Watermarking PRFs from Lattices via Extractable PRFs

Sam Kim and David J. Wu
Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]

Embed a “mark” within a program

If mark is removed, then program is destroyed

Two main algorithms (simplified):

- $\text{Mark}(C) \rightarrow C'$: Takes a circuit $C$ and outputs a marked circuit $C'$
- $\text{Verify}(C') \rightarrow \{0,1\}$: Tests whether a circuit $C'$ is marked or not
Watermarking Programs

[NS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]

Embed a "mark" within a program
If mark is removed, then program is destroyed

Two main algorithms (simplified):
- Mark\( (C) \rightarrow C' \): Takes a circuit \( C \) and outputs a marked circuit \( C' \)
- Verify\( (C') \rightarrow \{0,1\} \): Tests whether a circuit \( C' \) is marked or not

Notion extend to setting where watermark can be any string
Watermarking Programs

Functionality-preserving: On input a circuit $C$, the Mark algorithm outputs a circuit $C'$ where

$$C(x) = C'(x)$$

on almost all inputs $x$
Watermarking Programs

Unremovability: Given a marked program $C'$, no efficient adversary can construct a circuit $C^*$ where

- $C^*(x) = C'(x)$ on almost all inputs $x$
- The circuit $C^*$ is unmarked: $\text{Verify}(C^*) = 0$

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]
Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWM19]

Unremovability: Given a marked program $C'$, no efficient adversary can construct a circuit $C^*$ where

- $C^*(x) = C'(x)$ on almost all inputs $x$
- The circuit $C^*$ is unmarked: Verify$(C^*) = 0$

Adversary is very powerful: sees the code of the marked program $C'$ and has complete flexibility in crafting $C^*$
Watermarking Programs

Notion only achievable for functions that are not learnable
Focus has been on cryptographic functions

Learning the original (unmarked) function gives a way to remove the watermark

static void AES_enc_blk(block *blk, const AES_KEY *key) {
    unsigned j, rnds = round(key);
    const __m128i *sched = ((__m128i *) (key->ord_key));
    *blk = __mm_xor_si128(*blk, sched[0]);
    for (j = 1; j < rnds; ++j) {
        *blk = __mm_aesenc_si128(*blk, sched[j]);
    }
    *blk = __mm_aesenclast_si128(*blk, sched[j]);
}
Focus of this work: watermarking PRFs [CHNVW16, BLW17, KW17, QWZ18]
Watermarking Cryptographic Programs

[NS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]

A function whose input-output behavior is unpredictable (looks like a random function) – e.g., AES

Marking PRFs [CHNVW16, BLW17, KW17, QWZ18]
Watermarking Cryptographic Programs

Focus of this work: watermarking PRFs

\[ \text{pseudorandom function } \text{PRF}(k, \cdot) \]

Program has PRF key \( k \) hard-wired inside it and on input \( x \), outputs \( \text{PRF}(k, x) \)

Marking PRFs \([\text{CHNVW16, BLW17, KW17, QWZ18}]\)
Watermarkable PRFs

[CHNVW16]: Watermark PRFs from iO + OWFs
  Publicly verifiable

Can we watermark PRFs from standard assumptions?

[KW17]: Watermark PRFs from standard assumptions (LWE)
  Secretly verifiable
Watermarkable PRFs

Secretly Verifiable Watermarking from LWE [KW17]

Publicly Verifiable Watermarking [CHNVW16]
Secretly Verifiable Watermarking from LWE [KW17]

Problem: Knowledge of the verification key allows adversary to trivially remove watermark

In fact: Even oracle access to the verification key is sufficient to break unremovability (‘‘verifier rejection’’ problem)
Between Public and Secret Verification

Intermediate notion: Secret verification, but security in the presence of a verification oracle
Between Public and Secret Verification

Conceptually similar to a CCA security notion

“Minimal” stepping stone towards public verifiability

Intermediate notion: Secret verification, but security in the presence of a verification oracle
**Watermarkable PRFs**

**Secretly Verifiable Watermarking**
- from CCA [QWZ18]

**Secretly Verifiable Watermarking**
- from LWE [KW17]

**Publicly Verifiable Watermarking**
- [CHNVW16]
Watermarkable PRFs

**Secretly Verifiable Watermarking from LWE** \([\text{KW}17]\)

**Good:** Achieves security in the presence of a verification oracle

**Limitation:** Knowledge of the verification key breaks PRF security (even *unmarked* keys)
After seeing single query (on any $x$), authority can distinguish output of PRF from output of random function

Security Against the Authority

[QWZ18]
Security Against the Authority

Implication: Knowledge of the verification key completely breaks PRF security

(notion still seems far publicly-verifiable setting)

After seeing single query (on any $x$), authority can distinguish output of PRF from output of random function
Don’t We Have to Trust the Authority Anyways?

**Not necessarily:** marking algorithm can be implemented using a two-party computation, so authority never needs to see *any* PRF keys in the clear.
Don’t We Have to Trust the Authority Anyways?

This work: New watermarkable PRF that provides security even against the watermarking authority.
New **secretly verifiable** watermarking for PRF from LWE

- Unremovability holds in the presence of the verification oracle

- **weak pseudorandomness** even against authority
  
  \((T\text{-restricted pseudorandomness})\)

- As secure as any other PRF family from LWE
  
  - Relies on worst-case lattice problems with **nearly-polynomial** 
    \(n^{\omega(1)}\) approximation factors
Our Results

New *secretly verifiable* watermarking for PRF from LWE

- Unremovability holds in the presence of the verification oracle
- *Weak* pseudorandomness even against authority ($T$-restricted pseudorandomness)
- As secure as any other PRF family from LWE
  - Relies on worst-case lattice problems with nearly-polynomial ($n^{\omega(1)}$) approximation factors
- New abstraction: *extractable PRF*

Previous constructions (with message-embedding) required *private* constrained PRFs (which requires quasi-polynomial or sub-exponential approximation factors)
Starting Point: Puncturable PRF

Punctured key $k_{x^*}$ can be used to evaluate PRF on all points $x \neq x^*$
(value at $x^*$ is pseudorandom even given $k_{x^*}$)

Private puncturing: punctured key $k_{x^*}$ also hides $x^*$
Programmability: program $F(k_{x^*}, x^*) := y^*$
From Puncturing to Watermarking

[BLW17, KW17]

Marking algorithm:
1. Derive a special point \((x^*, y^*)\) from input/output behavior of PRF
2. Define a marked circuit to be \(F(k_{x^*}, \cdot)\)

Verification algorithm:
1. Test if \(C(x^*) = y^*\)

Security: Punctured point \(x^*\) is hidden
Intuition: Binary Search Attack

programmed key $k_{x^*}$

PRF Domain:

$\hat{x^*}$

Verify($vk, \cdot$)
Intuition: Binary Search Attack

Programmed key $k_{x^*}$

Verify $sk, \cdot$

Intuitively:
- if $Verify(sk, C) = 1$, then $x^* \notin S_2$
- if $Verify(sk, C) = 0$, then $x^* \in S_2$
Intuition: Binary Search Attack

Programmed key $k_{x^*}$

PRF Domain:

- $C(x) = \text{Eval}(k_{x^*}, x)$
- $C(x) \neq \text{Eval}(k_{x^*}, x)$

$C(x)$

Intuitively:
- if $\text{Verify}(sk, C) = 1$, then $x^* \notin S_2$
- if $\text{Verify}(sk, C) = 0$, then $x^* \in S_2$

Eventually, adversary recovers special point $x^*$

Verify($sk, \cdot$)
Intuition: Binary Search Attack

Very similar to a “verifier rejection” attack encountered in settings like designated-verifier proof systems, CCA-security, etc.

**Solution:** Make the set of “valid” circuits detectable (i.e., cannot change too many points and still preserve mark)

**PRF Domain:**

\[
C(x) = \text{Eval}(k_{x^*}, x) \\
\bigcirc x^* \\
S_1
\]

\[
C(x) \neq \text{Eval}(k_{x^*}, x) \\
S_2
\]

**Intuitively:**

- If \( \text{Verify}(sk, C) = 1 \), then \( x^* \notin S_2 \)
- If \( \text{Verify}(sk, C) = 0 \), then \( x^* \in S_2 \)

Eventually, adversary recovers special point \( x^* \)
Our Notion: Extractable PRF

Punctured key $k_{x^*}$ can be used to evaluate PRF on all points $x \neq x^*$

**Private puncturing:** punctured key $k_{x^*}$ also hides $x^*$

**Programmability:** program $F(k_{x^*}, x^*) := y^*$

**Extractability:** point $F(k_{x^*}, z) := \text{Encode}(k)$ encode original PRF key $k$
Our Notion: Extractable PRF

Punctured key $k_{x^*}$ can be used to evaluate PRF on all points $x \neq x^*$

Private puncturing: puncture
Programmability: program $F$
Extractability: point $F(k_{x^*}, z) := \text{Encode}(k)$ encode original PRF key $k$
Our Notion: Extractable PRF

Encode($k$)

Special point embeds information about the PRF key $k$ (specific to the PRF family, unknown to the key-holder)

Can recover $k$ from the encoding using trapdoor information
Extraction to Watermarking

Marking algorithm:
1. Derive a special point \((x^*, y^*)\) from input/output behavior of PRF
2. Define a marked circuit to be \(F(k_{x^*}, \cdot)\)

Verification algorithm:
1. Test if \(C(x^*) = y^*\)
2. Extract key \(k\) and test if \(C(\cdot) \approx F(k, \cdot)\)
   (output unmarked if key extraction fails)
3. Accept only if both conditions satisfied

In fact: extractability enables a simpler marking procedure

Adversary can rule out only a small fraction of domain
Extraction to Watermarking

Marking algorithm:
1. Derive a special point \((x^*, y^*)\) from input/output behavior of PRF
2. Define a marked circuit to be \(F(k_{x^*}, \cdot)\)

Verification algorithm:
1. Test if \(C(x^*) = y^*\)
2. Extract key \(k\) and test if \(C(\cdot) \approx F(k, x^*)\)
3. Accept only if both conditions satisfied

In fact: extractability enables a simpler marking procedure

Instead of programming the value at \(x^*\), puncture the PRF at \(x^*\): circuit is marked if \(C(x^*) \neq F(k, x^*)\) where \(k\) is the extracted key
Extraction to Watermarking

Marking algorithm:
1. Puncture key at $x^*$ to obtain a key $k_{x^*}$
2. Define a marked key to be $F(k_{x^*}, \cdot)$

Verification algorithm:
1. Extract key $k$ and test if $C(\cdot) \approx F(k, \cdot)$
   (output unmarked if key extraction fails)
2. Output marked if $C(x^*) \neq F(k, x^*)$ and unmarked otherwise

In fact: extractability enables a simpler marking procedure

To remove watermark, need to fix the value of the PRF at the punctured point (i.e., guess a pseudorandom value)
Extraction to Watermarking

Marking algorithm:
1. Puncture key at $x^*$ to obtain a key $k_{x^*}$
2. Define a marked key to be $F(k_{x^*}, \cdot)$

Verification algorithm:
1. Extract key $k$ and test if $C(\cdot) \approx F(k, \cdot)$
   (output unmarked if key extraction fails)
2. Output marked if $C(x^*) \neq F(k, x^*)$ and unmarked otherwise

Advantage: no longer require private puncturing (can base on weaker assumptions)
Extraction to Watermarking

Real PRF key

Encode($k$)
Extraction to Watermarking

To remove watermark, adversary has to “repair” the function at $x_3$ (needs to guess $y_3$)

marked key

puncture at a particular point

Encode($k$)
Extraction to Watermarking

Preventing verifier rejection: Queries on circuits that are far away from marked key will always reject, so binary search is no longer effective

To remove watermark, adversary has to “repair” the function at $x_3$ (needs to guess $y_3$)
Security Against the Authority

Encode($k$)

PRF keys are pseudorandom everywhere except at $z$ (even given the extraction trapdoor)

Implies weak pseudorandomness (more generally “$T$-restricted pseudorandomness” – pseudorandomness at all but a small number ($T$) of points)
High-level overview:

- **Marking**: Puncture PRF key at $x^*$
- **Verification**: Extract key from circuit, and check correctness of value at $x^*$

**Unremovability**: Key-extraction succeeds if circuit if adversary’s circuit is close to original PRF; removing the mark requires “patching” PRF at punctured point
Summary

Puncturable Extractable PRF

Watermarkable PRF

Property holds even in the presence of the verification oracle

Unremovability: Key-extraction succeeds if circuit if adversary’s circuit is close to original PRF; removing the mark requires “patching” PRF at punctured point
Constructing Extractable PRFs

Structure of lattice PRFs [BV15]:

PRF on $\ell$-bit inputs (e.g., domain $\{0,1\}^\ell$)

$A_1, \ldots, A_\ell \in \mathbb{Z}_{q}^{n \times m}$ public matrices (one for each bit of input)

PRF secret key: $s \in \mathbb{Z}_q^n$ (LWE secret)

\[
(A, s^T A + e^T) \approx_c (A, u)
\]

where

$A \leftarrow \mathbb{Z}_{q}^{n \times m}, \ s \leftarrow \mathbb{Z}_q^n, \ e \leftarrow \chi^m, \ u \leftarrow \mathbb{Z}_q^m$
Constructing Extractable PRFs

Structure of lattice PRFs [BV15]:

PRF on $\ell$-bit inputs (e.g., domain $\{0,1\}^\ell$)

\[ A_1, \ldots, A_\ell \in \mathbb{Z}_{q}^{n \times m} \]

public matrices (one for each bit of input)

PRF secret key: $s \in \mathbb{Z}_{q}^{n}$ (LWE secret)

PRF evaluation at input $x$: $\text{PRF}(s, x) := [s^T A_x]_p$

$A_x$: matrix derived from $A_1, \ldots, A_\ell, x$
Constructing Extractable PRFs

**Goal:** embed a trapdoor at $z$ such that evaluation at $z$ allows key recovery

Lattice trapdoors [Ajt99, GPV08, AP09, MP12]: can sample
random matrix $D \in \mathbb{Z}_{q}^{n \times m}$
trapdoor $td_{D}$
such that LWE is easy with respect to $D$:
given $s^{T}D + e^{T}$ and $td_{D}$, can recover LWE secret $s$

**Idea:** hide a lattice trapdoor in the public parameters
Constructing Extractable PRFs

\[ A_1, \ldots, A_\ell \in \mathbb{Z}_q^{n \times m} \] public matrices (one for each bit of input)

PRF secret key: \( s \in \mathbb{Z}_q^n \) (LWE secret)

PRF evaluation at input \( x \): \( \text{PRF}(s, x) := [s^T A_x]_p \)

Embed trapdoor at \( z \in \{0,1\}^\ell \):

- Compute \( A_z \) from \( A_1, \ldots, A_\ell \)
- Let \( W = D - A_z \)
- Include \( W \) in the public parameters
Constructing Extractable PRFs

\[ A_1, \ldots, A_\ell \in \mathbb{Z}_q^{n \times m} \] public matrices (one for each bit of input)

PRF secret key: \( s \in \mathbb{Z}_q^n \) (LWE secret)

PRF evaluation at input \( x \): \( \text{PRF}(s, x) := [s^T A_x]_p \)

\( W \) hides \( A_z \) (and thus, \( z \)) since 
\( D \) is statistically close to uniform

Include \( W \) in the public parameters

PRF evaluation at input \( x \): \( \text{PRF}(s, x) := [s^T (A_x + W)]_p \)
Constructing Extractable PRFs

\[ A_1, \ldots, A_\ell \in \mathbb{Z}_q^{n \times m} \] public matrices (one for each bit of input)

PRF secret key: \( s \in \mathbb{Z}_q^n \) (LWE secret)

PRF evaluation at input \( x \): \( \text{PRF}(s, x) := [s^T A_x]_p \)

Embed trapdoor at \( z \in \{0,1\}^\ell \):

- Compute \( A_z \) from \( A_1, \ldots, A_\ell \)
- Let \( \mathbf{W} = \mathbf{D} - A_z \)
- Include \( \mathbf{W} \) in the public parameters

PRF evaluation at input \( x \): \( \text{PRF}(s, x) := [s^T (A_x + \mathbf{W})]_p \)

Value everywhere else is still pseudorandom

Value at \( z \) is

\[ [s^T (A_z + \mathbf{W})]_p = [s^T \mathbf{D}]_p, \]

so can extract \( s \) using trapdoor \( \text{td}_\mathbf{D} \)
Summary

Puncturable extractable PRF can be built from LWE (with a nearly polynomial modulus-to-noise ratio)

Yields new watermarking scheme from LWE with security in the presence of verification oracle

Extensions: Message-embedding, mark-unforgeability [See paper...]
Open Problems

Extractable PRFs from generic techniques?

More applications of extractable PRFs?

Publicly-verifiable watermarking scheme for PRFs?

Thank you!