Post-Quantum Adaptive Multiparty NIKE

Shafik Nassar
UT Austin
shafik@cs.utexas.edu

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

November 22, 2026

Abstract

Non-interactive key exchange (NIKE) is a fundamental cryptographic primitive that enables a group of users to establish a shared secret key without any interaction. Constructing multiparty NIKE for groups larger than three has proven notoriously difficult, with all known constructions requiring indistinguishability obfuscation (iO) or multilinear maps. A major challenge in this area is achieving adaptive security, where the adversary chooses the target set of users after observing the system's public keys and can dynamically corrupt honest users, without incurring a sub-exponential loss in the security reduction. The only known construction up to this point has been that of Koppula, Waters, and Zhandry (TCC 2022), which relied on polynomially secure iO and a q-type assumption over pairing-free groups.

In this work, we present the first adaptively secure multiparty NIKE schemes from standard, static, and post-quantum polynomial assumptions. More broadly, we develop new techniques that allow us to construct adaptive multiparty NIKE from polynomially secure iO and either of Ring LWE, DDH or even cryptographic group actions. Specifically, we introduce a new abstraction called Key-Accumulative PRFs (KA-PRFs), which captures the structure common to all of the aforementioned algebraic assumptions, and construct our NIKE scheme from polynomially secure KA-PRFs and polynomially secure iO.

Our scheme supports an unbounded number of honest and malicious users, where the party size and the number of dynamic corruptions are fixed at setup.

Contents

1 Introduction **2**
1.1 Our Results . . . 2

2 Technical Overview **3**
2.1 Adaptive Multiparty NIKE from $i\mathcal{O}$ and DDH . . . 4
2.2 Adaptive Multiparty NIKE from $i\mathcal{O}$ and RLWE . . . 8
2.3 Generalizing our Approach . . . 9

3 Preliminaries **9**

4 Key-Accumulative Pseudorandom Functions **13**

5 Instantiating KA-PRFs from Standard Assumptions **15**
5.1 Construction from DDH . . . 15
5.2 Construction from Ring-LWE . . . 16

6 NIKE Construction **20**
6.1 Security Analysis . . . 21

A KA-PRF from Cryptographic Group Actions **69**

1 Introduction

An N-party non-interactive key exchange (NIKE) is a fundamental cryptographic primitive in which users publish their public keys, and then any set of N users can derive a shared, secret key without any further interaction.

Despite being a fundamental primitive, with the $N = 2$ case dating back to the seminal work of Diffie and Hellman [DH76] and $N = 3$ essentially kick-starting the modern era of pairing-based cryptography [Jou00], constructing N-party NIKE for $N > 3$ has proven to be notoriously difficult, with all viable constructions requiring indistinguishability obfuscation ($i\mathcal{O}$) [BZ14] or multilinear maps [BS02].

The strong, and often proper, notion of security in cryptography requires protection against *adaptive* adversaries that choose their challenge after interacting with the challenger. Although NIKE with static security can be generically upgraded to have adaptive security, this transformation requires the security reduction to guess the challenge set. When considering *multiparty* NIKE, where N can be an arbitrary polynomial in the security parameter λ, the aforementioned transformation incurs a sub-exponential loss in security, and thus requires sub-exponential hardness. The work of Koppula, Waters and Zhandry [KWZ22] presented the first and, as far as we know, the only adaptive multiparty NIKE from polynomial assumptions. Despite that, their construction has the following drawbacks:

1. Their construction relies on group-based assumptions (in addition to $i\mathcal{O}$), meaning that a quantum computer can break their scheme [Sho94]. Although current "well-founded" $i\mathcal{O}$ constructions can also be broken by quantum computers [JLS21, JLS22], the community is working on constructing post-quantum $i\mathcal{O}$ [CLW25, HJL25]. If post-quantum $i\mathcal{O}$ is ever built, the group-based assumption in Koppula et al. would be the quantumly "weak link".

2. The construction of Koppula et al. uses a dynamic q-type assumption over pairing-free groups, where q is equal to the set size N. This means that in order to support larger parties, we would have to make a stronger and "larger" assumption. In addition, techniques based on static assumptions in one algebraic setting, e.g., groups, are easier to port to other algebraic settings, e.g., lattices.

With those drawbacks, it is natural to ask whether we can construct adaptively secure multiparty NIKE from polynomially secure $i\mathcal{O}$ and standard, static lattice assumptions.

1.1 Our Results

In this work, we answer the question affirmatively. We start by strictly improving upon Koppula et al. [KWZ22] by developing new techniques that allow us to construct adaptive multiparty NIKE from $i\mathcal{O}$ and standard DDH, instead of $i\mathcal{O}$ and their q-type assumption,

Theorem 1.1. *Assuming polynomially secure $i\mathcal{O}$ and polynomially secure DDH, there exists an adaptively secure multiparty NIKE scheme with unbounded honest users and unbounded malicious users. The party size and the number of dynamic corruptions are a priori bounded.*

Theorem 1.1 should be compared to [KWZ22, Theorem 1] and [KWZ22, Theorem 2]: The former achieves a construction from DDH, but is limited to a bounded number of honest users. The latter gets the same properties as Theorem 1.1, but requires a q-type assumption. As such, Theorem 1.1 gets the best of both worlds.

We also observe that our construction readily applies to the group action setting [ADFMP20], and can be proven secure under the *weak pseudorandom effective group action* assumption, which is analogous to DDH in groups. Thus, we get the following theorem:

Theorem 1.2. *Assuming polynomially secure $i\mathcal{O}$ and polynomially secure weak pseudorandom effective group actions, there exists an adaptively secure multiparty NIKE scheme with unbounded honest users and unbounded malicious users. The party size and the number of dynamic corruptions are a priori bounded.*

Next, we show that we can adapt our previous construction to the lattice setting, and base it on the Ring Learning with Errors assumption (RLWE) instead of DDH:

Theorem 1.3. *Assuming polynomially secure $i\mathcal{O}$ and polynomially secure RLWE with sub-exponential noise-to-modulus ratio, there exists an adaptively secure multiparty NIKE scheme with unbounded honest users and unbounded malicious users. The party size and the number of dynamic corruptions are a priori bounded.*

We emphasize that although we require sub-exponential modulus, we still only make a standard polynomial hardness assumption. As such, Theorem 1.2 and Theorem 1.3 give the first potentially post-quantum secure instantiation of adaptive multiparty NIKE from polynomial assumptions.

To unify our approach for all 3 instantiations, we come up with an abstraction called Key-Accumulative PRF (KA-PRF), that captures the algebraic structure and cryptographic hardness that we exploit in DDH, group actions and RLWE. This abstraction allows us to present a single, unified construction of adaptive NIKE from $i\mathcal{O}$, thus getting all three algebraic birds with one stone. In addition, KA-PRFs help in distilling the interesting properties of each algebraic regime.

We point out that going from Theorem 1.1 to Theorem 1.3 emphasizes the advantage of using static assumptions: while adapting the construction to RLWE still comes with its own unique challenges, starting from standard DDH - rather than the "DDH powers" q-type assumption - greatly simplifies our task.

Finally, for a discussion on the importance of basing security on polynomial hardness instead of sub-exponential (even when using $i\mathcal{O}$) and bounding the number of dynamic corruptions, we refer the reader to [KWZ22, Section 1].

2 Technical Overview

In this section, we overview our NIKE constructions. Our simplest construction is the one based on DDH, and it serves as a stepping stone towards our RLWE-based construction. First, we recall that NIKE in the common reference string (CRS) model consists of the following algorithms:

- A setup algorithm takes in the security parameter λ, a bound on the party size N and a bound on the number of corruptions C, and outputs a CRS.
- A publish algorithm that takes the CRS and generates a key pair $(\mathsf{pk}, \mathsf{sk})$.
- An evaluation algorithm which takes in the CRS, a list of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ and a single secret key sk_i, and outputs a shared key y.

Correctness states that for any two (honestly-generated) key pairs $(\mathsf{pk}_0, \mathsf{sk}_0)$ and $(\mathsf{pk}_1, \mathsf{sk}_1)$ and any set U, if $\mathsf{pk}_0, \mathsf{pk}_1 \in U$, then the shared key derived using sk_0 is equal to the shared key derived using sk_1.

In the adaptive security game, the adversary chooses a party bound N and a corruptions bound C. The challenger samples a CRS and sends it to the adversary. The adversary can now make the following queries:

- **Honest user registration:** The challenger samples a key pair, and sends the public key to the adversary.
- **Honest user corruption:** The adversary picks an honest user, and the challenger responds with that user's secret key. The adversary can make up to C such queries.
- **Challenge query:** The adversary picks a set of N honest, uncorrupted users. The challenger either sends their shared key, or a random string. The adversary makes one such query at any time during the game.

We say that the scheme is adaptively secure if any efficient adversary can guess whether the challenger responded with the shared key or a random string with negligible advantage. Usually, the game also allows **malicious user registration** and **shared key evaluation** queries. However, Koppula et al. [KWZ22] showed that assuming $i\mathcal{O}$, we can generically upgrade an adaptively secure NIKE to support those queries. In addition, they showed that the trusted setup which samples the CRS can also be generically eliminated assuming $i\mathcal{O}$. Therefore, we work with this simplified definition.

2.1 Adaptive Multiparty NIKE from $i\mathcal{O}$ and DDH

Our DDH-based NIKE construction utilizes $i\mathcal{O}$, a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$, an algebraic group $\mathbb{G}$ of order p with generator g, and a puncturable PRF $F_K : \{0,1\}^\lambda \to \mathbb{G}$. The algorithms look as follows:

- **Setup:** The setup algorithm samples a PPRF key K and a random group element $v = g^\alpha$, where $\alpha \xleftarrow{R} \mathbb{Z}_p$, which we refer to as *the base*. The CRS is an obfuscation P_{crs} of the program described in Figure 1.

Constants: A PPRF key K, and a base v.
Inputs: A PRG seed s, a list of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
2. Computes $\gamma_j = F_K(\mathsf{pk}_j)$ for all $j \in [N]$.
3. Computes $\Gamma = \prod_{j=1}^{N} \gamma_j \mod p$.
4. Outputs v^Γ.

Figure 1: Program $\mathsf{crsProg}[K, v]$.

- **Key pair publishing:** The publish algorithm samples a random seed $s \xleftarrow{R} \{0,1\}^\lambda$ and sets the public key as $\mathsf{pk} = G(s)$. The secret key will be derived from the CRS by hardcoding the PRG seed in it. Namely, we hardcode s into the program P_{crs} and obfuscate it again: $\mathsf{sk} \leftarrow i\mathcal{O}(P_{\mathrm{crs}}(s, \cdot, \cdot))$. The secret key is now an obfuscated program that takes in two inputs. Observe that the resulting program has two layers of obfuscation.
- **Shared key evaluation:** The evaluation algorithm takes in a list of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ and a single secret key sk_i. Let pk_i be the public key corresponding to sk_i. The algorithm simply outputs $\mathsf{sk}(U, i)$. Namely, the shared key corresponding to the list U is v^Γ where

$$\Gamma = \prod_{j=1}^{N} F_K(\mathsf{pk}_j).$$

Altruistic key management. The natural question that rises here is why do we go through the effort of "re-obfuscating" the CRS program. After all, the user can just keep the seed s as her secret key, and compute the shared key as $P_{\mathrm{crs}}(s, U, i)$. However, if we just put the seed in the secret key "in the clear", we do not know how to prove adaptive security against dynamic corruptions without random oracles or sub-exponential security.[1] Our approach, in which honest users re-obfuscate P_{crs} and then discard their PRG seeds, ensures that even an adversary that dynamically corrupts honest users, only gets to see obfuscated programs that contain the seeds, which gives us some leverage to push the security proof through. We call this *altruistic key management* because by throwing away her PRG seed, the honest user protects others in the case she is compromised even though she will not get any individual benefit.

We remark that Koppula et al. [KWZ22] take a similar altruistic approach by attaching a random message, in addition to the PRG seed, to the secret key. In their security proof, they embed an obfuscated program in this random message, and plant a "hidden trigger" in their CRS program to run the embedded secret program. While we could take this approach as well, this requires additional cryptographic machinery and adds significant complication to the security proof.

Security Proof

We now describe our security proof, which goes through a sequence of hybrids. The first hybrid corresponds to the real security game. The final goal will be to take the base used for evaluating the challenge set out of the adversary's view, thus ensuring that the shared key for the challenge set is indistinguishable from random.

[1] See [NW26] for two generic transformations that achieve security against dynamic corruptions using a random oracle or complexity leveraging.

Step 1: Modifying the secret keys. The first step is to modify the challenger-generated *secret* keys: instead of re-obfuscating P_{crs} which has the PRG seed hardcoded in it, the secret key now has the public key itself hardcoded. That is, the secret key corresponding to the public key pk is an obfuscation of the program $\mathsf{skProg}[K, v, \mathsf{pk}]$ from Figure 2:

Constants: A PPRF key K, and a base v, a public key pk.
Inputs: A list of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
2. Computes $\gamma_j = F_K(\mathsf{pk}_j)$ for all $j \in [N]$.
3. Computes $\Gamma = \prod_{j=1}^{N} \gamma_j \mod p$.
4. Outputs v^{Γ}.

Figure 2: Program $\mathsf{skProg}[K, v, \mathsf{pk}]$.

When $\mathsf{pk} = G(s)$, we have that $\mathsf{skProg}[K, v, \mathsf{pk}]$ is functionally equivalent to the CRS program with s hardcoded in it $P_{\mathsf{crs}}(s, \cdot, \cdot)$, therefore the adversary cannot distinguish this switch by $i\mathcal{O}$ security. We refer to the resulting experiment as Hybrid A.

Step 2: Labeling public keys. In order to execute our next step, we will need a tool that allows distinguishing challenger-generated public keys from other strings in $\{0,1\}^{3\lambda}$, using some secret trapdoor. This can be done if the public keys are computed using a *pseudorandom deterministic encryption* scheme (PDE).[2] In a nutshell, a PDE is a secret key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ in which Enc is deterministic. The message space consists of integers in $[2^\lambda]$, and the ciphertexts are in $\{0,1\}^{3\lambda}$. A PDE should satisfy the two following properties:

- **Strong correctness:** For any encryption key $\mathsf{ek} \leftarrow \mathsf{Gen}(1^\lambda)$ and any $i \in [2^\lambda]$, we have that only $\mathsf{Enc}(\mathsf{ek}, i)$ decrypts to i. That is $\mathsf{Dec}(\mathsf{ek}, \mathsf{ct}) = i$ if and only if $\mathsf{ct} = \mathsf{Enc}(\mathsf{ek}, i)$. For any illegal ciphertext ct, we have $\mathsf{Dec}(\mathsf{ek}, \mathsf{ct}) = \perp$.
- **Pseudorandomness:** For any polynomial $Q(\lambda)$, the two following distributions are computationally indistinguishable:

$$\left\{(\mathsf{ct}_1, \ldots, \mathsf{ct}_Q) : \begin{array}{ll} \mathsf{ek} & \leftarrow \mathsf{Gen}(\lambda) \\ \mathsf{ct}_i & = \mathsf{Enc}(\mathsf{ek}, i) \end{array}\right\} \text{ and } \left\{(\mathsf{ct}_1, \ldots, \mathsf{ct}_Q) : \mathsf{ct}_i \xleftarrow{R} \{0,1\}^{3\lambda}\right\}$$

A pseudorandom deterministic encryption can be constructed from any PRF and therefore can be based on OWFs [GGM86, HILL99].

In the next step, we want to switch challenger-generated public keys from being PRG samples to being PDE ciphertexts. This is done by relying on PRG security, PDE pseudorandomness, and the fact that the PRG seeds and the encryption key do not appear in the adversary's view. We rely on the fact that in Hybrid A, the PRG seeds do not appear in the adversary's view. This allows us to reach a hybrid where the public key of challenger-generated user i is a ciphertext $\mathsf{Enc}(\mathsf{ek}, i)$. We refer to this experiment as Hybrid B. From now on, we refer to a public key of the form $\mathsf{Enc}(\mathsf{ct}, i)$ as a *labeled key*, and to i as the *label*. We assume that labeled keys do not intersect with the image of the PRG. This can be guaranteed with overwhelming probability by sampling a global uniform string pad at setup, and XORing it with the PDE ciphertexts. For ease of exposition, we drop this notation for this overview.

Step 3: Isolating fully labeled lists. Next, we want to ensure that fully labeled public key lists — lists that contain only labeled public keys — use a distinct base from lists that include at least one unlabeled key. For now, we will strengthen this requirement and require $N+1$ different, independent bases $v_0 = g^{\alpha_0}, \ldots, v_N = g^{\alpha_N}$ where $\alpha_0, \ldots, \alpha_N \xleftarrow{R} \mathbb{Z}_p$, such that a list U that contains k labeled keys will use the base v_k. That is, for an adversary that

[2] A stronger version of the primitive called puncturable pseudorandom deterministic encryption was first introduced by Koppula, Poelstra and Waters et al. [KPW17].

registers at most Q honest users, we want to get to a Hybrid C where the CRS program looks as follows (the secret key programs will also be modified analogously):

Constants: A PPRF key K, bases $v_0, \ldots, v_N$, PDE key ek.
Inputs: A PRG seed s, a list of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
2. Computes $\gamma_j = F_K(\mathsf{pk}_j)$ for all $j \in [N]$.
3. Computes $\Gamma = \prod_{j=1}^{N} \gamma_j \mod p$.
4. Let k be the number of labeled $\mathsf{pk} \in U$, i.e., $\mathsf{Dec}(\mathsf{ek}, \mathsf{pk}) \in [Q]$.
5. Outputs $v_k{}^{\Gamma}$.

Figure 3: Program $\mathsf{crsProg}[K, v_0, \ldots, v_N, \mathsf{ek}]$.

To get from Hybrid B to Hybrid C, we define a sequence of hybrids $\mathsf{C}_0, \ldots, \mathsf{C}_Q$, where for each $q \in [Q]$, the integer k only counts public keys that have labels in the set $[q]$ (the set $[0]$ is defined to be the empty set). That is, line 4 in the program is replaced with the following:

4. Let k be the number of labeled $\mathsf{pk} \in U$, i.e., $\mathsf{Dec}(\mathsf{ek}, \mathsf{pk}) \in [q]$.

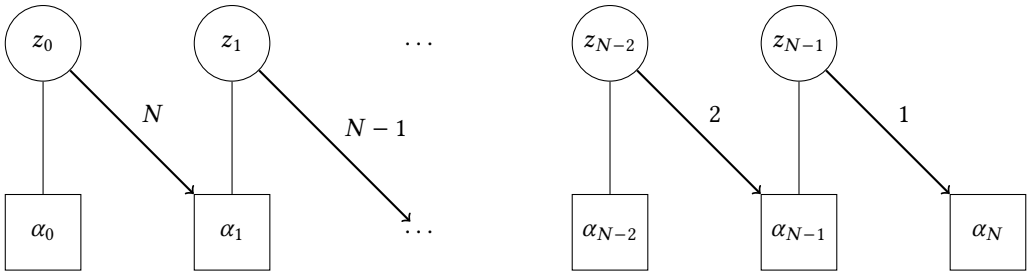
For $q = 0$, the programs are functionally equivalent to the programs in Hybrid B, so we can invoke iO security to get from B to C_0. For $q = Q$, the experiment program is identical to that of Hybrid C. The challenge is to get from C_q to C_{q+1}, i.e., make the public key $\widetilde{\mathsf{pk}}_{q+1} = \mathsf{Enc}(\mathsf{ek}, q+1)$ count towards k.

One-at-a-time base switching. We carefully devise an inner sequence of hybrids to get from C_q to C_{q+1}. We start by puncturing the PRF key K at $\widetilde{\mathsf{pk}}_{q+1}$. Clearly, the punctured key K' suffices to compute the shared key of any list that does not contain $\widetilde{\mathsf{pk}}_{q+1}$. Let $x = g^{F_K(\widetilde{\mathsf{pk}}_{q+1})}$. While hardcoding x into our obfuscated programs enables computing shared keys for lists that contain $\widetilde{\mathsf{pk}}_{q+1}$, it obstructs our base-switching steps, but this can be remedied by utilizing our knowledge of the discrete logs of the v_k values and changing the computation slightly. Instead, let $z_k = x^{\alpha_k}$ for each $k \in [N]$. The punctured key K' and the group elements $z_0, \ldots, z_{N-1}$ suffice to compute the shared key for any set U that contains $\widetilde{\mathsf{pk}}_{q+1}$:

- The program computes $\Gamma' = \prod_{\mathsf{pk} \in U \setminus \{\widetilde{\mathsf{pk}}_{q+1}\}} F_{K'}(\mathsf{pk})$, and outputs $z_k^{\Gamma'}$. By the choice of z_k and the commutativity of $\mathbb{Z}_p$, we have $z_k^{\Gamma'} = v_k^{\prod_{\mathsf{pk} \in U} F_K(\mathsf{pk})}$.

If only we could switch z_k from x^{α_k} to $x^{\alpha_{k+1}}$, then $z_k^{\Gamma'} = v_{k+1}^{\prod_{\mathsf{pk} \in U} F_K(\mathsf{pk})}$ and we would be in a good spot to move on to Hybrid C_{q+1}.

The idea is to use DDH to switch each z_k, one at a time. Note that the exponents α_k and $F_K(\widetilde{\mathsf{pk}}_{q+1})$ do not appear "in the clear" in the adversary's view — instead they only appear in the exponent — so intuitively, we can switch z_k from $g^{F_K(\widetilde{\mathsf{pk}}_{q+1}) \cdot \alpha_k}$ to g^r where r is random, by relying on PPRF security and DDH. Next, we want to switch z_k from g^r to $g^{F_K(\widetilde{\mathsf{pk}}_{q+1}) \cdot \alpha_{k+1}}$. However, the second DDH reduction would fail if the value $g^{F_K(\widetilde{\mathsf{pk}}_{q+1}) \cdot \alpha_{k+1}}$ already appears in the adversary's view in another form: namely, this value is exactly z_{k+1}, at least at the beginning. As a useful metaphor, think of z_k as a car that wants to park at the spot $g^{F_K(\widetilde{\mathsf{pk}}_{q+1}) \cdot \alpha_{k+1}}$, but the spot is already occupied by another car z_{k+1}. To overcome this problem, it is crucial we switch the values $z_0, \ldots, z_{N-1}$ from *right to left*.



This guarantees that when we handle z_k, the value of z_{k+1} is already switched from $g^{F_K(\widetilde{\mathsf{pk}}_{q+1})\cdot\alpha_{k+1}}$ to $g^{F_K(\widetilde{\mathsf{pk}}_{q+1})\cdot\alpha_{k+2}}$, which frees up "the parking spot". Note that the first car we handle z_{N-1} wants to move to the spot $g^{F_K(\widetilde{\mathsf{pk}}_{q+1})\cdot\alpha_N}$, which is not occupied by any other car.

The proof technique for this step is one of the crucial differentiating features from prior work [KWZ22], and is exactly why we are able to avoid using a q-type assumption.

Step 4: Base cleanup. While the extra bases do not hurt, what we really care about is just to have a distinct base for fully labeled lists. Using a similar proof strategy to the one used before, we can "merge" the cases of $k \in [0, N-1]$ to use one base v, and keep the case of $k = N$ separately using another distinct base u. We call this Hybrid D. All that remains to be done is to remove u from the adversary's view.

By inspecting the CRS program in D further, we observe the following: the base u is only used when all keys in U are labeled. But the first line in the program only passes if there exists at least one key in U that is also in the image of the PRG. As we stated before, with overwhelming probability, no labeled key is in the image of the PRG, which means that the CRS program never uses the base u. This allows us to safely remove u by appealing to $i\mathcal{O}$ security. Call this Hybrid E. If there were no dynamic corruption queries, we would already be done. But since there are, we still need to remove u from the corrupted secret keys.

Step 5: Handling dynamic corruptions. This step closely follows how the prior work of Koppula et al. [KWZ22] handle constrained key queries in their construction of adaptively secure symbol-fixing PRFs.

The high level idea is that each distinct corrupted key will use its own distinct base. However, evaluations using newly corrupted keys should still be consistent with older corrupted keys. To ensure this consistency, each corrupted secret key will have a list of the previously corrupted users, and their respective bases. In more detail, let $\mathsf{sk}'_1, ..., \mathsf{sk}'_C$ be the corrupted keys in the order they were *corrupted* (not the order they were created), let $\mathsf{pk}'_1, \ldots, \mathsf{pk}'_C$ be the corresponding public keys, and finally let $w_1, \ldots, w_C$ be the corresponding bases. For any $c \in [C]$, the obfuscated program sk'_c has its own base w_c hardcoded in it, as well as all of the corrupted public keys *up to this point* $\mathsf{pk}'_1, \ldots, \mathsf{pk}'_{c-1}$ and their corresponding bases $w_1, \ldots, w_{c-1}$. When evaluating the shared key for a list U, the program sk'_c looks for the minimal j such that $\mathsf{pk}'_j \in U$. That is, the first corrupted public key that appears in the list. It then uses the base w_j corresponding to that public key. Note that $\mathsf{pk}'_c \in U$ (otherwise the program aborts), so it always finds an appropriate base to use. This strategy allows us to bypass a meta-reduction attack formalized by Rao [Rao14], but bounds the number of dynamic corruptions we can support. Namely, the last corrupted key has to be big enough to fit all of the corrupted public keys in it, which is why the secret keys grow with the number of dynamic corruptions.

Crucial to this strategy, the corrupted secret keys are dynamically computed at the time of corruption. To make this work, we first switch to a semantically identical Hybrid F in which the corrupted secret keys are lazily computed (and the uncorrupted secret keys are never computed), and the challenge query is answered directly using the base u and the PPRF F_K. The mechanics of introducing the new bases are similar to the base switching technique that got us from B to C: At first, the secret keys all use the two bases v, u. Next, we start introducing one new base w_c at a time by stepping through an inner sequence of hybrids. For each such hybrid, the challenger first guesses the cth user that will be corrupted. This guess comes with only inverse polynomial loss because we are only making one guess at a time. From here on, we do the same puncturable security and DDH reduction to change the base for sets that include pk'_c (but not any earlier corrupted key) from u to this new base w_c. Specifically, this allows us to remove the base u from sk'_c since this program requires the set to include pk'_c. Eventually, after adding a new base for each corruption

query, we get to an experiment G where all corrupted secret keys do not have the base u. At this point, for any Γ, the value u^{Γ} is statistically close to a uniform element in $\mathbb{G}$, since nothing else depends on u, which proves that the adversary's advantage in Hybrid G is 0. This completes the security proof of our DDH-based NIKE scheme.

2.2 Adaptive Multiparty NIKE from $i\mathcal{O}$ and RLWE

To adapt our previous construction to the lattice world, we should first understand the structure and mechanisms supported by DDH that enabled the construction in the first place:

- **The objects:** The group setting provides two types of objects: bases (taken from the group $\mathbb{G}$) and exponents (taken from $\mathbb{Z}_p$).
- **The operations:** Any exponent can be *composed* with any base in order to derive a new base. Additionally, exponents can be *accumulated* together first, and composed with the base later on.
- **Algebraic structure and commutativity:** We heavily rely on the commutativity of accumulating exponents. Given a set of exponents, this allows us to first compose the base with any exponent in the set, and only later compose the result with the rest of the accumulated exponents.
- **Base switching:** Our proof strategy relied on DDH (alongside PPRF security) to replace the composition of a uniform base and a uniform exponent, with a new uniform and independent base.

Our goal is to adapt the mentioned structure into the lattice setting. It is natural to look at Ring LWE (RLWE) due to the commutativity of "exponents". Consider the polynomial ring $R_q = \mathbb{Z}_q[X]/(X^d+1)$ with modulus q and dimension d which we set later.

- **The objects:** The exponents will be replaced with short polynomials in R_q, e.g., think of a polynomial with binary coefficients. The base domain is all of R_q.
- **Operations and commutativity:** Both composition and accumulation are simple polynomial multiplications in the ring. To compose a short γ with a base v, simply multiply $\gamma \cdot v \mod q$. To accumulate a sequence $\gamma_1, \ldots, \gamma_N$, compute $\Gamma = \prod_{j=1}^{N} \gamma_j \mod q$. Commutativity follows by the commutativity of the ring.

Base switching via lattices. Base switching will rely on RLWE, but this requires adding noise without the adversary noticing. To do so, we will use an additional ring $R_p = \mathbb{Z}_p[X]/(X^d+1)$ for $p \ll q$, and we will add an "evaluation" step after the composition, which simply rounds the polynomial, coefficient-by-coefficient down to R_p. I.e., the output of the obfuscated program — even in the real construction — would typically look like this:

$$\left\lfloor v \cdot \prod_{j=1}^{N} \gamma_j \right\rceil_p$$

Intuitively, since v is uniform over $\mathbb{Z}_q$ and $\Gamma = \prod_{j=1}^{N} \gamma_j$ is invertible, then the product is also uniform and so it will be far from a rounding boundary with high probability. This means that if we add a small error polynomial, the rounded value will not change, thus allowing us to invoke $i\mathcal{O}$ security.

In more detail, in a typical base switching step, we hope to replace $v \cdot \gamma_i$ with $v \cdot \gamma_i + e$ without changing the rounded value. Namely, we hope to prove the following:

$$\left\lfloor (v \cdot \gamma_j + e) \cdot \prod_{j \neq i} \gamma_j \right\rceil_p = \left\lfloor v \cdot \prod_{j \in [N]}^{N} \right\rceil_p$$

By relying on e and $\gamma_1, \cdots, \gamma_N$ being short, and the fact that the group size N is bounded, we can set the modulus-to-noise ratio to be a large enough *sub-exponential* so that $e \cdot \prod_{j \neq i} \gamma_j$ is small and thus we get the desired equality for

any fixed sequence of $\gamma_1, \cdots, \gamma_N$:

$$\left\lfloor (v \cdot \gamma_j + e) \cdot \prod_{j \neq i} \gamma_j \right\rceil_p = \left\lfloor v \cdot \prod_{j \in [N]}^{N} \gamma_j + e \cdot \prod_{j \neq i} \gamma_j \right\rceil_p = \left\lfloor v \cdot \prod_{j \in [N]}^{N} \right\rceil_p$$

Next, considering that each γ_j is derived from a PPRF with domain of size $2^{3\lambda}$ — the number of possible public keys — we can union bound over all possible $2^{\lambda N}$ values for the sequence $\gamma_1, \cdots, \gamma_N$ to deduce functional equivalence of the obfuscated program for all inputs with overwhelming probability. We note that for this to work, we need to set the modulus q to be sufficiently large to absorb the loss from the party bound N.

This now puts us in a good spot to do an analogous strategy to the DDH construction: we make γ_i random using PPRF security, then invoke RLWE to replace $v \cdot \gamma_i + e$ with a uniform new base r. Using an analogous argument, we can replace r with $u \cdot \gamma_i + e$ for a new base u, then remove the error polynomial e without affecting the rounded value (with overwhelming probability). This concludes the main ideas for adapting the DDH construction to the lattice setting. We emphasize that although our strategy relied on sub-exponential ratio, we still only rely on standard polynomial hardness of RLWE.

2.3 Generalizing our Approach

At this point, it is clear that the DDH and RLWE constructions have a lot in common. By a deeper inspection, we conclude that it is possible to abstract out the useful properties of both DDH and RLWE that allow us to instantiate our construction. In our main body, we introduce a new abstraction called *Key-Accumulative PRFs* (KA-PRF) and show the following:

1. How to use polynomially secure KA-PRF and $i\mathcal{O}$ to construct adaptive multiparty NIKE.
2. How to instantiate KA-PRF from either DDH or RLWE. In fact, it is possible to instantiate it from cryptographic weak pseudorandom effective group actions [ADFMP20] as we show in Appendix A. This opens the door for the first construction of adaptive multiparty NIKE from polynomially secure $i\mathcal{O}$ and isogeny-based assumptions like CSIDH [CLM+18].

We elect to introduce the new abstraction for two main reasons: First, it is important that we understand the common structure of DDH/group actions and RLWE that enables our construction. Providing a clean cryptographic abstraction that captures those properties deepens this understanding, and makes it easier to adapt the construction in the future to other assumptions, like plain LWE. For example, it was easier to observe that KA-PRFs can be instantiated from group actions, rather than reason about the entire NIKE construction. Second, it is easy to see that if one writes the security proof of both the DDH and RLWE constructions, they will be highly similar with the base switching step as the main difference. Providing a cryptographic abstraction allows us to prove both constructions secure in a more condensed manner without compromising presentation nor formality. We present the new abstraction and its instantiations in Section 4.

3 Preliminaries

For any positive integer $n \in \mathbb{N}$, we denote $[n] := \{1, \ldots, n\}$. The security parameter will be denoted by λ. For a binary string $x \in \{0,1\}^*$, we denote the length of x by $|x|$. A function $\varepsilon(\lambda)$ is called negligible if for every positive constant c, we have $\varepsilon(\lambda) = o(\lambda^{-c})$. A function $f(\lambda)$ is polynomial if there exists a positive constant c such that $f(\lambda) = O(\lambda^c)$. We denote this by writing $f(\lambda) = \mathsf{poly}(\lambda)$. For any set S, when we write $x \xleftarrow{R} S$ we mean that x is sampled uniformly from S. Given two Boolean circuits $C_0, C_1 : \{0,1\}^n \to \{0,1\}^m$, we say that C_0 and C_1 are functionally equivalent if $C_0(x) = C_1(x)$ for all $x \in \{0,1\}^n$.

We begin by defining multiparty non-interactive key exchange (NIKE) with a trusted setup. We allow the NIKE setup to take in a bound C on the number of dynamic honest user corruptions it can support, and allow the parameters to grow polynomially with C.

Definition 3.1 (Multiparty Non-interactive Key Exchange)**.** A multiparty non-interactive key exchange scheme (NIKE) is a tuple of polynomial time algorithms (Setup, Publish, KeyGen) with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^C, 1^N) \to \mathsf{crs}$: On input a security parameter λ, a set size N, and a bound on the number of corruptions C, the setup algorithm outputs a common reference string crs.
- $\mathsf{Publish}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$: On input a common reference string crs, the key publishing algorithm outputs a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Eval}(\mathsf{crs}, \mathsf{sk}, U, i) \to \mathsf{sh}$: On input crs, a secret key sk, a set (without repetitions) of public keys U of size N, and an index i, the evaluation algorithm outputs a shared key sh. This algorithm is deterministic and can run in time $\mathsf{poly}(\lambda, C, N)$.

The algorithms must satisfy the following properties:

- **Perfect correctness:** For all $\lambda, N, C \in \mathbb{N}$, and any $j, j' \in [N]$, we have

$$\Pr\left[\mathsf{Eval}(\mathsf{crs}, \mathsf{sk}_j, U, j) = \mathsf{Eval}(\mathsf{crs}, \mathsf{sk}_{j'}, U, j') \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^C, 1^N) \\ \forall i \in [N] : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Publish}(\mathsf{crs}) \\ U = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_N\} \end{array}\right] = 1.$$

In addition, a NIKE must satisfy the following security property:

- **Adaptive security:** For any adversary $\mathcal{A}$, we define the following security experiment parameterized by a security parameter λ and a bit $b \in \{0, 1\}$:

  1. On input 1^λ, adversary $\mathcal{A}$ sends $1^C, 1^N$ to the challenger. The challenger initializes an empty table T_{users} and a counter $i \leftarrow 0$, and sends $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^C, 1^N)$ to $\mathcal{A}$.
  2. $\mathcal{A}$ can make the following queries:
     - **Honest user registration:** The challenger increments $i \leftarrow i + 1$, runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Publish}(\mathsf{crs})$, adds the entry $(i, \mathsf{pk}, \mathsf{sk}, \text{uncorrupted})$ to the table T_{users}, and sends pk to $\mathcal{A}$.
     - **Honest user corruption:** The adversary sends an index i'. The challenger looks in T_{users} for an entry $(i', \mathsf{pk}, \mathsf{sk}, \text{uncorrupted})$, updates the entry to $(i', \mathsf{pk}, \mathsf{sk}, \text{corrupted})$ and sends sk to $\mathcal{A}$. If the entry is not found, then we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0, 1\}$.

     The adversary is allowed to make up to C **honest user corruption** queries.
  3. At any point, the adversary sends the challenge set (without repetitions) $\mathcal{I}^*$, such that for each $j \in \mathcal{I}^*$ there exists an entry $(j, \mathsf{pk}_j, \mathsf{sk}_j, \text{uncorrupted})$ in T_{users}. The challenger then computes the set $U^* = \{\mathsf{pk}_j\}_{j \in \mathcal{I}^*}$ and sends y_b, where $y_0 = \mathsf{Eval}(\mathsf{crs}, \mathsf{sk}_{j^*}, U^*, j^*)$ for $j^* = \min \mathcal{I}^*$ and y_1 is uniform over the range of the Eval. If there exists $j \in \mathcal{I}^*$ such that $(j, \mathsf{pk}_j, \mathsf{sk}_j, \text{uncorrupted})$ is not in T_{users}, then we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0, 1\}$.
  4. The adversary outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  We say that Π_{NIKE} satisfies adaptive security with dynamic corruptions if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Definition 3.2 (Indistinguishability Obfuscation [BGI+12, GGH+13])**.** An indistinguishability obfuscator $i\mathcal{O}$ is an efficient algorithm that takes as input the security parameter λ, a size bound s, and a Boolean circuit $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ of size at most s, and outputs a circuit C'. An $i\mathcal{O}$ scheme should satisfy the following properties:

- **Functionality-preserving:** For all security parameters $\lambda \in \mathbb{N}$, all Boolean circuits $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$, and all inputs $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, we have that $C'(x) = C(x)$ where $C' \leftarrow i\mathcal{O}(1^\lambda, 1^s, C)$.

- **Security:** We say that $i\mathcal{O}$ is secure if for all polynomial time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all polynomials $s := s(\lambda)$, and all functionally equivalent Boolean circuits C_0, C_1 of size at most s, we have
$$\left|\Pr[\mathcal{A}(i\mathcal{O}(1^\lambda, 1^s, C_0)) = 1] - \Pr[\mathcal{A}(i\mathcal{O}(1^\lambda, 1^s, C_1)) = 1]\right| \leq \mathsf{negl}(\lambda),$$

Definition 3.3 (Multi-Challenge iO Security)**.** For an indistinguishability obfuscator $i\mathcal{O}$, we define the security experiment $\mathsf{Exp}^b_{i\mathcal{O},\mathcal{A}}(\lambda)$ parameterized by a bit $b \in \{0,1\}$ as follows:

1. The adversary $\mathcal{A}$ on input 1^λ, can adaptively make a polynomial number of **Obfuscation queries**. For the i-th query:
   (a) $\mathcal{A}$ sends an integer in unary 1^{s_i} and pair of Boolean circuits $(C_{i,0}, C_{i,1})$ of size at most s_i to the challenger.
   (b) The challenger checks if the circuits are of the same size ($|C_{i,0}| = |C_{i,1}|$) and are functionally equivalent ($C_{i,0}(x) = C_{i,1}(x)$ for all inputs x). If either condition fails, the challenger aborts the experiment.
   (c) The challenger computes the obfuscated program $P_i \leftarrow i\mathcal{O}(1^\lambda, 1^{s_i}, C_{i,b})$ and sends P_i to $\mathcal{A}$.
2. At any point, the adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

We say that $i\mathcal{O}$ satisfies *multi-challenge security* if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that
$$\left|\Pr[\mathsf{Exp}^0_{i\mathcal{O},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^1_{i\mathcal{O},\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Remark 3.4. Definition 3.3 follows from standard iO security via a hybrid argument.

Definition 3.5 (Pseudorandom Generator)**.** Let $\ell_{\mathsf{in}}(\lambda)$ and $\ell_{\mathsf{out}}(\lambda)$ be any polynomials such that $\ell_{\mathsf{in}}(\lambda) < \ell_{\mathsf{out}}(\lambda)$. A pseudorandom generator (PRG) on domain $\mathcal{X} = \left\{\{0,1\}^{\ell_{\mathsf{in}}(\lambda)}\right\}_{\lambda\in\mathbb{N}}$ and range $\mathcal{Y} = \left\{\{0,1\}^{\ell_{\mathsf{out}}(\lambda)}\right\}_{\lambda\in\mathbb{N}}$ is a deterministic polynomial-time algorithm $G : \mathcal{X} \to \mathcal{Y}$. We say that G is a secure PRG if for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that the following holds:
$$\left|\Pr\left[\mathcal{A}(1^\lambda, G(s)) = 1 : s \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{in}}(\lambda)}\right] - \Pr\left[\mathcal{A}(1^\lambda, x) = 1 : x \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}\right]\right| \leq \mathsf{negl}(\lambda).$$

By a simple hybrid argument, we can extend the definition to support any polynomial number of PRG samples. We define the property using an experiment $\mathsf{Exp}^b_{\mathsf{PRG},\mathcal{A}}(\lambda)$ parameterized by a bit $b \in \{0,1\}$:

1. The adversary $\mathcal{A}$ on input 1^λ, can adaptively make a polynomial number of sample queries. For the ith query, the challenger sends x_i to $\mathcal{A}$, where x_i is sampled as follows:
   (a) If $b = 0$, the challenger samples $s_i \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{in}}(\lambda)}$ and sets $x_i = G(s_i)$.
   (b) If $b = 1$, the challenger samples $x_i \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$.
2. At any point, the adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

We say that G is a secure PRG if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that
$$\left|\Pr[\mathsf{Exp}^0_{\mathsf{PRG},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^1_{\mathsf{PRG},\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Definition 3.6 (Puncturable PRF [BW13, KPTZ13, BGI14])**.** A puncturable pseudorandom function consists of a tuple of efficient algorithms $\Pi_{\mathsf{PPRF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Puncture})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to K$: On input a security parameter λ, input size ℓ_{in}, and an output size ℓ_{out}, the key generation algorithm outputs a key K. We assume the key contains an implicit description of ℓ_{in} and ℓ_{out}.
- $\mathsf{Puncture}(K, x) \to K^{(x)}$: On input a key K and a point $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, the puncture algorithm outputs a punctured key $K^{(x)}$. We assume the punctured key contains an implicit description of ℓ_{in} and ℓ_{out}.

- $\mathsf{Eval}(K, x) \to y$: On input a key K and an input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, the evaluation algorithm outputs a value $y \in \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\perp\}$.

In addition, Π_{PPRF} should satisfy the following properties:

- **Functionality-preserving:** For all $\lambda, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, every $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$, and every $x \neq x^*$,

$$\Pr\left[\mathsf{Eval}(K,x) = \mathsf{Eval}\left(K^{(x^*)}, x\right) : \begin{array}{c} K \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \\ K^{(x^*)} \leftarrow \mathsf{Puncture}(K, x^*) \end{array}\right] = 1$$

- **Punctured pseudorandomness:** For a bit $b \in \{0,1\}$ and a security parameter λ, we define the (selective) punctured pseudorandomness game between an adversary $\mathcal{A}$ and a challenger as follows:
  - On input a parameter 1^λ, the adversary outputs an input size 1^ℓ_{in}, and an output size 1^ℓ_{out}, and commits to a point $x^* \in \{0,1\}^{\ell_{\mathsf{in}}}$.
  - The challenger samples a key $K \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell_{\mathsf{in}}, 1^\ell_{\mathsf{out}})$ and gives the punctured key $K^{(x^*)} \leftarrow \mathsf{Puncture}(K, x^*)$ to $\mathcal{A}$.
  - If $b = 0$, then the challenger sets $y^* = \mathsf{Eval}(K, x^*)$. Otherwise it samples $y^* \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$. In any case, the challenger gives y^* to $\mathcal{A}$.
  - The adversary outputs a guess $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{PPRF} satisfies punctured pseudorandomness if for all polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \mathsf{negl}(\lambda)$$

Next, we describe the notion of a *pseudorandom deterministic encryption* (PDE), which can be constructed from pseudorandom functions. We note that a stronger *puncturable* version of PDE due to [KPW17] (which can be constructed from puncturable PRFs) was used in [KWZ22] to construct adaptively secure NIKE for bounded parties. For our construction, the unpunctured notion suffices.

Definition 3.7 (Pseudorandom Deterministic Encryption)**.** A pseudorandom deterministic encryption (PDE) with message space $\{0,1\}^\lambda$ and ciphertexts of length $\ell := \ell(\lambda)$ is a tuple of efficient algorithms $\Pi_{\mathsf{PDE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda) \to \mathsf{ek}$: On input a security parameter λ, the setup algorithm outputs a key ek.
- $\mathsf{Enc}(\mathsf{ek}, m) \to \mathsf{ct}$: On input a key ek and a message $m \in \{0,1\}^\lambda$, the encryption algorithm outputs a ciphertext $\mathsf{ct} \in \{0,1\}^\ell$. This algorithm is deterministic.
- $\mathsf{Dec}(\mathsf{ek}, \mathsf{ct}) \to m$: On input a key ek and a ciphertext $\mathsf{ct} \in \{0,1\}^\ell$, outputs a message $m \in \{0,1\}^\lambda$ or $\perp$.

In addition, Π_{PDE} must satisfy the following properties:

- **Strong Correctness:** For all $\lambda \in \mathbb{N}$, all keys in the support of $\mathsf{Gen}(1^\lambda)$, all $m \in \{0,1\}^\lambda$ and all ciphertexts $\mathsf{ct} \in \{0,1\}^\ell$:

$$\mathsf{Dec}(\mathsf{ek}, \mathsf{ct}) = m \Leftrightarrow \mathsf{Enc}(\mathsf{ek}, m) = \mathsf{ct}$$

- **Ciphertext pseudorandomness:** We define the following security game which is parameterized by a bit b:
  1. The challenger samples the key $\mathsf{ek} \leftarrow \mathsf{Gen}(1^\lambda)$.
  2. The adversary can make *unique* encryption queries. For each query m, the challenger responds with ct which is computed as follows:
     - If $b = 0$, the challenger samples $\mathsf{ct} \xleftarrow{R} \{0,1\}^\ell$.

– If $b = 1$, the challenger sets $\mathsf{ct} = \mathsf{Enc}(\mathsf{ek}, m)$.

3. The adversary outputs a guess $b' \in \{0, 1\}$ which is the output of the experiment.

We say that Π_{PDE} satisfies ciphertext pseudorandomness if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

4 Key-Accumulative Pseudorandom Functions

In this section, we formalize the notion of a *Key-Accumulative Pseudorandom Function* (KA-PRF). This abstraction captures the salient properties of DDH and RLWE that are used in our construction. In a nutshell, we define a pseudorandom function that allows composition: given two PRF keys γ_1, and the evaluation $F_{\gamma_2}(x)$ of *another* key γ_2 on an input x, it is possible to compute the evaluation $F_{\gamma_1 \cdot \gamma_2}(x)$ of the *accumulated key* $\gamma_1 \cdot \gamma_2$ on x. We also require a commutativity property stating that the order of which keys are applied does not matter. This is possible for DDH simply by exponentiation.

For RLWE, it is a bit more complicated. The typical PRF from lattices employs rounding [BPR12]: the key is an RLWE secret s, the point a is a polynomial in the large ring R_q. The evaluation of s on a is the product rounded to the small ring $F_s(a) = \lfloor s \cdot a \rceil_p \in R_p$. But once a polynomial is rounded to R_p, it is not clear how to compose with the "next key" which lives in R_q. The natural solution is to break the PRF evaluation into two steps:

- Multiplication, which we generically call *composition*. After the multiplication, the polynomials still live in the large ring R_q, allowing us to compose further.
- Rounding, which we generically call *evaluation*. This step rounds the product down to R_p, after which no further compositions are possible.

Finally, both the group and the lattice settings allow the keys to be accumulated first, and then applied to the input. We now formally define our abstraction. Later in this section, we instantiate it from DDH or RLWE. In Appendix A, we instantiate it from cryptographic group actions.

Definition 4.1 (Key-Accumulative Pseudorandom Function)**.** A Key-Accumulative Pseudorandom Function (KA-PRF) consists of a tuple of polynomial-time algorithms (Setup, KeyGen, SampleIn, Compose, Eval, Accumulate, Noisy) with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{\lambda_s}, 1^N) \to \mathsf{pp}$: Takes a security parameter λ, a correctness error parameter λ_s, and a maximum composition bound N, and outputs public parameters pp. The public parameters define a key space $\mathcal{K}$, a domain space $\mathcal{X}$, and a codomain $\mathcal{Y}$.
- $\mathsf{KeyGen}(\mathsf{pp}) \to \gamma$: Outputs an evaluation key $\gamma \in \mathcal{K}$.
- $\mathsf{SampleIn}(\mathsf{pp}) \to x$: Outputs a uniformly random element $x \in \mathcal{X}$ in the domain.
- $\mathsf{Compose}(\mathsf{pp}, \gamma, x) \to x'$: A deterministic algorithm that applies the key $\gamma \in \mathcal{K}$ to the input $x \in \mathcal{X}$ and outputs a new domain element $x' \in \mathcal{X}$.
- $\mathsf{Eval}(\mathsf{pp}, x') \to y$: A deterministic algorithm that takes as input a domain element $x' \in \mathcal{X}$ and outputs an element $y \in \mathcal{Y}$ in the codomain.
- $\mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_{N'}) \to \Gamma$: A deterministic algorithm that takes as input up to N keys $\gamma_1, \ldots, \gamma_{N'} \in \mathcal{K}$ (where $N' \leq N$) and outputs a new accumulated key Γ.
- $\mathsf{Noisy}(\mathsf{pp}, x) \to \tilde{x}$: A probabilistic algorithm that applies noise to $x \in \mathcal{X}$ and outputs a noisy domain element $\tilde{x} \in \mathcal{X}$.

A KA-PRF scheme must satisfy the following properties:

1. **Composability:** Sequential application of keys via the compose algorithm is functionally equivalent to a single evaluation using the accumulated key. Formally, for any $x \in \mathcal{X}$ and any $\gamma_1, \gamma_2 \in \mathcal{K}$, it holds that:

$$\mathsf{Compose}(\mathsf{pp}, \gamma_2, \mathsf{Compose}(\mathsf{pp}, \gamma_1, x)) = \mathsf{Compose}(\mathsf{pp}, \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \gamma_2), x)$$

2. **Commutativity:** The order of accumulating keys does not affect the final accumulated key. Formally, for any integer $N' \leq N$, any sequence of keys $\gamma_1, \ldots, \gamma_{N'} \in \mathcal{K}$, and any permutation π over $[N']$, it holds that:

$$\mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_{N'}) = \mathsf{Accumulate}(\mathsf{pp}, \gamma_{\pi(1)}, \ldots, \gamma_{\pi(N')})$$

3. **Bijection:** For any valid key $\gamma \in \mathcal{K}$, the function $F_\gamma : \mathcal{X} \to \mathcal{X}$ defined as $F_\gamma(x) \coloneqq \mathsf{Compose}(\mathsf{pp}, \gamma, x)$ is a permutation (bijection) over the domain $\mathcal{X}$.

4. **Efficiency:** The bit-representation sizes of the elements in $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{K}$, as well as the running times of all algorithms, are bounded by $\mathsf{poly}(\lambda, \lambda_s, N)$.

5. **Approximate Evaluation Correctness:** For any integer $N' \leq N$ and any sequence of keys $\gamma_1, \ldots, \gamma_{N'}$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$, let $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_{N'})$. The probability that the final evaluation differs when evaluated on a domain element versus its noisy counterpart is negligible in λ_s over the uniform choice of the domain element and the randomness of the noise generation. Formally:

$$\forall N' \leq N, \forall \gamma_1, \ldots, \gamma_{N'} \in \mathsf{supp}(\mathsf{KeyGen}(\mathsf{pp})), \text{ let } \Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_{N'}) :$$

$$\Pr_{\substack{x \leftarrow \mathsf{SampleIn}(\mathsf{pp}) \\ \tilde{x} \leftarrow \mathsf{Noisy}(\mathsf{pp}, x)}} \left[\mathsf{Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, x)) \neq \mathsf{Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, \tilde{x})) \right] \leq 2^{-\lambda_s}.$$

6. **Weak Pseudorandomness:** We define the weak pseudorandomness experiment $\mathsf{Exp}^b_{\mathsf{KA-PRF}, \mathcal{A}}(\lambda)$ for a polynomial-time adversary $\mathcal{A}$ and a challenge bit $b \in \{0, 1\}$ as follows:

   (a) The challenger computes the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$ and samples a secret key $\gamma \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. It gives pp to $\mathcal{A}$.

   (b) The adversary $\mathcal{A}$ can adaptively make a polynomial number of queries of the following types:

   - **Sample query:**
     i. The challenger samples a uniform domain element $x \leftarrow \mathsf{SampleIn}(\mathsf{pp})$.
     ii. The challenger computes $z = \mathsf{Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x))$.
     iii. The challenger sends the pair (x, z) to the adversary $\mathcal{A}$.
   - **Challenge query:**
     i. The challenger samples a uniform domain element $x^* \leftarrow \mathsf{SampleIn}(\mathsf{pp})$.
     ii. The challenger computes the target element y^* as follows:
       – If $b = 0$, the challenger computes $z^* = \mathsf{Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x^*))$.
       – If $b = 1$, the challenger samples a uniformly random element $y^* \xleftarrow{R} \mathcal{X}$.
     iii. The challenger sends the pair (x^*, z^*) to the adversary $\mathcal{A}$.

     The adversary can make a single **Challenge query**.

   (c) At any point, the adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

   We say that the KA-PRF satisfies weak pseudorandomness if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\left| \Pr[\mathsf{Exp}^0_{\mathsf{KA-PRF}, \mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^1_{\mathsf{KA-PRF}, \mathcal{A}}(\lambda) = 1] \right| \leq \mathsf{negl}(\lambda).$$

7. **Evaluation Uniformity:** The evaluation algorithm evaluated on a uniformly random domain element yields a uniform distribution over the codomain $\mathcal{Y}$. Formally, the distribution of $\mathsf{Eval}(\mathsf{pp}, x)$, where $x \leftarrow \mathsf{SampleIn}(\mathsf{pp})$, is uniform over $\mathcal{Y}$.

5 Instantiating KA-PRFs from Standard Assumptions

In this section, we instantiate the abstraction from Theorem 4.1 using either of DDH or LWE.

5.1 Construction from DDH

We instantiate a KA-PRF using the Decisional Diffie-Hellman (DDH) assumption. We start by recalling the assumption.

Definition 5.1 (Decisional Diffie-Hellman Assumption)**.** Let GroupGen be a polynomial-time algorithm that, on input 1^λ, outputs a description of a cyclic group $\mathbb{G}$ of prime order p (where the bit-length of p is $\mathrm{poly}(\lambda)$) along with a generator g. We define the Decisional Diffie-Hellman security experiment $\mathsf{Exp}^b_{\mathsf{DDH},\mathcal{A}}(\lambda)$ for a polynomial-time adversary $\mathcal{A}$ and a challenge bit $b \in \{0,1\}$ as follows:

1. The challenger computes $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$.
2. The challenger samples uniformly random elements $u, v \xleftarrow{R} \mathbb{Z}_p$.
3. If $b = 0$, the challenger computes $w = u \cdot v \pmod p$.
4. If $b = 1$, the challenger samples a uniformly random element $w \xleftarrow{R} \mathbb{Z}_p$.
5. The adversary $\mathcal{A}$ is given $((\mathbb{G}, p, g), g^u, g^v, g^w)$ and outputs a guess b'.

We say that the DDH assumption holds for GroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr[\mathsf{Exp}^0_{\mathsf{DDH},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^1_{\mathsf{DDH},\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Construction 5.2 (KA-PRF from DDH)**.** Let λ be a security parameter. This construction achieves perfect evaluation correctness and allows unbounded composition, therefore the algorithms ignore λ_s, N. The algorithms are defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$: Runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Outputs the public parameters $\mathsf{pp} = (\mathbb{G}, p, g)$. The key space is $\mathcal{K} = \mathbb{Z}_p^*$, the domain and codomain spaces are $\mathcal{X} = \mathcal{Y} = \mathbb{G}$.
- $\mathsf{KeyGen}(\mathsf{pp})$: Samples and outputs $\gamma \xleftarrow{R} \mathbb{Z}_p^*$ uniformly at random.
- $\mathsf{SampleIn}(\mathsf{pp})$: Samples and outputs $x \xleftarrow{R} \mathbb{G}$ uniformly at random.
- $\mathsf{Compose}(\mathsf{pp}, \gamma, x)$: Outputs $x' = x^\gamma \in \mathbb{G}$.
- $\mathsf{Eval}(\mathsf{pp}, x')$: Outputs $y = x'$ (the evaluation step is trivial and does nothing).
- $\mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_{N'})$: Outputs $\Gamma = \prod_{i=1}^{N'} \gamma_i \pmod p$.
- $\mathsf{Noisy}(\mathsf{pp}, x)$: Outputs $\tilde{x} = x$ (the noisy step is trivial and adds no noise).

We briefly verify that Construction 5.2 satisfies the statistical and algebraic properties of Definition 4.1:

- **Composability and Commutativity:** Both follow from the standard rules of exponentiation and scalar multiplication modulo p: $(x^{\gamma_1})^{\gamma_2} = x^{\gamma_1\gamma_2} \pmod p$ and $\gamma_1\gamma_2 \equiv \gamma_2\gamma_1 \pmod p$.
- **Bijection:** For any $\gamma \in \mathbb{Z}_p^*$, the function $F_\gamma(x) = x^\gamma$ is a bijection over $\mathbb{G}$, as p is prime.
- **Approximate Evaluation Correctness:** Since the Noisy algorithm is deterministic and outputs its input identically ($\tilde{x} = x$), and Eval is the identity function, both sides of the equivalence evaluate to exactly x^Γ. The probability of discrepancy is 0, which trivially satisfies the required bound.

- **Evaluation Uniformity:** Since x is sampled uniformly from $\mathcal{X} = \mathbb{G}$ and Eval outputs x directly, the output is distributed perfectly uniformly over the codomain $\mathcal{Y} = \mathbb{G}$.

Lemma 5.3. *Assume the DDH assumption holds for* GroupGen. *Then Construction 5.2 satisfies weak pseudorandomness of Definition 4.1.*

Proof. We proceed by a reduction from the Decisional Diffie-Hellman (DDH) problem. Let $\mathcal{A}$ be an efficient adversary that breaks the weak pseudorandomness of the instantiated KA-PRF with advantage ε. We construct a PPT reduction $\mathcal{B}$ that uses $\mathcal{A}$ to solve the DDH problem (as defined in Definition 5.1) with the same advantage ε.

The reduction $\mathcal{B}$ operates as follows:

1. Algorithm $\mathcal{B}$ receives the DDH challenge tuple $((\mathbb{G}, p, g), A, B, C)$, where $A = g^u$, $B = g^v$, and $C = g^w$, from its DDH challenger.
2. Algorithm $\mathcal{B}$ forwards the public parameters $\mathsf{pp} = (\mathbb{G}, p, g)$ to $\mathcal{A}$. The KA-PRF secret key is implicitly chosen as $\gamma = u$.
3. When $\mathcal{A}$ requests a **Sample query**, algorithm $\mathcal{B}$ samples a uniform scalar $r \xleftarrow{R} \mathbb{Z}_p$ and computes the element $x_i = g^r \in \mathbb{G}$. It then computes the evaluation as $y_i = A^r$. Notice that $y_i = (g^u)^r = (g^r)^u = x_i^\gamma$. Algorithm $\mathcal{B}$ sends the pair (x_i, y_i) to $\mathcal{A}$.
4. When $\mathcal{A}$ requests a **Challenge query**, algorithm $\mathcal{B}$ sets the challenge domain element to be $x^* = B = g^v$, and sets the target element to be $y^* = C = g^w$. Algorithm $\mathcal{B}$ sends the pair (x^*, y^*) to $\mathcal{A}$.
5. Eventually, algorithm $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. Algorithm $\mathcal{B}$ outputs the same guess b' as its solution to the DDH challenge.

We now analyze the view of $\mathcal{A}$ simulated by $\mathcal{B}$. For the **Sample queries**, since r is chosen uniformly from $\mathbb{Z}_p$, the element $x_i = g^r$ is uniformly distributed in $\mathbb{G}$, perfectly matching the distribution of $\mathsf{SampleIn}(\mathsf{pp})$. The evaluation $y_i = A^r = x_i^\gamma$ perfectly matches the deterministic, exact evaluation of $\mathsf{Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x_i))$ in this instantiation.

For the **Challenge query**, the element $x^* = B = g^v$ is uniformly distributed in $\mathbb{G}$, matching the requirement for the challenge base. The simulation then depends on the value C:

- If $C = g^{uv} = (g^v)^u = (x^*)^\gamma$ then this perfectly simulates the real evaluation corresponding to the real evaluation in the KA-PRF game.
- If $C = g^w$ for a uniformly random $w \xleftarrow{R} \mathbb{Z}_p$, then y^* is a uniformly random element in $\mathbb{G}$. This perfectly simulates the random response to the challenge query in the KA-PRF game.

Since $\mathcal{B}$ perfectly simulates the weak pseudorandomness game for $\mathcal{A}$ in both cases, the advantage of $\mathcal{B}$ in the DDH game is exactly equal to the advantage of $\mathcal{A}$ in breaking the weak pseudorandomness of KA-PRF. Therefore, if the DDH assumption holds, the advantage ε must be negligible. □

This concludes the instantiation from DDH. In Section A, we show that this instantiation naturally extends to cryptographic group actions.

5.2 Construction from Ring-LWE

We instantiate a KA-PRF using the Ring Learning with Errors assumption. We start by recalling the assumption.

Definition 5.4 (Decisional Ring-LWE Assumption [LPR13])**.** Let λ be the security parameter, let $q := q(\lambda)$ be a (potentially superpolynomial) function of λ, let $d := d(\lambda)$ be a polynomial, let $R_q = \mathbb{Z}_q[X]/(X^d + 1)$ be a cyclotomic ring. Let χ_s be a secret distribution over R_q, and let χ_e be an error distribution over R_q. We define the Decisional Ring-LWE security experiment $\mathsf{Exp}^b_{\mathsf{RLWE},\mathcal{A}}(\lambda)$ for a polynomial-time adversary $\mathcal{A}$ and a challenge bit $b \in \{0, 1\}$ as follows:

1. The challenger initializes the experiment by sampling a secret polynomial $s \leftarrow \chi_s$.
2. The adversary $\mathcal{A}$ can adaptively make a polynomial number of queries of the following types:
   - **Sample query:**
     (a) The challenger samples a uniform ring element $a \xleftarrow{R} R_q$ and an error polynomial $e \leftarrow \chi_e$.
     (b) The challenger computes $y = a \cdot s + e \pmod q$.
     (c) The challenger sends the pair (a, y) to the adversary $\mathcal{A}$.
   - **Challenge query:**
     (a) The challenger samples a uniform ring element $a^* \xleftarrow{R} R_q$.
     (b) The challenger computes the target element y^* as follows:
       – If $b = 0$, the challenger samples an error polynomial $e^* \leftarrow \chi_e$ and computes $y^* = a^* \cdot s + e^* \pmod q$.
       – If $b = 1$, the challenger samples a uniformly random ring element $y^* \xleftarrow{R} R_q$.
     (c) The challenger sends the pair (a^*, y^*) to the adversary $\mathcal{A}$.
3. At any point, the adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

We say that the Ring-LWE assumption holds for the parameters (d, q, χ_s, χ_e) if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr[\mathsf{Exp}^0_{\mathsf{RLWE},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^1_{\mathsf{RLWE},\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Fact 5.5. Fix $q, d \in \mathbb{N}$ and a polynomial ring $R_q = \mathbb{Z}_q[X]/(X^d+1)$. For any $a, b \in R_q$, it holds that $|a \cdot b|_\infty \leq d \cdot |a|_\infty \cdot |b|_\infty$.

Construction 5.6 (KA-PRF from Ring-LWE)**.** Let λ be a security parameter, correctness error parameter λ_s, and N be an integer bound. We define the following parameters used in our construction:

- Let $q := q(\lambda, \lambda_s, N)$ and $p := p(\lambda, \lambda_s, N)$ be moduli, such that $p \ll q$ and q is a multiple of p, and let $d := d(\lambda, \lambda_s, N)$ be a dimension.
- Let $R_q = \mathbb{Z}_q[X]/(X^d + 1)$ and $R_p = \mathbb{Z}_p[X]/(X^d + 1)$ be polynomial rings. Denote by $\lfloor \cdot \rceil_p : R_q \to R_p$ the coordinate-wise rounding function.
- Let B_s, B_e be positive integers such that $B_s, B_e < q$, and let χ_s, χ_e be distributions over R_q such that for every s (resp. e) in the support of χ_s (resp. χ_e), we have $|s|_\infty \leq B_s$ (resp. $|e|_\infty \leq B_e$). We will require that every polynomial in the support of χ_s is invertible (this is not necessary for χ_e).

The specific parameters and distributions will be chosen towards the end of this section. The algorithms are defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$: Sets the ring dimension d and moduli q and p, which will be specified later. The distributions χ_s and χ_e are picked as described above. Outputs the public parameters $\mathsf{pp} = (d, q, p, \chi_s, \chi_e)$. The key space $\mathcal{K}$ is the set of invertible polynomials in R_q. The domain space is $\mathcal{X} = R_q$ and the codomain is $\mathcal{Y} = R_p$. In this instantiation, the setup algorithm is deterministic.
- $\mathsf{KeyGen}(\mathsf{pp})$: Samples and outputs $\gamma \leftarrow \chi_s$ (see Remark 5.9).
- $\mathsf{SampleIn}(\mathsf{pp})$: Samples and outputs $x \xleftarrow{R} R_q$ uniformly at random.
- $\mathsf{Compose}(\mathsf{pp}, \gamma, x)$: Outputs the multiplication $x' = \gamma \cdot x \pmod q$.
- $\mathsf{Eval}(\mathsf{pp}, x')$: Outputs the deterministic rounding $y = \lfloor x' \rceil_p$.
- $\mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_{N'})$: Outputs $\Gamma = \prod_{i=1}^{N'} \gamma_i \pmod q$.

- $\mathsf{Noisy}(\mathsf{pp}, x)$: Samples an error polynomial $e \leftarrow \chi_e$ and outputs $\tilde{x} = x + e \pmod q$.

We briefly verify that Construction 5.6 satisfies the statistical and algebraic properties of Definition 4.1:

- **Composability and Commutativity:** Both properties follow directly from the standard associativity and commutativity of polynomial multiplication in the cyclotomic ring R_q: $(\gamma_1 \cdot \gamma_2) \cdot x = \gamma_2 \cdot (\gamma_1 \cdot x) \pmod q$ and $\gamma_1 \cdot \gamma_2 = \gamma_2 \cdot \gamma_1 \pmod q$.
- **Bijection:** By definition, the key space $\mathcal{K}$ consists entirely of invertible polynomials in R_q. Therefore, the function $F_\gamma(x) = \Gamma \cdot x \pmod q$ for any $\gamma \in \mathcal{K}$ acts as a bijection over R_q.
- **Evaluation Uniformity:** The domain element x is sampled uniformly from $\mathcal{X} = R_q$. The evaluation algorithm Eval applies coordinate-wise rounding from $\mathbb{Z}_q$ to $\mathbb{Z}_p$. Since q is a multiple of p, this many-to-one mapping perfectly preserves the uniform distribution, yielding a uniform element over the codomain $\mathcal{Y} = R_p$.

Lemma 5.7. *Assume the Decisional Ring-LWE assumption holds with parameters (d, q, χ_s, χ_e) as defined by* Setup. *Then the instantiation above satisfies weak pseudorandomness of Definition 4.1.*

Proof. We proceed by a reduction from the Decisional Ring-LWE problem. Let $\mathcal{A}$ be an efficient adversary that breaks the weak pseudorandomness of the instantiated KA-PRF with advantage ε. We construct a PPT reduction $\mathcal{B}$ that uses $\mathcal{A}$ to solve the Decisional Ring-LWE problem (as defined in Definition 5.4) with the same advantage ε.

The reduction $\mathcal{B}$ operates as follows:

1. $\mathcal{B}$ receives the public parameters (d, q, χ_s, χ_e) from its Ring-LWE challenger, chooses p accordingly, sets the parameters $\mathsf{pp} = (d, q, p, \chi_s, \chi_e)$ for the KA-PRF, and forwards pp to $\mathcal{A}$.
2. When $\mathcal{A}$ requests a **Sample query**, $\mathcal{B}$ queries its own Ring-LWE challenger for a **Sample query**. It receives a pair (a, y), where $a \xleftarrow{R} R_q$ and $y = a \cdot s + e \pmod q$, and forwards this pair to $\mathcal{A}$ as the response (x, y).
3. When $\mathcal{A}$ requests a **Challenge query**, $\mathcal{B}$ queries its Ring-LWE challenger for a **Challenge query**. It receives a challenge pair (a^*, y^*) and forwards it to $\mathcal{A}$ as the response (x^*, y^*).
4. Eventually, $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. $\mathcal{B}$ outputs the same guess b' as its solution to the Decisional Ring-LWE challenge.

We now analyze the view of $\mathcal{A}$ simulated by $\mathcal{B}$. Notice that the short secret polynomial $s \leftarrow \chi_s$ in the Ring-LWE experiment perfectly corresponds to the secret key $\gamma \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ in the KA-PRF experiment.

For **Sample queries**, the Ring-LWE challenger always returns real evaluations $(a, a \cdot s + e \pmod q)$. Since $a \xleftarrow{R} R_q$, this is identically distributed to a properly sampled domain element $x \leftarrow \mathsf{SampleIn}(\mathsf{pp})$. The evaluation $a \cdot s + e \pmod q$ perfectly matches the definition of $\mathsf{Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x))$. Thus, $\mathcal{B}$ perfectly simulates the Sample queries.

For the **Challenge query**, the simulation depends on the Ring-LWE challenger's hidden bit b:

- If $b = 0$, the Ring-LWE challenger returns $y^* = a^* \cdot s + e^* \pmod q$. This perfectly simulates the real noisy evaluation $y^* = \mathsf{Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x^*))$ corresponding to the $b = 0$ case in the KA-PRF game.
- If $b = 1$, the Ring-LWE challenger returns a uniformly random element $y^* \xleftarrow{R} R_q$. This perfectly simulates the random evaluation corresponding to the $b = 1$ case in the KA-PRF game.

Since $\mathcal{B}$ perfectly simulates the weak pseudorandomness game for $\mathcal{A}$ in both cases, $\mathcal{B}$'s advantage in deciding the Ring-LWE problem is exactly equal to $\mathcal{A}$'s advantage in breaking the weak pseudorandomness. Therefore, if the Decisional Ring-LWE assumption holds, the advantage ε must be negligible. □

We prove the following stronger version of the approximate evaluation correctness property:

Lemma 5.8 (Approximate Evaluation Correctness). *Let $R_q = \mathbb{Z}_q[X]/(X^d+1)$ and $R_p = \mathbb{Z}_p[X]/(X^d+1)$ for moduli $p \ll q$. Fix any $\gamma_1, \ldots, \gamma_{N'}$ in the support of* $\mathsf{KeyGen}(\mathsf{pp})$*, and let* $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_{N'})$*. Let $e \in R_q$ be an error polynomial with $|e|_\infty \le B_e$. Assume the modulus q satisfies the following condition: $q \ge 2^{\lambda_s+1} p \cdot B_e \cdot d^{N+1} \cdot B_s^N$. Then*

$$\Pr_{z \xleftarrow{R} R_q} \left[\lfloor z \cdot \Gamma \rceil_p \neq \lfloor (z+e) \cdot \Gamma \rceil_p \right] \le 2^{-\lambda_s}.$$

Proof. Assume that z is uniformly distributed over R_q. Since Γ is invertible in R_q, the product $u = z \cdot \Gamma \pmod q$ is also uniformly distributed over R_q.

Let $e' = e \cdot \Gamma \pmod q$. Since Γ is the product of at most N polynomials each of which has norm bounded by B_s, we can appeal to Fact 5.5 to conclude that $|\Gamma|_\infty \le d^{N-1} B_s^N$. Again by Fact 5.5, we have

$$|e'|_\infty \le d \cdot |e|_\infty \cdot |\Gamma|_\infty \le B_e \cdot (d \cdot B_s)^N.$$

The rounding function $\lfloor \cdot \rceil_p$ divides the modulus q into p intervals. A rounding discrepancy $\lfloor u \rceil_p \neq \lfloor u + e' \rceil_p$ occurs if and only if the addition of e' pushes at least one coefficient of u across a rounding boundary. For any single coefficient of u, which is uniformly distributed in $\mathbb{Z}_q$, the number of elements in $\mathbb{Z}_q$ that are within distance $|e'|_\infty$ of any of the p boundaries is at most $p \cdot 2|e'|_\infty$. Therefore, the probability that a specific coefficient crosses a boundary is bounded by:

$$\frac{2p \cdot |e'|_\infty}{q} \le \frac{2p \cdot B_e \cdot (d \cdot B_s)^N}{q}.$$

Applying the union bound over all d coefficients of u, the probability that $\lfloor u \rceil_p \neq \lfloor u + e' \rceil_p$ for a fixed Γ is at most:

$$d \cdot \frac{2p \cdot B_e \cdot (d \cdot B_s)^N}{q} = \frac{2p \cdot B_e \cdot d^{N+1} \cdot B_s^N}{q}.$$

The claim now follows given the condition on the modulus $q \ge 2^{\lambda_s+1} p \cdot B_e \cdot d^{N+1} \cdot B_s^N$. □

Parameter instantiation. Let λ, λ_s be security parameters and N be an integer bound. We now provide one possible instantiation of the parameters in the abstraction. In the following, we assume that N is some polynomial in λ.

- We set the rounding modulus $p = 2$.
- For any q that is a power of 2, it holds that if a polynomial $\gamma \in R_q$ has coefficients that sum up to an odd number, then γ is invertible in R_q. As such, we can pick χ_s to be a discrete Gaussian such that the infinity norm is bounded by $B_s = \lambda \cdot \sqrt{d}$ (with overwhelming probability), conditioned on the coefficients summing to an odd number.
- Next, we set the ring dimension d to be a power of two such that $d = \tilde{\Omega}(\lambda \cdot N)$.
- We pick the error distribution χ_e over R_q as a discrete Gaussian parameterized to satisfy standard lattice reductions, ensuring the infinity norm is bounded by $B_e = \lambda \cdot \sqrt{d}$ with all but negligible probability.
- Finally, we choose the main modulus q to be a power of 2 satisfying $q \ge 2^{\lambda_s+1} p \cdot B_e \cdot d^{N+1} \cdot B_s^N$.

We briefly verify that these parameters satisfy the necessary requirements:

- For efficiency, the dimension d and the bit-length of the modulus $\log q$ are polynomials in λ and N. All sampling, ring multiplication, and rounding operations run in polynomial time.
- The setting of q explicitly satisfies the bound required for Lemma 5.8.
- We require the Decisional Ring-LWE assumption to hold in R_q. The bit-length of the modulus is $\log q = O(\lambda \cdot N \log d)$. Setting $d = \tilde{\Omega}(\lambda \cdot N)$ ensures the modulus-to-noise ratio is small enough that the Ring-LWE problem remains hard against known lattice reduction algorithms.

Remark 5.9 (Sampling χ_s from Bounded Randomness)**.** The distribution χ_s above is an idealized discrete Gaussian. In practice, KeyGen instead samples γ from a distribution χ_s' produced by an efficient sampler that consumes a fixed number of random bits (e.g., [GPV08]), with statistical distance $\Delta(\chi_s, \chi_s') \leq \mathsf{negl}(\lambda)$. Since statistical distance upper-bounds the advantage of any distinguisher, Lemma 5.7 and Lemma 5.8 continue to hold verbatim with $\gamma \leftarrow \chi_s'$ in place of $\gamma \leftarrow \chi_s$, up to this additional negligible additive term.

6 NIKE Construction

In this section, we present a construction of adaptive multiparty NIKE from KA-PRF, iO, and one-way functions.

Construction 6.1 (NIKE from KA-PRF and iO)**.** Let λ be a computational security parameter, λ_s be a statistical security parameter, N be a party size parameter, and C be a corruption bound parameter. Our construction uses the following ingredients:

- An indistinguishability obfuscator $i\mathcal{O}$ as per Definition 3.2.
- A pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ as per Definition 3.5.
- A pseudorandom deterministic encryption scheme $\Pi_{\mathsf{PDE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as per Definition 3.7 with ciphertexts of length 3λ. This ingredient only appears in the security analysis, not in the construction itself.
- A key-accumulative pseudorandom function Π_{kaPRF} as per Definition 4.1. We denote the domain space by $\mathcal{X}$, the codomain space by $\mathcal{Y}$, and the key space by $\mathcal{K}$.
- A puncturable PRF scheme $\Pi_{\mathsf{PPRF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Puncture})$ as per Definition 3.6. We will use the PPRF with output length ℓ_{out} that matches the number of random bits required by kaPRF.KeyGen, i.e., there will be a mapping from the PPRF output to the base key space $\mathcal{K}$. For any $\gamma \in \{0,1\}^{\ell_{\mathsf{out}}}$, we abuse notation and write $\gamma \in \mathcal{K}$ to mean $\mathsf{kaPRF.KeyGen}(\mathsf{pp}; \gamma)$.

The algorithms are defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^C, 1^N)$:
  1. Samples public parameters $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$, where $\lambda_s = \lambda + 3\lambda N$.
  2. Samples a PPRF key $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
  3. Samples a uniformly random domain element $v \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$.
  4. Let $\mathsf{size}_{\mathsf{crs}} := \mathsf{size}_{\mathsf{crs}}(\lambda, N)$ be the maximum size of all the programs from Figs. 4, 7, 9, 11, 13, 15, 17, 19 and 23. Note that $\mathsf{size}_{\mathsf{crs}}$ depends on λ and N.
  5. Samples $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K, v])$ where crsProg is described in Figure 4.
  6. Outputs $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$.

Constants: KA-PRF parameters pp, a PPRF key K, and a domain element $v \in \mathcal{X}$.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
4. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
5. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.

Figure 4: Program $\mathsf{crsProg}[\mathsf{pp}, K, v]$.

- Publish(crs):
  1. Parses $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$.
  2. Samples a uniform seed $s \xleftarrow{R} \{0,1\}^\lambda$.
  3. Sets $\mathsf{pk} = G(s)$.
  4. Let $\mathsf{size}_{\mathsf{sk}} := \mathsf{size}_{\mathsf{sk}}(\lambda, N, C)$ be the maximum size of all the programs from Figs. 5, 6, 8, 10, 12, 14, 16, 18, 20 to 22, 24 and 25. Note that $\mathsf{size}_{\mathsf{sk}}$ depends on λ, N, and C.
  5. $\mathsf{sk} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}[P_{\mathsf{crs}}, s])$ where skProg is described in Figure 5
  6. Outputs $(\mathsf{pk}, \mathsf{sk})$.

Constants: A program P_{crs}, and a seed s.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. Outputs $P_{\mathsf{crs}}(s, U, i)$.

Figure 5: Program $\mathsf{skProg}[P_{\mathsf{crs}}, s]$.

- KeyGen(crs, sk, U, i):
  1. Parses sk as a program and outputs $\mathsf{sk}(U, i)$.

6.1 Security Analysis

Theorem 6.2. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator, Π_{PPRF} satisfies punctured security, G is a secure PRG, Π_{PDE} satisfies ciphertext pseudorandomness, and the underlying Key-Accumulative Pseudorandom Function satisfies weak pseudorandomness, approximate evaluation correctness, and evaluation uniformity. Then Construction 6.1 is an adaptively secure NIKE for parties of size N.*

Fix an efficient adversary $\mathcal{A}$. Throughout the analysis, we denote by Q (an upper bound on) the number of **Honest user registration** queries that $\mathcal{A}$ makes.

- Real^b: This is the real security game with parameter b.
  1. The setup phase:
     (a) On input 1^λ, the adversary $\mathcal{A}$ sends 1^N and 1^C to the challenger.
     (b) The challenger initializes an empty table $T_{\mathsf{users}} \leftarrow \emptyset$ and a counter $i \leftarrow 0$.
     (c) Computes $\lambda_s = \lambda + 3\lambda N$ and samples KA-PRF parameters $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$.
     (d) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
     (e) Samples a random domain element $v \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$.
     (f) Samples $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K, v])$ where crsProg is described in Figure 4.
     (g) The challenger sends $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$ to $\mathcal{A}$.
  2. The adversary $\mathcal{A}$ can adaptively make the following queries:
     – **Honest user registration:**
       (a) The challenger increments $i \leftarrow i + 1$.
       (b) The challenger samples a uniform seed $s_i \xleftarrow{R} \{0,1\}^\lambda$ and sets $\widetilde{\mathsf{pk}}_i = G(s_i)$.
       (c) The challenger computes $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}[P_{\mathsf{crs}}, s_i])$ where skProg is described in Figure 5.

(d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \text{uncorrupted})$ to the table T_{users}.

(e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

– **Honest user corruption:**

(a) The adversary sends an index i'.

(b) The challenger looks for the entry $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \text{uncorrupted})$ in T_{users}.

(c) If no such entry exists, outputs $\perp$.

(d) The challenger updates the entry to $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \text{corrupted})$.

(e) The challenger sends $\widetilde{\mathsf{sk}}_{i'}$ to $\mathcal{A}$.

The adversary is allowed to make up to C **Honest user corruption** queries.

3. At any point, $\mathcal{A}$ sends the challenge set of indices $\mathcal{I}^*$ of size N. The challenger responds as follows:

(a) If for any index $j \in \mathcal{I}^*$ there does not exist an entry $(j, \widetilde{\mathsf{pk}}_j, \widetilde{\mathsf{sk}}_j, \text{uncorrupted})$ in T_{users}, then the experiment aborts and outputs a random bit $b' \xleftarrow{R} \{0,1\}$.

(b) The challenger retrieves $\widetilde{\mathsf{pk}}_j$ for all $j \in \mathcal{I}^*$ from T_{users} and forms the ordered set $U^* = (\widetilde{\mathsf{pk}}_j)_{j \in \mathcal{I}^*}$.

(c) The challenger computes the challenge value y_b as follows:

– If $b = 0$, the challenger picks $j^* = \min \mathcal{I}^*$ and computes $y_0 = \widetilde{\mathsf{sk}}_{j^*}(U^*, 1)$.

– If $b = 1$, the challenger samples a uniformly random codomain element $y_1 \xleftarrow{R} \mathcal{Y}$.

(d) The challenger sends y_b to $\mathcal{A}$.

4. At any point, the adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

- A^b: This is the same as Real^b, except that the challenger generates the secret keys for honest users using a modified program that hardcodes the public key and checks that it is contained in the set, instead of hardcoding the secret string s and checking that its image is in the set. Specifically:

2. The challenger modifies how it responds to **Honest user registration** queries as follows:

(a) The challenger increments $i \leftarrow i + 1$.

(b) The challenger samples a uniform seed $s_i \xleftarrow{R} \{0,1\}^\lambda$ and sets $\widetilde{\mathsf{pk}}_i = G(s_i)$.

(c) The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, v, \widetilde{\mathsf{pk}}_i])$ where skProg' is described in Figure 6.

(d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \text{uncorrupted})$ to the table T_{users}.

(e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

Constants: KA-PRF parameters pp, a PPRF key K, a domain element $v \in \mathcal{X}$, and a public key pk.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
4. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
5. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.

Figure 6: Program $\mathsf{skProg}'[\mathsf{pp}, K, v, \mathsf{pk}]$.

- B^b: This is the same as A^b, except that the challenger samples the public keys for honest users as labeled keys using a PDE scheme and a one-time pad. Specifically:

1. During the setup, the challenger additionally samples a PDE key $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and a uniform string $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
2. The challenger modifies how it responds to **Honest user registration** queries as follows:
   (a) The challenger increments $i \leftarrow i+1$.
   (b) The challenger sets $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
   (c) The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, v, \widetilde{\mathsf{pk}}_i])$ where skProg' is described in Figure 6.
   (d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \text{uncorrupted})$ to the table T_{users}.
   (e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

- C^b: This is the same as B^b, except that the setup samples $N+1$ domain elements $v_0, \ldots, v_N \in \mathcal{X}$, and the programs are modified to use a domain element v_k where k is the number of labeled keys in the input set. Specifically:

1. The setup phase is modified to work as follows:
   (a) Samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^N)$.
   (b) Samples $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
   (c) Samples $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
   (d) Samples domain elements $v_0, \ldots, v_N \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$, and denote $\mathbf{v} = (v_0, \ldots, v_N)$.
   (e) Samples $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}])$ where crsProg is described in Figure 7.
   (f) Outputs $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$.
2. The challenger modifies how it responds to **Honest user registration** queries as follows:
   (a) The challenger increments $i \leftarrow i+1$.
   (b) The challenger sets $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
   (c) The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i])$ where skProg' is described in Figure 8.
   (d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \text{uncorrupted})$ to the table T_{users}.
   (e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements $\mathbf{v} = (v_0, \ldots, v_N)$, PDE key ek, pad pad.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k+1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$.

Figure 7: Program $\mathsf{crsProg}[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}]$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements $\mathbf{v} = (v_0, \ldots, v_N)$, PDE key ek, pad pad, public key pk.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$.

Figure 8: Program $\mathsf{skProg}'[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}]$.

- D^b: This is the same as C^b, except that we replace the vector of domain elements with two domain elements u, v. The programs are modified to use domain element u if all keys in the input set are labeled (i.e., $k = N$), and domain element v otherwise. Specifically:

  1. The setup phase is modified to work as follows:
     (a) Samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^N)$.
     (b) Samples $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
     (c) Samples $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
     (d) Samples domain elements $u, v \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$.
     (e) Samples $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}])$ where crsProg is described in Figure 9.
     (f) Outputs $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$.
  2. The challenger modifies how it responds to **Honest user registration** queries as follows:
     (a) The challenger increments $i \leftarrow i + 1$.
     (b) The challenger sets $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
     (c) The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i])$ where skProg' is described in Figure 10.
     (d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \mathsf{uncorrupted})$ to the table T_{users}.
     (e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements u, v, PDE key ek, pad pad.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. If $k = N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$.
9. Else, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.

Figure 9: Program $\mathsf{crsProg}[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}]$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements u, v, PDE key ek, pad pad, public key pk.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. If $k = N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$.
9. Else, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.

Figure 10: Program $\mathsf{skProg}'[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}]$.

- E^b: This is the same as D^b, except that the CRS program is modified to output $\perp$ if all keys in the input set are labeled (i.e., $k = N$). Specifically:

  1. The setup phase is modified to sample an obfuscation of the modified program Figure 11 below:
     (a) Samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$.
     (b) Samples $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
     (c) Samples $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
     (d) Samples domain elements $u, v \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$.
     (e) Samples $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}])$ where crsProg is described in Figure 11.
     (f) Outputs $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements u, v, PDE key ek, pad pad.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. If $k = N$, outputs $\perp$.
9. Else, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.

Figure 11: Program $\mathsf{crsProg}[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}]$.

- F^b: This is the same as E^b, except that the challenger generates secret keys for corrupted users lazily, and answers the challenge query by computing the shared key directly using the domain element u and the PPRF key. Specifically:

  2. The challenger modifies how it responds to queries as follows:
     - **Honest user registration:**
       (a) The challenger increments $i \leftarrow i + 1$.
       (b) The challenger sets $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
       (c) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \perp, \mathsf{uncorrupted})$ to the table T_{users}.
       (d) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.
     - **Honest user corruption:**
       (a) The adversary sends an index i'.
       (b) The challenger looks for the entry $(i', \widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{uncorrupted})$ in T_{users}.
       (c) If no such entry exists, outputs $\perp$.
       (d) The challenger samples $\widetilde{\mathsf{sk}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}])$ where skProg' is described in Figure 10.
       (e) The challenger updates the entry to $(i', \widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{corrupted})$.
       (f) The challenger sends $\widetilde{\mathsf{sk}}$ to $\mathcal{A}$.
  3. The challenger modifies how it responds to the **Challenge** query as follows:
     (a) The challenger retrieves $\widetilde{\mathsf{pk}}_j$ for all $j \in \mathcal{I}^*$ from T_{users} and forms the ordered set $U^* = (\widetilde{\mathsf{pk}}_j)_{j \in \mathcal{I}^*}$.
     (b) The challenger computes the challenge value y_b as follows:
       - If $b = 0$, the challenger computes $\gamma_j = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_j) \in \mathcal{K}$ for all $j \in \mathcal{I}^*$, computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{j \in \mathcal{I}^*})$, and sets $y_0 = \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$.
       - If $b = 1$, the challenger samples a uniformly random codomain element $y_1 \xleftarrow{R} \mathcal{Y}$.
     (c) The challenger sends y_b to $\mathcal{A}$.

- G^b: This is the same as F^b, except that the challenger handles corruptions by assigning a fresh domain element to each corrupted user. Every new secret key is generated with the list of previous domain elements hardcoded to ensure consistency with previous keys. Specifically:

1. During the setup phase, the challenger initializes a counter $\mathsf{cc} \leftarrow 0$ and an empty ordered list of corruption pairs $L_0 \leftarrow \emptyset$.
2. The challenger modifies how it responds to **Honest user corruption** queries as follows:
   (a) The adversary sends an index i'.
   (b) The challenger looks for the entry $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \text{uncorrupted})$ in T_{users}.
   (c) If no such entry exists, outputs $\perp$.
   (d) The challenger increments $\mathsf{cc} \leftarrow \mathsf{cc} + 1$.
   (e) The challenger samples a fresh domain element $w_{\mathsf{cc}} \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$.
   (f) The challenger appends $(\mathsf{cc}, \widetilde{\mathsf{pk}}_{i'}, w_{\mathsf{cc}})$ to the list $L_{\mathsf{cc}} \leftarrow L_{\mathsf{cc}-1} \circ (\mathsf{cc}, \widetilde{\mathsf{pk}}_{i'}, w_{\mathsf{cc}})$.
   (g) The challenger samples $\widetilde{\mathsf{sk}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_{i'}, L_{\mathsf{cc}}])$ where skProg' is described in Figure 12.
   (h) The challenger updates the entry to $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \text{corrupted})$.
   (i) The challenger sends $\widetilde{\mathsf{sk}}_{i'}$ to $\mathcal{A}$.
3. The challenger answers the **Challenge** query exactly as in F^b, using the domain element u (which is not included in any secret key) to compute $y_0 = \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$.

Constants: KA-PRF parameters pp, PPRF key K, domain element $v \in \mathcal{X}$, PDE key ek, pad pad, public key pk, list L.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$ and list of indices $I_{in} \leftarrow \emptyset$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. If $k < N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.
9. Else $(k = N)$:
10. Finds the minimum cc' such that $(\mathsf{cc}', \mathsf{pk}', w) \in L$ and $\mathsf{pk}' \in U$.
11. If such an entry exists, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, w))$.
12. Else, outputs $\perp$.

Figure 12: Program $\mathsf{skProg}'[\mathsf{pp}, K, v, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, L]$.

Hybrid A

In the first step, we just change the program used for generating the secret key. In the real scheme, the secret key is a "re-obfuscation" of P_{crs} with the seed s hardcoded. In Hybrid A, we obfuscated a program that is functionally equivalent to P_{crs}, with the only difference being that $G(s)$ is directly hardcoded in it. Therefore this step follows by $i\mathcal{O}$ security.

Lemma 6.3. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator and the algorithms in Figure 5 and Figure 6 are padded to the same size. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{A}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary for the adaptive security game. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ to directly show that Real^b and A^b are computationally indistinguishable as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes an empty table $T_{\mathsf{users}} \leftarrow \emptyset$ and a counter $i \leftarrow 0$. It samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$, $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$, and $v \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$. Algorithm $\mathcal{B}$ samples the obfuscated CRS program $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K, v])$ where crsProg is described in Figure 4 and sends $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$ to $\mathcal{A}$.
3. Whenever $\mathcal{A}$ makes an **Honest user registration** query, algorithm $\mathcal{B}$ increments $i \leftarrow i{+}1$, samples $s_i \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, and sets $\widetilde{\mathsf{pk}}_i = G(s_i)$. Algorithm $\mathcal{B}$ defines two circuits $C_{i,0}$ and $C_{i,1}$:
   - Circuit $C_{i,0}$ computes the program $\mathsf{skProg}[P_{\mathsf{crs}}, s_i]$.
   - Circuit $C_{i,1}$ computes the program $\mathsf{skProg}'[\mathsf{pp}, K, v, \widetilde{\mathsf{pk}}_i]$.

   Algorithm $\mathcal{B}$ submits the pair $(C_{i,0}, C_{i,1})$ to the multi-challenge $i\mathcal{O}$ challenger, receives an obfuscated program P_i^*, and sets $\widetilde{\mathsf{sk}}_i = P_i^*$. Algorithm $\mathcal{B}$ adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \mathsf{uncorrupted})$ to the table T_{users} and sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.
4. For any **Honest user corruption** or **Challenge** query, algorithm $\mathcal{B}$ answers exactly as specified in the experiment using the table T_{users}. Since algorithm $\mathcal{B}$ generated all keys (including the queried obfuscated programs), it can flawlessly simulate these responses.
5. When $\mathcal{A}$ outputs a guess b', algorithm $\mathcal{B}$ forwards b' as its own output to the $i\mathcal{O}$ challenger.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. By the definition of the multi-challenge $i\mathcal{O}$ game, the challenger samples a bit $b_{i\mathcal{O}} \in \{0,1\}$ and obfuscates either the first circuit or the second circuit for all submitted pairs. If $b_{i\mathcal{O}} = 0$, then P_i^* is an obfuscation of $C_{i,0}$ for all queries i, and algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Real}^b(\mathcal{A})$. If $b_{i\mathcal{O}} = 1$, then P_i^* is an obfuscation of $C_{i,1}$ for all queries i, and algorithm $\mathcal{B}$ perfectly simulates $\mathsf{A}^b(\mathcal{A})$. For algorithm $\mathcal{B}$ to be a valid adversary for the $i\mathcal{O}$ game, we must show that the circuits $C_{i,0}$ and $C_{i,1}$ are functionally equivalent for all possible inputs for every query index i. Fix any query index i and any input (U, j) where $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ and $j \in [N]$.

- Evaluating $C_{i,0}(U, j)$ executes $P_{\mathsf{crs}}(s_i, U, j)$. By the functionality-preserving property of $i\mathcal{O}$, evaluating the obfuscated program P_{crs} is functionally identical to evaluating the underlying circuit $\mathsf{crsProg}[\mathsf{pp}, K, v](s_i, U, j)$. The program crsProg checks if U has repetitions (outputting $\perp$), and then checks if $G(s_i) = \mathsf{pk}_j$. If so, it computes the accumulated PPRF product and outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.
- Evaluating $C_{i,1}(U, j)$ executes $\mathsf{skProg}'[\mathsf{pp}, K, v, \widetilde{\mathsf{pk}}_i](U, j)$. It checks if U has repetitions, and then checks if the hardcoded $\widetilde{\mathsf{pk}}_i = \mathsf{pk}_j$. If so, it computes the exact same accumulated PPRF product and outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.

Since algorithm $\mathcal{B}$ explicitly sets $\widetilde{\mathsf{pk}}_i = G(s_i)$ for every query i, the condition $G(s_i) = \mathsf{pk}_j$ in circuit $C_{i,0}$ is equivalent to the condition $\widetilde{\mathsf{pk}}_i = \mathsf{pk}_j$ in circuit $C_{i,1}$. Therefore, $C_{i,0}(U, j) = C_{i,1}(U, j)$ for all inputs with probability 1.

Since algorithm $\mathcal{B}$ is an admissible adversary, by the security of the multi-challenge $i\mathcal{O}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that the advantage of $\mathcal{B}$ is bounded by $\mathsf{negl}(\lambda)$. Thus, we conclude:

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{A}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

This concludes the proof. □

Hybrid B

In this step, we change how the public keys are computed. Relying on the fact that the PRG seeds no longer appear in the adversary's view, we first sample the public keys as random strings instead of PRG samples. Next, we use Π_{PDE} to replace the random strings with pseudorandom ciphertexts.

Lemma 6.4. *Assume that G is a secure PRG and Π_{PDE} satisfies ciphertext pseudorandomness. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{A}^b(\mathcal{A}) = 1] - \Pr[\mathsf{B}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary for the adaptive security game. Let Q be a polynomial upper bound on the number of **Honest user registration** queries made by $\mathcal{A}$. To prove the lemma, we define an intermediate experiment $\mathsf{A}^b_{\mathsf{rand}}$, which is identical to A^b except that the challenger samples the public keys for honest users uniformly at random. Specifically, the challenger modifies how it responds to **Honest user registration** queries as follows:

1. The challenger increments $i \leftarrow i + 1$.
2. The challenger samples a uniformly random string $\widetilde{\mathsf{pk}}_i \xleftarrow{\mathrm{R}} \{0,1\}^{3\lambda}$.
3. The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, v, \widetilde{\mathsf{pk}}_i])$ where skProg' is described in Figure 6.
4. The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \mathsf{uncorrupted})$ to the table T_{users}.
5. The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

We establish the indistinguishability of the sequence $\mathsf{A}^b \approx \mathsf{A}^b_{\mathsf{rand}} \approx \mathsf{B}^b$ through the following two claims.

Claim 6.5. *Assume G is a secure PRG. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{A}^b(\mathcal{A}) = 1] - \Pr[\mathsf{A}^b_{\mathsf{rand}}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. We construct an efficient adversary $\mathcal{B}_1$ against the multi-sample security of the PRG G as follows:

1. Algorithm $\mathcal{B}_1$ requests Q samples from the PRG challenger, receiving strings $y_1, \ldots, y_Q \in \{0,1\}^{3\lambda}$.
2. Algorithm $\mathcal{B}_1$ initializes the experiment, simulating the setup phase exactly as in A^b, and sends crs to $\mathcal{A}$.
3. Whenever $\mathcal{A}$ makes its i-th **Honest user registration** query:
   (a) Algorithm $\mathcal{B}_1$ increments the counter $i \leftarrow i + 1$ and sets $\widetilde{\mathsf{pk}}_i = y_i$.
   (b) Algorithm $\mathcal{B}_1$ computes $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, v, \widetilde{\mathsf{pk}}_i])$. Crucially, because the program skProg' only requires the public key $\widetilde{\mathsf{pk}}_i$ and not the PRG seed, $\mathcal{B}_1$ can compute the secret key without knowing the preimage of y_i.
   (c) Algorithm $\mathcal{B}_1$ adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \mathsf{uncorrupted})$ to T_{users} and sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.
4. Algorithm $\mathcal{B}_1$ answers all **Honest user corruption** queries and the **Challenge** query exactly as specified in the experiment by using the stored entries in the table T_{users}.
5. When $\mathcal{A}$ outputs a guess b', $\mathcal{B}_1$ outputs b' as its own guess.

If the samples y_i provided by the challenger are pseudorandom strings generated by G, then the public keys are correctly distributed as $\widetilde{\mathsf{pk}}_i = G(s_i)$, and $\mathcal{B}_1$ perfectly simulates A^b. If the samples y_i are uniformly random strings, then $\mathcal{B}_1$ perfectly simulates $\mathsf{A}^b_{\mathsf{rand}}$. By the security of the PRG (extended to multiple samples via a standard hybrid argument), the advantage of $\mathcal{B}_1$ is bounded by a negligible function $\mathsf{negl}(\lambda)$. □

Claim 6.6. *Assume Π_{PDE} satisfies ciphertext pseudorandomness. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{A}^b_{\mathsf{rand}}(\mathcal{A}) = 1] - \Pr[\mathsf{B}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. We construct an efficient adversary $\mathcal{B}_2$ for the ciphertext pseudorandomness game of Π_{PDE} as follows:

1. Algorithm $\mathcal{B}_2$ simulates the setup phase exactly as in A^b, but additionally samples a uniform one-time pad $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$. It sends crs to $\mathcal{A}$.
2. Whenever $\mathcal{A}$ makes its i-th **Honest user registration** query:
   (a) Algorithm $\mathcal{B}_2$ increments the counter $i \leftarrow i + 1$.
   (b) Algorithm $\mathcal{B}_2$ sends the index i as an encryption query to the Π_{PDE} challenger. Because i is a strictly incrementing counter, all encryption queries are guaranteed to be unique, satisfying the requirements of the Π_{PDE} security game.
   (c) The Π_{PDE} challenger responds with a ciphertext $c_i \in \{0,1\}^{3\lambda}$.
   (d) Algorithm $\mathcal{B}_2$ computes $\widetilde{\mathsf{pk}}_i = c_i \oplus \mathsf{pad}$.
   (e) Algorithm $\mathcal{B}_2$ computes the secret key $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, v, \widetilde{\mathsf{pk}}_i])$, updates T_{users}, and sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.
3. Algorithm $\mathcal{B}_2$ answers all **Honest user corruption** queries and the **Challenge** query exactly as specified using the table T_{users}.
4. When $\mathcal{A}$ outputs a guess b', $\mathcal{B}_2$ outputs b' as its own guess.

We analyze the simulation provided by $\mathcal{B}_2$. If the ciphertexts provided by the Π_{PDE} challenger are uniformly random strings, i.e., $c_i \xleftarrow{R} \{0,1\}^{3\lambda}$, then the computed public keys $\widetilde{\mathsf{pk}}_i = c_i \oplus \mathsf{pad}$ are uniformly random and independent (since applying a fixed, secret one-time pad to independent uniform strings preserves their uniform distribution). In this case, $\mathcal{B}_2$ perfectly simulates $\mathsf{A}^b_{\mathsf{rand}}$. If the ciphertexts are valid encryptions, i.e., $c_i = \mathsf{PDE.Enc}(\mathsf{ek}, i)$, then the public keys are computed exactly as $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$. In this case, $\mathcal{B}_2$ perfectly simulates B^b. By the ciphertext pseudorandomness of Π_{PDE}, the advantage of $\mathcal{B}_2$ is bounded by a negligible function $\mathsf{negl}(\lambda)$. □

Combining Claim 6.5 and Claim 6.6 via the triangle inequality, we conclude that there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\left|\Pr[\mathsf{A}^b(\mathcal{A}) = 1] - \Pr[\mathsf{B}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

This concludes the proof of Lemma 6.4. □

Hybrid C

In this step, we crucially rely on Π_{kaPRF}. Recall that the programs in Hybrid C count the number of labeled keys and choose the base accordingly. The strategy is to define an inner sequence of hybrids C_q, such that in Hybrid C_q only the public keys labeled up to q count. To go from C_q to C_{q+1}, we define a deeper level of inner hybrids $\mathsf{C}_{q,n}$ such that in $\mathsf{C}_{q,n}$, the "next public key" with label $q+1$ only counts when the number of labeled keys (with labels up to q) is *at least* n. This allows us to switch one base at a time, and eventually count the key $q+1$. Crucially, we go over the deeper level of hybrids from right to left starting with $n = N - 1$ and ending with $n = 0$, thus allowing us to switch each base without colliding with the next one.

Lemma 6.7. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator, Π_{PPRF} satisfies punctured pseudorandomness, and Π_{kaPRF} satisfies approximate evaluation correctness with error parameter $\lambda_s = \lambda + 3\lambda N$ and weak pseudorandomness. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{B}^b(\mathcal{A}) = 1] - \Pr[\mathsf{C}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

To do so, we define a sequence of inner hybrids to show that B^b and C^b are computationally indistinguishable. We define the sequence C^b_q for all $q \in \{0, \ldots, Q\}$ such that C^b_Q is identical to C^b and C^b_0 is computationally indistinguishable from B^b by $i\mathcal{O}$ security (Claim 6.8). The main challenge is to show that C^b_q is computationally indistinguishable from C^b_{q+1} for all $q \in \{0, \ldots, Q-1\}$.

- C^b_q (for $q \in \{0, \ldots, Q\}$): This is the same as B^b, except that the setup samples $N+1$ domain elements $v_0, \ldots, v_N \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$, and the programs are modified to use a domain element v_k where k is the number of labeled keys in the input set that have a label in the set $[q]$. Specifically:

  1. The setup phase is modified to work as follows:
     (a) Samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$.
     (b) Samples $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\text{out}}})$.
     (c) Samples $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
     (d) Samples domain elements $v_0, \ldots, v_N \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$, and denote $\mathbf{v} = (v_0, \ldots, v_N)$.
     (e) Samples $P_{\text{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\text{size}_{\text{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q])$ where crsProg is described in Figure 13.
     (f) Outputs $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\text{crs}})$.
  2. The challenger modifies how it responds to **Honest user registration** queries as follows:
     (a) The challenger increments $i \leftarrow i + 1$.
     (b) The challenger sets $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
     (c) The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\text{size}_{\text{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q])$ where skProg' is described in Figure 14.
     (d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \text{uncorrupted})$ to the table T_{users}.
     (e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements $v_0, \ldots, v_N \in \mathcal{X}$, PDE key ek, pad pad, integer q.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$.

Figure 13: Program $\mathsf{crsProg}[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q]$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements $v_0, \ldots, v_N \in \mathcal{X}$, PDE key ek, pad pad, public key pk, integer q.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$.

Figure 14: Program $\mathsf{skProg}'[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, q]$.

- $\mathsf{C}^b_{q,\mathsf{puncture}}$ (for $q \in \{0, \ldots, Q-1\}$): This is the same as C^b_q, except that the challenger pre-computes the key labeled $q+1$, punctures the PPRF key at that point, and modifies the programs to compute the shared key using pre-computed hardcoded domain elements $z_0, \ldots, z_{N-1}$ when user $q+1$ is involved. Specifically:

  1. The setup phase is modified to work as follows:
     (a) Samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$.
     (b) Samples $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
     (c) Samples $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
     (d) Computes the target public key $\widetilde{\mathsf{pk}}_{q+1} = \mathsf{PDE.Enc}(\mathsf{ek}, q+1) \oplus \mathsf{pad}$.
     (e) Punctures the key $K^* \leftarrow \mathsf{PPRF.Puncture}(K, \widetilde{\mathsf{pk}}_{q+1})$.
     (f) Computes $\beta = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1}) \in \mathcal{K}$.
     (g) Samples domain elements $v_0, \ldots, v_N \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$. Let $\mathbf{v} = (v_0, \ldots, v_N)$.
     (h) Computes $z_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_k)$ for all $k \in \{0, \ldots, N-1\}$. Let $\mathbf{z} = (z_0, \ldots, z_{N-1})$.
     (i) Samples $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z}])$ where crsProg is described in Figure 15.
     (j) Outputs $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$.
  2. The challenger modifies how it responds to **Honest user registration** queries as follows:
     (a) The challenger increments $i \leftarrow i + 1$.
     (b) If $i = q+1$, sets $\widetilde{\mathsf{pk}}_i = \widetilde{\mathsf{pk}}_{q+1}$. Else, sets $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
     (c) The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z}])$ where skProg′ is described in Figure 16.
     (d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \mathsf{uncorrupted})$ to the table T_{users}.
     (e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

Constants: KA-PRF parameters pp, Punctured PPRF key K^*, domain elements $v_0, \ldots, v_N \in \mathcal{X}$, PDE key ek, pad pad, target key $\widetilde{\mathsf{pk}}_{q+1}$, domain elements $z_0, \ldots, z_{N-1} \in \mathcal{X}$.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [q]$, increments $k \leftarrow k + 1$.
6. If $\widetilde{\mathsf{pk}}_{q+1} \notin U$:
7. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
8. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
9. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$.
10. Else ($\widetilde{\mathsf{pk}}_{q+1} \in U$):
11. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \in U \setminus \{\widetilde{\mathsf{pk}}_{q+1}\}$.
12. Computes the accumulated key $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \widetilde{\mathsf{pk}}_{q+1}})$.
13. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_k))$.

Figure 15: Program $\mathsf{crsProg}[\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z}]$.

Constants: KA-PRF parameters pp, Punctured PPRF key K^*, domain elements $v_0, \ldots, v_N \in \mathcal{X}$, PDE key ek, pad pad, public key pk, integer q, target key $\widetilde{\mathsf{pk}}_{q+1}$, domain elements $z_0, \ldots, z_{N-1} \in \mathcal{X}$.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [q]$, increments $k \leftarrow k + 1$.
6. If $\widetilde{\mathsf{pk}}_{q+1} \notin U$:
7. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
8. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
9. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$.
10. Else $(\widetilde{\mathsf{pk}}_{q+1} \in U)$:
11. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \in U \setminus \{\widetilde{\mathsf{pk}}_{q+1}\}$.
12. Computes the accumulated key $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \widetilde{\mathsf{pk}}_{q+1}})$.
13. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_k))$.

Figure 16: Program $\mathsf{skProg}'[\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z}]$.

- $\mathsf{C}^b_{q,\mathsf{uniform}}$: This is the same as $\mathsf{C}^b_{q,\mathsf{puncture}}$, except that β is sampled uniformly from the base key space $\mathcal{K}$. Specifically:
  1. In the setup phase, the challenger samples $\beta \leftarrow \mathsf{kaPRF.KeyGen}(\mathsf{pp})$ instead of computing it via the PPRF.
- $\mathsf{C}^b_{q,\mathsf{noise}}$: This is the same as $\mathsf{C}^b_{q,\mathsf{uniform}}$, except that we apply the noise function to the hardcoded elements. Specifically:
  1. In the setup phase, the challenger computes $z_k \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_k))$ for all $k \in \{0, \ldots, N-1\}$.
- $\mathsf{C}^b_{q,n}$ (for $q \in \{0, \ldots, Q-1\}$ and $n \in \{0, \ldots, N\}$): This is the same as $\mathsf{C}^b_{q,\mathsf{noise}}$, except that z_k for all $k \geq n$ are computed using the shifted domain element v_{k+1}. Specifically:
  1. In the setup phase, the challenger computes $z_0, \ldots, z_{N-1}$ as follows:

$$z_k \leftarrow \begin{cases} \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{k+1})) & k \geq n \\ \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_k)) & k < n \end{cases}$$

- $\overline{\mathsf{C}}^b_{q,n}$ (for $q \in \{0, \ldots, Q-1\}$ and $n \in \{1, \ldots, N\}$): This is the same as $\mathsf{C}^b_{q,n}$, except that z_{n-1} is replaced with a uniformly random domain element in $\mathcal{X}$. Specifically:
  1. In the setup phase, the challenger samples $y \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$ and computes $z_0, \ldots, z_{N-1}$ as follows:

$$z_k \leftarrow \begin{cases} \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{k+1})) & k \geq n \\ y & k = n-1 \\ \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_k)) & k < n-1 \end{cases}$$

- $\mathsf{C}^b_{q,\mathsf{exact}}$ (for $q \in \{0,\ldots,Q-1\}$): This is the same as the final loop state $\mathsf{C}^b_{q,N}$, except that we remove the noise. Specifically:

  1. In the setup phase, the challenger computes exact compositions $z_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_{k+1})$ for all $k \in \{0,\ldots,N-1\}$.

- $\mathsf{C}^b_{q,\mathsf{pseudorandom}}$ (for $q \in \{0,\ldots,Q-1\}$): This is the same as $\mathsf{C}^b_{q,\mathsf{exact}}$, except that β is reverted to being computed pseudo-randomly. Specifically:

  1. In the setup phase, the challenger sets $\beta = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1}) \in \mathcal{K}$.

Claim 6.8. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{B}^b(\mathcal{A}) = 1] - \Pr[\mathsf{C}^b_0(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes an empty table $T_{\mathsf{users}} \leftarrow \emptyset$ and a counter $i \leftarrow 0$.
3. Algorithm $\mathcal{B}$ samples the following:
   (a) Samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$.
   (b) Samples a PPRF key $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
   (c) Samples a PDE key $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and a uniform string $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
   (d) Samples domain elements $v_0,\ldots,v_N \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$, and sets $v = v_0$. Let $\mathbf{v} = (v_0,\ldots,v_N)$.
4. Algorithm $\mathcal{B}$ defines two circuits: circuit C_0 computes the program crsProg from Figure 4 hardcoded with $(\mathsf{pp}, K, v, \mathsf{ek}, \mathsf{pad})$, and circuit C_1 computes the program crsProg from Figure 13 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q = 0)$. Algorithm $\mathcal{B}$ submits the pair (C_0, C_1) to the $i\mathcal{O}$ challenger, receives the obfuscated program P, and sets $P_{\mathsf{crs}} = P$.
5. Algorithm $\mathcal{B}$ sends $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$ to $\mathcal{A}$.
6. Whenever $\mathcal{A}$ makes an **Honest user registration** query, algorithm $\mathcal{B}$ does the following:
   (a) Increments $i \leftarrow i + 1$.
   (b) Computes $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
   (c) Defines two circuits: circuit $S_{i,0}$ computes the program skProg′ from Figure 6 hardcoded with $(\mathsf{pp}, K, v, \widetilde{\mathsf{pk}}_i)$, and circuit $S_{i,1}$ computes the program skProg′ from Figure 14 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q = 0)$.
   (d) Submits the pair $(S_{i,0}, S_{i,1})$ to the $i\mathcal{O}$ challenger, receives the obfuscated program P_i, and sets $\widetilde{\mathsf{sk}}_i = P_i$.
   (e) Adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \mathsf{uncorrupted})$ to the table T_{users} and sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.
7. For any **Honest user corruption** or **Challenge** query that $\mathcal{A}$ makes, algorithm $\mathcal{B}$ has all the information to simulate the answer exactly as specified in the experiments using the table T_{users}.
8. When $\mathcal{A}$ outputs a guess b', algorithm $\mathcal{B}$ outputs b' to the $i\mathcal{O}$ challenger.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. By the definition of the $i\mathcal{O}$ security game, the challenger samples a bit $b_{i\mathcal{O}}$ and obfuscates either the first circuit or the second circuit for all queries. If $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{B}^b(\mathcal{A})$, and if $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}_0^b(\mathcal{A})$. Next, we need to prove that $\mathcal{B}$ is an admissible $i\mathcal{O}$ adversary. To do so, we prove the functional equivalence of the circuit pairs. The programs from Figure 13 and Figure 14 introduce a counter for the number of labeled keys k such that $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [0]$. Because valid honest user labels are in the range 1 to Q, this condition is never satisfied. Thus, the counter is deterministically $k = 0$ for all possible inputs. Consequently, the second circuits will always output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_0))$. Because we defined $v_0 = v$, this evaluates to $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$, which is identical to the output of the first circuits. Since $\mathcal{B}$ is an admissible adversary, by $i\mathcal{O}$ security it wins the security game with negligible probability. Since it perfectly simulates $\mathsf{B}^b(\mathcal{A})$ and $\mathsf{C}_0^b(\mathcal{A})$, the claim follows. □

Claim 6.9. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{C}_q^b(\mathcal{A}) = 1] - \Pr[\mathsf{C}_{q,\mathsf{puncture}}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes an empty table $T_{\mathsf{users}} \leftarrow \emptyset$ and a counter $i \leftarrow 0$.
3. Algorithm $\mathcal{B}$ simulates the setup phase exactly as in C_q^b, generating pp, PPRF key K, PDE key ek, pad pad, and domain elements $\mathbf{v} = (v_0, \ldots, v_N)$.
4. Algorithm $\mathcal{B}$ additionally pre-computes the target public key $\widetilde{\mathsf{pk}}_{q+1} = \mathsf{PDE.Enc}(\mathsf{ek}, q+1) \oplus \mathsf{pad}$. Algorithm $\mathcal{B}$ punctures the PPRF key to obtain $K^* \leftarrow \mathsf{PPRF.Puncture}(K, \widetilde{\mathsf{pk}}_{q+1})$, computes the evaluation $\beta = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1}) \in \mathcal{K}$ and the domain elements $z_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_k)$ for all $k \in \{0, \ldots, N-1\}$. Let $\mathbf{z} = (z_0, \ldots, z_{N-1})$.
5. Algorithm $\mathcal{B}$ defines two circuits for the CRS: circuit C_0 computes the program crsProg from Figure 13 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q)$, and circuit C_1 computes the program crsProg from Figure 15 hardcoded with $(\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z})$. Algorithm $\mathcal{B}$ submits the pair (C_0, C_1) to the $i\mathcal{O}$ challenger, receives the obfuscated program P, and sets $P_{\mathsf{crs}} = P$.
6. Algorithm $\mathcal{B}$ sends $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$ to $\mathcal{A}$.
7. Whenever $\mathcal{A}$ makes an **Honest user registration** query, algorithm $\mathcal{B}$ does the following:
   (a) Increments $i \leftarrow i + 1$.
   (b) If $i = q + 1$, sets $\widetilde{\mathsf{pk}}_i = \widetilde{\mathsf{pk}}_{q+1}$. Otherwise, computes $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
   (c) Defines two circuits: circuit $S_{i,0}$ computes the program skProg′ from Figure 14 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q)$, and circuit $S_{i,1}$ computes the program skProg′ from Figure 16 hardcoded with $(\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z})$.
   (d) Submits the pair $(S_{i,0}, S_{i,1})$ to the $i\mathcal{O}$ challenger, receives the obfuscated program P_i, and sets $\widetilde{\mathsf{sk}}_i = P_i$.
   (e) Adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \text{uncorrupted})$ to the table T_{users} and sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.
8. For any **Honest user corruption** or **Challenge** query that $\mathcal{A}$ makes, algorithm $\mathcal{B}$ simulates the answer exactly as specified in the experiments using the table T_{users}.
9. When $\mathcal{A}$ outputs a guess b', algorithm $\mathcal{B}$ outputs b' to the $i\mathcal{O}$ challenger.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. By the definition of the $i\mathcal{O}$ security game, the challenger samples a bit $b_{i\mathcal{O}}$ and obfuscates either the first circuit or the second circuit for all queries. If $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_q(\mathcal{A})$, and if $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{puncture}}(\mathcal{A})$. Next, we need to prove that $\mathcal{B}$ is an admissible $i\mathcal{O}$ adversary. To do so, we prove the functional equivalence of the circuit pairs. The evaluation logic in both crsProg and skProg′ remains consistent between the two programs. If $\widetilde{\mathsf{pk}}_{q+1} \notin U$, the first and second circuits compute the accumulated key Γ identically and output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$. If $\widetilde{\mathsf{pk}}_{q+1} \in U$, the first circuits evaluate the unpunctured key at all points, yielding an accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \beta, \Gamma')$, and output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \mathsf{Accumulate}(\mathsf{pp}, \beta, \Gamma'), v_k))$. The second circuits use the punctured key to compute Γ' (the accumulated key omitting $\widetilde{\mathsf{pk}}_{q+1}$) and output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_k))$. By the commutativity and composability properties, $\mathsf{Compose}(\mathsf{pp}, \mathsf{Accumulate}(\mathsf{pp}, \beta, \Gamma'), v_k) = \mathsf{Compose}(\mathsf{pp}, \Gamma', \mathsf{Compose}(\mathsf{pp}, \beta, v_k))$. Since $z_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_k)$ and $\widetilde{\mathsf{pk}}_{q+1}$ appears in U only once (the circuit aborts if there are repetitions), the outputs are identical. Since $\mathcal{B}$ is an admissible adversary, by $i\mathcal{O}$ security it wins the security game with negligible probability. Since it perfectly simulates $\mathsf{C}^b_q(\mathcal{A})$ and $\mathsf{C}^b_{q,\mathsf{puncture}}(\mathcal{A})$, the claim follows. □

Claim 6.10. *Assume that* Π_{PPRF} *satisfies punctured pseudorandomness. Then for any efficient adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{C}^b_{q,\mathsf{puncture}}(\mathcal{A}) = 1] - \Pr[\mathsf{C}^b_{q,\mathsf{uniform}}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the punctured pseudorandomness game of Π_{PPRF}. Algorithm $\mathcal{B}$ initializes the setup phase, generates pp, ek and pad, and computes the target key $\widetilde{\mathsf{pk}}_{q+1}$. Algorithm $\mathcal{B}$ submits $\widetilde{\mathsf{pk}}_{q+1}$ to the Π_{PPRF} challenger and receives the punctured key K^* and a challenge evaluation value $y^* \in \mathcal{K}$. Algorithm $\mathcal{B}$ sets $\beta = y^*$ and computes the hardcoded elements $z_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_k)$. Algorithm $\mathcal{B}$ uses K^* and $\mathbf{z}$ to locally obfuscate the programs exactly as specified in the experiments. Algorithm $\mathcal{B}$ answers all queries using the table T_{users}, and outputs $\mathcal{A}$'s guess b'. If $y^* = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1})$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{puncture}}(\mathcal{A})$. If y^* is sampled uniformly from the base key space $\mathcal{K}$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{uniform}}(\mathcal{A})$. By the punctured pseudorandomness of Π_{PPRF}, the distinguishing advantage of algorithm $\mathcal{B}$ is bounded by a negligible function $\mathsf{negl}(\lambda)$, and the claim follows. □

Claim 6.11. *Assume that* $i\mathcal{O}$ *is a secure indistinguishability obfuscator, and* kaPRF *satisfies approximate evaluation correctness with error parameter* $\lambda_s = \lambda + 3\lambda N$*. Then for any efficient adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{C}^b_{q,\mathsf{uniform}}(\mathcal{A}) = 1] - \Pr[\mathsf{C}^b_{q,\mathsf{noise}}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes an empty table $T_{\mathsf{users}} \leftarrow \varnothing$ and a counter $i \leftarrow 0$.
3. Algorithm $\mathcal{B}$ simulates the setup phase exactly as in $\mathsf{C}^b_{q,\mathsf{uniform}}$, generating pp, K, ek, pad, target public key $\widetilde{\mathsf{pk}}_{q+1}$, punctured key K^*, uniform element $\beta \leftarrow \mathsf{kaPRF.KeyGen}(\mathsf{pp})$, and domain elements $\mathbf{v} = (v_0, \ldots, v_N)$.
4. Algorithm $\mathcal{B}$ computes for all $k \in \{0, \ldots, N-1\}$ the exact elements $z_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_k)$ and the noisy elements $z'_k \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_k))$. Let $\mathbf{z} = (z_0, \ldots, z_{N-1})$ and $\mathbf{z}' = (z'_0, \ldots, z'_{N-1})$.
5. Algorithm $\mathcal{B}$ defines two circuits for the CRS: circuit C_0 computes the program crsProg from Figure 15 hardcoded with $(\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z})$, and circuit C_1 computes the same program hardcoded with the noisy vector $\mathbf{z}'$. Algorithm $\mathcal{B}$ submits the size bound $1^{\mathsf{size}_{\mathsf{crs}}}$ and the pair (C_0, C_1) to the $i\mathcal{O}$ challenger, receives the obfuscated program P, and sets $P_{\mathsf{crs}} = P$.

6. Algorithm $\mathcal{B}$ sends $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$ to $\mathcal{A}$.

7. Whenever $\mathcal{A}$ makes an **Honest user registration** query, algorithm $\mathcal{B}$ defines two circuits for the secret key mirroring the exact versus noisy vectors: circuit $C_{i,0}$ computes the program skProg' from Figure 16 hardcoded with $(\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z})$, where pk is the public corresponding to the user, and circuit $C_{i,1}$ computes the same program hardcoded with the noisy vector $\mathbf{z}'$ instead of $\mathbf{z}$. Algorithm $\mathcal{B}$ submits $(C_{i,0}, C_{i,1})$ to the challenger, sets $\widetilde{\mathsf{sk}}_i$ to the response, updates T_{users}, and sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

8. Algorithm $\mathcal{B}$ simulates all other queries using the table T_{users}, and finally outputs $\mathcal{A}$'s guess b' to the $i\mathcal{O}$ challenger.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. By the definition of the $i\mathcal{O}$ security game, if $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{uniform}}(\mathcal{A})$. If $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{noise}}(\mathcal{A})$.

Next, we prove that $\mathcal{B}$ is an admissible $i\mathcal{O}$ adversary by showing functional equivalence for all circuits submitted by $\mathcal{B}$. A functional discrepancy between the circuit pairs occurs if and only if there exists a possible value of Γ' and k such that $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_k)) \neq \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z'_k))$. Fix any list $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ that includes $\widetilde{\mathsf{pk}}_{q+1}$, and denote $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \neq \widetilde{\mathsf{pk}}_{q+1}$, and let $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \widetilde{\mathsf{pk}}_{q+1}})$. Since each v_k is sampled uniformly from $\mathcal{X}$, then $z_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_k)$ is also uniformly distributed by the bijection property of kaPRF. This allows us to invoke the approximate evaluation correctness property of kaPRF to conclude that a discrepancy between z_k and z'_k happens with probability at most $2^{-\lambda_s}$ for each v_k. By taking a union bound over all z_k, we get $N \cdot 2^{-\lambda_s}$ probability of a discrepancy for any fixed list U. Next, we take a union bound over all possible lists U: the list consists of $N-1$ public keys (other than $\widetilde{\mathsf{pk}}_{q+1}$), each of which has $2^{3\lambda}$ possible values, therefore we have less than $2^{3\lambda N}$ possible values of U. Therefore the probability that there exists an input for which there is a discrepancy is at most $N \cdot 2^{-\lambda_s} \cdot 2^{3\lambda N} \leq N \cdot 2^{-\lambda}$. Since algorithm $\mathcal{B}$ is an admissible adversary with overwhelming probability, the difference between the hybrids is bounded by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security. □

Proposition 6.12. *For any parameter $b \in \{0,1\}$ and any q, the experiment $\mathsf{C}^b_{q,\mathsf{noise}}$ is identical to $\mathsf{C}^b_{q,n}$ for $n = N$.*

Proof. This follows immediately by how z_k are computed in $\mathsf{C}^b_{q,n}$. □

Claim 6.13. *Assume that Π_{kaPRF} satisfies weak pseudorandomness. Then for any efficient adversary $\mathcal{A}$ and any $n \in \{1, \ldots, N\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{C}^b_{q,n}(\mathcal{A}) = 1] - \Pr[\overline{\mathsf{C}}^b_{q,n}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the weak pseudorandomness game of the KA-PRF as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.

2. Algorithm $\mathcal{B}$ receives pp from the KA-PRF challenger.

3. Algorithm $\mathcal{B}$ requests $N-1$ samples (x_k, y_k) from its KA-PRF oracle for indices $k \in \{0, \ldots, N-1\} \setminus \{n-1\}$, and receives one challenge pair (x^*, y^*) from the Challenge oracle. The implicit KA-PRF secret γ acts as the uniform element $\beta \in \mathcal{K}$.

4. Algorithm $\mathcal{B}$ simulates the setup phase. It assigns the public domain elements and the hardcoded elements to execute the right-to-left shift as follows:

   - For $k < n-1$: algorithm $\mathcal{B}$ sets $v_k = x_k$ and $z_k = y_k$.
   - For $k \geq n$: algorithm $\mathcal{B}$ sets $v_{k+1} = x_k$ and $z_k = y_k$.
   - For $k = n-1$: algorithm $\mathcal{B}$ sets $v_{n-1} = x^*$ and $z_{n-1} = y^*$.

5. Algorithm $\mathcal{B}$ explicitly samples the unassigned domain element v_n (and any other unused domain elements) via $\mathsf{kaPRF.SampleIn}(\mathsf{pp})$.

6. Algorithm $\mathcal{B}$ locally obfuscates the respective programs using these assigned values and answers all subsequent queries.

7. When $\mathcal{A}$ outputs a guess b', algorithm $\mathcal{B}$ outputs b' as its own guess.

If the challenge y^* is a true noisy sample $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x^*))$, then z_{n-1} is distributed exactly as $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{n-1}))$. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,n}(\mathcal{A})$. If the challenge y^* is a uniformly random element from $\mathcal{X}$, then z_{n-1} is distributed as $y \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$. In this case, algorithm $\mathcal{B}$ perfectly simulates $\overline{\mathsf{C}}^b_{q,n}(\mathcal{A})$. By the weak pseudorandomness of the KA-PRF, the distinguishing advantage of algorithm $\mathcal{B}$ is bounded by a negligible function $\mathsf{negl}(\lambda)$, and the claim follows. □

Claim 6.14. *Assume that Π_{kaPRF} satisfies weak pseudorandomness. Then for any efficient adversary $\mathcal{A}$ and any $n \in \{1, \ldots, N\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\overline{\mathsf{C}}^b_{q,n}(\mathcal{A}) = 1] - \Pr[\mathsf{C}^b_{q,n-1}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the weak pseudorandomness game of the KA-PRF as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.

2. Algorithm $\mathcal{B}$ receives pp from the KA-PRF challenger.

3. Algorithm $\mathcal{B}$ requests $N-1$ samples (x_k, y_k) from its KA-PRF oracle for indices $k \in \{0, \ldots, N-1\} \setminus \{n-1\}$, and receives one challenge pair (x^*, y^*) from the Challenge oracle.

4. Algorithm $\mathcal{B}$ simulates the setup phase. It assigns the domain elements and hardcoded elements to complete the shift of index $n-1$ to the new domain element v_n as follows:

   - For $k < n-1$: algorithm $\mathcal{B}$ sets $v_k = x_k$ and $z_k = y_k$.
   - For $k \geq n$: algorithm $\mathcal{B}$ sets $v_{k+1} = x_k$ and $z_k = y_k$.
   - For $k = n-1$: algorithm $\mathcal{B}$ sets $v_n = x^*$ and $z_{n-1} = y^*$.

5. Algorithm $\mathcal{B}$ explicitly samples the unassigned domain element v_{n-1} (and any other unused domain elements) via $\mathsf{kaPRF.SampleIn}(\mathsf{pp})$. Because the shift proceeds right-to-left, z_n has already been shifted to use v_{n+1}. Consequently, the domain element v_n is completely unbound from the rest of the elements, allowing algorithm $\mathcal{B}$ to map it to the independent challenge domain element x^* without collisions.

6. Algorithm $\mathcal{B}$ locally obfuscates the programs using these values, answers all queries, and outputs $\mathcal{A}$'s guess b'.

If the challenge y^* is a uniformly random element in $\mathcal{X}$, then z_{n-1} is uniformly random. In this case, algorithm $\mathcal{B}$ perfectly simulates $\overline{\mathsf{C}}^b_{q,n}(\mathcal{A})$. If the challenge y^* is a true noisy sample $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x^*))$, then z_{n-1} is distributed exactly as $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_n))$, which completes the shift to the new domain element. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,n-1}(\mathcal{A})$. By the weak pseudorandomness assumption, the distinguishing advantage is bounded by $\mathsf{negl}(\lambda)$. □

Claim 6.15. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator, and Π_{kaPRF} satisfies the approximate evaluation correctness with error $\lambda_s = \lambda + 3\lambda N$. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{C}^b_{q,0}(\mathcal{A}) = 1] - \Pr[\mathsf{C}^b_{q,\mathsf{exact}}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the experiment exactly as in $\mathsf{C}^b_{q,0}$. At the completion of the inner loop ($n = 0$), all elements have been successfully shifted right such that the intended evaluations are $z_k \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{k+1}))$ for all $k \in \{0, \ldots, N-1\}$.
3. Algorithm $\mathcal{B}$ computes the noisy elements z_k and the exact elements $z'_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_{k+1})$. Let $\mathbf{z} = (z_0, \ldots, z_{N-1})$ and $\mathbf{z}' = (z'_0, \ldots, z'_{N-1})$.
4. For the CRS and each honest user key, algorithm $\mathcal{B}$ submits a circuit pair evaluating the punctured programs. The first circuits are hardcoded with the noisy vector $\mathbf{z}$, and the second circuits are hardcoded with the exact vector $\mathbf{z}'$.
5. Algorithm $\mathcal{B}$ uses the returned obfuscated programs to answer $\mathcal{A}$'s queries, and finally outputs $\mathcal{A}$'s guess b'.

If the $i\mathcal{O}$ challenger bit $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,0}(\mathcal{A})$. If $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{exact}}(\mathcal{A})$.

Next, we prove that $\mathcal{B}$ is an admissible $i\mathcal{O}$ adversary. This follows again by the approximate evaluation correctness property of the KA-PRF and a similar argument to the proof of Claim 6.11. By $i\mathcal{O}$ security, the advantage of algorithm $\mathcal{B}$ is bounded by a negligible function $\mathsf{negl}(\lambda)$. □

Claim 6.16. *Assume that Π_{PPRF} satisfies punctured pseudorandomness. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{C}^b_{q,\mathsf{exact}}(\mathcal{A}) = 1] - \Pr[\mathsf{C}^b_{q,\mathsf{pseudorandom}}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the punctured pseudorandomness of Π_{PPRF} as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the setup phase, generates pp, ek and pad, and computes the target public key $\widetilde{\mathsf{pk}}_{q+1}$.
3. Algorithm $\mathcal{B}$ submits $\widetilde{\mathsf{pk}}_{q+1}$ to the Π_{PPRF} challenger and receives the punctured key K^* and a challenge evaluation value $y^* \in \mathcal{K}$.
4. Algorithm $\mathcal{B}$ sets $\beta = y^*$ and computes the exact elements $z_k = \mathsf{Compose}(\mathsf{pp}, \beta, v_{k+1})$.
5. Algorithm $\mathcal{B}$ locally obfuscates the programs using K^* and $\mathbf{z}$, answers all queries using the table T_{users}, and outputs $\mathcal{A}$'s guess b'.

If y^* is sampled uniformly from the base key space $\mathcal{K}$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{exact}}(\mathcal{A})$. If y^* is the true evaluation $\mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1})$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{pseudorandom}}(\mathcal{A})$. By the punctured pseudorandomness of Π_{PPRF}, the distinguishing advantage is bounded by $\mathsf{negl}(\lambda)$, and the claim follows. □

Claim 6.17. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{C}^b_{q,\mathsf{pseudorandom}}(\mathcal{A}) = 1] - \Pr[\mathsf{C}^b_{q+1}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.

2. Algorithm $\mathcal{B}$ initializes the setup phase, computing the full unpunctured key K, the target key $\widetilde{\mathsf{pk}}_{q+1}$, the punctured key K^*, and the shifted hardcoded elements $z_k = \mathsf{Compose}(\mathsf{pp}, \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1}), v_{k+1})$. Let $\mathbf{z} = (z_0, \ldots, z_{N-1})$.

3. For the CRS program, algorithm $\mathcal{B}$ submits a pair of circuits to the challenger: circuit C_0 computes the punctured program from Figure 15 hardcoded with $(\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z})$, and circuit C_1 computes the standard, unpunctured program crsProg from Figure 13 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q+1)$.

4. Algorithm $\mathcal{B}$ similarly constructs circuit pairs for each honest user registration query.

5. Algorithm $\mathcal{B}$ answers all queries using the returned obfuscated programs and outputs $\mathcal{A}$'s guess b'.

If $b_{iO} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q,\mathsf{pseudorandom}}(\mathcal{A})$. If $b_{iO} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b_{q+1}(\mathcal{A})$. Next, we verify the functional equivalence of the circuit pairs to prove admissibility. In the first circuits, if user $q+1$ is present in the input set U, the punctured program identifies it, computes the partial product Γ', and outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_k)) = \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', \mathsf{Compose}(\mathsf{pp}, \beta, v_{k+1})))$. Note that user $q+1$ may appear at most once in the set U, since the program aborts if there are repetitions. In the second circuits, the label threshold is incremented to $q+1$. The condition $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [q+1]$ will now correctly evaluate to true for user $q+1$. Thus, the internal label counter increments to $k+1$. The unpunctured program computes the full joint product $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \beta, \Gamma')$ natively and naturally outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_{k+1}))$. By the Composability and Commutativity properties, the outputs are exactly equivalent for all inputs. Thus, the circuits are functionally equivalent, algorithm $\mathcal{B}$ is admissible, and the claim follows by $i\mathcal{O}$ security. □

Proof of Lemma 6.7. By Claim 6.8, the initial experiment B^b is computationally indistinguishable from C^b_0 via the security of $i\mathcal{O}$. By definition, the experiment C^b_Q is identical to C^b. To bound the distance between C^b_0 and C^b_Q, we analyze the sequence of intermediate hybrids C^b_q for $q \in \{0, \ldots, Q-1\}$ and bound the distance of a single transition from C^b_q to C^b_{q+1}. For any fixed iteration q, the transition consists of the following sequential hybrid steps:

1. By Claim 6.9, the transition to the punctured programs bounds the distance between C^b_q and $\mathsf{C}^b_{q,\mathsf{puncture}}$ by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security.

2. By Claim 6.10, the distance between $\mathsf{C}^b_{q,\mathsf{puncture}}$ and $\mathsf{C}^b_{q,\mathsf{uniform}}$ is bounded by $\mathsf{negl}(\lambda)$ via the punctured pseudorandomness of Π_{PPRF}.

3. By Claim 6.11, the distance between $\mathsf{C}^b_{q,\mathsf{uniform}}$ and $\mathsf{C}^b_{q,\mathsf{noise}}$ is bounded by $\mathsf{negl}(\lambda)$.

4. By Proposition 6.12, the experiment $\mathsf{C}^b_{q,\mathsf{noise}}$ is identical to the start of the inner-inner loop $\mathsf{C}^b_{q,N}$.

5. To process the base shift, the sequence iterates n from N down to 1. For each iteration n, Claim 6.13 bounds the distance between $\mathsf{C}^b_{q,n}$ and $\overline{\mathsf{C}}^b_{q,n}$ by $\mathsf{negl}(\lambda)$, and Claim 6.14 bounds the distance between $\overline{\mathsf{C}}^b_{q,n}$ and $\mathsf{C}^b_{q,n-1}$ by $\mathsf{negl}(\lambda)$, both via the weak pseudorandomness of the KA-PRF. Thus, iterating this inner loop N times bounds the distance between $\mathsf{C}^b_{q,N}$ and the terminal inner state $\mathsf{C}^b_{q,0}$ by $O(N) \cdot \mathsf{negl}(\lambda)$.

6. By Claim 6.15, the distance between $\mathsf{C}^b_{q,0}$ and $\mathsf{C}^b_{q,\mathsf{exact}}$ by $\mathsf{negl}(\lambda)$ is bounded by $\mathsf{negl}(\lambda)$.

7. By Claim 6.16, the distance between $\mathsf{C}^b_{q,\mathsf{exact}}$ and $\mathsf{C}^b_{q,\mathsf{pseudorandom}}$ is bounded by $\mathsf{negl}(\lambda)$ via the punctured pseudorandomness of Π_{PPRF}.

8. Finally, by Claim 6.17, reverting to the standard unpunctured programs bounds the distance between $\mathsf{C}^b_{q,\mathsf{pseudorandom}}$ and C^b_{q+1} by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security.

Summing the advantages of these intermediate steps, the total distinguishing advantage between C^b_q and C^b_{q+1} is bounded by $O(N) \cdot \mathsf{negl}(\lambda)$. To complete the proof, we apply a standard hybrid argument over the outer sequence from $q = 0$ to $Q - 1$. The cumulative distance between C^b_0 and C^b_Q is bounded by:

$$O(Q \cdot N) \cdot \mathsf{negl}(\lambda).$$

The claim now follows by the fact that Q and N are polynomials in λ.

Hybrid D

This step will be analogous to the previous one. Recall that the programs in Hybrid C use base v_k where k is the number of labeled keys. In Hybrid D the programs use base u if $k = N$, and use base v otherwise. To get from C to D, we define an inner sequence of hybrids D_q. In all hybrids, if $k = N$ then the programs will use base v_N, but the behavior changes if $k < N$. In each of those hybrids, there will be an additional counter k' that counts the number of keys labeled up to q. In the case that $k < N$, the programs will compute $t = k - k'$ and use the base v_t. In $\mathsf{D}_{q,\mathsf{final}}$, when all labeled keys count toward k', the programs will be functionally equivalent to the programs in D. To move from D_q to D_{q+1}, we define a deeper level of inner hybrids $\mathsf{D}_{q,n}$ where the counter k' will count the "next public key" $\widetilde{\mathsf{pk}}_{q+1}$ only when $k - k'$ is *less* than n. This time, we avoid base collisions by going over the deeper level hybrids from left to right starting from $n = 0$ and ending with $n = N - 1$.

Lemma 6.18. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator, Π_{PPRF} satisfies punctured pseudorandomness, and Π_{kaPRF} satisfies approximate evaluation correctness with error parameter $\lambda_s = \lambda + 3\lambda N$ and weak pseudorandomness. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{C}^b(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

The proof will be analogous to that of Lemma 6.7 (which went from Hybrid B to Hybrid C). We define the sequence D^b_q for all $q \in \{0, \ldots, Q\}$ such that D^b_0 is computationally indistinguishable from C^b by $i\mathcal{O}$ security (Claim 6.19) and D^b_Q is computationally indistinguishable from D^b by $i\mathcal{O}$ security (Claim 6.29). The main challenge is to show that D^b_q is computationally indistinguishable from D^b_{q+1} for all $q \in \{0, \ldots, Q-1\}$.

- D^b_q (for $q \in \{0, \ldots, Q\}$): This is the same as C^b, except that the programs count both the total number of labeled keys (denoted k) and the number of keys with labels in the set $[q]$ (denoted k'). The domain element selection is modified based on these counts. Specifically:

  1. The setup phase is modified to work as follows:
     (a) Samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$.
     (b) Samples $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
     (c) Samples $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
     (d) Samples domain elements $v_0, \ldots, v_N \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$, and denote $\mathbf{v} = (v_0, \ldots, v_N)$.
     (e) Samples $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q])$ where crsProg is described in Figure 17.
     (f) Outputs $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$.
  2. The challenger modifies how it responds to **Honest user registration** queries as follows:
     (a) The challenger increments $i \leftarrow i + 1$.
     (b) The challenger sets $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
     (c) The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q])$ where skProg' is described in Figure 18.
     (d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \mathsf{uncorrupted})$ to the table T_{users}.
     (e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements $\mathbf{v}$, PDE key ek, pad pad, integer q.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counters $k \leftarrow 0$ and $k' \leftarrow 0$.
4. For each $j \in [N]$:
5. Let label = $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad})$.
6. If label $\in [Q]$:
7. Increments $k \leftarrow k + 1$.
8. If label $\leq q$, increments $k' \leftarrow k' + 1$.
9. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
10. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
11. If $k = N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_N))$.
12. Else ($k < N$), outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_{k-k'}))$.

Figure 17: Program $\mathsf{crsProg}[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q]$.

Constants: KA-PRF parameters pp, a PPRF key K, domain elements $\mathbf{v}$, PDE key ek, pad pad, public key pk, integer q.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\widetilde{\mathsf{pk}} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counters $k \leftarrow 0$ and $k' \leftarrow 0$.
4. For each $j \in [N]$:
5. Let label = $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad})$.
6. If label $\in [Q]$:
7. Increments $k \leftarrow k + 1$.
8. If label $\leq q$, increments $k' \leftarrow k' + 1$.
9. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
10. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
11. If $k = N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_N))$.
12. Else ($k < N$), outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_{k-k'}))$.

Figure 18: Program $\mathsf{skProg}'[\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, q]$.

- $\mathsf{D}^b_{q,\mathsf{puncture}}$ (for $q \in \{0, \ldots, Q-1\}$): This is the same as D^b_q, except that the challenger pre-computes the key $\widetilde{\mathsf{pk}}_{q+1}$, punctures the PPRF key at that point, and modifies the programs to compute the shared key using pre-computed hardcoded domain elements $Z = (z_1, \ldots, z_N)$ when user $q+1$ is involved. Specifically:

  1. The setup phase is modified to work as follows:

(a) Samples $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$.
(b) Samples $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\mathsf{out}}})$.
(c) Samples $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
(d) Computes the target public key $\widetilde{\mathsf{pk}}_{q+1} = \mathsf{PDE.Enc}(\mathsf{ek}, q+1) \oplus \mathsf{pad}$.
(e) Punctures the key $K^* \leftarrow \mathsf{PPRF.Puncture}(K, \widetilde{\mathsf{pk}}_{q+1})$.
(f) Computes $\beta = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1}) \in \mathcal{K}$.
(g) Samples domain elements $v_0, \ldots, v_N \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$. Let $\mathbf{v} = (v_0, \ldots, v_N)$.
(h) Computes $z_t = \mathsf{Compose}(\mathsf{pp}, \beta, v_t)$ for all $t \in \{1, \ldots, N\}$. Let $\mathbf{z} = (z_1, \ldots, z_N)$.
(i) Samples $P_{\mathsf{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{crs}}}, \mathsf{crsProg}[\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z}])$ where crsProg is described in Figure 19.
(j) Outputs $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$.

2. The challenger modifies how it responds to **Honest user registration** queries as follows:
(a) The challenger increments $i \leftarrow i + 1$.
(b) If $i = q + 1$, use the pre-computed $\widetilde{\mathsf{pk}}_{q+1}$. Else, set $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
(c) The challenger samples $\widetilde{\mathsf{sk}}_i \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z}])$ where skProg′ is described in Figure 20.
(d) The challenger adds the entry $(i, \widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i, \mathsf{uncorrupted})$ to the table T_{users}.
(e) The challenger sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

Constants: KA-PRF parameters pp, Punctured PPRF key K^*, domain elements $\mathbf{v}$, PDE key ek, pad pad, integer q, target key $\widetilde{\mathsf{pk}}_{q+1}$, domain elements $\mathbf{z}$.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counters $k \leftarrow 0$ and $k' \leftarrow 0$.
4. For each $j \in [N]$:
5. Let label = $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad})$.
6. If label $\in [Q]$:
7. Increments $k \leftarrow k + 1$.
8. If label $\leq q$, increments $k' \leftarrow k' + 1$.
9. If $\widetilde{\mathsf{pk}}_{q+1} \notin U$:
10. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
11. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$ and outputs

$$\begin{cases} \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_N)) & k = N \\ \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_{k-k'})) & k \neq N \end{cases}$$

12. Else ($\widetilde{\mathsf{pk}}_{q+1} \in U$):
13. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \in U \setminus \{\widetilde{\mathsf{pk}}_{q+1}\}$.
14. Computes the accumulated key $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \widetilde{\mathsf{pk}}_{q+1}})$ and outputs

$$\begin{cases} \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_N)) & k = N \\ \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_{k-k'})) & k \neq N \end{cases}$$

Figure 19: Program $\mathsf{crsProg}[\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z}]$.

Constants: KA-PRF parameters pp, Punctured PPRF key K^*, domain elements $\mathbf{v}$, PDE key ek, pad pad, public key pk, integer q, target key $\widetilde{\mathsf{pk}}_{q+1}$, domain elements $\mathbf{z}$.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\widetilde{\mathsf{pk}} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counters $k \leftarrow 0$ and $k' \leftarrow 0$.
4. For each $j \in [N]$:
5. Let label = $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad})$.
6. If label $\in [Q]$:
7. Increments $k \leftarrow k + 1$.
8. If label $\leq q$, increments $k' \leftarrow k' + 1$.
9. If $\widetilde{\mathsf{pk}}_{q+1} \notin U$:
10. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
11. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$ and outputs

$$\begin{cases} \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_N)) & k = N \\ \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_{k-k'})) & k \neq N \end{cases}$$

12. Else ($\widetilde{\mathsf{pk}}_{q+1} \in U$):
13. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \in U \setminus \{\widetilde{\mathsf{pk}}_{q+1}\}$.
14. Computes the accumulated key $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \widetilde{\mathsf{pk}}_{q+1}})$ and outputs

$$\begin{cases} \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_N)) & k = N \\ \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_{k-k'})) & k \neq N \end{cases}$$

Figure 20: Program $\mathsf{skProg}'[\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z}]$.

- $\mathsf{D}^b_{q,\mathsf{uniform}}$ (for $q \in \{0, \ldots, Q-1\}$): This is the same as $\mathsf{D}^b_{q,\mathsf{puncture}}$, except that β is sampled uniformly from the base key space $\mathcal{K}$. Specifically:
  1. In the setup phase, the challenger samples $\beta \leftarrow \mathsf{kaPRF.KeyGen}(\mathsf{pp})$ instead of computing it via the PPRF.
- $\mathsf{D}^b_{q,\mathsf{noise}}$: This is the same as $\mathsf{D}^b_{q,\mathsf{uniform}}$, except that we apply the noise function to the hardcoded domain elements. Specifically:
  1. In the setup phase, the challenger computes $z_t \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_t))$ for all $t \in \{1, \ldots, N\}$.
- $\mathsf{D}^b_{q,n}$ (for $q \in \{0, \ldots, Q-1\}$ and $n \in \{1, \ldots, N\}$): This is the same as $\mathsf{D}^b_{q,\mathsf{noise}}$, except that the shifted domain element is used for z_t where $t < n$. Crucially, z_N is never shifted. Specifically:
  1. In the setup phase, the challenger computes $z_1, \ldots, z_N$ as follows:
     - $z_N \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_N))$.

– For $t \in \{1, \ldots, N-1\}$:

$$z_t \leftarrow \begin{cases} \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{t-1})) & t < n \\ \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_t)) & t \geq n \end{cases}$$

- $\overline{\mathsf{D}}^b_{q,n}$ (for $q \in \{0, \ldots, Q-1\}$ and $n \in \{1, \ldots, N-1\}$): This is the same as $\mathsf{D}^b_{q,n}$, except that z_n is replaced with a uniformly random domain element in $\mathcal{X}$. Specifically:

  1. In the setup phase, the challenger samples $y \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$ and computes $z_1, \ldots, z_N$ as follows:
     – $z_N \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_N))$.
     – For $t \in \{1, \ldots, N-1\}$:

$$z_t \leftarrow \begin{cases} \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{t-1})) & t < n \\ y & t = n \\ \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_t)) & t > n \end{cases}$$

- $\mathsf{D}^b_{q,\mathsf{exact}}$ (for $q \in \{0, \ldots, Q-1\}$): This is the same as the final loop state $\mathsf{D}^b_{q,N}$, except that we remove the noise. Specifically:

  1. In the setup phase, the challenger computes the exact compositions:
     – $z_N = \mathsf{Compose}(\mathsf{pp}, \beta, v_N)$.
     – $z_t = \mathsf{Compose}(\mathsf{pp}, \beta, v_{t-1})$ for all $t \in \{1, \ldots, N-1\}$.

- $\mathsf{D}^b_{q,\mathsf{pseudorandom}}$ (for $q \in \{0, \ldots, Q-1\}$): This is the same as $\mathsf{D}^b_{q,\mathsf{exact}}$, except that β is reverted to being computed pseudo-randomly. Specifically:

  1. In the setup phase, the challenger sets $\beta = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1}) \in \mathcal{K}$.

Claim 6.19. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{C}^b(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b_0(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes an empty table $T_{\mathsf{users}} \leftarrow \varnothing$ and a counter $i \leftarrow 0$.
3. Algorithm $\mathcal{B}$ simulates the setup phase exactly as in C^b, generating pp, PPRF key K, PDE key ek, pad pad, and domain elements $\mathbf{v} = (v_0, \ldots, v_N)$.
4. Algorithm $\mathcal{B}$ defines two circuits for the CRS: circuit C_0 computes the program crsProg from Figure 13 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q = Q)$, and circuit C_1 computes the program crsProg from Figure 17 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q = 0)$. Algorithm $\mathcal{B}$ submits the pair (C_0, C_1) to the $i\mathcal{O}$ challenger, receives the obfuscated program P, and sets $P_{\mathsf{crs}} = P$.
5. Algorithm $\mathcal{B}$ sends $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\mathsf{crs}})$ to $\mathcal{A}$.
6. Whenever $\mathcal{A}$ makes an **Honest user registration** query, algorithm $\mathcal{B}$ increments i, computes $\widetilde{\mathsf{pk}}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$, and defines two circuits for the secret key: circuit $S_{i,0}$ computes the program skProg′ from Figure 14 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q = Q)$, and circuit $S_{i,1}$ computes the program skProg′ from Figure 18 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_i, q = 0)$. Algorithm $\mathcal{B}$ submits the pair $(S_{i,0}, S_{i,1})$ to the $i\mathcal{O}$ challenger, sets $\widetilde{\mathsf{sk}}_i$ to the returned obfuscated program, updates T_{users}, and sends $\widetilde{\mathsf{pk}}_i$ to $\mathcal{A}$.

7. Algorithm $\mathcal{B}$ simulates all other queries using the table T_{users}, and finally outputs $\mathcal{A}$'s guess b' to the $i\mathcal{O}$ challenger.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. By the definition of the $i\mathcal{O}$ security game, if the challenger bit $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{C}^b(\mathcal{A})$. If the bit $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}_0^b(\mathcal{A})$. Next, we prove that $\mathcal{B}$ is an admissible $i\mathcal{O}$ adversary by showing functional equivalence. The first circuits count the number of keys with labels $\leq Q$. Because all honest user labels fall in $[Q]$, this exactly equals the total number of honest keys k, and the program outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$. The second circuits compute the total honest keys k, and the keys with labels ≤ 0 as k'. Because valid labels are strictly positive, $k' = 0$ for all inputs. The program outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_N))$ if $k = N$, and $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$ if $k < N$. Thus, the second circuits identically evaluate to $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_k))$ for all valid $k \leq N$. The circuits are functionally equivalent, and the claim follows by $i\mathcal{O}$ security. □

Claim 6.20. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$ and any $q \in \{0, \ldots, Q-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{D}_q^b(\mathcal{A}) = 1] - \Pr[\mathsf{D}_{q,\mathsf{puncture}}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes an empty table $T_{\mathsf{users}} \leftarrow \varnothing$ and a counter $i \leftarrow 0$.
3. Algorithm $\mathcal{B}$ simulates the setup phase exactly as in D_q^b, generating pp, K, ek, pad pad, and bases $\mathbf{v}$.
4. Algorithm $\mathcal{B}$ pre-computes the target public key $\widetilde{\mathsf{pk}}_{q+1} = \mathsf{PDE.Enc}(\mathsf{ek}, q+1) \oplus \mathsf{pad}$. Algorithm $\mathcal{B}$ punctures the PPRF key to obtain $K^* \leftarrow \mathsf{PPRF.Puncture}(K, \widetilde{\mathsf{pk}}_{q+1})$, computes $\beta = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1}) \in \mathcal{K}$, and computes the elements $z_t = \mathsf{Compose}(\mathsf{pp}, \beta, v_t)$ for all $t \in \{1, \ldots, N\}$. Let $\mathbf{z} = (z_1, \ldots, z_N)$.
5. Algorithm $\mathcal{B}$ defines two circuits for the CRS: circuit C_0 computes the program crsProg from Figure 17 hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q)$, and circuit C_1 computes the program crsProg from Figure 19 hardcoded with $(\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z})$. Algorithm $\mathcal{B}$ submits (C_0, C_1) to the $i\mathcal{O}$ challenger, receives P, and sets $P_{\mathsf{crs}} = P$.
6. Algorithm $\mathcal{B}$ constructs analogous circuit pairs using the programs from Figure 18 and Figure 20 for each honest user registration query.
7. Algorithm $\mathcal{B}$ answers all queries using the returned obfuscated programs and outputs $\mathcal{A}$'s guess b' to the $i\mathcal{O}$ challenger.

If the challenger bit $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}_q^b(\mathcal{A})$. If $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}_{q,\mathsf{puncture}}^b(\mathcal{A})$.

Next, we verify the functional equivalence of the circuit pairs to prove admissibility. If $\widetilde{\mathsf{pk}}_{q+1} \notin U$, the first and second circuits compute the accumulated key Γ identically and output the correctly mapped base. If $\widetilde{\mathsf{pk}}_{q+1} \in U$, the first circuits evaluate the unpunctured key, yielding a product $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, (\mathsf{PPRF.Eval}(K, \mathsf{pk}))_{\mathsf{pk} \in U})$, and output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_t))$. The second circuits use the punctured key to compute $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\mathsf{PPRF.Eval}(K^*, \mathsf{pk}))_{\mathsf{pk} \in U \setminus \{\widetilde{\mathsf{pk}}_{q+1}\}})$ and output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_t))$. By construction, $z_t = v_t^{\mathsf{PPRF.Eval}(K, \mathsf{pk})}$, and by the composability and commutativity properties of Π_{kaPRF}, we have $\mathsf{Compose}(\mathsf{pp}, \Gamma, v_t) = \mathsf{Compose}(\mathsf{pp}, \Gamma', z_t)$. Since $\widetilde{\mathsf{pk}}_{q+1}$ appears in U only once (the circuit aborts if there are repetitions), the outputs are identical. Thus, the circuits are functionally equivalent, and the difference is bounded by $\mathsf{negl}(\lambda)$. □

Claim 6.21. *Assume that* Π_{PPRF} *satisfies punctured pseudorandomness. Then for any efficient adversary* $\mathcal{A}$ *and any* $q \in \{0, \ldots, Q-1\}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{D}^b_{q,\mathsf{puncture}}(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b_{q,\mathsf{uniform}}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the punctured pseudorandomness game of Π_{PPRF} as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the setup phase, generates pp, ek and pad, and computes the target key $\widetilde{\mathsf{pk}}_{q+1}$. Algorithm $\mathcal{B}$ submits $\widetilde{\mathsf{pk}}_{q+1}$ to the Π_{PPRF} challenger and receives the punctured key K^* and a challenge evaluation value $y^* \in \mathcal{K}$.
3. Algorithm $\mathcal{B}$ sets $\beta = y^*$ and computes the hardcoded elements $z_t = \mathsf{Compose}(\mathsf{pp}, \beta, v_t)$ for all $t \in \{1, \ldots, N\}$.
4. Algorithm $\mathcal{B}$ uses K^* and $\mathbf{z}$ to locally obfuscate the programs exactly as specified in the experiments. Algorithm $\mathcal{B}$ answers all queries using the table T_{users} and outputs $\mathcal{A}$'s guess b'.

If $y^* = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1})$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,\mathsf{puncture}}(\mathcal{A})$. If y^* is sampled uniformly from the valid range $\mathcal{K}$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,\mathsf{uniform}}(\mathcal{A})$. By the punctured pseudorandomness of Π_{PPRF}, the distinguishing advantage of algorithm $\mathcal{B}$ is bounded by a negligible function $\mathsf{negl}(\lambda)$, and the claim follows. □

Claim 6.22. *Assume that* $i\mathcal{O}$ *is a secure indistinguishability obfuscator and* kaPRF *satisfies approximate evaluation correctness with error parameter* $\lambda_s = \lambda + 3\lambda N$*. Then for any efficient adversary* $\mathcal{A}$ *and any* $q \in \{0, \ldots, Q-1\}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{D}^b_{q,\mathsf{uniform}}(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b_{q,\mathsf{noise}}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the setup exactly as in $\mathsf{D}^b_{q,\mathsf{uniform}}$, generating $\mathsf{pp}, K^*, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_{q+1}, \beta \leftarrow \mathsf{kaPRF.KeyGen}(\mathsf{pp})$, and the bases $\mathbf{v}$.
3. Algorithm $\mathcal{B}$ computes for all $t \in \{1, \ldots, N\}$ the exact elements $z_t = \mathsf{Compose}(\mathsf{pp}, \beta, v_t)$ and the noisy elements $z'_t \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_t))$. Let $\mathbf{z} = (z_1, \ldots, z_N)$ and $\mathbf{z}' = (z'_1, \ldots, z'_N)$.
4. Algorithm $\mathcal{B}$ constructs circuit pairs for both the CRS and all honest user keys. The first circuits evaluate the programs hardcoded with the exact vector $\mathbf{z}$. The second circuits evaluate the identical programs hardcoded with the noisy vector $\mathbf{z}'$.
5. Algorithm $\mathcal{B}$ answers all queries using the returned obfuscated programs and outputs $\mathcal{A}$'s guess b' to the $i\mathcal{O}$ challenger.

If the bit $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,\mathsf{uniform}}(\mathcal{A})$. If the bit $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,\mathsf{noise}}(\mathcal{A})$.

Next, we prove that $\mathcal{B}$ is an admissible $i\mathcal{O}$ adversary by showing functional equivalence for all circuits submitted by $\mathcal{B}$. A functional discrepancy between the circuit pairs occurs if and only if there exists a possible value of Γ' and t such that $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_t)) \neq \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z'_t))$. Fix any list $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ that includes $\widetilde{\mathsf{pk}}_{q+1}$, and denote $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \neq \widetilde{\mathsf{pk}}_{q+1}$, and let $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \widetilde{\mathsf{pk}}_{q+1}})$. Since each v_t is sampled uniformly from $\mathcal{X}$, then $z_t = \mathsf{Compose}(\mathsf{pp}, \beta, v_t)$ is also

uniformly distributed by the bijection property of kaPRF. This allows us to invoke the approximate evaluation correctness property of kaPRF to conclude that a discrepancy between z_t and z'_t happens with probability at most $2^{-\lambda_s}$ for each v_t. By taking a union bound over all z_t, we get $N \cdot 2^{-\lambda_s}$ probability of a discrepancy for any fixed list U. Next, we take a union bound over all possible lists U: the list consists of $N-1$ public keys (other than $\widetilde{\mathsf{pk}}_{q+1}$), each of which has $2^{3\lambda}$ possible values, therefore we have less than $2^{3\lambda N}$ possible values of U. Therefore the probability that there exists an input for which there is a discrepancy is at most $N \cdot 2^{-\lambda_s} \cdot 2^{3\lambda N} \leq N \cdot 2^{-\lambda}$. Since algorithm $\mathcal{B}$ is an admissible adversary with overwhelming probability, the difference between the hybrids is bounded by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security. □

Proposition 6.23. *For any parameter $b \in \{0,1\}$ and any $q \in \{0,\ldots,Q-1\}$, the experiment $\mathsf{D}^b_{q,\mathsf{noise}}$ is identical to $\mathsf{D}^b_{q,n}$ for $n = 1$.*

Proof. In the experiment $\mathsf{D}^b_{q,1}$, the assignment defines $z_t \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{t-1}))$ for $t < 1$, and $z_t \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_t))$ for $t \geq 1$. Because the evaluation index t ranges from 1 to $N-1$, the condition $t < 1$ is never satisfied. Thus, $z_t \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_t))$ for all t, which identically matches the distribution defined in $\mathsf{D}^b_{q,\mathsf{noise}}$. □

Claim 6.24. *Assume that Π_{kaPRF} satisfies weak pseudorandomness. Then for any efficient adversary $\mathcal{A}$, any $q \in \{0,\ldots,Q-1\}$, and any $n \in \{1,\ldots,N-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{D}^b_{q,n}(\mathcal{A}) = 1] - \Pr[\overline{\mathsf{D}}^b_{q,n}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the weak pseudorandomness game of the KA-PRF as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ requests $N-1$ samples (x_t, y_t) from its KA-PRF Sample oracle for indices $t \in \{1,\ldots,N\} \setminus \{n\}$, and receives one challenge pair (x^*, y^*) from the Challenge oracle. The implicit KA-PRF secret acts as the uniform element $\beta \in \mathcal{K}$.
3. Algorithm $\mathcal{B}$ simulates the setup phase. It assigns the public domain elements and the hardcoded elements z_t to execute the left-to-right shift as follows:
   - Algorithm $\mathcal{B}$ sets $v_N = x_N$ and $z_N = y_N$.
   - For $t < n$: algorithm $\mathcal{B}$ sets $v_{t-1} = x_t$ and $z_t = y_t$.
   - For $t > n$: algorithm $\mathcal{B}$ sets $v_t = x_t$ and $z_t = y_t$.
   - For $t = n$: algorithm $\mathcal{B}$ sets $v_n = x^*$ and $z_n = y^*$.
4. Algorithm $\mathcal{B}$ explicitly samples the unassigned domain element v_{n-1} uniformly at random via $\mathsf{kaPRF.SampleIn}(\mathsf{pp})$. Because the assignment routes z_t up to v_{n-2} and down to v_{n+1}, the domain element v_{n-1} is unbound.
5. Algorithm $\mathcal{B}$ locally obfuscates the programs using these values, answers all queries, and outputs $\mathcal{A}$'s guess b'.

If the challenge y^* is a true noisy evaluation $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x^*))$, then z_n is distributed exactly as $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_n))$. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,n}(\mathcal{A})$. If the challenge y^* is a uniformly random domain element in $\mathcal{X}$, then $z_n = y$. In this case, algorithm $\mathcal{B}$ perfectly simulates $\overline{\mathsf{D}}^b_{q,n}(\mathcal{A})$. By the weak pseudorandomness of the KA-PRF, the distinguishing advantage is bounded by $\mathsf{negl}(\lambda)$. □

Claim 6.25. *Assume that Π_{kaPRF} satisfies weak pseudorandomness. Then for any efficient adversary $\mathcal{A}$, any $q \in \{0,\ldots,Q-1\}$, and any $n \in \{1,\ldots,N-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\overline{\mathsf{D}}^b_{q,n}(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b_{q,n+1}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the weak pseudorandomness game of the KA-PRF as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ requests $N-1$ samples (x_t, y_t) for indices $t \in \{1,\dots,N\} \setminus \{n\}$, and receives one challenge pair (x^*, y^*).
3. Algorithm $\mathcal{B}$ assigns the bases and hardcoded elements to shift index n to the new domain element v_{n-1}:
   - Algorithm $\mathcal{B}$ sets $v_N = x_N$ and $z_N = y_N$.
   - For $t < n$: algorithm $\mathcal{B}$ sets $v_{t-1} = x_t$ and $z_t = y_t$.
   - For $t > n$: algorithm $\mathcal{B}$ sets $v_t = x_t$ and $z_t = y_t$.
   - For $t = n$: algorithm $\mathcal{B}$ sets $v_{n-1} = x^*$ and $z_n = y^*$.
4. Algorithm $\mathcal{B}$ explicitly samples the unassigned domain element v_n uniformly at random via $\mathsf{kaPRF.SampleIn}(\mathsf{pp})$. Because the shift mapped index n to use v_{n-1}, the domain element v_n is now completely unbound, preventing any assignment collisions.
5. Algorithm $\mathcal{B}$ locally obfuscates the programs using these values, answers all queries, and outputs $\mathcal{A}$'s guess b'.

If the challenge y^* is a uniformly random domain element in $\mathcal{X}$, then z_n is uniform. In this case, algorithm $\mathcal{B}$ perfectly simulates $\overline{\mathsf{D}}^b_{q,n}(\mathcal{A})$. If the challenge y^* is a true noisy evaluation, then z_n is distributed exactly as $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{n-1}))$. This successfully completes the shift for index n. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,n+1}(\mathcal{A})$. By the weak pseudorandomness of the KA-PRF, the distinguishing advantage is bounded by $\mathsf{negl}(\lambda)$. □

Claim 6.26. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator, and Π_{kaPRF} satisfies the approximate evaluation correctness with error $\lambda_s = \lambda + 3\lambda N$. Then for any efficient adversary $\mathcal{A}$ and any $q \in \{0,\dots,Q-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{D}^b_{q,N}(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b_{q,\mathsf{exact}}(\mathcal{A}) = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the experiment exactly as in $\mathsf{D}^b_{q,N}$. At the completion of the inner loop ($n = N$), all elements have been shifted such that $z_t \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_{t-1}))$ for $t \in \{1,\dots,N-1\}$, and $z_N \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v_N))$.
3. Algorithm $\mathcal{B}$ constructs the noisy vector $\mathbf{z}$ and the exact vector $\mathbf{z}'$, computing $z'_t = \mathsf{Compose}(\mathsf{pp}, \beta, v_{t-1})$ and $z'_N = \mathsf{Compose}(\mathsf{pp}, \beta, v_N)$.
4. For the CRS and each honest user key, algorithm $\mathcal{B}$ submits a circuit pair evaluating the punctured programs. The first circuits are hardcoded with $\mathbf{z}$, and the second circuits are hardcoded with $\mathbf{z}'$.
5. Algorithm $\mathcal{B}$ uses the returned obfuscated programs to answer $\mathcal{A}$'s queries, and finally outputs $\mathcal{A}$'s guess b'.

If the $i\mathcal{O}$ challenger bit $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,N}(\mathcal{A})$. If the bit $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,\mathsf{exact}}(\mathcal{A})$.

Next, we prove admissibility. Because all mapped domain elements are distributed uniformly in $\mathcal{X}$, the approximate evaluation correctness property guarantees that the deterministic evaluation will absorb the noise for all valid products Γ' with overwhelming probability. The circuits are therefore functionally equivalent. By $i\mathcal{O}$ security, the advantage is bounded by $\mathsf{negl}(\lambda)$. □

Claim 6.27. *Assume that* Π_{PPRF} *satisfies punctured pseudorandomness. Then for any efficient adversary* $\mathcal{A}$ *and any* $q \in \{0, \ldots, Q-1\}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{D}^b_{q,\mathsf{exact}}(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b_{q,\mathsf{pseudorandom}}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the punctured pseudorandomness of Π_{PPRF} as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the setup phase and computes the target public key $\widetilde{\mathsf{pk}}_{q+1}$. Algorithm $\mathcal{B}$ submits $\widetilde{\mathsf{pk}}_{q+1}$ to the Π_{PPRF} challenger and receives the punctured key K^* and a challenge evaluation value $y^* \in \mathcal{K}$.
3. Algorithm $\mathcal{B}$ sets $\beta = y^*$ and computes the exact elements $z_t = \mathsf{Compose}(\mathsf{pp}, \beta, v_{t-1})$ for $t \in \{1, \ldots, N-1\}$, and $z_N = \mathsf{Compose}(\mathsf{pp}, \beta, v_N)$.
4. Algorithm $\mathcal{B}$ locally obfuscates the programs using K^* and $\mathbf{z}$, answers all queries using the table T_{users}, and outputs $\mathcal{A}$'s guess b'.

If y^* is sampled uniformly from the valid range $\mathcal{K}$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,\mathsf{exact}}(\mathcal{A})$. If y^* is the true evaluation $\mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1})$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,\mathsf{pseudorandom}}(\mathcal{A})$. By the punctured pseudorandomness of Π_{PPRF}, the distinguishing advantage is bounded by $\mathsf{negl}(\lambda)$. □

Claim 6.28. *Assume that* $i\mathcal{O}$ *is a secure indistinguishability obfuscator. Then for any efficient adversary* $\mathcal{A}$ *and any* $q \in \{0, \ldots, Q-1\}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{D}^b_{q,\mathsf{pseudorandom}}(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b_{q+1}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the setup phase, computing the full unpunctured key K, the target key $\widetilde{\mathsf{pk}}_{q+1}$, the punctured key K^*, and the shifted hardcoded elements using the true evaluation $\beta = \mathsf{PPRF.Eval}(K, \widetilde{\mathsf{pk}}_{q+1})$. Let $\mathbf{z} = (z_1, \ldots, z_N)$.
3. For the CRS program, algorithm $\mathcal{B}$ submits a pair of circuits to the challenger: circuit C_0 computes the punctured program hardcoded with $(\mathsf{pp}, K^*, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q, \widetilde{\mathsf{pk}}_{q+1}, \mathbf{z})$, and circuit C_1 computes the standard, unpunctured program hardcoded with $(\mathsf{pp}, K, \mathbf{v}, \mathsf{ek}, \mathsf{pad}, q+1)$.
4. Algorithm $\mathcal{B}$ similarly constructs circuit pairs for each honest user registration query.
5. Algorithm $\mathcal{B}$ answers all queries using the returned obfuscated programs and outputs $\mathcal{A}$'s guess b'.

If the bit $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q,\mathsf{pseudorandom}}(\mathcal{A})$. If the bit $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_{q+1}(\mathcal{A})$.

Next, we verify the functional equivalence to prove admissibility. If user $q+1 \notin U$, both circuits compute the accumulated key Γ and select the base identically. If user $q+1 \in U$, let k be the total honest count and k' be the count of labels $\leq q$. The first circuits compute the partial accumulated key Γ', identify that $\widetilde{\mathsf{pk}}_{q+1} \in U$, and output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_{k-k'})) = \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', \mathsf{Compose}(\mathsf{pp}, \beta, v_{(k-k')-1})))$. Note that user $q+1$ may appear at most once in the set U. In the second circuits, the label threshold is incremented to $q+1$. The condition $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \leq q+1$ will evaluate to true for user $q+1$. Thus, the internal secondary counter k' increments by 1. The new base index explicitly becomes $k-(k'+1) = (k-k')-1$. The unpunctured program natively computes the full accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \beta, \Gamma')$ and outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_{(k-k')-1}))$. The outputs are exactly equivalent for all inputs by the Composability and Commutativity properties. Thus, the circuits are functionally equivalent, algorithm $\mathcal{B}$ is admissible, and the claim follows by $i\mathcal{O}$ security. □

Claim 6.29. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{D}^b_Q(\mathcal{A}) = 1] - \Pr[\mathsf{D}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the setup phase, generating $\mathsf{pp}, K, \mathsf{ek}, \mathsf{pad}$, and the public domain elements $\mathbf{v}$.
3. For the CRS program, algorithm $\mathcal{B}$ submits a pair of circuits: circuit C_0 computes the program crsProg from Figure 17 hardcoded with $q = Q$, and circuit C_1 computes the target final program crsProg defined in D^b. Algorithm $\mathcal{B}$ creates analogous circuit pairs for all honest user registration queries.
4. Algorithm $\mathcal{B}$ uses the returned obfuscated programs to answer $\mathcal{A}$'s queries, and finally outputs $\mathcal{A}$'s guess b' to the $i\mathcal{O}$ challenger.

If $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b_Q(\mathcal{A})$. If $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b(\mathcal{A})$. Next, we prove admissibility via functional equivalence. In the first circuits, the threshold is set to $q = Q$. Because every honest user inherently possesses a label in $[Q]$, the condition label $\leq Q$ is universally satisfied for all valid honest keys. Consequently, the secondary counter k' exactly mirrors the primary counter k for every possible input set U, ensuring $k - k' = 0$ universally. Thus, if $k < N$, the first circuits output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_0))$. If $k = N$, the first circuits explicitly output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v_N))$. This exactly matches the explicit conditional branching defined in the target programs of D^b. The circuits are functionally equivalent, algorithm $\mathcal{B}$ is admissible, and the claim follows by $i\mathcal{O}$ security. □

Proof of Lemma 6.18. By Claim 6.19, the initial experiment C^b is computationally indistinguishable from D^b_0 via the security of $i\mathcal{O}$. By Claim 6.29, the experiment D^b_Q is computationally indistinguishable from D^b via the security of $i\mathcal{O}$. To bound the distance between D^b_0 and D^b_Q, we analyze the sequence of intermediate hybrids D^b_q for $q \in \{0, \ldots, Q-1\}$ and bound the distance of a single transition from D^b_q to D^b_{q+1}. For any fixed iteration q, the transition consists of the following sequential hybrid steps:

1. By Claim 6.20, the transition to the punctured programs bounds the distance between D^b_q and $\mathsf{D}^b_{q,\mathsf{puncture}}$ by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security.
2. By Claim 6.21, the distance between $\mathsf{D}^b_{q,\mathsf{puncture}}$ and $\mathsf{D}^b_{q,\mathsf{uniform}}$ is bounded by $\mathsf{negl}(\lambda)$ via the punctured pseudorandomness of Π_{PPRF}.
3. By Claim 6.22, the addition of the KA-PRF noise bounds the distance between $\mathsf{D}^b_{q,\mathsf{uniform}}$ and $\mathsf{D}^b_{q,\mathsf{noise}}$ by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security and approximate evaluation correctness.
4. By Proposition 6.23, the experiment $\mathsf{D}^b_{q,\mathsf{noise}}$ is identical to the start of the inner-inner loop $\mathsf{D}^b_{q,1}$.
5. To process the base shift, the sequence iterates n from 1 up to $N-1$. For each iteration n, Claim 6.24 bounds the distance between $\mathsf{D}^b_{q,n}$ and $\overline{\mathsf{D}}^b_{q,n}$ by $\mathsf{negl}(\lambda)$, and Claim 6.25 bounds the distance between $\overline{\mathsf{D}}^b_{q,n}$ and $\mathsf{D}^b_{q,n+1}$ by $\mathsf{negl}(\lambda)$, both via the weak pseudorandomness assumption of the KA-PRF. Thus, iterating this inner loop $N-1$ times bounds the distance between $\mathsf{D}^b_{q,1}$ and the terminal inner state $\mathsf{D}^b_{q,N}$ by $O(N) \cdot \mathsf{negl}(\lambda)$.
6. By Claim 6.26, the removal of the KA-PRF noise bounds the distance between $\mathsf{D}^b_{q,N}$ and $\mathsf{D}^b_{q,\mathsf{exact}}$ by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security and approximate evaluation correctness.
7. By Claim 6.27, the distance between $\mathsf{D}^b_{q,\mathsf{exact}}$ and $\mathsf{D}^b_{q,\mathsf{pseudorandom}}$ is bounded by $\mathsf{negl}(\lambda)$ via the punctured pseudorandomness of Π_{PPRF}.

8. Finally, by Claim 6.28, reverting to the standard unpunctured programs bounds the distance between $\mathsf{D}^b_{q,\text{pseudorandom}}$ and D^b_{q+1} by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security.

Summing the advantages of these intermediate steps, the total distinguishing advantage between D^b_q and D^b_{q+1} is bounded by $O(N) \cdot \mathsf{negl}(\lambda)$. To complete the proof, we apply a standard hybrid argument over the outer sequence from $q = 0$ to $Q - 1$. The cumulative distance between D^b_0 and D^b_Q is bounded by:

$$O(Q \cdot N) \cdot \mathsf{negl}(\lambda).$$

The claim now follows by the fact that Q and N are polynomials in λ.

Hybrid E

In this step, we simply want to remove the base u — the base dedicated to fully labeled lists — from the CRS program. Intuitively, the CRS program only works when at least one of the keys is in the image of the PRG. However, the probability that any labeled key is in the PRG's image is negligible. Here, we require that the PRG sufficiently stretches its input. Therefore with overwhelming probability, the branch corresponding to the base u in the CRS program will never be used, therefore we can safely remove it.

Claim 6.30. *Fix a PDE key* ek *in the support of* $\mathsf{PDE.Gen}(1^\lambda)$. *Let* BadPad *be the event over the uniform choice of* $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$ *that there exists a PRG seed* $s \in \{0,1\}^\lambda$ *and a valid label* $L \in \{0,1\}^\lambda$ *such that* $G(s) = \mathsf{PDE.Enc}(\mathsf{ek}, L) \oplus \mathsf{pad}$. *Then* $\Pr[\mathsf{BadPad}] \leq 2^{-\lambda}$.

Proof. Fix the PDE key ek. For any specific seed $s \in \{0,1\}^\lambda$ and label $L \in \{0,1\}^\lambda$, the values $G(s)$ and $\mathsf{PDE.Enc}(\mathsf{ek}, L)$ are fixed. Because the one-time pad pad is sampled uniformly from $\{0,1\}^{3\lambda}$, the probability that $\mathsf{pad} = G(s) \oplus \mathsf{PDE.Enc}(\mathsf{ek}, L)$ is exactly $2^{-3\lambda}$. Applying a union bound over all 2^λ possible values for the seed s and all 2^λ possible values for the label L, the probability that such a collision exists is at most $2^{2\lambda} \cdot 2^{-3\lambda} = 2^{-\lambda}$ and the claim follows. □

Lemma 6.31. *Assume that* $i\mathcal{O}$ *is a secure indistinguishability obfuscator. Then for any efficient adversary* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{D}^b(\mathcal{A}) = 1] - \Pr[\mathsf{E}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the setup phase, generating $\mathsf{pp} \leftarrow \mathsf{kaPRF.Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$, $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^{3\lambda}, 1^{\ell_{\text{out}}})$ and the PDE key $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$, and sampling $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
3. Algorithm $\mathcal{B}$ samples the base elements $v_0, \ldots, v_N \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$ as specified in D^b. To map to the variables in E^b, algorithm $\mathcal{B}$ defines $u = v_0$ and $v = v_N$.
4. For the CRS program, algorithm $\mathcal{B}$ submits a pair of circuits to the $i\mathcal{O}$ challenger: circuit C_0 computes the original program crsProg from Figure 9, and circuit C_1 computes the modified program crsProg from Figure 11 hardcoded with $(\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad})$. Algorithm $\mathcal{B}$ receives the obfuscated program P and sets $P_{\text{crs}} = P$.
5. Algorithm $\mathcal{B}$ sends $\mathsf{crs} = (1^\lambda, \mathsf{pp}, P_{\text{crs}})$ to $\mathcal{A}$.
6. Algorithm $\mathcal{B}$ uses the generated bases to locally obfuscate the required secret key programs to answer all honest user registration queries exactly as specified in the experiments.
7. Algorithm $\mathcal{B}$ answers all other queries using the internal table T_{users} and finally outputs $\mathcal{A}$'s guess b' to the $i\mathcal{O}$ challenger.

If the $i\mathcal{O}$ challenger bit $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{D}^b(\mathcal{A})$. If the bit $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{E}^b(\mathcal{A})$. Next, we verify the functional equivalence of the circuit pair submitted to the $i\mathcal{O}$ challenger to establish the admissibility of algorithm $\mathcal{B}$. A functional discrepancy between circuit C_0 and circuit C_1 can only occur on an input (s, U, i) where circuit C_1 outputs $\perp$ due to the $k = N$ condition, while circuit C_0 would output a valid element. For circuit C_1 to reach the $k = N$ branch, every public key in the input set U must have a valid label. In particular, the targeted key $\mathsf{pk}_i \in U$ must satisfy $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_i \oplus \mathsf{pad}) = L$ for some valid label $L \in \{0,1\}^\lambda$, which implies $\mathsf{pk}_i = \mathsf{PDE.Enc}(\mathsf{ek}, L) \oplus \mathsf{pad}$. Furthermore, the program explicitly checks that the provided seed evaluates to the target key, requiring $G(s) = \mathsf{pk}_i$. Satisfying both conditions simultaneously requires the equality $G(s) = \mathsf{PDE.Enc}(\mathsf{ek}, L) \oplus \mathsf{pad}$ to hold true. This equality is exactly the definition of the event BadPad. Therefore, conditioned on the event ¬BadPad, no such seed s exists for any valid label $L \in \{0,1\}^\lambda$. Consequently, the logical condition $(k = N \wedge G(s) = \mathsf{pk}_i)$ is universally unsatisfiable for all possible inputs (s, U, i). Conditioned on ¬BadPad, the branch is dead code, the circuits are functionally equivalent, and algorithm $\mathcal{B}$ constitutes an admissible $i\mathcal{O}$ adversary. Since the event BadPad occurs with probability at most $2^{-\lambda}$ by Claim 6.30, the probability that algorithm $\mathcal{B}$ submits functionally inequivalent circuits is negligible. Thus, the distinguishing advantage of $\mathcal{A}$ between D^b and E^b is bounded by the sum of algorithm $\mathcal{B}$'s advantage against the $i\mathcal{O}$ security game and the probability of BadPad. Both are negligible functions of λ, and the claim follows. □

Hybrid F

In Hybrid F, the challenge query is computed directly using the PPRF key, without going through the secret key program. If $i\mathcal{O}$ is functionality preserving, then this change over Hybrid E is only syntactical. The second change is that the challenger computes the secret keys lazily, which is also just syntactical.

Proposition 6.32. *Assume that $i\mathcal{O}$ is functionality-preserving. Then for any adversary $\mathcal{A}$ and any parameter $b \in \{0,1\}$, the experiment E^b is identical to the experiment F^b. Thus,*

$$\Pr[\mathsf{E}^b(\mathcal{A}) = 1] = \Pr[\mathsf{F}^b(\mathcal{A}) = 1].$$

Proof. Let $\mathcal{A}$ be an adversary. The transition from the experiment E^b to F^b introduces two conceptual modifications that exactly preserve the view of the adversary. First, the challenger delays generating the obfuscated secret keys for honest users until an explicit corruption query is made. Because the obfuscation algorithm depends on variables fixed during the setup and registration phases (specifically $\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}$, and the public key $\widetilde{\mathsf{pk}_i}$), this lazy sampling produces identically distributed keys. Second, the challenger computes the challenge value y_0 directly instead of executing the obfuscated secret key $\widetilde{\mathsf{sk}}_{j^*}$. By the functionality-preserving property of the indistinguishability obfuscator, evaluating $\widetilde{\mathsf{sk}}_{j^*}$ on the challenge set U^* yields the exact output of the underlying program skProg'. Because every public key in the valid challenge set U^* corresponds to an uncorrupted honest user, the program's internal label counter will always reach $k = N$. For this specific branch, the program is defined to compute the accumulated key Γ and output $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$. Because the challenger in F^b directly executes this identical mathematical operation, the resulting challenge values are perfectly equivalent. Since all query responses remain identical, the proposition follows. □

Hybrid G

In this step, we want to get to Hybrid G, where each corrupted secret key $\mathsf{sk}'_{\mathrm{cc}}$ uses its own base w_{cc}. However, since multiple secret keys may evaluate the shared key for the same set of users, we need to ensure consistency among the corrupted keys. Therefore each corrupted secret key will have the bases corresponding to all previously corrupted keys, and it will always use the base corresponding to the public key in the list which was the first to be corrupted.

To actually get to Hybrid G, we define an inner sequence of hybrids G_c where only the first c corrupted keys have their own bases. The remaining keys fall back to the base u. To get from G_c to G_{c+1}, we first guess the next corrupted user, and then puncture the PPRF on its public key. Now, we can perform a base switching step to introduce a new base for this corrupted user.

Lemma 6.33. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator, Π_{PPRF} satisfies punctured pseudorandomness, and Π_{kaPRF} satisfies approximate evaluation correctness with error parameter $\lambda_s = \lambda + 3\lambda N$ and weak pseudorandomness. Then for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{F}^b(\mathcal{A}) = 1] - \Pr[\mathsf{G}^b(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

To do so, we define a sequence of inner hybrids G^b_c for $c \in \{0, \ldots, C\}$ to show that F^b and G^b are computationally indistinguishable. G^b_0 is identical to F^b, and G^b_C is identical to G^b.

- G^b_c (for $c \in \{0, \ldots, C\}$): This is the same as F^b, except that the challenger maintains a list L of the first c corruptions and assigns them fresh bases. The secret keys are generated differently depending on whether the corruption index is less than or equal to c. Both programs include the check for "unlabeled" keys (using domain element v if the set is not fully labeled). Specifically:

  1. During the setup phase, the challenger initializes a counter $\mathsf{cc} \leftarrow 0$ and an empty ordered list $L \leftarrow \emptyset$.
  2. The challenger modifies how it responds to **Honest user corruption** queries as follows:
     (a) The adversary sends an index i'.
     (b) The challenger looks for the entry $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \mathsf{uncorrupted})$ in T_{users}.
     (c) If no such entry exists, outputs $\perp$.
     (d) The challenger increments $\mathsf{cc} \leftarrow \mathsf{cc} + 1$.
     (e) If $\mathsf{cc} \leq c$:
        – The challenger samples a fresh base $w_{\mathsf{cc}} \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$.
        – The challenger appends $(\mathsf{cc}, \widetilde{\mathsf{pk}}_{i'}, w_{\mathsf{cc}})$ to the list L.
        – The challenger samples $\widetilde{\mathsf{sk}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_{i'}, L])$ where skProg' is described in Figure 21.
     (f) Else ($\mathsf{cc} > c$):
        – The challenger samples $\widetilde{\mathsf{sk}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_{i'}, L])$ where skProg' is described in Figure 22.
     (g) The challenger updates the entry to $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \mathsf{corrupted})$.
     (h) The challenger sends $\widetilde{\mathsf{sk}}_{i'}$ to $\mathcal{A}$.
  3. The challenger answers the **Challenge** query exactly as in F^b, using the domain element u.

Constants: KA-PRF parameters pp, PPRF key K, domain element $v \in \mathcal{X}$, PDE key ek, pad pad, public key pk, list L.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. If $k < N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.
9. Else ($k = N$):
10. Finds the minimum cc' such that $(\mathsf{cc}', \mathsf{pk}', w) \in L$ and $\mathsf{pk}' \in U$.
11. If such an entry exists, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, w))$.
12. Else, outputs $\perp$.

Figure 21: Program $\mathsf{skProg}'[\mathsf{pp}, K, v, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, L]$.

Constants: KA-PRF parameters pp, PPRF key K, domain elements $u, v \in \mathcal{X}$, PDE key ek, pad pad, public key pk, list L.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. Computes $\gamma_j = \mathsf{PPRF.Eval}(K, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
7. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
8. If $k < N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.
9. Else ($k = N$):
10. Finds the minimum cc' such that $(\mathsf{cc}', \mathsf{pk}', w) \in L$ and $\mathsf{pk}' \in U$.
11. If such an entry exists, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, w))$.
12. Else, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$.

Figure 22: Program $\mathsf{skProg}'[\mathsf{pp}, K, u, v, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, L]$.

- $\mathsf{G}^b_{c,1}$: This is the same as G^b_c, except that the challenger guesses the index of the user that will be corrupted in the $(c+1)$-th corruption query. Specifically:
  1. During the setup phase, the challenger additionally samples a uniformly random index $i^* \xleftarrow{\mathrm{R}} [Q]$.
  2. The challenger modifies how it responds to **Honest user corruption** queries as follows:

(a) The adversary sends an index i'.
(b) The challenger looks for the entry $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \text{uncorrupted})$ in T_{users}.
(c) If no such entry exists, outputs $\perp$.
(d) The challenger increments $\mathsf{cc} \leftarrow \mathsf{cc} + 1$.
(e) If $\mathsf{cc} = c + 1$ and $i' \neq i^*$, the experiment aborts and outputs a uniformly random bit $b' \xleftarrow{R} \{0,1\}$.
(f) Otherwise, the challenger proceeds exactly as in G^b_c.

- $\mathrm{G}^b_{c,2}$: This is the same as $\mathrm{G}^b_{c,1}$, except that the challenger punctures the PPRF key at pk^* and hardcodes the exact evaluations into all programs. Specifically:

1. During the setup phase, the challenger modifies its operations as follows:
   (a) Computes the key $\mathsf{pk}^* = \mathsf{PDE.Enc}(\mathsf{ek}, i^*) \oplus \mathsf{pad}$.
   (b) Computes $\beta = \mathsf{PPRF.Eval}(K, \mathsf{pk}^*) \in \mathcal{K}$.
   (c) Punctures the PPRF key $K^* \leftarrow \mathsf{PPRF.Puncture}(K, \mathsf{pk}^*)$.
   (d) Initializes an empty list $L_{\text{exp}} \leftarrow \emptyset$.
   (e) Computes the hardcoded domain elements $z_v = \mathsf{Compose}(\mathsf{pp}, \beta, v)$ and the fallback bridge $z_u = \mathsf{Compose}(\mathsf{pp}, \beta, u)$.
   (f) Samples $P_{\text{crs}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\text{size}_{\text{crs}}}, \mathsf{crsProg}''[\mathsf{pp}, K^*, \mathsf{pk}^*, v, z_v, \mathsf{ek}, \mathsf{pad}])$ where $\mathsf{crsProg}''$ is described in Figure 23.
2. The challenger modifies how it responds to **Honest user corruption** queries as follows:
   (a) The adversary sends an index i'.
   (b) The challenger looks for the entry $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \text{uncorrupted})$ in T_{users}.
   (c) If no such entry exists, outputs $\perp$.
   (d) The challenger increments $\mathsf{cc} \leftarrow \mathsf{cc} + 1$.
   (e) If $\mathsf{cc} \leq c$:
      – The challenger samples a fresh base $w_{\mathsf{cc}} \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$ and computes $z_{\mathsf{cc}} = \mathsf{Compose}(\mathsf{pp}, \beta, w_{\mathsf{cc}})$.
      – The challenger appends $(\mathsf{cc}, \widetilde{\mathsf{pk}}_{i'}, z_{\mathsf{cc}})$ to the list L_{exp}.
      – The challenger samples $\widetilde{\mathsf{sk}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\text{size}_{\text{sk}}}, \mathsf{skProg}'_{\text{stable}}[\mathsf{pp}, K^*, \mathsf{pk}^*, v, z_v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_{i'}, L_{\text{exp}}])$ where $\mathsf{skProg}'_{\text{stable}}$ is described in Figure 24 .
   (f) Else if $\mathsf{cc} = c + 1$:
      – (If $i' \neq i^*$, the experiment aborts and outputs a random bit as in $\mathrm{G}^b_{c,1}$).
      – The challenger samples $\widetilde{\mathsf{sk}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\text{size}_{\text{sk}}}, \mathsf{skProg}'_{\text{unstable}}[\mathsf{pp}, K^*, \mathsf{pk}^*, u, v, z_u, z_v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_{i'}, L_{\text{exp}}])$ where $\mathsf{skProg}'_{\text{unstable}}$ is described in Figure 25 .
   (g) Else ($\mathsf{cc} > c + 1$):
      – The challenger samples $\widetilde{\mathsf{sk}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\text{size}_{\text{sk}}}, \mathsf{skProg}'_{\text{unstable}}[\mathsf{pp}, K^*, \mathsf{pk}^*, u, v, z_u, z_v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_{i'}, L_{\text{exp}}])$ where $\mathsf{skProg}'_{\text{unstable}}$ is described in Figure 25 .
   (h) The challenger updates the entry to $(i', \widetilde{\mathsf{pk}}_{i'}, \widetilde{\mathsf{sk}}_{i'}, \text{corrupted})$.
   (i) The challenger sends $\widetilde{\mathsf{sk}}_{i'}$ to $\mathcal{A}$.
3. The challenger answers the **Challenge** query by computing $\gamma_j = \mathsf{PPRF.Eval}(K^*, \widetilde{\mathsf{pk}}_j) \in \mathcal{K}$ for all $j \in \mathcal{I}^*$, computing $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{j \in \mathcal{I}^*})$, and setting $y_0 = \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$. Note that using the punctured key is fine since the experiment aborts and outputs a random bit if i^* is in the challenge set.

Constants: KA-PRF parameters pp, Punctured PPRF key K^*, target pk^*, domain element $v \in \mathcal{X}$, element $z_v \in \mathcal{X}$, PDE key ek, pad pad.
Inputs: A PRG seed s, a set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $G(s) \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. If $k = N$, outputs $\perp$.
7. If $\mathsf{pk}^* \notin U$:
8. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
9. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
10. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.
11. Else ($\mathsf{pk}^* \in U$):
12. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \in U \setminus \{\mathsf{pk}^*\}$.
13. Computes $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \mathsf{pk}^*})$.
14. Outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_v))$.

Figure 23: Program $\mathsf{crsProg}''[\mathsf{pp}, K^*, \mathsf{pk}^*, v, z_v, \mathsf{ek}, \mathsf{pad}]$.

Constants: KA-PRF parameters pp, Punctured PPRF key K^*, target pk^*, domain element $v \in \mathcal{X}$, element $z_v \in \mathcal{X}$, PDE key ek, pad pad, public key pk, list L_{exp}.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. If $\mathsf{pk}^* \notin U$:
7. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
8. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
9. If $k < N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.
10. Else ($k = N$):
11. Finds the minimum cc' such that $(\mathsf{cc}', \mathsf{pk}', w) \in L_{\mathsf{exp}}$ and $\mathsf{pk}' \in U$.
12. If such an entry exists, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, w))$.
13. Else, outputs $\perp$.
14. Else ($\mathsf{pk}^* \in U$):
15. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \in U \setminus \{\mathsf{pk}^*\}$.
16. Computes $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \mathsf{pk}^*})$.
17. If $k < N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_v))$.
18. Else ($k = N$):
19. Finds the minimum cc' such that $(\mathsf{cc}', \mathsf{pk}', z') \in L_{\mathsf{exp}}$ and $\mathsf{pk}' \in U$.
20. If such an entry exists, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z'))$.
21. Else, outputs $\perp$.

Figure 24: Program $\mathsf{skProg}'_{\mathsf{stable}}[\mathsf{pp}, K^*, \mathsf{pk}^*, v, z_v, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, L_{\mathsf{exp}}]$.

Constants: KA-PRF parameters pp, Punctured PPRF key K^*, target pk^*, domain elements $u, v \in \mathcal{X}$, elements $z_u, z_v \in \mathcal{X}$, PDE key ek, pad pad, public key pk, list L_{exp}.
Inputs: A set of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, and an index $i \in [N]$.

1. If U has repetitions, outputs $\perp$.
2. If $\mathsf{pk} \neq \mathsf{pk}_i$, outputs $\perp$.
3. Initializes counter $k \leftarrow 0$.
4. For each $j \in [N]$:
5. If $\mathsf{PDE.Dec}(\mathsf{ek}, \mathsf{pk}_j \oplus \mathsf{pad}) \in [Q]$, increments $k \leftarrow k + 1$.
6. If $\mathsf{pk}^* \notin U$:
7. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $j \in [N]$.
8. Computes the accumulated key $\Gamma = \mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_N)$.
9. If $k < N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, v))$.
10. Else ($k = N$):
11. Finds the minimum cc' such that $(\mathsf{cc}', \mathsf{pk}', w) \in L_{\mathsf{exp}}$ and $\mathsf{pk}' \in U$.
12. If such an entry exists, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, w))$.
13. Else, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$.
14. Else ($\mathsf{pk}^* \in U$):
15. Computes $\gamma_j = \mathsf{PPRF.Eval}(K^*, \mathsf{pk}_j) \in \mathcal{K}$ for all $\mathsf{pk}_j \in U \setminus \{\mathsf{pk}^*\}$.
16. Computes $\Gamma' = \mathsf{Accumulate}(\mathsf{pp}, (\gamma_j)_{\mathsf{pk}_j \neq \mathsf{pk}^*})$.
17. If $k < N$, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_v))$.
18. Else ($k = N$):
19. Finds the minimum cc' such that $(\mathsf{cc}', \mathsf{pk}', z') \in L_{\mathsf{exp}}$ and $\mathsf{pk}' \in U$.
20. If such an entry exists, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z'))$.
21. Else, outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_u))$.

Figure 25: Program $\mathsf{skProg}'_{\mathsf{unstable}}[\mathsf{pp}, K^*, \mathsf{pk}^*, u, v, z_u, z_v, \mathsf{ek}, \mathsf{pad}, \mathsf{pk}, L_{\mathsf{exp}}]$.

- $\mathrm{G}^b_{c,3}$: This is the same as $\mathrm{G}^b_{c,2}$, except that the challenger applies the noise function to all hardcoded elements. Specifically:
  1. In the setup phase, the challenger computes $z_v \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, v))$ and $z_u \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, u))$.
  2. When generating elements for L_{exp}, the challenger computes $z_{\mathsf{cc}} \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, w_{\mathsf{cc}}))$.
- $\mathrm{G}^b_{c,4}$: This is the same as $\mathrm{G}^b_{c,3}$, except that the challenger replaces β with a uniformly random base key element. Specifically:
  1. During the setup phase, instead of computing $\beta = \mathsf{PPRF.Eval}(K, \mathsf{pk}^*) \in \mathcal{K}$, the challenger samples $\beta \leftarrow \mathsf{kaPRF.KeyGen}(\mathsf{pp})$.
- $\mathrm{G}^b_{c,5}$: This is the same as $\mathrm{G}^b_{c,4}$, except that the challenger replaces the fallback bridge element z_u with a uniformly random element in $\mathcal{X}$. Specifically:
  1. During the setup phase, instead of computing $z_u \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, u))$, the challenger samples a uniformly random element $y \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$ and sets $z_u = y$.

- $\mathrm{G}^b_{c,6}$: This is the same as $\mathrm{G}^b_{c,5}$, except that the challenger replaces z_u with a fresh, independent noisy evaluation sample. Specifically:
  1. During the setup phase, the challenger samples a fresh base $w_{c+1} \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$.
  2. The challenger computes $z_u \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, w_{c+1}))$.
- $\mathrm{G}^b_{c,7}$: This is the same as $\mathrm{G}^b_{c,6}$, except that the challenger extends the lists L and L_{exp}, and generates the secret key for the $(c+1)$-th corruption using the stable program. Specifically:
  1. Upon receiving the $(c+1)$-th **Honest user corruption** query (and verifying $i' = i^*$), the challenger extends the lists: it appends $(c+1, \mathsf{pk}^*, w_{c+1})$ to the list L, and appends $(c+1, \mathsf{pk}^*, z_{c+1})$ to the list L_{exp}, where $z_{c+1} = z_u$.
  2. The challenger samples $\widetilde{\mathsf{sk}} \leftarrow i\mathcal{O}(1^\lambda, 1^{\mathsf{size}_{\mathsf{sk}}}, \mathsf{skProg}'_{\mathsf{stable}}[\mathsf{pp}, K^*, \mathsf{pk}^*, v, z_v, \mathsf{ek}, \mathsf{pad}, \widetilde{\mathsf{pk}}_{i'}, L_{\mathsf{exp}}])$ where $\mathsf{skProg}'_{\mathsf{stable}}$ is described in Figure 24.
  3. For any subsequent corruptions ($\mathsf{cc} > c+1$), the challenger continues to use the unstable program, but now provides it with the newly extended list L_{exp}.
- $\mathrm{G}^b_{c,8}$: This is the same as $\mathrm{G}^b_{c,7}$, except that the challenger removes the noise from all pre-computed elements. Specifically:
  1. In the setup phase, the challenger computes the exact element $z_v = \mathsf{Compose}(\mathsf{pp}, \beta, v)$.
  2. It also computes exact elements $z_{\mathsf{cc}} = \mathsf{Compose}(\mathsf{pp}, \beta, w_{\mathsf{cc}})$ for all entries in L_{exp} (including $z_{c+1} = \mathsf{Compose}(\mathsf{pp}, \beta, w_{c+1})$).
  3. Because the fallback element z_u is now dead code in the punctured branch due to the extended list L_{exp}, the challenger safely reverts it directly back to the exact evaluation $z_u = \mathsf{Compose}(\mathsf{pp}, \beta, u)$ via $i\mathcal{O}$.
- $\mathrm{G}^b_{c,9}$: This is the same as $\mathrm{G}^b_{c,8}$, except that the challenger reverts β to the true PPRF evaluation. Specifically:
  1. During the setup phase, instead of sampling $\beta \leftarrow \mathsf{kaPRF.KeyGen}(\mathsf{pp})$, the challenger computes $\beta = \mathsf{PPRF.Eval}(K, \mathsf{pk}^*) \in \mathcal{K}$.
- $\mathrm{G}^b_{c,10}$: This is the same as $\mathrm{G}^b_{c,9}$, except that the challenger reverts to using the un-punctured PPRF key K and the standard programs, removing the hardcoded elements. Specifically:
  1. During the setup phase, the challenger no longer computes z_v and z_u. It simply samples bases $u, v \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$ and maintains the list L (which now naturally includes w_{c+1}). It still guesses the target index $i^* \xleftarrow{\mathrm{R}} [Q]$.
  2. The challenger responds to **Honest user corruption** queries exactly as in G^b_{c+1} (up to the abort condition for the $(c+1)$-th query).

Proposition 6.34. *For any adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, we have*

$$\left|\Pr[\mathrm{G}^0_{c,1}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^1_{c,1}(\mathcal{A}) = 1]\right| = \frac{1}{Q}\left|\Pr[\mathrm{G}^0_{c}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^1_{c}(\mathcal{A}) = 1]\right|.$$

Proof. The experiment $\mathrm{G}^b_{c,1}$ behaves identically to the experiment G^b_c, except that the challenger samples a uniformly random index $i^* \xleftarrow{\mathrm{R}} [Q]$ during setup and aborts if the $(c+1)$-th corruption query does not target i^*, outputting a random bit. Because i^* is sampled independently and is completely hidden from the adversary until the abort condition is evaluated, the probability that the experiment does not abort is exactly $1/Q$. This holds independently of the adversary's view, and the proposition follows. □

Claim 6.35. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,1}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,2}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the setup phase, computing the target key $\mathsf{pk}^* = \mathsf{PDE.Enc}(\mathsf{ek}, i^*) \oplus \mathsf{pad}$, puncturing the PPRF key to obtain K^*, and computing $\beta = \mathsf{PPRF.Eval}(K, \mathsf{pk}^*) \in \mathcal{K}$.
3. Algorithm $\mathcal{B}$ calculates the exact hardcoded elements $z_v = \mathsf{Compose}(\mathsf{pp}, \beta, v)$ and $z_u = \mathsf{Compose}(\mathsf{pp}, \beta, u)$. When answering corruption queries for $\mathsf{cc} \leq c$, algorithm $\mathcal{B}$ additionally computes $z_{\mathsf{cc}} = \mathsf{Compose}(\mathsf{pp}, \beta, w_{\mathsf{cc}})$.
4. Algorithm $\mathcal{B}$ submits circuit pairs substituting the standard, unpunctured programs from $\mathrm{G}^b_{c,1}$ with their corresponding punctured, hardcoded counterparts defined in $\mathrm{G}^b_{c,2}$ (i.e., programs from Figure 23, Figure 24, and Figure 25).
5. Algorithm $\mathcal{B}$ answers all queries using the returned obfuscated programs and outputs $\mathcal{A}$'s guess b' to the $i\mathcal{O}$ challenger.

If $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,1}(\mathcal{A})$. If $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,2}(\mathcal{A})$. Algorithm $\mathcal{B}$ is an admissible adversary because the program pairs it submits are all functionally equivalent by the functionality preserving of Π_{PRF}, the composability of Π_{kaPRF} and inspection of the programs. The claim follows by $i\mathcal{O}$ security. □

Claim 6.36. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator and Π_{kaPRF} satisfies approximate evaluation correctness with error parameter $\lambda_s = \lambda + 3\lambda N$ Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,2}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,3}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes the experiment as in $\mathrm{G}^b_{c,2}$, explicitly generating both the exact elements $z_{\mathsf{base}} = \mathsf{Compose}(\mathsf{pp}, \beta, \mathsf{base})$ and the noisy elements $z'_{\mathsf{base}} \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, z_{\mathsf{base}})$ for each $\mathsf{base} \in \{u, v\}$. Similarly, algorithm $\mathcal{B}$ samples the exact element $z_{\mathsf{cc}} = \mathsf{Compose}(\mathsf{pp}, \beta, w_{\mathsf{cc}})$ and the noisy element $z'_{\mathsf{cc}} \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, z_{\mathsf{cc}})$ for each corruption query cc.
3. Algorithm $\mathcal{B}$ submits circuit pairs for all programs. The first circuits evaluate the programs hardcoded with the exact elements, and the second circuits evaluate the identical programs hardcoded with the noisy elements.
4. Algorithm $\mathcal{B}$ answers all queries using the returned obfuscated programs and outputs $\mathcal{A}$'s guess b'.

If $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,2}(\mathcal{A})$. If $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,3}(\mathcal{A})$. Next, we prove admissibility. The proof is almost identical to those of Claim 6.11 and Claim 6.22. Namely, a discrepancy happens for any fixed input set U with probability at most $(C+2) \cdot 2^{-\lambda N}$ by the approximate evaluation correctness and a union bound, since there are $C+2$ uniformly sampled bases. As before, the list U has $2^{3\lambda \cdot N}$ possible values, and therefore there exists a discrepancy with negligible probability, proving that $\mathcal{B}$ is admissible with overwhelming probability. The claim follows by $i\mathcal{O}$ security. □

Claim 6.37. *Assume that Π_{PPRF} satisfies punctured pseudorandomness. Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,3}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,4}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the punctured pseudorandomness game of Π_{PPRF} as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ computes the target key pk^* and submits it to the Π_{PPRF} challenger, receiving the punctured key K^* and a challenge value $y^* \in \mathcal{K}$.
3. Algorithm $\mathcal{B}$ sets $\beta = y^*$ and locally computes all noisy hardcoded elements using this value.
4. Algorithm $\mathcal{B}$ obfuscates the programs internally, answers all queries, and outputs $\mathcal{A}$'s guess b'.

If $y^* = \mathsf{PPRF.Eval}(K, \mathsf{pk}^*)$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,3}(\mathcal{A})$. If y^* is sampled uniformly from $\mathcal{K}$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,4}(\mathcal{A})$. By the punctured pseudorandomness of Π_{PPRF}, the distinguishing advantage is bounded by a negligible function $\mathsf{negl}(\lambda)$. □

Claim 6.38. *Assume that Π_{kaPRF} satisfies weak pseudorandomness assumption. Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,4}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,5}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the weak pseudorandomness game of the KA-PRF as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ receives KA-PRF Sample oracle queries, mapping them to the bases and corresponding hardcoded elements for v and w_{cc}.
3. Algorithm $\mathcal{B}$ receives a challenge pair (x^*, y^*) from the Challenge oracle. Algorithm $\mathcal{B}$ sets the fallback base $u = x^*$ and sets the fallback element $z_u = y^*$.
4. Algorithm $\mathcal{B}$ locally obfuscates the programs using these mapped values, answers all queries, and outputs $\mathcal{A}$'s guess b'.

If $y^* \leftarrow \mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x^*))$, then z_u is distributed as $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, u))$. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,4}(\mathcal{A})$. If $y^* \xleftarrow{\mathrm{R}} \mathcal{X}$, then z_u is uniformly random. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,5}(\mathcal{A})$. By the weak pseudorandomness of the KA-PRF, the distinguishing advantage is bounded by $\mathsf{negl}(\lambda)$. □

Π_{kaPRF} satisfies weak pseudorandomness.

Claim 6.39. *Assume that Π_{kaPRF} satisfies weak pseudorandomness. Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,5}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,6}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the weak pseudorandomness game of the KA-PRF as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ receives KA-PRF oracle samples, mapping them to the bases and hardcoded elements for v and w_{cc}. Because the element z_u acts independently as a uniformly random element in $\mathrm{G}^b_{c,5}$, algorithm $\mathcal{B}$ natively samples the base $u \leftarrow \mathsf{kaPRF.SampleIn}(\mathsf{pp})$ completely independent of any oracle samples.

3. Algorithm $\mathcal{B}$ receives a challenge pair (x^*, y^*) from the Challenge oracle. Algorithm $\mathcal{B}$ sets the new base $w_{c+1} = x^*$ and sets the element $z_u = y^*$.

4. Algorithm $\mathcal{B}$ locally obfuscates the programs using these mapped values, answers all queries, and outputs $\mathcal{A}$'s guess b'.

If y^* is a uniformly random element in $\mathcal{X}$, then z_u remains uniform. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,5}(\mathcal{A})$. If y^* is a true noisy evaluation, then z_u is distributed as $\mathsf{kaPRF.Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \beta, w_{c+1}))$, which generates a fresh valid sample strictly mapped to the new base w_{c+1}. In this case, algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,6}(\mathcal{A})$. By the weak pseudorandomness of the KA-PRF, the distinguishing advantage is bounded by $\mathsf{negl}(\lambda)$. □

Claim 6.40. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,6}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,7}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.

2. Algorithm $\mathcal{B}$ simulates the experiment as defined in $\mathrm{G}^b_{c,6}$. When the $(c+1)$-th corruption query is received, algorithm $\mathcal{B}$ defines the extended lists $L_{c+1} = L \cup \{(c+1, \mathsf{pk}^*, w_{c+1})\}$ and $L^{(c+1)}_{\mathsf{exp}} = L_{\mathsf{exp}} \cup \{(c+1, \mathsf{pk}^*, z_{c+1})\}$, where $z_{c+1} = z_u$.

3. For the $(c+1)$-th corruption key, algorithm $\mathcal{B}$ submits a pair of circuits: the first circuit computes the unstable program from Figure 25 hardcoded with the unextended list L_{exp}, and the second circuit computes the stable program from Figure 24 utilizing the extended list $L^{(c+1)}_{\mathsf{exp}}$.

4. For all subsequent corruption keys, algorithm $\mathcal{B}$ submits circuit pairs where the first evaluates the unstable program with the unextended list L_{exp}, and the second evaluates the identical unstable program but with the newly extended list $L^{(c+1)}_{\mathsf{exp}}$.

5. Algorithm $\mathcal{B}$ answers queries using the returned obfuscated programs and outputs $\mathcal{A}$'s guess b'.

If $b_{i\mathcal{O}} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,6}(\mathcal{A})$. If $b_{i\mathcal{O}} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,7}(\mathcal{A})$. Next, we verify the functional equivalence to prove admissibility. For all programs, whenever the minimum cc' is at most c or $k < N$, the programs behave the same. Therefore we analyze the cases where the minimum $\mathsf{cc}' \geq c+1$ and $k = N$:

- For the $(c+1)$-th program, the target key is exactly pk^*, therefore the branch evaluating $\mathsf{pk}^* \notin U$ is logically impossible for any valid execution. Therefore there is no difference in behavior between a stable and unstable program. When evaluated on sets where $\mathsf{pk}^* \in U$, the first circuit fails to find an entry in L_{exp} and outputs the fallback $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_u))$. The second circuit correctly matches pk^* in $L^{(c+1)}_{\mathsf{exp}}$ and outputs the listed value $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma', z_{c+1}))$. Because the values satisfy $z_u = z_{c+1}$, the outputs are identical.

- For programs corresponding to corruptions $> c+1$, the case where $\mathsf{pk}^* \in U$ matches the $(c+1)$-th program. If $\mathsf{pk}^* \notin U$, then there are two cases. If the target key is not present, then regardless of the list used, the program aborts. If the target key is present, then regardless of the list since no key in the list is present and also $\mathsf{pk}^* \notin U$, the program outputs $\mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$.

Thus, all circuit pairs are functionally equivalent, algorithm $\mathcal{B}$ is admissible, and the claim follows by $i\mathcal{O}$ security. □

Claim 6.41. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator and Π_{kaPRF} satisfies approximate evaluation correctness with error parameter $\lambda_s = \lambda + 3\lambda N$. Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,7}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,8}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

The proof is completely analogous to that of Claim 6.36.

Claim 6.42. *Assume that Π_{PPRF} satisfies punctured pseudorandomness. Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,8}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,9}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ against the punctured pseudorandomness of Π_{PPRF}. Algorithm $\mathcal{B}$ submits pk^* to the Π_{PPRF} challenger, receiving the punctured key K^* and a challenge value $y^* \in \mathcal{K}$. Algorithm $\mathcal{B}$ sets $\beta = y^*$ and calculates the exact hardcoded elements. If y^* is sampled uniformly from $\mathcal{K}$, algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,8}(\mathcal{A})$. If $y^* = \mathsf{PPRF.Eval}(K, \mathsf{pk}^*)$, algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,9}(\mathcal{A})$. By the security of Π_{PPRF}, the distinguishing advantage is bounded by $\mathsf{negl}(\lambda)$. □

Claim 6.43. *Assume that $i\mathcal{O}$ is a secure indistinguishability obfuscator. Then for any efficient adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathrm{G}^b_{c,9}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^b_{c,10}(\mathcal{A}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient adversary. We construct an adversary $\mathcal{B}$ for the multi-challenge security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends 1^N and 1^C to algorithm $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ calculates the standard full key programs utilizing the unpunctured PPRF key K, omitting any pre-computed hardcoded elements.
3. Algorithm $\mathcal{B}$ submits circuit pairs substituting the punctured programs from $\mathrm{G}^b_{c,9}$ with their standard unpunctured counterparts from $\mathrm{G}^b_{c,10}$.
4. Algorithm $\mathcal{B}$ uses the returned obfuscated programs to answer queries and outputs $\mathcal{A}$'s guess b'.

If $b_{iO} = 0$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,9}(\mathcal{A})$. If $b_{iO} = 1$, then algorithm $\mathcal{B}$ perfectly simulates $\mathrm{G}^b_{c,10}(\mathcal{A})$. Next, we verify the functional equivalence to prove admissibility. The unpunctured programs compute the full accumulated key Γ using the complete key K, while the punctured programs yield identical values via the explicit pre-calculated elements generated using the identical true evaluation, and the fact that pk^* appears at most once in the set U. The circuits are structurally guaranteed to be functionally equivalent, algorithm $\mathcal{B}$ is admissible, and the claim follows by $i\mathcal{O}$ security. □

Proposition 6.44. *For any adversary $\mathcal{A}$ and any $c \in \{0, \ldots, C-1\}$, we have*

$$\left|\Pr[\mathrm{G}^0_{c,10}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^1_{c,10}(\mathcal{A}) = 1]\right| = \frac{1}{Q}\left|\Pr[\mathrm{G}^0_{c+1}(\mathcal{A}) = 1] - \Pr[\mathrm{G}^1_{c+1}(\mathcal{A}) = 1]\right|.$$

Proof. In the experiment $\mathrm{G}^b_{c,10}$, the maintained list L successfully encompasses the first $c+1$ corruptions, and all generated programs utilize standard unpunctured execution. This operational state is identical to the behavior defined in the base experiment G^b_{c+1}, except that $\mathrm{G}^b_{c,10}$ retains the logic that explicitly aborts and outputs a random bit if the $(c+1)$-th query does not match the uniformly sampled index i^*. Factoring out this independent $1/Q$ guessing probability yields the precise relation between the distinguishing advantages in the respective experiments. □

Proof of Lemma 6.33. By definition, the experiment G_0^b is identical to F^b, and G_C^b is identical to G^b. To bound the distance between G_0^b and G_C^b, we analyze the sequence of intermediate hybrids G_c^b for $c \in \{0, \ldots, C-1\}$ and bound the distance of a single transition from G_c^b to G_{c+1}^b. For any fixed iteration c, the transition consists of the following sequential hybrid steps:

1. By Proposition 6.34, the transition to guessing the index bounds the distance between G_c^b and $\mathsf{G}_{c,1}^b$ scaled by a factor of $1/Q$.
2. By Claim 6.35, the distance between $\mathsf{G}_{c,1}^b$ and $\mathsf{G}_{c,2}^b$ is bounded by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security.
3. By Claim 6.36, the distance between $\mathsf{G}_{c,2}^b$ and $\mathsf{G}_{c,3}^b$ is bounded by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security and approximate evaluation equivalence.
4. By Claim 6.37, the distance between $\mathsf{G}_{c,3}^b$ and $\mathsf{G}_{c,4}^b$ is bounded by $\mathsf{negl}(\lambda)$ via the punctured pseudorandomness of Π_{PPRF}.
5. By Claim 6.38, the distance between $\mathsf{G}_{c,4}^b$ and $\mathsf{G}_{c,5}^b$ is bounded by $\mathsf{negl}(\lambda)$ via weak pseudorandomness.
6. By Claim 6.39, the distance between $\mathsf{G}_{c,5}^b$ and $\mathsf{G}_{c,6}^b$ is bounded by $\mathsf{negl}(\lambda)$ via weak pseudorandomness.
7. By Claim 6.40, the distance between $\mathsf{G}_{c,6}^b$ and $\mathsf{G}_{c,7}^b$ is bounded by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security.
8. By Claim 6.41, the distance between $\mathsf{G}_{c,7}^b$ and $\mathsf{G}_{c,8}^b$ is bounded by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security and approximate evaluation equivalence.
9. By Claim 6.42, the distance between $\mathsf{G}_{c,8}^b$ and $\mathsf{G}_{c,9}^b$ is bounded by $\mathsf{negl}(\lambda)$ via the punctured pseudorandomness of Π_{PPRF}.
10. By Claim 6.43, the distance between $\mathsf{G}_{c,9}^b$ and $\mathsf{G}_{c,10}^b$ is bounded by $\mathsf{negl}(\lambda)$ via $i\mathcal{O}$ security.
11. By Proposition 6.44, the transition to G_{c+1}^b maintains the $1/Q$ scaling factor.

Summing the advantages of these intermediate steps and adjusting for the $1/Q$ scaling factor, the distinguishing advantage between G_c^b and G_{c+1}^b is bounded by $O(Q) \cdot \mathsf{negl}(\lambda)$. To complete the proof, we apply a standard hybrid argument over the outer sequence from $c = 0$ to $C - 1$. The cumulative distance between G_0^b and G_C^b is bounded by:

$$O(C \cdot Q) \cdot \mathsf{negl}(\lambda).$$

The claim now follows by the fact that C and Q are polynomials in λ.

Statistical Hardness of Hybrid G

We argue that G^0 is statistically identical to G^1 by the evaluation uniformity of Π_{kaPRF}. Specifically, we rely on the challenge base u being completely hidden from the adversary.

Proposition 6.45. *If Π_{kaPRF} satisfies evaluation uniformity then for any adversary $\mathcal{A}$, we have*

$$\left|\Pr[\mathsf{G}^0(\mathcal{A}) = 1] - \Pr[\mathsf{G}^1(\mathcal{A}) = 1]\right| \leq 0.$$

Proof. In the final experiment G^b (which is identical to G_C^b), the challenger successfully assigns a fresh, independent base w_{cc} to every one of the C requested corruptions. Because the adversary is inherently restricted to making at most C corruption queries, every secret key generated by the challenger natively utilizes the stable program from Figure 21. The unstable program from Figure 22 is completely unutilized.

Consequently, the challenge base u is entirely absent from the CRS and all corrupted secret keys. During the challenge phase, if $b = 0$, the challenger computes the exact challenge element $y_0 = \mathsf{kaPRF.Eval}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \Gamma, u))$. Because u is sampled uniformly from $\mathcal{X}$ via $\mathsf{kaPRF.SampleIn}(\mathsf{pp})$, the Evaluation Uniformity property of the KA-PRF mandates that evaluating the composition output yields a uniform distribution over the codomain $\mathcal{Y}$. Therefore, the computed value y_0 is statistically identical to the uniformly sampled codomain element y_1 provided when $b = 1$. and the proposition follows. □

Acknowledgments

This work was supported by NSF CNS-2318701 and a Simons Investigator award.

Statement on AI Use

Generative AI was not used to develop any conceptual ideas in this work. Google Gemini 2.5 was used strictly for text copy-editing. The authors manually reviewed and verified all AI-generated text.

References

[ADFMP20] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 411–439. Springer, 2020.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), 2012.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *International workshop on public key cryptography*, pages 501–519. Springer, 2014.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 719–737. Springer, 2012.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Cryptology ePrint Archive*, 2002.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology-ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II 19*, pages 280–300. Springer, 2013.

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

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. Csidh: an efficient post-quantum commutative group action. In *International conference on the theory and application of cryptology and information security*, pages 395–427. Springer, 2018.

[CLW25] Valerio Cini, Russell WF Lai, and Ivy KY Woo. Lattice-based obfuscation from ntru and equivocal lwe. In *Annual International Cryptology Conference*, pages 39–72. Springer, 2025.

[DH76] WHITFIELD DIFFIE and MARTIN E HELLMAN. New directions in cryptography. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 22(6), 1976.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin. Lattice-based post-quantum io from circular security with random opening assumption. In *Annual International Cryptology Conference*, pages 3–38. Springer, 2025.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb {F}_p$, dlin, and prgs in nc^0. In *EUROCRYPT*, 2022.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie–hellman. In *International algorithmic number theory symposium*, pages 385–393. Springer, 2000.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 669–684, 2013.

[KPW17] Venkata Koppula, Andrew Poelstra, and Brent Waters. Universal samplers with fast verification. In *IACR International Workshop on Public Key Cryptography*, pages 525–554. Springer, 2017.

[KWZ22] Venkata Koppula, Brent Waters, and Mark Zhandry. Adaptive multiparty nike. In *Theory of Cryptography Conference*, pages 244–273. Springer, 2022.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.

[NW26] Shafik Nassar and Brent Waters. Adaptive nike for unbounded parties. In *Annual International Cryptology Conference*, pages 350–381. Springer, 2026.

[Rao14] Vanishree Rao. Adaptive multiparty non-interactive key exchange without setup in the standard model. *Cryptology ePrint Archive*, 2014.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

A KA-PRF from Cryptographic Group Actions

We instantiate KA-PRFs from Definition 4.1 using Cryptographic Group Actions (CGA) [ADFMP20]. Following the work of Alamati et al. [ADFMP20], we model this using a Weak Pseudorandom Effective Group Action (WPR-EGA).

We start by recalling the algebraic notion of group actions. The following definition is entirely adapted from Alamati et al. [ADFMP20]

Definition A.1 (Group Action)**.** Let G be a finite abelian group and X be a finite set. We say that G acts on the set X if there is a map $* : G \times X \to X$ satisfying the following properties:

- Identity: If $e \in G$ is the identity element, then for any $x \in X$, we have $e * x = x$.
- Compatibility: For any $g, h \in G$ and any $x \in X$, we have $g * (h * x) = (g \cdot h) * x$.

We refer to the tuple $(G, X, *)$ as a *group action*. We say that $(G, X, *)$ is an *effective group action* if there exist polynomial-time algorithms to sample uniformly from G and X, to compute the group operation and inversion in G, and to evaluate the action of any group element on any set element.

Remark A.2. If $(G, X, *)$ is a group action, then for any $g \in G$, the map $\pi_G(x) := g * x$ is a permutation.

Definition A.3 (Weak Pseudorandom Effective Group Action)**.** Let EGA.Gen be a polynomial-time algorithm that, on input 1^λ, outputs public parameters $\mathsf{pp}_{\mathsf{EGA}} = (G, X, *)$ describing an effective group action.

We define the weak pseudorandom group action security experiment $\mathsf{Exp}^b_{\mathsf{EGA},\mathcal{A}}(\lambda)$ for a polynomial-time adversary $\mathcal{A}$ and a challenge bit $b \in \{0,1\}$ as follows:

1. The challenger computes $\mathsf{pp}_{\mathsf{EGA}} = (G, X, *) \leftarrow \mathsf{EGA.Gen}(1^\lambda)$, and sends $\mathsf{pp}_{\mathsf{EGA}}$ to the adversary $\mathcal{A}$.
2. The challenger samples a secret group element $g \xleftarrow{R} G$.
3. The adversary $\mathcal{A}$ can adaptively make a polynomial number of queries of the following types:
   - **Sample query:**
     (a) The challenger samples a uniformly random set element $x \xleftarrow{R} X$.
     (b) The challenger computes $y = g * x$.
     (c) The challenger sends the pair (x, y) to the adversary $\mathcal{A}$.
   - **Challenge query:**
     (a) The challenger samples a uniformly random set element $x^* \xleftarrow{R} X$.
     (b) The challenger computes the target element y^* as follows:
       – If $b = 0$, the challenger computes $y^* = g * x^*$.
       – If $b = 1$, the challenger samples a uniformly random element $y^* \xleftarrow{R} X$.
     (c) The challenger sends the pair (x^*, y^*) to the adversary $\mathcal{A}$.
4. At any point, the adversary $\mathcal{A}$ outputs a guess b', which is the output of the experiment.

We say that the Effective Group Action satisfies weak pseudorandomness if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr[\mathsf{Exp}^0_{\mathsf{EGA},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^1_{\mathsf{EGA},\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Construction A.4 (KA-PRF from Cryptographic Group Actions)**.** Let λ be a security parameter. This construction achieves perfect evaluation correctness and allows unbounded composition, therefore the algorithms ignore λ_s, N. The algorithms are defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\lambda_s}, 1^N)$: Runs $(G, X, *) \leftarrow \mathsf{EGA.Gen}(1^\lambda)$. Outputs the public parameters $\mathsf{pp} = (G, X, *)$. The key space is $\mathcal{K} = G$, the domain and codomain spaces are $\mathcal{X} = \mathcal{Y} = X$.
- $\mathsf{KeyGen}(\mathsf{pp})$: Samples and outputs $\gamma \xleftarrow{R} G$ uniformly at random.
- $\mathsf{SampleIn}(\mathsf{pp})$: Samples and outputs $x \xleftarrow{R} X$ uniformly at random.
- $\mathsf{Compose}(\mathsf{pp}, \gamma, x)$: Outputs the deterministic group action evaluation $x' = \gamma * x \in X$.
- $\mathsf{Eval}(\mathsf{pp}, x')$: Outputs $y = x'$ (the evaluation step is trivial and does nothing).
- $\mathsf{Accumulate}(\mathsf{pp}, \gamma_1, \ldots, \gamma_{N'})$: Outputs $\Gamma = \prod_{i=1}^{N'} \gamma_i \in G$.
- $\mathsf{Noisy}(\mathsf{pp}, x)$: Outputs $\tilde{x} = x$ (the noisy step is trivial and adds no noise).

We briefly verify that Construction A.4 satisfies the statistical and algebraic properties of Definition 4.1:

- **Composability and Commutativity:** Composability follows directly from the compatibility of the group action: $(\gamma_2 \cdot \gamma_1) * x = \gamma_2 * (\gamma_1 * x)$. Commutativity follows immediately from the fact that the underlying group G is abelian, so $\gamma_1 \cdot \gamma_2 = \gamma_2 \cdot \gamma_1$.
- **Bijection:** By Remark A.2, for any valid key $\gamma \in G$, the mapping $F_\gamma(x) = \gamma * x$ defines a permutation over the set X.

- **Approximate Evaluation Correctness:** Since the Noisy algorithm is deterministic and outputs its input identically ($\tilde{x} = x$), and Eval is the identity function, both sides of the equivalence evaluate to exactly $\Gamma * x$. The probability of discrepancy is 0, which trivially satisfies the required bound.
- **Evaluation Uniformity:** Since x is sampled uniformly from $\mathcal{X} = X$ and Eval outputs x directly, the output is distributed perfectly uniformly over the codomain $\mathcal{Y} = X$.

Lemma A.5. *Assume the effective group action generated by* EGA.Gen *satisfies weak pseudorandomness. Then Construction A.4 satisfies weak pseudorandomness of Definition 4.1.*

Proof. We proceed by a reduction from the weak pseudorandomness game of the Effective Group Action. Let $\mathcal{A}$ be an efficient adversary that breaks the weak pseudorandomness of the instantiated KA-PRF with advantage ε. We construct a PPT reduction $\mathcal{B}$ that uses $\mathcal{A}$ to break the EGA weak pseudorandomness assumption with the same advantage ε.

The reduction $\mathcal{B}$ operates as follows:

1. $\mathcal{B}$ receives the public parameters $\mathsf{pp} = (G, X, *)$ from its EGA challenger and forwards them to $\mathcal{A}$. The secret evaluation key γ corresponds exactly to the secret group element g implicitly chosen by the EGA challenger.
2. When $\mathcal{A}$ requests a **Sample query**, $\mathcal{B}$ queries its own EGA Sample oracle. It receives a pair (x_i, y_i), where $x_i \xleftarrow{R} X$ and $y_i = g * x_i$, and forwards this pair to $\mathcal{A}$.
3. When $\mathcal{A}$ requests a **Challenge query**, $\mathcal{B}$ queries its EGA Challenge oracle. It receives a challenge pair (x^*, y^*) and forwards it to $\mathcal{A}$.
4. Eventually, $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. $\mathcal{B}$ outputs the same guess b' as its solution to the EGA challenge.

We now analyze the view of $\mathcal{A}$ simulated by $\mathcal{B}$. For the **Sample queries**, the EGA challenger samples $x_i \xleftarrow{R} X$ and returns $y_i = g * x_i$. This perfectly matches the KA-PRF algorithm $\mathsf{SampleIn}(\mathsf{pp})$ producing x_i and the exact, deterministic evaluation of $\mathsf{Noisy}(\mathsf{pp}, \mathsf{Compose}(\mathsf{pp}, \gamma, x_i)) = \gamma * x_i = y_i$.

For the **Challenge query**, the element x^* is uniformly distributed in X. The simulation then depends on the EGA challenger's hidden bit b:

- If $b = 0$, then $y^* = g * x^* = \gamma * x^*$. This perfectly simulates the real evaluation corresponding to the $b = 0$ case in the KA-PRF game.
- If $b = 1$, then y^* is a uniformly random element in X. This perfectly simulates the random evaluation corresponding to the $b = 1$ case in the KA-PRF game.

Since $\mathcal{B}$ perfectly simulates the weak pseudorandomness game for $\mathcal{A}$ in both cases, $\mathcal{B}$'s advantage in deciding the EGA problem is exactly equal to $\mathcal{A}$'s advantage in breaking the weak pseudorandomness. Therefore, if the EGA assumption holds, the advantage ε must be negligible. □