsf{SampLan}$ and $\mathsf{SampWit}$, SP and SK be the secret parameter space and secret key space respectively, $\mathsf{hc}(\cdot)$ be its hardcore function from W to Y.[16] We build a PPEPRF $F : \hat{SK} \times \hat{X} \to Y \cup \perp$ from PEHPS, where $\hat{SK} = SP \times SK$ and $\hat{X} = X \times \Pi$.

- $\mathsf{Gen}(\lambda)$: run $(pp, sp) \leftarrow \Gamma.\mathsf{Setup}(\lambda)$, $(pk, sk) \leftarrow \Gamma.\mathsf{GenExt}(pp)$; output $\hat{pk} = (pp, pk)$ and $\hat{sk} = (sp, sk)$. For each $\hat{pk} = (pp, pk)$, the language $L_{\hat{pk}} = \{\hat{x} = (x, \pi) : \exists r \in R \text{ s.t. } x = \mathsf{SampLan}(r) \wedge \pi = \Gamma.\mathsf{Pub}(pk, r)\}$, where r serves as the witness for $\hat{x} \in L_{\hat{pk}}$.
- $\mathsf{PrivEval}(\hat{sk}, \hat{x})$: on input $\hat{sk} = (sp, sk)$ and $\hat{x} = (x, \pi) \in X \times \Pi$, compute $w \leftarrow \Gamma.\mathsf{Ext}(sk, x, \pi)$, output $y \leftarrow \mathsf{hc}(w)$. This algorithm implicitly defines $F_{sk}(\hat{x} = (x, \pi)) = \mathsf{hc}(\Gamma.\mathsf{Ext}(sk, x, \pi))$.
- $\mathsf{PubEval}(\hat{pk}, \hat{x}, r)$: on input $\hat{pk} = (pp, pk)$, $\hat{x} = (x, \pi) \in L_{\hat{pk}}$ and an associate witness r, compute $w \leftarrow \mathsf{SampWit}(r)$, output $y \leftarrow \mathsf{hc}(w)$.
- $\mathsf{Puncture}(\hat{sk}, \hat{x^*})$: on input $\hat{sk} = (sp, sk)$ and $\hat{x^*} = (x^*, \pi^*) \in L_{\hat{pk}}$, compute $sk_{x^*} \leftarrow \Gamma.\mathsf{Puncture}(sp, sk, x^*)$, output $\hat{sk}_{\hat{x^*}} = sk_{x^*}$.
- $\mathsf{PuncEval}(\hat{sk}_{\hat{x^*}}, \hat{x})$: on input $\hat{sk}_{\hat{x^*}} = sk_{x^*}$ and $\hat{x} = (x, \pi) \neq (x^*, \pi^*) = \hat{x^*}$, if $x = x^*$ output $\perp$, else compute $w \leftarrow \Gamma.\mathsf{PuncExt}(sk_{x^*}, x, \pi)$ and output $y \leftarrow \mathsf{hc}(w)$.

The correctness of the above construction follows from that of PEHPS. In particular, the correctness of $\mathsf{PuncEval}$ is ensured by the functionality of $\Gamma.\mathsf{PuncExt}$ and the fact that each element x has a unique hash proof.

Theorem C.4. *F is a weakly pseudorandom PPEPRF if R_{pp} is one-way.*

Proof. We proceed via a sequence of games. Let S_i be the event that $\mathcal{A}$ succeeds in Game i.

Game 0. This is the standard weak pseudorandomness game for PPEPRFs. $\mathcal{CH}$ interacts with $\mathcal{A}$ by generating the key pair in the extraction mode.

1. Setup: $\mathcal{CH}$ runs $(pp, sp) \leftarrow \Gamma.\mathsf{Setup}(\lambda)$, $(pk, sk) \leftarrow \Gamma.\mathsf{GenExt}(pp)$, sets $\hat{pk} = (pp, pk)$ and $\hat{sk} = (sp, sk)$.
2. Challenge: $\mathcal{CH}$ samples $(x^*, w^*) \leftarrow \mathsf{SampRel}(pp; r^*)$, then computes $\pi^* \leftarrow \mathsf{H}_{pk}(x^*)$ via $\Gamma.\mathsf{Pub}(pk, x^*, r^*)$, sets $\hat{x^*} = (x^*, \pi^*)$ and generates the punctured secret key $\hat{sk}_{\hat{x^*}} = sk_{x^*} \leftarrow \Gamma.\mathsf{Puncture}(sp, sk, x^*)$, computes $y_0^* \leftarrow \mathsf{hc}(w^*)$, picks $y_1^* \xleftarrow{R} Y$, $\beta \xleftarrow{R} \{0, 1\}$. Finally, $\mathcal{CH}$ sends $(\hat{pk}, \hat{sk}_{\hat{x^*}}, \hat{x^*}, y_\beta^*)$ to $\mathcal{A}$.
3. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

According to the definition of $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[S_0] - 1/2|$$

Game 1. Same as Game 0 except that $\mathcal{CH}$ generates $(x^*, w^*) \leftarrow \mathsf{SampRel}(pp; r^*)$ in the Setup stage. This is merely a conceptual change and thereby:

$$\Pr[S_0] = \Pr[S_1]$$

[16] We require $\mathsf{hc}(\cdot)$ outputs $\perp$ when its input is $\perp$.

Game 2. Same as Game 1 except that $\mathcal{CH}$ interacts with $\mathcal{A}$ by generating the key pair in the punctured mode.

1. Setup: $\mathcal{CH}$ runs $(pp, sp) \leftarrow \Gamma.\mathsf{Setup}(\lambda)$, samples $(x^*, w^*) \leftarrow \mathsf{SampRel}(pp; r^*)$, then runs $(pk, sk_{x^*}) \leftarrow \Gamma.\mathsf{GenPunc}(pp, x^*)$, sets $\hat{pk} = (pp, pk)$.
2. Challenge: $\mathcal{CH}$ computes $\pi^* \leftarrow \mathsf{H}_{pk}(x^*)$ via $\Gamma.\mathsf{PuncPriv}(sk_{x^*}, x^*)$, then sets $\hat{x^*} = (x^*, \pi^*)$ and $\hat{sk}_{\hat{x^*}} = sk_{x^*}$, computes $y_0^* \leftarrow \mathsf{hc}(w^*)$, picks $y_1^* \xleftarrow{R} Y$, $\beta \xleftarrow{R} \{0,1\}$. Finally, $\mathcal{CH}$ sends $(\hat{pk}, \hat{sk}_{\hat{x^*}}, \hat{x^*}, y_\beta^*)$ to $\mathcal{A}$.
3. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

Indistinguishability between the extraction mode and the punctured mode and the correctness of $\mathsf{GenPunc}$ and $\mathsf{PuncPriv}$ imply that the view $(\hat{pk}, \hat{sk}_{\hat{x^*}}, \hat{x^*}, y_\beta^*)$ in Game 1 and 2 are statistically indistinguishable. Thereby, we have:

$$|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{negl}(\lambda)$$

We then show that no PPT adversary has non-negligible advantage in Game 2. Let $\mathcal{A}$ be a PPT adversary that has advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ in Game 2, we build a reduction algorithm $\mathcal{B}$ that breaks the one-wayness of the underlying binary relation R_{pp} with same advantage. $\mathcal{B}$ interacts with $\mathcal{A}$ in Game 2 as below.

1. Setup and Challenge: $\mathcal{B}$ proceeds the same way as $\mathcal{CH}$ does in Game 2 except that it receives (x^*, y_β^*) from its own challenge.
2. Guess: $\mathcal{A}$ outputs its guess β' for β and $\mathcal{B}$ forwards β' to its own challenger.

Obviously, $\mathcal{B}$'s simulation is perfect. Therefore, we have:

$$\mathsf{Adv}_{\mathcal{B}}(\lambda) = \mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[S_2] - 1/2|$$

which is negligible in λ assuming the one-wayness of R_{pp}.

Putting all the above together, the theorem immediately follows. □

C.3 Construction from wPPRFs and $i\mathcal{O}$

Sahai and Waters [SW14] showed an ingenious CCA-secure KEM construction from wPPRFs and pseudorandom generator (PRG). We observe that their construction essentially provides a method of compiling wPPRFs to PEPRFs, and the resulting PEPRFs inherit the puncturable property from the underlying wPPRFs.

Next, we show how to adapt Sahai-Waters KEM to a generic construction of PPEPRFs from wPPRFs. Let $i\mathcal{O}$ be a secure indistinguishability obfuscation, $\mathsf{G} : W \to X$ be a PRG and $F : SK \times X \to Y$ be a wPPRF. Here, we assume $W = \{0,1\}^\lambda$ and $X = \{0,1\}^{2\lambda}$ for simplicity. We build a PPEPRF $\hat{F} : SK \times X \to Y$ for language L. In our construction, L is independent of $\hat{pk}$, i.e., $L = \{x : \exists w \in W \text{ s.t. } x = \mathsf{G}(w)\}$.

- $\mathsf{Gen}(\lambda)$: run $(pk, sk) \leftarrow F.\mathsf{Gen}(\lambda)$, create $\hat{pk} \leftarrow i\mathcal{O}(\text{PubEval})$, where the program PubEval is defined in Figure 16; output $(\hat{pk}, sk)$.
- $\mathsf{PrivEval}(sk, x)$: on input sk and $x \in X$, output $y \leftarrow F_{sk}(x)$. This algorithm defines $\hat{F}_{sk}(x) = F_{sk}(x)$. In other words, $\hat{F}$ is exactly the same as F.
- $\mathsf{PubEval}(\hat{pk}, x, w)$: on input $\hat{pk}$, $x \in L$ and an associate witness w such that $\mathsf{G}(w) = x$, output $y \leftarrow \hat{pk}(x, w)$.
- $\mathsf{Puncture}(sk, x^*)$: output $sk_{x^*} \leftarrow F.\mathsf{Puncture}(sk, x^*)$.

PubEval

Constants: wPPRF key sk

Input: $x \in X$, $w \in W$

1. If $x \neq \mathsf{G}(w)$, output $\perp$.
2. Else, output $F_{sk}(x)$.

Figure 16: Program PubEval. This program is appropriately padded to the maximum of the size of itself and the program PubEval* defined in Figure 17.

PubEval*

Constants: wPPRF punctured key sk_{x^*}, x^*

Input: $x \in X$, $w \in W$

1. If $x \neq \mathsf{G}(w)$, output $\perp$.
2. Else, output $F_{sk_{x^*}}(x)$ if $x \neq x^*$ and $\perp$ otherwise.

Figure 17: Program $\mathsf{PubEval}^*$

- $\mathsf{PuncPriv}(sk_{x^*}, x)$: output $y \leftarrow F.\mathsf{PuncPriv}(sk_{x^*}, x)$.

It is straightforward to verify the correctness of the above construction. For the security, we have the following theorem.

Theorem C.5. *If $i\mathcal{O}$ is indistinguishably secure, G is a secure PRG, and F is a secure wPPRF, then the above construction is a weakly pseudorandom PPEPRF.*

Proof. We proceed via a sequence of games. Let S_i be the event that $\mathcal{A}$ wins in Game i.

Game 0. This is the standard weak pseudorandom game for PPEPRFs. $\mathcal{CH}$ interacts with $\mathcal{A}$ as follows:

1. Setup: $\mathcal{CH}$ runs $(pk, sk) \leftarrow F.\mathsf{Gen}(\lambda)$ to obtain the secret key and creates $\hat{pk} \leftarrow i\mathcal{O}(\text{PrivEval})$ as the public key.
2. Challenge: $\mathcal{CH}$ first picks $w^* \xleftarrow{R} W$, then computes $x^* \leftarrow \mathsf{G}(w^*)$, $sk_{x^*} \leftarrow F.\mathsf{Puncture}(sk, x^*)$, $y_0^* \leftarrow F_{sk}(x^*)$, picks $y_1^* \xleftarrow{R} Y$, $\beta \xleftarrow{R} \{0,1\}$. $\mathcal{CH}$ sends $(\hat{pk}, sk_{x^*}, x^*, y_\beta^*)$ to $\mathcal{A}$ as the challenge.
3. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

According to the definition of Game 0, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[S_0] - 1/2|$$

Game 1. Same as Game 0 except that $\mathcal{CH}$ picks $w^* \xleftarrow{R} W$ and computes $x^* \leftarrow \mathsf{G}(w^*)$ in the Setup stage. This change is purely conceptual. Thus, $\mathcal{A}$'s view in Game 0 and Game 1 are identical. We have:

$$\Pr[S_1] = \Pr[S_0]$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ picks $x^* \xleftarrow{R} X$ rather than setting $x^* \leftarrow \mathsf{G}(w^*)$ (where $w^* \xleftarrow{R} W$) in the Setup stage. Assuming the security of PRG, we have:

$$|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathrm{PRG}}$$

Game 3. Same as Game 2 except that $\mathcal{CH}$ computes $sk_{x^*} \leftarrow F.\mathsf{Puncture}(sk, x^*)$, then creates $\hat{pk} \leftarrow i\mathcal{O}(\mathrm{PubEval}^*)$ in the Setup stage. Here, the program PubEval* (defined in Figure 17) is built from constants sk_{x^*} and x^*. We first observe that $\Pr[x^* \notin \mathrm{Img}(\mathsf{G})] = 1 - 1/2^\lambda$ for a randomly chosen x^*. Thus, with all but negligible probability neither PubEval nor PubEval* will evaluate $F(k, x^*)$ and hence the input/output behavior of programs are identical. By a direct reduction to the security of $i\mathcal{O}$, we have:

$$|\Pr[S_3] - \Pr[S_2]| \leq \mathsf{Adv}^{i\mathcal{O}}_{\mathcal{A}}$$

Assuming the security of wPPRF, we have:

$$|\Pr[S_3] - 1/2| \leq \mathsf{Adv}^{\mathrm{wPPRF}}_{\mathcal{A}}$$

Putting all the above together, the theorem immediately follows. □

D Leakage-Resilient Signature with Adaptive Security

Our selectively secure construction of Section 5 can be boosted with adaptive security via two generic methods, without compromising leakage resilience. One method is recently suggested by Zhandry [Zha16], which simply hashes the message with an extremely lossy function (ELF) before signing. The resulting signature scheme is still deterministic. However, the only known construction of ELF relies on the exponential hardness of the decisional Diffie-Hellman problem. Another method is to apply the "prefix-guessing technique" of Hohenberger-Waters [HW09]. The resulting signature scheme becomes randomized but public-coin.

In this section, we show how to construct a leakage-resilient signature scheme with adaptive security, by combining our selectively secure construction of Section 5 with the optimized "prefix-guessing technique" developed in [RW14].

Let $F_1 : K_1 \times \{0,1\}^{\log l + 1 + \lambda} \to \{0,1\}^n$ and $F_2 : K_2 \times \{0,1\}^{1 \leq i \leq l} \to \{0,1\}^n$ be two sPPRFs, $g : \{0,1\}^n \to \{0,1\}^\lambda$ be an ℓ-entropy-leakage-resilient injective OWF, SKE be an IND-CPA secure SKE with message space $\{0,1\}^n$ and ciphertext space $\{0,1\}^v$, LF be a family of (v,τ)-lossy functions. Let e_j be the j-bit string $0 \cdots 01$, $m[j]$ be the jth bit of m, and $t(j)$ be the first j bits of t. We build a leakage-resilient signature scheme with message space $M = \{0,1\}^\lambda$ as follows.

- $\mathsf{Gen}(\lambda)$: run $(pp_1, k_1) \leftarrow F_1.\mathsf{Gen}(\lambda)$, $(pp_2, k_2) \leftarrow F_2.\mathsf{Gen}(\lambda)$, $h \leftarrow \mathrm{LF}.\mathsf{GenInj}(\lambda)$, and $k_e \leftarrow \mathrm{SKE}.\mathsf{Gen}(\lambda)$, generate a dummy ciphertext $ct \leftarrow \mathrm{SKE}.\mathsf{Enc}(k_e, 0^n)$, compute $\eta^* \leftarrow h(ct)$, create $C_{\mathrm{sign}} \leftarrow i\mathcal{O}(\mathrm{Sign})$ and $C_{\mathrm{vefy}} \leftarrow i\mathcal{O}(\mathrm{Verify})$. The programs Sign and Verify are defined in Figure 18 and Figure 19 respectively. Finally, output $vk = (C_{\mathrm{vefy}}, C_{\mathrm{sign}})$ and $sk = ct$.
- $\mathsf{Sign}(sk, m)$: choose $t \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, output $\sigma \leftarrow C_{\mathrm{sign}}(sk, m, t)$.[17]
- $\mathsf{Verify}(vk, m, \sigma)$: output $C_{\mathrm{vefy}}(m, \sigma)$.

Theorem D.1. *If F_1 and F_2 are two secure sPPRFs, $i\mathcal{O}$ is indistinguishably secure, SKE is IND-CPA secure, LF is a family of (v,τ)-lossy functions, and g is ℓ-entropy-leakage-resilient one-way, then the above construction is ℓ'-leakage-resilient EUF-CMA as long as $\ell' \leq \ell - \tau$.*

Proof. By appropriate parameter choice (e.g. setting $v = n + o(n)$, $\ell = n - o(n)$, $\tau = o(v)$), we have $|sk| = v = n + o(n)$ and $\ell' = n - o(n)$ and thus the leakage rate is optimal. We then prove the security as follows.

[17] Here, we let $t \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ only for simplicity. We can make t shorter as discussed at the end of this section.

Sign

Constants: sPPRF keys k_1 and k_2, and η^*

Input: ct, message m, and randomness t

1. If $h(ct) \neq \eta^*$, output $\perp$.
2. Else, compute $s \leftarrow \oplus_{j=1}^{\lambda} F_1(k_1, j||m[j]||t) \oplus_{j=1}^{\lambda} F_2(k_2, t(j))$.
3. Output $\sigma = (s, t)$.

Figure 18: Program Sign. This program is appropriately padded to the maximum of itself and the programs Sign* and Sign** defined in Figure 20 and Figure 22.

Verify

Constants: sPPRF keys k_1 and k_2

Input: message m and signature $\sigma = (s, t)$

1. Test if $g(s) = g(\oplus_{j=1}^{\lambda} F_1(k_1, j||m[j]||t) \oplus_{j=1}^{\lambda} F_2(k_2, t(j)))$, output 1 if true and 0 if false.

Figure 19: Program Verify. This program is appropriately padded to the maximum of itself and the programs Verify* and Verify** defined in Figure 21 and Figure 23.

Let $\mathcal{A}$ be any PPT adversary, q be the maximum number of signing queries made by $\mathcal{A}$, m_i be the ith signing query, $\sigma_i = (s_i, t_i)$ be the answer to the ith signing query, and $(m^*, \sigma^* = (s^*, t^*))$ be the forgery generated by $\mathcal{A}$. To succeed in the ℓ-leakage-resilient EUF-CMA security experiment, $\mathcal{A}$ has to output a forgery which is one of the following two types.

- Type I: $m^* \notin \{m_1, m_2, \cdots, m_q\} \wedge t^* \notin \{t_1, t_2, \cdots, t_q\} \wedge C_{\text{vefy}}(m^*, \sigma^*) = 1$.
- Type II: $m^* \notin \{m_1, m_2, \cdots, m_q\} \wedge t^* \in \{t_1, t_2, \cdots, t_q\} \wedge C_{\text{vefy}}(m^*, \sigma^*) = 1$.

At first, we prove that the probability that $\mathcal{A}$ outputs a Type I forgery is negligible. We proceed via a sequence of games. Let S_i be the probability that $\mathcal{A}$ wins in Game i by outputting a Type I forgery.

Game 0. This game is same as the standard leakage-resilient EUF-CMA game for signature except that the winning condition is to output a Type I forgery. $\mathcal{CH}$ interacts with $\mathcal{A}$ as follows:

1. Setup: $\mathcal{CH}$ runs $(pp_1, k_1) \leftarrow F_1.\mathsf{Gen}(\lambda)$, $(pp_2, k_2) \leftarrow F_2.\mathsf{Gen}(\lambda)$, $h \leftarrow \text{LF}.\mathsf{GenInj}(\lambda)$, and $k_e \leftarrow \text{SKE}.\mathsf{Gen}(\lambda)$, generates a dummy ciphertext $ct \leftarrow \text{SKE}.\mathsf{Enc}(k_e, 0^n)$, computes $\eta^* \leftarrow h(ct)$, creates $C_{\text{sign}} \leftarrow i\mathcal{O}(\text{sign})$ and $C_{\text{Vefy}} \leftarrow i\mathcal{O}(\text{Verify})$, and sets $vk = (C_{\text{vefy}}, C_{\text{sign}})$ and $sk = ct$. Then $\mathcal{CH}$ sends vk to $\mathcal{A}$.
2. Signing Query: Upon receiving the ith signing query $\langle m_i \rangle$, $\mathcal{CH}$ samples $t_i \xleftarrow{R} \{0,1\}^\lambda$ and responds with $\sigma_i \leftarrow C_{\text{sign}}(sk, m_i, t_i)$.
3. Leakage Query: Upon receiving leakage query $\langle f \rangle$, $\mathcal{CH}$ responds with $f(sk)$.
4. Forge: $\mathcal{A}$ wins if it outputs a Type I forgery $(m^*, \sigma^* = (s^*, t^*))$.

According to the definition of $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_0]$$

Game 1. Same as in Game 0 except that $\mathcal{CH}$ also picks $t_1, \ldots, t_q \xleftarrow{R} \{0,1\}^\lambda$ in the Setup stage and uses t_i to answer the ith signing query.

This change is purely conceptual and thus $\mathcal{A}$'s view in Game 0 and Game 1 are identical. Thereby, we have:

$$\Pr[S_1] = \Pr[S_0]$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ also picks $\hat{i} \xleftarrow{R} [q]$ and $\hat{j} \xleftarrow{R} [\lambda]$, and $\mathcal{A}$ is considered to be successful if it outputs a Type I forgery $(m^*, \sigma^* = (s^*, t^*))$ such that $t^*(\hat{j}) = t_{\hat{i}}(\hat{j}) \oplus e_{\hat{j}}$ and $t^*(\hat{j}) \neq t_i(\hat{j})$ for all $i \in [q]$. In other words, $\mathcal{A}$ fails if (m^*, σ^*) is not a Type I forgery or $t_{\hat{i}}(\hat{j})$ is not the longest $t_i(j)$ such that $t_i(j) \oplus e_j = t^*(j)$ for $i \in [q]$ and $j \in [\lambda]$.

When $\mathcal{A}$ outputs a Type I forgery, we have $t^* \notin \{t_1, \ldots, t_q\}$ according to the definition. Thereby, the tuple $(\hat{i}, \hat{j})$ such that $t^*(\hat{j}) = t_{\hat{i}}(\hat{j}) \oplus e_{\hat{j}}$ and $t^*(\hat{j}) \neq t_i(\hat{j})$ for all $i \in [q]$ must exist. Moreover, the view of $\mathcal{A}$ in Game 2 is identical to its view in Game 1 and $\mathcal{A}$ learns no information on which $(\hat{i}, \hat{j})$ is chosen. Therefore, we have:

$$\Pr[S_2] \geq \Pr[S_1]/(q\lambda)$$

Game 3. Same as Game 2 except that $\mathcal{CH}$ uses a punctured key of F_2 to handle signing queries: (i) computes $k_2(\{a\}) \leftarrow F_2.\mathsf{Puncture}(k_2, a)$ where $a = t_{\hat{i}}(\hat{j}) \oplus e_{\hat{j}}$ in the Setup stage; (ii) on the ith signing query $m_i \neq m^*$, computes $s_i \leftarrow \oplus_{j=1}^{\lambda} F_1(k_1, j||m_i[j]||t_i) \oplus_{j=1}^{\lambda} F_2(k_2(\{a\}), t_i(j))$ and returns $\sigma_i = (s_i, t_i)$. Due to the the correctness of sPPRFs and the fact that $\mathcal{A}$ succeeds in Game 2 only if we have $t_{\hat{i}}(\hat{j}) \oplus e_{\hat{j}} \neq t_i(\hat{j})$ for all $i \in [q]$, in which case $F_2(k_2, a)$ is never computed when answering signing queries, Game 2 and Game 3 are identical in $\mathcal{A}$'s view and thus we have:

$$\Pr[S_3] = \Pr[S_2]$$

Game 4. Same as Game 3 except that $\mathcal{CH}$ computes $z^* \leftarrow F_2(k_2, a)$ and generates $ct \leftarrow \mathsf{SKE.Enc}(k_e, z^*)$ rather than $ct \leftarrow \mathsf{SKE.Enc}(k_e, 0^n)$ in the Setup stage. By the security of SKE, we have:

$$|\Pr[S_4] - \Pr[S_3]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathrm{SKE}}$$

Game 5. Same as Game 4 except that $\mathcal{CH}$ creates $C_{\mathrm{sign}} \leftarrow i\mathcal{O}(\mathrm{Sign}^*)$ in the Setup stage. Here, the program Sign^* (defined in Figure 20) is built from constants $\hat{j}$, k_1, $k_2(\{a\})$, η^*, and a.

Sign^*

Constant: $\hat{j}$, k_1, $k_2(\{a\})$, η^*, and a

Input: ct, message m, and randomness t

1. If $h(ct) \neq \eta^*$, output $\perp$.
2. If $t(\hat{j}) \oplus e_{\hat{j}} = a$, compute $z^* \leftarrow \mathsf{SKE.Dec}(k_e, ct)$ and $s \leftarrow \oplus_{j=1}^{\lambda} F_1(k_1, j||m[j]||t) \oplus_{j \neq \hat{j}} F_2(k_2(\{a\}), t(j)) \oplus z^*$, and then output $\sigma = (s, t)$.
3. Else, compute $s \leftarrow \oplus_{j=1}^{\lambda} F_1(k_1, j||m[j]||t) \oplus_{j=1}^{\lambda} F_2(k_2(\{a\}), t(j))$, and output $\sigma = (s, t)$.

Figure 20: Program Sign^*

It is easy to check that the two programs Sign and Sign^* agree on all inputs due to the injectivity of g. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_5] - \Pr[S_4]| \leq \mathsf{Adv}_{\mathcal{A}}^{i\mathcal{O}}$$

Verify*

Constant: $\hat{j}$, k_1, $k_2(\{a\})$, g, y^* and a

Input: message m and signature $\sigma = (s, t)$

1. If $a = t(\hat{j})$, output 1 if $g(s \oplus_{j \neq \hat{j}} F_2(k_2(\{a\}), t(j)) \oplus_{j=1}^{\lambda} F_1(k_1, j||m[j]||t)) = y^*$ and output 0 otherwise.
2. Else, output 1 if $g(s) = g(\oplus_{j=1}^{\lambda} F_1(k_1, j||m[j]||t) \oplus_{j=1}^{\lambda} F_2(k_2(\{a\}), t(j))$ and output 0 otherwise.

Figure 21: Program Verify*

Game 6. Same as Game 5 except that $\mathcal{CH}$ computes $y^* \leftarrow g(z^*)$, creates $C_{\text{vefy}} \leftarrow i\mathcal{O}(\text{Verify}^*)$ in the Setup stage. Here, the program Verify* (defined in Figure 21) is built from constants $\hat{j}$, k_1, $k_2(\{a\})$, g, y^* and a.

It is easy to check that the two programs Verify and Verify* agree on all inputs due to the injectivity of g. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_6] - \Pr[S_5]| \leq \mathsf{Adv}_{\mathcal{A}}^{i\mathcal{O}}$$

Game 7. Same as Game 6 except that $\mathcal{CH}$ picks $z^* \xleftarrow{R} \{0,1\}^n$ rather than setting $z^* \leftarrow F_2(k_2, a)$ in the Setup stage.

By the selective pseudorandomness of sPPRFs, we have:

$$|\Pr[S_7] - \Pr[S_6]| \leq \mathsf{Adv}_{\mathcal{A}}^{\text{sPPRF}}$$

It remains to analyze $\Pr[S_7]$. We have the following claim.

Claim D.2. *If g is injective and ℓ-entropy-leakage-resilient one-way, then the advantage of all PPT adversary in Game 7 is negligible in λ.*

Proof. Let $\mathcal{A}$ be a PPT adversary that wins in Game 7 with advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$. We build an adversary $\mathcal{B}$ that breaks the assumed entropy-leakage-resilient one-wayness of g with the same advantage, implying that $\Pr[S_7]$ must be negligible.

Given g, y^* where $y^* \leftarrow g(z^*)$ for some $z^* \xleftarrow{R} \{0,1\}^n$, $\mathcal{B}$ interacts with $\mathcal{A}$ in Game 7 with the aim to output z^*.

1. Setup: $\mathcal{B}$ picks $t_1, \ldots, t_q \xleftarrow{R} \{0,1\}^\lambda$, $\hat{\imath} \xleftarrow{R} [q]$, $\hat{j} \xleftarrow{R} [\lambda]$, computes $a \leftarrow t_{\hat{\imath}}(\hat{j}) \oplus e_{\hat{j}}$ generates $(pp_1, k_1) \leftarrow F_1.\mathsf{Gen}(\lambda)$, and $(pp_2, k_2) \leftarrow F_2.\mathsf{Gen}(\lambda)$. $\mathcal{B}$ then computes $k_2(\{a\}) \leftarrow F_2.\mathsf{Puncture}(k_2, a)$, and picks $k_e \leftarrow \text{SKE.}\mathsf{Gen}(\lambda)$ and $h \leftarrow \text{LF.GenLossy}(\lambda)$. Next $\mathcal{B}$ picks a fresh randomness r for encryption, defines function $\psi(\cdot) := \text{SKE.}\mathsf{Enc}(k_e, \cdot; r)$, and makes a leakage query $\langle h \circ \psi \rangle$ about z^* to obtain η^*. Then it creates $C_{\text{vefy}} \leftarrow i\mathcal{O}(\text{Verify}^*)$ and $C_{\text{sign}} \leftarrow i\mathcal{O}(\text{Sign}^*)$, and sends $vk = (C_{\text{vefy}}, C_{\text{sign}})$ to $\mathcal{A}$.
2. Signing Query: Upon receiving the ith signing query $\langle m_i \rangle$, $\mathcal{B}$ computes responds as follows: computes $s_i \leftarrow \oplus_{j=1}^{\lambda} F_1(k_1, j||m_i[j]||t_i) \oplus_{j=1}^{\lambda} F_2(k_2(\{a\}), t_i(j))$, then returns $\sigma_i = (s_i, t_i)$ to $\mathcal{A}$.
3. Leakage Query: Note that for any leakage query on $sk = ct$, $\mathcal{B}$ can transform it to leakage query on z^*. Upon receiving leakage query $\langle f \rangle$, $\mathcal{B}$ makes a leakage query $\langle f \circ \psi \rangle$ about z^* to its own challenger and forwards the reply to $\mathcal{A}$.
4. Forge: $\mathcal{A}$ outputs $(m^*, \sigma^* = (s^*, t^*))$ and wins if it is a Type I forgery such that $t^*(\hat{j}) = t_{\hat{\imath}}(\hat{j}) \oplus e_{\hat{j}}$ and $t^*(\hat{j}) \neq t_i(\hat{j})$ for all $i \in [q]$.

Finally, $\mathcal{B}$ forwards $s^* \oplus_{j\neq\hat{j}} F_2(k_2(\{a\}), t^*(j)) \oplus_{j=1}^{\lambda} F_1(k_1, j||m^*[j]||t^*)$ to its own challenger. Conditioned on $\mathcal{A}$ succeeds, we have $t^*(\hat{j}) = t_{\hat{i}}(\hat{j}) \oplus e_{\hat{j}}$. According to the definition of Verify*, this means $g(s^* \oplus_{j\neq\hat{j}} F_2(k_2(\{a\}), t^*(j)) \oplus_{j=1}^{\lambda} F_1(k_1, j||m^*[j]||t^*)) = y^*$. Moreover, η^* reveals at most τ bits information about ct and thus z^*, since now h is (v, τ) lossy. Hence, the leakage query made by $\mathcal{B}$ will be correctly answered by its challenger. As a result, the probability that $\mathcal{B}$ wins is $\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_7]$, which is negligible in λ by the security of g. This proves the claim. □

Since $q \cdot \lambda$ is a polynomial in λ, taken all together, $\Pr[S_7]$ is negligible in λ, i.e., the probability that $\mathcal{A}$ outputs a Type I forgery is negligible in λ. This finishes the first part of the proof.

It remains to show that the probability that $\mathcal{A}$ outputs a Type II forgery is also negligible in λ. We proceed via a sequence of games. Let S_i be the probability that $\mathcal{A}$ wins in Game i.

Game 0. This game is the standard leakage-resilient EUF-CMA security experiment for signature except that the winning condition is to output a Type II forgery. $\mathcal{CH}$ interacts with $\mathcal{A}$ as follows:

1. Setup: $\mathcal{CH}$ runs $(pp_1, k_1) \leftarrow F_1.\mathsf{Gen}(\lambda)$, $(pp_2, k_2) \leftarrow F_2.\mathsf{Gen}(\lambda)$, $h \leftarrow \mathrm{LF}.\mathsf{GenInj}(\lambda)$, and $k_e \leftarrow \mathrm{SKE}.\mathsf{Gen}(\lambda)$, generates a dummy ciphertext $ct \leftarrow \mathrm{SKE}.\mathsf{Enc}(k_e, 0^n)$, computes $\eta \leftarrow h(ct)$, creates $C_{\mathrm{sign}} \leftarrow i\mathcal{O}(\mathrm{Sign})$ and $C_{\mathrm{vefy}} \leftarrow i\mathcal{O}(\mathrm{Verify})$, and sets $vk = (C_{\mathrm{vefy}}, C_{\mathrm{sign}})$ and $sk = ct$. Then $\mathcal{CH}$ sends vk to $\mathcal{A}$.
2. Signing Query: Upon receiving the ith signing query $\langle m_i \rangle$, $\mathcal{CH}$ samples $t_i \xleftarrow{R} \{0,1\}^{\lambda}$ and responds with $\sigma_i \leftarrow C_{\mathrm{sign}}(sk, m_i, t_i)$.
3. Leakage Query: Upon receiving leakage query $\langle f \rangle$, $\mathcal{CH}$ responds with $f(sk)$.
4. Forge: $\mathcal{A}$ wins if it outputs a Type II forgery $(m^*, \sigma^* = (s^*, t^*))$.

According to the definition of $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_0]$$

Game 1. Same as in Game 0 except that $\mathcal{CH}$ also picks $t_1, \ldots, t_q \xleftarrow{R} \{0,1\}^{\lambda}$ in the Setup stage and uses t_i to answer the ith signing query. This change is purely conceptual and thus $\mathcal{A}$'s view in Game 0 and Game 1 are identical. Thereby, we have:

$$\Pr[S_1] = \Pr[S_0]$$

Game 2. Same as Game 1 except that in the Setup stage $\mathcal{CH}$ also picks $\hat{i} \xleftarrow{R} [q]$, $\hat{j} \xleftarrow{R} [\lambda]$, $\hat{b} \xleftarrow{R} \{0,1\}$, and $\mathcal{A}$ is considered to win if it outputs a Type II forgery $(m^*, \sigma^* = (s^*, t^*))$ such that $m^*[\hat{j}] = \hat{b} \wedge m_{\hat{i}}[\hat{j}] \neq \hat{b} \wedge t_{\hat{i}} = t^*$ and $t_{\hat{i}} \neq t_i$ for all $i \neq \hat{i}$.

When $\mathcal{A}$ outputs a Type II forgery, we have $t^* \in \{t_1, \ldots, t_q\}$. Thus, the triple $(\hat{i}, \hat{j}, \hat{b})$ such that $m^*[\hat{j}] = \hat{b} \wedge m_{\hat{i}}[\hat{j}] \neq \hat{b} \wedge t_{\hat{i}} = t^*$ must exist. Moreover, since $t_1, \ldots, t_q$ are randomly chosen from $\{0,1\}^{\lambda}$, the probability that $t_{\hat{i}} \neq t_i$ for all $i \neq \hat{i}$ is smaller than $q/2^{\lambda}$. Since the view of $\mathcal{A}$ in Game 1 is identical to its view in Game 2 and $\mathcal{A}$ learns no information on which $(\hat{i}, \hat{j}, \hat{b})$ is chosen, we have:

$$\Pr[S_2] \geq \Pr[S_1]/(2q\lambda) - q/2^{\lambda}$$

Game 3. Same as Game 2 except that $\mathcal{CH}$ uses a punctured key of F_1 to handle signing queries: (i) computes $k_1(\{a\}) \leftarrow F_1.\mathsf{Puncture}(k_1, a)$ where $a = \hat{j}||\hat{b}||t_{\hat{i}}$ in the Setup stage; (ii) on the ith signing queries $m_i \neq m^*$, computes $s_i = \oplus_{j=1}^{\lambda} F_1(k_1(\{a\}), j||m_i[j]||t_i) \oplus_{j=1}^{\lambda} F_2(k_2, t_i(j))$ and returns $\sigma_i = (s_i, t_i)$. Due to the the correctness of sPPRFs and the fact that $\mathcal{A}$ succeeds in

Game 2 only if we have $m^*[\hat{j}] \neq m_{\hat{i}}[\hat{j}]$ and $t^* \neq t_i$ for all $i \neq \hat{i}$, in which case $F_1(k_1, a)$ is never used when answering signing queries, Game 2 and Game 3 are identical in $\mathcal{A}$'s view and thus we have:

$$\Pr[S_3] = \Pr[S_2]$$

Game 4. Same as Game 3 except that $\mathcal{CH}$ computes $z^* \leftarrow F_2(k_2, a)$ and generates $ct \leftarrow \mathsf{SKE.Enc}(k_e, z^*)$ rather than $ct \leftarrow \mathsf{SKE.Enc}(k_e, 0^n)$ in the Setup stage. By the security of SKE, we have:

$$|\Pr[S_4] - \Pr[S_3]| \leq \mathsf{Adv}^{\mathrm{SKE}}_{\mathcal{A}}$$

Game 5. Same as Game 4 except that $\mathcal{CH}$ computes $z^* \leftarrow F_1(k_1, a)$ and creates $C_{\mathrm{sign}} \leftarrow i\mathcal{O}(\mathrm{Sign}^{**})$ in the Setup stage. Here, the program Sign** (defined in Figure 22) is built from constants $k_1(\{a\})$, k_2, z^*, η^*, and a.

Sign**

Constant: $k_1(\{a\})$, k_2, z^*, η^*, and a

Input: ct, message m, and randomness t

1. If $h(ct) \neq \eta^*$, output $\perp$.
2. If $\hat{j}||m[\hat{j}]||t = a$, then compute $z^* \leftarrow \mathsf{SKE.Dec}(k_e, ct)$ and $s \leftarrow \oplus_{j \neq \hat{j}} F_1(k_1(\{a\}), j||m[j]||t) \oplus z^* \oplus_{j=1}^{\lambda} F_2(k_2, t(j))$, and output $\sigma = (s, t)$.
3. Else, compute $s = \oplus_{j=1}^{\lambda} F_1(k(\{a\}), (j||m[j]||t) \oplus_{j=1}^{\lambda} F_2(k_2, t(j))$, and output $\sigma = (s, t)$.

Figure 22: Program Sign**

It is easy to check that the two programs Sign and Sign** agree on all inputs due to the injectivity of g. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_5] - \Pr[S_4]| \leq \mathsf{Adv}^{i\mathcal{O}}_{\mathcal{A}}$$

Game 6. Same as Game 5 except that $\mathcal{CH}$ computes $y^* \leftarrow g(z^*)$ and creates $C_{\mathrm{vefy}} \leftarrow i\mathcal{O}(\mathrm{Verify}^{**})$ in the Setup stage. Here, the program Verify** (defined in Figure 23) is built from constants $k_1(\{a\})$, k_2, g, y^*, and a.

Verify**

Constant: $k_1(\{a\})$, k_2, g, y^*, a

Input: message m and signature $\sigma = (s, t)$

1. If $a = \hat{j}||m[\hat{j}]||t$, then output "1" if $g(s \oplus_{j \neq \hat{j}} F_1(k_1(\{a\}), j||m[j]||t) \oplus_{j=1}^{\lambda} F_2(k_2, t(j)) = y^*$ and output "0" otherwise.
2. Else, output 1 if $g(s) = g(\oplus_{j=1}^{\lambda} F_1(k_1(\{a\}), j||m[j]||t) \oplus_{j=1}^{\lambda} F_2(k_2, t(j))$ and output "0" otherwise.

Figure 23: Program Verify**

It is easy to check that the two programs Verify and Verify** agree on all inputs due to the injectivity of g. By the security of $i\mathcal{O}$, we have:

$$|\Pr[S_6] - \Pr[S_5]| \leq \mathsf{Adv}^{i\mathcal{O}}_{\mathcal{A}}$$

Game 7. Same as Game 6 except that in the Setup stage $\mathcal{CH}$ picks $z^* \xleftarrow{R} \{0,1\}^n$ rather than setting $z^* \leftarrow F_1(k_1, a)$.

By the selective pseudorandomness of sPPRFs, we have:

$$|\Pr[S_7] - \Pr[S_6]| \leq \mathsf{Adv}^{\mathrm{sPPRF}}_{\mathcal{A}}$$

It remains to analyze $\Pr[S_7]$. We have the following claim.

Claim D.3. *If g is injective and ℓ-entropy-leakage-resilient one-way, then the advantage of any PPT adversary in Game 7 is negligible in λ.*

Proof. Let $\mathcal{A}$ be a PPT adversary that wins Game 7 with advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$. We build an adversary $\mathcal{B}$ that breaks the assumed entropy-leakage-resilient one-wayness of g with the same advantage, implying that $\Pr[S_7]$ must be negligible.

Given g, y^* where $y^* \leftarrow g(z^*)$ for some $z^* \xleftarrow{R} \{0,1\}^n$, $\mathcal{B}$ interacts with $\mathcal{A}$ in Game 7 with the aim to output z^*.

1. Setup: $\mathcal{B}$ picks $t_1, \ldots, t_q \xleftarrow{R} \{0,1\}^\lambda$, $\hat{i} \xleftarrow{R} [q]$, $\hat{j} \xleftarrow{R} [\lambda]$, $\hat{b} \xleftarrow{R} \{0,1\}$, generates $(pp_1, k_1) \leftarrow F_1.\mathsf{Gen}(\lambda)$, $(pp_2, k_2) \leftarrow F_2.\mathsf{Gen}(\lambda)$, sets $a = \hat{j}||\hat{b}||t_{\hat{i}}$, computes $k_1(\{a\}) \leftarrow F_1.\mathsf{Puncture}(k_1, a)$, and picks $k_e \leftarrow \mathrm{SKE}.\mathsf{Gen}(\lambda)$ and $h \leftarrow \mathrm{LF.GenLossy}(\lambda)$. Next $\mathcal{B}$ picks a fresh randomness r for encryption, defines function $\psi(\cdot) := \mathrm{SKE}.\mathsf{Enc}(k_e, \cdot; r)$, and makes a leakage query $\langle h \circ \psi \rangle$ about z^* to obtain η^*. Then it creates $C_{\mathrm{vefy}} \leftarrow i\mathcal{O}(\mathrm{Verify}^{**})$ and $C_{\mathrm{sign}} \leftarrow i\mathcal{O}(\mathrm{Sign}^{**})$, and sends $vk = (C_{\mathrm{vefy}}, C_{\mathrm{sign}})$ to $\mathcal{A}$.
2. Signing Query: Upon receiving the ith signing query $\langle m_i \rangle$, $\mathcal{B}$ responds as follows: computes $s_i \leftarrow \oplus_{j=1}^{\lambda} F_1(k_1(\{a\}), j||m_i[j]||t_i) \oplus_{j=1}^{\lambda} F_2(k_2, t_i(j))$, then returns $\sigma_i = (s_i, t_i)$ to $\mathcal{A}$.
3. Leakage Query: Note that for any leakage query on $sk = ct$, $\mathcal{B}$ can transform it to leakage query on z^*. Upon receiving leakage query $\langle f \rangle$, $\mathcal{B}$ makes a leakage query $\langle f \circ \psi \rangle$ about z^* to its own challenger and forwards the reply to $\mathcal{A}$.
4. Forge: $\mathcal{A}$ outputs $(m^*, \sigma^* = (s^*, t^*))$ and wins if it is a Type II forgery such that $m^*[\hat{j}] = \hat{b} \wedge m_{\hat{i}}[\hat{j}] \neq \hat{b} \wedge t_{\hat{i}} = t^*$, and $t^* \neq t_i$ for all $i \neq \hat{i}$.

Finally, $\mathcal{B}$ forwards $s^* \oplus_{j \neq \hat{j}} F_1(k_1(\{a\}), j||m^*[j]||t^*) \oplus_{j=1}^{\lambda} F_2(k_2, t^*(j))$ to its own challenger. Conditioned on $\mathcal{A}$ succeeds, we have $m^*[\hat{j}] = \hat{b} \wedge t_{\hat{i}} = t^*$. According to the definition of Verify**, this implies $g(s^* \oplus_{j \neq \hat{j}} F_1(k_1(\{a\}), j||m^*[j]||t^*) \oplus_{j=1}^{\lambda} F_2(k_2, t^*(j)) = y^*$. Moreover, η^* reveals at most τ bits information about ct and thus z^*, since h is a (v, τ)-lossy function. Hence, leakage queries made by $\mathcal{B}$ will be correctly answered by its challenger. As a result, the probability that $\mathcal{B}$ wins is $\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_7]$, which is negligible by the security of g. This proves the claim. □

Since $2q\lambda$ is a polynomial in λ and $q/2^\lambda$ is negligible, taken all together, $\Pr[S_0]$ is also negligible in λ, i.e., the probability of $\mathcal{A}$ outputs a Type II forgery is negligible in λ. This finishes the second part of the proof.

Putting all the above together, the theorem immediately follows. □

Remark D.1. An important ingredient of the above construction is entropy leakage-resilient *injective* OWFs, which are implied by lossy functions. Actually, by assuming the underlying OWF is entropy-leakage-resilient, the leakage-resilience of the above construction also hold in the entropy leakage model, due to the signing procedures are deterministic.

In the above construction, we sample the randomness t used by the signing algorithm from $\{0,1\}^\lambda$ only for simplicity. The security proof still holds if we sample t from $\{0,1\}^{\lambda^c}$ for any constant $c < 1$ instead. As a result, the length of signature is $(n + \lambda^c)$.