Watermarking and Traitor Tracing for PRFs

David Wu
April 2020

based on joint work with Rishab Goyal, Sam Kim, and Brent Waters
Software Watermarking

Applications: proving software ownership, preventing unauthorized distribution of software
Software Watermarking

Embed a “mark” within a program

If mark is removed, then program is destroyed

Two main algorithms:

- Mark\((C, m) \rightarrow C'\): Takes circuit \(C\) and mark \(m\) and outputs a marked circuit \(C'\)
- Extract\((C') \rightarrow m/\perp\): Extracts the mark from a circuit \(C'\)

[NS99, BGIRSVY01, HMW07, CHNVW16]
Software Watermarking

Functionality-preserving: On input a circuit $C$ (and mark $m$), the Mark algorithm outputs a circuit $C'$ where

$$C(x) = C'(x)$$
on almost all inputs $x$
Software Watermarking

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

- $C^*(x) = C'(x)$ on almost all inputs $x$
- The circuit $C^*$ does not preserve the mark: $\text{Extract}(C^*) \neq m$
Software Watermarking

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

- $C^*(x) = C'(x)$ on almost all inputs $x$
- The circuit $C^*$ does not preserve the mark: $\text{Extract}(C^*) \neq m$
Software Watermarking

- 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

```c
static void AES_enc_blk(block *blk, const AES_KEY *key) {
    unsigned j, rnds = ROUNDs(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]);
}
```
Previous works: watermarking PRFs [CHNVW16, BLW17, KW17, QWZ18, KW19]

Suffices for watermarking other symmetric primitives:
(e.g., MAC signing key, symmetric decryption key)
A Closer Look at Watermarking Security

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

- $C^*(x) = C'(x)$ on almost all inputs $x$
- The circuit $C^*$ does not preserve the mark: $\text{Extract}(C^*) \neq m$
Unremovability: Given a program $C'$ with mark $m$, no efficient adversary can construct a circuit $C^*$ where

- $C^*(x) = C'(x)$ on almost all inputs $x$
- The circuit $C^*$ does not preserve the mark: $\text{Extract}(C^*) \neq m$
A Closer Look at Watermarking Security

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

- $C^*(x) = C'(x)$ on almost all inputs $x$
- The circuit $C^*$ does not preserve the mark: $\text{Extract}(C^*) \neq m$

Adversary’s circuit does not preserve functionality
Unremovability: Given a program $C'$ with mark $m$, no efficient adversary can construct a circuit $C^*$ where

- $C^*(x) = C'(x)$ on almost all inputs $x$
- The circuit $C^*$ does not preserve the mark: $\text{Extract}(C^*) \neq m$

No guarantees on whether the mark is preserved or not!
A Closer Look at Watermarking Security

Suppose circuit that only outputs leading $n/4$ bits does not contain the watermark

Is this a problem?

For building blocks like PRFs, we do not necessarily need to recover exact output to "break" functionality

Suppose watermarkable PRF used to protect against unauthorized distribution of decryption keys

Encrypted image (PRF in counter mode)

Partial decryption (using program on left)

Adversary's program is "good enough" in most settings, but may not preserve watermark
A Closer Look at Watermarking Security

Suppose watermarkable PRF used to protect against unauthorized distribution of decryption keys

Watermarking cryptographic programs:
- Exact functionality preserving does not seem like the right security notion
- If adversary’s program can break the primitive, then watermark should be preserved

Encrypted image (PRF in counter mode)
Partial decryption (using program on left)

Adversary’s program is “good enough” in most settings, but may not preserve watermark
A Closer Look at Watermarking Security

Suppose watermarkable PRF used to protect against unauthorized distribution of decryption keys

```
PRF(k,·)
don input x:
output PRF(k, x)|_{1,...,n/4}
```

Watermarking cryptographic programs:

- Exact functionality preserving does not seem like the right security notion
- If adversary's program can break the primitive, then watermark should be preserved on input \( x \):
  
  \[
  \text{output } \text{PRF}(k, x)_{1,...,n/4}
  \]

Existing watermarking constructions are unable to recover the watermark from this type of program

Encrypted image (PRF in counter mode)

Partial decryption (using program on left)

Adversary’s program is “good enough” in most settings, but may not preserve watermark
Traceable PRFs

- **PRF security:**
  - $\text{PRF}(k, \cdot)$ indistinguishable from random function

- **Marking security (informal):**
  - If program $C$ can distinguish $\text{PRF}(k, \cdot)$ from random, then mark should be preserved
Traceable PRFs

**PRF**\((k, \cdot)\)
- on input \(x\):
- output \(\text{PRF}(k, x)\)

**Mark**

**Traitor tracing**: if program can distinguish ciphertexts, then mark is preserved

**Traceable PRF**: analog for PRFs

**Marking security (informal)**: if program \(C\) can *distinguish* \(\text{PRF}(k, \cdot)\) from random, then mark should be preserved
Traceable PRFs

Marking security (informal):
if program $C$ can distinguish $\text{PRF}(k, \cdot)$ from random, then mark should be preserved.

Problematic because $C$ could have $(x^*, \text{PRF}(k, x^*))$ hard-wired.
Traceable PRFs

Marking security (informal):
if program $C$ can distinguish $\text{PRF}(k, \cdot)$ from random
on randomly sampled inputs, then mark should be preserved

\[
x \leftarrow X \\
(x, \text{PRF}(k, x))
\]

Distinguisher can see arbitrarily many input-output pairs
Traceable PRFs

Marking security (informal):
if program $C$ can break weak pseudorandomness of PRF($k$,$\cdot$), then mark should be preserved

\[
(x, \text{PRF}(k, x)) \quad \text{or} \quad (x, f(x))
\]

Distinguisher can see arbitrarily many input-output pairs
Traceable PRFs

Setup($1^\lambda$) $\rightarrow$ (msk, tk)

KeyGen(msk, id) $\rightarrow$ sk_id

Eval(sk, x) $\rightarrow$ y

Trace^D(tk) $\rightarrow$ T $\subseteq$ {0,1}^\ell

msk: master PRF key

tk: tracing key (can be public or secret)

embeds id $\in$ {0,1}^\ell into the key

sk can be either msk or sk_id

tracing algorithm given oracle access to weak PRF distinguisher
Traceable PRFs

\[ \text{Setup}(1^\lambda) \rightarrow (\text{msk}, \text{tk}) \]

\[ \text{KeyGen}(\text{msk}, \text{id}) \rightarrow \text{sk}_\text{id} \]

\[ \text{Eval}(\text{sk}, x) \rightarrow y \]

\[ \text{Trace}^D(\text{tk}) \rightarrow T \subseteq \{0,1\}^\ell \]

- \( \text{msk} \): master PRF key
- \( \text{tk} \): tracing key (can be public or secret)

Tracing key is sampled with PRF key (tracing algorithm needs to be able to sample PRF evaluations)

- \( \text{sk} \) can be either \( \text{msk} \) or \( \text{sk}_\text{id} \)

- Tracing algorithm given oracle access to weak PRF distinguisher
Traceable PRFs

**Correctness:** marked and unmarked keys agree almost everywhere

\[
\Pr_{x \leftarrow \mathcal{X}} [\text{Eval}(\text{msk}, x) = \text{Eval}(\text{sk}_{\text{id}}, x)] = 1 - \text{negl}(\lambda)
\]

**Pseudorandomness:** \(\text{Eval}(\text{msk}, \cdot)\) is pseudorandom

**Tracing Security:**

if \(D\) breaks weak pseudorandomness of \(\text{Eval}(\text{msk}, \cdot)\) with advantage \(\varepsilon\), then \(\text{Trace}^D (\text{tk})\) outputs \(\text{id}\) with probability \(\approx \varepsilon\)
Traceable PRFs

Traceable PRF directly implies secret-key traitor tracing (via nonce-based encryption)

\[ \text{Encrypt}(k, m) := (r, \text{PRF}(k, r) \oplus m) \]

Instantiate PRF with a traceable PRF

Not the case if we start with watermarkable PRF!

**Tracing Security:**

- \( \text{id} \)
- \( \text{id} \leftarrow \text{KeyGen}(\text{msk}, \text{id}) \)
- \( D \)
- single-key setting

if \( D \) breaks weak pseudorandomness of \( \text{Eval}(\text{msk}, \cdot) \) with advantage \( \varepsilon \), then \( \text{Trace}^D(\text{tk}) \) outputs \text{id} with probability \( \approx \varepsilon \)
Traceable PRFs

Our results:

Assuming LWE, there exists a single-key traceable PRF with secret tracing

Assuming indistinguishability obfuscation and injective one-way functions, there exists a fully collusion-resistant traceable PRF with public tracing

Notably: assumptions are the same as those needed for watermarkable PRFs (and rely on similar building blocks)
Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

**Constrained PRF key:** can be used to evaluate at all points $x \in \mathcal{X}$ where $C(x) = 1$
Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

---

**PRF key**

**Constraint** $C$

**id**

**Privacy:** index associated with a domain element is hidden

---

**Linear constraint family:**

- Some PRF inputs are associated with a (secret) index $t$ between 0 and $2^\ell$
- Constrained key associated with $id \in [0, 2^\ell - 1]$ and can be used to evaluate on inputs whose index $t$ satisfies $t \leq id$ (or no index)
Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(alternate of private linear broadcast encryption from traitor tracing) [BSW06]

- **Input point** $\chi$
- **“Hidden” index** $t$
- **Key**

Can decrypt input points with indices $t \leq \text{id}$

Index $t$ (for PRF domain element)
Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

**Normal hiding:** domain element with index 0 indistinguishable from random domain elements

**Identity hiding:** domain elements with index $i$ and $j$ are indistinguishable without key for $i \leq id < j$

**Pseudorandomness:** PRF outputs on inputs with index $2^\ell$ are pseudorandom
(all of the properties should hold given constrained keys)

There exists a sampling algorithm to sample inputs with a specified index (could be secret-key algorithm)
Constructing Traceable PRFs

Tracing idea:

**Assumption:** Distinguisher $D$ can break weak pseudorandomness with advantage $\epsilon$

**Implication:** There must be a jump somewhere, and can only appear at $id$

Inputs with index 0 are indistinguishable from random inputs, so decoder has advantage $\epsilon$

Distinguishing advantage changes negligibly when $id \not\in [i, j - 1]$

Inputs with index $2^\ell$ are pseudorandom, so decoder has advantage 0
Constructing Private Linear Constrained PRF

Starting point: standard constrained PRF

Let domain $\mathcal{X} = \{0,1\}^\ell$

Problem: indices for domain element are public

\[ C_{id}(t) = \begin{cases} 0, & t > id \\ 1, & t \leq id \end{cases} \]

Can decrypt input points with tags $t \leq id$
Constructing Private Linear Constrained PRF

**Starting point:** standard constrained PRF

Let domain $\mathcal{X} = CT$ (ciphertext space for symmetric encryption scheme)

**Solution:** Encrypt indices

$C_{k, id}(ct) = \begin{cases} 0, & \text{Decrypt}(k, ct) > id \\ 1, & \text{otherwise} \end{cases}$

$k$: decryption key

Can decrypt input points corresponding to inputs that encrypt index greater than id
Constructing Private Linear Constrained PRF

**Starting point:** standard constrained PRF

Let domain $\mathcal{X} = \mathcal{C} \mathcal{T}$

**Problem:** constrained key might leak $k$ which leaks indices

$$C_{k, id}(ct) = \begin{cases} 0, & \text{Decrypt}(k, ct) > id \\ 1, & \text{otherwise} \end{cases}$$

$k$: decryption key

Can decrypt input points corresponding to inputs that encrypt index greater than $id$
Constructing Private Linear Constrained PRF

**Starting point:** standard constrained PRF

Let domain \( \mathcal{X} = C \mathcal{T} \)

**Solution:** use a private constrained PRF (constrained key hides constraint) \([\text{BLW}17, \text{CC}17]\)

\[ C_{k,\text{id}}(\text{ct}) = \begin{cases} 0, & \text{Decrypt}(k, \text{ct}) > \text{id} \\ 1, & \text{otherwise} \end{cases} \]

\( k: \text{decryption key} \)
Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

**Normal hiding:** domain element with index 0 indistinguishable from random domain elements

Holds as long as encryption scheme has pseudorandom ciphertexts
(and constrained key hides secret key)

\[
C_{k, id}(ct) = \begin{cases} 
0, & \text{Decrypt}(k, ct) > id \\
1, & \text{otherwise}
\end{cases}
\]
Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

**Identity hiding:** domain elements with index $i$ and $j$ are indistinguishable without key for $i \leq \text{id} < j$

Holds as long as encryption scheme is semantically secure
(and constrained key hides secret key)

\[
C_{k,\text{id}}(\text{ct}) = \begin{cases} 
0, & \text{Decrypt}(k, \text{ct}) > \text{id} \\
1, & \text{otherwise}
\end{cases}
\]
Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

**Pseudorandomness:** PRF outputs on inputs with index $2^\ell$ are pseudorandom

Holds by constrained security of constrained PRF
(constraint function always false if $id = 2^\ell$)

\[
C_{k,id}(ct) = \begin{cases} 
0, & \text{Decrypt}(k, ct) > id \\
1, & \text{otherwise}
\end{cases}
\]
Constructing Traceable PRFs

Rely on intermediate notion: private linear constrained PRF
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

\[ C_{k,id}(ct) = \begin{cases} 
0, & \text{Decrypt}(k, ct) > id \\ 
1, & \text{otherwise} 
\end{cases} \]

Public tracing: need a way to sample PRF evaluations (both inputs and outputs)
Unclear how to do so via private constrained PRFs, possible using indistinguishability obfuscation (with full collusion-resistance)
**Unremovability:** Any program that can distinguish PRF outputs (on random inputs) must preserve the watermark.

**More generally:** When considering software watermarking, should not always tie “functionality preserving” to “input-output preservation”
Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

\[ C_{k,\text{id}}(ct) = \begin{cases} 
0, & \text{Decrypt}(k, ct) > \text{id} \\
1, & \text{otherwise} \end{cases} \]

LWE \[ \rightarrow \] single-key private constrained PRF + symmetric encryption \[ \rightarrow \] single-key private linear constrained PRF (with secret sampling) \[ \downarrow \] single-key traceable PRF (with secret tracing)

https://eprint.iacr.org/2020/316
Private Constrained PRFs from Lattices

Overview of Brakerski-Vaikuntanathan and Brakerski-Tsabury-Vaikuntanathan-Wee constructions
Lattice-Based PRFs

[BR12, BLMR13, BP14]

Learning with errors (LWE):

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

Learning with rounding (LWR) [BR12]:

Replace error with deterministic rounding

\[\left(A, [s^T A]_p\right) \approx (A, u^T)\]
Lattice-Based PRFs

Learning with rounding (LWR) [BPR12]:
replace error with deterministic rounding

\[
\left(A, \lfloor s^T A \rfloor_p \right) \approx (A, u^T)
\]

General blueprint for lattice PRFs:

PRF family define by collection of public parameters: \(A_1, \ldots, A_\ell \in \mathbb{Z}_{q}^{n \times m}\)

PRF key: \(s \leftarrow \mathbb{Z}_{q}^{n}\)

PRF evaluation at \(x\): \(A_1, \ldots, A_\ell, x \mapsto A_x\)

\[
\text{PRF}(s, x) := \lfloor s^T A_x \rfloor_p
\]

\{ multiple ways to derive \(A_x\) from \(A_1, \ldots, A_\ell\) \}
The GSW FHE Scheme

Public key is an LWE matrix (columns are LWE samples)
\[ s^T A = e^T \approx 0^T \]

Ciphertext for \( x \in \{0,1\} \):
\[ A_x = AR + xG \] where \( R \) is random short matrix

Decryption:
\[ s^T A_x = s^T AR + x \cdot s^T G \approx x \cdot s^T G \]
The GSW/BGG$^+$ Homomorphisms

\[ A_1 = AR_1 + x_1 G \quad \cdots \quad A_\ell = AR_\ell + x_\ell G \]

Input-independent evaluation:

\[ A_1, \ldots, A_\ell, f \mapsto A_f \]

\[ A_f = AR_{f,x} + f(x)G \quad \text{where} \quad R_{f,x} = [R_1 | \cdots | R_\ell]H_{f,x} \]

and \( H_{f,x} \) is short

Input-dependent evaluation:

\[ [AR_1 | \cdots | AR_\ell]H_{f,x} = AR_{f,x} \]

\[ [A_1 - x_1 G | \cdots | A_\ell - x_\ell G]H_{f,x} = A_f - f(x)G \]
Lattice-Based Constrained PRFs

Domain: $\mathcal{X} = \{0,1\}^\rho$

Let $U_x(f) := f(x)$ be a universal circuit where $|f| = \ell$

Public parameters: $A_1, \ldots, A_\ell \leftarrow \mathbb{Z}_q^{n \times m}$

PRF key: $s \leftarrow \mathbb{Z}_q^n$

PRF evaluation at $x$:
$A_1, \ldots, A_\ell, x \mapsto A_{U_x}$

PRF($s, x$) := $[s^T A_{U_x}]_p$

Constrained key for $f$:
$s^T [A_1 - f_1 \cdot G | \cdots | A_\ell - f_\ell \cdot G] + e^T$

Constrained evaluation at $x$:
$s^T [A_1 - f_1 \cdot G | \cdots | A_\ell - f_\ell \cdot G] H_{U_x,f} + e^T H_{U_x,f}$
$\approx s^T (A_{U_x} - f(x) \cdot G)$
$= s^T A_{U_x}$ when $f(x) = 0$

can evaluate at $x$ where $f(x) = 0$

to argue pseudorandomness, need to also multiply by $G^{-1}(D)$ where $D$ is part of public parameters

[BV15]
Lattice-Based Constrained PRFs

Domain: \( \mathcal{X} = \{0,1\}^\rho \)

Let \( U_x(f) := f(x) \) be a universal circuit where \( |f| = \ell \)

Public parameters: \( A_1, \ldots, A_\ell \leftarrow \mathbb{Z}_{q \times m}^{n \times m} \)

PRF key: \( s \leftarrow \mathbb{Z}_q^n \)

PRF evaluation at \( x \):

\[
A_1, \ldots, A_\ell, x \mapsto A_{U_x}
\]

PRF \( (s, x) := [s^T A_{U_x}]_p \)

Computing \( H_{U_x,f} \) requires knowledge of \( f \)
(construction does not hide the constraint)

Constrained evaluation at \( x \):

\[
s^T [A_1 - f_1 \cdot G | \ldots | A_\ell - f_\ell \cdot G] H_{U_x,f} + e^T H_{U_x,f}
\]

\[
\approx s^T (A_{U_x} - f(x) \cdot G)
\]

\[
\approx s^T A_{U_x} \quad \text{when} \quad f(x) = 0
\]
Lattice-Based Private Constrained PRFs

[BTVW17]

**Approach:** encrypt the function $f$ using an FHE scheme, and homomorphically evaluate $U_x$

$$\hat{f} := \text{Encrypt}(pk, f) \quad |\hat{f}| = L$$

$$\tilde{U}_x(\hat{f}) := \text{FHE. Eval}(pk, U_x, \hat{f}) \quad \text{Homomorphic evaluation of } U_x \text{ on } f$$

Constrained key for $f$:

$$s^T [A_1 - \hat{f}_1 \cdot G | \cdots | A_\ell - \hat{f}_L \cdot G] + e^T$$

Constrained evaluation at $x$:

$$s^T [A_1 - \hat{f}_1 \cdot G | \cdots | A_\ell - \hat{f}_L \cdot G] H_{\tilde{U}_x, \hat{f}} + e^T H_{\tilde{U}_x, \hat{f}}$$

$$\approx s^T (A_{\tilde{U}_x} - \tilde{U}_x(\hat{f}) \cdot G)$$

**Problem:** $\tilde{U}_x(\hat{f})$ is a bit of the encryption of $f(x)$, not $f(x)$
Lattice-Based Private Constrained PRFs

Straightforward to generalize homomorphic operations to matrix-valued functions:

\[ A_1, \ldots, A_\ell, f \mapsto A_f \]

\[ [A_1 - x_1 G \mid \cdots \mid A_\ell - x_\ell G]H_{f,x} = A_f - f(x)G \]

\[ A_1, \ldots, A_\ell, f \mapsto A_f \]

\[ [A_1 - x_1 G \mid \cdots \mid A_\ell - x_\ell G]H_{f,x} = A_f - X_f \]

**Idea:** compute \( X_f \) bit-by-bit, and multiply encoding of the \( k \)th bit of the \( j \)th component of \( X_f \) by \( G^{-1}(2^k E_j) \), where \( E_j \) is 1 in the \( j \)th component and 0 everywhere else.

\[ f: \{0,1\}^\ell \mapsto b \in \{0,1\} \]

\[ f: \{0,1\}^\ell \mapsto X_f \in \mathbb{Z}_q^{n \times m} \]
Recall GSW decryption:

\[ A_x = AR + xG \]

Decryption:

\[ s^T A_x = s^T AR + x \cdot s^T G = x \cdot s^T G + \text{error} \]

**Property:** Multiplying secret key with ciphertext yields encoding of the plaintext message.
Lattice-Based Private Constrained PRFs

Approach: encrypt the function $f$ using an FHE scheme, and homomorphically evaluate $U_x$

\[
\hat{f} := \text{Encrypt}(pk, f) \quad |\hat{f}| = L
\]

\[
\tilde{U}_x(\hat{f}) := \text{FHE. Eval}(pk, U_x, \hat{f})
\]

Homomorphic evaluation of $U_x$ on $f$

Constrained key for $f$:

\[
s^T [A_1 - \hat{f}_1 \cdot G \mid \cdots \mid A_L - \hat{f}_L \cdot G] + e^T
\]

Constrained evaluation at $x$:

\[
s^T [A_1 - \hat{f}_1 \cdot G \mid \cdots \mid A_L - \hat{f}_L \cdot G]H_{\tilde{U}_x,\hat{f}} + e^T H_{\tilde{U}_x,\hat{f}}
\]

\[
\approx s^T (A_{\tilde{U}_x} - \tilde{U}_x(\hat{f}))
\]

\[
\approx s^T (A_{\tilde{U}_x} - f(x) \cdot G)
\]

Insight: If $s$ is also the secret key for the GSW encryption scheme, then

\[
s^T \tilde{U}_x(\hat{f}) = f(x) \cdot s^T G + \text{error}
\]

[BTVW17]
Lattice-Based Private Constrained PRFs

Approach: encrypt the function $f$ using an FHE scheme, and homomorphically evaluate $U_x$

$\hat{f} := \text{Encrypt}(pk, f)$ \hspace{1cm} $|\hat{f}| = L$

$\hat{U}_x(\hat{f}) := \text{FHE. Eval}(pk, U_x, \hat{f})$

Constrained key for $f$:

$s^T[A_1 - \hat{f}_1 \cdot G | \cdots | A_\ell - \hat{f}_L \cdot G] + e^T$

Constrained evaluation at $x$:

$s^T[A_1 - \hat{f}_1 \cdot G | \cdots | A_\ell - \hat{f}_L \cdot G]H_{\hat{U}_x,\hat{f}} + e^T H_{\hat{U}_x,\hat{f}}$

$\approx s^T(A_{\hat{U}_x} - \hat{U}_x(\hat{f}))$

$\approx s^T(A_{\hat{U}_x} - f(x) \cdot G)$

Defining $\hat{U}_x(\hat{f})$ to output the GSW ciphertext (matrix-valued) obtained from homomorphic evaluation

Some tweaks needed to argue security (see [BTVW17] for full details)
Summary

Input-independent evaluation:

\[ A_1, \ldots, A_\ell, f \mapsto A_f \]

Input-dependent evaluation:

\[ \left[ A_1 - x_1 G \mid \cdots \mid A_\ell - x_\ell G \right] H_{f,x} = A_f - f(x)G \]

Constraint privacy:

- Encrypt constraint using GSW FHE scheme
- LWE secret reused for PRF secret key and FHE secret key

Thank you!