Privately Constraining and Programming PRFs, the LWE Way

Chris Peikert* Sina Shiehian†

November 10, 2017

Abstract

Constrained pseudorandom functions allow for delegating "constrained" secret keys that let one compute the function at certain authorized inputs—as specified by a constraining predicate—while keeping the function value at unauthorized inputs pseudorandom. In the *constraint-hiding* variant, the constrained key hides the predicate. On top of this, *programmable* variants allow the delegator to explicitly set the output values yielded by the delegated key for a particular set of unauthorized inputs.

Recent years have seen rapid progress on applications and constructions of these objects for progressively richer constraint classes, resulting most recently in constraint-hiding constrained PRFs for *arbitrary* polynomial-time constraints from Learning With Errors (LWE) [Brakerski, Tsabary, Vaikuntanathan, and Wee, TCC'17], and privately programmable PRFs from indistinguishability obfuscation (*iO*) [Boneh, Lewi, and Wu, PKC'17].

In this work we give a unified approach for constructing both of the above kinds of PRFs from LWE with subexponential $\exp(n^\varepsilon)$ approximation factors. Our constructions follow straightforwardly from a new notion we call a *shift-hiding shiftable function*, which allows for deriving a key for the *sum* of the original function and any desired hidden shift function. In particular, we obtain the first privately programmable PRFs from non-*iO* assumptions.

*Computer Science and Engineering, University of Michigan. Email: cpeikert@umich.edu. This material is based upon work supported by the National Science Foundation under CAREER Award CCF-1054495 and CNS-1606362. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the National Science Foundation or the Sloan Foundation.

†Computer Science and Engineering, University of Michigan. Email: shiayan@umich.edu.

1 Introduction

Since the introduction of pseudorandom functions (PRFs) more than thirty years ago by Goldreich, Goldwasser, and Micali [GGM84], many variants of this fundamental primitive have been proposed. For example, *constrained* PRFs (also known as *delegatable* or *functional* PRFs) [KPTZ13, BW13, BGI14] allow issuing "constrained" keys which can be used to evaluate the PRF on an *authorized* subset of the domain, while keeping the values of the PRF on the remaining *unauthorized* inputs pseudorandom.

Assuming the existence of one-way functions, constrained PRFs were first constructed for the class of *prefix-fixing* constraints, i.e., the constrained key allows evaluating the PRF on inputs which start with a specified bit string [KPTZ13, BW13, BGI14]. Subsequently, by building on a sequence of works [BPR12, BLMR13, BP14] that gave PRFs from the Learning With Errors (LWE) problem [Reg05], Brakerski and Vaikuntanathan [BV15] constructed constrained PRFs where the set of authorized inputs can be specified by an *arbitrary* polynomial-time predicate, although for a weaker security notion that allows the attacker to obtain only a single constrained key and function value.

In the original notion of constrained PRF, the constrained key may reveal the constraint itself. Boneh, Lewi, and Wu [BLW17] proposed a stronger variant in which the constraint is hidden, calling them *privately constrained PRFs*—also known as *constraint-hiding constrained PRFs* (CHC-PRFs)—and gave several compelling applications, like searchable symmetric encryption, watermarking PRFs, and function secret sharing [BGI15]. They also constructed CHC-PRFs for arbitrary polynomial-time constraining functions under the strong assumption that indistinguishability obfuscation (*iO*) exists [BGI$^+$01, GGH$^+$13]. Soon after, CHC-PRFs for various constraint classes were constructed from more standard LWE assumptions:

- Boneh, Kim, and Montgomery [BKM17] constructed them for the class of point-function constraints (i.e., all but one input is authorized).
- Thorough a different approach, Canetti and Chen [CC17] constructed them for constraints in NC^1, i.e., polynomial-size formulas.
- Most recently, Brakerski, Tsabary, Vaikuntanathan, and Wee [BTVW17] improved on the construction from [BKM17] to support arbitrary polynomial-size constraints.

All these constructions have a somewhat weaker security guarantee compared to the *iO*-based construction of [BLW17], namely, the adversary gets only a single constrained key (but an unbounded number of function values), whereas in [BLW17] it can get unboundedly many constrained keys. Indeed, this restriction reflects a fundamental barrier: CHC-PRFs that are secure for even two constrained keys (for arbitrary constraining functions) imply *iO* [CC17].

Boneh *et al.* [BLW17] also defined and constructed what they call *privately programmable PRFs* (PP-PRFs), which are CHC-PRFs for the class of point functions along with an additional programmability property: when deriving a constrained key, one can specify the output values it yields at the unauthorized points. They showed how to use PP-PRFs to build *watermarking* PRFs, a notion defined in [CHN$^+$16]. While the PP-PRF and resulting watermarking PRF from [BLW17] were based on indistinguishability obfuscation, Kim and Wu [KW17] later constructed watermarking PRFs from LWE, though through a different route that does not require PP-PRFs. Whether PP-PRFs exist based on more standard (non-*iO*) assumptions has remained an open question.

1.1 Our Results

Our main contribution is a unified approach for constructing both constraint-hiding constrained PRFs for arbitrary polynomial-time constraints, and privately programmable PRFs, from LWE with subexponential $\exp(n^\varepsilon)$

approximation factors (i.e., inverse error rates), for any constant $\varepsilon > 0$. Both objects follow straightforwardly from a single LWE-based construction which we call a *shift-hiding shiftable function* (SHSF). Essentially, an SHSF allows for deriving a "shifted" key for a desired shift function, which remains hidden. The shifted key allows one to evaluate the *sum* of the original function and the shift function. We construct CHC-PRFs and PP-PRFs simply by using an appropriate shift function, which is zero at authorized inputs, and either pseudorandom or programmed at unauthorized inputs.

CHC-PRFs. In comparison with [BTVW17], while we achieve the same ultimate result of CHC-PRFs for arbitrary constraints (with essentially the same efficiency metrics), our construction is more modular and arguably a good deal simpler.[1] Specifically, our SHSF construction uses just a few well-worn techniques from the literature on LWE-based fully homomorphic and attribute-based encryption [GSW13, BGG$^+$14, GVW15], and we get a CHC-PRF by invoking our SHSF with an *arbitrary* PRF as the shift function. By contrast, the construction from [BTVW17] melds the FHE/ABE techniques with a specific LWE-based PRF [BP14], and involves a handful of ad-hoc techniques to deal with various technical complications that arise.

PP-PRFs. Our approach also yields the first privately programmable PRFs from LWE, or indeed, any non-*iO* assumption. In fact, our PP-PRF allows for programming any polynomial number of inputs. Previously, the only potential approach for constructing PP-PRFs without *iO* [KW17] was from CHC-PRFs having certain extra properties (which constructions prior to our work did not possess), and was limited to programming only a logarithmic number of inputs.

1.2 Techniques

As mentioned above, the main ingredient in our constructions is what we call a *shift-hiding shiftable function* (SHSF). We briefly describe its properties. We have a keyed function $\mathsf{Eval} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is some finite additive group, and an algorithm $\mathsf{Shift}(\cdot, \cdot)$ to derive *shifted keys*. Given a secret key $msk \in \mathcal{K}$ and a function $H : \mathcal{X} \to \mathcal{Y}$, we can derive a shifted key $sk_H \leftarrow \mathsf{Shift}(msk, H)$. This key has the following two main properties:

- sk_H hides the shifting function H, and
- given sk_H we can compute an *approximation* of $\mathsf{Eval}(msk, \cdot) + H(\cdot)$ at any input, i.e, there exists a "shifted evaluation" algorithm SEval such that for every $x \in \mathcal{X}$,

$$\mathsf{SEval}(sk_H, x) \approx \mathsf{Eval}(msk, x) + H(x). \tag{1.1}$$

We emphasize that the SHSF itself does not have any pseudorandomness property; this will come from "rounding" the function in our PRF constructions, described next.

CHC-PRFs and PP-PRFs. We first briefly outline how we use SHSFs to construct CHC-PRFs and PP-PRFs. To construct a CHC-PRF we instantiate the SHSF with range $\mathcal{Y} = \mathbb{Z}_q^m$ for a carefully chosen q. The CHC-PRF key is just a SHSF key msk.

- To evaluate on an input $x \in \mathcal{X}$ using msk we output $\lfloor \mathsf{Eval}(msk, x) \rceil_p$, where $\lfloor \cdot \rceil_p$ denotes (coordinate-wise) "rounding" from $\mathbb{Z}_q$ to $\mathbb{Z}_p$ for some appropriate $p \ll q$.

[1] Our construction was actually developed independently of [BTVW17], though not concurrently; we were unaware of its earlier non-public versions.

- To generate a constrained key for a constraint circuit $C : \mathcal{X} \to \{0,1\}$, we sample a key k for an ordinary PRF F, define the shift function $H_{C,k}(x) := C(x) \cdot F_k(x)$, and output the shifted key

$$sk_C \leftarrow \mathsf{Shift}(msk, H_{C,k}).$$

  Since Shift hides the circuit $H_{C,k}$, it follows that sk_C hides C.

- To evaluate on an input x using the constrained key sk_C, we output $\lfloor \mathsf{SEval}(sk_C, x) \rceil_p$.

Observe that for authorized inputs x (where $C(x) = 0$), we have $H_{C,k}(x) = 0$, so $\mathsf{SEval}(sk_C, x) \approx \mathsf{Eval}(msk, x)$ and therefore their rounded counterparts are equal with high probability. (This relies on the additional property that $\mathsf{Eval}(msk, x)$ is not to close to a "rounding border.") For unauthorized points x (where $C(x) = 1$), to see that the CHC-PRF output is pseudorandom given sk_C, notice that by Equation (1.1), the output is (with high probability)

$$\lfloor \mathsf{Eval}(msk, x) \rceil_p = \lfloor \mathsf{SEval}(sk_C, x) - H(x) \rceil_p. \tag{1.2}$$

Because F is a pseudorandom function, $H(x) = F_k(x)$ completely "randomizes" the right-hand side above.

Turning now to PP-PRFs, for simplicity consider the case where we want to program the constrained key at a single input x^* (generalizing to multiple inputs is obvious). A potential approach would be to use the same algorithms as in the above CHC-PRF, except that to program a key to output y at input x^* we define the shift function

$$H_{x^*,y}(x) = \begin{cases} y' - \mathsf{Eval}(msk, x^*) & \text{if } x = x^*, \\ \mathbf{0} & \text{otherwise,} \end{cases}$$

where $y' \in \mathbb{Z}_q^m$ is chosen uniformly conditioned on $\lfloor y' \rceil_p = y$. As before, the programmed key is just the shifted key $sk_{x^*,y} \leftarrow \mathsf{Shift}(msk, H_{x^*,y})$. By Equation (1.1), evaluating on the unauthorized input x^* using $sk_{x^*,y}$ indeed yields $\lfloor y' \rceil_p = y$. However, it is unclear whether the actual (non-programmed) value of the PP-PRF at the unauthorized input $x = x^*$ is pseudorandom given $sk_{x^*,y}$.

To address this issue, we observe that the above construction satisfies a weaker pseudorandomness guarantee: if the adversary does not specify y but instead y is uniformly random, then by Equation (1.2) the PP-PRF is pseudorandom at x^*. This observation leads us to our actual PP-PRF construction: we instantiate two of the above "weak" PP-PRFs with keys msk_1 and msk_2. To generate a programmed key for input x^* and output y, we first generate random additive secret shares y_1, y_2 such that $y = y_1 + y_2$, and output the programmed key $sk_{x^*,y} := (sk_{x^*,y_1}, sk_{x^*,y_2})$ where $sk_{x^*,y_i} \leftarrow \mathsf{Shift}(msk_i, H_{x^*,y_i})$ for $i = 1, 2$. Each evaluation algorithm (ordinary and programmed) is then defined simply as the sum of the corresponding evaluation algorithm from the "weak" construction using the two component keys. Because both programmed keys are generated for random target outputs y_i, we can prove pseudorandomness of the real function value using weak pseudorandomness.

Constructing SHSFs. We now give an overview of our construction of shift-hiding shifted functions. For simplicity, suppose the range of the functions is $\mathcal{Y} = \mathbb{Z}_q$; extending this to $\mathbb{Z}_q^m$ (as in our actual constructions) is straightforward. As in [KW17, BKM17] our main tools are the "gadget-matrix homomorphisms" developed in the literature on fully homomorphic and attribute-based encryption [GSW13, BGG+14, GVW15].

At a high level, our SHSF works as follows. The secret key is just an LWE secret $\mathbf{s}$ whose first coordinate is 1. A shifted key for a shift function $H\colon \mathcal{X} \to \mathbb{Z}_q$ consists of LWE vectors (using secret $\mathbf{s}$) relative to some public matrices that have been "shifted" by multiples of the gadget matrix $\mathbf{G}$ [MP12]; more specifically, the multiples are the bits of FHE ciphertexts encrypting H, and the $\mathbb{Z}_q$-entries of the FHE secret key sk. To compute the shifted function on an input x, we do the following:

1. Using the gadget homomorphisms for boolean gates [GSW13, BGG$^+$14] on the LWE vectors corresponding to the FHE encryption of H, we compute LWE vectors relative to some publicly computable matrices, but shifted by multiples of $\mathbf{G}$ corresponding to the bits of an FHE ciphertext encrypting $H(x)$.

2. Then, using the gadget homomorphisms for hidden linear functions [GVW15] with the LWE vectors corresponding to the FHE secret key, we compute LWE vectors relative to some publicly computable matrix $\mathbf{B}_x$, but shifted by $(H(x)+e)\mathbf{G}$ where $H(x)+e \approx H(x) \in \mathbb{Z}_q$ is the "noisy plaintext" arising as the inner product of the FHE ciphertext and secret key. Taking just the first column, we therefore have an LWE sample relative to some vector $\mathbf{b}_x + (H(x)+e)\mathbf{u}_1$, where $\mathbf{u}_1$ is the first standard basis (column) vector.

3. Finally, because the first coordinate of the LWE secret $\mathbf{s}$ is 1, the above LWE sample is simply $\langle \mathbf{s}, \mathbf{b}_x \rangle + H(x) + e \approx \langle \mathbf{s}, \mathbf{b}_x \rangle + H(x) \in \mathbb{Z}_q$.

With the above in mind, we then define the (unshifted) function itself on an input x to simply compute $\mathbf{b}_x$ from the public parameters as above, and output $\langle \mathbf{s}, \mathbf{b}_x \rangle$. This yields Equation (1.1).

2 Preliminaries

We denote row vectors by lower-case bold letters, e.g., $\mathbf{a}$. We denote matrices by upper-case bold letters, e.g., $\mathbf{A}$. The Kronecker product $\mathbf{A} \otimes \mathbf{B}$ of two matrices (or vectors) $\mathbf{A}$ and $\mathbf{B}$ is obtained by replacing each entry $a_{i,j}$ of $\mathbf{A}$ with the block $a_{i,j}\mathbf{B}$.

2.1 Homomorphic Encryption

We use the GSW (leveled) fully homomorphic encryption scheme [GSW13] $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Eval})$, whose relevant properties for our needs are summarized as follows:

- $\mathsf{KG}(1^\lambda, q)$, given a security parameter λ and a requested modulus q, outputs a secret key sk, which is a vector in $\mathbb{Z}_q^\tau$ (for some $\tau = \mathrm{poly}(\lambda, \log q)$).

- $\mathsf{Enc}(sk, m)$, given a secret key sk and a message $m \in \{0,1\}$, outputs a ciphertext ct, which is treated as a binary string.

- $\mathsf{Eval}(H, ct_1, \ldots, ct_\ell, i)$, given a boolean circuit $H\colon \{0,1\}^\ell \to \{0,1\}$, ciphertexts $ct_1, ct_2, \ldots, ct_\ell$ for arbitrary ℓ, and some $i \in \{0, \ldots, \lceil \lg q \rceil - 1\}$, outputs a ciphertext $ct \in \{0,1\}^\tau$.

Notice that in the above definition there is no explicit decryption algorithm. Instead we capture the essential "noisy decryption" relation between the secret key and a homomorphically evaluated ciphertext, as follows: for any $sk \leftarrow \mathsf{KG}(1^\lambda, q)$, any ℓ-input boolean circuit H of depth at most d, any messages $m_j \in \{0,1\}$ and ciphertexts $ct_j \leftarrow \mathsf{Enc}(sk, m_j)$ for $j = 1, \ldots, \ell$, and any $i \in \{0, \ldots, \lceil \lg q \rceil - 1\}$ we have

$$\langle sk, \mathsf{Eval}(H, ct_1, \ldots, ct_\ell, i) \rangle = 2^i \cdot H(m_1, \ldots, m_\ell) + e \pmod q \tag{2.1}$$

for some integer $e \in [-B, B]$, for some $B = \lambda^{O(d)}$. In other words, we can homomorphically evaluate any circuit of bounded depth times any desired power of two.

2.2 Gadget Homomorphisms

We use some important homomorphic properties of the "gadget" matrix [MP12] over $\mathbb{Z}_q$, many of which were implicit in [GSW13], and were developed and exploited further in [BGG+14, GVW15]. The gadget matrix is defined as

$$\mathbf{G}_n = (1, 2, 4, \ldots, 2^{\lceil \lg q \rceil - 1}) \otimes \mathbf{I}_n = [\mathbf{I}_n \mid 2\mathbf{I}_n \mid \cdots \mid 2^{\lceil \lg q \rceil - 1}\mathbf{I}_n] \in \mathbb{Z}_q^{n \times m},$$

where $m = n\lceil \lg q \rceil$. We often drop the subscript n when it is clear from context. We use algorithms $\mathsf{BoolEval}$ and $\mathsf{LinEval}$, which have the following properties. (All matrices are in $\mathbb{Z}_q^{n \times m}$, unless otherwise stated.)

- $\mathsf{BoolEval}(H, x, \mathbf{A})$, given a boolean circuit $H\colon \{0,1\}^\ell \to \{0,1\}$ of depth d, an $x \in \{0,1\}^\ell$, and some $\mathbf{A} \in \mathbb{Z}_q^{n \times (\ell+1)m}$, outputs an integral matrix $\mathbf{R}_{H,x} \in \mathbb{Z}^{(\ell+1)m \times m}$ with $m^{O(d)}$-bounded entries for which

$$(\mathbf{A} + (1, x) \otimes \mathbf{G}) \cdot \mathbf{R}_{H,x} = \mathbf{A}_H + H(x) \cdot \mathbf{G}, \tag{2.2}$$

  where $\mathbf{A}_H$ depends only on $\mathbf{A}$ and H (and not on x).

- $\mathsf{LinEval}(x, \mathbf{C})$, given a binary string $x \in \{0,1\}^\ell$ and a matrix $\mathbf{C} \in \mathbb{Z}_q^{n \times \ell m}$, outputs an integral matrix $\mathbf{R}_x \in \mathbb{Z}^{2\ell m \times m}$ with $\mathrm{poly}(m, \ell)$-bounded entries such that, for all matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$ and all $y \in \mathbb{Z}_q^\ell$,

$$[\mathbf{A} + x \otimes \mathbf{G} \mid \mathbf{C} + y \otimes \mathbf{G}] \cdot \mathbf{R}_x = \mathbf{B} + \langle x, y \rangle \cdot \mathbf{G}, \tag{2.3}$$

  where $\mathbf{B}$ depends only on $\mathbf{A}$ and $\mathbf{C}$ (and not on x or y).

2.3 Learning With Errors

For a positive integer dimension n and modulus q, and an error distribution χ over $\mathbb{Z}$, the LWE distribution and decision problem are defined as follows. For an $\mathbf{s} \in \mathbb{Z}^n$, the LWE distribution $A_{\mathbf{s},\chi}$ is sampled by choosing a uniformly random $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ and an error term $e \leftarrow \chi$, and outputting $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e) \in \mathbb{Z}_q^{n+1}$.

Definition 2.1. The decision-LWE$_{n,q,\chi}$ problem is to distinguish, with non-negligible advantage, between any desired (but polynomially bounded) number of independent samples drawn from $A_{\mathbf{s},\chi}$ for a single $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and the same number of *uniformly random* and independent samples over $\mathbb{Z}_q^{n+1}$.

In this work we use a form of LWE where the first coordinate of the secret vector $\mathbf{s}$ is 1, i.e. $\mathbf{s} = (1, \bar{\mathbf{s}})$ where $\bar{\mathbf{s}} \leftarrow \mathbb{Z}_q^{n-1}$. It is easy to see that this is equivalent to LWE with an $(n-1)$-dimensional secret: the transformation mapping $(\mathbf{a}, b) \in \mathbb{Z}_q^{n-1} \times \mathbb{Z}_q$ to $((r, \mathbf{a}), b + r)$ for a uniformly random $r \in \mathbb{Z}_q$ (chosen freshly for each sample) maps samples from $A_{\bar{\mathbf{s}},\chi}$ to samples from $A_{\mathbf{s},\chi}$, and maps uniform samples to uniform samples.

A standard instantiation of LWE is to let χ be a *discrete Gaussian* distribution (over $\mathbb{Z}$) with parameter $r = 2\sqrt{n}$. A sample drawn from this distribution has magnitude bounded by, say, $r\sqrt{n} = \Theta(n)$ except with probability at most 2^{-n}. For this parameterization, it is known that LWE is at least as hard as *quantumly* approximating certain "short vector" problems on n-dimensional lattices, in the worst case, to within $\tilde{O}(q\sqrt{n})$ factors [Reg05, PRS17]. Classical reductions are also known for different parameterizations [Pei09, BLP+13].

2.4 One Dimensional Rounded Short Integer Solution

As in [BV15, BKM17, KW17] we make use of a special "one-dimensional, rounded" variant of the short integer solution problem (SIS). For the parameters we will use, this problem is actually no easier to solve than LWE is, but it is convenient to define it separately.

Definition 2.2 (1D-R-SIS [BV15, BKM17]). Let $p \in \mathbb{N}$ and let $p_1 < p_2 < \cdots < p_k$ be pairwise coprime and coprime with p. Let $q = p \cdot \prod_{i=1}^{k} p_i$. Then for positive numbers $m \in \mathbb{N}$ and B, the 1D-R-SIS$_{m,p,q,B}$ problem is as follows: given a uniformly random vector $\mathbf{v} \leftarrow \mathbb{Z}_q^m$, find $\mathbf{z} \in \mathbb{Z}^m$ such that $\|\mathbf{z}\| \leq B$ and

$$\langle \mathbf{v}, \mathbf{z} \rangle \in \frac{q}{p}(\mathbb{Z} + \tfrac{1}{2}) + [-B, B].$$

For sufficiently large $p_1 \geq B \cdot \mathrm{poly}(k, \log q)$, solving 1D-R-SIS is at least as hard as approximating certain "short vector" problems on k-dimensional lattices, in the worst case, to within certain $B \cdot \mathrm{poly}(k)$ factors [Ajt96, MR04, BV15, BKM17].

3 Shift-Hiding Shiftable Functions

Here we present our construction of what we call *shift-hiding shiftable functions*, which we will use in our subsequent constructions of CHC-PRFs and PP-PRFs.

3.1 Notation

Let $\mathsf{GSW} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Eval})$ denote the GSW fully homomorphic encryption scheme (Section 2.1), where a fresh ciphertext encrypting a single bit is in $\{0,1\}^z$ for some $z = z(\lambda)$, and the secret key is in $\mathbb{Z}_q^\tau$ for some $\tau = \tau(\lambda)$. Recall that the output of GSW homomorphic evaluation is in $\{0,1\}^\tau$.

Let $U(H, x)$ denote a universal circuit for circuits H of size at most σ that map $\{0,1\}^\iota$ to $\mathbb{Z}_q^m$, where the circuit output is actually represented in binary, and let $U_x(\cdot) = U(\cdot, x)$. For every $\ell \in [m]$ define

$$U_{x,\ell}(H) = U_x(H)_\ell \in \mathbb{Z}_q,$$

i.e., the ℓth symbol of $U_x(H)$. For $i \in \{0, \ldots, \lceil \lg q \rceil - 1\}$ define $U_{x,\ell,i}(x) \in \{0,1\}$ to be the ith bit in the binary representation of $U_{x,\ell}(H)$, so that

$$U_{x,\ell}(H) = \sum_i 2^i \cdot U_{x,\ell,i}(C). \tag{3.1}$$

We now define homomorphic analogues of the above functions. Let $\overline{z} = z \cdot \sigma$ be the total length of fresh ciphertexts encrypting a function of size σ. For $x \in \{0,1\}^\iota$ and $i \in \{0, \ldots, \lceil \lg q \rceil - 1\}$ define $\overline{U}_{x,\ell,i} \colon \{0,1\}^{\overline{z}} \to \{0,1\}^\tau$ as

$$\overline{U}_{x,\ell,i}(ct) = \mathsf{GSW.Eval}(U_{x,\ell,i}, ct, i), \tag{3.2}$$

i.e., the homomorphic evaluation of $2^i \cdot U_{x,\ell,i}$ on an FHE-encrypted function. Observe that $\overline{U}_{x,\ell,i}$ has size (and hence depth) $\mathrm{poly}(\lambda, \sigma)$. Define

$$\overline{U}_{x,\ell}(ct) = [\overline{U}_{x,\ell,0}(ct) \mid \overline{U}_{x,\ell,1}(ct) \mid \cdots \mid \overline{U}_{x,\ell,\lceil \lg q \rceil - 1}(ct)] \in \{0,1\}^{\tau \cdot \lceil \lg q \rceil}.$$

Finally, for each $j \in [\tau]$ define $\overline{U}_{x,\ell,i,j}(ct) = \overline{U}_{x,\ell,i}(ct)_j$.

3.2 Construction

Here we give the tuple of algorithms $\mathsf{SHSF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Shift}, \mathsf{SEval}, \mathcal{S})$ that make up our SHSF. For security parameter λ and constraint circuit size σ the algorithms are parameterized by some $n = \mathrm{poly}(\lambda, \sigma)$ and $q = 2^{\mathrm{poly}(\lambda,\sigma)}$, with $m = n\lceil \lg q \rceil = \mathrm{poly}(\lambda, \sigma)$; we instantiate these more precisely in Section 3.4 below.

Construction 3.1. Let $\mathcal{X} = \{0,1\}^\iota$ and $\mathcal{Y} = \mathbb{Z}_q^m$. Define:

- $\mathsf{Setup}(1^\lambda, 1^\sigma)$: Sample uniformly random and independent matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times (\overline{z}+1)m}$ and $\mathbf{C} \in \mathbb{Z}_q^{n \times \tau m}$, and output $pp = (\mathbf{A}, \mathbf{C})$.
- $\mathsf{KeyGen}(pp)$: Sample $\mathbf{s}' \leftarrow \mathbb{Z}_q^{n-1}$ and set $\mathbf{s} = (1, \mathbf{s}')$. Output the master secret key $msk = \mathbf{s}$.
- $\mathsf{Eval}(pp, msk, x \in \{0,1\}^\iota)$: for each $\ell \in [m]$, $i \in \lceil \lg q \rceil$, and $j \in [\tau]$ compute

$$\mathbf{R}_{x,\ell,i,j} \leftarrow \mathsf{BoolEval}(\overline{U}_{x,\ell,i,j}, 0^{\overline{z}}, \mathbf{A})$$

  and let

$$\mathbf{A}_{x,\ell,i,j} = (\mathbf{A} + (1, 0^{\overline{z}}) \otimes \mathbf{G}) \cdot \mathbf{R}_{x,\ell,i,j} - \overline{U}_{x,\ell,i,j}(0^{\overline{z}}) \cdot \mathbf{G}.$$

  Then for each $\ell \in [m]$ compute:

$$\mathbf{R}_{x,\ell} \leftarrow \mathsf{LinEval}(\overline{U}_{x,\ell}(0^{\overline{z}}), [\underbrace{\mathbf{C} \mid \cdots \mid \mathbf{C}}_{\lceil \lg q \rceil \text{ times}}])$$

  and let

$$\mathbf{B}_{x,\ell} = (\mathbf{V}_{x,\ell} + (\overline{U}_{x,\ell}(0^{\overline{z}}), 0^{\tau \cdot \lceil \lg q \rceil}) \otimes \mathbf{G}) \cdot \mathbf{R}_{x,\ell},$$

  where $\mathbf{V}_{x,\ell} = [\mathbf{A}_{x,\ell,0,1} \mid \mathbf{A}_{x,\ell,0,2} \mid \cdots \mid \mathbf{A}_{x,\ell,\lceil \lg q \rceil - 1,\tau} \mid \underbrace{\mathbf{C} \mid \cdots \mid \mathbf{C}}_{\lceil \lg q \rceil \text{ times}}]$. Finally, output

$$\mathbf{s}\Big(\sum_{\ell \in [m]} \mathbf{B}_{x,\ell} \cdot \mathbf{U}_{1,\ell}\Big),$$

  where $\mathbf{U}_{1,\ell}$ has 1 in the $(1,\ell)$th entry and zeros elsewhere.

- $\mathsf{Shift}(pp, msk, H)$: On input a function $H\colon \{0,1\}^\iota \to \mathbb{Z}_q^m$ of size at most σ, sample a key for the homomorphic encryption scheme $sk \leftarrow \mathsf{GSW.KG}(1^\lambda, q)$, then encrypt H bit-by-bit under this key to obtain the ciphertext

$$ct \leftarrow \mathsf{GSW.Enc}_{sk}(H).$$

  Next, for $i \in [\overline{z}]$ and $j \in [\tau]$ define vectors

$$\mathbf{a} = \mathbf{s}(\mathbf{A} + (1, ct) \otimes \mathbf{G}) + \mathbf{e}$$
$$\mathbf{c} = \mathbf{s}(\mathbf{C} + sk \otimes \mathbf{G}) + \mathbf{e}'$$

  where $\mathbf{e}$ and $\mathbf{e}'$ are error vectors whose entries are sampled independently from χ. Output

$$sk_H = (ct, \mathbf{a}, \mathbf{c}).$$

- $\mathsf{SEval}(pp, sk_H, x)$: On input $sk_H = (ct, \mathbf{a}, \mathbf{c})$ and $x \in \{0,1\}^\iota$, first compute for each $\ell \in [m]$, $i \in \lceil \lg q \rceil$, and $j \in [\tau]$:

$$\mathbf{R}_{x,\ell,i,j} = \mathsf{BoolEval}(\overline{U}_{x,\ell,i,j}, ct, \mathbf{A}).$$

and set

$$\mathbf{a}_{x,\ell,i,j} := \mathbf{a} \cdot \mathbf{R}_{x,\ell,i,j}. \tag{3.3}$$

Then for each $\ell \in [m]$ compute

$$\mathbf{R}_{x,\ell} = \mathsf{LinEval}(\overline{H}_{x,\ell}(ct), [\underbrace{\mathbf{C} \mid \cdots \mid \mathbf{C}}_{\lceil \lg q \rceil \text{ times}}])$$

and set

$$\mathbf{b}_{x,\ell} := \mathbf{v}_{x,\ell} \cdot \mathbf{R}_{x,\ell}, \tag{3.4}$$

where $\mathbf{v}_{x,\ell} = [\mathbf{a}_{x,\ell,0,1} \mid \mathbf{a}_{x,\ell,0,2} \mid \cdots \mid \mathbf{a}_{x,\ell,\lceil \lg q \rceil - 1,\tau} \mid \underbrace{\mathbf{c} \mid \cdots \mid \mathbf{c}}_{\lceil \lg q \rceil \text{ times}}]$. Finally, output

$$\sum_{\ell \in [m]} \mathbf{b}_{x,\ell} \cdot \mathbf{U}_{1,\ell}.$$

- $\mathcal{S}(1^\lambda, 1^\sigma)$: Sample a GSW secret key $sk' \leftarrow \mathsf{GSW.KG}(1^\lambda, q)$ and compute (by encrypting bit-by-bit) $ct \leftarrow \mathsf{GSW.Enc}_{sk'}(0^\sigma)$. Sample uniformly random and independent $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times (\overline{z}+1)m}, \mathbf{a} \leftarrow \mathbb{Z}_q^{(\overline{z}+1)m}$, $\mathbf{C} \leftarrow \mathbb{Z}_q^{n \times \tau m}, \mathbf{c} \leftarrow \mathbb{Z}_q^{\tau m}$. Output $pp = (\mathbf{A}, \mathbf{C})$ and $sk = (ct, \mathbf{a}, \mathbf{c})$.

3.3 Properties

Here we prove the three main properties of our SHSF that we will use in subsequent sections.

procedure $\mathsf{RealKey}_{\mathcal{A}}(1^\lambda, 1^\rho)$
$H \leftarrow \mathcal{A}(1^\lambda, 1^\sigma)$
$pp \leftarrow \mathsf{Setup}(1^\lambda, 1^\rho)$
$msk \leftarrow \mathsf{KeyGen}(pp)$
$sk \leftarrow \mathsf{Shift}(pp, msk, H)$
$(pp, sk) \rightarrow \mathcal{A}$

(a) The real shifted key generation experiment

procedure $\mathsf{IdealKey}_{\mathcal{A}}(1^\lambda, 1^\sigma)$
$H \leftarrow \mathcal{A}(1^\lambda, 1^\sigma)$
$(pp, sk) \leftarrow \mathcal{S}(1^\lambda, 1^\sigma)$
$(pp, sk) \rightarrow \mathcal{A}$

(b) The random key generation experiment

Figure 1: The real and random shifted key generation experiments.

Lemma 3.2 (Shift Hiding). *Assuming the hardness of $\mathrm{LWE}_{n-1,q,\chi}$ and CPA security of the GSW encryption scheme, for any PPT $\mathcal{A}$ and any $\sigma = \sigma(\lambda) = \mathrm{poly}(\lambda)$,*

$$\{\mathsf{RealKey}_{\mathcal{A}}(1^\lambda, 1^\sigma)\}_{\lambda \in \mathbb{N}} \overset{c}{\approx} \{\mathsf{RandomKey}_{\mathcal{A}}(1^\lambda, 1^\sigma)\}_{\lambda \in \mathbb{N}}, \tag{3.5}$$

where RealKey *and* IdealKey *are the respective views of $\mathcal{A}$ in the experiments defined in Figure 1.*

Proof. Let $\mathcal{A}$ be any polynomial-time adversary. To show that Equation (3.5) holds we define a sequence of hybrid experiments and show that they are indistinguishable.

Hybrid H_0: This is the experiment $\mathsf{RealKey}$.

Hybrid H_1: This is the same as the previous experiment, except that we modify how the $\mathbf{A}$ and $\mathbf{C}$ are constructed as follows: after we generate ct and sk we choose uniformly random $\mathbf{A}'$ and $\mathbf{C}'$ and set

$$\mathbf{A} = \mathbf{A}' - (1, ct) \otimes \mathbf{G}$$
$$\mathbf{C} = \mathbf{C}' - sk \otimes \mathbf{G}.$$

Hybrid H_2: This is the same as H_1, except that we sample the $\mathbf{a}_i$ and $\mathbf{c}_j$ uniformly at random from $\mathbb{Z}_q^m$.

Hybrid H_3: This is the same as H_2, except that ct encrypts zeros instead of H, i.e., we set $ct \leftarrow \mathsf{GSW.Enc}_{sk'}(0^\rho)$.

Hybrid H_4: This is the same as H_3, except that the $\mathbf{A}$ and $\mathbf{C}$ are chosen uniformly at random. Observe that this is exactly the experiment $\mathsf{IdealKey}$.

Claim 3.3. *H_0 and H_1 are identical.*

Proof. This is because the $\mathbf{A}$ and $\mathbf{C}$ are uniformly random and independent of everything else in both experiments. □

Claim 3.4. *Assuming the hardness of $\mathsf{LWE}_{n-1,q,\chi}$, we have $H_1 \overset{c}{\approx} H_2$.*

Proof. We use any adversary $\mathcal{A}$ that attempts to distinguish H_1 from H_2 to build an adversary $\mathcal{A}'$ that solves $\mathsf{LWE}_{n-1,q,\chi}$ with the same advantage. First, $\mathcal{A}'$ receives samples $(\mathbf{A}', \mathbf{a}) \in \mathbb{Z}_q^{n \times (\overline{z}+1)m} \times \mathbb{Z}_q^{(\overline{z}+1)m}$ and $(\mathbf{C}', \mathbf{c}) \in \mathbb{Z}_q^{n \times \tau m} \times \mathbb{Z}_q^{\tau m}$, then proceeds exactly as in H_1 to interact with $\mathcal{A}$, and outputs what $\mathcal{A}$ outputs. If the samples are LWE samples from $A_{\mathbf{s},\chi}$ where $\mathbf{s} = (1, \mathbf{s}')$ for $\mathbf{s}' \leftarrow \mathbb{Z}_q^{n-1}$, then

$$\mathbf{a} = \mathbf{s} \cdot \mathbf{A}' + \mathbf{e} = \mathbf{s}(\mathbf{A} + (1, ct) \otimes \mathbf{G}) + \mathbf{e}$$
$$\mathbf{c} = \mathbf{s} \cdot \mathbf{C}' + \mathbf{e}' = \mathbf{s}(\mathbf{C} + sk \otimes \mathbf{G}) + \mathbf{e}'$$

for error vectors $\mathbf{e}, \mathbf{e}'$ whose entries are drawn from χ, therefore $\mathcal{A}$'s view is identical to its view in H_1. If the samples are uniformly random, then $\mathcal{A}$'s view is identical to its view in H_2. This proves the claim. □

Claim 3.5. *If GSW is CPA-secure then $H_2 \overset{c}{\approx} H_3$.*

Proof. This follows immediately from the fact that the GSW secret key $sk' \leftarrow \mathsf{GSW.KG}(1^\lambda, q)$ is used only to encrypt H (yielding ct) or zeros, respectively, in H_2 and H_3. □

Claim 3.6. *H_3 and H_4 are identical.*

Proof. This follows from the fact that the $\mathbf{A}$ and $\mathbf{C}$ are uniformly random and independent of everything else in both games. □

This completes the proof of Lemma 3.2. □

Lemma 3.7 (Border Avoiding). *For any PPT $\mathcal{A}$, $i \in [m]$, $\lambda \in \mathbb{N}$ and $\sigma = \mathrm{poly}(\lambda)$, assuming the hardness of 1D-R-SIS$_{(\overline{z}+\tau+1)m,p,q,B}$ for some large enough $B = m^{\mathrm{poly}(\lambda,\sigma)} = \lambda^{\mathrm{poly}(\lambda)}$, we have*

$$\Pr_{\substack{(pp,sk) \leftarrow \mathcal{S}(1^\lambda, 1^\sigma) \\ x \leftarrow \mathcal{A}(pp,sk)}} \left[\mathsf{Eval}(pp, sk, x)_i \in \tfrac{q}{p}(\mathbb{Z} + \tfrac{1}{2}) + [-B, +B] \right] \leq \mathrm{negl}(\lambda).$$

Proof. We show how to use an adversary which finds an $x \in \mathcal{X}$ such that

$$\mathsf{SEval}(pp, sk, x)_i \in \tfrac{q}{p}(\mathbb{Z} + \tfrac{1}{2}) + [-B, +B] \tag{3.6}$$

for some $i \in [\ell]$ to solve 1D-R-SIS.

Given a (uniformly random) 1D-R-SIS$_{(\overline{z}+\tau+1)m,p,q,B}$ challenge $\mathbf{v} = (\mathbf{a}, \mathbf{c}) \in \mathbb{Z}_q^{(\overline{z}+1)m \times \tau m}$, we put $\mathbf{a}, \mathbf{c}$ in the sk given to $\mathcal{A}$, and generate pp in the same way as in the $\mathcal{S}$ algorithm. Let x be a query output by $\mathcal{A}$ and let

$$\mathbf{y}_x = \mathsf{SEval}(pp, (ct, \mathbf{a}, \mathbf{c})), x).$$

We have $\mathbf{y}_x = \sum_{\ell \in [m]} \mathbf{b}_{x,\ell} \cdot \mathbf{U}_{1,\ell}$ for the $\mathbf{b}_{x,\ell}$ as computed by SEval. Observe that because we use $\mathsf{BoolEval}$ on a circuit of depth $\mathrm{poly}(\lambda, \sigma)$, by the properties from Section 2.2, for each $\ell \in [m]$ we can efficiently compute a matrix $\mathbf{R}'_{x,\ell}$ with B-bounded entries such that $\mathbf{b}_{x,\ell} = \mathbf{v} \cdot \mathbf{R}'_{x,\ell}$, thus $\mathbf{y}_x = \mathbf{v} \cdot (\sum_{\ell \in [m]} \mathbf{R}'_{x,\ell} \cdot \mathbf{U}_{1,\ell})$. Now if Equation (3.6) holds for some $i \in [\ell]$ then

$$(\mathbf{y}_x)_i \in \tfrac{q}{p}(\mathbb{Z} + \tfrac{1}{2}) + [-B, B],$$

which means that the ith column of

$$\sum_{\ell \in [m]} \mathbf{R}'_{x,\ell} \cdot \mathbf{U}_{1,\ell},$$

i.e., the first column of $\mathbf{R}'_{x,i}$, is a valid 1D-R-SIS$_{(\overline{z}+\tau+1)m,p,q,B}$ solution to the challenge $\mathbf{v}$. □

Lemma 3.8 (Approximate Shift Correctness). *For any function $H\colon \{0,1\}^\iota \to \mathbb{Z}_q^m$ of size at most σ, input $x \in \{0,1\}^\iota$, $pp \leftarrow \mathsf{Setup}(1^\lambda, 1^\rho)$, $msk \leftarrow \mathsf{KeyGen}(pp)$ and $sk_H \leftarrow \mathsf{Shift}(pp, msk, H)$, we have*

$$\mathsf{SEval}(pp, sk_H, x) \approx \mathsf{Eval}(pp, msk, x) + H(x)$$

where the approximation hides some $\lambda^{\mathrm{poly}(\lambda)}$-bounded error vector.

Proof. First, observe that for each ℓ, i, j, by definition of the $\mathbf{a}_{x,\ell,i,j}$ (Equation (3.3)) and Equation (2.2), we have

$$\mathbf{a}_{x,\ell,i,j} \approx \mathbf{s}(\mathbf{A}_{x,\ell,i,j} + \overline{U}_{x,\ell,i,j}(ct) \cdot \mathbf{G}),$$

where the approximation hides an error vector bounded by $m^{\mathrm{poly}(\lambda,\sigma)} = \lambda^{\mathrm{poly}(\lambda)}$. Similarly, by definition of the $\mathbf{b}_{x,\ell}$ (Equation (3.4)) and Equation (2.3) we have

$$\mathbf{b}_{x,\ell} \approx \mathbf{s}\Big(\mathbf{B}_{x,\ell} + \sum_{i=0}^{\lceil \lg q \rceil - 1} \langle sk, \overline{U}_{x,\ell,i}(ct) \rangle \cdot \mathbf{G}\Big),$$

where the approximation hides some $\lambda^{\mathrm{poly}(\lambda)}$-bounded error vector. Then by definition of $\overline{U}_{x,\ell,i}$ (Equation (3.2)), the GSW noisy decryption relation (Equation (2.1)), and Equation (3.1), for some $\lambda^{\mathrm{poly}(\lambda)}$-bounded error terms $\overline{e}_{x,\ell,i}$ and a $\lambda^{\mathrm{poly}(\lambda)}$-bounded term $\overline{e}_{x,\ell}$ we have

$$\begin{aligned}
\mathbf{b}_{x,\ell} &\approx \mathbf{s}\Big(\mathbf{B}_{x,\ell} + \sum_{i=0}^{\lceil \lg q \rceil - 1} (2^i \cdot U_{x,\ell,i}(H) + \overline{e}_{x,\ell,i}) \cdot \mathbf{G}\Big) \\
&= \mathbf{s}\Big(\mathbf{B}_{x,\ell} + (U_{x,\ell}(H) + \overline{e}_{x,\ell}) \cdot \mathbf{G}\Big) \\
&= \mathbf{s}\Big(\mathbf{B}_{x,\ell} + (H(x)_\ell + \overline{e}_{x,\ell}) \cdot \mathbf{G}\Big),
\end{aligned}$$

where again the approximation hides a $\lambda^{\mathrm{poly}(\lambda)}$-bounded error vector. Because the first column of $\mathbf{G}$ is $\mathbf{u}_1^t$ and the first coordinate of $\mathbf{s}$ is 1, for the same approximation error bound we have

$$\mathbf{b}_{x,\ell} \cdot \mathbf{U}_{1,\ell} \approx \mathbf{s} \cdot \mathbf{B}_{x,\ell} \cdot \mathbf{U}_{1,\ell} + (H(x)_\ell + \overline{e}_{x,\ell}) \cdot \mathbf{u}_\ell,$$

where $\mathbf{u}_\ell$ is the ℓth standard basis vector. Hence, by summing over all $\ell \in [m]$ we have

$$\sum_{\ell \in [m]} \mathbf{b}_{x,\ell} \cdot \mathbf{U}_{1,\ell} = \mathbf{s} \sum_{\ell \in [m]} \mathbf{B}_{x,\ell} \cdot \mathbf{U}_{1,\ell} + (H(x) + \overline{\mathbf{e}}_x),$$

where $\overline{\mathbf{e}}_x = (\overline{e}_{x,\ell})_\ell$ is $\lambda^{\mathrm{poly}(\lambda)}$-bounded, as claimed. □

The following is an immediate consequence of Lemma 3.8.

Corollary 3.9. *Fix any function $H\colon \{0,1\}^\iota \to \mathbb{Z}_q^m$ of size at most σ, $pp \leftarrow \mathsf{Setup}(1^\lambda, 1^\rho)$, $msk \leftarrow \mathsf{KeyGen}(pp)$, and $sk_H \leftarrow \mathsf{Shift}(pp, msk, H)$, and input $x \in \{0,1\}^\iota$. If for all $i \in [m]$ we have*

$$(\mathsf{SEval}(pp, sk, x) - H(x))_i \notin \tfrac{q}{p}(\mathbb{Z} + \tfrac{1}{2}) + [-B, +B]$$

for the $B = \lambda^{\mathrm{poly}(\lambda)}$ from Lemma 3.8, then

$$\lfloor \mathsf{SEval}(pp, sk, x) - H(x) \rceil_p = \lfloor \mathsf{Eval}(pp, msk, x) \rceil_p.$$

3.4 Parameter Instantiation

We now instantiate the LWE parameters n, q and the 1D-R-SIS parameter k to correspond with subexponential $\exp(n^\varepsilon)$ and $\exp(k^\varepsilon)$ approximation factors for the underlying worst-case lattice problems, for an arbitrary desired constant $\varepsilon > 0$. Let $B = \lambda^{\mathrm{poly}(\lambda)}$ be the bound from Corollary 3.9. For 1D-R-SIS we need to choose k sufficiently large primes $p_i = B \cdot \mathrm{poly}(\lambda) = \lambda^{\mathrm{poly}(\lambda)}$ to get an approximation factor of

$$B \cdot \mathrm{poly}(\lambda) = \lambda^{\mathrm{poly}(\lambda)}$$

for k-dimensional lattices. Therefore, we can choose a sufficiently large $k = \mathrm{poly}(\lambda)$ to make this factor $\exp(k^\varepsilon)$. We then set

$$q = p \prod_{i=1}^{k} p_i = p \cdot \lambda^{k \cdot \mathrm{poly}(\lambda)} = \lambda^{\mathrm{poly}(\lambda)},$$

which corresponds to some $\lambda^{\mathrm{poly}(\lambda)}$ approximation factor for n-dimensional lattices. Again, we can choose a sufficiently large $n = \mathrm{poly}(\lambda)$ to make this factor $\exp(n^\varepsilon)$.

4 Constraint-Hiding Constrained PRF

In this section we formally define constraint-hiding constrained PRFs (CHC-PRFs) and give a construction based on our shiftable PRF from Section 3.

4.1 Definition

Here we give the definition of CHC-PRFs, specializing the simulation-based definition of [CC17] to the case of a single constrained-key query.

Definition 4.1. A *constrained function* is a tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ having the following interfaces (where the domain $\mathcal{X}$ and range $\mathcal{Y}$ may depend on the security parameter):

- $\mathsf{Setup}(1^\lambda, 1^\sigma)$, given the security parameter λ and an upper bound σ on the size of the constraining circuit, outputs public parameters pp.
- $\mathsf{KeyGen}(pp)$, given the public parameters pp, outputs a master secret key msk.
- $\mathsf{Eval}(pp, msk, x)$, given the master secret key and an input $x \in \mathcal{X}$, outputs some $y \in \mathcal{Y}$.
- $\mathsf{Constrain}(pp, msk, C)$, given the master secret key and a circuit C of size at most σ, outputs a constrained key sk_C.
- $\mathsf{CEval}(pp, sk_C, x)$, given a constrained key sk_C and an input $x \in \mathcal{X}$, outputs some $y \in \mathcal{Y}$.

Definition 4.2. A constrained function is a *constraint-hiding constrained PRF* (CHC-PRF) if there is a PPT simulator $\mathcal{S}$ such that, for any PPT adversary $\mathcal{A}$ (that without loss of generality never repeats a query) and any $\sigma = \sigma(\lambda) = \mathrm{poly}(\lambda)$,

$$\{\mathsf{Real}_{\mathcal{A}}(1^\lambda, 1^\sigma)\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda, 1^\sigma)\}_{\lambda\in\mathbb{N}},$$

where Real and Ideal are the respective views of $\mathcal{A}$ in the experiments defined in Figure 2.

The above simulation-based definition simultaneously captures privacy of the constraining function, pseudorandomness on unauthorized inputs, and correctness of constrained evaluation on authorized inputs. The first two properties (privacy and pseudorandomness) follow because in the ideal experiment, the simulator must generate a constrained key without knowing the constraining function, and the adversary gets oracle access to a function that is uniformly random on unauthorized inputs.

For correctness, we claim that the real experiment is computationally indistinguishable from a modified one where each query x is answered as $\mathsf{CEval}(pp, sk_C, x)$ if x is authorized (i.e., $C(x) = 0$), and as $\mathsf{Eval}(pp, msk, x)$ otherwise. In particular, this implies that $\mathsf{Eval}(pp, msk, x) = \mathsf{CEval}(pp, sk_C, x)$ with all but negligible probability for all the adversary's authorized queries x. Indistinguishability of the real and modified experiments follows by a routine hybrid argument, with the ideal experiment as the intermediate one. In particular, the reduction that links the ideal and modified real experiments itself answers authorized queries x using CEval, and handles unauthorized queries by passing them to its oracle.

4.2 Construction

We now describe our construction of a CHC-PRF for domain $\mathcal{X} = \{0,1\}^\iota$ and range $\mathcal{Y} = \mathbb{Z}_p^m$, which handles constraining circuits of size σ. It uses the following components:

- A pseudorandom function $\mathsf{PRF} = (\mathsf{PRF.KG}, \mathsf{PRF.Eval})$ having domain $\{0,1\}^\iota$ and range $\mathbb{Z}_q^m$, with key space $\{0,1\}^\kappa$.
- The shift hiding shiftable function $\mathsf{SHSF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Shift}, \mathsf{SEval}, \mathsf{Sim})$ from Section 3, which has parameters q, B that appear in the analysis below.

procedure $\mathsf{Real}_{\mathcal{A}}(1^\lambda, 1^\sigma)$
$C \leftarrow \mathcal{A}(1^\lambda, 1^\sigma)$
$pp \leftarrow \mathsf{Setup}(1^\lambda)$
$msk \leftarrow \mathsf{KeyGen}(pp)$
$sk_C \leftarrow \mathsf{Constrain}(pp, msk, C)$
$(pp, sk_C) \to \mathcal{A}$
repeat
$x \leftarrow \mathcal{A}$
$\mathsf{Eval}(pp, msk, x) \to \mathcal{A}$
until $\mathcal{A}$ halts

(a) The real experiment

procedure $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda, 1^\sigma)$
$C \leftarrow \mathcal{A}(1^\lambda, 1^\sigma)$
$(pp, sk) \leftarrow \mathcal{S}(1^\lambda, 1^\sigma)$
$(pp, sk) \to \mathcal{A}$
repeat
$x \leftarrow \mathcal{A}$
if $C(x) = 0$ **then**
$\mathsf{CEval}(pp, sk, x) \to \mathcal{A}$
else
$y \leftarrow \mathcal{Y}$; $y \to \mathcal{A}$
until $\mathcal{A}$ halts

(b) The ideal experiment

Figure 2: The real and ideal experiments.

For a circuit C of size at most σ and some $k \in \{0,1\}^\kappa$ define the function $H_{C,k} \colon \{0,1\}^\iota \to \mathbb{Z}_q^m$ as

$$H_{C,k}(x) = C(x) \cdot \mathsf{PRF.Eval}(k, x) = \begin{cases} \mathsf{PRF.Eval}(k, x) & \text{if } U(C, x) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Notice that the size of $H_{C,k}$ is upper bounded by

$$\sigma' = \sigma + s + \mathrm{poly}(n, \log q), \tag{4.1}$$

where s denotes the circuit size of $\mathsf{PRF.Eval}(k, \cdot)$.

Construction 4.3. Our CHC-PRF with domain $\mathcal{X} = \{0,1\}^\iota$ and range $\mathcal{Y} = \mathbb{Z}_p^m$ is defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^\sigma)$: output $pp \leftarrow \mathsf{SHSF.Setup}(1^\lambda, 1^{\sigma'})$ where σ' is defined as in Equation (4.1).
- $\mathsf{KeyGen}(pp)$: output $msk \leftarrow \mathsf{SHSF.KeyGen}(pp)$.
- $\mathsf{Eval}(pp, msk, x \in \{0,1\}^\iota)$: compute $\mathbf{y}_x = \mathsf{SHSF.Eval}(pp, msk, x)$ and output $\lfloor \mathbf{y}_x \rceil_p$.
- $\mathsf{Constrain}(pp, msk, C)$: on input a circuit C of size at most σ, sample a PRF key $k \leftarrow \mathsf{PRF.KG}(1^\lambda)$ and output $sk_C \leftarrow \mathsf{SHSF.Shift}(pp, msk, H_{C,k})$.
- $\mathsf{CEval}(pp, sk_C, x)$: on input a constrained key sk_C and $x \in \{0,1\}^\iota$, output $\lfloor \mathsf{SHSF.SEval}(pp, sk_C, x) \rceil_p$.

4.3 Security Proof

Theorem 4.4. *Construction 4.3 is a constraint-hiding constrained PRF assuming the hardness of* $\mathsf{LWE}_{n-1,q,\chi}$ *and 1D-R-SIS*$_{(z\sigma'+\tau+1)m,p,q,B}$ *(where z, τ are respectively the lengths of fresh GSW ciphertexts and secret keys as used in* SHSF*), the CPA security of the GSW encryption scheme, and that* PRF *is a pseudorandom function.*

Proof. Our simulator $\mathcal{S}(1^\lambda, 1^\sigma)$ for Construction 4.3 simply outputs $\mathsf{SHSF}.\mathcal{S}(1^\lambda, 1^{\sigma'})$. Now let $\mathcal{A}$ be any polynomial-time adversary. To show that $\mathcal{S}$ satisfies Definition 4.2 we define a sequence of hybrid experiments and show that they are indistinguishable. Before defining the experiments in detail, we first define a particular "bad" event in all but one of them.

Definition 4.5. In each of the following hybrid experiments except H_0, each query x is answered as $\lfloor \mathbf{y}_x \rceil_p$ for some $\mathbf{y}_x$ that is computed in a certain way. Define Borderline to be the event that at least one such $\mathbf{y}_x$ has some coordinate in $\frac{q}{p}(\mathbb{Z} + \frac{1}{2}) + [-B, B]$.

Hybrid H_0: This is the ideal experiment $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}$.

Hybrid H_1: This is the same as H_0, except that on every unauthorized query x (i.e., where $C(x) = 1$), instead of returning a uniformly random value from $\mathbb{Z}_p^m$, we choose $\mathbf{y}_x \leftarrow \mathbb{Z}_q^m$ and output $\lfloor \mathbf{y}_x \rceil_p$.

Hybrid H_2: This is the same as H_1, except that we abort the experiment if Borderline happens.

Hybrid H_3: This is the same as H_2, except that we initially choose a PRF key $k \leftarrow \mathsf{PRF.KG}(1^\lambda)$ and change how unauthorized queries x (i.e., where $C(x) = 1$) are handled, answering all queries according to a slightly modified CEval. Specifically, for any query x we answer $\lfloor \mathbf{y}_x \rceil_p$ where

$$\mathbf{y}_x = \mathsf{SHSF.SEval}(pp, sk, x) - C(x) \cdot \mathsf{PRF.Eval}(k, x).$$

Hybrid H_4: This is the same as H_3, except that (pp, sk) are generated as in the real experiment. More formally we instantiate $pp \leftarrow \mathsf{SHSF.Setup}(1^\lambda, 1^{\sigma'})$, $msk \leftarrow \mathsf{SHSF.KeyGen}(pp)$ and compute $sk \leftarrow \mathsf{SHSF.Shift}(pp, msk, H_{C,k})$.

Hybrid H_5: This is the same as H_4, except that we answer all evaluation queries as in the Eval algorithm, i.e., we output $\lfloor \mathbf{y}_x \rceil_p$ where

$$\mathbf{y}_x = \mathsf{SHSF.Eval}(pp, msk, x).$$

Hybrid H_6: This is the same as H_5, except that we no longer abort when Borderline happens. Observe that this is exactly the real experiment $\mathsf{Real}_{\mathcal{A}}$.

We now prove that adjacent pairs of hybrid experiments are indistinguishable.

Claim 4.6. *Experiments H_0 and H_1 are identical.*

Proof. This follows directly from the fact that p divides q. □

Claim 4.7. *Assuming that 1D-R-SIS$_{(z\sigma'+\tau+1)m,p,q,B}$ is hard, we have $H_1 \overset{c}{\approx} H_2$. In particular, in H_1 the event* Borderline *happens with negligible probability.*

Proof. Let $\mathcal{A}$ be an adversary attempting to distinguish H_1 and H_2. We want to show that in H_1 event Borderline happens with negligible probability. Let x be a query made by $\mathcal{A}$. If $C(x) = 1$ then $\mathbf{y}_x$ is uniformly random in $\mathbb{Z}_q^m$, so for any $i \in [m]$ we have

$$\Pr[(\mathbf{y}_x)_i \in \tfrac{q}{p}(\mathbb{Z} + \tfrac{1}{2}) + [-B, B]] \leq 2 \cdot B \cdot p/q = \mathrm{negl}(\lambda).$$

If $C(x) = 0$, the claim follows immediately by the border-avoiding property of SHSF (Lemma 3.7). □

Claim 4.8. *If* PRF *is a pseudorandom function then $H_2 \overset{c}{\approx} H_3$.*

Proof. We use any adversary $\mathcal{A}$ that attempts to distinguish H_2 from H_3 to build an adversary $\mathcal{A}'$ having the same advantage against the pseudorandomness of PRF. Here $\mathcal{A}'$ is given access to an oracle $\mathcal{O}$ which is

either $\mathsf{PRF.Eval}(k, \cdot)$ for $k \leftarrow \mathsf{PRF.KG}(1^\lambda)$, or a uniformly random function $f\colon \{0,1\}^\iota \to \mathbb{Z}_q^m$. We define $\mathcal{A}'$ to proceed as in H_2 to simulate the view of $\mathcal{A}$, except that on each query x it sets

$$\mathbf{y}_x = \mathsf{SHSF.SEval}(pp, sk, x) - C(x) \cdot \mathcal{O}(x)$$

and answers $\lfloor \mathbf{y}_x \rceil_p$. Finally, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs. Clearly, if $\mathcal{O}$ is $\mathsf{PRF.Eval}(k, \cdot)$ then the view of $\mathcal{A}$ is identical to H_3, whereas if the oracle is $f(\cdot)$ then the view of $\mathcal{A}$ is identical to its view in H_2. This proves the claim. □

Claim 4.9. *Assuming the hardness of* $\mathsf{LWE}_{n-1,q,\chi}$ *and CPA-security of* GSW, $H_3 \overset{c}{\approx} H_4$.

Proof. This follows immediately from the shift hiding property of SHSF, i.e., Lemma 3.2. □

Claim 4.10. H_4 *and* H_5 *are identical.*

Proof. This follows by Corollary 3.9 and noticing that both experiments abort if Borderline happens. □

Claim 4.11. *Under the hypotheses of Theorem 4.4, we have* $H_5 \overset{c}{\approx} H_6$.

Proof. This follows by combining all the previous claims and recalling that we have proved that Borderline happens with negligible probability in H_1. □

This completes the proof of Theorem 4.4. □

5 Privately Programmable PRF

In this section we formally define privately programmable PRFs (PP-PRFs) and give a construction based on our shiftable PRF from Section 3.

5.1 Definitions

We start by giving a variety of definitions related to "programmable functions" and privately programmable PRFs. In particular, we give a simulation-based definition that is adapted from [BLW17].

Definition 5.1. A *programmable function* is a tuple $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Program}, \mathsf{PEval})$ of efficient algorithms having the following interfaces (where the domain $\mathcal{X}$ and range $\mathcal{Y}$ may depend on the security parameter):

- $\mathsf{Setup}(1^\lambda, 1^k)$, given the security parameter λ and a number k of programmable inputs, outputs public parameters pp.
- $\mathsf{KeyGen}(pp)$, given the public parameters pp, outputs a master secret key msk.
- $\mathsf{Eval}(pp, msk, x)$, given the master secret key and an input $x \in \mathcal{X}$, outputs some $y \in \mathcal{Y}$.
- $\mathsf{Program}(pp, msk, \mathcal{P} = \{(x_i, y_i)\})$, given the master secret key msk and k pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ for distinct x_i, outputs a programmed key $sk_\mathcal{P}$.
- $\mathsf{PEval}(pp, sk_\mathcal{P}, x)$, given a programmed key $sk_\mathcal{P}$ and an input $x \in \mathcal{X}$, outputs some $y \in \mathcal{Y}$.

We now give several definitions that capture various functionality and security properties for programmable functions. We start with the following correctness property for *programmed* inputs.

Definition 5.2. A programmable function is *statistically programmable* if for all $\lambda, k = \mathrm{poly}(\lambda) \in \mathbb{N}$, all sets of k pairs $\mathcal{P} = \{(x_i, y_i)\} \subseteq \mathcal{X} \times \mathcal{Y}$ (with distinct x_i), and all $i \in [k]$ we have

$$\Pr_{\substack{pp \leftarrow \mathsf{Setup}(1^\lambda, 1^k) \\ msk \leftarrow \mathsf{KeyGen}(pp) \\ sk_{\mathcal{P}} \leftarrow \mathsf{Program}(pp, msk, \mathcal{P})}} [\mathsf{PEval}(pp, sk_{\mathcal{P}}, x_i) \neq y_i] = \mathrm{negl}(\lambda).$$

We now define a notion of *weak* simulation security, in which the adversary names the inputs at which the function is programmed, but the outputs are chosen at random (and not revealed to the adversary). As before, we always assume without loss of generality that the adversary never queries the same input x more than once in the various experiments we define.

procedure $\mathsf{RealWeakPPRF}_{\mathcal{A}}(1^\lambda, 1^k)$
- $\{x_i\}_{i \in [k]} \leftarrow \mathcal{A}(1^\lambda, 1^k)$
- $\{y_i\}_{i \in [k]} \leftarrow \mathcal{Y}$
- $pp \leftarrow \mathsf{Setup}(1^\lambda, 1^k)$
- $msk \leftarrow \mathsf{KeyGen}(pp)$
- $sk \leftarrow \mathsf{Program}(pp, msk, \{(x_i, y_i)\})$
- $(pp, sk) \rightarrow \mathcal{A}$
- **repeat**
  - $x \leftarrow \mathcal{A}$
  - $\mathsf{Eval}(pp, msk, x) \rightarrow \mathcal{A}$
- **until** $\mathcal{A}$ halts

(a) The real experiment

procedure $\mathsf{IdealWeakPPRF}_{\mathcal{A},\mathcal{S}}(1^\lambda, 1^k)$
- $\{x_i\}_{i \in [k]} \leftarrow \mathcal{A}(1^\lambda, 1^k)$
- $(pp, sk) \leftarrow \mathcal{S}(1^\lambda, 1^k)$
- $(pp, sk) \rightarrow \mathcal{A}$
- **repeat**
  - $x \leftarrow \mathcal{A}$
  - **if** $x \notin \{x_i\}$ **then**
    - $\mathsf{PEval}(pp, sk, x) \rightarrow \mathcal{A}$
  - **else**
    - $y \leftarrow \mathcal{Y}$; $y \rightarrow \mathcal{A}$
- **until** $\mathcal{A}$ halts

(b) The ideal experiment

Figure 3: The (weak) real and ideal experiments.

Definition 5.3. A programmable function is *weakly simulation secure* if there is a PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$ and any polynomial $k = k(\lambda)$,

$$\{\mathsf{RealWeakPPRF}_{\mathcal{A}}(1^\lambda, 1^k)\}_{\lambda \in \mathbb{N}} \overset{c}{\approx} \{\mathsf{IdealWeakPPRF}_{\mathcal{A},\mathcal{S}}(1^\lambda, 1^k)\}_{\lambda \in \mathbb{N}},$$

where RealWeakPPRF and IdealWeakPPRF are the respective views of $\mathcal{A}$ in the procedures defined in Figure 3.

Similarly to Definition 4.2, the above definition simultaneously captures privacy of the programmed inputs given the programmed key, pseudorandomness on those inputs, and correctness of PEval on *non-programmed* inputs.

Definition 5.4. A programmable function is a *weak privately programmable PRF* if it is statistically programmable (Definition 5.2) and weakly simulation secure (Definition 5.3).

We now define a notion of (non-weak) simulation security for programmable functions. This differs from the weak notion in that the adversary specifies the programmed inputs *and* corresponding outputs, and

the simulator in the ideal game is also given these input-output pairs. The simulator needs this information because otherwise the adversary could trivially distinguish the real and ideal experiments by checking whether $\mathsf{PEval}(pp, sk_{\mathcal{P}}, x_i) = y_i$ for one of the programmed input-output pairs (x_i, y_i). Simulation security itself therefore does not guarantee any privacy of the programmed inputs; below we give a separate simulation-based definition which does.

procedure $\mathsf{RealPPRF}_{\mathcal{A}}(1^\lambda, 1^k)$
  $\mathcal{P} = \{(x_i, y_i)\} \leftarrow \mathcal{A}(1^\lambda, 1^k)$
  $pp \leftarrow \mathsf{Setup}(1^\lambda, 1^k)$
  $msk \leftarrow \mathsf{KeyGen}(pp)$
  $sk_{\mathcal{P}} \leftarrow \mathsf{Program}(pp, msk, \mathcal{P})$
  $(pp, sk_{\mathcal{P}}) \rightarrow \mathcal{A}$
  repeat
    $x \leftarrow \mathcal{A}$
    $\mathsf{Eval}(pp, msk, x) \rightarrow \mathcal{A}$
  until $\mathcal{A}$ halts

(a) The real experiment

procedure $\mathsf{IdealPPRF}_{\mathcal{A},\mathcal{S}}(1^\lambda, 1^k)$
  $\mathcal{P} = \{(x_i, y_i)\} \leftarrow \mathcal{A}(1^\lambda, 1^k)$
  $(pp, sk_{\mathcal{P}}) \leftarrow \mathcal{S}(1^\lambda, \mathcal{P})$
  $(pp, sk_{\mathcal{P}}) \rightarrow \mathcal{A}$
  repeat
    $x \leftarrow \mathcal{A}$
    if $x \notin \{x_i\}$ **then**
      $\mathsf{PEval}(pp, sk_{\mathcal{P}}, x) \rightarrow \mathcal{A}$
    else
      $y \leftarrow \mathcal{Y}$; $y \rightarrow \mathcal{A}$
  until $\mathcal{A}$ halt

(b) The ideal experiment

Figure 4: The real and ideal experiments

Definition 5.5. A programmable function is *simulation secure* if there is a PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$ and any polynomial $k = k(\lambda)$,

$$\{\mathsf{RealPPRF}_{\mathcal{A}}(1^\lambda, 1^k)\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{IdealPPRF}_{\mathcal{A},\mathcal{S}}(1^\lambda, 1^k)\}_{\lambda\in\mathbb{N}},$$

where Real and Ideal are the respective views of $\mathcal{A}$ in the procedures defined in Figure 4.

We mention that a straightforward hybrid argument similar to one from [BKM17] shows that simulation security implies that $(\mathsf{KeyGen}, \mathsf{Eval})$ is a pseudorandom function.

Finally, we define a notion of privacy for the programmed inputs. This says that a key programmed on adversarially chosen inputs and *random* corresponding outputs (that are not revealed to the adversary) does not reveal anything about the programmed inputs.

procedure $\mathsf{RealPPRFPrivacy}_{\mathcal{A}}(1^\lambda, 1^k)$
  $\{x_i\}_{i\in[k]} \leftarrow \mathcal{A}(1^\lambda, 1^k)$
  $\{y_i\}_{i\in[k]} \leftarrow \mathcal{Y}$
  $pp \leftarrow \mathsf{Setup}(1^\lambda, 1^k)$
  $msk \leftarrow \mathsf{KeyGen}(pp)$
  $sk \leftarrow \mathsf{Program}(pp, msk, \{(x_i, y_i)\})$
  $(pp, sk) \rightarrow \mathcal{A}$

(a) The real experiment

procedure $\mathsf{IdealPPRFPrivacy}_{\mathcal{A},\mathcal{S}}(1^\lambda, 1^k)$
  $\{x_i\}_{i\in[k]} \leftarrow \mathcal{A}(1^\lambda, 1^k)$
  $(pp, sk) \leftarrow \mathcal{S}(1^\lambda, 1^k)$
  $(pp, sk) \rightarrow \mathcal{A}$

(b) The ideal experiment

Figure 5: The real and ideal privacy experiments

Definition 5.6. A programmable function is *privately programmable* if there is a PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$ and any polynomial $k = k(\lambda)$,

$$\{\mathsf{RealPPRFPrivacy}_{\mathcal{A}}(1^\lambda, 1^k)\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{IdealPPRFPrivacy}_{\mathcal{A}}(1^\lambda, 1^k)\}_{\lambda\in\mathbb{N}},$$

where RealPPRFPrivacy and IdealPPRFPrivacy are the respective views of $\mathcal{A}$ in the procedures defined in Figure 5.

We now give our main security definition for PP-PRFs.

Definition 5.7. A programmable function is a *privately programmable PRF* if it is statistically programmable, simulation secure, and privately programmable.

5.2 From Weak PP-PRFs to PP-PRFs

In this section we describe a general construction of a privately programmable PRF from any weak privately programmable PRF. Let $\Pi' = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Program}, \mathsf{PEval})$ be a programmable function with domain $\mathcal{X}$ and range $\mathcal{Y}$, where we assume that $\mathcal{Y}$ is a finite additive group. The basic idea behind the construction is simple: define the function as the sum of two parallel copies of Π', and program it by programming the copies according to additive secret-sharings of the desired outputs. Each component is therefore programmed to uniformly random outputs, as required by weak simulation security.

Construction 5.8. We construct a programmable function Π as follows:

- $\Pi.\mathsf{Setup}(1^\lambda, 1^k)$: generate $pp_i \leftarrow \Pi'.\mathsf{Setup}(1^\lambda, 1^k)$ for $i = 1, 2$ and output $pp = (pp_1, pp_2)$.
- $\Pi.\mathsf{KeyGen}(pp)$: on input $pp = (pp_1, pp_2)$ generate $msk_i \leftarrow \Pi'.\mathsf{KeyGen}(pp_i)$ for $i = 1, 2$, and output $msk = (msk_1, msk_2)$.
- $\Pi.\mathsf{Eval}(pp, msk, x)$: on input $pp = (pp_1, pp_2)$, $msk = (msk_1, msk_2)$, and $x \in \mathcal{X}$ output

$$\Pi'.\mathsf{Eval}(pp_1, msk_1, x) + \Pi'.\mathsf{Eval}(pp_2, msk_2, x).$$

- $\Pi.\mathsf{Program}(pp, msk, \mathcal{P})$: on input $pp = (pp_1, pp_2)$, $msk = (msk_1, msk_2)$, k pairs $(x_i, y_i) \subset \mathcal{X} \times \mathcal{Y}$, first sample uniformly random $r_i \leftarrow \mathcal{Y}$ for $i \in [k]$, then output $sk_{\mathcal{P}} = (sk_1, sk_2)$ where

$$sk_1 \leftarrow \Pi'.\mathsf{Program}(pp_1, msk_1, \mathcal{P}_1 = \{(x_i, r_i)\})$$
$$sk_2 \leftarrow \Pi'.\mathsf{Program}(pp_2, msk_2, \mathcal{P}_2 = \{(x_i, y_i - r_i)\}).$$

- $\Pi.\mathsf{PEval}(pp, sk_{\mathcal{P}}, x)$: on input $pp = (pp_1, pp_2)$, $sk_{\mathcal{P}} = (sk_1, sk_2)$, and $x \in \mathcal{X}$ output

$$\Pi'.\mathsf{PEval}(pp_1, sk_1, x) + \Pi'.\mathsf{PEval}(pp_2, sk_2, x).$$

Theorem 5.9. *If Π' is a weak privately programmable PRF then Construction 5.8 is a privately programmable PRF.*

Proof. This follows directly from Theorem 5.10 and Theorem 5.15, which respectively prove the simulation security and private programmability of Construction 5.8, and from the statistical programmability of Π', which obviously implies the statistical programmability of Construction 5.8. □

Theorem 5.10. *If Π' is a weak privately programmable PRF then Π is simulation secure.*

Proof. Let $\mathcal{S}'$ be the simulator algorithm for the weak simulation security of Π' (Definition 5.3). Our simulator $\mathcal{S}(1^\lambda, \mathcal{P} = \{(x_i, y_i)\})$ for the simulation security of Π (Definition 5.5) works as follows:

1. Compute $(pp_2, sk_2) \leftarrow \mathcal{S}'(1^\lambda, 1^k)$ where $k = |\mathcal{P}|$.
2. Compute $pp_1 \leftarrow \Pi'.\mathsf{Setup}(1^\lambda, 1^k)$, $msk_1 \leftarrow \Pi'.\mathsf{KeyGen}(pp_1)$, and
$$sk_1 \leftarrow \Pi'.\mathsf{Program}(pp_1, msk_1, \{(x_i, y_i - \Pi'.\mathsf{PEval}(pp_2, sk_2, x_i))\}).$$
3. Output $(pp = (pp_1, pp_2), sk = (sk_1, sk_2))$.

We now define the following hybrids and show that they are indistinguishable.

Hybrid H_0: This is the real experiment $\mathsf{RealPPRF}_{\mathcal{A}}$ from Figure 4, up to negligible statistical distance. Specifically: the adversary $\mathcal{A}$ first outputs $\{(x_i, y_i)\}$. We then generate $pp_i \leftarrow \Pi'.\mathsf{Setup}(1^\lambda, 1^k)$ and $msk_i \leftarrow \Pi'.\mathsf{KeyGen}(pp_i)$ for $i = 1, 2$. We choose uniformly random $r_i \leftarrow \mathcal{Y}$ for $i \in [k]$, and let
$$sk_1 \leftarrow \Pi'.\mathsf{Program}(pp_1, msk_1, \{(x_i, r_i)\})$$
$$sk_2 \leftarrow \Pi'.\mathsf{Program}(pp_2, msk_2, \{(x_i, y_i - \Pi'.\mathsf{PEval}(pp_1, sk_1, x_i))\}).$$
(Note that in $y_i - r_i$ we have replaced r_i with a call to $\Pi'.\mathsf{PEval}$; these are equivalent up to negligible statistical distance by statistical programmability.) We give $(pp = (pp_1, pp_2), sk = (sk_1, sk_2))$ to $\mathcal{A}$. Then $\mathcal{A}$ is given query access to $\Pi'.\mathsf{Eval}(pp_1, msk_1, \cdot) + \Pi'.\mathsf{Eval}(pp_2, msk_2, \cdot)$.

Hybrid H_1: This is the same as the previous experiment, except we change how pp_1 and sk_1 are generated, and how queries are answered. We generate $(pp_1, sk_1) \leftarrow \mathcal{S}'(1^\lambda, 1^k)$, and no longer generate msk_1. To answer each query x, if $x \notin \{x_i\}$ then we output $\Pi'.\mathsf{PEval}(pp_1, sk_1, x) + \Pi'.\mathsf{Eval}(pp_2, msk_2, x)$; otherwise, we output a uniformly random value from $\mathcal{Y}$.

Hybrid H_2: This is the same as the (real) experiment H_0, except that for queries $x \notin \{x_i\}$, just as in H_1 we output $\Pi'.\mathsf{PEval}(pp_1, sk_1, x) + \Pi'.\mathsf{Eval}(pp_2, msk_2, x)$.

Hybrid H_3: This is the same as the previous experiment, except we replace $y_i - \Pi'.\mathsf{PEval}(pp_1, sk_1, x_i)$ with $y_i - r_i$, then we swap the roles of r_i and $y_i - r_i$ (using r_i when computing sk_2 and $y_i - r_i$ when computing sk_1), then replace $y_i - r_i$ with $y_i - \Pi'.\mathsf{PEval}(pp_2, sk_2, x_i)$.

Hybrid H_4: This is the same as the previous experiment, except we change how pp_2 and sk_2 are generated and how queries are answered. We generate $(pp_2, sk_2) \leftarrow \mathcal{S}'(1^\lambda, 1^k)$, and no longer generate msk_2. To answer a query x, if $x \notin \{x_i\}$ then we respond with $\Pi'.\mathsf{PEval}(pp_1, sk_1, x) + \Pi'.\mathsf{PEval}(pp_2, sk_2, x)$; otherwise, we respond with a uniformly random value from $\mathcal{Y}$.

Observe that this hybrid is identical to the ideal experiment $\mathsf{IdealPPRF}_{\mathcal{A},\mathcal{S}'}$ from Figure 4.

We now show that adjacent hybrid experiments are indistinguishable.

Claim 5.11. *We have $H_0 \overset{c}{\approx} H_1$.*

Proof. Let $\mathcal{A}$ be an adversary attempting to distinguish H_0 from H_1. We build an adversary $\mathcal{A}'$ against the weak simulation security of Π' that runs $\mathcal{A}$ internally and works as follows:

- When $\mathcal{A}$ outputs $\{(x_i, y_i)\}$, $\mathcal{A}'$ outputs $\{x_i\}$ and receives some (pp_1, sk_1).
- $\mathcal{A}'$ computes $pp_2 \leftarrow \Pi'.\mathsf{Setup}(1^\lambda, 1^k)$, $msk_2 \leftarrow \Pi'.\mathsf{KeyGen}(pp_2)$, and

$$sk_2 \leftarrow \Pi'.\mathsf{Program}(pp_2, msk_2, \{(x_i, y_i - \Pi'.\mathsf{PEval}(pp_1, sk_1, x_i))\}),$$

  and gives $(pp = (pp_1, pp_2), sk = (sk_1, sk_2))$ to $\mathcal{A}$.
- When $\mathcal{A}$ queries some x, $\mathcal{A}'$ queries x and receives a response y', and returns $y' + \Pi'.\mathsf{Eval}(pp_2, msk_2, x)$ as the answer to $\mathcal{A}$.

Because the r_i are uniformly random in H_0, it is straightforward to verify that if $\mathcal{A}'$ is in the $\mathsf{RealWeakPPRF}$ (respectively, $\mathsf{IdealWeakPRF}$) experiment, then the view of $\mathcal{A}$ is identical to its view in H_0 (resp., H_1). So by weak simulation security of Π', we have $H_0 \overset{c}{\approx} H_1$. □

Claim 5.12. *We have* $H_1 \overset{c}{\approx} H_2$.

Proof. Let $\mathcal{A}$ be an adversary attempting to distinguish H_1 from H_2. We build an adversary $\mathcal{A}'$ against the weak simulation security of Π' that works exactly like the $\mathcal{A}'$ from the proof of Claim 5.11, except for how it handles queries x: if $x \notin \{x_i\}$, then $\mathcal{A}'$ responds with $\Pi'.\mathsf{PEval}(pp_1, sk_1, x) + \Pi'.\mathsf{Eval}(pp_2, msk_2, x)$; otherwise, $\mathcal{A}'$ queries x and receives a response y', then responds with $y' + \Pi'.\mathsf{Eval}(pp_2, msk_2, x)$.

It is straightforward to verify that if $\mathcal{A}'$ is in experiment $\mathsf{IdealWeakPPRF}$ (respectively, $\mathsf{IdealRealPPRF}$), then the view of $\mathcal{A}$ is identical to its view in H_1 (resp., H_2). So by weak simulation security of Π', we have $H_0 \overset{c}{\approx} H_1$. □

Claim 5.13. *We have* $H_2 \overset{s}{\approx} H_3$.

Proof. This follows by statistical programmability, the fact that $r_i, y_i - r_i$ is a random secret sharing of y_i, so we can swap their roles, and statistical programmability again. □

Claim 5.14. *We have* $H_3 \overset{c}{\approx} H_4$.

Proof. This is entirely symmetrical to the proof of Claim 5.11, so we omit a detailed proof. □

This completes the proof of Theorem 5.10. □

Theorem 5.15. *If* Π' *is weakly simulation secure then* Π *is privately programmable.*

Proof. Let $\mathcal{S}'$ be the simulator algorithm for the weak simulation security of Π'. Our simulator $\mathcal{S}(1^\lambda, 1^k)$ for the private programmability of Π simply generates $(pp_i, sk_i) \leftarrow \mathcal{S}'(1^\lambda, 1^k)$ for $i = 1, 2$ and outputs $(pp = (pp_1, pp_2), sk = (sk_1, sk_2))$. To show that $\mathcal{S}$ satisfies Definition 5.7 we define the following hybrids and show that they are indistinguishable.

Hybrid H_0**:** This is the experiment $\mathsf{RealPPRFPrivacy}_{\mathcal{A}}$ from Figure 5.

Hybrid H_1**:** This experiment is the same as the previous one, except that we generate $(pp_1, sk_1) \leftarrow \mathcal{S}'(1^\lambda, 1^k)$.

Hybrid H_2**:** This experiment is the same as the previous one, except that we generate $(pp_2, sk_2) \leftarrow \mathcal{S}'(1^\lambda, 1^k)$. Observe that this experiment is identical to the experiment $\mathsf{IdealPPRFPrivacy}_{\mathcal{A},\mathcal{S}}$ from Figure 5.

Claim 5.16. *We have* $H_0 \overset{c}{\approx} H_1$.

Proof. Let $\mathcal{A}$ be an adversary attempting to distinguish H_0 and H_1. We build an adversary $\mathcal{A}'$ against the weak simulation security of Π', which runs $\mathcal{A}$ internally. When $\mathcal{A}$ outputs $\{x_i\}$, $\mathcal{A}'$ also outputs $\{x_i\}$, receiving (pp_1, sk_1) in response. Then $\mathcal{A}'$ generates $pp_2 \leftarrow \Pi'.\mathsf{Setup}(1^\lambda, 1^k)$ and $msk_2 \leftarrow \Pi'.\mathsf{KeyGen}(pp_2)$, and chooses uniformly random $r_i \leftarrow \mathcal{Y}$ for $i \in [k]$. It then generates $sk_2 \leftarrow \Pi'.\mathsf{Program}(pp_2, msk_2, \{(x_i, r_i)\})$. Finally it gives $(pp = (pp_1, pp_2), sk = (sk_1, sk_2))$ to $\mathcal{A}$. It is straightforward to see that if $\mathcal{A}'$ is in $\mathsf{RealWeakPPRF}$ (respectively, $\mathsf{IdealWeakPPRF}$) then the view of $\mathcal{A}$ is identical to its view H_0 (resp., H_1). So by weak simulation security of Π', we have $H_0 \overset{c}{\approx} H_1$. □

Claim 5.17. *We have* $H_1 \overset{c}{\approx} H_2$.

Proof. This is entirely symmetrical to the proof of Claim 5.16, so we omit it. □

This completes the proof of Theorem 5.15. □

5.3 Construction of Weak Privately Programmable PRFs

In this section we construct a weak privately programmable PRF from our shiftable function of Section 3. We first define the auxiliary function that the construction will use. For $\{(x_i, \mathbf{y}_i)\}_{i\in[k]} \subset \{0,1\}^\iota \times \mathbb{Z}_q^m$ where the x_i are distinct, define the function $H_{\{(x_i,\mathbf{w}_i)\}_{i\in[k]}} : \{0,1\}^\iota \to \mathbb{Z}_q^m$ as

$$H_{\{(x_i,\mathbf{w}_i)\}_{i\in[k]}}(x) \begin{cases} \mathbf{w}_i & \text{if } x = x_i \text{ for some } i, \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Notice that the circuit size of $H_{\{(x_i,\mathbf{w}_i)\}_{i\in[k]}}$ is upper bounded by some $\sigma' = \mathrm{poly}(n, k, \log q)$.

Construction 5.18. Our weak privately programmable PRF with input space $\mathcal{X} = \{0,1\}^\iota$ and output space $\mathcal{Y} = \mathbb{Z}_p^m$ uses the SHSF from Section 3 with parameters q, B chosen as in Section 3.4, and is defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^k)$: Output $pp \leftarrow \mathsf{SHSF.Setup}(1^\lambda, 1^{\sigma'})$.
- $\mathsf{KeyGen}(pp)$: Output $msk \leftarrow \mathsf{SHSF.KeyGen}(pp)$.
- $\mathsf{Eval}(pp, msk, x \in \{0,1\}^\iota)$: Compute $\mathbf{y}_x = \mathsf{SHSF.Eval}(pp, msk, x)$ and output $\lfloor \mathbf{y}_x \rceil_p$.
- $\mathsf{Program}(pp, msk, \mathcal{P})$: Given k pairs $(x_i, \mathbf{y}_i) \in \{0,1\}^\iota \times \mathbb{Z}_p^m$ where the x_i are distinct, for each $i \in [k]$ compute $\mathbf{w}_i$ as follows: choose $\mathbf{y}'_i \leftarrow \mathbb{Z}_q^m$ uniformly at random conditioned on $\lfloor \mathbf{y}'_i \rceil_p = \mathbf{y}_i$, and set
$$\mathbf{w}_i = \mathbf{y}'_i - \mathsf{SHSF.Eval}(pp, msk, x_i).$$
Output $sk_\mathcal{P} \leftarrow \mathsf{SHSF.Shift}(pp, msk, H_{\{(x_i,\mathbf{w}_i)\}})$.
- $\mathsf{PEval}(pp, sk_\mathcal{P}, x)$: output $\lfloor \mathsf{SHSF.SEval}(pp, sk_\mathcal{P}, x) \rceil_p$.

5.4 Security Proof

Theorem 5.19. *Construction 5.18 is a weak privately programmable PRF (Definition 5.4) assuming the hardness of* $\mathsf{LWE}_{n-1,q,\chi}$ *and 1D-R-SIS*$_{(z\sigma'+\tau+1)m,p,q,B}$ *(where* z, τ *are respectively the lengths of fresh GSW ciphertexts and secret keys as used in* SHSF*) and the CPA security of the GSW encryption scheme.*

Proof. The proof follows immediately by Theorem 5.20 and Theorem 5.28 below. □

Theorem 5.20. *Assuming the hardness of* $\mathsf{LWE}_{n-1,q,\chi}$ *and 1D-R-SIS*$_{(z\sigma'+\tau+1)m,p,q,B}$ *and the CPA security of the GSW encryption scheme, Construction 5.18 is weakly simulation secure.*

Proof. Our simulator $\mathcal{S}(1^\lambda, 1^k)$ for Construction 5.18 simply outputs $(pp, sk) \leftarrow \mathsf{SHSF}.\mathcal{S}(1^\lambda, 1^{\sigma'})$. Let $\mathcal{A}$ be any polynomial-time adversary. To show that $\mathcal{S}$ satisfies Definition 5.5 we define a sequence of hybrid experiments and show that they are indistinguishable.

Hybrid H_0**:** This is the simulated experiment $\mathsf{IdealWeakPPRF}_{\mathcal{A},\mathcal{S}}$ (Figure 3).

Hybrid H_1**:** This is the same as the previous experiment, except that on query $x \in \{x_i\}$, instead of returning a uniformly random value from $\mathbb{Z}_p^m$, we choose $\mathbf{y}_x \leftarrow \mathbb{Z}_q^m$ and output $\lfloor \mathbf{y}_x \rceil_p$.

Hybrid H_2**:** This is the same as the previous experiment, except that we abort if the event $\mathsf{Borderline}$ happens, where $\mathsf{Borderline}$ is as in Definition 4.5.

Hybrid H_3**:** This is the same as the previous experiment, except that we initially choose uniformly random $\mathbf{w}'_i \leftarrow \mathbb{Z}_q^m$ for $i \in [k]$ and change how queries for $x \in \{x_i\}$ are answered (the "else" clause in $\mathsf{IdealWeakPPRF}_{\mathcal{A},\mathcal{S}}$): for $x = x_j$, we answer as $\lfloor \mathbf{y}_x \rceil_p$, where

$$\mathbf{y}_x = \mathsf{SHSF.SEval}(pp, sk, x) - \mathbf{w}'_j.$$

Hybrid H_4**:** This is the same as the previous experiment, except that we generate pp and sk as follows: we generate $pp \leftarrow \mathsf{Setup}(1^\lambda, 1^k)$, $msk \leftarrow \mathsf{KeyGen}(pp)$ and $sk \leftarrow \mathsf{SHSF.Shift}(pp, msk, H_{\{(x_i, \mathbf{w}'_i)\}})$.

Hybrid H_5**:** This is the same as the previous experiment, except that we answer all queries as in the Eval algorithm, i.e., we output

$$\lfloor \mathsf{SHSF.Eval}(pp, msk, x) \rceil_p.$$

Hybrid H_6**:** This is the same as the previous experiment, except that here we generate sk as in the real game. Specifically, for each $i \in [k]$ we choose a uniformly random vector $\mathbf{y}_i \leftarrow \mathbb{Z}_p^m$ and uniformly random $\mathbf{y}'_i \leftarrow \mathbb{Z}_q^m$ conditioned on $\lfloor \mathbf{y}'_i \rceil_p = \mathbf{y}_i$, and then set

$$\mathbf{w}_i = \mathbf{y}'_i - \mathsf{SHSF.Eval}(pp, msk, x).$$

We then set $sk \leftarrow \mathsf{SHSF.Shift}(pp, msk, H_{\{(x_i, \mathbf{w}_i)\}})$.

Hybrid H_7**:** This is the same as the previous experiment, except that we no longer abort when $\mathsf{Borderline}$ happens. Observe that this is the real experiment $\mathsf{IdealRealPPRF}_{\mathcal{A}}$ (Figure 3).

Claim 5.21. *Experiments* H_0 *and* H_1 *are identical.*

Proof. This follows directly from the fact that p divides q. □

Claim 5.22. *Under 1D-R-SIS$_{(z\sigma'+\tau+1)m,p,q,B}$ assumption, $H_1 \overset{c}{\approx} H_2$. In particular, in H_1 the event* Borderline *happens with negligible probability.*

Proof. The proof is identical to the one for Claim 4.7. □

Claim 5.23. *Experiments H_2 and H_3 are identical.*

Proof. This simply follows by observing that for $x \in \{x_i\}$, $\mathbf{y}_x$ is distributed uniformly at random in both hybrids H_2 and H_3. □

Claim 5.24. *Assuming the hardness of* $\mathsf{LWE}_{n-1,q,\chi}$ *and CPA-security of* GSW*, $H_3 \overset{c}{\approx} H_4$.*

Proof. This follows immediately from the shift-hiding property of SHSF (Lemma 3.2). □

Claim 5.25. *H_4 and H_5 are identical.*

Proof. This follows by Corollary 3.9 (rounded shift correctness) and the fact that if Borderline happens we abort. □

Claim 5.26. *Experiments H_5 and H_6 are identical.*

Proof. Because p divides q, for each $i \in [k]$ each $\mathbf{y}'_i$ in H_6 is uniformly distributed, which implies that each $\mathbf{w}_i$ is also uniformly distributed, as in H_5. □

Claim 5.27. *Assuming the hypotheses of Theorem 5.20, $H_6 \overset{c}{\approx} H_7$.*

Proof. This follows by combining all the previous claims and recalling that Borderline happens with negligible probability in H_1. □

This completes the proof of Theorem 5.20. □

Theorem 5.28. *Construction 5.18 is statistically programmable.*

Proof. Fix any $\mathcal{P} = \{(x_i, \mathbf{y}_i)\}_{i\in[k]} \subset \mathcal{X} \times \mathcal{Y}$. We need to show that for any $i \in [k]$,

$$\Pr_{\substack{pp\leftarrow\mathsf{Setup}(1^\lambda,1^\mathsf{k})\\ msk\leftarrow\mathsf{KeyGen}(pp)\\ sk_\mathcal{P}\leftarrow\mathsf{Program}(pp,msk,\mathcal{P})}}\left[\left\lfloor \mathsf{SHSF.SEval}(pp, sk_\mathcal{P}, x_i)\right\rceil_p \neq \mathbf{y}_i\right] = \mathrm{negl}(\lambda). \tag{5.1}$$

By Lemma 3.8 we have

$$\begin{aligned}\mathsf{SHSF.SEval}(pp, sk_\mathcal{P}, x_i) &\approx \mathsf{SHSF.Eval}(pp, msk, x_i) + H_{\{(x_i,\mathbf{w}_i)\}}(x_i)\\ &= \mathsf{SHSF.Eval}(pp, msk, x_i) + \mathbf{w}_i\\ &= \mathbf{y}'_i,\end{aligned}$$

where the approximation hides some B-bounded error and the last equality holds because $\mathbf{w}_i = \mathbf{y}'_i - \mathsf{SHSF.Eval}(pp, msk, x_i)$. Because $\mathbf{y}'_i$ is chosen uniformly at random such that $\lfloor \mathbf{y}'_i \rceil_p = \mathbf{y}_i$, the probability that some coordinate of $\mathsf{SHSF.SEval}(pp, sk_\mathcal{P}, x_i)$ is in $\frac{q}{p}(\mathbb{Z} + \frac{1}{2}) + [-B, B]$ is at most $2mBp/q = \mathrm{negl}(\lambda)$, which establishes Equation (5.1). □

References

[Ajt96] M. Ajtai. Generating hard instances of lattice problems. *Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.

[BGG+14] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556. 2014.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012. Preliminary version in CRYPTO 2001.

[BGI14] E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519. 2014.

[BGI15] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *EUROCRYPT*, pages 337–367. 2015.

[BKM17] D. Boneh, S. Kim, and H. W. Montgomery. Private puncturable PRFs from standard lattice assumptions. In *EUROCRYPT*, pages 415–445. 2017.

[BLMR13] D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan. Key homomorphic PRFs and their applications. In *CRYPTO*, pages 410–428. 2013.

[BLP+13] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584. 2013.

[BLW17] D. Boneh, K. Lewi, and D. J. Wu. Constraining pseudorandom functions privately. In *PKC*, pages 494–524. 2017.

[BP14] A. Banerjee and C. Peikert. New and improved key-homomorphic pseudorandom functions. In *CRYPTO*, pages 353–370. 2014.

[BPR12] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, pages 719–737. 2012.

[BTVW17] Z. Brakerski, R. Tsabary, V. Vaikuntanathan, and H. Wee. Private constrained PRFs (and more) from LWE. In *TCC*, pages ??–?? 2017.

[BV15] Z. Brakerski and V. Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC*, pages 1–30. 2015.

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

[CC17] R. Canetti and Y. Chen. Constraint-hiding constrained PRFs for nc^1 from LWE. In *EUROCRYPT*, pages 446–476. 2017.

[CHN+16] A. Cohen, J. Holmgren, R. Nishimaki, V. Vaikuntanathan, and D. Wichs. Watermarking cryptographic capabilities. In *STOC*, pages 1115–1127. 2016.

[GGH+13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49. 2013.

[GGM84] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. Preliminary version in FOCS 1984.

[GSW13] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, pages 75–92. 2013.

[GVW15] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, pages 503–523. 2015.

[KPTZ13] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *CCS*, pages 669–684. 2013.

[KW17] S. Kim and D. J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In *CRYPTO*, pages 503–536. 2017.

[MP12] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718. 2012.

[MR04] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[Pei09] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342. 2009.

[PRS17] C. Peikert, O. Regev, and N. Stephens-Davidowitz. Pseudorandomness of Ring-LWE for any ring and modulus. In *STOC*, pages 461–473. 2017.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):1–40, 2009. Preliminary version in STOC 2005.