Privately Puncturing PRFs from Lattices: Adaptive Security and Collusion Resistant Pseudorandomness

Rupeng Yang

Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong NSW, Australia
rupengy@uow.edu.au

Abstract. A private puncturable pseudorandom function (PRF) enables one to create a constrained version of a PRF key, which can be used to evaluate the PRF at all but some punctured points. In addition, the constrained key reveals no information about the punctured points and the PRF values on them. Existing constructions of private puncturable PRFs are only proven to be secure against a restricted adversary that must commit to the punctured points before viewing any information. It is an open problem to achieve the more natural adaptive security, where the adversary can make all its choices on-the-fly.
In this work, we solve the problem by constructing an adaptively secure private puncturable PRF from standard lattice assumptions. To achieve this goal, we present a new primitive called explainable hash, which allows one to reprogram the hash function on a given input. The new primitive may find further applications in constructing more cryptographic schemes with adaptive security. Besides, our construction has collusion resistant pseudorandomness, which requires that even given multiple constrained keys, no one could learn the values of the PRF at the punctured points. Private puncturable PRFs with collusion resistant pseudorandomness were only known from multilinear maps or indistinguishability obfuscations in previous works, and we provide the first solution from standard lattice assumptions.

1 Introduction

A constrained pseudorandom function (PRF) [BW13,KPTZ13,BGI14] is a family of PRF [GGM84] that allows one to derive a constrained key for a predicate from a PRF key. The constrained key can be used to evaluate the PRF on inputs satisfying the predicate, but it reveals no information about the PRF values at other points. The latter requirement is denoted as (constrained) pseudorandomness and is the main security property of a constrained PRF. Besides, a constrained PRF is said to be private [BLW17] if the constrained keys also hide the constraint predicates.

As shown in [BW13, KPTZ13, BGI14], private constrained PRFs for the prefix-fixing constraint, where the predicate outputs 1 on inputs starting with a specified string, can be constructed from any one-way function via the GGM framework [GGM84]. From this framework, we can also construct constrained PRFs for the puncturing constraint (a.k.a. puncturable PRFs), where the predicate outputs 1 on all but some punctured points. This simple construction does not provide privacy, and the first private puncturable PRF is constructed from multilinear maps in [BLW17]. Then in [BKM17], Boneh et al. construct private puncturable PRFs from standard lattice assumptions.

Constrained PRFs for more complicated constraint predicates are also proposed in the linterature. In particular, constrained PRFs for circuits are constructed from multilinear maps and indistinguishability obfuscation in [BW13, CRV16] and [BZ14], respectively. Moreover, via using (differing-input) indistinguishability obfuscation, constrained PRFs for Turing machines are presented in [AFP16, AF16, DKW16, DDM17]. Besides, private constrained PRFs for circuits are constructed from indistinguishability obfuscation in [BLW17].

Subsequent works focus on constructing constrained PRFs for general constraints without using heavy tools such as multilinear maps or obfuscations. In [BV15], Brakerski and Vaikuntanathan construct the first constrained PRF for circuits from standard lattice assumptions. Then in [CC17, BTVW17, PS18, CVW18, PS20], lattice-based private constrained PRFs for circuits are provided. Besides, in [Bit17, GHKW17, AMN$^+$18], (private) constrained PRFs are also constructed from Diffie-Hellman type assumptions in traditional groups.

Adaptively Secure (Private) Constrained PRFs. When defining security properties of a (private) constrained PRF, we usually consider an adversary that is able to query some oracles, and the scheme has *adaptive security* if the adversary can query these oracles in an arbitrary order. Most previous (private) constrained PRFs are only proved to have a weaker *selective security*, where the adversary has to query the oracles in some predefined order. To achieve adaptive security generically, one can use complexity leveraging, but this would introduce an exponentially large reduction loss. In addition, the GGM framework based constrained PRFs are proved to have adaptive pseudorandomness in [FKPR14, JKK$^+$17], but the reduction loss is still super-polynomial. Besides, (private) constrained PRFs with adaptive security for various constraints are also proposed in the random oracle model in [BW13, HKKW19, AMN$^+$18].

The first adaptively secure constrained PRF in the standard model with a polynomial reduction loss is given in [HKW15], for the puncturing constraint. In the same setting, adaptively secure constrained PRFs for NC^1 circuits and any polynomial-size circuits are presented in [AMN$^+$19] and [DKN$^+$20], respectively. However, all three constructions need an indistinguishability obfuscation and are not private. Recently, (private) constrained PRFs with adaptive pseudorandomness are also constructed from simple assumptions such as one-way function and standard lattice assumptions in [DKN$^+$20], but the constructions only support constraints that can be implemented by an inner-product predicate and do not have adaptive privacy.

Collusion Resistant (Private) Constrained PRFs. A (private) constrained PRF is *collusion resistant* if its security properties hold against an adversary that sees multiple constrained keys, and in contrast, it is *single-key* secure if it is only secure against an adversary that sees one constrained key. Collusion resistance is generally satisfied by constructions from multilinear maps or indistinguishability obfuscation (e.g., [BW13, BZ14, BLW17]). However, it is quite difficult to achieve it without using these strong primitives. Especially, as shown in [CC17], a private constrained PRF with collusion resistant privacy (for certain constraints) implies indistinguishability obfuscation. Besides, previous constructions of collusion resistant constrained PRFs from standard assumptions [BW13, KPTZ13, BGI14, BFP+15, DKN+20] only support constraints in subclasses of the inner-product predicate, including the prefix-fixing constraint, the left/right predicate, and the $O(1)$-CNF predicate. We refer the readers to [DKN+20] for definitions of these constraints and their relations with the inner-product predicate.

This Work. In this work, we consider private constrained PRFs with adaptive security and collusion resistant pseudorandomness. Both security requirements are necessary for many applications illustrated in [BW13, BZ14, BLW17] and would also be useful in future applications. In addition, to prevent potential security risk (e.g., quantum attacks), we focus on constructions in the standard model from standard lattice assumptions, with a polynomial reduction loss. Existing private constrained PRFs constructed in this "*standard setting*" with either adaptive security or collusion resistant pseudorandomness [BW13, KPTZ13, BGI14, DKN+20] only support constraints that can be implemented by the inner-product predicate. This raises the following natural question:

Can we construct private constrained PRFs with the desired security requirements in the standard setting for beyond inner-product predicates?

To answer the question, we focus on private puncturable PRFs. Note that as demonstrated in [PTW20], in some special cases, constrained PRFs for the inner-product predicate exist, but it is impossible to construct a secure puncturable PRF. Thus, the puncturing constraint *cannot* be implemented by the inner-product predicate. Besides, private puncturable PRFs are useful in constructing many advanced cryptographic primitives, including symmetric deniable encryption [CDNO97], cryptographic watermarking [CHN+16], restricted searchable symmetric encryption [SWP00, BLW17], and distributed point function [GI14, BGI15]. Some of the applications (e.g., collusion resistant watermarking) need a collusion resistant (private) puncturable PRF, and some applications will achieve new desirable features immediately if the employed private puncturable PRF has adaptive security[1]. Moreover, the new security properties might

[1] For example, if we use an adaptively secure private puncturable PRF in the construction of restricted searchable encryption given in [BLW17], the scheme will additionally achieve adaptive security, which allows the database owner to issue restricted search keys on restrictions determined after the system has been put in use.

	Pseudorandomness		Privacy		Constraint
[BW13, KPTZ13, BGI14]	selective	*poly*	selective	*poly*	Prefix
[BFP+15]	selective	*poly*	✗	✗	Prefix
[BV15]	selective	1	✗	✗	P/Poly
[BKM17]	selective	1	selective	1	Puncturing
[CC17, CVW18]	selective	1	selective	1	NC^1
[BTVW17, PS18, PS20]	selective	1	selective	1	P/Poly
[DKN+20]	adaptive	$O(1)$	weak	1	$O(1)$-CNF
	adaptive	1	weak	1	IP
This Work	adaptive	*poly*	adaptive	1	Puncturing

Table 1: Properties achieved by constrained PRFs that can be instantiated from standard lattice assumptions (including one-way function) in the standard model. For either pseudorandomness or privacy, we use "adaptive" to denote adaptive security and use "selective" to denote selective security. Both the adaptive security and the selective security consider adversaries that can make queries to an evaluation oracle (see Sec. 4.1 for more details), and we use "weak" to denote that the scheme has privacy against a weaker adversary that is not allowed to query the evaluation oracle. Besides, we use the terms 1, $O(1)$, and *poly* to denote that the adversary can obtain 1, constant, and polynomial constrained key(s) when attacking the security properties. For the constraints, "Prefix" denotes the prefix-fixing constraint and "Puncturing" denotes the puncturing constraint. We use "NC^1" and "P/Poly" to denote NC^1 circuits and any polynomial-size circuits. Also, we use "IP" to denote the inner-product predicate and use "$O(1)$-CNF" to denote the $O(1)$-CNF predicate. Note that the predicates Prefix $\subseteq$ $O(1)$-CNF $\subseteq$ IP.

inspire more potential applications. Therefore, it is of both theoretical and practical interest to study private puncturable PRFs with adaptive security and collusion resistance.

1.1 Our Results

In this work, we construct a private puncturable PRF from standard lattice assumptions in the standard model, where the reduction loss is polynomial in the security parameter. The scheme has collusion resistant pseudorandomness against an adaptive adversary. In addition, it has adaptive (single-key) privacy. The latter property (i.e., adaptive privacy) is not achieved in previous construction of private constrained PRFs for any constraint from any assumption in the standard model without using complexity leveraging. We summarize features of our construction and compare it with previous constructions of constrained PRFs in the standard setting in Table 1.

To accomplish our goal, we provide new techniques for constructing adaptively secure and collusion resistant private constrained PRFs. Especially, we present a new primitive called explainable hash and construct it from lattices. The new primitive enables us to upgrade a selectively secure private puncturable PRF to have adaptive security, and it could be applied to construct other adaptively secure cryptographic schemes. We also introduce a new approach to achieve collusion resistance from standard assumptions. The idea is very different from previous methods and would inspire new constructions of collusion resistant constrained PRFs for a wider class of constraints.

1.2 Technical Overview

In this section, we provide an overview of our main techniques for constructing private puncturable PRFs with collusion resistant pseudorandomness and adaptive security. We first describe our main ideas for achieving either adaptive security or collusion resistance. Then we demonstrate how to combine the ideas to construct a private puncturable PRF with both desirable security properties.

On Achieving Adaptive Security. First, we explain how to achieve adaptive security. The adaptive security requires that the adversary cannot break security of the scheme even if it can make queries to a constrain oracle and an evaluation oracle in an arbitrary order, where the constrain oracle returns a constrained key punctured on the submitted set, and the evaluation oracle evaluates the PRF on the submitted input. Here, we consider private 1-*puncturable*[2] PRF with *single-key* security and present a general construction that upgrades a selectively secure scheme to have adaptive security in this setting.[3]

The Difficulty. First, note that it is easy to answer evaluation oracle queries after the constrain oracle query, since the evaluation results can be computed by the constrained key returned to the adversary and will not leak additional information. However, for the evaluation oracle queries before the constrain oracle query, it seems that they must be answered by the original PRF key since the puncture point is still unknown now. Thus, the evaluation results may leak information about the PRF key, which may help the adversary to break security of the scheme. This is the main difficulty for achieving adaptive security.

Our solution. To overcome the difficulty, we introduce a new primitive called explainable hash function. At a high level, an explainable hash H is an injective keyed function that can reprogram the output on a given input to a predefined value. More precisely, in its security definition, the adversary can first make queries to an evaluation oracle $\mathsf{H}(hk, \cdot)$ before viewing the hash key hk, and then it receives hk after submitting a challenge input x^* that is not queried before. Its security requires that the adversary's view in above experiment can be simulated by a simulator, and it is guaranteed that the returned hash key hk

[2] A 1-puncturable PRF punctures each PRF key on only one input.

[3] The general construction also works for larger puncture sets if we use a stronger building block in the construction. Looking ahead, this needs an explainable hash that can reprogram the outputs on multiple inputs simultaneously, which is much more difficult to construct (compared to the standard explainable hash constructed in this work).

satisfies $\mathsf{H}(hk, x^*) = u^*$, where u^* is a uniform output sampled in the beginning of the security experiment.[4]

Next, let PRF_0 be a private puncturable PRF with selective security, i.e., it is secure against an adversary that can make queries to the evaluation oracle after querying the constrain oracle. Then we show how to construct adaptively secure private puncturable PRF from PRF_0 and an explainable hash H. In our new construction, the PRF key is a PRF key k of PRF_0 and a hash key hk of H. Then, given an input x, the PRF outputs $\mathsf{PRF}_0(k, \mathsf{H}(hk, x))$. Besides, on input a punctured point x^*, the constraining algorithm punctures k on $\mathsf{H}(hk, x^*)$ and outputs the constrained version of k and the hash key hk. Since H is injective, $\mathsf{H}(hk, x) \neq \mathsf{H}(hk, x^*)$ if $x \neq x^*$. Therefore, the constrained key allows one to evaluate the PRF at all points not equal to x^*.

Now, to prove adaptive security (either pseudorandomness or privacy) of the above construction, we can puncture the secret key k on a random string u^* in the beginning and then use this constrained key (denoted as k_{u^*}) and the simulator of H to answer the evaluation oracle queries from the adversary. Next, after receiving the puncture point x^*, we can use the simulator of H to generate a hash key hk s.t. $\mathsf{H}(hk, x^*) = u^*$ and return (k_{u^*}, hk) to the adversary. Adaptive security then comes from security of H and selective security properties of PRF_0.

Constructing Explainable Hash with 1-bit output. It remains to show how to construct an explainable hash function. We first present a basic construction of (non-injective) explainable hash with 1-bit output. In a nutshell, the construction embeds an admissible hash function [BB04] into a lattice-based PRF using the matrix embedding mechanism given in [BGG+14].

An admissible hash allows one to partition an input space such that for any polynomial-size set $\mathcal{Q}$ of inputs and any input $x^* \notin \mathcal{Q}$, we have

$$\forall x \in \mathcal{Q}, \mathtt{P}(K, x) = 0 \quad \wedge \quad \mathtt{P}(K, x^*) = 1$$

with a non-negligible probability, where $\mathtt{P}$ is the partitioning function and K is a random partitioning key. Again, we omit the non-negligible failing probability here and only consider the case that the partitioning succeeds.

To embed the partitioning key $K = (K_1, \ldots, K_N)$ into a matrix $\boldsymbol{A}$, we set

$$\boldsymbol{A} = \left[\boldsymbol{B}_1 - K_1 \cdot \boldsymbol{G} \mid \ldots \mid \boldsymbol{B}_N - K_N \cdot \boldsymbol{G}\right]$$

where $\boldsymbol{B}_1, \ldots, \boldsymbol{B}_N \in \mathbb{Z}_q^{n \times m}$ are random matrices and $\boldsymbol{G}$ is the standard powers-of-two gadget matrix [MP12]. Then given the matrix $\boldsymbol{A}$ and an input x (note that the partitioning key K is *not* needed), one can get an encoding of $\mathtt{P}(K, x)$ as

$$\boldsymbol{A}_x = \left[\boldsymbol{B}_1 \mid \ldots \mid \boldsymbol{B}_N\right] \cdot \boldsymbol{T} - \mathtt{P}(K, x) \cdot \boldsymbol{G}$$

where $\boldsymbol{T}$ is a low-norm matrix.

[4] In the formal definition of explainable hash, the simulator may fail and abort with a non-negligible probability. In this overview, we assume that the simulator always succeeds for simplicity.

Now, we are ready to describe our construction of the explainable hash H_0. The hash key is a random matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times m \cdot N}$ and a random vector $\boldsymbol{s} \in \mathbb{Z}_q^n$. Given an input x, the evaluation algorithm first computes $\boldsymbol{A}_x$ from $\boldsymbol{A}$ and x. Then it outputs 0 if

$$\boldsymbol{s}^\intercal \cdot \boldsymbol{A}_x \cdot \boldsymbol{G}^{-1}(\boldsymbol{v}_1) \in [0, \frac{q}{2}]$$

and outputs 1 otherwise, where $\boldsymbol{v}_1 = (\frac{q-1}{2}, 0, \ldots, 0)^\intercal \in \mathbb{Z}_q^n$, and $\boldsymbol{G}^{-1}(\boldsymbol{v}_1)$ decomposes each element in $\boldsymbol{v}_1$ into bits and satisfies $\boldsymbol{G} \cdot \boldsymbol{G}^{-1}(\boldsymbol{v}_1) = \boldsymbol{v}_1$.

Next, we demonstrate how the simulator works. Recall that the simulator will first answer the evaluation oracle queries from an adversary, and then after the adversary submits an input x^*, the simulator needs to output a hash key, which is compatible with the evaluation oracle outputs and can map x^* to a given bit u^*.[5] Inspired by [LST18, DKN+20], we use the lossy mode of $\boldsymbol{A}$ for the simulator. More precisely, let $\bar{n} \ll n$ be an integer, the simulator embeds a random partitioning key K to the matrix $\boldsymbol{A}$ as follows:

$$\boldsymbol{A} = \left[\boldsymbol{B}_1 - K_1 \cdot \boldsymbol{G} \mid \ldots \mid \boldsymbol{B}_N - K_N \cdot \boldsymbol{G}\right]$$

where

$$\forall i \in [1, N],\ \boldsymbol{B}_i = \begin{pmatrix} \boldsymbol{r}^\intercal \cdot \bar{\boldsymbol{B}} \\ \bar{\boldsymbol{B}} \end{pmatrix} \cdot \boldsymbol{S}_i + \boldsymbol{E}_i$$

$$\boldsymbol{r} \xleftarrow{\$} \{0,1\}^{n-1}, \quad \bar{\boldsymbol{B}} \xleftarrow{\$} \mathbb{Z}_q^{(n-1)\times \bar{n}}, \quad \forall i \in [1, N],\ \boldsymbol{S}_i \xleftarrow{\$} \mathbb{Z}_q^{\bar{n} \times m}$$

and $\boldsymbol{E}_i$ is a low-norm noise matrix. Note that $\boldsymbol{A}$ still looks uniform in $\mathbb{Z}_q^{n \times m \cdot N}$ due to the learning with errors (LWE) assumption and the leftover hash lemma. In addition, for any input x, we have

$$\begin{aligned}\boldsymbol{A}_x &= \left[\begin{pmatrix} \boldsymbol{r}^\intercal \cdot \bar{\boldsymbol{B}} \\ \bar{\boldsymbol{B}} \end{pmatrix} \cdot \boldsymbol{S}_1 + \boldsymbol{E}_1 \mid \ldots \mid \begin{pmatrix} \boldsymbol{r}^\intercal \cdot \bar{\boldsymbol{B}} \\ \bar{\boldsymbol{B}} \end{pmatrix} \cdot \boldsymbol{S}_N + \boldsymbol{E}_N\right] \cdot \boldsymbol{T} - \mathsf{P}(K, x) \cdot \boldsymbol{G} \\ &\approx \begin{pmatrix} \boldsymbol{r}^\intercal \cdot \bar{\boldsymbol{B}} \\ \bar{\boldsymbol{B}} \end{pmatrix} \cdot \left[\boldsymbol{S}_1 \mid \ldots \mid \boldsymbol{S}_N\right] \cdot \boldsymbol{T} - \mathsf{P}(K, x) \cdot \boldsymbol{G}\end{aligned}$$

The simulator also samples a random vector $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and uses the hash key $(\boldsymbol{s}, \boldsymbol{A})$ to answer the evaluation oracle queries from the adversary. Then given an input x^* and a bit u^*, the simulator computes $u^\dagger = \mathsf{H}_0((\boldsymbol{s}, \boldsymbol{A}), x^*)$. It outputs $(\boldsymbol{s}, \boldsymbol{A})$ if $u^\dagger = u^*$ and outputs $(\boldsymbol{s} + \boldsymbol{d}, \boldsymbol{A})$ otherwise, where $\boldsymbol{d} = (-1, \boldsymbol{r}^\intercal)^\intercal$.

Notice that if the partitioning is successful (i.e., $\mathsf{P}(K, x) = 0$ for all queried x and $\mathsf{P}(K, x^*) = 1$), then for any queried x, we have

$$\boldsymbol{d}^\intercal \cdot \boldsymbol{A}_x \cdot \boldsymbol{G}^{-1}(\boldsymbol{v}_1) \approx \boldsymbol{d}^\intercal \cdot \begin{pmatrix} \boldsymbol{r}^\intercal \cdot \bar{\boldsymbol{B}} \\ \bar{\boldsymbol{B}} \end{pmatrix} \cdot \left[\boldsymbol{S}_1 \mid \ldots \mid \boldsymbol{S}_N\right] \cdot \boldsymbol{T} \cdot \boldsymbol{G}^{-1}(\boldsymbol{v}_1) = 0$$

[5] Here, the adversary cannot view the hash key before submitting x^*, and this allows the simulator to choose a suitable hash key after receiving x^*.

Thus, $\mathsf{H}_0((\boldsymbol{s},\boldsymbol{A}),x) = \mathsf{H}_0((\boldsymbol{s}+\boldsymbol{d},\boldsymbol{A}),x)$ for all queried x,[6] and therefore the hash key outputted by the simulator, which is either $(\boldsymbol{s},\boldsymbol{A})$ or $(\boldsymbol{s}+\boldsymbol{d},\boldsymbol{A})$, is always compatible with its answers to the evaluation oracle. In addition,

$$\boldsymbol{d}^{\mathsf{T}} \cdot \boldsymbol{A}_{x^*} \cdot \boldsymbol{G}^{-1}(\boldsymbol{v}_1) \approx \boldsymbol{d}^{\mathsf{T}} \cdot \begin{pmatrix} \boldsymbol{r}^{\mathsf{T}} \cdot \bar{\boldsymbol{B}} \\ \bar{\boldsymbol{B}} \end{pmatrix} \cdot \begin{bmatrix} \boldsymbol{S}_1 \mid \ldots \mid \boldsymbol{S}_N \end{bmatrix} \cdot \boldsymbol{T} \cdot \boldsymbol{G}^{-1}(\boldsymbol{v}_1) - \boldsymbol{d}^{\mathsf{T}} \cdot \boldsymbol{v}_1 = \frac{q-1}{2}$$

Thus, we have $\mathsf{H}_0((\boldsymbol{s},\boldsymbol{A}),x^*) \neq \mathsf{H}_0((\boldsymbol{s}+\boldsymbol{d},\boldsymbol{A}),x^*)$, i.e., if the bit $u^* \neq \mathsf{H}_0((\boldsymbol{s},\boldsymbol{A}),x^*)$, then $u^* = \mathsf{H}_0((\boldsymbol{s}+\boldsymbol{d},\boldsymbol{A}),x^*)$. Therefore the simulator can succeed in mapping x^* to u^*.

Explainable hash with injectivity. We next describe how to construct injective explainable hash functions from the above (non-injective) explainable hash H_0. The construction runs multiple instances of H_0 and rerandomize the outputs.

More precisely, let l be the length of the inputs, then we define the new hash function as

$$\mathsf{H}(HK,x) = (\mathsf{H}_0(hk_{i,j},x) \oplus v_{i,j,x[i]})_{i\in[1,l],j\in[1,L]}$$

where $L = O(l)$ is large enough, $HK = (hk_{i,j}, v_{i,j,0}, v_{i,j,1})_{i\in[1,l],j\in[1,L]}$, and for $i \in [1,l], j \in [1,L]$, $hk_{i,j}$ is a random hash key of H_0 and $v_{i,j,0}, v_{i,j,1}$ are random bits.

For any inputs $x \neq x'$, there exists i s.t. $x[i] \neq x'[i]$. Thus, $v_{i,j,x[i]}$ and $v_{i,j,x'[i]}$ are random and independent bits and therefore, for all $j \in [1,L]$, we have

$$\Pr[\mathsf{H}_0(hk_{i,j},x) \oplus v_{i,j,x[i]} = \mathsf{H}_0(hk_{i,j},x') \oplus v_{i,j,x'[i]}] = \frac{1}{2}$$

This implies that $\Pr[\mathsf{H}(HK,x) = \mathsf{H}(HK,x')] \leq \frac{1}{2^L}$. Then, as there are at most 2^{2l} possible pairs of distinct inputs (x,x'), we have

$$\Pr[\exists x,x' \text{ s.t. } x \neq x' \ \wedge \ \mathsf{H}(HK,x) = \mathsf{H}(HK,x')] \leq \frac{2^{2l}}{2^L}$$

which can be made negligible for large enough L. That is, with all but negligible probability over the choice of the random hash key, the hash function will be injective. Besides, given an input x^* and a string $u^* \in \{0,1\}^{l\cdot L}$, the simulator of H can invoke the simulator of H_0 to generate $hk_{i,j}$ satisfying

$$\mathsf{H}_0(hk_{i,j},x^*) = u^*_{i,j} \oplus v_{i,j,x^*[i]}$$

for $i \in [1,l], j \in [1,L]$, and security of the new construction follows.

On Achieving Collusion Resistant Pseudorandomness. Next, we describe how to achieve collusion resistant pseudorandomness, which requires that given a constrained key punctured on a set $\mathcal{P}_1$ and a constrained key punctured on a set $\mathcal{P}_2$, the adversary cannot learn the PRF value at an input $x \in \mathcal{P}_1 \cap \mathcal{P}_2$[7].

[6] This also relies on the fact that $\boldsymbol{s}^{\mathsf{T}} \cdot \boldsymbol{A}_x \cdot \boldsymbol{G}^{-1}(\boldsymbol{v}_1)$ is not close to the borders (i.e., 0 and $\frac{q}{2}$), which can be guaranteed by adding an additional random element to it.

[7] Note that if $x \notin \mathcal{P}_1 \cap \mathcal{P}_2$, i.e., $x \notin \mathcal{P}_1$ or $x \notin \mathcal{P}_2$, then the PRF value at x can be trivially learned from one of the constrained keys.

The starting point is a single-key secure private puncturable PRF with special properties. Concretely, we will use the private constrained PRF given in [PS18].

The [PS18] PRF. In a nutshell, the secret key of the PRF is a vector $\boldsymbol{s} \in \mathbb{Z}_q^n$, where $\boldsymbol{s}[1] = 1$ and for $i \in [2, n]$, $\boldsymbol{s}[i]$ is a random element in $\mathbb{Z}_q$. Also, given an input x, the PRF outputs

$$\lfloor \frac{p}{q} \cdot (\boldsymbol{s}^\intercal \cdot \boldsymbol{A}_x)[1] \rceil$$

where $\boldsymbol{A}_x \in \mathbb{Z}_q^{n \times m}$ is a random matrix determined by x and some public matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{N+k}$. The constrained key for a constraint predicate $\mathtt{C}$ includes a vector

$$\begin{aligned} \boldsymbol{a}_{\mathtt{C}} = \boldsymbol{s}^\intercal \cdot [\boldsymbol{A}_1 + ct_1 \cdot \boldsymbol{G} \mid \ldots \mid \boldsymbol{A}_N + ct_N \cdot \boldsymbol{G} \\ \mid \boldsymbol{A}_{N+1} + sk_1 \cdot \boldsymbol{G} \mid \ldots \mid \boldsymbol{A}_{N+k} + sk_k \cdot \boldsymbol{G}] + \boldsymbol{e}^\intercal \end{aligned}$$

and a ciphertext $ct = (ct_1, \ldots, ct_N)$, where ct is the ciphertext that encrypts the constraint $\mathtt{C}$ using a fully homomorphic encryption (FHE) scheme, sk is the secret key of the FHE scheme, and $\boldsymbol{e}$ is a low-norm noise vector. Besides, given the constrained key $(\boldsymbol{a}_{\mathtt{C}}, ct)$ and an input x, the constrained evaluation algorithm first computes

$$\boldsymbol{a}_x \approx (\boldsymbol{s}^\intercal \cdot \boldsymbol{A}_x)[1] + (1 - \mathtt{C}(x)) \cdot r_x$$

via another version of the matrix embedding technique [BGG+14, GVW15], where r_x is a pseudorandom element in $\mathbb{Z}_q$ determined by x. Then, it rounds $\boldsymbol{a}_x$ to $\mathbb{Z}_p$ and outputs the rounding result. Note that $\boldsymbol{a}_x$ is close to $(\boldsymbol{s}^\intercal \cdot \boldsymbol{A}_x)[1]$ if $\mathtt{C}(x) = 1$, and is pseudorandom (and thus hides the real PRF value) if $\mathtt{C}(x) = 0$. Then the correctness[8] and the pseudorandomness follow. In addition, its privacy comes from security of the FHE scheme and the LWE assumption, which implies that $\boldsymbol{a}_{\mathtt{C}}$ is a pseudorandom vector and thus hides sk.

The difficulty for achieving collusion resistance. In above construction (denoted as PRF_0 here), one can recover the PRF key $\boldsymbol{s}$ from constrained keys for two different constraints $\mathtt{C}^{(1)}$ and $\mathtt{C}^{(2)}$. First, since the constraints are different, the ciphertexts $ct^{(1)}$ and $ct^{(2)}$ that encrypt $\mathtt{C}^{(1)}$ and $\mathtt{C}^{(2)}$ respectively will also be different. That is, there exists i s.t. $ct_i^{(1)} \neq ct_i^{(2)}$ and w.l.o.g., assume that $ct_i^{(1)} = 1$ and $ct_i^{(2)} = 0$. Then, from the constrained keys, one can get

$$(\boldsymbol{s}^\intercal(\boldsymbol{A}_i + ct_i^{(1)} \cdot \boldsymbol{G}) + \boldsymbol{e}_1^\intercal) - (\boldsymbol{s}^\intercal(\boldsymbol{A}_i + ct_i^{(2)} \cdot \boldsymbol{G}) + \boldsymbol{e}_2^\intercal) = \boldsymbol{s}^\intercal \cdot \boldsymbol{G} + \boldsymbol{e}^\intercal$$

from which recovering $\boldsymbol{s}$ is easy. Similar collusion attacks also work for many other lattice-based constrained PRFs (e.g., [BV15, BKM17, BTVW17]).

The first attempt. To get around the above obstacle and construct collusion resistant τ-puncturable PRFs (i.e., the PRF can be punctured at τ points), our initial idea is to split the PRF key into τ parts and puncture each part on one

[8] This also relies on the fact that $(\boldsymbol{s}^\intercal \cdot \boldsymbol{A}_x)[1]$ is not close to the "rounding border", which can be ensured either by the 1D-SIS assumption [Reg04, BV15, BKM17] or via adding an additional random element to it. In this work, we use the latter method.

input.[9] In particular, let $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_\tau$ be τ independent secret keys of PRF_0, then the new PRF key is $(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_\tau)$. Moreover, given an input x, the PRF outputs

$$\sum_{i=1}^{\tau} \mathsf{PRF}_0(\boldsymbol{t}_i, x)$$

and on input a puncture set $\mathcal{P} = \{x_1, \ldots, x_\tau\}$, the constraining algorithm punctures the secret key $\boldsymbol{t}_i$ on x_i and outputs the constrained versions of all $\boldsymbol{t}_i$.

Now, given two puncture sets $\mathcal{P}^{(1)} = \{x_1^{(1)}, \ldots, x_\tau^{(1)}\}$ and $\mathcal{P}^{(2)} = \{x_1^{(2)}, \ldots, x_\tau^{(2)}\}$ and supposing $x_1^{(1)} = x_1^{(2)} = x_1$, then $\boldsymbol{t}_1$ is punctured on the same input x_1 in the two constrained keys. Next, by the single-key security of PRF_0, $\mathsf{PRF}_0(\boldsymbol{t}_1, x_1)$ is pseudorandom given the constrained keys, which implies the pseudorandomness of the PRF value at x_1. Note that if $x_2^{(1)} \neq x_2^{(2)}$, then one can still recover $\boldsymbol{t}_2$ from the constrained keys via the attack described above. Nonetheless, this will not affect security of the $\boldsymbol{t}_1$ part, since $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are independent.

The above approach works only if we can assign the "correct" input to each part of the PRF key. Especially, let $x \in \mathcal{P}_1 \cap \mathcal{P}_2$, then we need to assign x (but no other inputs) to the same $\boldsymbol{t}_i$ in both constrained keys. We can ensure that x is assigned to a fixed $\boldsymbol{t}_i$ by using a deterministic function that maps each input into an index in $[1, \tau]$ in the constraining algorithm. However, since the input space is exponentially large, there will be collisions here and we have to puncture $\boldsymbol{t}_i$ also on some other inputs, which will cause the attacks. On the other hand, if we do not use such map, it seems impossible to assign x to the same index i in independent constraining procedures.

Our solution. To solve this problem, we generate the secret vector $\boldsymbol{t}_x$ for each punctured point x on-the-fly.[10] In more detail, the PRF key of our construction is a PRF key $\boldsymbol{s}$ of PRF_0, and the PRF outputs $\mathsf{PRF}_0(\boldsymbol{s}, x)$ given an input x. Then to puncture $\boldsymbol{s}$ on a set $\mathcal{P} = \{x_1, \ldots, x_\tau\}$, the constraining algorithm first derives $\boldsymbol{t}_{x_i}$, which is also a PRF key of PRF_0, from x_i via a standard PRF. Then it punctures $\boldsymbol{t}_{x_i}$ on x_i and computes $\boldsymbol{t}_0 = \boldsymbol{s} - \sum_{i=1}^{\tau} \boldsymbol{t}_{x_i}$. The constrained key for $\mathcal{P}$ includes $\boldsymbol{t}_0$ and the constrained version of each $\boldsymbol{t}_{x_i}$.

Correctness of PRF_0 guarantees that given a constrained key for $\mathcal{P} = \{x_1, \ldots, x_\tau\}$ and an input $x \notin \mathcal{P}$, one can compute $\mathsf{PRF}_0(\boldsymbol{t}_{x_i}, x)$ and $\mathsf{PRF}_0(\boldsymbol{t}_0, x)$.

[9] A similar idea is also employed in [BKM17] to achieve τ-puncture PRF from 1-puncture PRF. However, as discussed below, we cannot achieve collusion resistance merely from this approach.

[10] As a byproduct, this also leads to puncturable PRFs for puncture sets of unbounded sizes.

Then by the key-homomorphism property of PRF_0,[11] we have

$$\mathsf{PRF}_0(\boldsymbol{t}_0, x) + \sum_{i=1}^{\tau} \mathsf{PRF}_0(\boldsymbol{t}_{x_i}, x) = \mathsf{PRF}_0(\boldsymbol{t}_0 + \sum_{i=1}^{\tau} \boldsymbol{t}_{x_i}, x) = \mathsf{PRF}_0(\boldsymbol{s}, x)$$

and the correctness of our new construction follows.

Next, we explain why the construction has collusion resistant pseudorandomness. Given constrained keys for $\mathcal{P}_1$ and $\mathcal{P}_2$, and let $x \in \mathcal{P}_1 \cap \mathcal{P}_2$. Then $\boldsymbol{t}_x$ is punctured on the same input x in both constrained keys. Thus, the adversary cannot learn any information about $\mathsf{PRF}_0(\boldsymbol{t}_x, x)$ from the constrained version of $\boldsymbol{t}_x$ due to the single-key security of PRF_0. We also need to show that other parts of the constrained keys reveal no information about $\boldsymbol{t}_x$ and $\mathsf{PRF}_0(\boldsymbol{t}_x, x)$. Note that we have

$$\boldsymbol{t}_0^{(1)} + \sum_{x' \in \mathcal{P}_1 \setminus \{x\}} \boldsymbol{t}_{x'} = \boldsymbol{t}_0^{(2)} + \sum_{x' \in \mathcal{P}_2 \setminus \{x\}} \boldsymbol{t}_{x'} = \boldsymbol{s} - \boldsymbol{t}_x$$

where $\boldsymbol{t}_0^{(1)}$ and $\boldsymbol{t}_0^{(2)}$ are the $\boldsymbol{t}_0$ vectors in the two constrained keys. Since $\boldsymbol{s}$ and $\boldsymbol{t}_x$ are random vectors with the first coordinate set to be 1[12], $\boldsymbol{t}_x$ can be masked by $\boldsymbol{s}$ and cannot be learned from other secret vectors, namely, $\boldsymbol{t}_0^{(1)}$, $\boldsymbol{t}_0^{(2)}$ and $\boldsymbol{t}_{x'}$ for $x' \in (\mathcal{P}_1 \cup \mathcal{P}_2) \setminus \{x\}$. As $\mathsf{PRF}_0(\boldsymbol{t}_x, x)$ (and thus $\mathsf{PRF}_0(\boldsymbol{s}, x)$) is pseudorandom for an adversary that sees multiple constrained keys, the collusion resistant pseudorandomness property follows.

The above proof strategy, however, cannot be applied to prove the collusion resistant privacy of our construction. This is because the solution does not provide any protection for an input $x \in (\mathcal{P}_1 \cup \mathcal{P}_2) - (\mathcal{P}_1 \cap \mathcal{P}_2)$, given the constrained keys for $\mathcal{P}_1$ and $\mathcal{P}_2$. Thus, the adversary can still learn these inputs and know if it belongs to $\mathcal{P}_1$ or $\mathcal{P}_2$, from the constrained keys. Therefore, our construction only has 1-key privacy, which is guaranteed by the 1-key privacy of PRF_0.

Remark 1.1. The construction described above is nearly generic. In particular, it can transform a single-key secure private puncturable PRF F into a private puncturable PRF with collusion resistant pseudorandomness if (1) F is key-homomorphic and (2) the distribution of $\boldsymbol{t}_1 + \boldsymbol{t}_2$ is identical to the distribution of $\boldsymbol{t}_3$, where $\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3$ are PRF keys of F. Although there are no lattice-based private puncturable PRFs satisfying either of the properties, the transform still works for some concrete instantiations (e.g., the one presented in [PS18]) with weaker form of key-homomorphism and suitable PRF key distribution, as we have just shown.

[11] The key-homomorphism property requires that $\mathsf{PRF}_0(\boldsymbol{t}_1, x) + \mathsf{PRF}_0(\boldsymbol{t}_2, x) = \mathsf{PRF}_0(\boldsymbol{t}_1 + \boldsymbol{t}_2, x)$. Actually, due to the rounding operation, PRF_0 is only "almost key-homomorphic", i.e., there may exist a small difference between $\mathsf{PRF}_0(\boldsymbol{t}_1, x)+\mathsf{PRF}_0(\boldsymbol{t}_2, x)$ and $\mathsf{PRF}_0(\boldsymbol{t}_1 + \boldsymbol{t}_2, x)$. We close the gap by summing the variables before rounding and then rounding the result to $\mathbb{Z}_p$.

[12] Recall that both $\boldsymbol{s}$ and $\boldsymbol{t}_x$ are PRF keys of PRF_0.

Putting It All Together. We have described a general construction of adaptively secure private 1-puncturable PRF from any selectively secure private 1-puncturable PRF. Also, we have shown how to construct a private puncturable PRF with collusion resistant pseudorandomness from the private puncturable PRF given in [PS18]. Next, we explain how to combine the techniques to get a private puncturable PRF with both adaptive security and collusion resistance.

The construction proceeds in two steps. First, we apply the general construction for achieving adaptive security to the private puncturable PRF from [PS18]. This leads to an adaptively secure private 1-puncturable PRF with single-key security. Note that the new scheme has the same PRF key distribution (excluding the hash key of explainable hash) as the original one, and it is still key-homomorphic (before rounding). So, we can apply our ideas for obtaining collusion resistance to upgrade this scheme to have both adaptive security and collusion resistant pseudorandomness.

1.3 Related Work

Constrained PRFs with Additional Features. There are many works constructing (private) constrained PRFs with additional features. For example, in [CRV14, Fuc14, DDM17], constrained PRFs supporting verifiability of evaluation results are constructed. Also, in [BKW17], Boneh et al. present constrained PRFs that allow one to invert the PRF evaluation with a (constrained) key. Besides, in order to construct watermarking schemes for PRFs [CHN+16], (private) puncturable PRFs that support testing of punctured points are proposed in [BLW17, KW17, KW19]. We note that while watermarkable PRFs and puncturable PRFs are highly related, and collusion resistant watermarkable PRFs have been constructed from standard lattice assumptions in [YAYX20], the construction ideas cannot be applied to construct collusion resistant puncturable PRFs.

Private Programmable PRFs. Our notion of explainable hash is close to the notion of private programmable PRF [BLW17], which is a private puncturable PRF that allows one to reprogram the PRF output on a punctured point. It seems that a private programmable PRF with adaptive privacy and injectivity implies an explainable hash. However, existing private programmable PRFs [BLW17, PS18, PS20] only have selective security. On the other hand, a private programmable PRF with adaptive privacy can be constructed from a selectively-secure private programmable PRF and an explainable hash using the techniques provided in this work.

2 Preliminaries

In this section, we give notations and background knowledge that we require. In Appendix A, we recall definitions of cryptographic primitives that we employ.

Notations. We write $negl(\cdot)$ to denote a negligible function, and write $poly(\cdot)$ to denote a polynomial. For integers $a \le b$, we write $[a, b]$ to denote all integers from a to b. Let s be a string, we use $|s|$ to denote the length of s. For integers $a \le |s|$, we use $s[a]$ to denote the a-th character of s and for integers $a \le b \le |s|$, we use $s[a : b]$ to denote the substring $(s[a], s[a+1], \dots, s[b])$. Let $\mathcal{S}$ be a finite set, we use $|\mathcal{S}|$ to denote the size of $\mathcal{S}$, and use $s \xleftarrow{\$} \mathcal{S}$ to denote sampling an element s uniformly from set $\mathcal{S}$. Let $\mathcal{D}$ be a distribution, we use $d \leftarrow \mathcal{D}$ to denote sampling d according to $\mathcal{D}$.

We will use bold lower-case letters to denote vectors, and use bold upper-case letters to denote matrices. All elements in vectors and matrices are integers unless otherwise specified. Let $\boldsymbol{v}$ be a vector of length n, we use $\boldsymbol{v}[i]$ to denote the i-th element of $\boldsymbol{v}$ for $i \in [1, n]$ and use $\boldsymbol{v}[i : j]$ to denote the vector $(\boldsymbol{v}[i], \boldsymbol{v}[i+1], \dots, \boldsymbol{v}[j])^\intercal$ for $1 \le i < j \le n$. We use $\|\boldsymbol{v}\|_\infty = \max_{i \in [n]} |\boldsymbol{v}[i]|$ to denote the infinity-norm of $\boldsymbol{v}$. For an m-by-n matrix $\boldsymbol{A}$, we use $\boldsymbol{A}[i, j]$ to denote the element on the i-th row and the j-th columon of $\boldsymbol{A}$ for $i \in [1, m]$ and $j \in [1, n]$.

For any positive integers p, q s.t. $p \le q$, and for any $y \in \mathbb{Z}_q$, we define $\lfloor y \rceil_p = \lfloor \frac{p}{q} \cdot y \rceil \in \mathbb{Z}_p$. Without loss of generality, we use integers in $[0, q-1]$ (resp. $[0, p-1]$) to represent elements in $\mathbb{Z}_q$ (resp. $\mathbb{Z}_p$).

Leftover Hash Lemma. In this work, we use the following corollary of the leftover hash lemma [ILL89, DRS04].

Lemma 2.1. *Let λ be the security parameter. Let $q > 2$ be a prime. Let n, m be positive integers that are polynomial in λ and satisfy $m \ge n \cdot \lceil \log q \rceil + \omega(\log \lambda)$. Let $\boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, $\boldsymbol{r} \xleftarrow{\$} \{0,1\}^m$, $\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{r} \mod q$, and $\boldsymbol{b}' \xleftarrow{\$} \mathbb{Z}_q^n$. Then $(\boldsymbol{A}, \boldsymbol{b})$ is statistically indistinguishable from $(\boldsymbol{A}, \boldsymbol{b}')$.*

Discrete Gaussian Distribution. We use D_σ to denote the discrete Gaussian distribution over $\mathbb{Z}$ with standard deviation σ. Let λ be a security parameter, we use $\tilde{D}_\sigma$ to denote the truncated discrete Gaussian distribution over $\mathbb{Z}$, which samples $x \leftarrow D_\sigma$, and then it outputs x if $|x| \le \lambda \cdot \sigma$ and outputs 0 otherwise. By the following lemma, D_σ and $\tilde{D}_\sigma$ are statistically indistinguishable.

Lemma 2.2 ([Lyu12]). *For any $k > 0$, $\Pr[|z| > k\sigma : z \leftarrow D_\sigma] \le 2e^{\frac{-k^2}{2}}$.*

Gadget Matrix. For any positive integers n, m, q s.t. $m = n \cdot \lceil \log q \rceil$, we define the gadget matrix $\boldsymbol{G}_{n,q} \in \mathbb{Z}^{n \times m}$ as

$$\boldsymbol{G}_{n,q} = \begin{bmatrix} 1\ 2\ 4 \dots 2^{\lceil \log q \rceil - 1} & & & \\ & 1\ 2\ 4 \dots 2^{\lceil \log q \rceil - 1} & & \\ & & \ddots & \\ & & & 1\ 2\ 4 \dots 2^{\lceil \log q \rceil - 1} \end{bmatrix} \quad (1)$$

For any positive integer l, we also define the inverse function $\boldsymbol{G}_{n,q}^{-1} : \mathbb{Z}_q^{n \times l} \to \{0, 1\}^{m \times l}$ to be a function that decomposes each element $a \in \mathbb{Z}_q$ of a matrix into a column of size $\lceil \log q \rceil$ consisting of the binary representation of a. For any matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times l}$, we have

$$\boldsymbol{G}_{n,q} \cdot \boldsymbol{G}_{n,q}^{-1}(\boldsymbol{A}) = \boldsymbol{A}$$

The LWE Assumption. We will use the LWE assumption in this paper.

Definition 2.1 (Decision-LWE$_{n,m,q,\chi}$). *Given a random matrix $\boldsymbol{A} \in \mathbb{Z}_q^{m \times n}$, and a vector $\boldsymbol{b} \in \mathbb{Z}_q^m$, where $\boldsymbol{b}$ is generated according to either of the following two cases:*

1. $\boldsymbol{b} = \boldsymbol{A} \cdot \boldsymbol{s} + \boldsymbol{e} \mod q$*, where* $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$ *and* $\boldsymbol{e} \leftarrow \chi^m$
2. $\boldsymbol{b} \xleftarrow{\$} \mathbb{Z}_q^m$

distinguish which is the case with non-negligible advantage.

Let $m = poly(n)$, $poly(n) \leq q \leq 2^{poly(n)}$, and $\chi = D_\sigma$ (or $\tilde{D}_\sigma$) be a (truncated) discrete Gaussian error distribution with standard deviation $\sigma \geq O(\sqrt{n})$, then solving the decision-LWE$_{n,m,q,\chi}$ problem is as hard as solving the GapSVP$_\gamma$ problem on arbitrary n-dimensional lattices by a quantum algorithm [Reg05], where $\gamma = \tilde{O}(nq/\sigma)$. In subsequent works [Pei09, BLP$^+$13], classical reductions from LWE to GapSVP are also presented for different parameterizations.

Note that the hardness of the LWE problem depends only on n, q, σ, thus, we write LWE$_{n,m,q,\chi}$ as LWE$_{n,q,\chi}$ for short.

A Border Avoiding Lemma. We use the following lemma in our security proofs, which is implicitly used in [DKN$^+$20].

Lemma 2.3. *Let λ be the security parameter. Let q be a positive integer and let $\mathcal{B} \subseteq \mathbb{Z}_q$ be a set. Let l be a positive integer that is polynomial in λ and let f be a function from $\{0,1\}^l$ to $\mathbb{Z}_q$. If $q \geq 2^{l+\omega(\log \lambda)} \cdot |\mathcal{B}|$, then we have*

$$\Pr[v \xleftarrow{\$} \mathbb{Z}_q : \exists x \in \{0,1\}^l, y \in \mathcal{B} \ s.t. \ v + f(x) = y \mod q] \leq negl(\lambda)$$

Proof. For any fixed $x \in \{0,1\}^l$ and fixed $y \in \mathcal{B}$, we have

$$\Pr[v \xleftarrow{\$} \mathbb{Z}_q : v + f(x) = y \mod q] = \frac{1}{q}$$

Then by the union bound, we have

$$\Pr[v \xleftarrow{\$} \mathbb{Z}_q : \exists x \in \{0,1\}^l, y \in \mathcal{B} \text{ s.t. } v + f(x) = y \mod q] \leq \frac{2^l \cdot |\mathcal{B}|}{2^{l+\omega(\log \lambda)} \cdot |\mathcal{B}|} \leq \frac{1}{2^{\omega(\log \lambda)}}$$

which is negligible. □

3 Explainable Hash Functions

3.1 The Definition

In this section, we provide the definition of explainable hash. Roughly speaking, an explainable hash is an injective function that can generate a hash key mapping a given input to a predefined output and being compatible with previous hash evaluations. Formally, an explainable hash $\mathsf{H} = (\mathtt{KeyGen}, \mathtt{Eval})$ with input space $\mathcal{X}$ and output space $\mathcal{U}$ consists of the following probabilistic polynomial-time (PPT) algorithms:

- $\mathtt{KeyGen}(1^\lambda) \to hk$: On input the security parameter 1^λ, the key generation algorithm outputs the hash key hk.
- $\mathtt{Eval}(hk, x) \to u$: On input the hash key hk and an input $x \in \mathcal{X}$, the deterministic evaluation algorithm outputs an output $u \in \mathcal{U}$.

We require that the hash function is injective for nearly all hash keys.

Definition 3.1 (Injectivity). *Let $hk \leftarrow \mathtt{KeyGen}(1^\lambda)$, then the probability that there exist distinct $x_1, x_2 \in \mathcal{X}$ s.t. $\mathtt{Eval}(hk, x_1) = \mathtt{Eval}(hk, x_2)$ is negligible.*

Besides, its explainability property requires that there exists a simulator that can simulate the hash function evaluation oracle to an adversary, and then after the adversary submits an input, the simulator can generate a hash key that maps this input to a predefined uniform output and is compatible with previous evaluation oracle outputs. Here, we allow the simulator to abort with a non-negligible probability and require that the simulator aborts if and only if the inputs submitted to the evaluation oracle and the final input do not pass a validity check algorithm.

Definition 3.2 (Explainability). *For any polynomial Q and non-negligible real value $\delta \in (0, 1]$, we first define two algorithms $(\mathtt{VKeyGen}_{Q,\delta}, \mathtt{Verify}_{Q,\delta})$ as follows:*

- $\mathtt{VKeyGen}_{Q,\delta}(1^\lambda) \to vk$: *On input the security parameter 1^λ, the verification key generation algorithm outputs the verification key vk.*
- $\mathtt{Verify}_{Q,\delta}(vk, \mathcal{Q}, x^*) \to \alpha$: *On input the verification key vk, a set $\mathcal{Q} \subset \mathcal{X}$ s.t. $|\mathcal{Q}| \leq Q$ and an input $x^* \in \mathcal{X}$, the deterministic verification algorithm outputs a bit $\alpha \in \{0, 1\}$.*

The explainability property has the following two requirements:

- ***Abort Probability.*** *There exists $\mathit{\Gamma}_{min}$, $\mathit{\Gamma}_{max}$ that for any set $\mathcal{Q} \subset \mathcal{X}$ s.t. $|\mathcal{Q}| \leq Q$ and for any input $x^* \in \mathcal{X} \backslash \mathcal{Q}$*

$$\mathit{\Gamma}_{min} \leq \Pr\left[vk \leftarrow \mathtt{VKeyGen}_{Q,\delta}(1^\lambda) : \mathtt{Verify}_{Q,\delta}(vk, \mathcal{Q}, x^*) = 1\right] \leq \mathit{\Gamma}_{max}$$

  and

$$\mathcal{T} = \mathit{\Gamma}_{min} \cdot \delta - (\mathit{\Gamma}_{max} - \mathit{\Gamma}_{min})$$

  is a non-negligible positive real value.
- ***Indistinguishability.*** *There exists a* stateful *simulator* $\mathtt{SIM}$ *such that for any PPT and* stateful *adversary $\mathcal{A}$, we have*

$$|\Pr[\mathtt{ExpReal}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathtt{ExpIdeal}_{\mathcal{A},\mathtt{SIM}}(1^\lambda) = 1]| \leq negl(\lambda)$$

where the experiments `ExpReal` *and* `ExpIdeal` *are defined as follows*

$\texttt{ExpReal}_{\mathcal{A}}(1^\lambda)$:

$vk \leftarrow \texttt{VKeyGen}_{Q,\delta}(1^\lambda)$;

$hk \leftarrow \texttt{KeyGen}(1^\lambda)$;

$x^* \leftarrow \mathcal{A}^{\texttt{Eval}(hk,\cdot)}(1^\lambda)$;

If $\texttt{Verify}_{Q,\delta}(vk, \mathcal{Q}, x^*) = 1$:

$out = (hk, \texttt{Eval}(hk, x^*))$;

Otherwise :

$out = \perp$;

$b \leftarrow \mathcal{A}(out)$;

Output b;

$\texttt{ExpIdeal}_{\mathcal{A},\texttt{SIM}}(1^\lambda)$:

$vk \leftarrow \texttt{VKeyGen}_{Q,\delta}(1^\lambda)$;

$u^* \xleftarrow{\$} \mathcal{U}$;

$\texttt{SIM}(vk, u^*)$;

$x^* \leftarrow \mathcal{A}^{\texttt{SIM}(\cdot)}(1^\lambda)$;

If $\texttt{Verify}_{Q,\delta}(vk, \mathcal{Q}, x^*) = 1$:

$hk \leftarrow \texttt{SIM}(x^*)$;

$out = (hk, u^*)$;

Otherwise :

$out = \perp$;

$b \leftarrow \mathcal{A}(out)$;

Output b;

In above experiments, $\mathcal{Q}$ *is the set of inputs submitted to the (simulated) evaluation oracle and we require that (1)* $|\mathcal{Q}| \leq Q$ *and (2)* $x^* \notin \mathcal{Q}$*. In the third step of the experiment* `ExpIdeal`*, the stateful simulator* `SIM` *takes as input* (vk, u^*) *and updates its internal state, but it does not output anything in this step.*

3.2 The Construction

In this section, we present our construction of explainable hash. Let λ be the security parameter. Let l, k, t be positive integers that are polynomial in λ such that $k = 4l + \lambda$. Let $\bar{n}, n, m, \sigma, \Sigma$ be positive integers that are polynomial in λ, and let q be a positive odd prime, which satisfy: $m = n \cdot \lceil \log q \rceil$, $n = \bar{n} \cdot \lceil \log q \rceil + \lambda$, $\Sigma = 2tm^3\lambda\sigma$, and $q \geq 2^{l+\omega(\log \lambda)}(4\Sigma + 2)$. Let

$$\mathsf{H}_{adm} : \{0,1\}^l \to \{0,1\}^t$$

be a balanced admissible hash function and let

$$\texttt{EvalAdm} : (\mathbb{Z}_q^{n\times m})^{2t} \times \{0,1\}^t \to \mathbb{Z}_q^{n\times m}$$

be the algorithm defined in Lemma A.7. Let $\boldsymbol{G} = \boldsymbol{G}_{n,q}$ and write $\boldsymbol{G}_{n,q}^{-1}$ as $\boldsymbol{G}^{-1}$. Let

$$\boldsymbol{h} = \boldsymbol{G}^{-1}((\frac{q-1}{2}, 0, \ldots, 0)^\intercal) \in \{0,1\}^m$$

We construct the explainable hash function $\mathsf{H} = (\texttt{KeyGen}, \texttt{Eval})$, which has input space $\{0,1\}^l$ and output space $\{0,1\}^{l\cdot k}$ as follows:

- `KeyGen`. On input the security parameter 1^λ, the key generation algorithm samples

$$\begin{aligned} &\boldsymbol{A}_z \xleftarrow{\$} \mathbb{Z}_q^{n\times m} && \text{for } z \in [1, 2t] \\ &\boldsymbol{s}_{i,j} \xleftarrow{\$} \mathbb{Z}_q^{n} && \text{for } i \in [1, l], j \in [1, k] \\ &v_{i,j,\iota} \xleftarrow{\$} \mathbb{Z}_q && \text{for } i \in [1, l], j \in [1, k], \iota \in \{0, 1\} \end{aligned}$$

  and outputs the hash key

$$hk = ((\boldsymbol{A}_z)_{z\in[1,2t]}, (\boldsymbol{s}_{i,j})_{i\in[1,l],j\in[1,k]}, (v_{i,j,\iota})_{i\in[1,l],j\in[1,k],\iota\in\{0,1\}})$$

- `Eval`. On input the hash key $hk = ((\boldsymbol{A}_z)_{z\in[1,2t]}, (\boldsymbol{s}_{i,j})_{i\in[1,l],j\in[1,k]}, (v_{i,j,\iota})_{i\in[1,l],j\in[1,k],\iota\in\{0,1\}})$, and an input $x \in \{0, 1\}^l$, the evaluation algorithm first computes:

$$w = \mathsf{H}_{adm}(x), \quad \boldsymbol{A}_w = \texttt{EvalAdm}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{2t}, w), \quad \boldsymbol{b}_w = \boldsymbol{A}_w \cdot \boldsymbol{h} \mod q$$

  Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_l$ be k-dimension binary vectors, then for $i \in [1, l]$, $j \in [1, k]$, it computes:

$$y_{i,j} = \boldsymbol{s}_{i,j}^{\intercal} \cdot \boldsymbol{b}_w + v_{i,j,x[i]} \mod q$$

  and sets

$$\boldsymbol{u}_i[j] = \begin{cases} 0 & \text{if } y_{i,j} \in [0, \frac{q-1}{2}] \\ 1 & \text{otherwise} \end{cases}$$

  Finally, it outputs

$$\boldsymbol{u} = (\boldsymbol{u}_1^{\intercal}, \ldots, \boldsymbol{u}_l^{\intercal})^{\intercal}$$

Theorem 3.1. *If H_{adm} is a balanced admissible hash as defined in Definition A.1, then H is a secure explainable hash assuming the hardness of $LWE_{\bar{n},q,\tilde{D}_\sigma}$.*

We present proof of Theorem 3.1 in Appendix B.

Parameters. Next, we give an instantiation for the parameters of H. Security of H relies on the hardness of $\mathrm{LWE}_{\bar{n},q,\tilde{D}_\sigma}$. In addition, we require that

$$q \geq 2^{l+\omega(\log\lambda)} \cdot (4\Sigma + 2) \geq 2^{l+\omega(\log\lambda)} \cdot poly(\lambda) \geq 2^{l+\omega(\log\lambda)}$$

Let $\varepsilon \in (0, 1)$ be a constant real value. We set $\bar{n} = (l + \lambda)^{\frac{1}{\varepsilon}}$ and set $q = 2^{O(l+\lambda)}$. This makes the approximation factor $\gamma = O(\bar{n}q/\sigma)$ of the underlying worst-case lattices problems to be $2^{O(\bar{n}^\varepsilon)}$.

Now, assume that the input length $l = O(\lambda)$, then the output length will be in $O(\lambda^2)$ and the hash key size will be

$$|hk| = 2t \cdot nm\lceil\log q\rceil + lk \cdot n\lceil\log q\rceil + 2lk \cdot \lceil\log q\rceil = O(\lambda^{5+\frac{2}{\varepsilon}})$$

4 Private Puncturable PRFs

4.1 The Definition

In this section, we provide the definition of private puncturable PRF, which is adapted from definitions in previous works (e.g., [BW13, HKW15, BLW17, BKM17, DKN+20]). More precisely, a private puncturable pseudorandom function $\mathsf{PRF} = (\texttt{KeyGen}, \texttt{Eval}, \texttt{Constrain}, \texttt{ConstrainEval})$ with key space $\mathcal{K}$, input space $\mathcal{X}$, and output space $\mathcal{Y}$ consists of the following four PPT algorithms:

- $\texttt{KeyGen}(1^\lambda) \to k$: On input the security parameter 1^λ, the key generation algorithm outputs the secret key $k \in \mathcal{K}$.
- $\texttt{Eval}(k, x) \to y$: On input the secret key $k \in \mathcal{K}$ and an input $x \in \mathcal{X}$, the evaluation algorithm outputs an output $y \in \mathcal{Y}$.
- $\texttt{Constrain}(k, \mathcal{P}) \to ck$: On input the secret key $k \in \mathcal{K}$ and a polynomial-size set[13] $\mathcal{P} \subset \mathcal{X}$, the constraining algorithm outputs a constrained key ck.
- $\texttt{ConstrainEval}(ck, x) \to y$: On input the constrained key ck and an input $x \in \mathcal{X}$, the constrained evaluation algorithm outputs an output $y \in \mathcal{Y}$.

Besides, it should satisfy the correctness, pseudorandomness, and privacy properties defined as follows.

Correctness. The correctness of a private puncturable PRF requires that the constrained key can preserve the functionality of the PRF on unpunctured points. In this work, we consider a statistical notion of correctness.

Definition 4.1 (Correctness). *Let $k \leftarrow \texttt{KeyGen}(1^\lambda)$, then the probability that there exists polynomial-size set $\mathcal{P}^* \subset \mathcal{X}$, input $x^* \in \mathcal{X} \backslash \mathcal{P}^*$, and constrained key $ck \leftarrow \texttt{Constrain}(k, \mathcal{P}^*)$ satisfying $\texttt{Eval}(k, x^*) \neq \texttt{ConstrainEval}(ck, x^*)$ is negligible.*

Pseudorandomness. The pseudorandomness of a private puncturable PRF requires that given a constrained key, the PRF values at the punctured points are pseudorandom. As shown in [BKM17], this property implies the standard pseudorandomness of the PRF.

In this work, we consider adaptive collusion resistant pseudorandomness, i.e., the adversary can make queries to the evaluation oracle and the constrain oracle both before and after seeing the challenge in an adaptive manner, and it can make a priori unbounded number of queries to the constrain oracle.

Definition 4.2 (Pseudorandomness). *For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have*

$$\Pr[b \xleftarrow{\$} \{0,1\}, \texttt{ExpPR}_{\mathcal{A},b}(1^\lambda) = 1] \leq 1/2 + negl(\lambda)$$

[13] We implicitly assume that a set $\mathcal{P}$ is described by listing all elements in $\mathcal{P}$, thus, the puncture set is always of polynomial-size in this paper.

$\texttt{ExpPR}_{\mathcal{A},b}(1^\lambda)$:	$\texttt{ExpPriv}_{\mathcal{A},b}(1^\lambda)$:
1. $k \leftarrow \texttt{KeyGen}(1^\lambda)$;	1. $k \leftarrow \texttt{KeyGen}(1^\lambda)$;
2. $(x^*, state) \leftarrow \mathcal{A}_1^{\texttt{Eval}(k,\cdot),\texttt{Constrain}(k,\cdot)}(1^\lambda)$;	2. $(\mathcal{P}_0^*, \mathcal{P}_1^*, state) \leftarrow \mathcal{A}_1^{\texttt{Eval}(k,\cdot)}(1^\lambda)$;
3. $y_0^* = \texttt{Eval}(k, x^*)$, $y_1^* \xleftarrow{\$} \mathcal{Y}$;	3. $ck^* \leftarrow \texttt{Constrain}(k, \mathcal{P}_b^*)$;
4. $b' \leftarrow \mathcal{A}_2^{\texttt{Eval}(k,\cdot),\texttt{Constrain}(k,\cdot)}(state, y_b^*)$;	4. $b' \leftarrow \mathcal{A}_2^{\texttt{Eval}(k,\cdot)}(state, ck^*)$;
5. If $b = b'$, output 1; If $b \neq b'$, output 0.	5. Output b';

Fig. 1 The experiments $\texttt{ExpPR}$ and $\texttt{ExpPriv}$.

where the experiment $\texttt{ExpPR}$ *is defined in Figure 1. Let* $x_1, \ldots, x_{Q_e}$ *be the inputs submitted to the evaluation oracle* $\texttt{Eval}(\texttt{k}, \cdot)$ *and let* $\mathcal{P}_1, \ldots, \mathcal{P}_{Q_c}$ *be the sets submitted to the constrain oracle* $\texttt{Constrain}(k, \cdot)$*. To prevent the adversary from trivially winning in the experiment, we require that:*

$$\forall i \in [1, Q_e], x^* \neq x_i \quad \wedge \quad \forall i \in [1, Q_c], x^* \in \mathcal{P}_i \tag{2}$$

Remark 4.1 (Weak Adaptivity). We say that a private puncturable PRF has *weakly adaptive pseudorandomness* if the adversary $\mathcal{A}_1$ in Definition 4.2 is not allowed to query the constrain oracle.[14]

The following theorem states that weakly adaptive pseudorandomness implies the fully adaptive pseudorandomness defined in Definition 4.2, and we provide the proof of Theorem 4.1 in Appendix C.

Theorem 4.1. *Let* PRF *be a private puncturable PRF with weakly adaptive pseudorandomness, then it also satisfies the pseudorandomness property defined in Definition 4.2.*

Privacy. The privacy of a private puncturable PRF requires that the constrained key can hide the punctured points. In this work, we consider adaptive 1-key privacy, i.e., the adversary can only obtain 1 constrained key, and it can make queries to the evaluation oracle both before and after seeing the constrained key adaptively.

Definition 4.3 (Privacy). *For any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, we have*

$$|\Pr[\texttt{ExpPriv}_{\mathcal{A},0}(1^\lambda) = 1] - \Pr[\texttt{ExpPriv}_{\mathcal{A},1}(1^\lambda) = 1]| \leq negl(\lambda)$$

where the experiment $\texttt{ExpPriv}$ *is defined in Figure 1. Let* $x_1, \ldots, x_{Q_e}$ *be the inputs submitted to the evaluation oracle* $\texttt{Eval}(\texttt{k}, \cdot)$*, to prevent the adversary from trivially winning in the experiment, we require that:*

$$\forall i \in [1, Q_e], (x_i \in \mathcal{P}_0^* \wedge x_i \in \mathcal{P}_1^*) \vee (x_i \notin \mathcal{P}_0^* \wedge x_i \notin \mathcal{P}_1^*) \tag{3}$$

Besides, we require that

$$|\mathcal{P}_0^*| = |\mathcal{P}_1^*| \tag{4}$$

[14] In this setting, $\mathcal{A}_1$ can still make queries to the evaluation oracle, and $\mathcal{A}_2$ can still query the evaluation oracle and the constrain oracle adaptively for a priori unbounded number of times.

Remark 4.2. The requirement of Equation (4) is *necessary* if $|\mathcal{X}|$ is superpolynomial in λ and we do not have an a priori bound on the sizes of the puncture sets. In particular, assume there exists a private puncturable PRF $\mathsf{PRF} = (\mathtt{KeyGen}, \mathtt{Eval}, \mathtt{Constrain}, \mathtt{ConstrainEval})$ that can achieve privacy without requiring Equation (4). Let x be an arbitrary input, let n be the upper bound on the size of $ck \leftarrow \mathtt{Constrain}(K, \{x\})$, and let $\mathcal{U} \subset \mathcal{X}$ be an arbitrary set with $(n + \lambda)$ inputs. Also, let $\mathcal{P}$ be a random subset of $\mathcal{U}$ and let $ck' \leftarrow \mathtt{Constrain}(K, \mathcal{P})$. Then with all but negligible probability, we have $|ck'| \leq n$ as otherwise, the adversary can distinguish the constrained key for x from the constrained key for $\mathcal{P}$ by comparing the lengths of the constrained keys. In addition, given a constrained key ck', we can test if ck' is punctured on an input $x \in \mathcal{U}$ via checking if $\mathtt{Eval}(K, x) \neq \mathtt{ConstrainEval}(ck', x)$, this will succeed with all but negligible probability due to the correctness and the pseudorandomness of PRF. That is, from PRF, we can construct a mechanism that compresses a random $(n + \lambda)$-bit string[15] into an $(n + 1)$-bit code[16] and then recover the input from the code, with all but negligible probability. This is information theoretically impossible.

Remark 4.3. Previous works on puncturable PRFs (e.g., [HKW15,BKM17,PS18]) mainly consider the τ-puncturable PRF, where the sizes of the puncture sets should be equal to (or not greater than) a predefined polynomial τ.[17] One can construct τ-puncturable PRF from our puncturable PRF $\mathsf{PRF} = (\mathtt{KeyGen}, \mathtt{Eval}, \mathtt{Constrain}, \mathtt{ConstrainEval})$ as follows:

- $\mathtt{KeyGen}'(1^\lambda, 1^\tau)$. Output $k \leftarrow \mathtt{KeyGen}(1^\lambda)$.
- $\mathtt{Eval}'(k, x)$. Output $y = \mathtt{Eval}(k, 0\|x)$.
- $\mathtt{Constrain}'(k, \mathcal{P})$. Pad $\mathcal{P}' = \{0\|x\}_{x \in \mathcal{P}} \cup \{1\|\bar{x}_i\}_{i \in [\tau - |\mathcal{P}|]}$ and output $ck \leftarrow \mathtt{Constrain}(k, \mathcal{P}')$.
- $\mathtt{ConstrainEval}'(ck, x)$. Output $y = \mathtt{ConstrainEval}(ck, 0\|x)$.

where $\bar{x}_i$ are some random inputs and are used to pad the puncture set $\mathcal{P}$. Note that the real input and the dummy inputs for padding $\mathcal{P}$ have different prefix. It is easy to check that correctness, pseudorandomness, and privacy of the new construction follow from the security properties of PRF. Especially, after padding all puncture sets to be of size τ, Equation (4) in the privacy game will always be satisfied. In contrast, it seems difficult to extend existing constructions of τ-puncturable PRF to be the puncturable PRF defined in this work.

Remark 4.4. A simulation-based definition, which can capture the correctness, pseudorandomness, and privacy in a single definition, is used in [CC17, PS18]. As shown in [CC17], our indistinguishability-based definitions (from Definition 4.1 to Definition 4.3) implies this simulation-based definition.

15 Note that there are $2^{n+\lambda}$ possible subsets of $\mathcal{U}$.

16 We can use $(n+1)$-bit strings to represent all strings with length not larger than n.

17 Since the sizes of the puncture sets are a priori bounded, the restriction described by Equation (4) is not needed.

4.2 The Construction

In this section, we present our main construction of private puncturable PRF. Our construction can be roughly divided into two steps. In this first step, we construct an adaptively secure private puncturable PRF from the selectively secure private puncturable PRF given in [PS18], and then in the second step, we upgrade the scheme to have collusion resistant pseudorandomness. An overview for how both steps proceed and how to combine the steps is provided in Sec. 1.2, and below we give a concrete construction of private puncturable PRF with both adaptive security and collusion resistant pseudorandomness from scratch.

Let λ be the security parameter. Let $l, L, c, k, n, m, q, p, \varkappa, N, \sigma, \Sigma', \Sigma, d, D$ be positive integers that satisfy:

- l, L, c, k, n, p, σ are polynomial in λ.
- $\varkappa = \lceil \log q \rceil$, $m = n \cdot \varkappa$, and $N = (L + \varkappa) \cdot c$.
- $d = O(\log L) = O(\log \lambda)$.
- $D = d \cdot poly(\log \lambda, \log \log q) = poly(\log \lambda, \log \log q)$.
- $\Sigma' \geq \varkappa \cdot \lambda \cdot \sigma \cdot (k^{O(d)} + k \cdot m^{O(D)})$.
- $\Sigma \geq 2^{\omega(\log \lambda)} \cdot \Sigma'$.
- $q \geq 2^{L+\omega(\log \lambda)} \cdot p \cdot (2\Sigma + 1)$.
- p is an odd prime and q is a power of p.

Let $\boldsymbol{G} = \boldsymbol{G}_{n,q}$ and write $\boldsymbol{G}_{n,q}^{-1}$ as $\boldsymbol{G}^{-1}$. Let $\mathtt{GS} : \mathcal{R}_{\mathtt{GS}} \to \mathbb{Z}^m$ be an algorithm that takes as input a random string from its randomness space $\mathcal{R}_{\mathtt{GS}}$ and outputs an m-dimension vector $\boldsymbol{e}$ that follows the truncated discrete Gaussian distribution $\tilde{D}_{\sigma}^{m}$.

The construction is built on the following building blocks:

- The GSW fully homomorphic encryption scheme $\mathsf{FHE} = (\mathsf{FHE}.\mathtt{KeyGen}, \mathsf{FHE}.\mathtt{Enc}, \mathsf{FHE}.\mathtt{Dec}, \mathsf{FHE}.\mathtt{Eval})$, where the message space is $\{0,1\}$, the ciphertext space is $\{0,1\}^c$, and the secret key space is $\mathbb{Z}_q^k$. Here, we use $\mathcal{R}_{\mathtt{KeyGen}}$ and $\mathcal{R}_{\mathtt{Enc}}$ to denote the randomness space for the algorithms $\mathsf{FHE}.\mathtt{KeyGen}$ and $\mathsf{FHE}.\mathtt{Enc}$ respectively.
- An explainable hash function $\mathsf{H} = (\mathsf{H}.\mathtt{KeyGen}, \mathsf{H}.\mathtt{Eval})$ with input space $\{0,1\}^l$ and output space $\{0,1\}^L$.
- A PRF $\mathsf{F} = (\mathsf{F}.\mathtt{KeyGen}, \mathsf{F}.\mathtt{Eval})$ with input space $\{0,1\}^L$ and output space $\mathcal{R}_{\mathtt{KeyGen}} \times \mathcal{R}_{\mathtt{Enc}}^{L+\varkappa} \times \mathbb{Z}_q \times \mathbb{Z}_q^{n-1} \times \mathcal{R}_{\mathtt{GS}}^{N+k+1}$.

Besides, for any $u \in \{0,1\}^L$ and $j \in [1, \varkappa]$, we define $\mathtt{eq}_{u,j} : \{0,1\}^L \times \{0,1\}^{\varkappa} \to \{0,1\}$ of depth d as

$$\mathtt{eq}_{u,j}(u^*, r) = \begin{cases} r[j] & \text{if } u^* = u \\ 0 & \text{otherwise} \end{cases}$$

and for any $u \in \{0,1\}^L$, $j \in [1, \varkappa]$, and $\iota \in [1, k]$, we define $\mathtt{C}_{u,j,\iota} : \{0,1\}^N \to \{0,1\}$ of depth D as

$$\mathtt{C}_{u,j,\iota}(\boldsymbol{ct}) = \mathsf{FHE}.\mathtt{Eval}(j, \mathtt{eq}_{u,j}, \boldsymbol{ct})[\iota]$$

Let

$$\texttt{EvalPK} : \mathcal{C}_{N,D} \times (\mathbb{Z}_q^{n\times m})^{N+1} \to \mathbb{Z}_q^{n\times m}$$
$$\texttt{EvalCT} : \mathcal{C}_{N,D} \times (\mathbb{Z}_q^{n\times m})^{N+1} \times (\mathbb{Z}_q^{m})^{N+1} \times \{0,1\}^N \to \mathbb{Z}_q^{m}$$

be the algorithms defined in Lemma A.1, where we use $\mathcal{C}_{N,D}$ to denote the set of polynomial-size circuit from $\{0,1\}^N \to \{0,1\}$ with depth at most D, and let

$$\texttt{IPEvalPK} : (\mathbb{Z}_q^{n\times m})^{k} \times (\mathbb{Z}_q^{n\times m})^{k} \to \mathbb{Z}_q^{n\times m}$$
$$\texttt{IPEvalCT} : (\mathbb{Z}_q^{n\times m})^{k} \times (\mathbb{Z}_q^{m})^{k} \times (\mathbb{Z}_q^{m})^{k} \times \{0,1\}^k \to \mathbb{Z}_q^{m}$$

be the algorithms defined in Lemma A.2.

We construct the private puncturable PRF $\mathsf{PRF} = (\texttt{KeyGen}, \texttt{Eval}, \texttt{Constrain}, \texttt{ConstrainEval})$ with input space $\{0,1\}^l$ and output space $\mathbb{Z}_p$ as follows:

- $\texttt{KeyGen}$. On input the security parameter 1^λ, the key generation algorithm generates:

$$\boldsymbol{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m} \quad \text{for } i \in [0, N]$$
$$\boldsymbol{B}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m} \quad \text{for } i \in [1, k]$$
$$\bar{\boldsymbol{s}} \xleftarrow{\$} \mathbb{Z}_q^{n-1} \qquad \boldsymbol{s} = (1, \bar{\boldsymbol{s}}^\intercal)^\intercal \qquad v \xleftarrow{\$} \mathbb{Z}_q$$
$$k_{\mathsf{H}} \leftarrow \mathsf{H}.\texttt{KeyGen}(1^\lambda) \qquad k_{\mathsf{F}} \leftarrow \mathsf{F}.\texttt{KeyGen}(1^\lambda)$$

  and outputs the PRF key

$$K = ((\boldsymbol{A}_i)_{i\in[0,N]}, (\boldsymbol{B}_i)_{i\in[1,k]}, \boldsymbol{s}, v, k_{\mathsf{H}}, k_{\mathsf{F}})$$

- $\texttt{Eval}$. On input the PRF key $K = ((\boldsymbol{A}_i)_{i\in[0,N]}, (\boldsymbol{B}_i)_{i\in[1,k]}, \boldsymbol{s}, v, k_{\mathsf{H}}, k_{\mathsf{F}})$ and an input $x \in \{0,1\}^l$, the evaluation algorithm first computes $u = \mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x)$. Then it computes

$$\boldsymbol{C}_{j,\iota} = \texttt{EvalPK}(\mathsf{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N) \quad \text{for } j \in [1, \varkappa], \iota \in [1, k]$$

  and

$$\boldsymbol{D}_j = \texttt{IPEvalPK}(\boldsymbol{C}_{j,1}, \ldots, \boldsymbol{C}_{j,k}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k) \quad \text{for } j \in [1, \varkappa]$$

  Finally, it computes

$$\bar{y} = (\sum_{j=1}^{\varkappa} \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j)[1] + v \mod q$$

  and outputs

$$y = \lfloor \bar{y} \rceil_p \mod p$$

- $\texttt{Constrain}$. The constraining algorithm takes as input the PRF key $K = ((\boldsymbol{A}_i)_{i\in[0,N]}, (\boldsymbol{B}_i)_{i\in[1,k]}, \boldsymbol{s}, v, k_{\mathsf{H}}, k_{\mathsf{F}})$ and a set $\mathcal{P} \subset \{0,1\}^l$.
  Let $\mathcal{P} = \{x_1, \ldots, x_{|\mathcal{P}|}\}$, then for $i \in [1, |\mathcal{P}|]$, the constraining algorithm first prepares:
  1. $u_i = \mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x_i)$.

2. $(R_{\mathtt{K},i}, (R_{\mathtt{E},i,j})_{j\in[1,L+\varkappa]}, \bar{r}_i, \bar{\boldsymbol{t}}_i, (R_{\mathtt{G},i,j})_{j\in[0,N+k]}) = \mathsf{F}.\mathtt{Eval}(k_{\mathsf{F}}, u_i)$.
3. $\boldsymbol{t}_i = \begin{pmatrix} 1 \\ \bar{\boldsymbol{t}}_i \end{pmatrix}$.
4. $\boldsymbol{r}_i = \boldsymbol{G}_{1,q}^{-1}(\bar{r}_i)$.
5. $(pk_i, sk_i) = \mathsf{FHE}.\mathtt{KeyGen}(1^\lambda, 1^d; R_{\mathtt{K},i})$.
6. $\boldsymbol{e}_{i,j} = \mathtt{GS}(R_{\mathtt{G},i,j})$ for $j \in [0, N+k]$.

Next, it computes the ciphertexts

$$ct_{i,j} = \mathsf{FHE}.\mathtt{Enc}(pk_i, u_i[j]; R_{\mathtt{E},i,j}) \quad \text{for } j \in [1, L]$$

$$ct_{i,L+j} = \mathsf{FHE}.\mathtt{Enc}(pk_i, \boldsymbol{r}_i[j]; R_{\mathtt{E},i,L+j}) \quad \text{for } j \in [1, \varkappa]$$

and encodes the ciphertexts and the secret key into matrices as

$$\boldsymbol{a}_{i,0}^\intercal = \boldsymbol{t}_i^\intercal \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_{i,0}^\intercal$$

$$\boldsymbol{a}_{i,j}^\intercal = \boldsymbol{t}_i^\intercal \cdot (\boldsymbol{A}_j + ct_i[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{i,j}^\intercal \quad \text{for } j \in [1, N]$$

$$\boldsymbol{b}_{i,j}^\intercal = \boldsymbol{t}_i^\intercal \cdot (\boldsymbol{B}_j + sk_i[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{i,N+j}^\intercal \quad \text{for } j \in [1, k]$$

where $ct_i = (ct_{i,1} \| \ldots \| ct_{i,L+\varkappa})$.
Besides, it computes

$$\boldsymbol{t}_0 = \boldsymbol{s} - \sum_{i=1}^{|\mathcal{P}|} \boldsymbol{t}_i$$

and generates encodings of 0 as

$$\begin{aligned}
&\boldsymbol{e}_{0,0} \leftarrow \tilde{D}_\sigma^m, \quad \boldsymbol{a}_{0,0}^\intercal = \boldsymbol{t}_0^\intercal \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_{0,0}^\intercal \\
&\boldsymbol{e}_{0,j} \leftarrow \tilde{D}_\sigma^m, \quad \boldsymbol{a}_{0,j}^\intercal = \boldsymbol{t}_0^\intercal \cdot (\boldsymbol{A}_j + 0 \cdot \boldsymbol{G}) + \boldsymbol{e}_{0,j}^\intercal \quad \text{for } j \in [1, N] \\
&\boldsymbol{e}_{0,N+j} \leftarrow \tilde{D}_\sigma^m, \quad \boldsymbol{b}_{0,j}^\intercal = \boldsymbol{t}_0^\intercal \cdot (\boldsymbol{B}_j + 0 \cdot \boldsymbol{G}) + \boldsymbol{e}_{0,N+j}^\intercal \quad \text{for } j \in [1, k]
\end{aligned}$$

Finally, the algorithm outputs:

$$\begin{aligned}
CK = (&(\boldsymbol{A}_i)_{i\in[0,N]}, (\boldsymbol{B}_i)_{i\in[1,k]}, v, k_{\mathsf{H}}, \\
&(\boldsymbol{a}_{0,j})_{j\in[0,N]}, (\boldsymbol{b}_{0,j})_{j\in[1,k]}, \\
&\{(\boldsymbol{a}_{i,j})_{j\in[0,N]}, (\boldsymbol{b}_{i,j})_{j\in[1,k]}, ct_i\}_{i\in[1,|\mathcal{P}|]})
\end{aligned}$$

- `ConstrainEval`. On input the constrained key $CK = ((\boldsymbol{A}_i)_{i\in[0,N]}, (\boldsymbol{B}_i)_{i\in[1,k]}, v, k_{\mathsf{H}}, (\boldsymbol{a}_{0,j})_{j\in[0,N]}, (\boldsymbol{b}_{0,j})_{j\in[1,k]}, \{(\boldsymbol{a}_{i,j})_{j\in[0,N]}, (\boldsymbol{b}_{i,j})_{j\in[1,k]}, ct_i\}_{i\in[1,P]})$ and an input $x \in \{0,1\}^l$, the constrained evaluation algorithm first computes $u = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x)$. Let $ct_0 = 0^N$, then for $i \in [0, P]$, $j \in [1, \varkappa]$, and $\iota \in [1, k]$, the algorithm computes

$$\widetilde{ct}_{i,j,\iota} = \mathtt{C}_{u,j,\iota}(ct_i)$$

$$\boldsymbol{c}_{i,j,\iota} = \mathtt{EvalCT}(\mathtt{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{a}_{i,0}, \ldots, \boldsymbol{a}_{i,N}, ct_i)$$

Also, for $i \in [0, P]$ and $j \in [1, \varkappa]$, it computes

$$\boldsymbol{d}_{i,j} = \texttt{IPEvalCT}(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_k, \boldsymbol{c}_{i,j,1}, \ldots, \boldsymbol{c}_{i,j,k}, \boldsymbol{b}_{i,1}, \ldots, \boldsymbol{b}_{i,k}, \widetilde{ct}_{i,j,1}, \ldots, \widetilde{ct}_{i,j,k})$$

Finally, it computes

$$\bar{y} = (\sum_{i=0}^{P} \sum_{j=1}^{\varkappa} \boldsymbol{d}_{i,j})[1] + v \mod q$$

and outputs

$$y = \lfloor \bar{y} \rceil_p \mod p$$

Theorem 4.2. *If* FHE *is a secure FHE scheme with additional properties defined in Lemma A.4,* H *is a secure explainable hash function, and* F *is a secure PRF, then* PRF *is a secure private puncturable PRF as defined in Sec. 4.1 assuming the hardness of* $LWE_{n-1,q,\tilde{D}_\sigma}$.

We present proof of Theorem 4.2 in Appendix D.

Parameters. Next, we give an instantiation for the parameters of PRF. Security of PRF relies on the hardness of $\text{LWE}_{n-1,q,\tilde{D}_\sigma}$ and $\text{LWE}_{O(k/\lceil \log q \rceil),q,\tilde{D}_\sigma}$, where the latter is required to guarantee the security of FHE. Besides, we require

$$\begin{aligned} q &\geq 2^{L+\omega(\log \lambda)} \cdot p \cdot (2\Sigma + 1) \geq 2^{L+\omega(\log \lambda)} \cdot p \cdot 2^{\omega(\log \lambda)} \cdot \Sigma' \\ &\geq 2^{L+\omega(\log \lambda)} \cdot p \cdot \varkappa \cdot \lambda \cdot \sigma \cdot (k^{O(d)} + k \cdot m^{O(D)}) \\ &\geq 2^{L+\omega(\log \lambda)} \cdot 2^{poly(\log \lambda, \log \log q)} \geq 2^{L+poly(\log \lambda, \log \log q)} \end{aligned}$$

Let $\varepsilon \in (0, 1)$ be a constant real value. We set $k = O(n \cdot \lceil \log q \rceil)$, $n = (L + \lambda)^{\frac{1}{\varepsilon}}$ and $q = 2^{O(L+\lambda)}$. This makes the approximation factor $\gamma = O(nq/\sigma)$ of the underlying worst-case lattices problems to be $2^{O(n^\varepsilon)}$.

Now, assume that the input length and the output length are in $O(\lambda)$, then the size of the PRF key will be

$$|K| = (N + 1) \cdot nm\lceil \log q \rceil + k \cdot nm\lceil \log q \rceil + n\lceil \log q \rceil + |k_{\mathsf{H}}| + |k_{\mathsf{F}}| = O(\lambda^{10+\frac{8}{\varepsilon}})$$

In addition, assume that the size of the puncture set is constant, then the size of the constrained key will be

$$\begin{aligned} |CK| = (N + 1) \cdot nm\lceil \log q \rceil + k \cdot nm\lceil \log q \rceil + \lceil \log q \rceil + |k_{\mathsf{H}}| + |\mathcal{P}| \cdot (L + \kappa) \cdot c \\ + (|\mathcal{P}| + 1) \cdot ((N + 1) \cdot m\lceil \log q \rceil + k \cdot m\lceil \log q \rceil) = O(\lambda^{10+\frac{8}{\varepsilon}}) \end{aligned}$$

Given the huge key sizes and the large approximation factor of the underlying lattice problems, our construction is far from practical. It is an interesting and challenging open problem to reduce the parameters and construct a practical private puncturable PRF with the desired security properties.

Acknowledgement. We appreciate the anonymous reviewers for their valuable comments.

References

[AF16] Hamza Abusalah and Georg Fuchsbauer. Constrained PRFs for unbounded inputs with short keys. In *ACNS*, pages 445–463. Springer, 2016.

[AFP16] Hamza Abusalah, Georg Fuchsbauer, and Krzysztof Pietrzak. Constrained PRFs for unbounded inputs. In *CT-RSA*, pages 413–428. Springer, 2016.

[AMN+18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In *CRYPTO*, pages 543–574. Springer, 2018.

[AMN+19] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively single-key secure constrained PRFs for NC^1. In *PKC*, pages 223–253. Springer, 2019.

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459. Springer, 2004.

[BFP+15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In *TCC*, pages 31–60. Springer, 2015.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556. Springer, 2014.

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

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT*, pages 337–367. Springer, 2015.

[Bit17] Nir Bitansky. Verifiable random functions from non-interactive witness-indistinguishable proofs. In *TCC*, pages 567–594. Springer, 2017.

[BKM17] Dan Boneh, Sam Kim, and Hart Montgomery. Private puncturable PRFs from standard lattice assumptions. In *EUROCRYPT*, pages 415–445. Springer, 2017.

[BKW17] Dan Boneh, Sam Kim, and David J Wu. Constrained keys for invertible pseudorandom functions. In *TCC*, pages 237–263. Springer, 2017.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.

[BLW17] Dan Boneh, Kevin Lewi, and David J Wu. Constraining pseudorandom functions privately. In *PKC*, pages 494–524. Springer, 2017.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In *TCC*, pages 264–302. Springer, 2017.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions. In *TCC*, pages 1–30. Springer, 2015.

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

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, pages 480–499. Springer, 2014.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In *EUROCRYPT*, pages 446–476. Springer, 2017.

[CDNO97] Rein Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *CRYPTO*, pages 90–104. Springer, 1997.

[CHN$^+$16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *STOC*, pages 1115–1127, 2016.

[CRV14] Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Constrained pseudorandom functions: Verifiable and delegatable. Cryptology ePrint Archive, Report 2014/522, 2014. https://ia.cr/2014/522.

[CRV16] Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Reducing depth in constrained PRFs: From bit-fixing to NC^1. In *PKC*, pages 359–385. Springer, 2016.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In *CRYPTO*, pages 577–607. Springer, 2018.

[DDM17] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Constrained pseudorandom functions for unconstrained inputs revisited: achieving verifiability and key delegation. In *PKC*, pages 463–493. Springer, 2017.

[DKN$^+$20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In *CRYPTO*, pages 559–589. Springer, 2020.

[DKW16] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. In *EUROCRYPT*, pages 124–153. Springer, 2016.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, pages 523–540. Springer, 2004.

[FHPS13] Eduarda SV Freire, Dennis Hofheinz, Kenneth G Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In *CRYPTO*, pages 513–530. Springer, 2013.

[FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained PRFs. In *ASIACRYPT*, pages 82–101. Springer, 2014.

[Fuc14] Georg Fuchsbauer. Constrained verifiable random functions. In *SCN*, pages 95–114. Springer, 2014.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *FOCS*, pages 464–479. IEEE, 1984.

[GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In *TCC*, pages 537–566. Springer, 2017.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, pages 640–658. Springer, 2014.

[Gol08] Oded Goldreich. Computational complexity: a conceptual perspective. *ACM Sigact News*, 39(3):35–39, 2008.

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

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, pages 503–523. Springer, 2015.

[HKKW19] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. In *FC*, pages 357–376. Springer, 2019.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In *ASIACRYPT*, pages 79–102. Springer, 2015.

[ILL89] Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *STOC*, pages 12–24, 1989.

[Jag15] Tibor Jager. Verifiable random functions from weaker assumptions. In *TCC*, pages 121–143. Springer, 2015.

[JKK+17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. In *CRYPTO*, pages 133–163. Springer, 2017.

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

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

[KW19] Sam Kim and David J. Wu. Watermarking PRFs from lattices: Stronger security via extractable PRFs. In *CRYPTO*, pages 335–366. Springer, 2019.

[KY16] Shuichi Katsumata and Shota Yamada. Partitioning via non-linear polynomial functions: more compact IBEs from ideal lattices and bilinear maps. In *ASIACRYPT*, pages 682–712. Springer, 2016.

[LST18] Benoît Libert, Damien Stehlé, and Radu Titiu. Adaptively secure distributed PRFs from LWE. In *TCC*, pages 391–421. Springer, 2018.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *CRYPTO*, pages 597–612. Springer, 2002.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, pages 738–755. Springer, 2012.

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

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

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In *PKC*, pages 675–701. Springer, 2018.

[PS20] Chris Peikert and Sina Shiehian. Constraining and watermarking PRFs from milder assumptions. In *PKC*, pages 431–461. Springer, 2020.

[PTW20] Naty Peter, Rotem Tsabary, and Hoeteck Wee. One-one constrained pseudorandom functions. In *ITC*, 2020.

[Reg04] Oded Regev. Lattices in computer science-average case hardness. *Lecture Notes for Class (scribe: Elad Verbin)*, 2004.
[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93. ACM, 2005.
[SS96] Michael Sipser and Daniel A Spielman. Expander codes. *IEEE transactions on Information Theory*, 42(6):1710–1722, 1996.
[SWP00] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *S&P*, pages 44–55. IEEE, 2000.
[Yam17] Shota Yamada. Asymptotically compact adaptively secure lattice IBEs and verifiable random functions via generalized partitioning techniques. In *CRYPTO*, pages 161–193. Springer, 2017.
[YAYX20] Rupeng Yang, Man Ho Au, Zuoxia Yu, and Qiuliang Xu. Collusion resistant watermarkable PRFs from standard assumptions. In *CRYPTO*, pages 590–620. Springer, 2020.
[Zém01] Gillés Zémor. On expander codes. *IEEE Transactions on Information Theory*, 47(2):835–837, 2001.

A Additional Preliminaries

A.1 Pseudorandom Function

A PRF [GGM84] with input space $\mathcal{X}$ and output space $\mathcal{Y}$ consists of two PPT algorithms:

- `KeyGen`. On input the security parameter 1^λ, the key generation algorithm outputs a PRF key k.
- `Eval`. On input the PRF key k and an input $x \in \mathcal{X}$, the evaluation algorithm outputs an output $y \in \mathcal{Y}$.

Also, it satisfies the following condition:

- **Pseudorandomness.** Let $k \leftarrow \texttt{KeyGen}(1^\lambda)$, and f be a random function from $\mathcal{X}$ to $\mathcal{Y}$. Also, let $\mathcal{O}_0(\cdot)$ be an oracle that takes as input a string $x \in \mathcal{X}$ and returns $\texttt{Eval}(k, x)$, and let $\mathcal{O}_1(\cdot)$ be an oracle that takes as input a string $x \in \mathcal{X}$ and returns $f(x)$. Then for all PPT adversary $\mathcal{A}$, we have:

$$| \Pr[\mathcal{A}^{\mathcal{O}_0(\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1(\cdot)}(1^\lambda) = 1] | \leq negl(\lambda)$$

A.2 Matrix Embeddings

The matrix embedding technique [BGG+14,GVW15,BV15,BKM17] embeds bits $x_1, \ldots, x_N$ into matrices $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_N \in \mathbb{Z}_q^{n \times n \cdot \lceil \log q \rceil}$ and then computes circuits on these matrices.

Let $m = n \cdot \lceil \log q \rceil$, $\boldsymbol{G} = \boldsymbol{G}_{n,q}$ and write $\boldsymbol{G}_{n,q}^{-1}$ as $\boldsymbol{G}^{-1}$. Also, let $s \in \mathbb{Z}_q^n$, then each bit x_i is encoded into the matrix $\boldsymbol{A}_i$ as an LWE sample

$$\boldsymbol{a}_i^\intercal = \boldsymbol{s}^\intercal(\boldsymbol{A}_i + x_i \cdot \boldsymbol{G}) + \boldsymbol{e}_i^\intercal$$

where $\boldsymbol{e}_i$ is the error term as required in the LWE assumption. We also need an auxiliary encoding of 1 defined as

$$\boldsymbol{a}_0^\intercal = \boldsymbol{s}^\intercal(\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_0^\intercal$$

Given two encodings $\boldsymbol{a}_i, \boldsymbol{a}_j$ of x_i, x_j, we can add them as

$$\begin{aligned}\boldsymbol{a}_+^\intercal =& \boldsymbol{a}_i^\intercal + \boldsymbol{a}_j^\intercal \\ =& \boldsymbol{s}^\intercal(\boldsymbol{A}_i + x_i \cdot \boldsymbol{G}) + \boldsymbol{e}_i^\intercal + \boldsymbol{s}^\intercal(\boldsymbol{A}_j + x_j \cdot \boldsymbol{G}) + \boldsymbol{e}_j^\intercal \\ =& \boldsymbol{s}^\intercal(\boldsymbol{A}_i + \boldsymbol{A}_j + (x_i + x_j) \cdot \boldsymbol{G}) + (\boldsymbol{e}_i^\intercal + \boldsymbol{e}_j^\intercal)\end{aligned}$$

and multiply them as

$$\begin{aligned}\boldsymbol{a}_\times^\intercal =& x_j \cdot \boldsymbol{a}_i^\intercal - \boldsymbol{a}_j^\intercal \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) \\ =& x_j \cdot (\boldsymbol{s}^\intercal(\boldsymbol{A}_i + x_i \cdot \boldsymbol{G}) + \boldsymbol{e}_i^\intercal) - (\boldsymbol{s}^\intercal(\boldsymbol{A}_j + x_j \cdot \boldsymbol{G}) + \boldsymbol{e}_j^\intercal) \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) \\ =& \boldsymbol{s}^\intercal(x_j \cdot \boldsymbol{A}_i + x_i x_j \cdot \boldsymbol{G}) + x_j \cdot \boldsymbol{e}_i^\intercal - \boldsymbol{s}^\intercal(\boldsymbol{A}_j \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) + x_j \cdot \boldsymbol{A}_i) - \boldsymbol{e}_j^\intercal \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) \\ =& \boldsymbol{s}^\intercal(-\boldsymbol{A}_j \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) + x_i x_j \cdot \boldsymbol{G}) + (x_j \cdot \boldsymbol{e}_i^\intercal - \boldsymbol{e}_j^\intercal \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i))\end{aligned} \tag{5}$$

Then we can compute the encoding for the NAND of x_i and x_j as

$$\begin{aligned}\boldsymbol{a}_{\bar\wedge}^\intercal =& \boldsymbol{a}_0^\intercal - \boldsymbol{a}_\times^\intercal \\ =& \boldsymbol{s}^\intercal(\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_0^\intercal - \boldsymbol{s}^\intercal(-\boldsymbol{A}_j \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) + x_i x_j \cdot \boldsymbol{G}) - (x_j \cdot \boldsymbol{e}_i^\intercal - \boldsymbol{e}_j^\intercal \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i)) \\ =& \boldsymbol{s}^\intercal(\boldsymbol{A}_0 + \boldsymbol{A}_j \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) + (1 - x_i x_j) \cdot \boldsymbol{G}) + (\boldsymbol{e}_0^\intercal - x_j \cdot \boldsymbol{e}_i^\intercal + \boldsymbol{e}_j^\intercal \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i))\end{aligned} \tag{6}$$

Let

$$\boldsymbol{A}_{\bar\wedge} = \boldsymbol{A}_0 + \boldsymbol{A}_j \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) \tag{7}$$

and $\boldsymbol{e}_{\bar\wedge}^\intercal = \boldsymbol{e}_0^\intercal - x_j \cdot \boldsymbol{e}_i^\intercal + \boldsymbol{e}_j^\intercal \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i)$, then we can write

$$\boldsymbol{a}_{\bar\wedge}^\intercal = \boldsymbol{s}^\intercal(\boldsymbol{A}_{\bar\wedge} + (1 - x_i x_j) \cdot \boldsymbol{G}) + \boldsymbol{e}_{\bar\wedge}^\intercal$$

Now, if we perform Equation (7) on the matrices and perform Equation (5) and Equation (6) on the encodings inductively, going through the gates of a circuit $\mathtt{C}$ (composed exclusively by NAND gates), then we get the encoding for $\mathtt{C}(x)$ as

$$\boldsymbol{a}_{\mathtt{C}}^\intercal = \boldsymbol{s}^\intercal(\boldsymbol{A}_{\mathtt{C}} + \mathtt{C}(x) \cdot \boldsymbol{G}) + \boldsymbol{e}_{\mathtt{C}}^\intercal$$

Formally, we have the following Lemma.

Lemma A.1 ([BGG+14]). *Let n, m, q, B, d, N be positive integers that $m = n \cdot \lceil \log q \rceil$. Let $\mathtt{C} : \{0,1\}^N \to \{0,1\}$ be a depth-d Boolean circuit. Also, let $\boldsymbol{s} \in \mathbb{Z}_q^n$, $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_N \in \mathbb{Z}_q^{n \times m}$, $x_1, \ldots, x_N \in \{0,1\}$, and $\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_N \in \mathbb{Z}_q^m$, where*

$$\|\boldsymbol{a}_0^\intercal - \boldsymbol{s}^\intercal(\boldsymbol{A}_0 + \boldsymbol{G})\|_\infty \le B \quad \wedge \quad \forall i \in [1, N], \|\boldsymbol{a}_i^\intercal - \boldsymbol{s}^\intercal(\boldsymbol{A}_i + x_i \cdot \boldsymbol{G})\|_\infty \le B$$

There exists the following two deterministic algorithms:

- $\mathtt{EvalPK}(\mathtt{C}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N) \to \boldsymbol{A}_{\mathtt{C}} \in \mathbb{Z}_q^{n \times m}$.

- $\texttt{EvalCT}(\mathtt{C}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_N, x_1, \ldots, x_N) \to \boldsymbol{a}_{\mathtt{C}} \in \mathbb{Z}_q^m$.

such that for $\boldsymbol{A}_{\mathtt{C}} = \texttt{EvalPK}(\mathtt{C}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N)$ *and* $\boldsymbol{a}_{\mathtt{C}} = \texttt{EvalCT}(\mathtt{C}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_N, x_1, \ldots, x_N)$*, we have*

$$\|\boldsymbol{a}_{\mathtt{C}}^{\mathsf{T}} - \boldsymbol{s}^{\mathsf{T}}(\boldsymbol{A}_{\mathtt{C}} + \mathtt{C}(x) \cdot \boldsymbol{G})\|_\infty \le (m+2)^d \cdot B$$

Note that in the description above, to multiply the two encodings $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$, we do not need to know x_i. In addition, the error term of the new encoding is independent of x_i. Thus, we can compute encoding of the inner product of a binary vector $\boldsymbol{x}$ and a vector $\boldsymbol{y}$ in $\mathbb{Z}_q$, without knowing $\boldsymbol{y}$. Formally, we have:

Lemma A.2 ([GVW15]). *Let* n, m, q, B, k *be positive integers that* $m = n \cdot \lceil \log q \rceil$*. Also, let* $\boldsymbol{s} \in \mathbb{Z}_q^n$*,* $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_k, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k \in \mathbb{Z}_q^{n\times m}$*,* $x_1, \ldots, x_k \in \{0,1\}$*,* $y_1, \ldots, y_k \in \mathbb{Z}_q$*, and* $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_k \in \mathbb{Z}_q^m$*, where*

$$\forall i \in [1,k],\ \|\boldsymbol{a}_i^{\mathsf{T}} - \boldsymbol{s}^{\mathsf{T}}(\boldsymbol{A}_i + x_i \cdot \boldsymbol{G})\|_\infty \le B \ \wedge\ \|\boldsymbol{b}_i^{\mathsf{T}} - \boldsymbol{s}^{\mathsf{T}}(\boldsymbol{B}_i + y_i \cdot \boldsymbol{G})\|_\infty \le B$$

There exists the following two deterministic algorithms:

- $\texttt{IPEvalPK}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_k, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k) \to \boldsymbol{C}_{\mathtt{IP}} \in \mathbb{Z}_q^{n\times m}$.
- $\texttt{IPEvalCT}(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_k, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_k, x_1, \ldots, x_k) \to \boldsymbol{c}_{\mathtt{IP}} \in \mathbb{Z}_q^m$.

such that for $\boldsymbol{C}_{\mathtt{IP}} = \texttt{IPEvalPK}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_k, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k)$ *and* $\boldsymbol{c}_{\mathtt{IP}} = \texttt{IPEvalCT}(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_k, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_k, x_1, \ldots, x_k)$*, we have*

$$\|\boldsymbol{c}_{\mathtt{IP}}^{\mathsf{T}} - \boldsymbol{s}^{\mathsf{T}}(\boldsymbol{C}_{\mathtt{IP}} + \sum_{i=1}^{k} x_i \cdot y_i \cdot \boldsymbol{G})\|_\infty \le k \cdot (m+1) \cdot B$$

Now, consider the case that each $\boldsymbol{A}_i$ is also embedded with a bit, that is

$$\boldsymbol{A}_0 = \boldsymbol{B}_0 - \boldsymbol{G} \ \text{ and } \ \forall i \in [1,N], \boldsymbol{A}_i = \boldsymbol{B}_i - x_i \cdot \boldsymbol{G}$$

Then if we compute Equation (7), we get

$$\begin{aligned}
\boldsymbol{A}_{\bar{\wedge}} =& \boldsymbol{A}_0 + \boldsymbol{A}_j \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) \\
=& \boldsymbol{B}_0 - \boldsymbol{G} + (\boldsymbol{B}_j - x_j \cdot \boldsymbol{G}) \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) \\
=& \boldsymbol{B}_0 - \boldsymbol{G} + \boldsymbol{B}_j \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) - x_j \cdot (\boldsymbol{B}_i - x_i \cdot \boldsymbol{G}) \\
=& (\boldsymbol{B}_0 + \boldsymbol{B}_j \cdot \boldsymbol{G}^{-1}(\boldsymbol{A}_i) - x_j \cdot \boldsymbol{B}_i) - (1 - x_i x_j) \cdot \boldsymbol{G} \\
=& (\boldsymbol{B}_0, \boldsymbol{B}_j, \boldsymbol{B}_i) \cdot \boldsymbol{R} - (1 - x_i x_j) \cdot \boldsymbol{G}
\end{aligned}$$

where

$$\boldsymbol{R} = \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{G}^{-1}(\boldsymbol{A}_i) \\ -x_j \cdot \boldsymbol{I} \end{pmatrix} \in \{-1,0,1\}^{3m\times m}$$

has at most $(m+2)$ non-zero items in each column.

Lemma A.3 ([BGG+14]). *Let n, m, q, d, N be positive integers that $m = n \cdot \lceil \log q \rceil$. Let $\mathtt{C} : \{0,1\}^N \to \{0,1\}$ be a depth-d Boolean circuit. Also, let $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_N, \boldsymbol{B}_0, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_N \in \mathbb{Z}_q^{n\times m}$, and $x_1, \ldots, x_N \in \{0,1\}$, where*

$$\boldsymbol{A}_0 = \boldsymbol{B}_0 - \boldsymbol{G} \quad \wedge \quad \forall i \in [1, N], \boldsymbol{A}_i = \boldsymbol{B}_i - x_i \cdot \boldsymbol{G}$$

There exists the following deterministic algorithm:

- $\mathtt{EvalPK}(\mathtt{C}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N) \to \boldsymbol{A}_{\mathtt{C}} \in \mathbb{Z}_q^{n\times m}$.

such that for $\boldsymbol{A}_{\mathtt{C}} = \mathtt{EvalPK}(\mathtt{C}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N)$, there exists $\boldsymbol{T} \in [-(m+2)^d, (m+2)^d]^{m\cdot(N+1)\times m}$ satisfying

$$\boldsymbol{A}_{\mathtt{C}} = (\boldsymbol{B}_0, \ldots, \boldsymbol{B}_N) \cdot \boldsymbol{T} - \mathtt{C}(x) \cdot \boldsymbol{G}$$

A.3 Fully Homomorphic Encryption

We use the (leveled) fully homomorphic encryption [Gen09] scheme proposed in [GSW13]. A leveled FHE scheme consists of four PPT algorithms:

- $\mathtt{KeyGen}(1^\lambda, 1^d) \to (pk, sk)$. On input the security parameter 1^λ and a depth bound d, the key generation algorithm outputs a public key pk and a secret key sk.
- $\mathtt{Enc}(pk, \mu) \to ct$. On input the public key pk and a message $\mu \in \{0,1\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathtt{Dec}(sk, ct) \to \mu$. On input the secret key sk and a ciphertext ct, the decryption algorithm outputs a message μ.
- $\mathtt{Eval}(\mathtt{C}, \boldsymbol{ct}) \to ct'$. For any polynomial N, on input a circuit $\mathtt{C} : \{0,1\}^N \to \{0, 1\}$ of depth at most d and a vector of N ciphertexts $\boldsymbol{ct}$, the evaluation algorithm outputs a ciphertext ct'.

Also, it satisfies the following properties:

- **Perfect Correctness.** For any N, d, any messages $\mu_1, \ldots, \mu_N \in \{0,1\}$, any circuit $\mathtt{C} : \{0,1\}^N \to \{0,1\}$ of depth at most d, and any $(pk, sk) \leftarrow \mathtt{KeyGen}(1^\lambda, 1^d)$, we have:

$$\Pr[\mathtt{Dec}(sk, \mathtt{Eval}(\mathtt{C}, (\mathtt{Enc}(pk, \mu_i))_{i\in[1,N]})) = \mathtt{C}(\mu_1, \ldots, \mu_N)] = 1$$

- **Security.** For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and for any $d = poly(\lambda)$, we have:

$$\Pr\left[\begin{array}{l} (pk, sk) \leftarrow \mathtt{KeyGen}(1^\lambda, 1^d); \\ (m_0^*, m_1^*, state) \leftarrow \mathcal{A}_1(pk); \\ b \xleftarrow{\$} \{0,1\}; \\ ct^* \leftarrow \mathtt{Enc}(pk, m_b^*) \end{array} : \mathcal{A}_2(ct^*, state) = b \right] \leq 1/2 + negl(\lambda)$$

The GSW FHE Scheme. Our construction relies on some specific properties of the GSW FHE scheme:

Lemma A.4 ([GSW13]). *Let λ be the security parameter and $d = poly(\lambda)$. Let n, k, q, c, σ be positive integers such that n, σ are polynomial in λ, $k = O(n \cdot \lceil \log q \rceil)$, $c = O(n^2 \log^2 q)$, and $q > \lambda \cdot \sigma \cdot k^{O(d)}$. Then there exists a secure FHE scheme $\mathsf{FHE} = (\mathsf{FHE.}\mathtt{KeyGen}, \mathsf{FHE.}\mathtt{Enc}, \mathsf{FHE.}\mathtt{Dec}, \mathsf{FHE.}\mathtt{Eval})$ for circuits with depth at most d with the following properties, assuming the $LWE_{n,q,\tilde{D}_\sigma}$ assumption:*

- *The secret key of FHE is in $\mathbb{Z}_q^k$.*
- *The encryption algorithm takes in a message in $\{0,1\}$ and outputs a ciphertext in $\{0,1\}^c$.*
- *The evaluation algorithm can additionally take as input an integer $\ell \in [1, \lceil \log q \rceil]$ and output a ciphertext in $\{0,1\}^k$.*
- *Given any boolean circuit of depth at most d, the evaluation algorithm can be evaluated by a Boolean circuit of depth at most $D = d \cdot poly(\log \lambda, \log \log q)$.*
- *For any polynomial N, d, any $\ell \in [1, \lceil \log q \rceil]$, any messages $\mu_1, \ldots, \mu_N \in \{0, 1\}$, and any boolean circuit $\mathtt{C} : \{0,1\}^N \to \{0,1\}$ of depth at most d, let $(pk, sk) \leftarrow \mathsf{FHE.}\mathtt{KeyGen}(1^\lambda, 1^d)$ and for $i \in [1, N]$, let $ct_i \leftarrow \mathsf{FHE.}\mathtt{Enc}(pk, \mu_i)$. Also let*
$$ct' \leftarrow \mathtt{Eval}(\ell, \mathtt{C}, (ct_1, \ldots, ct_N))$$
$$\nu = \sum_{i=1}^{k} sk[i] \cdot ct'[i] \mod q$$
then we have
$$|\nu - \mathtt{C}(\mu_1, \ldots, \mu_N) \cdot 2^{\ell-1}| \leq \lambda \cdot \sigma \cdot k^{O(d)}$$

A.4 Admissible Hash

We use the (balanced) admissible hash function [BB04, Jag15] in our construction. The following definition is adapted from the definition given in [Jag15], where we modify the definition of τ in Equation (9) below.

Definition A.1. *Let λ be the security parameter. Let l, t be positive integers that are polynomial in λ. Let*
$$\mathsf{H}_{adm} : \{0,1\}^l \to \{0,1\}^t$$
be an efficiently computable function. For $K \in \{0, 1, \perp\}^t$, let $\mathsf{P}_K : \{0,1\}^t \to \{0, 1\}$ be defined as
$$\mathsf{P}_K(w) = \begin{cases} 1 & \text{if } \forall i \in [1,t], K[i] = \perp \ \vee \ K[i] = w[i] \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

We say that H_{adm} is a balanced admissible hash function *if for any polynomial Q and non-negligible real value $\delta \in (0,1]$, there exits a PPT algorithm*

- $\texttt{AdmSmp}_{Q,\delta}(1^\lambda) \rightarrow K$. *On input the security parameter* 1^λ*, the algorithm outputs* $K \in \{0, 1, \perp\}^t$.

and non-negligible real values γ_{min} *and* γ_{max} *such that for all* $x_1, \ldots, x_Q, x^* \in \{0, 1\}^l$ *with* $x^* \notin \{x_1, \ldots, x_Q\}$[18]*, we have*

$$\gamma_{min} \leq \Pr[\mathsf{P}_K(\mathsf{H}_{adm}(x^*)) = 1 \ \wedge \ \forall i \in [1, Q], \mathsf{P}_K(\mathsf{H}_{adm}(x_i)) = 0] \leq \gamma_{max}$$

and

$$\tau = \gamma_{min} \cdot \delta - (\gamma_{max} - \gamma_{min}) \tag{9}$$

is a non-negligible positive real value, where the probability is taken over the choice of $K \leftarrow \texttt{AdmSmp}_{Q,\delta}(1^\lambda)$.

The (balanced) admissible hash function presented in [Lys02,FHPS13,Jag15], which are constructed from an error correcting code with suitable minimal distance (see e.g., [SS96, Zém01, Gol08] for explicit constructions of such codes), also satisfy Definition A.1. We formally state this in Lemma A.5 and for completeness, we give its proof in Appendix A.4.1.

Lemma A.5. *Let* c *be a constant and* $t = O(l)$*. Let* $\mathsf{C} : \{0, 1\}^l \rightarrow \{0, 1\}^t$ *be a family of code with minimal distance* $c \cdot t$ *(i.e., for any distinct* $x_1, x_2 \in \{0, 1\}^l$*,* $\mathsf{C}(x_1)$ *and* $\mathsf{C}(x_2)$ *differ in at least* $c \cdot t$ *positions). Then* C *is a balanced admissible hash function defined in Definition A.1.*

We need the following lemma from [KY16, DKN+20] when using admissible hash functions in the security proof.

Lemma A.6 ([KY16, DKN+20]). *Let* $\mathcal{D}$ *be a PPT algorithm that takes as input a bit* $b \in \{0, 1\}$ *and outputs* $(x, b') \in \mathcal{X} \times \{0, 1\}$*, where* $\mathcal{X}$ *is some domain. Define*

$$\delta = |\Pr[b \xleftarrow{\$} \{0, 1\}, (x, b') \leftarrow \mathcal{D}(b) : b = b'] - \frac{1}{2}|$$

Let γ *be a map that maps an element in* $\mathcal{X}$ *to a real value in* $[0, 1]$ *and* $\gamma_{min}, \gamma_{max}$ *be real values such that* $\forall x \in \mathcal{X}$*,* $\gamma_{min} \leq \gamma(x) \leq \gamma_{max}$.

Let $\mathcal{D}'$ *be a PPT algorithm that takes as input a bit* $b \in \{0, 1\}$ *and outputs* $(x, b') \in \mathcal{X} \times \{0, 1\}$*. In particular, given an input* b*,* $\mathcal{D}'$ *first runs* $(x, b_1) \leftarrow \mathcal{D}(b)$ *and samples* $b_2 \xleftarrow{\$} \{0, 1\}$*, then it outputs* (x, b_1) *with probability* $\gamma(x)$ *and outputs* (x, b_2) *with probability* $1 - \gamma(x)$*. Then, the following holds:*

$$|\Pr[b \xleftarrow{\$} \{0, 1\}, (x, b') \leftarrow \mathcal{D}'(b) : b = b'] - \frac{1}{2}| \geq \gamma_{min} \cdot \delta - \frac{\gamma_{max} - \gamma_{min}}{2}$$

Embedding P_K into Matrices. In our constructions, we need to embed the evaluation of P_K into matrices. The task can be completed by applying Lemma A.3 directly, where each element of $\boldsymbol{T}$ is roughly bounded by $m^{O(\log t)}$. In [Yam17], Yamada improves the parameter to be roughly $O(m^2 \cdot t)$ by refining the process of evaluation. We give a variant of this construction, which is simpler and also improves the parameter. We present the formal description of the result in Lemma A.7 and provide the proof in Appendix A.4.2.

[18] We allow $x_i = x_j$ for some distinct $i, j \in [1, Q]$.

Lemma A.7. *Let t be a polynomial in λ. Let $K \in \{0,1,\perp\}^t$ and let $\mathsf{P}_K : \{0,1\}^t \to \{0,1\}$ be the function defined in Equation (8). For $i \in [1,t]$, let*

$$(K_{i,0}, K_{i,1}) = \begin{cases} (0,0) & \text{if } K[i] = \perp \\ (1, K[i]) & \text{otherwise} \end{cases}$$

Let n, m, q be positive integers that $m = n \cdot \lceil \log q \rceil$. Also, let $\boldsymbol{A}_{1,0}, \boldsymbol{A}_{1,1}, \ldots, \boldsymbol{A}_{t,0}, \boldsymbol{A}_{t,1}, \boldsymbol{B}_{1,0}, \boldsymbol{B}_{1,1}, \ldots, \boldsymbol{B}_{t,0}, \boldsymbol{B}_{t,1} \in \mathbb{Z}_q^{n\times m}$, where

$$\forall i \in [1,t],\ \ \boldsymbol{A}_{i,0} = \boldsymbol{B}_{i,0} - K_{i,0} \cdot \boldsymbol{G},\ \ \boldsymbol{A}_{i,1} = \boldsymbol{B}_{i,1} - K_{i,1} \cdot \boldsymbol{G}$$

There exists the following deterministic algorithm:

- $\mathtt{EvalAdm}(\boldsymbol{A}_{1,0}, \boldsymbol{A}_{1,1}, \ldots, \boldsymbol{A}_{t,0}, \boldsymbol{A}_{t,1}, w) \to \boldsymbol{A}_{\mathsf{P}} \in \mathbb{Z}_q^{n\times m}$.

such that for any $w \in \{0,1\}^t$ and for $\boldsymbol{A}_{\mathsf{P}} = \mathtt{EvalAdm}(\boldsymbol{A}_{1,0}, \boldsymbol{A}_{1,1}, \ldots, \boldsymbol{A}_{t,0}, \boldsymbol{A}_{t,1}, w)$, there exists $\boldsymbol{T} \in [-m,m]^{2tm\times m}$ satisfying

$$\boldsymbol{A}_{\mathsf{P}} = (\boldsymbol{B}_{1,0}, \boldsymbol{B}_{1,1}, \ldots, \boldsymbol{B}_{t,0}, \boldsymbol{B}_{t,1}) \cdot \boldsymbol{T} - \mathsf{P}_K(w) \cdot \boldsymbol{G}$$

A.4.1 Proof of Lemma A.5

Following [Jag15], we set the output space of the algorithm $\mathtt{AdmSmp}_{Q,\delta}$ as the set of all strings in $\{0,1\}^t$ with exactly

$$d = \lceil \frac{\log(2Q + 2Q/\delta)}{-\log(1-c)} \rceil$$

components not equal to $\perp$, and the algorithm outputs a uniform element from this set.[19] As shown in [Jag15], we can set[20]

$$\gamma_{max} = 2^{-d} \quad \text{and} \quad \gamma_{min} = (1 - Q(1-c)^d) \cdot 2^{-d}$$

[19] We change the value of d here.
[20] The computation of γ_{max} and γ_{min} does not use the concrete value of d.

Thus, we have

$$
\begin{aligned}
\tau &= \gamma_{min} \cdot \delta - (\gamma_{max} - \gamma_{min}) \\
&= (1 - Q(1-c)^d) \cdot 2^{-d} \cdot \delta - (2^{-d} - (1 - Q(1-c)^d) \cdot 2^{-d}) \\
&= 2^{-d} \cdot (\delta - \delta \cdot Q(1-c)^d - 1 + 1 - Q(1-c)^d) \\
&= 2^{-d} \cdot (\delta - (\delta + 1) \cdot Q(1-c)^{\lceil \frac{\log(2Q+2Q/\delta)}{-\log(1-c)} \rceil}) \\
&\geq 2^{-d} \cdot (\delta - (\delta + 1) \cdot Q(1-c)^{\frac{\log(2Q+2Q/\delta)}{-\log(1-c)}}) \\
&= 2^{-d} \cdot (\delta - (\delta + 1) \cdot \frac{Q}{2Q + 2Q/\delta}) \\
&= 2^{-d} \cdot (\delta - (\delta + 1) \cdot \frac{\delta}{2\delta + 2}) \\
&= 2^{-\lceil \frac{\log(2Q+2Q/\delta)}{-\log(1-c)} \rceil} \cdot \frac{\delta}{2} \\
&\geq 2^{-(\frac{\log(2Q+2Q/\delta)}{-\log(1-c)} + 1)} \cdot \frac{\delta}{2} \\
&= \frac{1}{(2Q + 2Q/\delta)^{c'}} \cdot \frac{\delta}{4}
\end{aligned}
$$

where $c' = \frac{-1}{\log(1-c)}$ is a positive constant. Since Q is a polynomial and δ is non-negligible, τ is also non-negligible. That completes the proof of Lemma A.5.

A.4.2 Proof of Lemma A.7

We can compute $\mathsf{P}_K(w)$ as follows:

1. For $i \in [1, t]$:
   (a) $L_i = 1 - w[i] \cdot K_{i,1} - (1 - w[i])(1 - K_{i,1})$.
   (b) $M_i = 1 - K_{i,0} \cdot L_i$.
2. Output $\prod_{i=1}^{t} M_i$.

It is easy to see $L_i, M_i \in \{0, 1\}$ and the output is equal to $\mathsf{P}_K(w)$.

Given inputs $\boldsymbol{A}_{1,0}, \boldsymbol{A}_{1,1}, \ldots, \boldsymbol{A}_{t,0}, \boldsymbol{A}_{t,1}$ and $w \in \{0,1\}^t$, the `EvalAdm` algorithm works as follows:

1. For $i \in [1, t]$:
   (a) $\boldsymbol{D}_i = -\boldsymbol{G} - w[i] \cdot \boldsymbol{A}_{i,1} - (1 - w[i]) \cdot (-\boldsymbol{G} - \boldsymbol{A}_{i,1})$.
   (b) $\boldsymbol{F}_i = -\boldsymbol{G} + \boldsymbol{A}_{i,0} \cdot \boldsymbol{G}^{-1}(\boldsymbol{D}_i)$.
2. $\boldsymbol{H}_1 = \boldsymbol{F}_1$.
3. For $i \in [2, t]$:
   (a) $\boldsymbol{H}_i = -\boldsymbol{F}_i \cdot \boldsymbol{G}^{-1}(\boldsymbol{H}_{i-1})$.
4. Output $\boldsymbol{A}_{\mathsf{P}} = \boldsymbol{H}_t$.

Note that for $i \in [1,t]$,

$$\begin{aligned}
\boldsymbol{D}_i &= -\boldsymbol{G} - w[i] \cdot \boldsymbol{A}_{i,1} - (1 - w[i]) \cdot (-\boldsymbol{G} - \boldsymbol{A}_{i,1}) \\
&= -\boldsymbol{G} - w[i] \cdot (\boldsymbol{B}_{i,1} - K_{i,1} \cdot \boldsymbol{G}) - (1 - w[i]) \cdot (-\boldsymbol{B}_{i,1} - (1 - K_{i,1}) \cdot \boldsymbol{G}) \\
&= (1 - 2w[i]) \cdot \boldsymbol{B}_{i,1} - (1 - w[i] \cdot K_{i,1} - (1 - w[i]) \cdot (1 - K_{i,1})) \cdot \boldsymbol{G} \\
&= (1 - 2w[i]) \cdot \boldsymbol{B}_{i,1} - L_i \cdot \boldsymbol{G}
\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{F}_i &= -\boldsymbol{G} + \boldsymbol{A}_{i,0} \cdot \boldsymbol{G}^{-1}(\boldsymbol{D}_i) \\
&= -\boldsymbol{G} + (\boldsymbol{B}_{i,0} - K_{i,0} \cdot \boldsymbol{G}) \cdot \boldsymbol{G}^{-1}(\boldsymbol{D}_i) \\
&= -\boldsymbol{G} + \boldsymbol{B}_{i,0} \cdot \boldsymbol{G}^{-1}(\boldsymbol{D}_i) - K_{i,0} \cdot ((1 - 2w[i]) \cdot \boldsymbol{B}_{i,1} - L_i \cdot \boldsymbol{G}) \\
&= \boldsymbol{B}_{i,0} \cdot \boldsymbol{G}^{-1}(\boldsymbol{D}_i) - K_{i,0} \cdot (1 - 2w[i]) \cdot \boldsymbol{B}_{i,1} - (1 - K_{i,0} \cdot L_i) \cdot \boldsymbol{G} \\
&= \boldsymbol{E}_i - M_i \cdot \boldsymbol{G}
\end{aligned}$$

where $\boldsymbol{E}_i = (\boldsymbol{B}_{i,0}, \boldsymbol{B}_{i,1}) \cdot \boldsymbol{R}_i$ and

$$\boldsymbol{R}_i = \begin{pmatrix} \boldsymbol{G}^{-1}(\boldsymbol{D}_i) \\ -K_{i,0} \cdot (1 - 2w[i]) \cdot \boldsymbol{I} \end{pmatrix}$$

is a matrix in $\{-1, 0, 1\}^{2m \times m}$.

Also, assume that there exists $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_{i-1} \in \{-1, 0, 1\}^{m \times m}$ s.t.

$$\boldsymbol{H}_{i-1} = \sum_{j=1}^{i-1} \boldsymbol{E}_j \cdot \boldsymbol{S}_j - \prod_{j=1}^{i-1} M_j \cdot \boldsymbol{G}$$

then we have

$$\begin{aligned}
\boldsymbol{H}_i &= -\boldsymbol{F}_i \cdot \boldsymbol{G}^{-1}(\boldsymbol{H}_{i-1}) \\
&= -(\boldsymbol{E}_i - M_i \cdot \boldsymbol{G}) \cdot \boldsymbol{G}^{-1}(\boldsymbol{H}_{i-1}) \\
&= -\boldsymbol{E}_i \cdot \boldsymbol{G}^{-1}(\boldsymbol{H}_{i-1}) + M_i \cdot (\sum_{j=1}^{i-1} \boldsymbol{E}_j \cdot \boldsymbol{S}_j - \prod_{j=1}^{i-1} M_j \cdot \boldsymbol{G}) \\
&= \sum_{j=1}^{i} \boldsymbol{E}_j \cdot \boldsymbol{S}'_j - \prod_{j=1}^{i} M_j \cdot \boldsymbol{G}
\end{aligned}$$

where $\boldsymbol{S}'_j = M_i \cdot \boldsymbol{S}_j$ for $j \in [1, i-1]$ and $\boldsymbol{S}'_i = -\boldsymbol{G}^{-1}(\boldsymbol{H}_{i-1})$, i.e. $\boldsymbol{S}'_j \in \{-1, 0, 1\}^{m \times m}$ for $j \in [1, i]$. Since $\boldsymbol{H}_1 = \boldsymbol{F}_1 = \boldsymbol{E}_1 - M_1 \cdot \boldsymbol{G}$ also satisfies the assumption, there exists $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_t \in \{-1, 0, 1\}^{m \times m}$ s.t.

$$\boldsymbol{H}_t = \sum_{j=1}^{t} \boldsymbol{E}_j \cdot \boldsymbol{S}_j - \mathsf{P}_K(w) \cdot \boldsymbol{G}$$

Now, let

$$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{R}_1 \cdot \boldsymbol{S}_1 \\ \vdots \\ \boldsymbol{R}_t \cdot \boldsymbol{S}_t \end{pmatrix}$$

then we have

$$\boldsymbol{H}_t = (\boldsymbol{B}_{1,0}, \boldsymbol{B}_{1,1}, \ldots, \boldsymbol{B}_{t,0}, \boldsymbol{B}_{t,1}) \cdot \boldsymbol{T} - \mathsf{P}_K(w) \cdot \boldsymbol{G}$$

In addition, as elements in $\boldsymbol{R}_i$ and $\boldsymbol{S}_i$ are in $\{-1, 0, 1\}$, each element in $\boldsymbol{T}$ is in $[-m, m]$, i.e., $\boldsymbol{T} \in [-m, m]^{2tm \times m}$. This completes proof of Lemma A.7.

B Security Analysis of Explainable Hash

We present proof of Theorem 3.1 in this section. More precisely, we will prove the injectivity and the explainability of H.

Injectivity. For any fixed integer $i^* \in [1, l]$ and for any fixed inputs $x^{(1)}, x^{(2)} \in \{0,1\}^l$ s.t. $x^{(1)}[i^*] \neq x^{(2)}[i^*]$, we first bound the probability that $\mathtt{Eval}(hk, x^{(1)}) = \mathtt{Eval}(hk, x^{(2)})$ for a random hash key hk.

Let

$$y^{(1)}_{i^*,j} = \boldsymbol{s}^{\mathsf{T}}_{i^*,j} \cdot \boldsymbol{b}_{\mathsf{H}_{adm}(x^{(1)})} + v_{i^*,j,x^{(1)}[i^*]} \mod q$$

and $\boldsymbol{u}^{(1)}_{i^*}[j]$ be the variables used in evaluating $\mathtt{Eval}(hk, x^{(1)})$. For a random hk, we have $v_{i^*,j,x^{(1)}[i^*]}$ uniform in $\mathbb{Z}_q$ for any $j \in [1, k]$, and thus $y^{(1)}_{i^*,j}$ is also uniform in $\mathbb{Z}_q$. Therefore, we have

$$\Pr[\boldsymbol{u}^{(1)}_{i^*}[j] = 0] = \frac{1}{2} + \frac{1}{2q}$$

Similarly, let

$$y^{(2)}_{i^*,j} = \boldsymbol{s}^{\mathsf{T}}_{i^*,j} \cdot \boldsymbol{b}_{\mathsf{H}_{adm}(x^{(2)})} + v_{i^*,j,x^{(2)}[i^*]} \mod q$$

and $\boldsymbol{u}^{(2)}_{i^*}[j]$ be the variables used in evaluating $\mathtt{Eval}(hk, x^{(2)})$, then we also have

$$\Pr[\boldsymbol{u}^{(2)}_{i^*}[j] = 0] = \frac{1}{2} + \frac{1}{2q}$$

Note that as $x^{(1)}[i^*] \neq x^{(2)}[i^*]$, the random variables $\boldsymbol{u}^{(1)}_{i^*}[j]$ and $\boldsymbol{u}^{(2)}_{i^*}[j]$ are independent. Therefore, we have

$$\Pr[\boldsymbol{u}^{(1)}_{i^*}[j] = \boldsymbol{u}^{(2)}_{i^*}[j]] = (\frac{1}{2} + \frac{1}{2q})^2 + (\frac{1}{2} - \frac{1}{2q})^2 = \frac{1}{2} + \frac{1}{2q^2} \leq \frac{2}{3}$$

for any $j \in [1, k]$, and this implies that

$$\Pr[\boldsymbol{u}^{(1)}_{i^*} = \boldsymbol{u}^{(2)}_{i^*}] \leq (\frac{2}{3})^k \leq \frac{1}{2^{2l+\lambda/2}}$$

i.e.,

$$\Pr[\mathtt{Eval}(hk, x^{(1)}) = \mathtt{Eval}(hk, x^{(2)})] \leq \frac{1}{2^{2l+\lambda/2}}$$

Next, by the union bound, for any fixed $i^* \in [1, l]$, the probability that there exists $x^{(1)}, x^{(2)} \in \{0,1\}^l$ s.t. $x^{(1)}[i^*] \neq x^{(2)}[i^*]$ and $\texttt{Eval}(hk, x^{(1)}) = \texttt{Eval}(hk, x^{(2)})$ will not exceed

$$\frac{1}{2^{2l+\lambda/2}} \cdot 2^{2l} = \frac{1}{2^{\lambda/2}}$$

Finally, using the union bound again, the probability that there exists $x^{(1)}, x^{(2)} \in \{0,1\}^l$ s.t. $x^{(1)} \neq x^{(2)}$ and $\texttt{Eval}(hk, x^{(1)}) = \texttt{Eval}(hk, x^{(2)})$ will not exceed $\frac{l}{2^{\lambda/2}}$, which is negligible.

This completes the proof of injectivity.

Explainability. For any polynomial Q and non-negligible δ, let $\texttt{AdmSmp}_{Q,\delta}$ and P be the algorithm and function defined in Definition A.1, then the algorithms $(\texttt{VKeyGen}_{Q,\delta}, \texttt{Verify}_{Q,\delta})$ work as follows:

- $\texttt{VKeyGen}_{Q,\delta}$. On input the security parameter 1^λ, the verification key generation algorithm computes $K \leftarrow \texttt{AdmSmp}_{Q,\delta}(1^\lambda)$ and sets $vk = K$.
- $\texttt{Verify}_{Q,\delta}$. On input the verification key $vk = K$, a set $\mathcal{Q} \subset \{0,1\}^l$ s.t. $|\mathcal{Q}| \leq Q$ and an input $x^* \in \{0,1\}^l$, the deterministic verification algorithm outputs 1 if

$$\mathsf{P}_K(\mathsf{H}_{adm}(x^*)) = 1 \ \wedge \ \forall x \in \mathcal{Q}, \mathsf{P}_K(\mathsf{H}_{adm}(x)) = 0$$

and outputs 0 otherwise.

We next argue that the two requirements in Definition 3.2 are satisfied.

Abort Probability. For any set $\mathcal{Q} \subset \{0,1\}^l$ s.t. $|\mathcal{Q}| \leq Q$ and for any input $x^* \notin \mathcal{Q}$, let $x_1, \ldots, x_{|\mathcal{Q}|}$ be the distinct inputs in $\mathcal{Q}$. Also, if $|\mathcal{Q}| < Q$, let $x_{|\mathcal{Q}|+1} = x_{|\mathcal{Q}|+2} = \ldots = x_Q = x_1$. Then we have

$$\begin{aligned}
&\Pr\left[vk \leftarrow \texttt{VKeyGen}_{Q,\delta}(1^\lambda) : \texttt{Verify}_{Q,\delta}(vk, \mathcal{Q}, x^*) = 1\right] \\
=&\Pr\left[K \leftarrow \texttt{AdmSmp}_{Q,\delta}(1^\lambda) : \mathsf{P}_K(\mathsf{H}_{adm}(x^*)) = 1 \ \wedge \ \forall i \in [1, |\mathcal{Q}|], \mathsf{P}_K(\mathsf{H}_{adm}(x_i)) = 0\right] \\
=&\Pr\left[K \leftarrow \texttt{AdmSmp}_{Q,\delta}(1^\lambda) : \mathsf{P}_K(\mathsf{H}_{adm}(x^*)) = 1 \ \wedge \ \forall i \in [1, Q], \mathsf{P}_K(\mathsf{H}_{adm}(x_i)) = 0\right] \\
\in&[\gamma_{min}, \gamma_{max}]
\end{aligned}$$

where $\gamma_{min}, \gamma_{max}$ are defined in Definition A.1. Thus, we can define $\Gamma_{min} = \gamma_{min}$ and $\Gamma_{max} = \gamma_{max}$. Note that since in Definition A.1, we require that $\gamma_{min} \cdot \delta - (\gamma_{max} - \gamma_{min})$ is a non-negligible positive real value, we also have that

$$\mathcal{T} = \Gamma_{min} \cdot \delta - (\Gamma_{max} - \Gamma_{min})$$

is a non-negligible positive real value. Therefore, the algorithms $(\texttt{VKeyGen}_{Q,\delta}, \texttt{Verify}_{Q,\delta})$ satisfy the requirement on abort probability.

Indistinguishability. Next, we prove the indistinguishability property. First, we define the simulator $\texttt{SIM}$ as follows:

1. In the beginning, on receiving a verification key $vk = K \in \{0, 1, \perp\}^t$ and a random string $u^* \in \{0,1\}^{lk}$, the simulator computes:

(a) $\tilde{\boldsymbol{A}} \xleftarrow{\$} \mathbb{Z}_q^{(n-1)\times \bar{n}}$.

(b) $\boldsymbol{r} \xleftarrow{\$} \{0,1\}^{n-1}$.

(c) $\bar{\boldsymbol{A}} = \begin{pmatrix} \boldsymbol{r}^{\intercal} \cdot \tilde{\boldsymbol{A}} \\ \tilde{\boldsymbol{A}} \end{pmatrix} \mod q$.

(d) For $z \in [1, 2t]$:

   i. $\boldsymbol{S}_z \xleftarrow{\$} \mathbb{Z}_q^{\bar{n}\times m}$.

   ii. $\boldsymbol{E}_z \leftarrow \tilde{D}_\sigma^{n\times m}$.

   iii. $\boldsymbol{A}'_z = \bar{\boldsymbol{A}} \cdot \boldsymbol{S}_z + \boldsymbol{E}_z \mod q$.

(e) For $i \in [1, t]$:

   i. If $K[i] = \perp$:

      A. $\boldsymbol{A}_{2i-1} = \boldsymbol{A}'_{2i-1}$.

      B. $\boldsymbol{A}_{2i} = \boldsymbol{A}'_{2i}$.

   ii. Otherwise:

      A. $\boldsymbol{A}_{2i-1} = \boldsymbol{A}'_{2i-1} - \boldsymbol{G} \mod q$.

      B. $\boldsymbol{A}_{2i} = \boldsymbol{A}'_{2i} - K[i] \cdot \boldsymbol{G} \mod q$.

(f) $\boldsymbol{d} = (-1, \boldsymbol{r}^{\intercal})^{\intercal}$.

(g) For $i \in [1, l], j \in [1, k]$:

   i. $\hat{\boldsymbol{s}}_{i,j} \xleftarrow{\$} \mathbb{Z}_q^n$.

   ii. $\check{\boldsymbol{s}}_{i,j} = \hat{\boldsymbol{s}}_{i,j} + \boldsymbol{d} \mod q$.

(h) For $i \in [1, l], j \in [1, k], \iota \in \{0, 1\}$: $v_{i,j,\iota} \xleftarrow{\$} \mathbb{Z}_q$.

(i) Set $\hat{hk} = ((\boldsymbol{A}_z)_{z\in[1,2t]}, (\hat{\boldsymbol{s}}_{i,j})_{i\in[1,l],j\in[1,k]}, (v_{i,j,\iota})_{i\in[1,l],j\in[1,k],\iota\in\{0,1\}})$.

(j) Set $\check{hk} = ((\boldsymbol{A}_z)_{z\in[1,2t]}, (\check{\boldsymbol{s}}_{i,j})_{i\in[1,l],j\in[1,k]}, (v_{i,j,\iota})_{i\in[1,l],j\in[1,k],\iota\in\{0,1\}})$.

(k) Initialize $\alpha = 0$.

2. Then, the simulator answers evaluation queries from $\mathcal{A}$. In particular, given an input $x^{(\ell)}$, the simulator proceeds as follows:

(a) If $\alpha = 0$:

   i. If $\mathsf{P}_K(\mathsf{H}_{adm}(x^{(\ell)})) = 0$:

      A. **Return** $\texttt{Eval}(\hat{hk}, x^{(\ell)})$.

   ii. If $\mathsf{P}_K(\mathsf{H}_{adm}(x^{(\ell)})) = 1$:

      A. For $i \in [1, l], j \in [1, k]$:

$$c_{i,j} \xleftarrow{\$} \{0, 1\}.$$

$$\boldsymbol{s}_{i,j} = \begin{cases} \hat{\boldsymbol{s}}_{i,j} & \text{if } c_{i,j} = 0 \\ \check{\boldsymbol{s}}_{i,j} & \text{if } c_{i,j} = 1 \end{cases}.$$

$$hk = ((\boldsymbol{A}_z)_{z\in[1,2t]}, (\boldsymbol{s}_{i,j})_{i\in[1,l],j\in[1,k]}, (v_{i,j,\iota})_{i\in[1,l],j\in[1,k],\iota\in\{0,1\}}).$$

      B. **Return** $\texttt{Eval}(hk, x^{(\ell)})$.

      C. Set $\alpha = 1$.

(b) If $\alpha = 1$:

i. **Return** $\texttt{Eval}(hk, x^{(\ell)})$.

3. Next, if the simulator is further given an input x^* (i.e., α is still 0 now), the simulator proceeds as follows:
   (a) $\hat{\boldsymbol{u}}^* = \texttt{Eval}(\hat{hk}, x^*)$.
   (b) For $i \in [1, l], j \in [1, k]$:
      i. $\boldsymbol{s}_{i,j} = \begin{cases} \hat{\boldsymbol{s}}_{i,j} & \text{if } u^*[(i-1)k+j] = \hat{\boldsymbol{u}}^*[(i-1)k+j] \\ \check{\boldsymbol{s}}_{i,j} & \text{if } u^*[(i-1)k+j] \neq \hat{\boldsymbol{u}}^*[(i-1)k+j] \end{cases}$.
   (c) **Return** $hk = ((\boldsymbol{A}_z)_{z\in[1,2t]}, (\boldsymbol{s}_{i,j})_{i\in[1,l],j\in[1,k]}, (v_{i,j,\iota})_{i\in[1,l],j\in[1,k],\iota\in\{0,1\}})$

Then, to prove the indistinguishability between the two experiments, we define the following games between the challenger and a PPT adversary $\mathcal{A}$:

- **Game 0.** This is the experiment $\texttt{ExpReal}$. In particular, the challenger interacts with the adversary as follows:
  1. In the beginning, the challenger first samples

  $$\begin{aligned} &\boldsymbol{A}_z \xleftarrow{\$} \mathbb{Z}_q^{n\times m} && \text{for } z \in [1, 2t] \\ &\boldsymbol{s}_{i,j} \xleftarrow{\$} \mathbb{Z}_q^n && \text{for } i \in [1, l], j \in [1, k] \\ &v_{i,j,\iota} \xleftarrow{\$} \mathbb{Z}_q && \text{for } i \in [1, l], j \in [1, k], \iota \in \{0, 1\} \end{aligned}$$

  and $K \leftarrow \texttt{AdmSmp}_{Q,\delta}(1^\lambda)$.
  2. Then it answers evaluation queries from $\mathcal{A}$. In particular, given an input $x^{(\ell)}$, the challenger proceeds as follows:
     (a) $w^{(\ell)} = \mathsf{H}_{adm}(x^{(\ell)})$.
     (b) $\boldsymbol{A}_{w^{(\ell)}} = \texttt{EvalAdm}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{2t}, w^{(\ell)})$.
     (c) $\boldsymbol{b}_{w^{(\ell)}} = \boldsymbol{A}_{w^{(\ell)}} \cdot \boldsymbol{h} \mod q$.
     (d) Let $\boldsymbol{u}_1^{(\ell)}, \ldots, \boldsymbol{u}_l^{(\ell)}$ be k-dimension binary vectors.
     (e) For $i \in [1, l]$:
        i. For $j \in [1, k]$:
           A. $y_{i,j}^{(\ell)} = \boldsymbol{s}_{i,j}^\mathsf{T} \cdot \boldsymbol{b}_{w^{(\ell)}} + v_{i,j,x^{(\ell)}[i]} \mod q$.
           B. $\boldsymbol{u}_i^{(\ell)}[j] = \begin{cases} 0 & \text{if } y_{i,j}^{(\ell)} \in [0, \frac{q-1}{2}] \\ 1 & \text{otherwise} \end{cases}$.
     (f) **Return** $\boldsymbol{u}^{(\ell)} = \begin{pmatrix} \boldsymbol{u}_1^{(\ell)} \\ \vdots \\ \boldsymbol{u}_l^{(\ell)} \end{pmatrix}$.
  3. Next, after $\mathcal{A}$ submits $x^* \in \{0,1\}^l$, the challenger **returns** $\perp$ to the adversary if

  $$\mathsf{P}_K(\mathsf{H}_{adm}(x^*)) = 0 \quad \vee \quad \exists \ell \in [1, Q'], \mathsf{P}_K(\mathsf{H}_{adm}(x^{(\ell)})) = 1$$

  where $Q' \leq Q$ is the number of inputs submitted to the evaluation oracle. Otherwise, it proceeds as follows:

(a) $w^* = \mathsf{H}_{adm}(x^*)$.
(b) $\boldsymbol{A}_{w^*} = \texttt{EvalAdm}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{2t}, w^*)$.
(c) $\boldsymbol{b}_{w^*} = \boldsymbol{A}_{w^*} \cdot \boldsymbol{h} \mod q$.
(d) Let $\boldsymbol{u}_1^*, \ldots, \boldsymbol{u}_l^*$ be k-dimension binary vectors.
(e) For $i \in [1, l]$:
   i. For $j \in [1, k]$:
      A. $y_{i,j}^* = \boldsymbol{s}_{i,j}^\intercal \cdot \boldsymbol{b}_{w^*} + v_{i,j,x^*[i]} \mod q$.
      B. $\boldsymbol{u}_i^*[j] = \begin{cases} 0 & \text{if } y_{i,j}^* \in [0, \frac{q-1}{2}] \\ 1 & \text{otherwise} \end{cases}$.
(f) Set $\boldsymbol{u}^* = \begin{pmatrix} \boldsymbol{u}_1^* \\ \vdots \\ \boldsymbol{u}_l^* \end{pmatrix}$.
(g) Set $hk = ((\boldsymbol{A}_z)_{z \in [1,2t]}, (\boldsymbol{s}_{i,j})_{i \in [1,l], j \in [1,k]}, (v_{i,j,\iota})_{i \in [1,l], j \in [1,k], \iota \in \{0,1\}})$ and **returns** $(hk, \boldsymbol{u}^*)$ to the adversary.

4. Finally, the challenger outputs what $\mathcal{A}$ outputs.

- **Game 1.** This is identical to Game 0 except that after receiving an input $x^{(\ell)}$ from the adversary, the challenger aborts and outputs 2 if

$$\exists (i, j) \in [1, l] \times [1, k],\ y_{i,j}^{(\ell)} \in [\frac{q-1}{2} - \Sigma, \frac{q-1}{2} + \Sigma] \cup [0, \Sigma] \cup [q - \Sigma, q - 1] \quad (10)$$

In addition, after receiving x^*, the challenger aborts and outputs 2 if

$$\exists (i, j) \in [1, l] \times [1, k],\ y_{i,j}^* \in [\frac{q-1}{2} - \Sigma, \frac{q-1}{2} + \Sigma] \cup [0, \Sigma] \cup [q - \Sigma, q - 1] \quad (11)$$

- **Game 2.** This is identical to Game 1 except that the challenger changes the way to generate $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{2t}$. In particular, in the beginning, the challenger samples $\boldsymbol{A}'_z \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ for $z \in [1, 2t]$. Then for $i \in [1, t]$, it sets

$$\boldsymbol{A}_{2i-1} = \boldsymbol{A}'_{2i-1} \quad \text{and} \quad \boldsymbol{A}_{2i} = \boldsymbol{A}'_{2i}$$

if $K[i] = \perp$, and sets

$$\boldsymbol{A}_{2i-1} = \boldsymbol{A}'_{2i-1} - \boldsymbol{G} \mod q \quad \text{and} \quad \boldsymbol{A}_{2i} = \boldsymbol{A}'_{2i} - K[i] \cdot \boldsymbol{G} \mod q$$

otherwise (i.e., if $K[i] \in \{0, 1\}$).

- **Game 3.** This is identical to Game 2 except that the challenger changes the way to generate $\boldsymbol{A}'_1, \ldots, \boldsymbol{A}'_{2t}$. In particular, the challenger samples $\bar{\boldsymbol{A}} \xleftarrow{\$} \mathbb{Z}_q^{n \times \bar{n}}$. Then for $z \in [1, 2t]$, it samples $\boldsymbol{S}_z \xleftarrow{\$} \mathbb{Z}_q^{\bar{n} \times m}$, $\boldsymbol{E}_z \leftarrow \tilde{D}_\sigma^{n \times m}$ and computes

$$\boldsymbol{A}'_z = \bar{\boldsymbol{A}} \cdot \boldsymbol{S}_z + \boldsymbol{E}_z \mod q$$

- **Game 4.** This is identical to Game 3 except that the challenger changes the way to generate $\bar{\boldsymbol{A}}$. In particular, the challenger samples $\tilde{\boldsymbol{A}} \xleftarrow{\$} \mathbb{Z}_q^{(n-1)\times\bar{n}}$ and $\boldsymbol{r} \xleftarrow{\$} \{0,1\}^{n-1}$, then it sets

$$\bar{\boldsymbol{A}} = \begin{pmatrix} \boldsymbol{r}^{\mathsf{T}} \cdot \tilde{\boldsymbol{A}} \\ \tilde{\boldsymbol{A}} \end{pmatrix} \mod q$$

- **Game 5.** This is identical to Game 4 except that the challenger changes the way to generate $\boldsymbol{s}_{i,j}$. Let

$$\boldsymbol{d} = \begin{pmatrix} -1 \\ \boldsymbol{r} \end{pmatrix}$$

then for $i \in [1,l], j \in [1,k]$, the challenger first samples $\hat{\boldsymbol{s}}_{i,j} \xleftarrow{\$} \mathbb{Z}_q^n$ and computes $\check{\boldsymbol{s}}_{i,j} = \hat{\boldsymbol{s}}_{i,j} + \boldsymbol{d} \mod q$. Next, it samples $c_{i,j} \xleftarrow{\$} \{0,1\}$ and sets

$$\boldsymbol{s}_{i,j} = \begin{cases} \hat{\boldsymbol{s}}_{i,j} & \text{if } c_{i,j} = 0 \\ \check{\boldsymbol{s}}_{i,j} & \text{if } c_{i,j} = 1 \end{cases}$$

- **Game 6.** This is identical to Game 5 except that after receiving an input $x^{(\ell)}$ from the adversary and checking if Equation (10) is satisfied, the challenger additionally computes

$$\hat{y}_{i,j}^{(\ell)} = \hat{\boldsymbol{s}}_{i,j}^{\mathsf{T}} \cdot \boldsymbol{b}_{w^{(\ell)}} + v_{i,j,x^{(\ell)}[i]} \mod q$$

$$\check{y}_{i,j}^{(\ell)} = \check{\boldsymbol{s}}_{i,j}^{\mathsf{T}} \cdot \boldsymbol{b}_{w^{(\ell)}} + v_{i,j,x^{(\ell)}[i]} \mod q$$

and

$$\hat{\boldsymbol{u}}_i^{(\ell)}[j] = \begin{cases} 0 & \text{if } \hat{y}_{i,j}^{(\ell)} \in [0, \frac{q-1}{2}] \\ 1 & \text{otherwise} \end{cases}$$

$$\check{\boldsymbol{u}}_i^{(\ell)}[j] = \begin{cases} 0 & \text{if } \check{y}_{i,j}^{(\ell)} \in [0, \frac{q-1}{2}] \\ 1 & \text{otherwise} \end{cases}$$

for $i \in [1,l], j \in [1,k]$. Then it aborts and outputs 3 if

$$\mathsf{P}_K(\mathsf{H}_{adm}(x^{(\ell)})) = 0 \quad \wedge \quad \exists (i,j) \in [1,l] \times [1,k],\ \hat{\boldsymbol{u}}_i^{(\ell)}[j] \neq \check{\boldsymbol{u}}_i^{(\ell)}[j]$$

- **Game 7.** This is identical to Game 6 except that after receiving the input x^* from the adversary and checking if Equation (11) is satisfied, the challenger additionally computes

$$\hat{y}_{i,j}^{*} = \hat{\boldsymbol{s}}_{i,j}^{\mathsf{T}} \cdot \boldsymbol{b}_{w^*} + v_{i,j,x^*[i]} \mod q$$

$$\check{y}_{i,j}^{*} = \check{\boldsymbol{s}}_{i,j}^{\mathsf{T}} \cdot \boldsymbol{b}_{w^*} + v_{i,j,x^*[i]} \mod q$$

and

$$\hat{\boldsymbol{u}}_i^{*}[j] = \begin{cases} 0 & \text{if } \hat{y}_{i,j}^{*} \in [0, \frac{q-1}{2}] \\ 1 & \text{otherwise} \end{cases}$$

$$\check{\boldsymbol{u}}_i^*[j] = \begin{cases} 0 & \text{if } \check{y}_{i,j}^* \in [0, \frac{q-1}{2}] \\ 1 & \text{otherwise} \end{cases}$$

for $i \in [1, l], j \in [1, k]$. Then it aborts and outputs 3 if

$$\mathsf{P}_K(\mathsf{H}_{adm}(x^*)) = 1 \quad \wedge \quad \exists (i,j) \in [1,l] \times [1,k],\ \hat{\boldsymbol{u}}_i^*[j] = \check{\boldsymbol{u}}_i^*[j]$$

- **Game 8.** This is identical to Game 7 except that the challenger does not perform the checks introduced in Game 1.
- **Game 9.** This is identical to Game 8 except that the challenger uses $\hat{\boldsymbol{s}}_{i,j}$ instead of $\boldsymbol{s}_{i,j}$ to answer evaluation oracle queries from the adversary until the adversary submits an input $x^{(\ell)}$ s.t. $\mathsf{P}_K(\mathsf{H}_{adm}(x^{(\ell)})) = 1$. Note that in Game 9, the challenger can postpone the generation of $c_{i,j}$ and $\boldsymbol{s}_{i,j}$ after the adversary submits such "bad" evaluation oracle queries, or after the adversary submits x^* in case $\mathsf{P}_K(\mathsf{H}_{adm}(x^{(\ell)})) = 0$ for all $\ell \in [1, Q']$.
- **Game 10.** This is identical to Game 9 except that the challenger changes the way to compute $\boldsymbol{s}_{i,j}$ in case $\texttt{Verify}_{Q,\delta}(K, \{x^{(1)}, \ldots, x^{(Q')}\}, x^*) = 1$. In more detail, in this case, after receiving x^*, the challenger samples $u_{i,j} \xleftarrow{\$} \{0,1\}$ and sets
$$\boldsymbol{s}_{i,j} = \begin{cases} \hat{\boldsymbol{s}}_{i,j} & \text{if } u_{i,j} = \hat{\boldsymbol{u}}_i^*[j] \\ \check{\boldsymbol{s}}_{i,j} & \text{if } u_{i,j} \neq \hat{\boldsymbol{u}}_i^*[j] \end{cases}$$
for $i \in [1, l], j \in [1, k]$.
- **Game 11.** This is identical to Game 10 except that the challenger samples $u_{i,j}$ in the beginning. In addition, at the end of the game, it sets $\boldsymbol{u}^* = (u_{1,1}, \ldots, u_{l,k})^\intercal$ if $\texttt{Verify}_{Q,\delta}(K, \{x^{(1)}, \ldots, x^{(Q')}\}, x^*) = 1$.
- **Game 12.** This is identical to Game 11 except that the challenger does not perform the check introduced in Game 6 and Game 7.

It is easy to see that Game 12 is identical to the experiment $\texttt{ExpIdeal}_{\mathcal{A},\texttt{SIM}}$ for the simulator $\texttt{SIM}$ defined above. Let $\mathcal{E}_i$ be the output of Game i for $i \in [0, 12]$. We next show that $|\Pr[\mathcal{E}_0 = 1] - \Pr[\mathcal{E}_{12} = 1]| \leq negl(\lambda)$ via proving the following lemmas.

Lemma B.1. $|\Pr[\mathcal{E}_0 = 1] - \Pr[\mathcal{E}_1 = 1]| \leq Pr[\mathcal{E}_1 = 2] \leq negl(\lambda)$.

Proof. First, Game 0 and Game 1 are identical unless the challenger aborts and outputs 2 in Game 1. Thus, we have

$$|\Pr[\mathcal{E}_0 = 1] - \Pr[\mathcal{E}_1 = 1]| \leq Pr[\mathcal{E}_1 = 2]$$

Next, define the function $f_{i,j} : \{0,1\}^l \to \mathbb{Z}_q$ as

$$f_{i,j}(x) = \boldsymbol{s}_{i,j}^\intercal \cdot \texttt{EvalAdm}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{2t}, \mathsf{H}_{adm}(x)) \cdot \boldsymbol{h}$$

Note that for $i, j, \iota \in [1, l] \times [1, k] \times \{0, 1\}$, $v_{i,j,\iota}$ is a random element in $\mathbb{Z}_q$ that is independent of $f_{i,j}$. Then by Lemma 2.3 and the fact that $q \geq 2^{l+\omega(\log\lambda)}(4\Sigma+2)$, for any $i, j, \iota \in [1, l] \times [1, k] \times \{0, 1\}$, the probability that there exists $x \in \{0, 1\}^l$ s.t.

$$v_{i,j,\iota} + f(x) \in [\frac{q-1}{2} - \Sigma, \frac{q-1}{2} + \Sigma] \cup [0, \Sigma] \cup [q - \Sigma, q - 1]$$

is negligible. Therefore, the probability that Equation (10) or Equation (11) is satisfied is also negligible and as a result, $Pr[\mathcal{E}_1 = 2] \leq negl(\lambda)$.

□

Lemma B.2. *For* $\mathfrak{b} \in \{1, 2\}$*,* $|\Pr[\mathcal{E}_1 = \mathfrak{b}] - \Pr[\mathcal{E}_2 = \mathfrak{b}]| = 0$.

Proof. Game 1 and Game 2 only differ in the way for generating $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{2t}$. Since in both Game 1 and Game 2, $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{2t}$ are uniform matrices in $\mathbb{Z}_q^{n \times m}$, the two games are identical. □

Lemma B.3. *For* $\mathfrak{b} \in \{1, 2\}$*,* $|\Pr[\mathcal{E}_2 = \mathfrak{b}] - \Pr[\mathcal{E}_3 = \mathfrak{b}]| \leq negl(\lambda)$.

Proof. Indistinguishability between Game 2 and Game 3 comes from the hardness of $\text{LWE}_{\bar{n},q,\tilde{D}_\sigma}$ by viewing $\bar{\boldsymbol{A}}$ as the public matrix, each column of $\boldsymbol{S}_z$ as the secret vector, and each column of $\boldsymbol{E}_z$ as the error vector. □

Lemma B.4. *For* $\mathfrak{b} \in \{1, 2\}$*,* $|\Pr[\mathcal{E}_3 = \mathfrak{b}] - \Pr[\mathcal{E}_4 = \mathfrak{b}]| \leq negl(\lambda)$.

Proof. Indistinguishability between Game 3 and Game 4 comes from the leftover hash lemma (Lemma 2.1) directly. □

Lemma B.5. *For* $\mathfrak{b} \in \{1, 2\}$*,* $|\Pr[\mathcal{E}_4 = \mathfrak{b}] - \Pr[\mathcal{E}_5 = \mathfrak{b}]| = 0$.

Proof. Game 4 and Game 5 only differ in the way for generating $\boldsymbol{s}_{i,j}$. Since for $i \in [1, l], j \in [1, k]$, $\boldsymbol{s}_{i,j}$ is a uniform vector in $\mathbb{Z}_q^n$ in both games, thus, the two games are identical. □

Lemma B.6. *For* $\mathfrak{b} \in \{1, 2\}$*,* $|\Pr[\mathcal{E}_5 = \mathfrak{b}] - \Pr[\mathcal{E}_6 = \mathfrak{b}]| = \Pr[\mathcal{E}_6 = 3] = 0$.

Proof. First, by Lemma A.7, for any $\ell \in [1, Q']$, there exists $\boldsymbol{T}^{(\ell)} \in [-m, m]^{2tm \times m}$ s.t.

$$\boldsymbol{A}_{w^{(\ell)}} = (\boldsymbol{A}'_1, \ldots, \boldsymbol{A}'_{2t}) \cdot \boldsymbol{T}^{(\ell)} - \mathsf{P}_K(w^{(\ell)}) \cdot \boldsymbol{G}$$

Thus, for any $i \in [1, l], j \in [1, k]$, we have

$$\begin{aligned}
&\check{y}_{i,j}^{(\ell)} - \hat{y}_{i,j}^{(\ell)} \\
=&(\check{\boldsymbol{s}}_{i,j}^\intercal \cdot \boldsymbol{b}_{w^{(\ell)}} + v_{i,j,x^{(\ell)}[i]}) - (\hat{\boldsymbol{s}}_{i,j}^\intercal \cdot \boldsymbol{b}_{w^{(\ell)}} + v_{i,j,x^{(\ell)}[i]}) \\
=&(\check{\boldsymbol{s}}_{i,j}^\intercal - \hat{\boldsymbol{s}}_{i,j}^\intercal) \cdot \boldsymbol{b}_{w^{(\ell)}} \\
=&\boldsymbol{d}^\intercal \cdot \boldsymbol{b}_{w^{(\ell)}} \\
=&\boldsymbol{d}^\intercal \cdot \boldsymbol{A}_{w^{(\ell)}} \cdot \boldsymbol{h} \\
=&\boldsymbol{d}^\intercal \cdot ((\boldsymbol{A}'_1, \ldots, \boldsymbol{A}'_{2t}) \cdot \boldsymbol{T}^{(\ell)} - \mathsf{P}_K(w^{(\ell)}) \cdot \boldsymbol{G}) \cdot \boldsymbol{h} \\
=&\boldsymbol{d}^\intercal \cdot ((\bar{\boldsymbol{A}} \cdot (\boldsymbol{S}_1, \ldots, \boldsymbol{S}_{2t}) + (\boldsymbol{E}_1, \ldots, \boldsymbol{E}_{2t})) \cdot \boldsymbol{T}^{(\ell)} - \mathsf{P}_K(w^{(\ell)}) \cdot \boldsymbol{G}) \cdot \boldsymbol{h} \\
=&\boldsymbol{d}^\intercal \cdot \bar{\boldsymbol{A}} \cdot (\boldsymbol{S}_1, \ldots, \boldsymbol{S}_{2t}) \cdot \boldsymbol{T}^{(\ell)} \cdot \boldsymbol{h} + E^{(\ell)} - \boldsymbol{d}^\intercal \cdot \mathsf{P}_K(w^{(\ell)}) \cdot \boldsymbol{G} \cdot \boldsymbol{h} \\
=&E^{(\ell)} - \boldsymbol{d}^\intercal \cdot \mathsf{P}_K(w^{(\ell)}) \cdot \boldsymbol{G} \cdot \boldsymbol{h}
\end{aligned} \quad \mod q \quad (12)$$

where the last equality comes from the fact that $\boldsymbol{d}^\intercal \cdot \bar{\boldsymbol{A}} = 0 \mod q$ and we define

$$E^{(\ell)} = \boldsymbol{d}^\intercal \cdot (\boldsymbol{E}_1, \dots, \boldsymbol{E}_{2t}) \cdot \boldsymbol{T}^{(\ell)} \cdot \boldsymbol{h}$$

Note that $\boldsymbol{d} \in \{-1, 0, 1\}^n$, $(\boldsymbol{E}_1, \dots, \boldsymbol{E}_{2t}) \in [-\lambda \cdot \sigma, \lambda \cdot \sigma]^{n \times 2tm}$, $\boldsymbol{T}^{(\ell)} \in [-m, m]^{2tm \times m}$, and $\boldsymbol{h}$ is a binary vector with at most $\lceil \log q \rceil$ non-zero elements, thus we have

$$|E^{(\ell)}| \le 2tm^3 \lambda \sigma = \varSigma$$

Therefore, if $\mathsf{P}_K(w^{(\ell)}) = 0$, we have

$$|\check{y}_{i,j}^{(\ell)} - \hat{y}_{i,j}^{(\ell)}| \le \varSigma$$

Note that, we have either $y_{i,j}^{(\ell)} = \check{y}_{i,j}^{(\ell)}$ or $y_{i,j}^{(\ell)} = \hat{y}_{i,j}^{(\ell)}$, and w.l.o.g., we assume $y_{i,j}^{(\ell)} = \check{y}_{i,j}^{(\ell)}$. Now, further assuming that Equation (10) is not satisfied (i.e., the challenger will not abort and output 2 before performing the new check introduced in Game 6), then we have

$$\check{y}_{i,j}^{(\ell)} \in (\varSigma, \frac{q-1}{2} - \varSigma) \cup (\frac{q-1}{2} + \varSigma, q - \varSigma)$$

That is, if $\check{\boldsymbol{u}}_i^{(\ell)}[j] = 0$ (i.e., $\check{y}_{i,j}^{(\ell)} \in (\varSigma, \frac{q-1}{2} - \varSigma)$), then we have $\hat{y}_{i,j}^{(\ell)} \in [1, \frac{q-1}{2} - 1]$ and therefore $\hat{\boldsymbol{u}}_i^{(\ell)}[j] = 0$; and if $\check{\boldsymbol{u}}_i^{(\ell)}[j] = 1$ (i.e., $\check{y}_{i,j}^{(\ell)} \in (\frac{q-1}{2} + \varSigma, q - \varSigma)$), then we have $\hat{y}_{i,j}^{(\ell)} \in [\frac{q+1}{2}, q-1]$ and therefore $\hat{\boldsymbol{u}}_i^{(\ell)}[j] = 1$. That is, we always have $\hat{\boldsymbol{u}}_i^{(\ell)}[j] = \check{\boldsymbol{u}}_i^{(\ell)}[j]$.

To summarize, if the challenger does not outputs 2 and $\mathsf{P}_K(w^{(\ell)}) = 0$, we always have $\hat{\boldsymbol{u}}_i^{(\ell)}[j] = \check{\boldsymbol{u}}_i^{(\ell)}[j]$ for any $i \in [1, l], j \in [1, k]$. Therefore, the challenger will never outputs 3 in Game 6 and as a result, Game 5 and Game 6 are identical. □

Lemma B.7. *For* $\mathfrak{b} \in \{1, 2\}$, $|\Pr[\mathcal{E}_6 = \mathfrak{b}] - \Pr[\mathcal{E}_7 = \mathfrak{b}]| = \Pr[\mathcal{E}_7 = 3] = 0$.

Proof. First, similar to Equation (12), there exists $E^* \in [-\varSigma, \varSigma]$ s.t.

$$\begin{aligned}
&\check{y}_{i,j}^* - \hat{y}_{i,j}^* \\
=&E^* - \boldsymbol{d}^\intercal \cdot \mathsf{P}_K(w^*) \cdot \boldsymbol{G} \cdot \boldsymbol{h} \\
=&E^* - \mathsf{P}_K(w^*) \cdot \boldsymbol{d}^\intercal \cdot \begin{pmatrix} \frac{q-1}{2} \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \mod q \\
=&E^* + \mathsf{P}_K(w^*) \cdot \frac{q-1}{2}
\end{aligned}$$

Again, assume w.l.o.g. that $y_{i,j}^* = \check{y}_{i,j}^*$ and suppose that Equation (11) is not satisfied, then we have

$$\check{y}_{i,j}^* \in (\varSigma, \frac{q-1}{2} - \varSigma) \cup (\frac{q-1}{2} + \varSigma, q - \varSigma)$$

Also assume that $\mathsf{P}_K(w^*) = 1$. Then if $\check{\boldsymbol{u}}_i^*[j] = 0$ (i.e., $\check{y}_{i,j}^* \in (\Sigma, \frac{q-1}{2} - \Sigma)$), then we have

$$\hat{y}_{i,j}^* = \check{y}_{i,j}^* - E^* - \frac{q-1}{2} \in [\frac{q+3}{2}, q-1]$$

and therefore $\hat{\boldsymbol{u}}_i^*[j] = 1$; and if $\check{\boldsymbol{u}}_i^*[j] = 1$ (i.e., $\check{y}_{i,j}^* \in (\frac{q-1}{2} + \Sigma, q - \Sigma)$), then we have

$$\hat{y}_{i,j}^* = \check{y}_{i,j}^* - E^* - \frac{q-1}{2} \in [1, \frac{q-1}{2}]$$

and therefore $\hat{\boldsymbol{u}}_i^*[j] = 0$. That is, we always have $\hat{\boldsymbol{u}}_i^*[j] \neq \check{\boldsymbol{u}}_i^*[j]$.

To summarize, if the challenger does not outputs 2 and $\mathsf{P}_K(w^*) = 1$, we always have $\hat{\boldsymbol{u}}_i^*[j] \neq \check{\boldsymbol{u}}_i^*[j]$ for any $i \in [1, l], j \in [1, k]$. Therefore, the challenger will never outputs 3 in Game 7 and as a result, Game 6 and Game 7 are identical. □

Lemma B.8. *For* $\mathfrak{b} \in \{1, 3\}$, $|\Pr[\mathcal{E}_7 = \mathfrak{b}] - \Pr[\mathcal{E}_8 = \mathfrak{b}]| \leq negl(\lambda)$.

Proof. Game 7 and Game 8 are identical unless Game 7 outputs 2. From Lemma B.1 to Lemma B.7, we have $\Pr[\mathcal{E}_7 = 2] \leq negl(\lambda)$. Thus, the probability that the two games differ is also negligible. □

Lemma B.9. *For* $\mathfrak{b} \in \{1, 3\}$, $|\Pr[\mathcal{E}_8 = \mathfrak{b}] - \Pr[\mathcal{E}_9 = \mathfrak{b}]| = 0$.

Proof. In Game 9, the challenger outputs

$$\hat{\boldsymbol{u}}^{(\ell)} = \begin{pmatrix} \hat{\boldsymbol{u}}_1^{(\ell)} \\ \vdots \\ \hat{\boldsymbol{u}}_l^{(\ell)} \end{pmatrix}$$

instead of $\boldsymbol{u}^{(\ell)}$ for the first $Q^\dagger$ evaluation oracle queries if $\mathsf{P}_K(w^{(\ell)}) = 0$ for $\ell \in [1, Q^\dagger]$. As for all $i \in [1, l], j \in [1, k]$, $\hat{\boldsymbol{u}}_i^{(\ell)}[j] = \check{\boldsymbol{u}}_i^{(\ell)}[j] = \boldsymbol{u}_i^{(\ell)}[j]$ unless the challenger outputs 3[21], the outputs of the two games are identically distributed. □

Lemma B.10. *For* $\mathfrak{b} \in \{1, 3\}$, $|\Pr[\mathcal{E}_9 = \mathfrak{b}] - \Pr[\mathcal{E}_{10} = \mathfrak{b}]| = 0$.

Proof. Game 9 and Game 10 only differ in the way for generating $\boldsymbol{s}_{i,j}$ in case that $\texttt{Verify}_{Q,\delta}(K, \{x^{(1)}, \ldots, x^{(Q')}\}, x^*) = 1$ (i.e., $\mathsf{P}_K(x^{(\ell)}) = 0$ for all $\ell \in [1, Q']$ and $\mathsf{P}_K(x^*) = 1$). In Game 10, as $u_{i,j}$ is a uniform bit, the probability that $u_{i,j} = \hat{\boldsymbol{u}}_i^*[j]$ is $\frac{1}{2}$. Thus, in both Game 9 and Game 10, we have

$$\Pr[\boldsymbol{s}_{i,j} = \hat{\boldsymbol{s}}_{i,j}] = \Pr[\boldsymbol{s}_{i,j} = \check{\boldsymbol{s}}_{i,j}] = \frac{1}{2}$$

Therefore, the two games are identical. □

Lemma B.11. *For* $\mathfrak{b} \in \{1, 3\}$, $|\Pr[\mathcal{E}_{10} = \mathfrak{b}] - \Pr[\mathcal{E}_{11} = \mathfrak{b}]| = 0$.

[21] In this case, the challenger aborts the game and will not return $\hat{\boldsymbol{u}}^{(\ell)}$ to the adversary.

Proof. First, as $u_{i,j}$ is sampled uniformly at random and is independent from other variables, it will not change the game if we sample $u_{i,j}$ in beginning.

In addition, assume that $\texttt{Verify}_{Q,\delta}(K, \{x^{(1)}, \ldots, x^{(Q')}\}, x^*) = 1$ and the challenger does not output 3, then for all $i \in [1,l], j \in [1,k]$, $\hat{\boldsymbol{u}}_i^*[j] \neq \check{\boldsymbol{u}}_i^{(\ell)}[j]$. Thus, if $u_{i,j} \neq \hat{\boldsymbol{u}}_i^*[j]$, then $u_{i,j} = \check{\boldsymbol{u}}_i^*[j]$. This implies that

$$u_{i,j} = \boldsymbol{u}_i^*[j] \quad \text{for all } i \in [1,l], j \in [1,k]$$

Thus, if the challenger does not output 3, then it will not change the game if we replace $\boldsymbol{u}_i^*[j]$ with $u_{i,j}$. On the other hand, if the challenger outputs 3 here, then $\boldsymbol{u}^*$ will not be returned to the adversary, and the two games are still identical. □

Lemma B.12. $|\Pr[\mathcal{E}_{11} = 1] - \Pr[\mathcal{E}_{12} = 1]| \leq \mathit{negl}(\lambda)$.

Proof. Game 11 and Game 12 are identical unless Game 11 outputs 3. From Lemma B.7 to Lemma B.11, we have $\Pr[\mathcal{E}_{11} = 3] \leq \mathit{negl}(\lambda)$. Thus, the probability that the two games differ is also negligible. □

Combining Lemma B.1 to Lemma B.12, we have $|\Pr[\mathcal{E}_0 = 1] - \Pr[\mathcal{E}_{12} = 1]| \leq \mathit{negl}(\lambda)$, i.e., $|\Pr[\texttt{ExpReal}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\texttt{ExpIdeal}_{\mathcal{A},\texttt{SIM}}(1^\lambda) = 1]| \leq \mathit{negl}(\lambda)$. This completes the proof of indistinguishability required in the explainability property.

C Proof of Theorem 4.1

We present proof of Theorem 4.1 in this section. Let $\mathsf{PRF} = (\texttt{KeyGen}, \texttt{Eval}, \texttt{Constrain}, \texttt{ConstrainEval})$ be a private puncturable PRF with weakly adaptive pseudorandomness. We prove that it also has the fully adaptive pseudorandomness defined in Definition 4.2. We first define the following games between the challenger and a PPT adversary $\mathcal{A}$. Here, we use $\tau = \mathit{poly}(\lambda)$ to denote the upper bound on the sizes of sets submitted to the constrain oracle by $\mathcal{A}$.

We also assume w.l.o.g. that the adversary $\mathcal{A}$ has made at least one constrain oracle query before submitting the challenge x^*. This is because for any adversary $\mathcal{A}$, we can construct an adversary $\mathcal{A}'$ that is identical to $\mathcal{A}$ except that before submitting the challenge x^*, it first submits $\{x^*\}$ to its constrain oracle. Note that $\mathcal{A}'$ can succeed in breaking the pseudorandomness of PRF if $\mathcal{A}$ succeeds and it satisfies Equation (2) in Definition 4.2 if $\mathcal{A}$ satisfies this equation.

- **Game 0.** This is the experiment $\texttt{ExpPR}$ defined in Definition 4.2 (with a random b).
- **Game 1.** This is identical to Game 0 except that the challenger samples an integer $\alpha \xleftarrow{\$} [1,\tau]$ and $b^\dagger \xleftarrow{\$} \{0,1\}$ in the beginning of the game. Let $\mathcal{P}_1 = \{x_{1,1}, \ldots, x_{1,t}\}$ be the first set submitted to the constrain oracle, where $x_{1,1}$ to $x_{1,t}$ are in lexicographic order. Then at the end of the game, if $x^* = x_{1,\alpha}$, the challenger outputs 1 if $b' = b$ and output 0 if $b' \neq b$. Otherwise, it outputs 1 if $b^\dagger = b$ and outputs 0 if $b^\dagger \neq b$.

- **Game 2.** This is identical to Game 2 except that after receiving x^* from the adversary, the challenger aborts the game if $x_{1,\alpha} \neq x^*$.[22] In case the challenger aborts, it outputs 1 if $b^\dagger = b$ and outputs 0 if $b^\dagger \neq b$.

Let $\mathcal{E}_i$ be the output of Game i and let $\delta_i = |\Pr[\mathcal{E}_i = 1] - \frac{1}{2}|$ for $i \in [0, 2]$. We next show that δ_0 is negligible via proving the following lemmas.

Lemma C.1. $\delta_1 \geq \frac{\delta_0}{\tau}$.

Proof. First, by Equation (2), we always have $x^* \in \mathcal{P}_1$, i.e., there exists $i^* \in [1, t]$ s.t. $x_{1,i^*} = x^*$. Also, by definition, we have $t \leq \tau$. Since α is a random integer in $[1, \tau]$, the probability that $\alpha = i^*$ is $\frac{1}{\tau}$. That is, the probability that Game 1 resamples the output of $\mathcal{A}$ is $1 - \frac{1}{\tau}$ and then by Lemma A.6 we have $\delta_1 \geq \frac{\delta_0}{\tau}$. $\square$

Lemma C.2. $\delta_2 = \delta_1$.

Proof. Game 2 and Game 1 are identically proceeded if $x_{1,\alpha} = x^*$. In case that $x_{1,\alpha} \neq x^*$, both games will output 1 with probability 1/2. Thus, the output of the two games are identical. $\square$

Lemma C.3. $\delta_2 \leq negl(\lambda)$.

Proof. This comes from the weakly adaptive pseudorandomness of PRF by a direct reduction. More precisely, assume δ_2 is non-negligible, then we can construct an adversary $\mathcal{B}$ that breaks the weakly adaptive security of PRF.

The adversary $\mathcal{B}$ first samples $\alpha \xleftarrow{\$} [1, \tau]$ and $b^\dagger \xleftarrow{\$} \{0, 1\}$. Then it invokes the adversary $\mathcal{A}$ and answers its evaluation oracle queries via redirecting the query to its own evaluation oracle. Then after $\mathcal{A}$ makes the first constrain oracle query $\mathcal{P}_1 = \{x_{1,1}, \ldots, x_{1,t}\}$, where $x_{1,1}$ to $x_{1,t}$ are in lexicographic order, $\mathcal{B}$ submits $x_{1,\alpha}$ as its own challenge and receives a response y^*. Next, $\mathcal{B}$ answers the evaluation oracle queries and constrain oracle queries from $\mathcal{A}$ via redirecting the queries to its own evaluation oracle and constrain oracle. Then on receiving the challenge x^* from $\mathcal{A}$, $\mathcal{B}$ outputs $b^\dagger$ if $x_{1,\alpha} \neq x^*$. Otherwise it returns y^* to $\mathcal{A}$. The adversary $\mathcal{B}$ then answers the evaluation oracle queries and constrain oracle queries from $\mathcal{A}$ via redirecting the queries to its own evaluation oracle and constrain oracle. Finally, on receiving a bit b' from $\mathcal{A}$, $\mathcal{B}$ outputs b'.

It is easy to see, $\mathcal{B}$ can simulate Game 2 perfectly and $\mathcal{B}$ succeeds in guessing b iff Game 1 outputs 1. Thus, $\mathcal{B}$ breaks the weakly adaptive pseudorandomness of PRF if δ_2 is non-negligible. This completes the reduction. $\square$

Combining Lemma C.1 to Lemma C.3, we have $\delta_0 \leq \tau \cdot negl(\lambda)$, which is also negligible. This completes the proof of Theorem 4.1.

[22] Recall that we assume w.l.o.g. that the adversary will query the constrain oracle for at least one time before submitting x^*, thus the challenger knows $x_{1,\alpha}$ at this time.

D Security Analysis of Private Puncturable PRF

We present proof of Theorem 4.2 in this section. More precisely, we will prove the correctness, pseudorandomness, and privacy of PRF.

The following lemmas are useful in the proofs of these three properties.

Lemma D.1. *Let $\boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k \in \mathbb{Z}_q^{n \times m}$, and $\boldsymbol{s} \in \mathbb{Z}_q^n$ s.t. $\boldsymbol{s}[1] = 1$. Let $u^* \in \{0,1\}^L$, $\bar{r} \in \mathbb{Z}_q$ and $r = \boldsymbol{G}_{1,q}^{-1}(\bar{r})$. Also, let $(pk, sk) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda, 1^d)$ and let ct be encryption of (u^*, r) under the public key pk. Let*

$$\boldsymbol{a}_0^\intercal = \boldsymbol{s}^\intercal \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_0^\intercal, \quad \boldsymbol{a}_j^\intercal = \boldsymbol{s}^\intercal \cdot (\boldsymbol{A}_j + ct[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_j^\intercal \quad \text{for } j \in [1, N]$$

$$\boldsymbol{b}_j^\intercal = \boldsymbol{s}^\intercal \cdot (\boldsymbol{B}_j + sk[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{N+j}^\intercal \quad \text{for } j \in [1, k]$$

where $\boldsymbol{e}_j \leftarrow \tilde{D}_\sigma^m$ for $j \in [0, N+k]$. Let $u \in \{0,1\}^L$ and let

$$\boldsymbol{C}_{j,\iota} = \mathtt{EvalPK}(\mathtt{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N) \quad \text{for } j \in [1, \varkappa], \iota \in [1, k]$$

$$\boldsymbol{D}_j = \mathtt{IPEvalPK}(\boldsymbol{C}_{j,1}, \ldots, \boldsymbol{C}_{j,k}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k) \quad \text{for } j \in [1, \varkappa]$$

Let

$$\widetilde{ct}_{j,\iota} = \mathtt{C}_{u,j,\iota}(ct) \quad \text{for } j \in [1, \varkappa], \iota \in [1, k]$$

$$\boldsymbol{c}_{j,\iota} = \mathtt{EvalCT}(\mathtt{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_N, ct) \quad \text{for } j \in [1, \varkappa], \iota \in [1, k]$$

$$\boldsymbol{d}_j = \mathtt{IPEvalCT}(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_k, \boldsymbol{c}_{j,1}, \ldots, \boldsymbol{c}_{j,k}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_k, \widetilde{ct}_{j,1}, \ldots, \widetilde{ct}_{j,k}) \quad \text{for } j \in [1, \varkappa]$$

Now, let β be a bit that $\beta = 1$ if $u = u^*$ and $\beta = 0$ if $u \neq u^*$. Then we have

$$|\sum_{j=1}^{\varkappa} (\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j)[1] - \beta \cdot \bar{r}| \leq \Sigma'$$

Proof. First, by Lemma A.1, we have

$$\|\boldsymbol{c}_{j,\iota}^\intercal - \boldsymbol{s}^\intercal (\boldsymbol{C}_{j,\iota} + \mathtt{C}_{u,j,\iota}(ct) \cdot \boldsymbol{G})\|_\infty \leq (m+2)^D \cdot \lambda \cdot \sigma$$

Then by Lemma A.2, we have

$$\|\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal (\boldsymbol{D}_j + \sum_{\iota=1}^{k} \mathtt{C}_{u,j,\iota}(ct) \cdot sk[\iota] \cdot \boldsymbol{G})\|_\infty \leq k \cdot (m+1) \cdot (m+2)^D \cdot \lambda \cdot \sigma$$

This implies that

$$|(\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal (\boldsymbol{D}_j + \sum_{\iota=1}^{k} \mathtt{C}_{u,j,\iota}(ct) \cdot sk[\iota] \cdot \boldsymbol{G}))[1]| \leq k \cdot (m+1) \cdot (m+2)^D \cdot \lambda \cdot \sigma$$

Since $\boldsymbol{s}[1] = 1$, we have $(\boldsymbol{s}^\intercal \cdot \boldsymbol{G})[1] = 1$. Therefore, we have

$$|(\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal \boldsymbol{D}_j)[1] - \sum_{\iota=1}^{k} \mathtt{C}_{u,j,\iota}(ct) \cdot sk[\iota]| \leq k \cdot (m+1) \cdot (m+2)^D \cdot \lambda \cdot \sigma \tag{13}$$

In addition, as ct is encryption of $u^*\|r$, by Lemma A.4, we have

$$|\sum_{\iota=1}^{k} \mathtt{C}_{u,j,\iota}(ct) \cdot sk[\iota] - 2^{j-1} \cdot \mathtt{eq}_{u,j}(u^*, r)| \leq \lambda \cdot \sigma \cdot k^{O(d)}$$

Since $\mathtt{eq}_{u,j}(u^*, r)$ is equal to $r[j]$ if $u = u^*$ and is equal to 0 otherwise, we have

$$|\sum_{\iota=1}^{k} \mathtt{C}_{u,j,\iota}(ct) \cdot sk[\iota] - 2^{j-1} \cdot \beta \cdot r[j]| \leq \lambda \cdot \sigma \cdot k^{O(d)} \tag{14}$$

Then by the triangle inequality, Equation (13) and Equation (14), we have

$$\begin{aligned}
&|(\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal \boldsymbol{D}_j)[1] - 2^{j-1} \cdot \beta \cdot r[j]| \\
\leq &|\sum_{\iota=1}^{k} \mathtt{C}_{u,j,\iota}(ct) \cdot sk[\iota] - 2^{j-1} \cdot \beta \cdot r[j]| + |(\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal \boldsymbol{D}_j)[1] - \sum_{\iota=1}^{k} \mathtt{C}_{u,j,\iota}(ct) \cdot sk[\iota]| \\
\leq &\lambda \cdot \sigma \cdot k^{O(d)} + k \cdot (m+1) \cdot (m+2)^D \cdot \lambda \cdot \sigma
\end{aligned}$$

Finally, we have

$$\begin{aligned}
&|\sum_{j=1}^{\varkappa} (\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j)[1] - \beta \cdot \bar{r}| \\
=&|\sum_{j=1}^{\varkappa} (\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j)[1] - \sum_{j=1}^{\varkappa} \beta \cdot 2^{j-1} \cdot r[j]| \\
=&|\sum_{j=1}^{\varkappa} ((\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j)[1] - \beta \cdot 2^{j-1} \cdot r[j])| \\
\leq &\sum_{j=1}^{\varkappa} |(\boldsymbol{d}_j^\intercal - \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j)[1] - \beta \cdot 2^{j-1} \cdot r[j]| \\
\leq &\varkappa \cdot (\lambda \cdot \sigma \cdot k^{O(d)} + k \cdot (m+1) \cdot (m+2)^D \cdot \lambda \cdot \sigma) \\
\leq &\varkappa \cdot \lambda \cdot \sigma \cdot (k^{O(d)} + k \cdot m^{O(D)}) \\
\leq &\Sigma'
\end{aligned}$$

□

Lemma D.2. *Let $\bar{y}, \bar{y}' \in \mathbb{Z}_q$ and $E \in [-\Sigma, \Sigma]$ satisfying $\bar{y} - \bar{y}' = E \mod q$. Also, let $y = \lfloor \bar{y} \rceil_p \mod p$ and $y' = \lfloor \bar{y}' \rceil_p \mod p$. If*

$$\bar{y}' \notin \{z \in \mathbb{Z}_q : \exists a \in [0, p-1], |z - \frac{q}{p} \cdot (a + \frac{1}{2})| \leq \Sigma\}$$

then we have $y = y' \mod p$.

Proof. First, since $y' = \lfloor \bar{y}' \rceil_p \mod p$, there exists $\Delta \in [-\frac{q}{2p}, \frac{q}{2p})$ s.t.

$$\bar{y}' = (\frac{q}{p} \cdot y' + \Delta) \mod q$$

Also, since $\bar{y}' \notin \{z \in \mathbb{Z}_q : \exists a \in [0, p-1], |z - \frac{q}{p} \cdot (a + \frac{1}{2})| \leq \Sigma\}$, we have

$$\bar{y}' - \frac{q}{p} \cdot (y' - \frac{1}{2}) > \Sigma \quad \wedge \quad \frac{q}{p} \cdot (y' + \frac{1}{2}) - \bar{y}' > \Sigma$$

and thus we can restrict the range of Δ to be $(-(\frac{q}{2p} - \Sigma), \frac{q}{2p} - \Sigma)$. So, we have

$$\bar{y} = \bar{y}' + E = (\frac{q}{p} \cdot y' + \Delta + E) = (\frac{q}{p} \cdot y' + \Delta') \mod q$$

where $\Delta' \in (-\frac{q}{2p}, \frac{q}{2p})$. Therefore, we have

$$y = \lfloor \frac{p}{q} \cdot \bar{y} \rceil = \lfloor \frac{p}{q} \cdot (\frac{q}{p} \cdot y' + \Delta') \rceil = \lfloor y' + \frac{p}{q} \cdot \Delta' \rceil = y' \mod p$$

□

Correctness. Let $K = ((\boldsymbol{A}_i)_{i \in [0,N]}, (\boldsymbol{B}_i)_{i \in [1,k]}, \boldsymbol{s}, v, k_{\mathsf{H}}, k_{\mathsf{F}})$ be a random PRF key of PRF. We define `Col` to be the event that

$$\exists x_1, x_2 \in \{0,1\}^l : \ x_1 \neq x_2 \ \wedge \ \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x_1) = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x_2)$$

In addition, for any $u \in \{0,1\}^L$, let

$$\boldsymbol{C}_{j,\iota}^{(u)} = \mathtt{EvalPK}(\mathsf{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N) \quad \text{for } j \in [1, \varkappa], \iota \in [1, k]$$

$$\boldsymbol{D}_j^{(u)} = \mathtt{IPEvalPK}(\boldsymbol{C}_{j,1}^{(u)}, \ldots, \boldsymbol{C}_{j,k}^{(u)}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k) \quad \text{for } j \in [1, \varkappa]$$

$$\bar{y}^{(u)} = (\sum_{j=1}^{\varkappa} \boldsymbol{s}^{\mathsf{T}} \cdot \boldsymbol{D}_j^{(u)})[1] + v \mod q$$

Then we define `Borderline` to be the event that

$$\exists u \in \{0,1\}^L : \ \bar{y}^{(u)} \in \{z \in \mathbb{Z}_q : \exists a \in [0, p-1], |z - \frac{q}{p} \cdot (a + \frac{1}{2})| \leq \Sigma\}$$

We prove the correctness of PRF via arguing the following lemmas.

Lemma D.3. $\Pr[\mathtt{Col}] \leq negl(\lambda)$.

Proof. This comes from the injectivity of the explainable hash H directly. □

Lemma D.4. $\Pr[\mathtt{Borderline}] \leq negl(\lambda)$.

Proof. We define the function

$$f(u) = (\sum_{j=1}^{\varkappa} \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j^{(u)})[1] \mod q$$

Note that

$$\bar{y}^{(u)} = f(u) + v \mod q$$

where v is sampled uniformly at random and is independent of f. Then by Lemma 2.3 and the fact that $q \geq 2^{L+\omega(\log\lambda)} \cdot p \cdot (2\Sigma+1)$, we have $\Pr[\texttt{Borderline}] \leq negl(\lambda)$. $\square$

Lemma D.5. *If neither* Col *nor* Borderline *occurs, then for any polynomial-size set* $\mathcal{P} \subset \{0,1\}^l$ *and any input* $x \in \{0,1\}^l \backslash \mathcal{P}$*, we have*

$$\Pr[CK \leftarrow \texttt{Constrain}(K, \mathcal{P}) : \texttt{Eval}(K, x) = \texttt{ConstrainEval}(CK, x)] = 1$$

Proof. For any $\mathcal{P} = \{x_1, \ldots, x_{|\mathcal{P}|}\}$ of polynomial size and any $x \notin \mathcal{P}$, let

$$\begin{aligned} CK = (&(\boldsymbol{A}_i)_{i\in[0,N]}, (\boldsymbol{B}_i)_{i\in[1,k]}, v, k_{\mathsf{H}}, \\ &(\boldsymbol{a}_{0,j})_{j\in[0,N]}, (\boldsymbol{b}_{0,j})_{j\in[1,k]}, \\ &\{(\boldsymbol{a}_{i,j})_{j\in[0,N]}, (\boldsymbol{b}_{i,j})_{j\in[1,k]}, ct_i\}_{i\in[1,|\mathcal{P}|]}) \end{aligned}$$

be the output of $\texttt{Constrain}(K, \mathcal{P})$. Let $u = \mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x)$. Let $y^{(1)} = \texttt{Eval}(K, x)$ and let

$$(\boldsymbol{C}_{j,\iota})_{j\in[1,\varkappa],\iota\in[1,k]}, (\boldsymbol{D}_j)_{j\in[1,\varkappa]}, \bar{y}^{(1)}$$

be the variables used in evaluating $\texttt{Eval}(K, x)$. Also, let $y^{(2)} = \texttt{ConstrainEval}(CK, x)$ and let

$$(\boldsymbol{c}_{i,j,\iota})_{i\in[0,|\mathcal{P}|],j\in[1,\varkappa],\iota\in[1,k]}, (\boldsymbol{d}_{i,j})_{i\in[0,|\mathcal{P}|],j\in[1,\varkappa]}, \bar{y}^{(2)}$$

be variables used in evaluating $\texttt{ConstrainEval}(CK, x)$.

First, by Lemma D.1, for $i \in [1, |\mathcal{P}|]$, we have

$$|\sum_{j=1}^{\varkappa} (\boldsymbol{d}_{i,j}^\intercal - \boldsymbol{t}_i^\intercal \cdot \boldsymbol{D}_j)[1] - \beta_i \cdot \bar{r}_i| \leq \Sigma'$$

where $\beta_i = 1$ if $\mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x_i) = u$ and $\beta_i = 0$ if $\mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x_i) \neq u$. Since $x \notin \mathcal{P}$ and the event Col does not occur, we have $\mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x_i) \neq \mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x) = u$. Therefore, we have

$$|\sum_{j=1}^{\varkappa} (\boldsymbol{d}_{i,j}^\intercal - \boldsymbol{t}_i^\intercal \cdot \boldsymbol{D}_j)[1]| \leq \Sigma'$$

Moreover, similar to the proof of Lemma D.1, for $j \in [1, \varkappa]$, we have

$$\|\boldsymbol{d}_{0,j}^\intercal - \boldsymbol{t}_0^\intercal (\boldsymbol{D}_j + \sum_{\iota=1}^{k} \mathsf{C}_{u,j,\iota}(ct_0) \cdot sk_0[\iota] \cdot \boldsymbol{G})\|_\infty \leq k \cdot (m+1) \cdot (m+2)^D \cdot \lambda \cdot \sigma$$

by Lemma A.1 and Lemma A.2. Also, note that $sk_0[\iota] = 0$ for $\iota \in [1, k]$, thus we have

$$\|\boldsymbol{d}_{0,j}^{\intercal} - \boldsymbol{t}_0^{\intercal}\boldsymbol{D}_j\|_\infty \leq k \cdot (m+1) \cdot (m+2)^D \cdot \lambda \cdot \sigma$$

which implies that

$$|(\boldsymbol{d}_{0,j}^{\intercal} - \boldsymbol{t}_0^{\intercal}\boldsymbol{D}_j)[1]| \leq k \cdot (m+1) \cdot (m+2)^D \cdot \lambda \cdot \sigma \leq k \cdot m^{O(D)} \cdot \lambda \cdot \sigma$$

Therefore, we have

$$|\sum_{j=1}^{\varkappa}(\boldsymbol{d}_{0,j}^{\intercal} - \boldsymbol{t}_0^{\intercal} \cdot \boldsymbol{D}_j)[1]| \leq \sum_{j=1}^{\varkappa}|(\boldsymbol{d}_{0,j} - \boldsymbol{t}_0^{\intercal} \cdot \boldsymbol{D}_j)[1]| \leq \varkappa \cdot k \cdot m^{O(D)} \cdot \lambda \cdot \sigma \leq \varSigma'$$

Now, for $i \in [0, |\mathcal{P}|]$, let $E_i = \sum_{j=1}^{\varkappa}(\boldsymbol{d}_{i,j}^{\intercal} - \boldsymbol{t}_i^{\intercal} \cdot \boldsymbol{D}_j)[1]$, then we have $|E_i| \leq \varSigma'$ and

$$\begin{aligned}
&\bar{y}^{(2)} - \bar{y}^{(1)} \\
=&((\sum_{i=0}^{|\mathcal{P}|}\sum_{j=1}^{\varkappa}\boldsymbol{d}_{i,j})[1] + v) - ((\sum_{j=1}^{\varkappa}\boldsymbol{s}^{\intercal} \cdot \boldsymbol{D}_j)[1] + v) \\
=&(\sum_{i=0}^{|\mathcal{P}|}\sum_{j=1}^{\varkappa}\boldsymbol{d}_{i,j}^{\intercal} - \sum_{j=1}^{\varkappa}\boldsymbol{s}^{\intercal} \cdot \boldsymbol{D}_j)[1] \\
=&(\sum_{i=0}^{|\mathcal{P}|}\sum_{j=1}^{\varkappa}\boldsymbol{d}_{i,j}^{\intercal} - \sum_{i=0}^{|\mathcal{P}|}\sum_{j=1}^{\varkappa}\boldsymbol{t}_i^{\intercal} \cdot \boldsymbol{D}_j)[1] \qquad\qquad \mod q \\
=&(\sum_{i=0}^{|\mathcal{P}|}\sum_{j=1}^{\varkappa}(\boldsymbol{d}_{i,j}^{\intercal} - \boldsymbol{t}_i^{\intercal} \cdot \boldsymbol{D}_j))[1] \\
=&\sum_{i=0}^{|\mathcal{P}|}E_i
\end{aligned}$$

Let $E = \sum_{i=0}^{|\mathcal{P}|}E_i$, then we have

$$|E| \leq (|\mathcal{P}| + 1) \cdot \varSigma' \leq 2^{\omega(\log\lambda)} \cdot \varSigma' \leq \varSigma$$

and

$$\bar{y}^{(2)} - \bar{y}^{(1)} = E \mod q$$

Next, since the event `Borderline` does not occur, we have

$$\bar{y}^{(1)} \notin \{z \in \mathbb{Z}_q : \exists a \in [0, p-1], |z - \frac{q}{p} \cdot (a + \frac{1}{2})| \leq \varSigma\}$$

Then by Lemma D.2, we have $y^{(1)} = y^{(2)} \mod p$. This completes the proof of Lemma D.5. □

Correctness of PRF comes from Lemma D.3 to Lemma D.5 directly.

Pseudorandomness. Next, we prove the pseudorandomness of PRF. Note that by Theorem 4.1, it it sufficient to prove that PRF has a weak version of adaptive pseudorandomness, where the adversary is not allowed to make queries to the constrain oracle before submitting the challenge x^*.

If PRF does not have such pseudorandomness, then there exists a PPT adversary $\mathcal{A}$ that wins in the experiment with probability $\frac{1}{2}+\delta$ for some non-negligible δ. We next prove that such adversary does not exist. First, we define the following games between a challenger and the adversary $\mathcal{A}$.

- **Game 0.** This is the experiment ExpPR defined in Definition 4.2 (with a random b), where the adversary cannot query the constrain oracle in Step 2. More precisely, the challenger interacts with the adversary as follows.
  1. First, the challenger samples $b \xleftarrow{\$} \{0,1\}$ and generates
     (a) For $i \in [0, N]$: $\boldsymbol{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$.
     (b) For $i \in [1, k]$: $\boldsymbol{B}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$.
     (c) $\bar{\boldsymbol{s}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$, $\boldsymbol{s} = (1, \bar{\boldsymbol{s}}^\mathsf{T})^\mathsf{T}$.
     (d) $v \xleftarrow{\$} \mathbb{Z}_q$.
     (e) $k_\mathsf{H} \leftarrow \mathsf{H}.\mathtt{KeyGen}(1^\lambda)$.
     (f) $k_\mathsf{F} \leftarrow \mathsf{F}.\mathtt{KeyGen}(1^\lambda)$.
  2. Then it answers the evaluation oracle queries from the adversary. In particular, given an input x, it computes y as follows and returns y to the adversary.
     (a) $u = \mathsf{H}.\mathtt{Eval}(k_\mathsf{H}, x)$.
     (b) For $j \in [1, \varkappa]$:
        i. For $\iota \in [1, k]$: $\boldsymbol{C}_{j,\iota} = \mathtt{EvalPK}(\mathtt{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N)$.
        ii. $\boldsymbol{D}_j = \mathtt{IPEvalPK}(\boldsymbol{C}_{j,1}, \ldots, \boldsymbol{C}_{j,k}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k)$.
     (c) $\bar{y} = (\sum_{j=1}^{\varkappa} \boldsymbol{s}^\mathsf{T} \cdot \boldsymbol{D}_j)[1] + v \mod q$.
     (d) $y = \lfloor \bar{y} \rceil_p \mod p$.
  3. Next, after the adversary submits a challenge x^*, the challenger samples $y_1^* \xleftarrow{\$} \mathbb{Z}_p$ and computes y_0^* as follows. Then it returns y_b^* to the adversary.
     (a) $u^* = \mathsf{H}.\mathtt{Eval}(k_\mathsf{H}, x^*)$.
     (b) For $j \in [1, \varkappa]$:
        i. For $\iota \in [1, k]$: $\boldsymbol{C}^*_{j,\iota} = \mathtt{EvalPK}(\mathtt{C}_{u^*,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N)$.
        ii. $\boldsymbol{D}^*_j = \mathtt{IPEvalPK}(\boldsymbol{C}^*_{j,1}, \ldots, \boldsymbol{C}^*_{j,k}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k)$.
     (c) $\bar{y}^* = (\sum_{j=1}^{\varkappa} \boldsymbol{s}^\mathsf{T} \cdot \boldsymbol{D}^*_j)[1] + v \mod q$.
     (d) $y_0^* = \lfloor \bar{y}^* \rceil_p \mod p$.
  4. Then it answers the evaluation oracle queries and the constrain oracle queries from the adversary. The evaluation oracle queries are answered identically as in Step 2, and given a polynomial-size set $\mathcal{P}$, the challenger computes CK as follows and returns it to the adversary.

(a) Let $\mathcal{P} = \{x_1, \ldots, x_{|\mathcal{P}|}\}$.

(b) For $i \in [1, |\mathcal{P}|]$:

i. $u_i = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x_i)$.

ii. $(R_{\mathtt{K},i}, (R_{\mathtt{E},i,j})_{j\in[1,L+\varkappa]}, \bar{r}_i, \bar{\boldsymbol{t}}_i, (R_{\mathtt{G},i,j})_{j\in[0,N+k]}) = \mathsf{F}.\mathtt{Eval}(k_{\mathsf{F}}, u_i)$.

iii. $\boldsymbol{t}_i = \begin{pmatrix} 1 \\ \bar{\boldsymbol{t}}_i \end{pmatrix}$.

iv. $\boldsymbol{r}_i = \boldsymbol{G}_{1,q}^{-1}(\bar{r}_i)$.

v. $(pk_i, sk_i) = \mathsf{FHE}.\mathtt{KeyGen}(1^\lambda, 1^d; R_{\mathtt{K},i})$.

vi. $\boldsymbol{e}_{i,j} = \mathtt{GS}(R_{\mathtt{G},i,j})$ for $j \in [0, N+k]$.

vii. For $j \in [1, L]$: $ct_{i,j} = \mathsf{FHE}.\mathtt{Enc}(pk_i, u_i[j]; R_{\mathtt{E},i,j})$.

viii. For $j \in [1, \varkappa]$: $ct_{i,L+j} = \mathsf{FHE}.\mathtt{Enc}(pk_i, \boldsymbol{r}_i[j]; R_{\mathtt{E},i,L+j})$.

ix. $ct_i = (ct_{i,1} \| \ldots \| ct_{i,L+\varkappa})$.

x. $\boldsymbol{a}_{i,0}^\intercal = \boldsymbol{t}_i^\intercal \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_{i,0}^\intercal$.

xi. For $j \in [1, N]$: $\boldsymbol{a}_{i,j}^\intercal = \boldsymbol{t}_i^\intercal \cdot (\boldsymbol{A}_j + ct_i[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{i,j}^\intercal$.

xii. For $j \in [1, k]$: $\boldsymbol{b}_{i,j}^\intercal = \boldsymbol{t}_i^\intercal \cdot (\boldsymbol{B}_j + sk_i[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{i,N+j}^\intercal$.

(c) $\boldsymbol{t}_0 = \boldsymbol{s} - \sum_{i=1}^{|\mathcal{P}|} \boldsymbol{t}_i$.

(d) For $j \in [0, N+k]$, $\boldsymbol{e}_{0,j} \leftarrow \tilde{D}_\sigma^m$.

(e) $\boldsymbol{a}_{0,0}^\intercal = \boldsymbol{t}_0^\intercal \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_{0,0}^\intercal$.

(f) For $j \in [1, N]$: $\boldsymbol{a}_{0,j}^\intercal = \boldsymbol{t}_0^\intercal \cdot (\boldsymbol{A}_j + 0 \cdot \boldsymbol{G}) + \boldsymbol{e}_{0,j}^\intercal$.

(g) For $j \in [1, k]$: $\boldsymbol{b}_{0,j}^\intercal = \boldsymbol{t}_0^\intercal \cdot (\boldsymbol{B}_j + 0 \cdot \boldsymbol{G}) + \boldsymbol{e}_{0,N+j}^\intercal$.

(h) Set $CK = ((\boldsymbol{A}_i)_{i\in[0,N]}, (\boldsymbol{B}_i)_{i\in[1,k]}, v, k_{\mathsf{H}}, (\boldsymbol{a}_{0,j})_{j\in[0,N]}, (\boldsymbol{b}_{0,j})_{j\in[1,k]}, \{(\boldsymbol{a}_{i,j})_{j\in[0,N]}, (\boldsymbol{b}_{i,j})_{j\in[1,k]}, ct_i\}_{i\in[1,|\mathcal{P}|]})$.

5. Finally, after the adversary submits a bit b', the challenger outputs 1 if $b = b'$ and outputs 0 otherwise.

- **Game 1.** Let $Q = poly(\lambda)$ be the upper bound on the number of the evaluation oracle queries made by $\mathcal{A}$ in Game 0, and recall that we assume Game 0 outputs 1 with probability $1/2 + \delta$. Let $(\mathtt{VKeyGen}_{Q,\delta}, \mathtt{Verify}_{Q,\delta})$ be the algorithms used in defining the explainability of H (Definition 3.2).
  Game 1 and Game 0 are identical except that in the beginning of the game, the challenger samples $vk \leftarrow \mathtt{VKeyGen}_{Q,\delta}(1^\lambda)$ and $b^\dagger \xleftarrow{\$} \{0,1\}$. Then at Step 5 of the game, after the adversary submits a bit b', the challenger runs

$$\zeta = \mathtt{Verify}_{Q,\delta}(vk, \{x_1, \ldots, x_{Q'}\}, x^*)$$

  where $x_1, \ldots, x_{Q'}$ are the inputs submitted to the evaluation oracle at Step **2**. If $\zeta = 1$, it outputs 1 if $b' = b$ and outputs 0 if $b' \neq b$; otherwise, it outputs 1 if $b^\dagger = b$ and outputs 0 if $b^\dagger \neq b$.
- **Game 2.** This is identical to Game 1 except that at Step 3 of the game, the challenger aborts the game if $\mathtt{Verify}_{Q,\delta}(vk, \{x_1, \ldots, x_{Q'}\}, x^*) = 0$. In case the challenger aborts, it outputs 1 if $b^\dagger = b$ and outputs 0 if $b^\dagger \neq b$.

- **Game 3.** In Game 3, the challenger maintains a list $\mathcal{L}$, which is initialized as an empty list. Then for each evaluation oracle query x, it puts $(x, \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x))$ to the list $\mathcal{L}$. Also, for each constrain oracle query $\mathcal{P} = \{x_1, \ldots, x_{|\mathcal{P}|}\}$ and for $i \in [1, |\mathcal{P}|]$, it puts $(x_i, \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x_i))$ to $\mathcal{L}$. Besides, in Step 3, it puts (x^*, u^*) to $\mathcal{L}$. Finally, at Step 5, if there exists $(x, u), (x', u') \in \mathcal{L}$ s.t.

$$(x \neq x' \ \wedge u = u') \vee (x = x' \ \wedge u \neq u') \tag{15}$$

then the challenger outputs 1 if $b^\dagger = b$ and outputs 0 if $b^\dagger \neq b$.
- **Game 4.** This is identical to Game 3 except that the challenger uses the simulator of H. In more detail, let $\mathtt{SIM}$ be the stateful simulator of H as defined in Definition 3.2, then the challenger modifies its behaviors at each step as follows:
  1. At step 1, the challenger does not generate k_{H}. In addition, it samples $u^* \xleftarrow{\$} \{0,1\}^L$ and invokes $\mathtt{SIM}(vk, u^*)$.
  2. At Step 2, for each evaluation oracle query x, it computes $u \leftarrow \mathtt{SIM}(x)$ instead of computing it as $u = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x)$, where u is used to answer the evaluation oracle query and is recorded in $\mathcal{L}$.
  3. At Step 3, if $\mathtt{Verify}_{Q,\delta}(vk, \{x_1, \ldots, x_{Q'}\}, x^*) = 1$, it computes $k_{\mathsf{H}} \leftarrow \mathtt{SIM}(x^*)$. Note that, the challenger has sampled u^* in the beginning.
  4. It proceeds identically as in Game 3 for Step 4 and Step 5.
- **Game 5.** This is identical to Game 4 except that the challenger uses a random function f instead of the pseudorandom function $\mathsf{F}.\mathtt{Eval}(k_{\mathsf{F}}, \cdot)$ when answering the constrain oracle queries.
- **Game 6.** This is identical to Game 5 except that when answering the constrain oracle queries, the challenger samples $(R_{\mathtt{K}}^{(x)}, (R_{\mathtt{E},j}^{(x)})_{j\in[1,L+\varkappa]}, \bar{r}^{(x)}, \bar{\boldsymbol{t}}^{(x)}, (R_{\mathtt{G},j}^{(x)})_{j\in[0,N+k]})$ from the output space of F instead of computing it as

$$(R_{\mathtt{K}}^{(x)}, (R_{\mathtt{E},j}^{(x)})_{j\in[1,L+\varkappa]}, \bar{r}^{(x)}, \bar{\boldsymbol{t}}^{(x)}, (R_{\mathtt{G},j}^{(x)})_{j\in[0,N+k]}) = f(\mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x))$$

  for each $x \in \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{Q_c}$, where $\mathcal{P}_1, \ldots, \mathcal{P}_{Q_c}$ are the sets submitted to the constrain oracle.
- **Game 7.** This is identical to Game 6 except that at Step 1 of the game, the challenger generates
  1. $\bar{\boldsymbol{t}}^* \xleftarrow{\$} \mathbb{Z}_q^{n-1}$, $\boldsymbol{t}^* = (1, \bar{\boldsymbol{t}}^{*\mathsf{T}})^{\mathsf{T}}$.
  2. $\bar{r}^* \xleftarrow{\$} \mathbb{Z}_q$, $\boldsymbol{r}^* = \boldsymbol{G}_{1,q}^{-1}(\bar{r}^*)$.
  3. $(pk^*, sk^*) \leftarrow \mathsf{FHE}.\mathtt{KeyGen}(1^\lambda, 1^d)$.
  4. $\boldsymbol{e}_j^* \leftarrow \tilde{D}_\sigma^m$ for $j \in [0, N+k]$.
  5. For $j \in [1, L]$: $ct_j^* \leftarrow \mathsf{FHE}.\mathtt{Enc}(pk^*, u^*[j])$.
  6. For $j \in [1, \varkappa]$: $ct_{L+j}^* \leftarrow \mathsf{FHE}.\mathtt{Enc}(pk^*, \boldsymbol{r}^*[j])$.
  7. $ct^* = (ct_1^* \| \ldots \| ct_{L+\varkappa}^*)$.
  8. $\boldsymbol{a}_0^{*\mathsf{T}} = \boldsymbol{t}^{*\mathsf{T}} \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_0^{*\mathsf{T}}$.

9. For $j \in [1, N]$: $\boldsymbol{a}_j^{*\mathsf{T}} = \boldsymbol{t}^{*\mathsf{T}} \cdot (\boldsymbol{A}_j + ct^*[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_j^{*\mathsf{T}}$.
10. For $j \in [1, k]$: $\boldsymbol{b}_j^{*\mathsf{T}} = \boldsymbol{t}^{*\mathsf{T}} \cdot (\boldsymbol{B}_j + sk^*[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{N+j}^{*\mathsf{T}}$.
11. $\tilde{\boldsymbol{s}} = \boldsymbol{s} - \boldsymbol{t}^* \mod q$.

Then at Step 4, for each constrain oracle query $\mathcal{P} = \{x_1, \ldots, x_{|\mathcal{P}|}\}$, let $x_{i^*} = x^*$[23], then the challenger sets

$$ct_{i^*} = ct^*, \quad \boldsymbol{a}_{i^*,j} = \boldsymbol{a}_j^* \text{ for } j \in [0, N], \quad \boldsymbol{b}_{i^*,j} = \boldsymbol{b}_j^* \text{ for } j \in [1, k]$$

and computes

$$\boldsymbol{t}_0 = \tilde{\boldsymbol{s}} - \sum_{i \in [1, |\mathcal{P}|] \setminus \{i^*\}} \boldsymbol{t}_i$$

- **Game 8.** This is identical to Game 7 except that the challenger changes the way to answer the evaluation oracle query. In particular, given an evaluation oracle query x, the challenger first computes

$$\begin{cases} u \leftarrow \mathtt{SIM}(x) & \text{At Step 2} \\ u = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x) & \text{At Step 4} \end{cases}$$

Then it computes y as follows and returns y to the adversary.
1. For $j \in [1, \varkappa]$:
   (a) For $\iota \in [1, k]$: $\boldsymbol{C}_{j,\iota} = \mathtt{EvalPK}(\mathtt{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N)$.
   (b) $\boldsymbol{D}_j = \mathtt{IPEvalPK}(\boldsymbol{C}_{j,1}, \ldots, \boldsymbol{C}_{j,k}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k)$.
2. For $j \in [1, \varkappa]$:
   (a) For $\iota \in [1, k]$:
      i. $\widetilde{ct}_{j,\iota} = \mathtt{C}_{u,j,\iota}(ct^*)$.
      ii. $\boldsymbol{c}_{j,\iota} = \mathtt{EvalCT}(\mathtt{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{a}_0^*, \ldots, \boldsymbol{a}_N^*, ct^*)$.
   (b) $\boldsymbol{d}_j = \mathtt{IPEvalCT}(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_k, \boldsymbol{c}_{j,1}, \ldots, \boldsymbol{c}_{j,k}, \boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_k^*, \widetilde{ct}_{j,1}, \ldots, \widetilde{ct}_{j,k})$.
3. $\bar{y} = (\sum_{j=1}^{\varkappa} (\tilde{\boldsymbol{s}}^{\mathsf{T}} \cdot \boldsymbol{D}_j + \boldsymbol{d}_j))[1] + v \mod q$.
4. $y = \lfloor \bar{y} \rceil_p \mod p$.

- **Game 9.** This is identical to Game 8 except that the challenger changes the way to generate y_0^*. In particular, it computes y_0^* as follows:
1. For $j \in [1, \varkappa]$:
   (a) For $\iota \in [1, k]$: $\boldsymbol{C}_{j,\iota}^* = \mathtt{EvalPK}(\mathtt{C}_{u^*,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N)$.
   (b) $\boldsymbol{D}_j^* = \mathtt{IPEvalPK}(\boldsymbol{C}_{j,1}^*, \ldots, \boldsymbol{C}_{j,k}^*, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k)$.
2. For $j \in [1, \varkappa]$:
   (a) For $\iota \in [1, k]$:
      i. $\widetilde{ct}_{j,\iota}^* = \mathtt{C}_{u^*,j,\iota}(ct^*)$.
      ii. $\boldsymbol{c}_{j,\iota}^* = \mathtt{EvalCT}(\mathtt{C}_{u^*,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{a}_0^*, \ldots, \boldsymbol{a}_N^*, ct^*)$.
   (b) $\boldsymbol{d}_j^* = \mathtt{IPEvalCT}(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_k, \boldsymbol{c}_{j,1}^*, \ldots, \boldsymbol{c}_{j,k}^*, \boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_k^*, \widetilde{ct}_{j,1}^*, \ldots, \widetilde{ct}_{j,k}^*)$.
3. $\bar{y}^* = (\sum_{j=1}^{\varkappa} (\tilde{\boldsymbol{s}}^{\mathsf{T}} \cdot \boldsymbol{D}_j^* + \boldsymbol{d}_j^*))[1] + v - \bar{r}^* \mod q$.

[23] Recall that such i^* always exists as required in Equation (2).

4. $y_0^* = \lfloor \bar{y}^* \rceil_p \mod p$.

- **Game 10.** This is identical to Game 9 except that the challenger samples $\bar{\tilde{\boldsymbol{s}}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$ and sets $\tilde{\boldsymbol{s}} = (0, \bar{\tilde{\boldsymbol{s}}}^{\intercal})^{\intercal}$ instead of computing it from $\boldsymbol{s}$ and $\boldsymbol{t}^*$.
- **Game 11.** This is identical to Game 10 except that the challenger samples

$$\boldsymbol{a}_j^* \xleftarrow{\$} \mathbb{Z}_q^m \text{ for } j \in [0, N], \quad \boldsymbol{b}_j^* \xleftarrow{\$} \mathbb{Z}_q^m \text{ for } j \in [1, k]$$

- **Game 12.** This is identical to Game 11 except that the challenger computes

$$ct_j^* \leftarrow \mathsf{FHE}.\mathtt{Enc}(pk^*, 0)$$

for $j \in [1, L + \varkappa]$ and sets $ct^* = (ct_1^* \| \dots \| ct_{L+\varkappa}^*)$.

Let $\mathcal{E}_i$ be the output of Game i for $i \in [0, 12]$. We have the following lemmas.

Lemma D.6. *If δ is non-negligible, then $|\Pr[\mathcal{E}_1 = 1] - \frac{1}{2}|$ is also non-negligible.*

Proof. First, as Q is the upper bound on the number of the evaluation oracle queries made by $\mathcal{A}$, we have $Q' \leq Q$. Also, by Equation (2) required in the definition of pseudorandomness, for all $i \in [1, Q']$, we have $x^* \neq x_i$. Then by the "Abort Probability" requirement in the explainability property of H, there exists $\mathit{\Gamma}_{min}$, $\mathit{\Gamma}_{max}$ that

$$\mathit{\Gamma}_{min} \leq \Pr[\mathtt{Verify}_{Q,\delta}(vk, \{x_1, \dots, x_{Q'}\}, x^*) = 1] \leq \mathit{\Gamma}_{max}$$

and

$$\mathit{\Gamma}_{min} \cdot \delta - (\mathit{\Gamma}_{max} - \mathit{\Gamma}_{min})$$

is a non-negligible positive real value.

Next, note that $\delta = |\Pr[\mathcal{E}_0 = 1] - \frac{1}{2}| = |\Pr[b = b'] - \frac{1}{2}|$, then by Lemma A.6, we have

$$|\Pr[\mathcal{E}_1 = 1] - \frac{1}{2}| \geq \mathit{\Gamma}_{min} \cdot \delta - \frac{\mathit{\Gamma}_{max} - \mathit{\Gamma}_{min}}{2} \geq \mathit{\Gamma}_{min} \cdot \delta - (\mathit{\Gamma}_{max} - \mathit{\Gamma}_{min})$$

which is non-negligible. □

Lemma D.7. $|\Pr[\mathcal{E}_1 = 1] - \Pr[\mathcal{E}_2 = 1]| = 0$.

Proof. Game 2 and Game 1 are identically proceeded if $\mathtt{Verify}_{Q,\delta}(vk, \{x_1, \dots, x_{Q'}\}, x^*) = 1$. In case that $\mathtt{Verify}_{Q,\delta}(vk, \{x_1, \dots, x_{Q'}\}, x^*) = 0$, both games will output 1 with probability 1/2. Thus, the output distributions of the two games are identical. □

Lemma D.8. $|\Pr[\mathcal{E}_2 = 1] - \Pr[\mathcal{E}_3 = 1]| \leq negl(\lambda)$.

Proof. Game 2 and Game 3 are identical unless Equation (15) is satisfied. First, for all items $(x, u) \in \mathcal{L}$ (including (x^*, u^*)), we have $u = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x)$. Since $\mathsf{H}.\mathtt{Eval}$ is a deterministic algorithm, for any $(x, u), (x', u') \in \mathcal{L}$ we always have $u = u'$ if $x = x'$. Besides, by the injectivity of H, the probability that there exists distinct $x, x' \in \{0, 1\}^l$ s.t. $\mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x) = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x')$ is negligible. That is, the probability that there exists $(x, u), (x', u') \in \mathcal{L}$ s.t. $u = u'$ but $x \neq x'$ is also negligible. Thus, the probability that Equation (15) is satisfied is negligible, and indistinguishability between Game 2 and Game 3 follows. □

Lemma D.9. $|\Pr[\mathcal{E}_3 = 1] - \Pr[\mathcal{E}_4 = 1]| \leq negl(\lambda)$.

Proof. This comes from the indistinguishability requirement in the explainability property of H by a direct reduction. $\square$

Lemma D.10. $|\Pr[\mathcal{E}_4 = 1] - \Pr[\mathcal{E}_5 = 1]| \leq negl(\lambda)$.

Proof. This comes from the pseudorandomness of F by a direct reduction. $\square$

Lemma D.11. $|\Pr[\mathcal{E}_5 = 1] - \Pr[\mathcal{E}_6 = 1]| = 0$.

Proof. The adversary's views in Game 5 and Game 6 are identical unless there exists $x, x' \in \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{Q_c}$ s.t. $x \neq x'$ and $\mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x) = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x')$, and in this case, since both $(x, \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x))$ and $(x', \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x'))$ are in the list $\mathcal{L}$, the games will output 1 with probability 1/2. Therefore, the output distributions of the two games are identical. $\square$

Lemma D.12. $|\Pr[\mathcal{E}_6 = 1] - \Pr[\mathcal{E}_7 = 1]| = 0$.

Proof. In Game 7, the challenger precomputes parts of the constrained keys. The adversary's views in Game 6 and Game 7 are identical unless $u^* \neq \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x^*)$, and in this case, since both (x^*, u^*) and $(x^*, \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x^*))$ are in the list $\mathcal{L}$, the games will output 1 with probability 1/2. Therefore, the output distributions of the two games are identical. $\square$

Lemma D.13. $|\Pr[\mathcal{E}_7 = 1] - \Pr[\mathcal{E}_8 = 1]| \leq negl(\lambda)$.

Proof. Let $u, \boldsymbol{C}_{j,\iota}, \boldsymbol{D}_j, \boldsymbol{c}_{j,\iota}, \boldsymbol{d}_j, \bar{y}$ be the variables used in computing the response y to an evaluation oracle query x in Game 8. Then by Lemma D.1, we have

$$|\sum_{j=1}^{\varkappa}(\boldsymbol{d}_j^\intercal - \boldsymbol{t}^{*\intercal} \cdot \boldsymbol{D}_j)[1] - \beta \cdot \bar{r}^*| \leq \Sigma'$$

where $\beta = 1$ if $u = u^*$ and $\beta = 0$ if $u \neq u^*$. Next, assuming that $u \neq u^*$, we have

$$|\sum_{j=1}^{\varkappa}(\boldsymbol{d}_j^\intercal - \boldsymbol{t}^{*\intercal} \cdot \boldsymbol{D}_j)[1]| \leq \Sigma'$$

Let $E = \sum_{j=1}^{\varkappa}(\boldsymbol{d}_j^\intercal - \boldsymbol{t}^{*\intercal}\boldsymbol{D}_j)[1]$, then we have

$$|E| \leq \Sigma' \leq \Sigma$$

and

$$\begin{aligned}\bar{y} =& (\sum_{j=1}^{\varkappa}(\tilde{\boldsymbol{s}}^\intercal \cdot \boldsymbol{D}_j + \boldsymbol{d}_j^\intercal))[1] + v \\ =& (\sum_{j=1}^{\varkappa}((\boldsymbol{s} - \boldsymbol{t}^*)^\intercal \cdot \boldsymbol{D}_j + \boldsymbol{d}_j^\intercal))[1] + v \mod q \\ =& (\sum_{j=1}^{\varkappa} \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j)[1] + E + v\end{aligned}$$

Now, let $\bar{y}'$ be the variable used in computing the response y' to the evaluation oracle query x in Game 7. Note that

$$\bar{y}' = (\sum_{j=1}^{\varkappa} \boldsymbol{s}^{\intercal} \cdot \boldsymbol{D}_j)[1] + v \mod q$$

Thus, we have

$$\bar{y} - \bar{y}' = E \mod q$$

Also, similar to the proof of Lemma D.4, we have

$$\Pr[\bar{y}' \in \{z \in \mathbb{Z}_q : \exists a \in [0, p-1], |z - \frac{q}{p} \cdot (a + \frac{1}{2})| \leq \varSigma\}] \leq negl(\lambda)$$

from Lemma 2.3. Then by Lemma D.2, we have $\Pr[y \neq y' \mod p] \leq negl(\lambda)$. That is, each evaluation oracle query is answered identically in Game 7 and Game 8 with all but negligible probability if $u \neq u^*$.

Next, as required by Equation (2), we have $x^* \neq x$. Then, if $u = u^*$, the games will output 1 with probability 1/2 if it does not abort at Step 3. In addition, the games still outputs 1 with probability 1/2 if it aborts at Step 3.

To summarize, the adversary's view in these two games are close before it submits an evaluation oracle query making $u \neq u^*$, and even if such query is made, the games' outputs are identical. Thus, the difference between the output distributions of the two games is negligible. □

Lemma D.14. $|\Pr[\mathcal{E}_8 = 1] - \Pr[\mathcal{E}_9 = 1]| \leq negl(\lambda)$.

Proof. Let $u^*, \boldsymbol{C}^*_{j,\iota}, \boldsymbol{D}^*_j, \boldsymbol{c}^*_{j,\iota}, \boldsymbol{d}^*_j, \bar{y}^*$ be the variables used in computing the challenge y^*_0 in Game 9. First, by Lemma D.1, we have

$$|\sum_{j=1}^{\varkappa} (\boldsymbol{d}^{*\intercal}_j - \boldsymbol{t}^{*\intercal} \cdot \boldsymbol{D}^*_j)[1] - \bar{r}^*| \leq \varSigma'$$

Let $E = \sum_{j=1}^{\varkappa} (\boldsymbol{d}^{*\intercal}_j - \boldsymbol{t}^{*\intercal} \boldsymbol{D}^*_j)[1] - \bar{r}^*$, then we have

$$|E| \leq \varSigma' \leq \varSigma$$

and

$$\begin{aligned}\bar{y}^* =& (\sum_{j=1}^{\varkappa} (\tilde{\boldsymbol{s}}^{\intercal} \cdot \boldsymbol{D}^*_j + \boldsymbol{d}^{*\intercal}_j))[1] + v - \bar{r}^* \\ =& (\sum_{j=1}^{\varkappa} ((\boldsymbol{s} - \boldsymbol{t}^*)^{\intercal} \cdot \boldsymbol{D}^*_j + \boldsymbol{d}^{*\intercal}_j))[1] + v - \bar{r}^* \mod q \\ =& (\sum_{j=1}^{\varkappa} \boldsymbol{s}^{\intercal} \cdot \boldsymbol{D}^*_j)[1] + E + v\end{aligned}$$

Now, let $\bar{y}^{*\prime}$ be the variable used in computing the challenge $y_0^{*\prime}$ in Game 8. Note that

$$\bar{y}^{*\prime} = (\sum_{j=1}^{\varkappa} \boldsymbol{s}^{\intercal} \cdot \boldsymbol{D}_j^*)[1] + v \mod q$$

Thus, we have

$$\bar{y}^* - \bar{y}^{*\prime} = E \mod q$$

Also, similar to the proof of Lemma D.4, we have

$$\Pr[\bar{y}^{*\prime} \in \{z \in \mathbb{Z}_q : \exists a \in [0, p-1], |z - \frac{q}{p} \cdot (a + \frac{1}{2})| \leq \Sigma\}] \leq negl(\lambda)$$

from Lemma 2.3. Then by Lemma D.2, we have $\Pr[y_0^* \neq y_0^{*\prime} \mod p] \leq negl(\lambda)$.

Since with all but negligible probability, the view of the adversary in Game 8 and Game 9 are identical, the difference between the output distributions of the two games is negligible. □

Lemma D.15. $|\Pr[\mathcal{E}_9 = 1] - \Pr[\mathcal{E}_{10} = 1]| = 0$.

Proof. Game 9 and Game 10 only differ in the way for generating $\tilde{\boldsymbol{s}}$. In Game 9, $\tilde{\boldsymbol{s}} = \boldsymbol{s} - \boldsymbol{t}^* = (0, (\bar{\boldsymbol{s}} - \bar{\boldsymbol{t}}^*)^{\intercal})^{\intercal} \mod q$, and in Game 10, the challenger samples $\bar{\tilde{\boldsymbol{s}}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$ and sets $\tilde{\boldsymbol{s}} = (0, \bar{\tilde{\boldsymbol{s}}}^{\intercal})^{\intercal}$. Note that, in Game 9, $\bar{\boldsymbol{s}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$ and is only used for generating $\tilde{\boldsymbol{s}}$. Thus, the adversary's views are identical in these two games. □

Lemma D.16. $|\Pr[\mathcal{E}_{10} = 1] - \Pr[\mathcal{E}_{11} = 1]| \leq negl(\lambda)$.

Proof. Indistinguishability between Game 10 and Game 11 comes from the hardness of $\mathrm{LWE}_{n-1,q,\tilde{D}_\sigma}$ by viewing $\boldsymbol{A} = (\boldsymbol{A}_0', \ldots, \boldsymbol{A}_N', \boldsymbol{B}_1', \ldots, \boldsymbol{B}_k')^{\intercal}$ as the public matrix, $\bar{\boldsymbol{t}}^*$ as the secret vector, and $(\boldsymbol{e}_0^{*\intercal}, \ldots, \boldsymbol{e}_{N+k}^{*\intercal})^{\intercal}$ as the error vector, where $\boldsymbol{A}_0'$ is the last $n-1$ rows of $(\boldsymbol{A}_0 + \boldsymbol{G})$, for $j \in [1, N]$, $\boldsymbol{A}_j'$ is the last $n-1$ rows of $(\boldsymbol{A}_j + ct^*[j] \cdot \boldsymbol{G})$, and for $j \in [1, k]$, $\boldsymbol{B}_j'$ is the last $n-1$ rows of $(\boldsymbol{B}_j + sk^*[j] \cdot \boldsymbol{G})$. Note that $\boldsymbol{A}$ is a random matrix in $\mathbb{Z}_q^{m \cdot (N+1+k) \times (n-1)}$ since $\boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k$ are all random matrices. □

Lemma D.17. $|\Pr[\mathcal{E}_{11} = 1] - \Pr[\mathcal{E}_{12} = 1]| \leq negl(\lambda)$.

Proof. In Game 11, sk^* is hidden from the adversary's view, thus, indistinguishability between Game 11 and Game 12 comes from the security of FHE by a direct reduction. □

Lemma D.18. $|\Pr[\mathcal{E}_{12} = 1] - \frac{1}{2}| = 0$.

Proof. In Game 12, $\bar{r}^*$ is sampled uniformly at random and is only used for generating $\bar{y}^*$. Thus, we have $\bar{y}^*$ uniform in $\mathbb{Z}_q$. In addition, since q is a multiple of p, $y_0^* = \lfloor \bar{y}^* \rceil_p \mod p$ is also uniform in $\mathbb{Z}_p$. As both y_0^* and y_1^* are uniform in $\mathbb{Z}_p$, the adversary's views are identical in cases that $b = 0$ and that $b = 1$. Therefore, we always have $\Pr[b' = b] = \frac{1}{2}$ and $\Pr[b^\dagger = b] = \frac{1}{2}$, and this implies that Game 12 will output 1 with probability 1/2. □

From Lemma D.6 to Lemma D.17, $|\Pr[\mathcal{E}_{12}=1]-\frac{1}{2}|$ is non-negligible if the adversary's advantage δ in Game 0 is non-negligible. This contradicts Lemma D.18. Thus, such adversary does not exist and pseudorandomness of PRF follows.

Privacy. Next, we prove the privacy of PRF. We first define the following games between a challenger and a PPT adversary $\mathcal{A}$. Here, we use $\tau = poly(\lambda)$ to denote the upper bound on the size of the puncture sets submitted by $\mathcal{A}$.

- **Game 0.** This is the experiment $\texttt{ExpPriv}_{\mathcal{A},0}$ defined in Definition 4.3.
  Let $\mathcal{P}_0^* = \{x_1^{(0)},\dots,x_P^{(0)}\}$ and $\mathcal{P}_1^* = \{x_1^{(1)},\dots,x_P^{(1)}\}$ be the puncture sets submitted by the adversary, where $P=|\mathcal{P}_0^*|=|\mathcal{P}_1^*|$. We can sort elements in $\mathcal{P}_0^*$ and $\mathcal{P}_1^*$ in arbitrary order and this will not change the view of the adversary since the variables $((\boldsymbol{a}_{i,j})_{j\in[0,N]},(\boldsymbol{b}_{i,j})_{j\in[1,k]},ct_i)$ in CK, which are related to $x_i^{(0)}$ or $x_i^{(1)}$, are put into an unordered set. Concretely, let $\hat{P} = |\mathcal{P}_0^*\cap\mathcal{P}_1^*|$ and let $\check{P}=P-\hat{P}$, we sort elements in $\mathcal{P}_0^*$ and $\mathcal{P}_1^*$ as follows:
  1. For $b\in\{0,1\}$, $x_1^{(b)},\dots,x_{\check{P}}^{(b)}\in\mathcal{P}_b^*-(\mathcal{P}_0^*\cap\mathcal{P}_1^*)$ and they are sorted in lexicographic order.
  2. For $b\in\{0,1\}$, $x_{\check{P}+1}^{(b)},\dots,x_P^{(b)}\in(\mathcal{P}_0^*\cap\mathcal{P}_1^*)$ and they are sorted in lexicographic order.

  Note that for $i\in[\check{P}+1,P]$, $x_i^{(0)}=x_i^{(1)}$.
- **Game $\mathbf{H}_h$.** For $h\in[0,\tau]$, Game H_h is identical to Game 0 except that, the challenger returns $ck^*\leftarrow\texttt{Constrain}(k,\mathcal{P}^*)$ to the adversary after the adversary submits the puncture sets, where

$$\mathcal{P}^*=\{x_i^{(1)}\}_{i\in[1,\min(h,P)]}\cup\{x_i^{(0)}\}_{i\in[h+1,P]}$$

  Remark D.1. Note that for any $i\in[1,\min(h,P)], j\in[h+1,P]$, we always have $x_i^{(1)}\neq x_j^{(0)}$. This is because if $i\le\check{P}$, then we have $x_i^{(1)}\notin\mathcal{P}_0^*$ and thus $x_i^{(1)}\neq x_j^{(0)}$; and if $i>\check{P}$, then we have $x_i^{(1)}=x_i^{(0)}<x_j^{(0)}$. Therefore, we still have $|\mathcal{P}^*|=P$.

- **Game 1.** This is the experiment $\texttt{ExpPriv}_{\mathcal{A},1}$ defined in Definition 4.3.

It is obvious that Game H_0 is identical to Game 0 and Game H_τ is identical to Game 1. Thus privacy of PRF (i.e., indistinguishability between Game 0 and Game 1) comes from the following lemma by a standard hybrid argument.

Lemma D.19. *For $h\in[0,\tau]$, let $\mathcal{H}_h$ be the output of Game H_h. Then we have $|\Pr[\mathcal{H}_{h-1}=1]-\Pr[\mathcal{H}_h=1]|\le negl(\lambda)$ for $h\in[1,\tau]$.*

Proof. First, for any $h\in[0,\tau]$, we define the experiment $\texttt{Exp}_{h,\mathcal{A}}$ as follows:

- In the beginning, the challenger samples a bit $b\xleftarrow{\$}\{0,1\}$;
- Then it interacts with the adversary $\mathcal{A}$ as in Game H_{h-1} if $b=0$ and it interacts with $\mathcal{A}$ as in Game H_h if $b=1$;

- Finally, after the adversary $\mathcal{A}$ returns a bit b', the challenger outputs 1 if $b = b'$ and it outputs 0 if $b \neq b'$.

Then we have

$$\begin{aligned}
&|\Pr[\texttt{Exp}_{h,\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}| \\
=&|\frac{1}{2} \cdot (1 - \Pr[\mathcal{H}_{h-1} = 1]) + \frac{1}{2} \cdot \Pr[\mathcal{H}_h = 1] - \frac{1}{2}| \\
=&\frac{1}{2} \cdot |\Pr[\mathcal{H}_h = 1] - \Pr[\mathcal{H}_{h-1} = 1]|
\end{aligned}$$

Thus, it is sufficient to prove that for any $h \in [1, \tau]$,

$$|\Pr[\texttt{Exp}_{h,\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}| \leq negl(\lambda) \tag{16}$$

Now, if Equation (16) is not satisfied, then there exists a non-negligible probability that $\delta = |\Pr[\texttt{Exp}_{h,\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}|$. Next, we prove that this does not occur. We first define the following hybrids between a challenger and the adversary $\mathcal{A}$.

- **Hybrid 0.** This is the experiment $\texttt{Exp}_{h,\mathcal{A}}$. More precisely, the challenger interacts with the adversary as follows.
  1. First, the challenger samples $b \xleftarrow{\$} \{0, 1\}$ and generates
     (a) For $i \in [0, N]$: $\boldsymbol{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$.
     (b) For $i \in [1, k]$: $\boldsymbol{B}_i \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$.
     (c) $\bar{\boldsymbol{s}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$, $\boldsymbol{s} = (1, \bar{\boldsymbol{s}}^\intercal)^\intercal$.
     (d) $v \xleftarrow{\$} \mathbb{Z}_q$.
     (e) $k_{\mathsf{H}} \leftarrow \mathsf{H}.\texttt{KeyGen}(1^\lambda)$.
     (f) $k_{\mathsf{F}} \leftarrow \mathsf{F}.\texttt{KeyGen}(1^\lambda)$.
  2. Then it answers the evaluation oracle queries from the adversary. In particular, given an input x, it computes y as follows and returns y to the adversary.
     (a) $u = \mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x)$.
     (b) For $j \in [1, \varkappa]$:
        i. For $\iota \in [1, k]$: $\boldsymbol{C}_{j,\iota} = \texttt{EvalPK}(\texttt{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N)$.
        ii. $\boldsymbol{D}_j = \texttt{IPEvalPK}(\boldsymbol{C}_{j,1}, \ldots, \boldsymbol{C}_{j,k}, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k)$.
     (c) $\bar{y} = (\sum_{j=1}^{\varkappa} \boldsymbol{s}^\intercal \cdot \boldsymbol{D}_j)[1] + v \mod q$.
     (d) $y = \lfloor \bar{y} \rceil_p \mod p$.
  3. Next, the adversary will submit two sets $\mathcal{P}_0^* = \{x_1^{(0)}, \ldots, x_P^{(0)}\}$ and $\mathcal{P}_1^* = \{x_1^{(1)}, \ldots, x_P^{(1)}\}$, where $P = |\mathcal{P}_0^*| = |\mathcal{P}_1^*|$. Let $\hat{P} = |\mathcal{P}_0^* \cap \mathcal{P}_1^*|$ and let $\check{P} = P - \hat{P}$. Then, if $h - 1 \geq \check{P}$, the challenger sets

$$x_i^* = x_i^{(1)} \quad \text{for } i \in [1, P]$$

and otherwise (i.e., $h-1 < \check{P}$), it sets

$$x_i^* = \begin{cases} x_i^{(1)} & \text{if } i \in [1, h-1] \\ x_h^{(b)} & \text{if } i = h \\ x_i^{(0)} & \text{if } i \in [h+1, P] \end{cases}$$

Then it computes the constrained key CK for $\{x_1^*, \ldots, x_P^*\}$ as follows and returns it to the adversary.[24]

(a) For $i \in [1, P]$:
   i. $u_i = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x_i^*)$.
   ii. $(R_{\mathtt{K},i}, (R_{\mathtt{E},i,j})_{j\in[1,L+\varkappa]}, \bar{r}_i, \bar{\boldsymbol{t}}_i, (R_{\mathtt{G},i,j})_{j\in[0,N+k]}) = \mathsf{F}.\mathtt{Eval}(k_{\mathsf{F}}, u_i)$.
   iii. $\boldsymbol{t}_i = \begin{pmatrix} 1 \\ \bar{\boldsymbol{t}}_i \end{pmatrix}$.
   iv. $\boldsymbol{r}_i = \boldsymbol{G}_{1,q}^{-1}(\bar{r}_i)$.
   v. $(pk_i, sk_i) = \mathsf{FHE}.\mathtt{KeyGen}(1^\lambda, 1^d; R_{\mathtt{K},i})$.
   vi. $\boldsymbol{e}_{i,j} = \mathtt{GS}(R_{\mathtt{G},i,j})$ for $j \in [0, N+k]$.
   vii. For $j \in [1, L]$: $ct_{i,j} = \mathsf{FHE}.\mathtt{Enc}(pk_i, u_i[j]; R_{\mathtt{E},i,j})$.
   viii. For $j \in [1, \varkappa]$: $ct_{i,L+j} = \mathsf{FHE}.\mathtt{Enc}(pk_i, \boldsymbol{r}_i[j]; R_{\mathtt{E},i,L+j})$.
   ix. $ct_i = (ct_{i,1} \| \ldots \| ct_{i,L+\varkappa})$.
   x. $\boldsymbol{a}_{i,0}^{\mathsf{T}} = \boldsymbol{t}_i^{\mathsf{T}} \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_{i,0}^{\mathsf{T}}$.
   xi. For $j \in [1, N]$: $\boldsymbol{a}_{i,j}^{\mathsf{T}} = \boldsymbol{t}_i^{\mathsf{T}} \cdot (\boldsymbol{A}_j + ct_i[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{i,j}^{\mathsf{T}}$.
   xii. For $j \in [1, k]$: $\boldsymbol{b}_{i,j}^{\mathsf{T}} = \boldsymbol{t}_i^{\mathsf{T}} \cdot (\boldsymbol{B}_j + sk_i[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{i,N+j}^{\mathsf{T}}$.

(b) $\boldsymbol{t}_0 = \boldsymbol{s} - \sum_{i=1}^{P} \boldsymbol{t}_i$.
(c) For $j \in [0, N+k]$, $\boldsymbol{e}_{0,j} \leftarrow \tilde{D}_\sigma^m$.
(d) $\boldsymbol{a}_{0,0}^{\mathsf{T}} = \boldsymbol{t}_0^{\mathsf{T}} \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_{0,0}^{\mathsf{T}}$.
(e) For $j \in [1, N]$: $\boldsymbol{a}_{0,j}^{\mathsf{T}} = \boldsymbol{t}_0^{\mathsf{T}} \cdot (\boldsymbol{A}_j + 0 \cdot \boldsymbol{G}) + \boldsymbol{e}_{0,j}^{\mathsf{T}}$.
(f) For $j \in [1, k]$: $\boldsymbol{b}_{0,j}^{\mathsf{T}} = \boldsymbol{t}_0^{\mathsf{T}} \cdot (\boldsymbol{B}_j + 0 \cdot \boldsymbol{G}) + \boldsymbol{e}_{0,N+j}^{\mathsf{T}}$.
(g) Set $CK = ((\boldsymbol{A}_i)_{i\in[0,N]}, (\boldsymbol{B}_i)_{i\in[1,k]}, v, k_{\mathsf{H}}, (\boldsymbol{a}_{0,j})_{j\in[0,N]}, (\boldsymbol{b}_{0,j})_{j\in[1,k]}, \{(\boldsymbol{a}_{i,j})_{j\in[0,N]}, (\boldsymbol{b}_{i,j})_{j\in[1,k]}, ct_i\}_{i\in[1,P]})$.

4. Then it answers the evaluation oracle queries as in Step 2.
5. Finally, after the adversary submits a bit b', the challenger outputs 1 if $b = b'$ and it outputs 0 otherwise.

- **Hybrid 1.** Let $Q = poly(\lambda)$ be the upper bound on the number of the evaluation oracle queries made by the adversary $\mathcal{A}$ in Hybrid 0. Let ($\mathtt{VKeyGen}_{Q,\delta/2}$, $\mathtt{Verify}_{Q,\delta/2}$) and $\Gamma_{min}, \Gamma_{max}$ be the algorithms and parameters used in defining the explainability of H (Definition 3.2). Also, let $\mathcal{Q}$ be the set of inputs submitted to the evaluation oracle at Step **2**.

[24] From Remark D.1, there does not exist distinct i, j s.t. $x_i^* = x_j^*$, thus, the following procedure is identical to running the constraining algorithm with input $\{x_1^*, \ldots, x_P^*\}$.

Check	Check′
Input: $(\beta, \alpha, vk), (\check{P}, \mathcal{Q}, x_h^*)$	**Input:** $(\beta, \alpha), \check{P}$
1. If $\beta = 0$: (a) If $h-1 \geq \check{P} \wedge \alpha = 1$: Output 1. (b) Otherwise: Output 0. 2. If $\beta = 1$: (a) If $h-1 < \check{P} \wedge \mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^*) = 1$: Output 1. (b) Otherwise: Output 0.	1. If $\beta = 0$: (a) If $h-1 \geq \check{P} \wedge \alpha = 1$: Output 1. (b) Otherwise: Output 0. 2. If $\beta = 1$: (a) If $h-1 < \check{P} \wedge \alpha = 1$: Output 1. (b) Otherwise: Output 0.

Fig. 2 The functions Check and Check′.

Hybrid 1 and Hybrid 0 are identical except that in the beginning of the hybrid, the challenger samples a bit $\beta \xleftarrow{\$} \{0,1\}$, sets the bit α to be 1 with probability $(\mathit{\Gamma}_{min} + \mathit{\Gamma}_{max})/2$, and generates a verification key $vk \leftarrow \mathtt{VKeyGen}_{Q,\delta/2}(1^\lambda)$. It also samples $b^\dagger \xleftarrow{\$} \{0,1\}$. Then at Step 5 of the hybrid, after the adversary submits a bit b', the challenger runs

$$\zeta = \mathtt{Check}((\beta, \alpha, vk), (\check{P}, \mathcal{Q}, x_h^*))$$

where we define the function Check in Figure 2. Finally, if $\zeta = 1$, it outputs 1 if $b' = b$ and outputs 0 if $b' \neq b$; otherwise, it outputs 1 if $b^\dagger = b$ and outputs 0 if $b^\dagger \neq b$.

- **Hybrid 2.** This is identical to Hybrid 1 except that at Step 3 of the hybrid, the challenger runs $\zeta = \mathtt{Check}((\beta, \alpha, vk), (\check{P}, \mathcal{Q}, x_h^*))$.[25] It aborts the hybrid if $\zeta = 0$. In case it aborts, it outputs 1 if $b^\dagger = b$ and outputs 0 if $b^\dagger \neq b$.
- **Hybrid 3.** In Hybrid 3, the challenger maintains a list $\mathcal{L}$, which is initialized as an empty list. Then for each evaluation oracle query x, it puts $(x, \mathsf{H}.\mathtt{Eval}(k_\mathsf{H}, x))$ to the list $\mathcal{L}$. Also, for $i \in [1, P]$, it puts $(x_i^{(0)}, \mathsf{H}.\mathtt{Eval}(k_\mathsf{H}, x_i^{(0)}))$ and $(x_i^{(1)}, \mathsf{H}.\mathtt{Eval}(k_\mathsf{H}, x_i^{(1)}))$ into $\mathcal{L}$ after receiving $\mathcal{P}_0^*$ and $\mathcal{P}_1^*$. Finally, at Step 5 of the hybrid, if there exists $(x, u), (x', u') \in \mathcal{L}$ s.t.

$$x \neq x' \ \wedge u = u' \tag{17}$$

the challenger outputs 1 if $b^\dagger = b$ and outputs 0 if $b^\dagger \neq b$.
- **Hybrid 4.** This is identical to Hybrid 3 except that if $\beta = 1$, the challenger uses the simulator of H. In more detail, let SIM be the stateful simulator of H as defined in Definition 3.2, then the challenger modifies its behaviors at each step as follows if $\beta = 1$:
  1. At step 1, the challenger does not generate k_H. In addition, it samples $u^* \xleftarrow{\$} \{0,1\}^L$ and invokes $\mathtt{SIM}(vk, u^*)$.
  2. At Step 2, for each evaluation oracle query x, it computes $u \leftarrow \mathtt{SIM}(x)$ instead of computing it as $u = \mathsf{H}.\mathtt{Eval}(k_\mathsf{H}, x)$, where u is used to answer the evaluation oracle query and is recorded in $\mathcal{L}$.

[25] Note that all inputs to Check are known at Step 3.

3. At Step 3, if the challenger does not abort, it computes $k_{\mathsf{H}} \leftarrow \mathtt{SIM}(x_h^*)$ and sets $u_h = u^*$. It also stores $(x_h^{(b)}, u^*)$ instead of $(x_h^{(b)}, \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x_h^{(b)}))$ into $\mathcal{L}$. Then it computes the remaining parts identically as in Hybrid 3.
4. It proceeds identically as in Hybrid 3 for Step 4 and Step 5.

- **Hybrid 5.** This is identical to Hybrid 4 except that if $\beta = 1$, the challenger uses a random function f instead of the PRF $\mathsf{F}.\mathtt{Eval}(k_{\mathsf{F}}, \cdot)$ at Step 3.
- **Hybrid 6.** This is identical to Hybrid 5 except that at Step 1 of the hybrid, the challenger generates
  1. $\bar{\boldsymbol{t}}^* \xleftarrow{\$} \mathbb{Z}_q^{n-1}$, $\boldsymbol{t}^* = (1, \bar{\boldsymbol{t}}^{*\intercal})^\intercal$.
  2. $\bar{r}^* \xleftarrow{\$} \mathbb{Z}_q$, $\boldsymbol{r}^* = \boldsymbol{G}_{1,q}^{-1}(\bar{r}^*)$.
  3. $(pk^*, sk^*) \leftarrow \mathsf{FHE}.\mathtt{KeyGen}(1^\lambda, 1^d)$.
  4. $\boldsymbol{e}_j^* \leftarrow \tilde{D}_\sigma^m$ for $j \in [0, N+k]$.
  5. For $j \in [1, L]$: $ct_j^* \leftarrow \mathsf{FHE}.\mathtt{Enc}(pk^*, u^*[j])$.
  6. For $j \in [1, \varkappa]$: $ct_{L+j}^* \leftarrow \mathsf{FHE}.\mathtt{Enc}(pk^*, \boldsymbol{r}^*[j])$.
  7. $ct^* = (ct_1^* \| \dots \| ct_{L+\varkappa}^*)$.
  8. $\boldsymbol{a}_0^{*\intercal} = \boldsymbol{t}^{*\intercal} \cdot (\boldsymbol{A}_0 + \boldsymbol{G}) + \boldsymbol{e}_0^{*\intercal}$.
  9. For $j \in [1, N]$: $\boldsymbol{a}_j^{*\intercal} = \boldsymbol{t}^{*\intercal} \cdot (\boldsymbol{A}_j + ct^*[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_j^{*\intercal}$.
  10. For $j \in [1, k]$: $\boldsymbol{b}_j^{*\intercal} = \boldsymbol{t}^{*\intercal} \cdot (\boldsymbol{B}_j + sk^*[j] \cdot \boldsymbol{G}) + \boldsymbol{e}_{N+j}^{*\intercal}$.
  11. $\tilde{\boldsymbol{s}} = \boldsymbol{s} - \boldsymbol{t}^* \mod q$.

  Then at Step 3, if $\beta = 1$ and the challenger does not abort, it sets
  $$ct_h = ct^*, \quad \boldsymbol{a}_{h,j} = \boldsymbol{a}_j^* \text{ for } j \in [0, N], \quad \boldsymbol{b}_{h,j} = \boldsymbol{b}_j^* \text{ for } j \in [1, k]$$
  and computes
  $$\boldsymbol{t}_0 = \tilde{\boldsymbol{s}} - \sum_{i \in [1,P] \setminus \{h\}} \boldsymbol{t}_i$$
- **Hybrid 7.** This is identical to Hybrid 6 except that the challenger changes the way to answer the evaluation oracle query if $\beta = 1$. In particular, given an evaluation oracle query x, the challenger first computes
  $$\begin{cases} u \leftarrow \mathtt{SIM}(x) & \text{At Step 2} \\ u = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x) & \text{At Step 4} \end{cases}$$
  Then it computes y as follows and returns y to the adversary.
  1. For $j \in [1, \varkappa]$:
     (a) For $\iota \in [1, k]$: $\boldsymbol{C}_{j,\iota} = \mathtt{EvalPK}(\mathtt{C}_{u,j,\iota}, \boldsymbol{A}_0, \dots, \boldsymbol{A}_N)$.
     (b) $\boldsymbol{D}_j = \mathtt{IPEvalPK}(\boldsymbol{C}_{j,1}, \dots, \boldsymbol{C}_{j,k}, \boldsymbol{B}_1, \dots, \boldsymbol{B}_k)$.
  2. For $j \in [1, \varkappa]$:
     (a) For $\iota \in [1, k]$:
        i. $\widetilde{ct}_{j,\iota} = \mathtt{C}_{u,j,\iota}(ct^*)$.

ii. $\boldsymbol{c}_{j,\iota} = \texttt{EvalCT}(\mathtt{C}_{u,j,\iota}, \boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{a}_0^*, \ldots, \boldsymbol{a}_N^*, ct^*)$.

(b) $\boldsymbol{d}_j = \texttt{IPEvalCT}(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_k, \boldsymbol{c}_{j,1}, \ldots, \boldsymbol{c}_{j,k}, \boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_k^*, \widetilde{ct}_{j,1}, \ldots, \widetilde{ct}_{j,k})$.

3. $\bar{y} = (\sum_{j=1}^{\varkappa}(\tilde{\boldsymbol{s}}^\intercal \cdot \boldsymbol{D}_j + \boldsymbol{d}_j))[1] + v \mod q$.
4. $y = \lfloor \bar{y} \rceil_p \mod p$.

- **Hybrid 8.** This is identical to Hybrid 7 except that if $\beta = 1$, the challenger samples $\bar{\tilde{\boldsymbol{s}}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$ and sets $\tilde{\boldsymbol{s}} = (0, \bar{\tilde{\boldsymbol{s}}}^\intercal)^\intercal$.
- **Hybrid 9.** This is identical to Hybrid 8 except that if $\beta = 1$, the challenger samples
$$\boldsymbol{a}_j^* \xleftarrow{\$} \mathbb{Z}_q^m \text{ for } j \in [0, N], \quad \boldsymbol{b}_j^* \xleftarrow{\$} \mathbb{Z}_q^m \text{ for } j \in [1, k]$$
- **Hybrid 10.** This is identical to Hybrid 9 except that if $\beta = 1$, the challenger computes
$$ct_j^* \leftarrow \mathsf{FHE}.\texttt{Enc}(pk^*, 0)$$
for $j \in [1, L + \varkappa]$ and sets $ct^* = (ct_1^* \| \ldots \| ct_{L+\varkappa}^*)$.
- **Hybrid 11.** This is identical to Hybrid 10 except that the challenger stops using the simulator of H. In more detail, the challenger modifies its behaviors at each step as follows if $\beta = 1$:
  1. At step 1, the challenger samples $k_{\mathsf{H}} \leftarrow \mathsf{H}.\texttt{KeyGen}(1^\lambda)$. Also, it does not sample u^* and invoke $\texttt{SIM}(vk, u^*)$.
  2. At Step 2, for each evaluation oracle query x, it computes $u = \mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x)$ instead of computing it with the simulator, where u is used to answer the evaluation oracle query and is recorded in $\mathcal{L}$.
  3. At Step 3, if the challenger does not abort, it does not invoke the simulator to generate a hash key of H. Besides, it computes $u_h = \mathsf{H}.\texttt{Eval}(k_{\mathsf{H}}, x_h^{(b)})$ and stores $(x_h^{(b)}, u_h)$ instead of $(x_h^{(b)}, u^*)$ into $\mathcal{L}$ (note that u_h is not needed in generating the constrained key). Then it computes the remaining parts identically as in Hybrid 10.
  4. It proceeds identically as in Hybrid 10 for Step 4 and Step 5.
- **Hybrid 12.** This is identical to Hybrid 11 except that the challenger does not check if Equation (17) is satisfied (introduced in Hybrid 3) at Step 5. Also it does not maintain the list $\mathcal{L}$.
- **Hybrid 13.** This is identical to Hybrid 12 except that the challenger does not compute the `Check` function at Step 3. Note that, it still runs the `Check` function at Step 5.
- **Hybrid 14.** This is identical to Hybrid 13 except that at Step 5 of the hybrid, the challenger runs `Check'` instead of `Check`, where the function `Check'` is defined in Figure 2 and we use red underline to highlight the difference between `Check` and `Check'`.

Let $\mathcal{E}_i$ be the output of Hybrid i for $i \in [0, 14]$. We have the following claims.

Claim D.1. *If δ is non-negligible, then* $|\Pr[\mathcal{E}_1 = 1] - \frac{1}{2}| \geq \frac{\Gamma_{min}}{2} \cdot \delta - \frac{\Gamma_{max} - \Gamma_{min}}{4}$.

Proof. Hybrid 1 is identical to Hybrid 0 except that at the end of the hybrid, the challenger uses a random bit (i.e., $b^\dagger$) to replace the response of the adversary if the output of Check is 0. Let $\mathtt{X} = (\check{P}, \mathcal{Q}, x_h^*) \in \mathbb{Z} \times (\{0,1\}^l)^* \times \{0,1\}^l$ be any possible (partial) input to the function Check in Hybrid 1, and let

$$\gamma(\mathtt{X}) = \Pr[\beta \xleftarrow{\$} \{0,1\}, \alpha \leftarrow B_{(\mathit{\Gamma}_{min}+\mathit{\Gamma}_{max})/2}, \\ vk \leftarrow \mathtt{VKeyGen}_{Q,\delta/2}(1^\lambda) : \mathtt{Check}((\beta, \alpha, vk), \mathtt{X}) = 1]$$

where the distribution $B_{(\mathit{\Gamma}_{min}+\mathit{\Gamma}_{max})/2}$ outputs 1 with probability $(\mathit{\Gamma}_{min}+\mathit{\Gamma}_{max})/2$.

Note that if $h - 1 \geq \check{P}$, then

$$\gamma(\mathtt{X}) = \Pr[\beta = 0 \wedge \alpha = 1] = \frac{\mathit{\Gamma}_{min} + \mathit{\Gamma}_{max}}{4}$$

On the other hand, if $h - 1 < \check{P}$, then

$$\gamma(\mathtt{X}) = \Pr[\beta = 1 \wedge \mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^*) = 1] = \frac{\Pr[\mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^*) = 1]}{2}$$

Next, we bound the probability $\Pr[\mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^*) = 1]$ in case $h-1 < \check{P}$.

First, as Q is the upper bound on the number of the evaluation oracle queries made by $\mathcal{A}$, we have $|\mathcal{Q}| \leq Q$. In addition, since $h - 1 < \check{P}$, we have $h \leq \check{P}$, and thus $x_h^{(b)} \notin \mathcal{P}_{1-b}^*$. Then by Equation (3) required in the definition of privacy, $x_h^* = x_h^{(b)} \notin \mathcal{Q}$. Next, by the "Abort Probability" requirement in the explainability property of H, we have $\mathit{\Gamma}_{min} \leq \Pr[\mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x^*) = 1] \leq \mathit{\Gamma}_{max}$. Therefore, if $h - 1 < \check{P}$, we have

$$\frac{\mathit{\Gamma}_{min}}{2} \leq \gamma(\mathtt{X}) \leq \frac{\mathit{\Gamma}_{max}}{2}$$

Since we also have $\frac{\mathit{\Gamma}_{min}+\mathit{\Gamma}_{max}}{4} \in [\frac{\mathit{\Gamma}_{min}}{2}, \frac{\mathit{\Gamma}_{max}}{2}]$, for any possible input X, we have

$$\gamma(\mathtt{X}) \in [\frac{\mathit{\Gamma}_{min}}{2}, \frac{\mathit{\Gamma}_{max}}{2}]$$

Next, note that $\delta = |\Pr[\mathcal{E}_0 = 1] - \frac{1}{2}| = |\Pr[b = b'] - \frac{1}{2}|$, then by Lemma A.6, we have

$$|\Pr[\mathcal{E}_1 = 1] - \frac{1}{2}| \geq \frac{\mathit{\Gamma}_{min}}{2} \cdot \delta - \frac{\mathit{\Gamma}_{max} - \mathit{\Gamma}_{min}}{4}$$

□

Claim D.2. $|\Pr[\mathcal{E}_1 = 1] - \Pr[\mathcal{E}_2 = 1]| = 0$.

Proof. Hybrid 2 and Hybrid 1 are identically proceeded if the function Check outputs 1, and in case that it outputs 0, both hybrids will output 1 with probability 1/2. Thus, the output distributions of the two hybrids are identical. □

Claim D.3. $|\Pr[\mathcal{E}_2 = 1] - \Pr[\mathcal{E}_3 = 1]| \leq negl(\lambda)$.

Proof. Hybrid 2 and Hybrid 3 are identical unless Equation (17) is satisfied. This will occur with negligible probability due to the injectivity of H. $\square$

Claim D.4. $|\Pr[\mathcal{E}_3 = 1] - \Pr[\mathcal{E}_4 = 1]| \leq negl(\lambda)$.

Proof. This comes from the indistinguishability requirement in the explainability property of H. More precisely, assume that $|\Pr[\mathcal{E}_3 = 1] - \Pr[\mathcal{E}_4 = 1]|$ is non-negligible, then we can construct an adversary $\mathcal{B}$ that breaks the indistinguishability of H.

In the beginning, the adversary $\mathcal{B}$ samples $b \xleftarrow{\$} \{0,1\}$ and generates $(\boldsymbol{A}_0, \ldots, \boldsymbol{A}_N, \boldsymbol{B}_1, \ldots, \boldsymbol{B}_k, \boldsymbol{s}, v, k_{\mathsf{F}})$ as in Hybrid 3 and Hybrid 4. Then it answers the evaluation oracle queries from $\mathcal{A}$. In particular, given an input x, it submits x to its own oracle, and on receiving a response u, it stores (x, u) to a list $\mathcal{L}$, runs the remaining parts of the evaluation algorithm with u and returns the result y to $\mathcal{A}$. Next, $\mathcal{A}$ will submit two sets $\mathcal{P}_0^* = \{x_1^{(0)}, \ldots, x_P^{(0)}\}$ and $\mathcal{P}_1^* = \{x_1^{(1)}, \ldots, x_P^{(1)}\}$, and let $\hat{P} = |\mathcal{P}_0^* \cap \mathcal{P}_1^*|$, $\check{P} = P - \hat{P}$. Then if $h - 1 \geq \check{P}$, $\mathcal{B}$ chooses an input x^* that is different from all evaluation queries, submits it as its challenge, and outputs 1 with probability $1/2$ no matter what response it receives from its challenger. Otherwise, it submits $x_h^{(b)}$ as its challenge[26]. If the response is $\perp$, it outputs 1 with probability $1/2$. Otherwise, given the response (k_{H}, u^*) it sets $u_h = u^*$, stores $(x_h^{(b)}, u^*)$ to $\mathcal{L}$ and then proceeds identically as in Hybrid 3 and Hybrid 4. Finally, after $\mathcal{A}$ submits a bit b', $\mathcal{B}$ outputs 1 with probability $1/2$ if Equation (17) is satisfied. Otherwise, it outputs 1 if $b' = b$.

It is easy to see, $\mathcal{B}$ can simulate Hybrid 3 (with $\beta = 1$) perfectly if it is in the experiment `ExpReal`, and it can simulate Hybrid 4 (with $\beta = 1$) perfectly if it is in the experiment `ExpIdeal`. In addition, $\mathcal{B}$ outputs 1 iff the hybrids outputs 1. Also, note that Hybrid 3 and Hybrid 4 are identical if $\beta = 0$. Thus, from the indistinguishability of H, we have $|\Pr[\mathcal{E}_3 = 1] - \Pr[\mathcal{E}_4 = 1]| \leq negl(\lambda)$. This completes the reduction. $\square$

Claim D.5. $|\Pr[\mathcal{E}_4 = 1] - \Pr[\mathcal{E}_5 = 1]| \leq negl(\lambda)$.

Proof. This comes from the pseudorandomness of F by a direct reduction. $\square$

Claim D.6. $|\Pr[\mathcal{E}_5 = 1] - \Pr[\mathcal{E}_6 = 1]| = 0$.

Proof. In Hybrid 6, the challenger precomputes parts of the constrained key. The adversary's view in Hybrid 5 and Hybrid 6 are identical unless there exists $i \neq h$ s.t. $u^* = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x_i^*)$. First, by Remark D.1, we have $x_i^* \neq x_h^*$. Thus, if $u^* = \mathsf{H}.\mathtt{Eval}(k_{\mathsf{H}}, x_i^*)$, both hybrids will output 1 with probability $1/2$. Therefore, the output distributions of the two hybrids are identical. $\square$

Claim D.7. $|\Pr[\mathcal{E}_6 = 1] - \Pr[\mathcal{E}_7 = 1]| \leq negl(\lambda)$.

[26] Note that as shown in the proof of Claim D.1, $x_h^{(b)}$ is different from all evaluation queries in case $h - 1 < \check{P}$.

Proof. Hybrid 6 and Hybrid 7 only differ in the way for answering the evaluation oracle if $\beta = 1$. Let $\mathcal{B}_1$ be the event that x_h^b has been submitted to the evaluation oracle, and let $\mathcal{B}_2$ be the event that Equation (17) is satisfied. Then, we can prove that the responses to the evaluation oracle queries are identical in Hybrid 6 and Hybrid 7 with all but negligible probability unless $\mathcal{B}_1$ or $\mathcal{B}_2$ occurs.[27] Note that as shown in the proof of Claim D.1, $\mathcal{B}_1$ occurs only if $h - 1 \geq \check{P}$, and in this case (recall that $\beta = 1$) both hybrids will output 1 with probability 1/2. In addition, if $\mathcal{B}_2$ occurs and the abortion condition at Step 3 is not triggered, then both hybrids will also output 1 with probability 1/2. Besides, if a hybrid aborts at Step 3, it still outputs 1 with probability 1/2. That is, even if the adversary may detect the non-negligible difference between the two hybrids in case $\mathcal{B}_1$ or $\mathcal{B}_2$ occurs, the hybrid will always output 1 with probability 1/2 in this case.

To summarize, the adversary's view in Hybrid 6 and Hybrid 7 are close unless the event $\mathcal{B}_1$ or $\mathcal{B}_2$ occurs, and in case at least one of them occurs, the outputs of the two hybrids are identical. Thus, the difference between the probabilities that the two hybrids outputs 1 is negligible. ☐

Claim D.8. $|\Pr[\mathcal{E}_7 = 1] - \Pr[\mathcal{E}_8 = 1]| = 0$.

Proof. Hybrid 7 and Hybrid 8 only differ in the way for generating $\tilde{\boldsymbol{s}}$ in case $\beta = 1$. In Hybrid 7, $\tilde{\boldsymbol{s}} = \boldsymbol{s} - \boldsymbol{t}^* = (0, (\bar{\boldsymbol{s}} - \bar{\boldsymbol{t}}^*)^\intercal)^\intercal \mod q$, and in Hybrid 8, the challenger samples $\bar{\tilde{\boldsymbol{s}}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$ and sets $\tilde{\boldsymbol{s}} = (0, \bar{\tilde{\boldsymbol{s}}}^\intercal)^\intercal$. Note that, in Hybrid 7, $\bar{\boldsymbol{s}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$ and is only used for generating $\tilde{\boldsymbol{s}}$. Thus, the adversary's views are identical in these two hybrids. ☐

Claim D.9. $|\Pr[\mathcal{E}_8 = 1] - \Pr[\mathcal{E}_9 = 1]| \leq negl(\lambda)$.

Proof. Indistinguishability between Hybrid 8 and Hybrid 9 comes from the hardness of $\mathrm{LWE}_{n-1,q,\tilde{D}_\sigma}$ by a direct reduction. ☐

Claim D.10. $|\Pr[\mathcal{E}_9 = 1] - \Pr[\mathcal{E}_{10} = 1]| \leq negl(\lambda)$.

Proof. Indistinguishability between Hybrid 9 and Hybrid 10 comes from the security of FHE by a direct reduction. ☐

Claim D.11. $|\Pr[\mathcal{E}_{10} = 1] - \Pr[\mathcal{E}_{11} = 1]| \leq negl(\lambda)$.

Proof. This comes from the indistinguishability requirement in the explainability property of H. The reduction is similar to the proof of Claim D.4, and we omit the details here. ☐

Claim D.12. $|\Pr[\mathcal{E}_{11} = 1] - \Pr[\mathcal{E}_{12} = 1]| \leq negl(\lambda)$.

Proof. This comes from the injectivity of the explainable hash H directly. ☐

Claim D.13. $|\Pr[\mathcal{E}_{12} = 1] - \Pr[\mathcal{E}_{13} = 1]| = 0$.

[27] The proof is identical to the proof of Lemma D.13 and we omit the details here.

Proof. Hybrid 12 and Hybrid 13 are identically proceeded if the function `Check` outputs 1. In case that the function outputs 0, both hybrids will output 1 with probability 1/2. Thus, the output distributions of the two hybrids are identical. □

Claim D.14. $|\Pr[\mathcal{E}_{13} = 1] - \Pr[\mathcal{E}_{14} = 1]| \leq \frac{3(\Gamma_{max} - \Gamma_{min})}{4}$.

Proof. Hybrid 13 and Hybrid 14 are identical before the adversary outputs b'. Let $b, \beta, \alpha, vk, b^\dagger$ be the random variables sampled by the challenger at the beginning of the hybrid. Let $\mathcal{Q}$ be the set of inputs submitted to the evaluation oracle at Step 2 of the hybrid. Let $\mathcal{P}_0^* = \{x_1^{(0)}, \ldots, x_P^{(0)}\}$ and $\mathcal{P}_1^* = \{x_1^{(1)}, \ldots, x_P^{(1)}\}$ be the two puncture sets submitted by the adversary at Step 3, and let $\hat{P} = |\mathcal{P}_0^* \cap \mathcal{P}_1^*|$, $\check{P} = P - \hat{P}$. Also, let b' be the output of the adversary. We consider the following four cases:

- $\mathtt{C}_1 : \beta = 0 \wedge h - 1 \geq \check{P} \wedge \alpha = 1$. In this case, both the `Check` algorithm and the `Check`$'$ algorithm will output 1. Thus, we have

$$\Pr[\mathcal{E}_{13} = 1 \mid \mathtt{C}_1] = \Pr[b = b' \mid \mathtt{C}_1] = \Pr[\mathcal{E}_{14} = 1 \mid \mathtt{C}_1]$$

- $\mathtt{C}_2 : \beta = 0 \wedge (h - 1 < \check{P} \vee \alpha = 0)$. In this case, both the `Check` algorithm and the `Check`$'$ algorithm will output 0. Thus, we have

$$\Pr[\mathcal{E}_{13} = 1 \mid \mathtt{C}_2] = \Pr[b = b^\dagger \mid \mathtt{C}_2] = \Pr[\mathcal{E}_{14} = 1 \mid \mathtt{C}_2]$$

- $\mathtt{C}_3 : \beta = 1 \wedge h - 1 \geq \check{P}$. In this case, both the `Check` algorithm and the `Check`$'$ algorithm will output 0. Thus, we have

$$\Pr[\mathcal{E}_{13} = 1 \mid \mathtt{C}_3] = \Pr[b = b^\dagger \mid \mathtt{C}_3] = \Pr[\mathcal{E}_{14} = 1 \mid \mathtt{C}_3]$$

- $\mathtt{C}_4 : \beta = 1 \wedge h - 1 < \check{P}$. In this case, we have

$$\begin{aligned}
&|\Pr[\mathcal{E}_{13} = 1 \mid \mathtt{C}_4] - \Pr[\mathcal{E}_{14} = 1 \mid \mathtt{C}_4]| \\
=&|\Pr[b = b' \wedge \mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^{(b)}) = 1 \mid \mathtt{C}_4] \\
&+ \Pr[b = b^\dagger \wedge \mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^{(b)}) = 0 \mid \mathtt{C}_4] \\
&- (\Pr[b = b' \wedge \alpha = 1 \mid \mathtt{C}_4] + \Pr[b = b^\dagger \wedge \alpha = 0 \mid \mathtt{C}_4])| \\
=&|\Pr[\mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^{(b)}) = 1 \mid b = b' \wedge \mathtt{C}_4] \cdot \Pr[b = b' \mid \mathtt{C}_4] \\
&+ \frac{1}{2}(1 - \Pr[\mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^{(b)}) = 1 \mid \mathtt{C}_4]) \\
&- (\frac{\Gamma_{min} + \Gamma_{max}}{2} \cdot \Pr[b = b' \mid \mathtt{C}_4] + \frac{1}{2} \cdot (1 - \frac{\Gamma_{min} + \Gamma_{max}}{2}))|
\end{aligned}$$

  Note that vk is not used before running the check algorithms in both Hybrid 13 and Hybrid 14. In addition, as shown in the proof of Claim D.1, $x_h^{(b)} \notin \mathcal{Q}$ if $h - 1 < \check{P}$. Then by the "Abort Probability" requirement in the explainability property of H, there exists $p_1, p_2 \in [\Gamma_{min}, \Gamma_{max}]$ s.t.

$$p_1 = \Pr[\mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^{(b)}) = 1 \mid b = b' \wedge \mathtt{C}_4]$$

$$p_2 = \Pr[\mathtt{Verify}_{Q,\delta/2}(vk, \mathcal{Q}, x_h^{(b)}) = 1 \mid \mathtt{C}_4]$$

Therefore, we have

$$\begin{aligned}
&|\Pr[\mathcal{E}_{13} = 1 \mid \mathtt{C}_4] - \Pr[\mathcal{E}_{14} = 1 \mid \mathtt{C}_4]| \\
=&|(p_1 - \frac{\Gamma_{min} + \Gamma_{max}}{2}) \cdot \Pr[b = b' \mid \mathtt{C}_4] + \frac{1}{2}(\frac{\Gamma_{min} + \Gamma_{max}}{2} - p_2)| \\
\leq&\frac{3(\Gamma_{max} - \Gamma_{min})}{4}
\end{aligned}$$

To summarize, we have

$$\begin{aligned}
|\Pr[\mathcal{E}_{13} = 1] - \Pr[\mathcal{E}_{14} = 1]| &\leq \sum_{i=1}^{4} \Pr[\mathtt{C}_i] \cdot |\Pr[\mathcal{E}_{13} = 1 \mid \mathtt{C}_i] - \Pr[\mathcal{E}_{14} = 1 \mid \mathtt{C}_i]| \\
&\leq |\Pr[\mathcal{E}_{13} = 1 \mid \mathtt{C}_4] - \Pr[\mathcal{E}_{14} = 1 \mid \mathtt{C}_4]| \leq \frac{3(\Gamma_{max} - \Gamma_{min})}{4}
\end{aligned}$$

□

Claim D.15. $\Pr[\mathcal{E}_{14} = 1] = \frac{1}{2}$.

Proof. In Hybrid 14, the uniform bit b is only used to check if $b = b'$ or if $b^\dagger = b$ at the end of the hybrid. Thus, we always have $\Pr[b' = b] = \frac{1}{2}$ and $\Pr[b^\dagger = b] = \frac{1}{2}$, and this implies that Hybrid 14 will output 1 with probability 1/2. □

Combining Claim D.1 to Claim D.14, we have

$$\begin{aligned}
|\Pr[\mathcal{E}_{14} = 1] - \frac{1}{2}| \geq&|\Pr[\mathcal{E}_1 = 1] - \frac{1}{2}| - \sum_{i=1}^{13} |\Pr[\mathcal{E}_i = 1] - \Pr[\mathcal{E}_{i+1} = 1]| \\
\geq&\frac{\Gamma_{min}}{2} \cdot \delta - \frac{\Gamma_{max} - \Gamma_{min}}{4} - negl(\lambda) - \frac{3(\Gamma_{max} - \Gamma_{min})}{4} \\
=&\Gamma_{min} \cdot \frac{\delta}{2} - (\Gamma_{max} - \Gamma_{min}) - negl(\lambda)
\end{aligned}$$

which is non-negligible due to the "Abort Probability" requirement in the explainability property of H (recall that the parameter we used is $(Q, \delta/2)$). This contradicts Claim D.15. Thus, the advantage of the adversary $\mathcal{A}$ in distinguishing Game H_{h-1} and Game H_h cannot be non-negligible. This completes proof of Lemma D.19, and privacy of PRF follows. □