Alpirez Bock, Estuardo; Brzuska, Chris; Lai, Russell W. F.

Simple Watermarking Pseudorandom Functions from Extractable Pseudorandom Generators

Published in:
IACR Communications in Cryptology

DOI:
10.62056/aevur-10k

Published: 01/01/2024

Document Version
Publisher's PDF, also known as Version of record

IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 2, 27 pages.

https://doi.org/10.62056/aevur-10k

Simple Watermarking Pseudorandom Functions from Extractable Pseudorandom Generators

Estuardo Alpirez Bock[1], Chris Brzuska[2] and Russell W. F. Lai[2]

[1] Independent, United Kingdom
[2] Aalto University, Finland

Abstract. Watermarking pseudorandom functions (PRF) allow an authority to embed an unforgeable and unremovable watermark into a PRF while preserving its functionality. In this work, we extend the work of Kim and Wu [Crypto'19] who gave a simple two-step construction of watermarking PRFs from a class of extractable PRFs satisfying several other properties – first construct a mark-embedding scheme, and then upgrade it to a message-embedding scheme.
While the message-embedding scheme of Kim and Wu is based on complex homomorphic evaluation techniques, we observe that much simpler constructions can be obtained and from a wider range of assumptions, if we forego the strong requirement of security against the watermarking authority. Concretely, we introduce a new notion called extractable PRGs (xPRGs), from which extractable PRFs (without security against authorities) suitable for the Kim-Wu transformations can be simply obtained via the Goldreich-Goldwasser-Micali (GGM) construction. We provide simple constructions of xPRGs from a wide range of assumptions such as hardness of computational Diffie-Hellman (CDH) in the random oracle model, as well as LWE and RSA in the standard model.

Keywords: watermarking · extractable pseudorandom function · extractable pseudorandom generator

1 Introduction

Software watermarking, formally introduced by Barak et al. [BGI+12] and Hopper et al. [HMW07], allows an authority to embed a secret message into a program while preserving its functionality, e.g. to identify the program's owner or validate the program's authenticity. *Unremovability* and *unforgeability* require that an unauthorised party can neither remove the watermark of a program while preserving its functionality, nor forge programs which are considered watermarked.

1.1 Constructions from Obfuscation and Lattices

Cohen et al. [CHN+16] prove that watermarking classes of *learnable* functions is impossible, and use indistinguishability obfuscation (iO) to watermark *cryptographic* functionalities. Subsequent works [KW17, QWZ18, KW19] focus on constructing watermarking pseudorandom functions (PRF), under different computational assumptions and achieving different combinations of properties in addition to unremovability and unforgeability.

Currently, regardless of which additional properties are achieved, most known watermarking PRF constructions are either based on iO or the learning with rounding (LWR) assumption [BPR12] (with unbounded polynomial number of samples), the latter

E-mail: chris.brzuska@aalto.fi (Chris Brzuska), russell.lai@aalto.fi (Russell W. F. Lai)

Received: 2024-04-07 Accepted: 2024-06-03

of which is implied [BGM+16] by the learning with errors (LWE) assumption [Reg05] with a super-polynomial modulus-to-noise ratio.[1]

We highlight two landmark constructions based on LWE. The first is a construction of Quach et al. [QWZ18] from CCA-secure public-key encryption schemes with pseudorandom ciphertexts and privately puncturable PRFs, both of which can be construct based on LWE. Their construction is publicly watermarkable but does not provide security against authorities. The second is the construction of Kim and Wu [KW19] from extractable PRFs satisfying several other properties, which they constructed from the LWE assumption. Their construction achieves weak-pseudorandomness against semi-honest authorities (i.e. authorities which have no choice over the generation of the public parameters and the trapdoors, but who have knowledge of *how* these are generated), and is publicly watermarkable in the random oracle model.

A notable exception to the above classification is another construction of Quach et al. [QWZ18] from CCA-secure public-key encryption schemes with pseudorandom ciphertexts and (standard) puncturable PRFs, which can be instantiated from a wide range of assumptions such as CDH (with random oracles), DDH, DCR, QR, and LWE. However, this construction only achieves a weaker functionality known as mark-embedding which, as opposed to the standard notion called message-embedding, only allows to watermark the PRF with a single message, e.g. "marked". Although Kim and Wu [KW19] presented a way to extend mark-embedding schemes to message-embedding schemes, the transformation is non-black-box and does not seem to apply to the mark-embedding scheme of Quach et al.

1.2 Extractable PRFs

Watermarkable PRFs. The starting point of this work is the simple and generic transformation of Kim and Wu [KW19], which turns PRFs satisfying an assortment of different properties – what we call *watermarkable* PRFs – to *watermarking* PRFs.

In addition to pseudorandomness and (standard) puncturability, a core property of watermarkable PRFs is *robust extractability*, as coined by Kim and Wu [KW19].[2] Roughly, a PRF f with trapdoored public parameters has robust extractability, if the PRF key k can be extracted by the authority from any circuit C which approximates $f(k, \cdot)$. Intuitively, this means that $f(k, .)$ must encode k redundantly, and thus it should be difficult to come up with a circuit C which approximates $f(k, .)$ but loses information about k.

From Watermarkability to Watermarking. We summarise the Kim-Wu transformation [KW19] from watermarkable PRFs to watermarking PRFs, and highlight the use of the various properties of watermarkable PRFs when relevant. For formal details, we refer to Appendix B and [KW19]. The transformation proceeds in two steps: 1) From a watermarkable PRF f and an ordinary PRF g, construct a watermarking PRF F which achieves the weaker functionality of mark-embedding. 2) Upgrade mark-embedding to message-embedding via parellel composition (in a non-black-box manner).

The watermarking key for the mark-embedding scheme consists of the trapdoor of the underlying watermarkable PRF f and a secret key of an ordinary PRF g. To mark a PRF key k of f, the authority derives an input x_k using g, then punctures the key k of f at the input x_k. The watermarked PRF program of F is then simply the PRF evaluation algorithm for f with the punctured key hardwired. To check whether a circuit C is marked, using the robust extractability of f, the authority extracts a candidate secret key k' by querying C, and outputs `MARKED` if $C(x_{k'}) \neq f(k', x_{k'})$ and `UNMARKED` otherwise.

[1] Since LWR is implied by LWE when the number of samples is a priori bounded or when the modulus-to-noise ratio is super-polynomial, from here on we will use LWR and LWE interchangeably whenever the security of a construction relies on LWR.

[2] Kim and Wu [KW19] also required key-injectivity, which we gloss over here.

The pseudorandomness of F is inherited from that of f. The unremovability of F is simply based on the puncturing security of f. To achieve unforgeability, the accuracy of the above check can be amplified by puncturing a larger set X_k of inputs and outputting `MARKED` if $C(x) \neq f(k', x)$ for all x in $X_{k'}$.

To extend the mark-embedding scheme to a scheme which embeds a secret message $m \in \{0,1\}^t$, the authority could derive $2t$ sets $X_{k,i,b}$ for $i \in [t]$ and $b \in \{0,1\}$ and puncture the key k at $\bigcup_{i\in[t]} X_{k,i,m_i}$. For recovering the watermark, the authority extracts a candidate key k as described above, derives the sets $X_{k,i,b}$ and checks for which b the circuit agrees the most with the PRF on the candidate key. Finally, the scheme can be made publicly markable by replacing the ordinary PRF with a random oracle.

As shown above, the overhead of turning a watermarkable PRF into a watermarking PRF is minimal. Therefore, the main complexity of constructing the latter under this framework lies in constructing the former.

Constructing Watermarkable PRFs. The watermarkable PRF constructed by Kim and Wu [KW19] is very strong, in the sense that it achieves (weak-)pseudorandomness against (semi-honest) authorities. To achieve this, Kim and Wu rely extensively on complex lattice-based homomorphic evaluation techniques, which were originally developed in the context of attribute-based encryption for circuits [BGG+14]. On the other end of the spectrum, if we were to drop the properties of robust extractability and security against authorities, then even the classic Goldreich-Goldwasser-Micali (GGM) construction [GGM84] can achieve the remaining properties of watermarkable PRFs, solely assuming the existence of one-way functions.

Since security against authorities might not be necessary for applications where the authority is naturally trusted, this poses a natural question:

Can we construct simpler watermarkable PRFs? In particular, can we achieve robust extractability without homomorphic computation?

1.3 Our Contributions

We give affirmative answers to the above question. In short, our main results are simple constructions of watermarkable PRFs from CDH (in the random oracle model), LWE, or trapdoor permutations (hence RSA) which, through the simple transformation of Kim and Wu [KW19], also yield simple watermarking PRFs in settings where the authority is trusted. Along the way, we show that the robust extractability property required by Kim and Wu is implied generically by other simpler properties. The core of our constructions is a new primitive which we call *extractable pseudorandom generator (xPRG)* which implies watermarkable PRFs through the GGM construction. Due to the general applicability of PRGs, the new notion of xPRGs may by of independent interest.

Simplifying Robust Extractability. We define a watermarkable PRF to be a PRF which is pseudorandom, extractable, and (publicly) puncturable. Specifically, our notion of extractability means that, given any valid input-output pair $(x, f(k, x))$ of the PRF f with key k, the trapdoored extraction algorithm is able to extract the key k. In Section 3.3, we show that the above simple notion of extractability suffices to imply the robust extractability property required by Kim and Wu [KW19].[3]

Watermarkable PRFs from Extractable PRGs. While Kim and Wu [KW19] construct watermarkable PRFs directly from lattice-based homomorphic evaluation techniques,

[3]We also observe that extractability immediately implies key-injectivity.

our construction of watermarkable PRFs follows a completely different strategy and, arguably, is much simpler. More concretely, our approach is to first construct an intermediate object which we call extractable pseudorandom generators (xPRG), which map a seed x to e outputs $(y_1, \ldots, y_e)$ such that x can be efficiently recovered given the trapdoor and any output y_i. We then instantiate the GGM construction in Section 5 with these xPRGs to obtain watermarkable PRFs.

For an xPRG to be compatible with the GGM construction, it must satisfy two functional properties. First, for recursive composability, the inputs and outputs of the xPRG should be over a common alphabet.[4] Second, for the output length to remain polynomial after composition, it is crucial that the underlying xPRG is "rate-1", i.e. $|y_i| = |x| + \mathsf{poly}(\lambda)$, where $|\cdot|$ denotes the description length and the additive polynomial overhead is independent of $|x|$.

Simple Constructions of Extractable PRGs. We provide three constructions of xPRGs in Section 4:

(1) First we construct a rate-1 trapdoor function (TDF) family under the computational Diffie-Hellman (CDH) assumption in the random oracle model, and show that concatenating multiple instances of the TDF gives an xPRG.

(2) The second construction is based on the observation that an LWR function is itself a trapdoor function, and on a technique of extending the output length of a trapdoor function to achieve rate-1. Similar to the first construction, we prove that concatenating multiple TDFs yields an xPRG under the LWE assumption.[5]

(3) Since not all pseudorandom TDFs retain pseudorandomness under concatenation (e.g. the RSA-based trapdoor permutation), our third construction generically turns a trapdoor permutation into a rate-1 xPRG. This construction is based on the "doubly-half-injective PRGs" construction by Alpirez Bock et al. [AAB+19] from one-way permutation. We observe that their construction yields an xPRG if the one-way permutation is replaced with a trapdoor permutation, and generalise their construction with expansion factor $e = 2$ to arbitrary $e = \mathsf{poly}(\lambda)$.

Simple Watermarking PRFs. Putting everything together, we obtain simple constructions of watermarking PRFs via the following chain of transformations:

$$\text{CDH/LWE/RSA} \overset{(4)}{\to} \text{xPRG} \overset{\text{GGM } (5)}{\to} \text{Watermarkable PRF} \overset{\text{Kim-Wu } (B)}{\to} \text{Watermarking PRF}$$

In particular, we obtain watermarking PRFs which are unremovable and unforgeable in the presence of marking and mark-extraction oracles, and pseudorandom to any party except for the watermarking authority.

1.4 Related Work

Mark-Embedding Scheme of Quach et al. The watermarking PRF construction most similar to ours in terms of underlying assumptions and security properties is the mark-embedding scheme of Quach et al. [QWZ18]. In terms of security, both constructions achieve pseudorandomness in presence of an extraction oracle, but provide no security against the watermarking authority. In terms of underlying assumptions, the Quach et al. mark-embedding construction relies only on the existence of a CCA-secure public-key encryption scheme with pseudorandom ciphertexts, which can be instantiated with CDH

[4] Retaining extractability forbids non-injective strategies, e.g. hashing to bit strings.

[5] Concatenating LWE (instead of LWR) functions does not yield a PRG. Indeed, such a function is efficiently invertible by linear algebra. See Section 4.3 for details.

in the random oracle model, DDH, DCR, QR, or LWE. In contrast, our construction can be instantiated with CDH in the random oracle model, RSA, or LWE. In summary, in terms of differences in underlying assumptions, our construction admits an instantiation from RSA which, being a search assumption, can be viewed as slightly weaker than DCR and QR, which are decisional assumptions.

The main difference in the two constructions is in functionality. While our construction is message-embedding, the Quach et al. construction achieves only mark-embedding. Recall that our construction is obtained by applying the transformations by Kim and Wu [KW19], first (1) from a watermarkable PRF to a mark-embedding watermarking PRF, then (2) to a message-embedding watermarking PRF via a non-black-box extension of transform (1). It is therefore natural to ask if transform (2) can be applied to the mark-embedding scheme of Quach et al. or whether the mark-embedding scheme of Quach et al. can be modified to become compatible with (2). We are uncertain about the answer to this question. On the one hand, (2) does not apply directly to the Quach et al. scheme since the (single) punctured point is hardwired in the PRF key k (even before watermarking), while in the Kim-Wu transformation the (multiple) punctured points are derived from k using an additional secret PRF key held by the watermarking authority. On the other hand, it is conceivable that it is possible to apply these two modifications to the Quach et al. scheme while recovering the original security arguments.

Finally, we consider our construction conceptually simpler (in a pedagogical sense) since we do not rely on CCA-secure public-key encryption schemes which, under the hood, require hash-proof systems, double encryption with zero-knowledge proofs, or transformations such as Fujisaki-Okomoto transform.

Additional Properties. Some additional properties that watermarking PRFs could have are public watermarkability [QWZ18, KW19], public watermark verifiability [CHN+16], and security against the watermarking authorities [CHN+16, KW17]. In particular, security against authorities is useful in applications where the watermarked programs protect the privacy of the users against an untrusted watermarking authority. However, there are also applications where the watermarking authority is naturally trusted, such as when the watermarked program is used for authenticated or confidential communication with the authority, e.g. for software updates or broadcasting sports content.

Collusion Resistance. All watermarking PRFs discussed so far are insecure against colluding adversaries, whose goal is to remove the watermark given multiple markings of the same PRF. This issue was identified by Yang et al.[YAL+19] who construct collusion-resistant watermarkable PRFs from iO. Subsequently, Yang et al. [YAYX20] proposed a generic transformation turning watermarking PRFs, such as those constructed in this work, to collusion-resistant ones.

Tracing vs. Watermarking. Goyal et al. [GKWW21] argue that watermarking PRFs are *too weak* for traitor-tracing applications since programs can easily be useful without retaining input-output behaviour on a large fraction of the input, e.g. by recovering only half of the output value, or when used as subroutines, e.g. as part of a decryption program. In these cases, the watermarking authority may not be able to extract the mark from a forged program, since extraction is only guaranteed if given complete and correct outputs of the PRF.[6] They therefore propose the strengthened notion of traceable

[6]Strictly speaking, an adversarial program which drops logarithmically many bits would still be useful to a watermarking PRF extractor, since these logarithmically many bits can be guessed (even if somewhat inconvenient in practice). However, if the remaining bits can be guessed, then this also holds for the authority who can still extract the key, now by guessing the bits.

PRFs which allow extraction given oracle access to any distinguisher which breaks the weak-pseudorandomness of the PRF.

Goyal et al. [GKWW21] construct traceable PRFs from iO, and a single-key variant from LWE. Maitra and Wu [MW22] provide a generic compiler which upgrades traceable PRFs from single-key to multi-key. Since the extractor of a traceable PRF is only given oracle access to a distinguisher, to extract a high-entropy PRF key, it must run the distinguisher many times to extract multiple bits out of it. Designing a traceable-PRF-style extractor for our GGM-based construction of extractable PRFs appears to be a challenging problem.

While the argument by Goyal et al. against the usefulness of watermarking seems to apply to most applications of *confidentiality* where pseudorandomness is required, we argue that watermarking programs for *authenticating* data is still useful for traitor tracing, since using a PRF for authentication relies on the *unpredictability* of PRF values on a distribution defined by the application.[7] For example, if the PRF is used as a message-authentication code (MAC), then a useful adversarial program must produce complete PRF values.

2 Preliminaries

We define security games $\mathtt{G}$ as sets of stateful oracles $\{\mathsf{O}_1, \ldots, \mathsf{O}_n\}$ that the adversary can interact with. All games implicitly input a security parameter 1^λ and a length parameter 1^ℓ, which are omitted. We write $\mathcal{A} \xrightarrow{\mathsf{O}_1,\ldots,\mathsf{O}_\ell} \mathtt{G}$ for the adversary interacting with game $\mathtt{G}$ by calling oracles O_1,...,O_n or simply $\mathcal{A} \to \mathtt{G}$ if the oracles are clear from context. In particular, we write $\Pr\left[1 = \mathcal{A} \xrightarrow{\mathsf{O}_1,\ldots,\mathsf{O}_n} \mathtt{G}\right]$ or $\Pr[1 = \mathcal{A} \to \mathtt{G}]$ for the probability that $\mathcal{A}$ returns 1 when interacting with the oracles $\mathsf{O}_1, \ldots, \mathsf{O}_n$ of game $\mathtt{G}$. In our pseudocode, we write "**assert** x" to enforce call restrictions on the adversary. If the condition x does not hold, the oracle call aborts and returns an error to the adversary.

2.1 Groups

Let ggen be a PPT algorithm which inputs the security parameter 1^λ and outputs (the description of) a cyclic group $\mathbb{G}$, its (prime) order $q \in \mathbb{N}$, and a generator $[1] \in \mathbb{G}$. We assume that $\mathbb{G}$ and q depend deterministically on λ. We write group operations additively, i.e. $[x] + [y] = [x + y]$ and $x \cdot [y] = [x \cdot y]$ for all $x, y \in \mathbb{Z}_q$.

Definition 1 (CDH)**.** The computational Diffie-Hellman (CDH) assumption is said to hold relative to $(\mathbb{G}, q, [1]) \in \mathsf{ggen}(1^\lambda)$ if for any PPT algorithm $\mathcal{A}$

$$\Pr\left[\mathcal{A}(\mathbb{G}, q, [1], [g], [h], [g \cdot s]) = [h \cdot s] \;\middle|\; g, h, s \leftarrow\!\!\$\; \mathbb{Z}_q\right] = \mathsf{negl}(\lambda).$$

2.2 Lattices

For $p, q \in \mathbb{N}$ with $p \leq q$, let $\lfloor\cdot\rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ denote the function which sends $x \in \mathbb{Z}_q$ to $\lfloor \frac{p \cdot x}{q} \rceil \bmod p$, where $\lfloor\cdot\rceil$ rounds a real number to its closest integer. If $\mathbf{x} = (x_i)_{i \in \mathbb{Z}_n}$ is a vector over $\mathbb{Z}$, we denote by $\|\mathbf{x}\| = \max_{i \in \mathbb{Z}_n} |x_i|$ the infinity norm of $\mathbf{x}$. If $\mathbf{x}, \mathbf{y}$ are vectors over $\mathbb{Z}$ satisfy $\|\mathbf{x} - \mathbf{y}\| \leq B$, we write $\mathbf{x} \approx_B \mathbf{y}$.

Definition 2 (LWR)**.** Let $(n, m, p, q) = (n, m, p, q)(\lambda)$ with $2 \leq p \leq q$ and χ be a distribution over $\mathbb{Z}_q$. The learning with rounding (LWR) assumption is said to hold for (n, m, p, q, χ) if for any PPT algorithm $\mathcal{A}$

$$\left| \begin{array}{l} \Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{b}) = 1 \;\middle|\; \mathbf{s} \leftarrow\!\!\$\; \chi^n,\ \mathbf{A} \leftarrow\!\!\$\; \mathbb{Z}_q^{n \times m},\ \mathbf{b} \leftarrow \lfloor \mathbf{s} \cdot \mathbf{A} \rceil_p\right] \\ -\ \Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{b}) = 1 \;\middle|\; \mathbf{A} \leftarrow\!\!\$\; \mathbb{Z}_q^{n \times m},\ \mathbf{b} \leftarrow\!\!\$\; \mathbb{Z}_p^m\right] \end{array} \right| = \mathsf{negl}(\lambda).$$

[7] Hashing before running the PRF thus means that we rely on the PRF *unpredictability* on the *uniform* distribution which is compatible with watermarking weak PRFs.

Throughout this work we focus on the case where χ is the uniform distribution over $\mathbb{Z}_p \subseteq \mathbb{Z}_q$, denoted simply by $\mathbb{Z}_p$. A series of works (e.g. [BGM+16]) showed that the LWR problem is as hard as the learning with errors (LWE) problem in cases where the number of samples m is polynomially bounded or the modulus-to-noise ratio is super-polynomial. Direct constructions of PRFs from LWR usually require the LWR assumption to hold for an a priori unbounded number of samples. In this work, however, we only require the LWR assumption to hold for some fixed polynomial m number of samples, since we are constructing (x)PRFs from (x)PRGs using the GGM construction, where the security of each (x)PRG instance relies on LWR with a fixed polynomial number of samples.

Lemma 1 (Lattice Trapdoors, e.g. [MP12])**.** *There exist PPT algorithms:*

- $(\mathbf{A}, td) \leftarrow\!\!\$\ \mathsf{TrapGen}(1^n, 1^m, p, q)$*: The trapdoor generation algorithm inputs dimensions and moduli $n, m, p, q \in \mathbb{N}$ with $2 \leq p \leq q$ and outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor td.*
- $\mathbf{s} \leftarrow \mathsf{LWEInvert}(td, \mathbf{y})$*: The deterministic LWE inversion algorithm inputs a trapdoor td and a vector $\mathbf{y} \in \mathbb{Z}_p^m$ and outputs a vector $\mathbf{s} \in \mathbb{Z}^n$.*

Furthermore, for any $n, m, p, q \in \mathbb{N}$ with $m = \Omega(n \log q)$ and $q = \Omega(np\sqrt{\log q})$,

- *the distribution $\left\{\mathbf{A} : (\mathbf{A}, td) \leftarrow\!\!\$\ \mathsf{TrapGen}(1^h, 1^k, p, q)\right\}$ and the uniform distribution over $\mathbb{Z}_q^{h \times k}$ are statistically close in λ, and*
- *for any $\mathbf{s} \in \mathbb{Z}_p^h$, it holds that $\mathsf{LWEInvert}(td, \lfloor \mathbf{s} \cdot \mathbf{A} \rceil_p) = \mathbf{s}$.*

3 Extractable PRFs and PRGs

In this section, we introduce the new notion of extractable pseudorandom generators (Section 3.1), and recall the definitions of extractable pseudorandom functions and their properties (Section 3.2). The properties that we define for xPRFs slightly differ from existing ones [KW19]. In Section 3.3, we show that the robust extractability property required by Kim and Wu [KW19] is generically implied by other simpler properties.

3.1 Extractable Pseudorandom Generators (xPRG)

We provide a natural definition of extractable pseudorandom generators (xPRG). Extractability for a PRG with expansion factor e means that, given the trapdoor and one of the e outputs of the PRG, the seed which produces the output can be efficiently recovered.

Definition 3 (Extractable PRG)**.** An extractable pseudorandom generator (xPRG) family $\mathcal{G}$ with family of domains $(\mathcal{X}_{\lambda,\ell})_{\lambda,\ell \in \mathbb{N}}$ and codomains $(\mathcal{Y}^e_{\lambda,\ell})_{\lambda,\ell \in \mathbb{N}}$ and expansion factor e consists of three PPT algorithms $(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ with the following syntax:

- $(pp, td) \leftarrow\!\!\$\ \mathsf{setup}(1^\lambda, 1^\ell)$: The setup algorithm inputs 1^λ and length parameter 1^ℓ and outputs the public parameters pp and a trapdoor td.
- $(y_i)_{i \in \mathbb{Z}_e} \leftarrow \mathsf{eval}(pp, x)$: The deterministic evaluation algorithm inputs the public parameters pp and a preimage $x \in \mathcal{X}_{\lambda,\ell}$ and outputs e images $(y_i)_{i \in \mathbb{Z}_e} \in \mathcal{Y}^e_{\lambda,\ell}$. To ease notation, we write $y_i \leftarrow \mathsf{eval}(pp, i, x)$ for $i \in \mathbb{Z}_e$.
- $x \leftarrow \mathsf{inv}(td, i, y)$: The deterministic inversion algorithm inputs a trapdoor td, an index $i \in \mathbb{Z}_e$, and an image $y \in \mathcal{Y}_{\lambda,\ell}$ and outputs a preimage $x \in \mathcal{X}_{\lambda,\ell}$.

The rate $r(\lambda)$ of the xPRG is defined as $r(\lambda) := \lim_{\ell \to \infty} \log |\mathcal{X}_{\lambda,\ell}| / \log |\mathcal{Y}_{\lambda,\ell}|$.

Definition 4 (Pseudorandomness)**.** An xPRG family $\mathcal{G}$ is pseudorandom if for any $\ell = \mathsf{poly}(\lambda)$ and any PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{prg}}_{\mathcal{G},\mathcal{A}}(\lambda) := \left|\Pr\left[1 = \mathcal{A}^{\overset{\mathsf{setupO},\mathsf{evalO}}{\longrightarrow}}\, \mathtt{Prg}^0_{\mathcal{G}}\right] - \Pr\left[1 = \mathcal{A}^{\overset{\mathsf{setupO},\mathsf{evalO}}{\longrightarrow}}\, \mathtt{Prg}^1_{\mathcal{G}}\right]\right| = \mathsf{negl}(\lambda),$$

where Figure 1 defines $\mathtt{Prg}^b_{\mathcal{G}}$ for $b \in \{0,1\}$.

Definition 5 (t-Extractability)**.** Let $t \in [e]$. An xPRG family $\mathcal{G}$ is t-extractable if for any $\lambda, \ell \in \mathbb{N}$, $(pp, td) \in \mathsf{setup}(1^\lambda, 1^\ell)$, $x \in \mathcal{X}_{\lambda,\ell}$, $(y_0, \ldots, y_{e-1}) \in \mathsf{eval}(pp, x)$, $i \in \mathbb{Z}_t \subseteq \mathbb{Z}_e$, it holds that $\mathsf{inv}(td, i, \mathsf{eval}(pp, x)) = x$. If $\mathcal{G}$ is e-extractable, we simply say that it is extractble. If $\mathcal{G}$ is 1-extractable, we call $\mathcal{G}$ a 1xPRG family.

3.2 Extractable Pseudorandom Functions (xPRF)

We give slightly different definitions of extractable pseudorandom functions (xPRF) and their properties than those of Kim and Wu [KW19]. The main difference is that our notion of extractability requires to extract the PRF secret key given a valid input-output pair, while the robust extractability property [KW19] requires to extract the PRF secret key given a circuit that approximates the PRF. Another difference is that we omit security properties against the authority, such as "T-Restricted Pseudorandomness Given the Trapdoor" [KW19, Definition 4.7]. In Section 3.3, we discuss how our simpler notion of extractability generically implies robust extractability.

Definition 6 (Extractable PRF)**.** An extractable pseudorandom function (xPRF) family $\mathcal{F}$ with family of key spaces $(\mathcal{K}_\lambda)_{\lambda\in\mathbb{N}}$, domains $(\mathcal{X}_{\lambda,\ell})_{\lambda,\ell\in\mathbb{N}}$ and codomains $(\mathcal{Y}_{\lambda,\ell})_{\lambda,\ell\in\mathbb{N}}$ consists of three PPT algorithms $(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ with the following syntax:

- $(pp, td) \leftarrow\!\!\$\ \mathsf{setup}(1^\lambda, 1^\ell)$: The setup algorithm takes 1^λ and length parameter 1^ℓ as input and outputs the public parameters pp and a trapdoor td.
- $y \leftarrow \mathsf{eval}(pp, k, x)$: The deterministic evaluation algorithm takes pp, a secret key $k \in \mathcal{K}_\lambda$, and a value $x \in \mathcal{X}_{\lambda,\ell}$ and outputs an image $y \in \mathcal{Y}_{\lambda,\ell}$.
- $k \leftarrow \mathsf{inv}(td, x, y)$: the deterministic invert algorithm takes as input a trapdoor, a pre-image x and an image y and returns the secret key k.

Definition 7 (Pseudorandomness)**.** An xPRF family $\mathcal{F}$ is pseudorandom if for any $\ell = \mathsf{poly}(\lambda)$ and any PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F},\mathcal{A}}(\lambda) := \left|\Pr\left[1 = \mathcal{A}^{\overset{\mathsf{setupO},\mathsf{evalO}}{\longrightarrow}}\, \mathtt{Prf}^0_{\mathcal{F}}\right] - \Pr\left[1 = \mathcal{A}^{\overset{\mathsf{setupO},\mathsf{evalO}}{\longrightarrow}}\, \mathtt{Prf}^1_{\mathcal{F}}\right]\right| = \mathsf{negl}(\lambda),$$

where Figure 1 defines $\mathtt{Prf}^b_{\mathcal{F}}$ for $b \in \{0,1\}$.

Let $\mathcal{K}' = (\mathcal{K}'_\lambda)_\lambda$ be a family of key spaces and let $\mathcal{D}$ be a PPT function generator which inputs $(1^\lambda, 1^\ell)$ and outputs (the description of) a function $\delta : \mathcal{K}'_\lambda \times \mathcal{X}_{\lambda,\ell} \to \mathcal{K}_\lambda$. An xPRF family $\mathcal{F}$ is weakly-pseudorandom against semi-honest extraction authorities under $\mathcal{D}$-weak-input-related-key-attacks if for any $\ell = \mathsf{poly}(\lambda)$ and PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{wprfaa}}_{\mathcal{F},\mathcal{D},\mathcal{A}}(\lambda) := \left|\Pr\left[1 = \mathcal{A}^{\overset{\mathsf{setupO}}{\mathsf{evalO}}}\, \mathtt{wPrfaa}^0_{\mathcal{F},\mathcal{D}}\right] - \Pr\left[1 = \mathcal{A}^{\overset{\mathsf{setupO}}{\mathsf{evalO}}}\, \mathtt{wPrfaa}^1_{\mathcal{F},\mathcal{D}}\right]\right| = \mathsf{negl}(\lambda),$$

where Figure 1 defines $\mathtt{wPrfaa}^b_{\mathcal{F},\mathcal{D}}$ for $b \in \{0,1\}$. In case δ_x is the identity function for all x, then we simply omit $\mathcal{D}$ from all subscripts and say that $\mathcal{F}$ is weakly-pseudorandom against semi-honest extraction authorities.

Definition 8 (t-Extractability)**.** Let $t = t(\lambda, \ell)$. An xPRF family $\mathcal{F}$ is t-extractable if for any $\lambda, \ell \in \mathbb{N}$ and $(pp, td) \in \mathsf{setup}(1^\lambda, \ell)$,

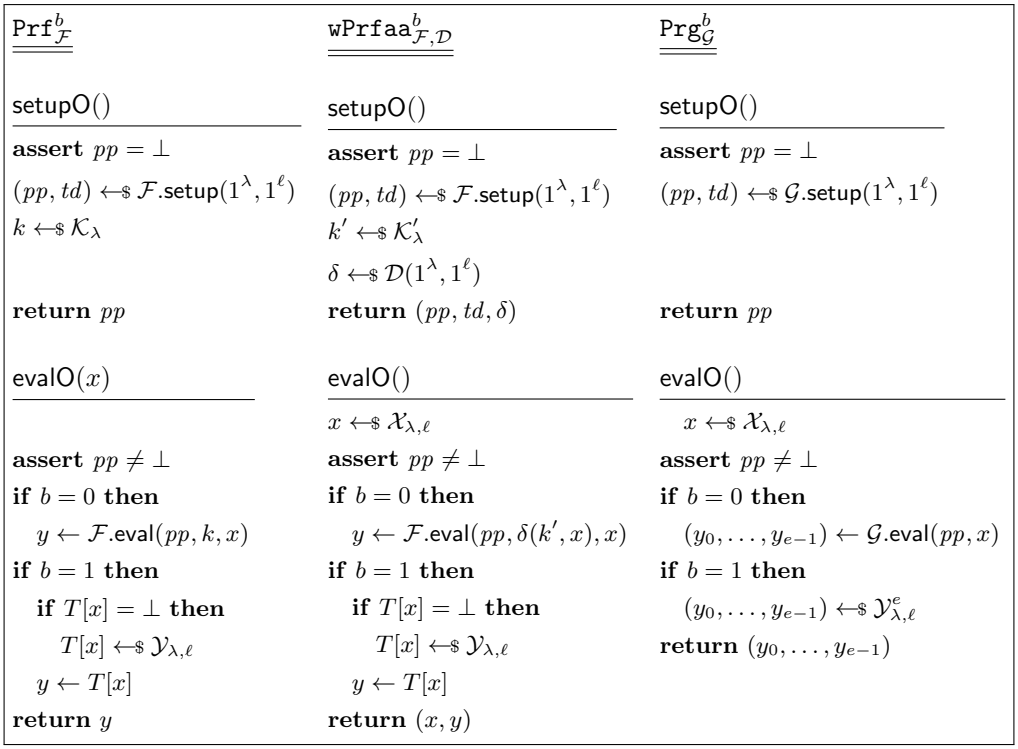

Figure 1: Pseudorandomness of xPRFs (left), weak-pseudorandomness of xPRFs against semi-honest extraction authorities under weak-input-related-key attacks (middle), and pseudorandomness of xPRGs (right).

- the trapdoor td (implicitly) specifies a t-subset $\mathcal{S} = \mathcal{S}(td) \subseteq \mathcal{X}_{\lambda,\ell}$, and
- for any $k \in \mathcal{K}_\lambda$ and $x \in \mathcal{S}(td)$, it holds that $\mathsf{inv}(td, x, \mathsf{eval}(pp, k, x)) = k$.

If $\mathcal{S}(td) \equiv \mathcal{X}$, then we simply say that $\mathcal{F}$ is extractable.

Remark 1. Weak pseudorandomness as specified in Figure 1 can only be achieved for t-extractable PRFs where $t/|\mathcal{X}|$ is considerably small, such that a randomly sampled x is only in $\mathcal{S}$ with negligible probability. More generally, for $\ell = \ell(\lambda)$, weak pseudorandomness is not achievable for values t with $t/|\mathcal{X}| = 1/\mathsf{poly}(\lambda)$.

Definition 9 (Key-Injectivity)**.** An xPRF is key-injective if for $\ell = \mathsf{poly}(\lambda)$,

$$\Pr\left[\begin{array}{l}\exists k, k' \in \mathcal{K}_\lambda \text{ s.t. } k \neq k'; x \in \mathcal{X}_{\lambda,\ell} : \\ \quad \mathsf{eval}(pp, k, x) = \mathsf{eval}(pp, k', x))\end{array} \middle| \; (pp, td) \leftarrow\$ \mathsf{setup}(1^\lambda, 1^\ell)\right] = \mathsf{negl}(\lambda).$$

Definition 10 (Puncturable xPRFs)**.** An xPRF family $\mathcal{F}$ is puncturable if there additionally exist the following deterministic, polynomial-time algorithms

- $k_{\mathcal{S}} \leftarrow \mathsf{punct}(pp, k, \mathcal{S})$: The puncturing algorithm takes the public parameters pp, a key $k \in \mathcal{K}_\lambda$, and a set $\mathcal{S} \subseteq \mathcal{X}_{\lambda,\ell}$ and outputs a punctured key $k_{\mathcal{S}}$.
- $y \leftarrow \mathsf{punctEval}(pp, k_{\mathcal{S}}, x)$: The punctured evaluation algorithm takes the public parameters pp, the punctured key $k_{\mathcal{S}}$, and a value $x \in \mathcal{X}_{\lambda,\ell}$, and outputs $y \in \mathcal{Y}_{\lambda,\ell}$.

such that for all $\lambda, \ell \in \mathbb{N}$, $(pp, td) \in \mathsf{setup}(1^\lambda, 1^\ell)$, $k \in \mathcal{K}_\lambda$, $\mathcal{S} \subseteq \mathcal{X}_{\lambda,\ell}$, $x \in \mathcal{X}_{\lambda,\ell} \setminus \mathcal{S}$, $k_{\mathcal{S}} \in \mathsf{punct}(pp, k, \mathcal{S})$, it holds that $\mathsf{eval}(pp, k, x) = \mathsf{punctEval}(pp, k_{\mathcal{S}}, x)$.

$\underline{\texttt{Punct-xPrf}^b_{\mathcal{F}}}$

$\mathsf{setupO}(\mathcal{S})$	$\mathsf{evalO}(x)$
assert $pp = \bot$	**assert** $pp \neq \bot \wedge x \in \mathcal{S}$
$(pp, td) \leftarrow\!\$\ \mathcal{F}.\mathsf{setup}(1^\lambda, 1^\ell)$	**if** $b = 0$ **then** $y \leftarrow \mathcal{F}.\mathsf{eval}(pp, k, x)$
$k \leftarrow\!\$\ \mathcal{K}_\lambda,\ k_{\mathcal{S}} \leftarrow \mathsf{punct}(pp, k, \mathcal{S})$	**if** $b = 1$ **then**
return $pp, k_{\mathcal{S}}$	**if** $T[x] = \bot$ **then** $T[x] \leftarrow\!\$\ \mathcal{Y}_{\lambda,\ell}$
	$y \leftarrow T[x]$
	return y

Figure 2: xPRF puncturing security.

Definition 11 (Puncturing Security)**.** An xPRF family $\mathcal{F}$ is puncturably secure if for any $\ell = \mathsf{poly}(\lambda)$ and any PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{Punct-xPrf}}_{\mathcal{F},\mathcal{A}}(\lambda) := \left| \Pr\left[1 = \mathcal{A} \overset{\substack{\mathsf{setupO}\\ \mathsf{evalO}}}{\longrightarrow} \texttt{Punct-xPrf}^0_{\mathcal{F}}\right] - \Pr\left[1 = \mathcal{A} \overset{\substack{\mathsf{setupO}\\ \mathsf{evalO}}}{\longrightarrow} \texttt{Punct-xPrf}^1_{\mathcal{F}}\right] \right|$$

is negligible, where Figure 2 defines $\texttt{Punct-xPrf}^b_{\mathcal{F}}$ for $b \in \{0, 1\}$.

We define special families of xPRFs which vary depending on their functionality and security properties.

Definition 12 (Watermarkable PRF)**.** An xPRF family $\mathcal{F}$ is called a *watermarkable* PRF if it is 1. extractable (Definition 8), 2. puncturable (Definition 10), 3. pseudorandom (Definition 7), and 4. puncturably secure (Definition 11).

3.3 Robust Extractability

The compiler of Kim and Wu [KW19] from watermarkable to watermarking PRFs requires the former to satisfy *robust extractability*, i.e., given a program (represented by a circuit) which approximates the PRF with a hardwired key, the trapdoored extraction algorithm should be able to extract the hardwired key. The following defines a measure of how well programs approximate each other.

Definition 13 (Far and Close Programs)**.** For two programs C and C' with domain $\mathcal{X}$, we denote by $\mathsf{SD}(C, C') := \Pr_{x \leftarrow\!\$ \mathcal{X}}[C(x) \neq C'(x)]$ the *distance* between two programs C and C'. C and C' are δ-far if $\mathsf{SD}(C, C') \geq \delta$. C and C' are ϵ-close if $\mathsf{SD}(C, C') \leq \epsilon$.

To define and prove robust extractability, Kim and Wu [KW19, Remark 4.10] uses notions that they call "Extract-and-Test" and "Generalized Candidate Testing", i.e. given the public parameters pp (but not the trapdoor td), a circuit C, and a PRF key k, one can test whether C is close to $\mathsf{eval}(pp, k, .)$. We summarise these notions into a "public testability" property where the only differences are syntactical.

Definition 14 (Public Testability)**.** Let $0 \leq \epsilon < \delta \leq 1$. Family $\mathcal{F}$ is publicly (ϵ, δ)-testable if there exists a PPT algorithm test such that for all $\lambda, \ell \in \mathbb{N}$, $(pp, td) \in \mathsf{setup}(1^\lambda, 1^\ell)$, $k \in \mathcal{K}_\lambda$, and all programs C with domain $\mathcal{X}_\lambda$:

Close programs. $\mathsf{SD}(C, \mathsf{eval}(pp, k, .)) \leq \epsilon \implies \Pr[\mathsf{test}(pp, k, C)] = 1 - \mathsf{negl}(\lambda)$.

Far programs. $\mathsf{SD}(C, \mathsf{eval}(pp, k, .)) \geq \delta \implies \Pr[\mathsf{test}(pp, k, C)] = \mathsf{negl}(\lambda)$.

$\mathcal{F}.\mathsf{Extr}(pp, td, C)$

$(x_1, \ldots, x_t) \leftarrow \mathsf{S}(td)$
for $i \in [t]$ **do**
  $k_i \leftarrow \mathcal{F}.\mathsf{inv}(td, x_i, C(x_i))$
  if $\mathsf{Test}_{\epsilon,\delta}(pp, k_i, C) = 1$
    return k_i
return $\bot$

$\mathcal{F}.\mathsf{Test}_{\epsilon,\delta}(pp, k, C)$

$\text{thld} \leftarrow \frac{\epsilon+\delta}{2}$, $\text{bnd} \leftarrow \lambda^2 \cdot (\frac{1}{\text{thld}-\epsilon})^2$, $\text{ctr} \leftarrow \text{bnd}$
for $i \in [\text{bnd}]$ **do**
  $x_i \leftarrow\$ \mathcal{X}$
  if $C(x_i) = \mathcal{F}.\mathsf{eval}(pp, k, x_i)$ **then** $\text{ctr} \leftarrow \text{ctr} - 1$
if $\frac{\text{ctr}}{\text{bnd}} \leq \text{thld}$ **then return** 1
else return 0

Figure 3: Construction of the $\mathsf{Test}_{\epsilon,\delta}$ and Extr algorithms.

$\underline{\underline{\mathtt{REXT}^b_{\mathcal{F}}}}$

$\mathsf{setupO}()$

$(pp, td) \leftarrow\$ \mathsf{setup}(1^\lambda)$
return pp

$\mathsf{extrO}(C)$

if $b = 0$ **then return** $\mathsf{extr}(td, C)$
if $\exists!\, k :\ \mathsf{SD}(C, \mathsf{eval}(pp, k, .)) \leq \delta \wedge \mathsf{test}(pp, k, C) = 1$ **then**
  return k
return $\bot$

Figure 4: Robust extractability of xPRFs.

In Figure 3, we construct $\mathsf{Test}_{\epsilon,\delta}$ and Extr algorithms for any xPRF family $\mathcal{F}$.

Lemma 2 (Public Testability)**.** *For any xPRF family $\mathcal{F}$, any $0 \leq \epsilon < \delta \leq 1$, $\mathcal{F}$ is publicly (ϵ, δ)-testable.*

Proof. We show that the $\mathcal{F}.\mathsf{Test}_{\epsilon,\delta}$ algorithm constructed in Figure 3 satisfies the criteria for (ϵ, δ)-public testability (Definition 14). $\mathcal{F}.\mathsf{Test}_{\epsilon,\delta}$ used the fraction $\frac{\text{ctr}}{\text{bnd}}$ as an approximation of the distance of program C to $\mathcal{F}.\mathsf{eval}(pp, k, \cdot)$. If the two programs are ϵ-close, then each iteration of the **for** loop has independent probability $\leq \epsilon$ of succeeding to substract 1 from ctr. Hence, the sum has $\mathbb{E}\left[\frac{\text{ctr}}{\text{bnd}}\right] \leq \epsilon$. Using a Chernoff bound, the probability that $\frac{\text{ctr}}{\text{bnd}}$ is greater than $\frac{\epsilon+\delta}{2} = \epsilon + \frac{\delta-\epsilon}{2}$ is negligible and $\mathcal{F}.\mathsf{Test}_{\epsilon,\delta}$ returns 1 with overwhelming probability. Analogously, when C is δ-far from $\mathcal{F}.\mathsf{eval}(pp, k, \cdot)$, then a Chernoff bound shows that the probability that $\mathcal{F}.\mathsf{Test}_{\epsilon,\delta}$ returns 1 is negligible. □

Definition 15 (Robust Extractability)**.** Let $\mathtt{REXT}^b_{\mathcal{F}}$ be the experiment in Figure 4. Family $\mathcal{F}$ has robust extractability if $\mathcal{F}$ is public (ϵ, δ)-testable for some $0 \leq \epsilon < \delta \leq 1$ with some PPT algorithm $\mathsf{test}_{\epsilon,\delta}$ and there exists a PPT extr algorithm such that for any PPT $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{REXT}}_{\mathcal{F},\mathcal{A},\mathsf{extr},\mathsf{test}}(1^\lambda) := \left| \Pr\left[1 = \mathcal{A} \overset{\substack{\mathsf{setupO}\\ \mathsf{extrO}}}{\longrightarrow} \mathtt{REXT}^0_{\mathcal{F}}\right] - \Pr\left[1 = \mathcal{A} \overset{\substack{\mathsf{setupO}\\ \mathsf{extrO}}}{\longrightarrow} \mathtt{REXT}^1_{\mathcal{F}}\right] \right| = \mathsf{negl}(\lambda).$$

Next, we show that any extractable xPRF must also have robust extractability. In Appendix A, we handle a slightly more complicated case where the xPRF is only t-extractable for $t < |\mathcal{X}|$ but not extractable.

Lemma 3. *Let $\mathcal{F}$ be a xPRF family. If $\mathcal{F}$ is extractable (Definition 8), then it has robust extractability (Definition 15).*

Proof. By Lemma 2, $\mathcal{F}$ is (ϵ, δ)-publicly testable with the algorithm $\mathcal{F}.\mathsf{Test}_{\epsilon,\delta}$ constructed in Figure 3. The proof idea is implicit in the proof of [KW19, Theorem 4.26]. The proof proceeds via a sequence of games, described below:

- Game 0: The real robust extractability game $\mathtt{RExt}^0_{\mathcal{F}}$.
- Game 1: extrO queries are now answered using the following inefficient algorithm $\mathsf{IneffExtr}$: Evaluate C on all $x_i \in \mathcal{S}$ to obtain pairs (x_i, y_i). For each of these pairs, check whether there is a unique k_i such that $\mathcal{F}.\mathsf{eval}(pp, k_i, x_i) = y_i$. If yes, then run $\mathsf{Test}_{\epsilon,\delta}(pp, k_i, C)$, and if $\mathsf{Test}_{\epsilon,\delta}$ outputs 1, return k_i.
- Game 2: extrO now performs the ideal test of whether the program is δ-close to $\mathcal{F}.\mathsf{eval}(pp, k, .)$. This game is precisely $\mathtt{RExt}^1_{\mathcal{F}}$.

Below, we prove the computational indistinguishability of the games.

Game 0 to 1: We prove that the two games are statistically indistinguishable. First, we note that extractability implies key-injectivity. Indeed, assume that $y = \mathsf{eval}(pp, k, x) = \mathsf{eval}(pp, k', x)$, then by extractability, $k = \mathsf{inv}(td, x, y) = k'$.

By key-injectivity and by the use of identical extraction points, with overwhelming probability, the inefficient extraction algorithm considers the same set of candidate keys as $\mathcal{F}.\mathsf{extr}$. Assuming this is the case, then both algorithms run $\mathsf{Test}_{\epsilon,\delta}$ which succeeds/fails with the same probability in both games.

Game 1 to 2: Using a Chernoff bound, testing whether the program C is δ-close to $\mathcal{F}.\mathsf{eval}(pp, k, .)$ can be reasonably closely approximated by extracting from C on a polynomial number of random points and testing the resulting keys. We use the two ways of testing closeness equivalently in the following.

This proof is a hybrid argument over the number of queries which the adversary makes. Before the first extraction query of the adversary, the adversary has information-theoretically no information about the extraction points $\mathcal{S}$ which extrO uses. When the adversary submits a close program C, it will be tested on uniformly random values (in the view of the adversary) chosen from a spuer-polynomially-large space and thus, when it is δ-close to $\mathcal{F}.\mathsf{eval}(pp, k, .)$, the oracle will return k with overwhelming probability in Game 2, and we can replace the extraction oracle with an ideal oracle which does not sample extraction points. Thus, the answer to the first extraction query does not leak any information to the adversary about the extraction points, and now, the argument can be applied to the second query and so on. □

4 Extractable PRG Constructions

We now construct extractable PRGs (xPRGS) from the computational Diffie-Hellman (CDH) assumption in the random oracle model (Section 4.2), from the learning with rounding (LWR) assumption (Section 4.3), and from trapdoor permutations (Section 4.4). All three constructions have an arbitrary polynomial expansion factor $e = e(\lambda)$, i.e. provide an output which consists of $e(\lambda)$ symbols. Note that the input alphabet $\mathcal{X}_{\lambda,\ell}$ and output alphabet $\mathcal{Y}_{\lambda,\ell}$ might be different. Since we want to compose extractable PRGs recursively in a GGM tree, $\mathcal{X}_{\lambda,\ell}$ and $\mathcal{Y}_{\lambda,\ell}$ need to be over the same alphabet, say $\mathcal{X}_{\lambda,\ell} = \Sigma_\lambda^\ell$ and $\mathcal{Y}_{\lambda,\ell} = \Sigma_\lambda^{\ell'}$, and $\log \mathcal{Y}_{\lambda,\ell}$ is bigger than $\log \mathcal{X}_{\lambda,\ell}$ at most by an *additive* factor, since else, the output length would increase exponentially with the depth of the GGM tree.

4.1 Trapdoor Functions and Permutations

We recall the standard definition of trapdoor functions and permutations.

Definition 16 (Trapdoor Functions and Permutations)**.** A trapdoor function (TDF) family $\mathcal{T}$ with domains $(\mathcal{X}_{\lambda,\ell})_{\lambda,\ell\in\mathbb{N}}$ and codomains $(\mathcal{Y}_{\lambda,\ell})_{\lambda,\ell\in\mathbb{N}}$ consists of three PPT algorithms $(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ with the following syntax:

$\mathcal{G}.\mathsf{setup}(1^\lambda, 1^\ell)$	$\mathcal{G}.\mathsf{eval}(pp, x)$	$\mathcal{G}.\mathsf{inv}(td, i, y)$
for $i \in \mathbb{Z}_e$ **do**	$(pp_0, \ldots, pp_{e-1}) \leftarrow pp$	$(td_0, \ldots, td_{e-1}) \leftarrow td$
$\quad (pp_i, td_i) \leftarrow\!\!\$\ \mathcal{T}.\mathsf{setup}(1^\lambda, 1^\ell)$	**for** $i \in \mathbb{Z}_e$ **do**	$x \leftarrow \mathcal{T}.\mathsf{inv}(td_i, y)$
$pp \leftarrow (pp_0, \ldots, pp_{e-1})$	$\quad y_i \leftarrow \mathcal{T}.\mathsf{eval}(pp_i, x)$	**return** x
$td \leftarrow (td_0, \ldots, td_{e-1})$	**return** $(y_0, \ldots, y_{e-1})$	
return (pp, td)		

Figure 5: Construction of xPRGs from TDFs.

- $(pp, td) \leftarrow\!\!\$\ \mathsf{setup}(1^\lambda, 1^\ell)$: The setup algorithm inputs 1^λ and length parameter 1^ℓ and outputs the public parameters pp and a trapdoor td.
- $y \leftarrow \mathsf{eval}(pp, x)$: The deterministic evaluation algorithm inputs the public parameters pp and a preimage $x \in \mathcal{X}_{\lambda,\ell}$ and outputs an image $y \in \mathcal{Y}_{\lambda,\ell}$.
- $x \leftarrow \mathsf{inv}(td, y)$: The deterministic inversion algorithm inputs a trapdoor td and an image $y \in \mathcal{Y}_{\lambda,\ell}$ and outputs a preimage $x \in \mathcal{X}_{\lambda,\ell}$.

The family $\mathcal{T}$ is said to be correct if for any $\lambda, \ell \in \mathbb{N}$, $(pp, td) \in \mathsf{setup}(1^\lambda, 1^\ell)$, $x \in \mathcal{X}_{\lambda,\ell}$, it holds that $\mathsf{inv}(td, \mathsf{eval}(pp, x)) = x$. The family $\mathcal{T}$ is said to be a trapdoor permutation (TDP) family if $|\mathcal{X}_{\lambda,\ell}| = |\mathcal{Y}_{\lambda,\ell}|$ for all $\lambda, \ell \in \mathbb{N}$. The rate $r(\lambda)$ of the TDF is defined as $r(\lambda) := \lim_{\ell\to\infty} \log|\mathcal{X}_{\lambda,\ell}| / \log|\mathcal{Y}_{\lambda,\ell}|$.

Figure 5 constructs a simple xPRG by concatenating multiple instances of a TDF family. The following functionality and efficiency properties are immediate.

Proposition 1. *Let $e \in \mathbb{N}$. The xPRG family $\mathcal{G}$ constructed in Figure 5 has expansion factor e. Furthermore, if the TDF family $\mathcal{T}$ is correct, then $\mathcal{G}$ is extractable.*

Definition 17 (e-Correlated Pseudorandomness)**.** Let $e \in \mathbb{N}$. The TDF family $\mathcal{T}$ is said to be e-correlated-pseudorandom if the xPRG family $\mathcal{G}$ constructed in Figure 5 is pseudorandom. We call a 1-correlated-pseudorandom TDF a *trapdoor PRG*.

4.2 xPRG from CDH

An e-correlated pseudorandom trapdoor function is trivially an extractable PRG, cf. Figure 5 and Definition 17. Thus, all we need to construct is a CDH-based e-correlated pseudorandom trapdoor function, which is *rate-1*, i.e., referring to the property that $\log \mathcal{Y}_{\lambda,\ell}$ is bigger than $\log \mathcal{X}_{\lambda,\ell}$ at most by an *additive constant*.

Döttling et al. [DGH+19] gave a simple construction of TDFs over a prime-order cyclic group $\mathbb{G}$. In a nutshell, the TDF of Döttling et al. inputs $x \in \{0,1\}^n$ and outputs $[u] \leftarrow \langle \mathbf{x}, [\mathbf{g}] \rangle$ and $v_i \leftarrow \mathsf{bl}(\langle \mathbf{x}, [\mathbf{h}_i] \rangle) \oplus x_i$ for $i \in [n]$, where $[\mathbf{g}]$ and $[\mathbf{h}_i]$ are public vectors of $\mathbb{G}$ elements and bl is a randomness extractor which outputs a single bit. In other words, an output of the TDF is longer than its input by a single $\mathbb{G}$ element. Since $\log|\mathbb{G}| = \mathsf{poly}(\lambda)$ is a fixed polynomial in λ independent of n, the TDF is rate-1. It is also not difficult to prove that their TDF family is in fact e-correlated pseudorandom for any $e = \mathsf{poly}(\lambda)$.

Unfortunately, since we later compose extractable PRGs recursively in GGM style to obtain extractable PRFs, the mismatching domains and codomains of the TDFs of Döttling et al. constitute an obstacle. Instead, we would like the domains and codomains of a TDF to fall into the same family. Our natural strategy is to modify the domains and codomains of the TDFs so that they both contain tuples of $\mathbb{H}$ elements for some group $\mathbb{H}$, e.g. the binary field $\{0,1\}$, any finite field $\mathbb{F}$, or the group $\mathbb{G}$ itself. Executing this

$\mathcal{G}.\mathsf{setup}(1^\lambda, 1^\ell)$	$\mathcal{G}.\mathsf{eval}(pp, [\mathbf{x}] \in \mathbb{G}^\ell)$	$\mathcal{G}.\mathsf{inv}(td, ([u], [\mathbf{v}]) \in \mathbb{G}^{\ell+1})$
$td \leftarrow r \leftarrow\!\!\$\ \mathbb{Z}_q$	$([g], [h], h_1, h_2) \leftarrow pp$	$r \leftarrow td$
$[g] \leftarrow\!\!\$\ \mathbb{G},\ [h] \leftarrow r \cdot [g]$	$s \leftarrow h_1([\mathbf{x}])$	$[\mathbf{w}] \leftarrow [\mathbf{v}] - h_2(r \cdot [u])$
$h_1 \leftarrow\!\!\$\ \mathcal{H}_{1,\lambda,\ell}$	$[u] \leftarrow [g] \cdot s$	**return** $[\mathbf{w}]$
$h_2 \leftarrow\!\!\$\ \mathcal{H}_{2,\lambda,\ell}$	$[\mathbf{v}] \leftarrow h_2([h] \cdot s) + [\mathbf{x}]$	
$pp \leftarrow ([g], [h], h_1, h_2)$	**return** $([u], [\mathbf{v}])$	
return (pp, td)		

Figure 6: Construction of rate-1 trapdoor functions from CDH in the ROM.

strategy requires two changes to the construction. The first, simpler, change is to replace the randomness extractor bl with one that outputs an $\mathbb{H}$ element, in case $\mathbb{H} \neq \{0,1\}$. The second step, in case $\mathbb{H} \neq \mathbb{G}$, is to encode a $\mathbb{G}$ element by a k-tuple of $\mathbb{H}$ elements for some $k \in \mathbb{N}$, such that the set of encodings is of size negligibly close to $|\mathbb{H}|^k$. Although there exist methods to encode $\mathbb{G}$ elements for *some* choice of $\mathbb{G}$, e.g. [BHKL13], they often only encode a (noticeable) subset of $\mathbb{G}$ and the number of encodings is practically but not negligibly close to $|\mathbb{H}|^k$.[8] Using a random oracle is, in principle, a good approach for an encoding, but we need the encoding to be *invertible*.

Facing the above, we instead set $\mathbb{H} := \mathbb{G}$ and let $\mathcal{H}_1 = (\mathcal{H}_{1,\lambda,\ell})_{\lambda,\ell\in\mathbb{N}}$ be a family of hash functions (modelled as a random oracle) where each $h_1 \in \mathcal{H}_{1,\lambda,\ell}$ maps $\mathbb{G}^\ell$ to $\mathbb{Z}_q$, where $(\mathbb{G}, q, [1]) \in \mathsf{ggen}(1^\lambda)$. Additionally, to break the circular dependency of the secrecy of an input $[\mathbf{x}]$ and the randomness $s \leftarrow h_1([\mathbf{x}])$ used to mask $[\mathbf{x}]$, let $\mathcal{H}_2 = (\mathcal{H}_{2,\lambda,\ell})$ be a family of hash functions (modelled as well as a random oracle) where each $h_2 \in \mathcal{H}_{2,\lambda,\ell,\mathbb{G}}$ maps $\mathbb{G}$ to $\mathbb{G}^\ell$. Our modified construction is shown in Figure 6.

Remark 2. The TDF construction in Figure 6 can be seen as the deterministic encryption (DE) construction of [BBO07] where the public-key encryption scheme is instantiated with the ElGamal encryption scheme and the random oracle is replaced by a universal hash function. Alternatively, we can view it as a variant of the DE scheme of [BFO08], where the message space $\{0,1\}^\ell$ is replaced with $\mathbb{G}^\ell$ and the specific universal hash function mapping $\{0,1\}^\ell$ to $\mathbb{Z}_q$ is replaced with a general one which maps $\mathbb{G}^\ell$ to $\mathbb{Z}_q$. The security proof technique of their DE construction in the standard model relies on the domain being $\{0,1\}^\ell \subseteq \mathbb{Z}_q$ and fails when the domain is replaced by $\mathbb{G}^\ell$.

Theorem 1. *Let $e = \mathsf{poly}(\lambda)$ and $(\mathbb{G}, q, [1]) \leftarrow\!\!\$\ \mathsf{ggen}(1^\lambda)$. If the CDH assumption holds relative to $(\mathbb{G}, q, [1])$, and the hash function families $\mathcal{H}_1$ and $\mathcal{H}_2$ are modeled as random oracles, then the TDF family constructed in Figure 6 is e-correlated pseudorandom.*

Proof. Let $\mathcal{G}$ denote the xPRG constructed in Figure 5 using the trapdoor function from Figure 6. Suppose that $\mathcal{A}$ makes at most $Q = \mathsf{poly}(\lambda)$ queries to the evalO oracle. We will index the queries to evalO by $i \in [Q]$. For $j \in [e]$, let $([g_j], [h_j], h_{j,1}, h_{j,2})$ denote the public parameters of the j-th instance of the TDF. We write evalO_j for the part of evalO which computes the j-th output component. We reduce the security of the xPRG from CDH by a two-layer hybrid argument. For $i \in [Q]$, we define the first layer of hybrid experiments as follows:

- Hybrid Hyb_0: Identical to the $\mathtt{Prg}^0_{\mathcal{G}}$ experiment.
- Hybrid Hyb_i: Identical to Hyb_{i-1}, except that evalO answers the i-th query by returning random samples from $\mathbb{G}^{\ell+1}$ for all e branches.

[8] There also exist methods, e.g. [BF01], which only work for very specific choices of $\mathbb{G}$ and achieve negligible closeness.

- Hybrid Hyb_Q: Identical to the $\mathtt{Prg}^1_{\mathcal{G}}$ experiment.

It suffices to prove that, for each $i \in [Q]$, the hybrids Hyb_{i-1} and Hyb_i are computationally indistinguishable. To show this, we define the second-layer hybrid experiments:

- Hybrid $\mathsf{Hyb}_{i,0}$: Identical to Hyb_{i-1}. In other words, on the first $i-1$ queries, evalO outputs a random sample from $\mathbb{G}^{\ell+1}$ for each branch. The remaining queries are computed honestly.

  In particular, on the i-th query, evalO proceeds as follows: It samples $[\mathbf{x}] \leftarrow\!\!\$\ \mathbb{G}^\ell$ and then, for each $j \in [e]$, it fetches $s_j \leftarrow h_{j,1}([\mathbf{x}])$ if $h_{j,1}([\mathbf{x}])$ is already programmed, and otherwise samples $s_j \leftarrow\!\!\$\ \mathbb{Z}_q$ and program $h_{j,1}([\mathbf{x}]) := s_j$. Also, it fetches $[\mathbf{t}_j] \leftarrow h_{j,2}([h_j]\cdot s_j)$ if $h_{j,2}([h_j]\cdot s_j)$ is already programmed, and otherwise samples $[\mathbf{t}_j] \leftarrow\!\!\$\ \mathbb{G}^\ell$ and programs $h_{j,2}([h_j]\cdot s_j) := [\mathbf{t}_j]$. Finally, evalO outputs $([g_i]\cdot s_j, [\mathbf{t}_j] + [\mathbf{x}])_{j\in[e]}$.
- Hybrid $\mathsf{Hyb}_{i,1}$: Identical to $\mathsf{Hyb}_{i,0}$, except that evalO implements the following additional abortion logic: For any $j \in [e]$, if $h_{j,1}([\mathbf{x}])$ is already programmed or if $h_{j,2}([h_j]\cdot s_j)$ is already programmed, abort.
- Hybrid $\mathsf{Hyb}_{i,2}$: Identical to $\mathsf{Hyb}_{i,1}$, except that the experiment aborts if, at any time and for any $j \in [e]$, random oracle $h_{j,2}$ is queried on $[h_j]\cdot s_j$.
- Hybrid $\mathsf{Hyb}_{i,3}$: Identical to $\mathsf{Hyb}_{i,2}$, except that the experiment aborts if, at any time and for any $j \in [e]$, random oracle $h_{j,1}$ is queried on $[\mathbf{x}]$.
- Hybrid $\mathsf{Hyb}_{i,4}$: Identical to $\mathsf{Hyb}_{i,3}$, except that evalO answers the i-th query by outputting a random sample of $\mathbb{G}^{\ell+1}$ on all branches.
- Hybrid $\mathsf{Hyb}_{i,5}$: Identical to $\mathsf{Hyb}_{i,4}$, except that all abort conditions are dropped. In other words, this hybrid is identical to Hyb_i.

We show that the above second-layer hybrids are computationally indistinguishable.

$\mathsf{Hyb}_{i,0} \approx_s \mathsf{Hyb}_{i,1}$. We show that $\mathsf{Hyb}_{i,0}$ and $\mathsf{Hyb}_{i,1}$ are statistically close. First, we observe that, conditioned on the event that, before the i-th query to evalO, $[\mathbf{x}]$ is never queried to the random oracle $h_{j,1}$ and $[h_j]\cdot s_j$ is never queried to the random oracle $h_{j,2}$ for any $j \in [e]$, the adversary's views in $\mathsf{Hyb}_{i,0}$ and $\mathsf{Hyb}_{i,1}$ are identical. However, before the i-th query to evalO, the adversary's views in $\mathsf{Hyb}_{i,0}$ and $\mathsf{Hyb}_{i,1}$ are independent of $[\mathbf{x}]$ and $(s_j)_{j\in[e]}$, and thus the probability that the adversary queried $h_{j,1}$ on $[\mathbf{x}]$ or $h_{j,2}$ on $[h_j]\cdot s_j$ for any $j \in [e]$ before the j-th query to evalO is at most $e \cdot (\mathsf{poly}(\lambda)/|\mathbb{G}|^\ell + \mathsf{poly}(\lambda)/q) \leq \mathsf{negl}(\lambda)$, where the $\mathsf{poly}(\lambda)$ factors hide the polynomial number of queries that the adversary makes.

$\mathsf{Hyb}_{i,1} \approx_c \mathsf{Hyb}_{i,2}$. We show that $\mathsf{Hyb}_{i,1}$ and $\mathsf{Hyb}_{i,2}$ are computationally indistinguishable based on the CDH assumption. More precisely, we will apply the CDH assumption e times, once per each $j \in [e]$. First, we observe that, before $h_{j,2}$ is queried on $[h_j]\cdot s_j$, the vector $[\mathbf{x}]$ is information-theoretically hidden from the view of the adversary, and therefore the probability of the adversary querying $h_{j,1}$ on $[\mathbf{x}]$ before querying $h_{j,2}$ on $[h_j]\cdot s_j$ is at most $\mathsf{poly}(\lambda)/|\mathbb{G}|^\ell \leq \mathsf{negl}(\lambda)$.

Now, suppose that the adversary queries $h_{j,2}$ on $[h_j]\cdot s_j$ before querying $h_{j,1}$ on $[\mathbf{x}]$ with non-negligible probability. We turn this adversary into a CDH solver. On input a CDH instance $(\mathbb{G}, q, [1], [g_i], [h_j], [g_i]\cdot s_j)$, simulate the hybrid experiment $\mathsf{Hyb}_{i,j,1}$ or $\mathsf{Hyb}_{i,j,2}$ until either $h_{j,1}$ is queried on $[\mathbf{x}]$ or the experiment returns, whichever is sooner. Output a random query which was submitted to the $h_{j,2}$ oracle.

Since the adversary queries $h_{j,2}$ on $[h_j]\cdot s_j$ before querying $h_{j,1}$ on $[\mathbf{x}_i]$ with non-negligible probability, one of the $h_{j,2}$ oracle queries is the CDH solution $[h_j]\cdot s_j$ with non-negligible probability, and conditioned on that the CDH solver picks this as its output with probability at least $1/\mathsf{poly}(\lambda)$.

$\mathcal{G}.\mathsf{setup}(1^\lambda, 1^\ell)$	$\mathcal{G}.\mathsf{eval}(pp, (\mathbf{r}, \mathbf{s}) \in \mathbb{Z}_p^{n+\ell})$	$\mathcal{G}.\mathsf{inv}(td, (\mathbf{u}, \mathbf{v}) \in \mathbb{Z}_p^{m+\ell})$
$(\mathbf{A}, td) \leftarrow\!\!\$\ \mathsf{TrapGen}(1^n, 1^m, p, q)$	$(\mathbf{A}, \mathbf{B}) \leftarrow pp$	$\mathbf{r} \leftarrow \mathsf{LWEInvert}(td, \mathbf{u})$
$\mathbf{B} \leftarrow\!\!\$\ \mathbb{Z}_q^{n \times \ell}$	$\mathbf{u} \leftarrow \lfloor \mathbf{r} \cdot \mathbf{A} \rceil_p$	$\mathbf{s} \leftarrow \mathbf{v} - \lfloor \mathbf{r} \cdot \mathbf{B} \rceil_p \bmod p$
$pp \leftarrow (\mathbf{A}, \mathbf{B})$	$\mathbf{v} \leftarrow \lfloor \mathbf{r} \cdot \mathbf{B} \rceil_p + \mathbf{s} \bmod p$	**return** $(\mathbf{r}, \mathbf{s})$
return (pp, td)	**return** $(\mathbf{u}, \mathbf{v})$	

Figure 7: Construction of an e-correlated, pseudorandom trapdoor functions from LWE. See Lemma 1 for the algorithm $\mathsf{TrapGen}$.

$\mathsf{Hyb}_{i,2} \approx_s \mathsf{Hyb}_{i,3}$**.** We show that $\mathsf{Hyb}_{i,2}$ and $\mathsf{Hyb}_{i,3}$ are statistically close, by an analogous argument as before. To be precise, we realise that in these two hybrids the vector $[\mathbf{x}]$ is information-theoretically hidden from the view of the adversary. Therefore, the probability of the adversary querying $h_{j,1}$ on $[\mathbf{x}]$ for any $j \in [e]$ is at most $e \cdot (\mathsf{poly}(\lambda)/|\mathbb{G}|^\ell) \leq \mathsf{negl}(\lambda)$.

$\mathsf{Hyb}_{i,3} \equiv \mathsf{Hyb}_{i,4}$**.** These two hybrid games are functionally equivalent. Indeed, since $[\mathbf{x}]$ and $(s_j)_{j\in[e]}$ are information-theoretically hidden from the adversary, all components of $([g_i] \cdot s_j, [\mathbf{t}_j] + [\mathbf{x}])_{j\in[e]}$ are independently and uniformly distributed.

$\mathsf{Hyb}_{i,4} \equiv \mathsf{Hyb}_{i,5}$**.** These two hybrid games are statistically close. Indeed, since $[\mathbf{x}]$ and $(s_j)_{j\in[e]}$ are information-theoretically hidden from the adversary, the probability that the adversary queries random oracle $h_{j,1}$ on $[\mathbf{x}]$ or random oracle $h_{j,2}$ on $[h_j] \cdot s_j$ for any $j \in [e]$ is at most $e \cdot (\mathsf{poly}(\lambda)/|\mathbb{G}|^\ell + \mathsf{poly}(\lambda)/q) \leq \mathsf{negl}(\lambda)$. □

4.3 xPRG from LWR

We now provide an analogous lattice-based e-correlated pseudorandom TDF family in Figure 7. Our main idea is that an LWR function $\mathbf{r} \mapsto \lfloor \mathbf{r}\mathbf{A} \rceil_p$ itself is already a trapdoor function, and that the following extension yields a new, related trapdoor function with the desired properties: Suppose $f(x)$ is a trapdoor function, then we build another trapdoor function $f'(x, x')$ which first computes $f(x)$ and then uses x as a secret key to encrypt x'.

Theorem 2. *Let $e = \mathsf{poly}(\lambda)$. If the LWR assumption holds for $(n, e \cdot (m + \ell), p, q, \mathbb{Z}_p)$, then the TDF $\mathcal{T}$ constructed in Figure 7 is e-correlated pseudorandom.*

Proof. Let $\mathcal{G}$ denote the xPRG constructed in Figure 5 using the TDF from Figure 7. We proceed by a hybrid argument over the Q queries which $\mathcal{A}$ makes.

- Hybrid 1: Replace $(\mathbf{A}_i)_{i\in\mathbb{Z}_e}$ in pp by uniformly random matrices in $\mathbb{Z}_q^{n\times m}$.
- Hybrid $(2, k)$ for $k \leq [Q]$: For the first k queries to evalO, replace $(\lfloor \mathbf{r}_k \cdot \mathbf{A}_i \rceil_p, (\lfloor \mathbf{r}_k \cdot \mathbf{B}_i \rceil_p))_{i\in\mathbb{Z}_e}$ by a sample from $U(\mathbb{Z}_p^{e\cdot(m+\ell)})$, where $\mathbf{r}_k$ is part of the PRG seed $x_k = (\mathbf{r}_k, \mathbf{s}_k)$ sampled for the k-th query.

By Lemma 1, $\mathtt{Prg}_{\mathcal{G}}^0$ and Hybrid 1 are statistically close in λ. Observe that Hybrid 1 is equal to Hybrid $(2, 0)$. For each $k \in [Q]$, Hybrid $(2, k-1)$ and Hybrid $(2, k)$ are computationally indistinguishable due to the LWR assumption. Finally, Hybrid $(2, Q)$ is identical to the $\mathtt{Prg}_{\mathcal{G}}^1$ experiment. □

4.3.1 On (not) using LWE instead of LWR for our xPRGs.

As mentioned earlier, since the LWR assumption is implied by the LWE assumption (e.g. [BGM+16]), the TDF constructed in Figure 7 is ultimately based on LWE (and so are the xPRGs we obtain from the TDF).

Since the LWE function $(\mathbf{r}, \mathbf{e}) \mapsto \mathbf{rA} + \mathbf{e} \bmod q$ is also a trapdoor function and is pseudorandom (under the LWE assumption), a natural question is why we need to specifically use LWR in Figure 7. The answer to this is that, if the LWR function is replaced by the LWE function in Figure 7, then the resulting TDF is not e-correlated pseudorandom for sufficiently large e.

Indeed, plugging the resulting TDF into the xPRG construction in Figure 5, an xPRG output consists of LWE samples $(\mathbf{rA}_i + \mathbf{e} \bmod q)_{i=1}^{e}$ with the same secret $\mathbf{r}$ and error $\mathbf{e}$ and different matrices $\mathbf{A}_1, \ldots, \mathbf{A}_e$. For sufficiently large e, the tuple $(\mathbf{r}, \mathbf{e})$ could be recovered by linear algebra, hence violating pseudorandomness (even one-wayness).

The TDF based on the LWR function does not suffer from the same vulnerability since, if we were to write $\mathbf{r} \mapsto \lfloor \mathbf{rA}_i \rceil_p = \mathbf{rA}_i + \mathbf{e}_i \bmod q$, where $\mathbf{e}_i$ is the deterministic error induced by rounding, an xPRG output would instead consist of LWR samples $(\mathbf{rA}_i + \mathbf{e}_i \bmod q)_{i=1}^{e}$ with different error terms $\mathbf{e}_1, \ldots, \mathbf{e}_e$.

4.3.2 GGM-based PRFs from LWR.

In the next section we will use our xPRGs for instantiating the GGM construction and thus obtaining extractable PRFs. Thus for our xPRGs from LWR, we will obtain a GGM-based PRF from LWR. Chuengsatiansup and Stehlé [CS19] construct a PRF from LWR, based as well on the GGM. They obtain a PRG from a single LWR instance (instead of e instances) and plug that PRG in the GGM construction. The authors also provide concrete parameter instantiations for adjusting the input-output rate of the PRGs and thus obtaining optimal efficiency in the GGM construction.

The LWR-based PRGs presented in [CS19] are certainly simpler than our LWR-based (x)PRGs obtained from the construction presented in this section. We namely concatenate two or more of the LWR-based TDF described in Figure 7 for obtaining a PRG with the desired e-expansion factor (see Figure 5). The reason why we cannot use a single LWR instance for each PRG in the GGM as done in [CS19] is that we want our final GGM-based PRF to be extractable, i.e. a GGM output and the trapdoor should allow to go back to the seed of the initial PRG in the tree. We can achieve this by going backwards through the GGM and extracting the seed from each PRG in the tree. Our PRG outputs correspond to concatenated TDF-outputs, whereby all trapdoors are instantiated with the same input (see Figure 5). Thus, the seed of the PRGs in the tree need to come from *complete* TDF outputs so that once that seed is recovered, that value can be used for going back to the input of the TDF used in the previous PRG.

4.4 xPRG from TDP

The Goldreich-Levin hardcore bit [GL89] allows to build a PRG from *any* one-way permutation. We here re-state their lemma for a keyed trapdoor permutation.

Lemma 4 (Goldreich-Levin)**.** *If* $\Pi.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *is a TDP (Definition 16) with* $\mathcal{X}_\lambda = \mathcal{Y}_\lambda = \{0,1\}^\lambda$*, then* $\mathcal{G}.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *is a trapdoor PRG (Definition 17) with domain* $\mathcal{X}_\lambda = \{0,1\}^{2\lambda}$ *and codomain* $\mathcal{Y}_\lambda = \{0,1\}^{2\lambda+1}$*, where* $\mathcal{G}.\mathsf{setup} = \Pi.\mathsf{setup}$*,* $\mathcal{G}.\mathsf{eval}(pp, x||r) := \Pi.\mathsf{eval}(pp, x)||r|| \bigoplus_{i=1}^{n} x_i \cdot r_i$*, and* $\mathcal{G}.\mathsf{inv}(td, y||r||b) := \Pi.\mathsf{inv}(td, y)||r$*.*

By iterating the PRG, we obtain a PRG which expands its input by an arbitrary polynomial $p(n)$, see Goldreich [Gol01, Theorem 3.3.3] for a proof.

Lemma 5 (PRG Length-Extension)**.** *Let* $\Pi.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *be a TDP (Definition 16),* $\mathcal{G}.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *be the resulting trapdoor PRG from Lemma 4, and let* $p = p(\lambda) > 2\lambda$ *and* $t = t(\lambda) = p(\lambda) - 2\lambda$ *be polynomials, then* $\mathcal{G}'.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *is a trapdoor PRG (Definition 17) with domain* $\mathcal{X}_\lambda = \{0,1\}^{2\lambda}$ *and comain* $\mathcal{Y}_\lambda = \{0,1\}^p$*, where* $\mathcal{G}'.\mathsf{setup} = \mathcal{G}.\mathsf{setup} = \Pi.\mathsf{setup}$ *and Figure 8 defines* $\mathcal{G}'.\mathsf{eval}$ *and* $\mathcal{G}'.\mathsf{inv}$*.*

$\mathcal{G}'.\mathsf{eval}(pp, x\|\|r)$	$\mathcal{G}'.\mathsf{inv}(td, y\|\|r\|\|z_1\|\|\ldots\|\|z_t)$
$x^{(0)} \leftarrow x$	$x^{(p)} \leftarrow y$
for $i \in [t]$ **do**	**for** $i \in [t]$ **do**
$x^{(i)}\|\|r\|\|z_i \leftarrow \mathcal{G}.\mathsf{eval}(pp, x^{(i-1)}\|\|r)$	$x^{(t-i)} \leftarrow \Pi.\mathsf{inv}(td, x^{(t-i+1)})$
return $x^{(t)}\|\|r\|\|z_1\|\|\ldots\|\|z_t$	**return** $x^{(0)}$

Figure 8: Construction of $\mathcal{G}'.\mathsf{eval}$ and $\mathcal{G}'.\mathsf{inv}$.

It is immediate that $\mathcal{G}'.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ is actually a 1xPRG up to syntactical changes.

Lemma 6 (1xPRG)**.** *Let* $\Pi.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *be a TDP (Definition 16),* $\mathcal{G}'.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *be the resulting trapdoor PRG from first applying Lemma 4 and subsequently Lemma 5 using* $p(\lambda) = e \cdot 2\lambda$*. Interpreting each* 2λ *chuck of the output of* $\mathcal{G}'.\mathsf{eval}$ *as a symbol, then* $\mathcal{G}'_{1x}.(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *is a 1xPRG, where* $\mathcal{G}'_{1x}.(\mathsf{setup}, \mathsf{eval}) := \mathcal{G}'.(\mathsf{setup}, \mathsf{eval})$*,* $\mathcal{G}'_{1x}.\mathsf{inv}(td, 1, y\|\|r) := \mathcal{G}'_{1x}.\mathsf{inv}(td, y\|\|r\|\|0\|\|\ldots\|\|0)$*, and* $\mathcal{G}'_{1x}.\mathsf{inv}(\cdot, i, \cdot)$ *is defined arbitrarily for* $i > 1$*.*

In Figure 9, we show how to build an extractable PRG $\mathcal{G}$ with domains and codomains $\mathcal{X}_\lambda = \mathcal{Y}_\lambda = \Sigma_\lambda^e$ from a 1-extractable PRG $\mathcal{G}$ with domains and codomains $\mathcal{X}'_\lambda = \mathcal{Y}'_\lambda = \Sigma_\lambda$, where Σ_λ is equipped with group operation $+$.

Lemma 7 (xPRG)**.** *If* $\mathcal{G}'$ *is a 1xPRG with* $\mathcal{X}_\lambda = \Sigma$ *and* $\mathcal{Y}_\lambda = \Sigma^e$*, then* $\mathcal{G}$ *(defined in Figure 9) is an xPRG with domain* Σ_λ^e *and range* $\Sigma_\lambda^{e^2}$*. I.e. from a 1xPRG with expansion factor* e*, we build an xPRG with expansion factor* e*.*

Proof. Extractability. We show that for all $i \in \mathbb{Z}_e$, $\mathcal{G}.\mathsf{inv}(td, i, y_i)$ indeed successfully returns a pre-image. Firstly, for $k \in [j-1]$, the values x_{i+k} have already been defined in previous loops and thus, $\mathcal{G}.\mathsf{inv}$ is well-defined. Secondly, the j-th symbol of $\mathbf{y}_i$ is computed as $g'_0(x_{i+j}) + \sum_{k=1}^{j} g'_{j-k}(x_{i+k})$, and thus, substracting from it $\sum_{k=1}^{j} g'_{j-k}(x_{i+k})$ yields $g'_j(x_{i+j})$ so that running it through $\mathcal{G}'.\mathsf{inv}(td, \cdot)$ yields x_{i+j} by correctness of $\mathcal{G}'$.

Pseudorandomness. Pseudorandomness reduces to pseudorandomness of $\mathcal{G}'$. For all $0 \le i, j \le e-1$, denote $z_{i,j} := g'_j(x_i)$. Then, we can re-write $\mathcal{G}'.\mathsf{eval}(x)$ as first computing $(z_{i,0}, .., z_{i,e-1}) \leftarrow \mathcal{G}'.\mathsf{eval}(x_i)$ and then computing $\mathbf{y}_i[j] \leftarrow z_{i,j} + z_{i+1,j-1} + .. + z_{i+j,0}$. Note that the x_i are now only used to compute the $(z_{i,0}, .., z_{i,e-1})$ and nowhere else. Thus, to replace all $z_{i,j}$ by uniformly random values, we can proceed via a hybrid argument over all e invocations of $\mathcal{G}'$: In the i-th hybrid game, for all $i' < i$, we sample $(z_{i',0}, .., z_{i',e-1})$ uniformly at random, and for all $i' \ge i$, we sample $x_{i'}$ and compute $(z_{i',0}, .., z_{i',e-1}) \leftarrow \mathcal{G}'.\mathsf{eval}(x_{i'})$. The 0-th hybrid corresponds to the construction $\mathcal{G}.\mathsf{eval}$ while in the e-th hybrid, all $z_{i,j}$ have been replaced by random values as desired. The reduction from hybrid i to $i+1$ to $\mathcal{G}'$ security embeds its input challenge as $(z_{i+1,0}, .., z_{i+1,e-1})$, samples all $(z_{i',0}, .., z_{i',e-1})$ uniformly random for $i' \le i$ and for $i' > i+1$, it samples $x_{i'}$ and computes $(z_{i',0}, .., z_{i',e-1}) \leftarrow \mathcal{G}'.\mathsf{eval}(x_{i'})$. The reduction emulates both hybrids perfectly.

Now that we have replaced all $z_{i,j}$ by uniformly random strings, we argue that the returned distribution is already uniformly random. To see this, consider the following information-theoretic argument. Since $\mathbf{y}_i[j] = z_{i,j} + z_{i+1,j-1} + .. + z_{i+j,0}$ for all $0 \le i, j \le e-1$, we can observe that terms $z_{i,j}$ with $j = e-1$ only ever in $\mathbf{y}_i[e-1]$, because all other sums only sum over smaller j-values. Since for all i, the term $z_{i,e-1}$ is only used in one sum, namely in $\mathbf{y}_i[e-1]$, the sum $\mathbf{y}_i[e-1] = z_{i,e-1} + z_{i+1,e-2} + .. + z_{i+e-1,0}$ is uniformly random and independent of everything else, since it sums the term $z_{i,e-1}$ which is uniformly random and independent of everything else. Therefore, we can replace $\mathbf{y}_i[e-1] = z_{i,e-1} + z_{i+1,e-2} + .. + z_{i+e-1,0}$ by $\mathbf{y}_i[e-1] = z_{i,e-1}$ while maintaining the same

$\mathcal{G}.\mathsf{eval}(pp, x \in \Sigma_\lambda^e)$

$(x_0, \ldots, x_{e-1}) \leftarrow x$

$(g'_0, \ldots, g'_{e-1}) \leftarrow \mathcal{G}'.\mathsf{eval}(\cdot)$

for $i \in \mathbb{Z}_e$ **do**

$$\mathbf{y}_i \leftarrow \begin{pmatrix} g'_0 & & & \\ g'_1 & g'_0 & & \\ \vdots & \vdots & \ddots & \\ g'_{e-1} & g'_{e-2} & \cdots & g'_0 \end{pmatrix} \begin{pmatrix} x_i \\ x_{i+1} \\ \vdots \\ x_{i+e-1} \end{pmatrix}$$

return $(\mathbf{y}_0, \ldots, \mathbf{y}_{e-1})$

$\mathcal{G}.\mathsf{inv}(td, i, y \in \Sigma_\lambda^e)$

$(y_0, \ldots, y_{e-1}) \leftarrow y$

$(g'_0, \ldots, g'_{e-1}) \leftarrow \mathcal{G}'.\mathsf{eval}(\cdot)$

for $j \in \mathbb{Z}_e$ **do**

$$x_{i+j} \leftarrow \mathcal{G}'.\mathsf{inv}\left(td, 1, y_j - \sum_{k=1}^{j} g'_{j-k}(x_{i+k})\right)$$

$x \leftarrow (x_0, \ldots, x_{e-1})$

return x

Figure 9: Construction of xPRGs from 1xPRGs, where $\mathcal{G}.\mathsf{setup} := \mathcal{G}'.\mathsf{setup}$. Subscripts arithmetic are modulo e. g'_j denotes the function returning the j-th output of $\mathcal{G}'.\mathsf{eval}(\cdot)$. Analogous to matrix multiplication, the expression assigned to $\mathbf{y}_i$ means that the j-th entry of $\mathbf{y}_i$ is given by $g'_j(x_i) + g'_{j-1}(x_{i+1}) + \ldots + g'_0(x_{i+j})$.

$\mathcal{F}.\mathsf{setup}(1^\lambda)$

for $i \in \mathbb{Z}_d$ **do**

  $(pp_i, td_i) \leftarrow\!\!\$\ \mathcal{G}_i.\mathsf{setup}(1^\lambda)$

$pp \leftarrow (pp_0, \ldots, pp_{d-1})$

$td \leftarrow (td_0, \ldots, td_{d-1})$

return (pp, td)

$\mathcal{F}.\mathsf{eval}(pp, k, x)$

$(pp_0, \ldots, pp_{d-1}) \leftarrow pp$

$(x_0, \ldots, x_{d-1}) \leftarrow x$

$k_0 \leftarrow k$

for $i = 0, 1, \ldots, d-1$ **do**

  $k_{i+1} \leftarrow \mathcal{G}_i.\mathsf{eval}(pp_i, x_i, k_i)$

return k_d

$\mathcal{F}.\mathsf{inv}(td, x, y)$

$(td_0, \ldots, td_{d-1}) \leftarrow td$

$(x_0, \ldots, x_{d-1}) \leftarrow x$

$k_d \leftarrow y$

for $i = d, d-1, \ldots, 1$ **do**

  $k_{i-1} \leftarrow \mathcal{G}_i.\mathsf{inv}(td_i, x_i, k_i)$

return k_0

Figure 10: Construction of an xPRF $\mathcal{F}$ with key space $\mathcal{X}_0$, domain $\mathbb{Z}_d^\ell$, and codomain $\mathcal{X}_d$ from xPRGs $(\mathcal{G}_i)_{i \in \mathbb{Z}_d}$ where $\mathcal{G}_i$ has domain $\mathcal{X}_i$ and codomain $\mathcal{X}_{i+1}$.

output distribution. After this step, $z_{i,e-2}$ is only used in one sum, namely in $\mathbf{y}_i[e-2]$ and thus, we can replace $\mathbf{y}_i[e-2] = z_{i,e-2} + z_{i+1,e-3} + .. + z_{i+e-2,0}$ by $\mathbf{y}_i[e-2] = z_{i,e-2}$ while maintaining the same output distribution. Continuing this process, we eventually end up defining $\mathbf{y}_i[j] = z_{i,j}$ for $0 \le i, j \le e-1$, all of which are uniformly random and independent values. □

As a corollary of Lemmas 4 to 7, we obtain the following theorem.

Theorem 3 (xPRG from TDP)**.** *Let* $(\mathsf{setup}, \mathsf{eval}, \mathsf{inv})$ *be a TDP and* $e = \mathsf{poly}(\lambda)$*. The family*$\mathcal{G}$ *constructed in Figure 9 is an xPRG with domain* Σ_λ^e*, range* $\Sigma_\lambda^{e^2}$ *and expansion factor* e*, where* $\Sigma_\lambda = \{0,1\}^e$*.*

5 Watermarkable PRF Construction

We now construct watermarkable PRFs from xPRGs, e.g. those provided in Section 4. The core idea is to use the xPRG in a GGM tree [GGM84] to inherit the pseudorandomness and puncturability of GGM. At the same time, the extraction properties of the xPRG carry over to the watermarkable PRF.

Theorem 4. *Let* $d = \mathsf{poly}(\lambda)$*. For* $i \in \mathbb{Z}_d$*, let* $\mathcal{G}_i$ *be an xPRG with input domain* $\mathcal{X}_{\lambda,i}$ *and, for* $i \in \mathbb{Z}_{d-1}$*, let the output domain be* $\mathcal{Y}_{\lambda,i} = \mathcal{X}_{\lambda,i+1}^\kappa$*, and for* $i = d-1$*, let* $\mathcal{Y}_{\lambda,i} = \Sigma^\kappa$*. Then* $\mathcal{F}$ *(Figure 10) is a watermarkable PRF with key space* $\mathcal{X}_{\lambda,0}$*, input domain* $\{0,1\}^{\ell \cdot \log(\kappa)}$ *and output domain* Σ*.*

$\mathsf{punctEval}(pp, k_{\mathcal{S}}, x^*)$

$(K_0, \ldots, K_{d-1}) \leftarrow k_{\mathcal{S}},\ j \leftarrow -1$
for $i = 0, 1, \ldots, d-1$ **do**
  if $j = -1 \wedge x_i^* \neq x_i$ **then** $j \leftarrow i$
if $j = -1$ **then abort**
$k_j \leftarrow k_{x_0,\ldots,x_{i-1},x_j^*}$
for $i = j+1, \ldots, d-1$ **do**
  $k_{i+1} \leftarrow \mathcal{G}_i.\mathsf{eval}(pp_i, x_i, k_i)$
return k_d

Figure 11: Description of punctEval.

Proof. We show that $\mathcal{F}$ is extractable (Definition 8), puncturable (Definition 10), pseudorandom (Definition 7), and puncturably secure (Definition 11).

Syntax and extractability. The algorithms are well-defined and satisfy extractability (Definition 8), which demands that for any key $k \in \mathcal{X}_{\lambda,0}$, any input $x \in \{0,1\}^{\ell \cdot \log(\kappa)}$, it holds that $\mathsf{inv}(td, x, \mathsf{eval}(pp, k, x)) = k$. Indeed, this follows direcly from the extractability property of the xPRGs $\mathcal{G}_0, \ldots, \mathcal{G}_{d-1}$.

Puncturability. The puncturing is standard for the GGM tree and only included here for completeness. We first describe the puncturing procedure for puncturing at a singleton $\mathcal{S} := \{x\}$ and then explain how to puncture at general sets.

The key punctured at $x = (x_0, \ldots, x_{d-1})$ consists of the following d sets $K_i := \{k_{x_0,\ldots,x_{i-1},w^*} | w^* \in \{0,1\}^{\log(\kappa)} \setminus \{x_i\}\}$ for $i \in \mathbb{Z}_d$, where $k_w := \mathcal{G}_0.\mathsf{eval}(pp_0, w, k_0)$ for $w \in \{0,1\}^{\log(\kappa)}$ and $k_{x_0,\ldots,x_{i-1},w} := \mathcal{G}_i.\mathsf{eval}(pp_i, w, k_0)$ for $w \in \{0,1\}^{\log(\kappa)}$. In order to evaluate the PRF on a punctured key $(K_0, \ldots, K_{d-1})$ and input $x^* \neq x$, parse $x^* = (x_0^*, \ldots, x_{d-1}^*)$, take the first j such that $x_j^* \neq x_j$ (which exists since $x^* \neq x$), retrieve $k_{x_0,\ldots,x_{j-1},x_j^*}$ and continue evaluation from there, see Figure 11.

Pseudorandomness. The security reduction for Theorem 4 is the standard GGM security reduction [GGM84]. The existence of trapdoors does not affect the reduction since the adversary is not given the trapdoors. The reduction is slightly generalized to expansion κ instead of just a length-doubling PRG which has $\kappa = 2$.

Puncturing Security. Puncturing security is also the standard proof for puncturing security of GGM, see, e.g. [GGM84], again generalized to expansion κ. It applies the PRG security once on each layer to show that all keys in K_i are indistinguishable from uniformly random keys from the appropriate key spaces. □

Acknowledgements

We thank the anonymous reviewers of CiC for insightful feedback and discussions. Chris Brzuska and Russell W. F. Lai are supported by the Research Council of Finland grants 358950 and 358951 respectively.

References

[AAB+19] Estuardo Alpirez Bock, Alessandro Amadori, Joppe W. Bos, Chris Brzuska, and Wil Michiels. Doubly half-injective PRGs for incompressible white-box

cryptography. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 189–209. Springer, Heidelberg, March 2019. doi:10.1007/978-3-030-12612-4_10.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552. Springer, Heidelberg, August 2007. doi:10.1007/978-3-540-74143-5_30.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_13.

[BFO08] Alexandra Boldyreva, Serge Fehr, and Adam O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 335–359. Springer, Heidelberg, August 2008. doi:10.1007/978-3-540-85174-5_19.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014. doi:10.1007/978-3-642-55220-5_30.

[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):6:1–6:48, 2012. doi:10.1145/2160158.2160159.

[BGM+16] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 209–224. Springer, Heidelberg, January 2016. doi:10.1007/978-3-662-49096-9_9.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013. doi:10.1145/2508859.2516734.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_42.

[CHN+16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1115–1127. ACM Press, June 2016. doi:10.1145/2897518.2897651.

[CS19] Chitchanok Chuengsatiansup and Damien Stehlé. Towards practical GGM-based PRF from (module-)learning-with-rounding. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 693–713. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-38471-5_28.

[DGH+19] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Kevin Liu, and Giulio Malavolta. Rate-1 trapdoor functions from the Diffie-Hellman problem. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 585–606. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-34618-8_20.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 276–288. Springer, Heidelberg, August 1984. doi:10.1007/3-540-39568-7_22.

[GKWW21] Rishab Goyal, Sam Kim, Brent Waters, and David J. Wu. Beyond software watermarking: Traitor-tracing for pseudorandom functions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 250–280. Springer, Heidelberg, December 2021. doi:10.1007/978-3-030-92078-4_9.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989. doi:10.1145/73007.73010.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[HMW07] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 362–382. Springer, Heidelberg, February 2007. doi:10.1007/978-3-540-70936-7_20.

[KW17] Sam Kim and David J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 503–536. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63688-7_17.

[KW19] Sam Kim and David J. Wu. Watermarking PRFs from lattices: Stronger security via extractable PRFs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 335–366. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26954-8_11.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_41.

[MW22] Sarasij Maitra and David J. Wu. Traceable PRFs: Full collusion resistance and active security. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 439–469. Springer, Heidelberg, March 2022. doi:10.1007/978-3-030-97121-2_16.

[QWZ18] Willy Quach, Daniel Wichs, and Giorgos Zirdelis. Watermarking PRFs under standard assumptions: Public marking and security with extraction queries. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 669–698. Springer, Heidelberg, November 2018. doi:10.1007/978-3-030-03810-6_24.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. doi:10.1145/1060590.1060603.

[YAL+19] Rupeng Yang, Man Ho Au, Junzuo Lai, Qiuliang Xu, and Zuoxia Yu. Collusion resistant watermarking schemes for cryptographic functionalities. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 371–398. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-34578-5_14.

[YAYX20] Rupeng Yang, Man Ho Au, Zuoxia Yu, and Qiuliang Xu. Collusion resistant watermarkable PRFs from standard assumptions. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 590–620. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56784-2_20.

A From t-Extractability to Robust Extractability

For xPRFs which are only t-extractable for $t < |\mathcal{X}|$ to achieve robust extractability [KW19], they need to additionally satisfy a property which we call extraction-points hiding, i.e. the public parameters statistically hide $S(td)$.

Definition 18 (Extraction-Points Hiding)**.** A t-extractable xPRF family $\mathcal{F}$ is extraction-points hiding if for $\ell = \mathsf{poly}(\lambda)$ and any (unbounded) adversary $\mathcal{A}$

$$\left| \begin{array}{l} \Pr\left[\mathcal{A}(pp, \mathcal{S}) = 1 \;\middle|\; (pp, td) \leftarrow\!\$\; \mathcal{F}.\mathsf{setup}(1^\lambda, 1^\ell),\ \mathcal{S} := \mathcal{S}(td)\right] \\ - \Pr\left[\mathcal{A}(pp, \mathcal{S}) = 1 \;\middle|\; (pp, td) \leftarrow\!\$\; \mathcal{F}.\mathsf{setup}(1^\lambda, 1^\ell),\ \mathcal{S} \leftarrow\!\$\; \binom{\mathcal{X}_{\lambda,\ell}}{t}\right] \end{array} \right| = \mathsf{negl}(\lambda)$$

where $\mathcal{S} \leftarrow\!\$\; \binom{\mathcal{X}_{\lambda,\ell}}{t}$ denotes the sampling of a random t-subset from $\mathcal{X}_{\lambda,\ell}$.

Lemma 8 (Alternative to Lemma 3)**.** *Let $\mathcal{F}$ be a xPRF family. If $t = \mathsf{poly}(\lambda)$, $t/|\mathcal{X}_{\lambda,\ell}| = \mathsf{negl}(\lambda)$ for any $\ell = \mathsf{poly}(\lambda)$, and $\mathcal{F}$ is key-injective (Definition 9), t-extractable (Definition 8), and extraction-points hiding (Definition 18), then $\mathcal{F}$ has robust extractability (Definition 15).*

Proof. By Lemma 2, $\mathcal{F}$ is (ϵ, δ)-publicly testable with the algorithm $\mathcal{F}.\mathsf{Test}_{\epsilon,\delta}$ constructed in Figure 3. The proof idea is implicit in the proof of [KW19, Theorem 4.26]. The proof proceeds via a sequence of games, described below:

- Game 0: The real robust extractability game $\mathtt{RExt}^0_{\mathcal{F}}$.
- Game 1: The extraction points $\mathcal{S}$ are now sampled uniformly at random, as in the second distribution of Definition 18.
- Game 2: extrO queries are now answered using the following inefficient algorithm $\mathsf{IneffExtr}$: Evaluate C on all $x_i \in \mathcal{S}$ to obtain pairs (x_i, y_i). For each of these pairs, check whether there is a unique k_i such that $\mathcal{F}.\mathsf{eval}(pp, k_i, x_i) = y_i$. If yes, then run $\mathsf{Test}_{\epsilon,\delta}(pp, k_i, C)$, and if $\mathsf{Test}_{\epsilon,\delta}$ outputs 1, return k_i.
- Game 3: extrO now performs the ideal test of whether the program is δ-close to $\mathcal{F}.\mathsf{eval}(pp, k, .)$.
- Game 4: The ideal robust extractability game $\mathtt{RExt}^1_{\mathcal{F}}$. Extraction points are now defined by the trapdoor, as in the first distribution of Definition 18.

Below, we prove the computational indistinguishability of the games.

Game 0 to 1: The two games are statistically indistinguishable due to (statistical) extraction-point hiding.

Game 1 to 2: We prove that the two games are statistically indistinguishable. By key-injectivity and by the use of identical extraction points, with overwhelming probability, the inefficient extraction algorithm considers the same set of candidate keys as $\mathcal{F}.\mathsf{extr}$. Assuming this is the case, then both algorithms run $\mathsf{Test}_{\epsilon,\delta}$ which succeeds/fails with the same probability in both games.

Game 2 to 3: First, observe that using a Chernoff bound testing whether the program C is δ-close to $\mathcal{F}.\mathsf{eval}(pp, k, .)$ can be reasonably closely approximated by extracting from C on a polynomial number of random points and testing the resulting keys. We use the two ways of testing closeness equivalently in the following.

This proof is a hybrid argument over the number of queries which the adversary makes. Before the first extraction query of the adversary, the adversary has information-theoretically no information about the extraction points $\mathcal{S}$ which extrO uses. When the adversary submits a close program C, it will be tested on uniformly random values (in the view of the adversary) chosen from a spuer-polynomially-large space and thus, when it is δ-close to $\mathcal{F}.\mathsf{eval}(pp, k, .)$, the oracle will return k with overwhelming probability in Game 2, and we can replace the extraction oracle with an ideal oracle which does not sample extraction points. Thus, the answer to the first extraction query does not leak any information to the adversary about the extraction points, and now, the argument can be applied to the second query and so on.

Game 3 to 4: This step is the same as the hop from Game 1 to 0. □

B The Kim-Wu Transformations

We summarise, in our notation, the transformations given by Kim and Wu [KW19], from watermarkable to mark- and message-embedding watermarking PRFs.

B.1 Definitions for Watermarking PRFs

Definition 19 (Watermarking Pseudorandom Functions)**.** A (secretly markable and extractable) watermarking pseudorandom function family $\mathcal{F}$ with message spaces $(\mathcal{M}_\lambda)_{\lambda\in\mathbb{N}}$, key spaces $(\mathcal{K}_\lambda)_{\lambda\in\mathbb{N}}$, domains $(\mathcal{X}_{\lambda,\ell})_{\lambda,\ell\in\mathbb{N}}$ and codomains $(\mathcal{Y}_{\lambda,\ell})_{\lambda,\ell\in\mathbb{N}}$ consists of four PPT algorithms $(\mathsf{setup}, \mathsf{eval}, \mathsf{mark}, \mathsf{extr})$ with the following syntax:

- $(pp, td) \leftarrow\!\!\$\ \mathsf{setup}(1^\lambda, 1^\ell)$: The setup algorithm takes 1^λ and length parameter 1^ℓ as input and outputs the public parameters pp and a trapdoor td.
- $y \leftarrow \mathsf{eval}(pp, k, x)$: The deterministic evaluation algorithm takes pp, a secret key $k \in \mathcal{K}_\lambda$, and a value $x \in \mathcal{X}_{\lambda,\ell}$ and outputs an image $y \in \mathcal{Y}_{\lambda,\ell}$.
- $C \leftarrow \mathsf{mark}(td, k, m)$: The watermarking algorithm inputs the trapdoor td, a PRF key $k \in \mathcal{K}_\lambda$, and a message $m \in \mathcal{M}_\lambda$ and outputs a circuit $C : \mathcal{X}_{\lambda,\ell} \to \mathcal{Y}_{\lambda,\ell}$.
- $m \leftarrow \mathsf{extr}(td, C)$: The mark extraction algorithm inputs the trapdoor td and a circuit $C : \mathcal{X}_{\lambda,\ell} \to \mathcal{Y}_{\lambda,\ell}$ and outputs a message $m \in \mathcal{M}_\lambda \cup \{\perp\}$.

If $|\mathcal{M}_\lambda| \geq 2$ for all λ then we say that $\mathcal{F}$ is a message-embedding watermarking PRF family. If $|\mathcal{M}_\lambda| = 1$ for all λ then we say that $\mathcal{F}$ is a mark-embedding watermarking PRF family. In this case, we drop the input m from the mark algorithm. Similarly, extr outputs either marked or $\mathsf{unmarked}$ rather than a message m.

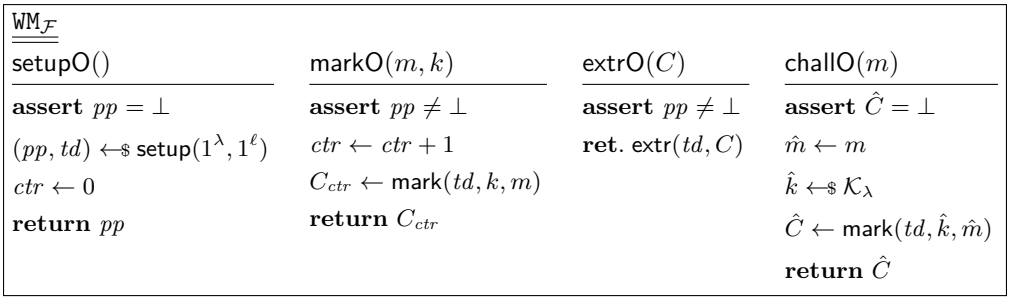

Figure 12: Watermarking game to define unforgeability and unremoveability for mark-embedding and message-embedding watermarking PRFs. For *mark*-embedding watermarking, the oracles and algorithms do not take a message m as input.

Correctness and pseudorandomness. Correctness requires that the watermarked program $C \leftarrow \mathsf{mark}(td, k, m)$ (or $C \leftarrow \mathsf{mark}(td, k)$ for mark-embedding schemes) behaves as $\mathsf{eval}(pp, k, \cdot)$ except for a negligible fraction of the domain. Pseudorandomness requires that the marked program is still indistinguishable from a random function when given black-box access to it, even when the distinguisher has access to an extraction oracle. Recall that we do not consider pseudorandomness against the authority.

Unforgeability and Unremoveability. We recall unforgeability and unremoveability, which are the two main security properties for watermarking PRFs. Unremoveability states that the adversary cannot remove the message (or mark) from the program without substantially sacrificing the program's functionality. Unforgeability states that the adversary cannot create new marked programs.

Concretely, both unremoveability and unforgeability are defined with respect to a watermarking game $\mathtt{WM}_{\mathcal{F}}$ (cf. Frg. 12). For unremoveability, the adversary gets a challenge program $\hat{C}$ and tries to create a program C^* with a similar functionality as $\hat{C}$, but not containing the embedded message (resp. mark). The formal definitions is as follows.

Definition 20 (ϵ-unremoveability)**.** A message-embedding watermarking pseudorandom function family $\mathcal{F}$ is ϵ-unremovable if, for all PPT adversaries $\mathcal{A}$, the probability

$$\Pr\Big[(C^*, \hat{C})\ \epsilon\text{-close} \wedge \mathsf{extr}(td, C^*) \neq \hat{m}\Big]$$

is negligible, where

$$C^* \leftarrow\!\!\$\ \mathcal{A}^{\substack{\mathsf{setupO}\\ \mathsf{markO}\\ \mathsf{extrO}\\ \mathsf{challO}}} \mathtt{WM}_{\mathcal{F}},$$

and $\hat{C}$ and $\hat{m}$ are defined by $\mathcal{A}$'s query to challO. Analogously, a mark-embedding watermarking pseudorandom function family $\mathcal{F}$ is ϵ-unremovable if, for all PPT adversaries $\mathcal{A}$, the probability

$$\Pr\Big[(C^*, \hat{C})\ \epsilon\text{-close} \wedge \mathsf{extr}(td, C^*) = \mathsf{unmarked}\Big]$$

is negligible, where

$$C^* \leftarrow\!\!\$\ \mathcal{A}^{\substack{\mathsf{setupO}\\ \mathsf{markO}\\ \mathsf{extrO}\\ \mathsf{challO}}} \mathtt{WM}_{\mathcal{F}},$$

and $\hat{C}$ is defined by $\mathcal{A}$'s query to challO.

$\mathcal{F}'.\mathsf{setup}(1^\lambda)$

$(pp_{\mathcal{F}}, td_{\mathcal{F}}) \leftarrow\!\$\ \mathcal{F}.\mathsf{setup}(1^\lambda)$
$pp_{\mathcal{G}} \leftarrow\!\$\ \mathcal{G}.\mathsf{setup}(1^\lambda)$
$k_{\mathcal{G}} \leftarrow\!\$\ \mathcal{G}.\mathcal{K}_\lambda$
return $((pp_{\mathcal{F}}, pp_{\mathcal{G}}), (td_{\mathcal{F}}, k_{\mathcal{G}}))$

$\mathcal{F}'.\mathsf{eval}(pp, k, x)$

return $\mathcal{F}.\mathsf{eval}(pp_{\mathcal{F}}, k, x)$

$\mathcal{F}'.\mathsf{mark}(td, k)$

for $i \in [\lambda]$ **do**
  $x_{k,i} \leftarrow \mathcal{G}.\mathsf{eval}(pp_{\mathcal{G}}, k_{\mathcal{G}}, (k, i))$
$\mathcal{S} \leftarrow \{x_{k,i} : i \in [\lambda]\}$
$k_{\mathcal{S}} \leftarrow \mathcal{F}.\mathsf{punct}(pp_{\mathcal{F}}, k, \mathcal{S})$
$C(\cdot) \leftarrow \mathcal{F}.\mathsf{punctEval}(pp_{\mathcal{F}}, k_{\mathcal{S}}, \cdot)$
return C

$\mathcal{F}'.\mathsf{extr}(td, C)$

$k \leftarrow \mathcal{F}.\mathsf{extr}(td, C)$
for $i \in [\lambda]$ **do**
  $x_{k,i} \leftarrow \mathcal{G}.\mathsf{eval}(pp_{\mathcal{G}}, k_{\mathcal{G}}, (k, i))$
  if $\mathcal{F}.\mathsf{eval}(pp_{\mathcal{F}}, k, x_{k,i}) = C(x_{k,i})$
    then return unmarked
return marked

Figure 13: Construction of mark-embedding watermarking PRF $\mathcal{F}'$ from a watermarkable PRF $\mathcal{F}$ and an ordinary PRF $\mathcal{G}$.

For unforgeability, the adversary does not make any query to the challenge oracle but instead tries to come up with a new, forged marked program C^*. The adversary is considered successful if it creates a program C^* that is far from all marked programs $C_1, \ldots, C_\ell$ that it received from the markO oracle, but is considered marked by extr.

Definition 21 (δ-unforgeability)**.** A message-embedding watermarking pseudorandom function family $\mathcal{F}$ is δ-unforgeable if for all PPT adversaries $\mathcal{A}$, the probability

$$\Pr[\forall i \in [ctr],\ (C^*, C_i)\ \delta\text{-far} \wedge \mathsf{extr}(td, C^*) \neq \bot]$$

is negligible, where

$$C^* \leftarrow\!\$\ \mathcal{A}^{\substack{\mathsf{setupO}\\ \mathsf{markO}\\ \mathsf{extrO}}} \mathtt{WM}_{\mathcal{F}},$$

and ctr and C_i are defined by $\mathcal{A}$'s queries to markO. Analogously, a mark-embedding watermarking pseudorandom function family $\mathcal{F}$ is δ-unforgeable if for all PPT adversaries $\mathcal{A}$, the probability

$$\Pr[\forall i \in [ctr],\ (C^*, C_{ctr})\ \delta\text{-far} \wedge \mathsf{extr}(td, C^*) = \mathsf{marked}]$$

is negligible, where

$$C^* \leftarrow\!\$\ \mathcal{A}^{\substack{\mathsf{setupO}\\ \mathsf{markO}\\ \mathsf{extrO}}} \mathtt{WM}_{\mathcal{F}},$$

and ctr and C_i are defined by $\mathcal{A}$'s queries to markO.

B.2 From Watermarkable to Watermarking

We here recall a simple, elegant transformation by Kim and Wu [KW19] to transform, in our terminologies, a watermarkeable PRF into a mark-embedding watermarking PRF. In order to embed a mark into a PRF key k, the mark algorithm punctures k at λ many points $x_{k,1}, \ldots, x_{k,\lambda}$ which are *deterministically* computed from k (using a standard PRF keyed with a private watermarking key). To determine whether the program is marked or not, the extr algorithm first recovers k, re-computes $x_{k,1}, \ldots, x_{k,\lambda}$ and checks whether the given program behaves as $\mathsf{eval}(k, \cdot)$ or not. See Figure 13 for a formal description.

Then, in Figure 14, we recall a generalisation of the transform, also presented in [KW19], which turns a watermarkable PRF into a message-embedding watermarking PRF. The idea, here, is that for every message bit, 2λ many points are derived and the watermarkeable PRF key is punctured on half of them, depending on the value of the message bit. Analogously, the extr algorithm then recovers the message bit by bit.

$\mathcal{F}'.\mathsf{setup}(1^\lambda)$

$(pp_{\mathcal{F}}, td_{\mathcal{F}}) \leftarrow\!\$\ \mathcal{F}.\mathsf{setup}(1^\lambda)$
$pp_{\mathcal{G}} \leftarrow\!\$\ \mathcal{G}.\mathsf{setup}(1^\lambda)$
$k_{\mathcal{G}} \leftarrow\!\$\ \mathcal{G}.\mathcal{K}_\lambda$
return $((pp_{\mathcal{F}}, pp_{\mathcal{G}}), (td_{\mathcal{F}}, k_{\mathcal{G}}))$

$\mathcal{F}'.\mathsf{eval}(pp, k, x)$

return $\mathcal{F}.\mathsf{eval}(pp_{\mathcal{F}}, k, x)$

$\mathcal{F}'.\mathsf{mark}(td, k, m)$

for $(i, j) \in [\lambda]^2$ **do**
  $x_{k,i,j,m_j} \leftarrow \mathcal{G}.\mathsf{eval}(pp_{\mathcal{G}}, k_{\mathcal{G}}, (k, i, j, m_j))$
$\mathcal{S} \leftarrow \{x_{k,i,j,m_j} : (i, j) \in [\lambda]^2\}$
$k_{\mathcal{S}} \leftarrow \mathcal{F}.\mathsf{punct}(pp_{\mathcal{F}}, k, \mathcal{S})$
$C(\cdot) \leftarrow \mathcal{F}.\mathsf{punctEval}(pp_{\mathcal{F}}, k_{\mathcal{S}}, \cdot)$
return C

$\mathcal{F}'.\mathsf{extr}(td, C)$

$k \leftarrow \mathcal{F}.\mathsf{extr}(td, C)$
for $i, j, b \in [\lambda]^2 \times \{0, 1\}$ **do**
  $x_{k,i,j,b} \leftarrow \mathcal{G}.\mathsf{eval}(pp_{\mathcal{G}}, k_{\mathcal{G}}, (k, i, j, b))$
for $j, b \in [\lambda] \times \{0, 1\}$ **do**
  $N_{j,b} \leftarrow |\{i : \mathcal{F}.\mathsf{eval}(pp_{\mathcal{F}}, k, x_{k,i,j,b}) \neq C(x_{k,i,j,b})\}|$
  if $\exists j,\ N_{j,0}, N_{j,1} < 2\lambda/3 \vee N_{j,0}, N_{j,1} > 2\lambda/3$
    then return $\perp$
for $j \in [\lambda]$ **do**
  $m_j \leftarrow \arg\max_{b \in \{0,1\}} \{N_{j,b}\}$
return m

Figure 14: Construction of message-embedding watermarking PRF $\mathcal{F}'$ with message space $\mathcal{M} = \{0, 1\}^\lambda$ from a watermarkable, puncturable PRF $\mathcal{F}$ and an ordinary PRF $\mathcal{G}$.