# Adaptively Single-key Secure Constrained PRFs for $\mathbf{NC}^1$

Nuttapong Attrapadung[1], Takahiro Matsuda[1], Ryo Nishimaki[2], Shota Yamada[1], Takashi Yamakawa[2]

[1]National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan
{n.attrapadung,t-matsuda,yamada-shota}@aist.go.jp
[2]NTT Secure Platform Laboratories, Tokyo, Japan
{nishimaki.ryo,yamakawa.takashi}@lab.ntt.co.jp

October 17, 2018

**Abstract**

We present a construction of an adaptively single-key secure constrained PRF (CPRF) for $\mathbf{NC}^1$ assuming the existence of indistinguishability obfuscation (IO) and the subgroup hiding assumption over a (pairing-free) composite order group. This is the first construction of such a CPRF in the standard model without relying on a complexity leveraging argument.

To achieve this, we first introduce the notion of partitionable CPRF, which is a CPRF accommodated with partitioning techniques and combine it with shadow copy techniques often used in the dual system encryption methodology. We present a construction of partitionable CPRF for $\mathbf{NC}^1$ based on IO and the subgroup hiding assumption over a (pairing-free) group. We finally prove that an adaptively single-key secure CPRF for $\mathbf{NC}^1$ can be obtained from a partitionable CPRF for $\mathbf{NC}^1$ and IO.

# Contents

# 1 Introduction

## 1.1 Background

Constrained pseudorandom function (CPRF) [BW13][1] is a PRF with an additional functionality to "constrain" the ability of a secret key. A constrained key associated with a boolean function $f$ enables us to compute a PRF value on inputs $x$ such that $f(x) = 0$.[2] Security of CPRF roughly requires that for a "challenge input" $x^*$ such that $f(x^*) = 1$, the PRF value on $x^*$ remains pseudorandom given $\mathsf{sk}_f$. There are many applications of CPRFs including broadcast encryption [BW13], attribute-based encryption (ABE) [AMN$^+$18], identity-based non-interactive key exchange [BW13], and policy-based key distribution [BW13].

Since the proposal of the concept of CPRF, there have been significant progresses in constructing CPRFs [BW13, KPTZ13, BGI14, BZ14, BFP$^+$15, BV15, DKW16, AFP16, Bit17, GHKW17, BLW17, BKM17, CC17b, BTVW17, PS18, AMN$^+$18]. However, most known collusion-resistant[3] CPRFs (e.g. [BW13]) only satisfy a weaker security called "selective-challenge" security, where an adversary must declare a challenge input at the beginning of the security game. In the single-key setting where an adversary is given only one constrained key (e.g. [BV15]), we often consider "selective-constraint" security where an adversary must declare a constraint for which it obtains a constrained key at the beginning of the security game whereas it is allowed to later choose a challenge input.[4] In a realistic scenario, adversaries should be able to choose a constraint and a challenge input in an arbitrary order. We call such security "adaptive security".

An easy way to obtain an adaptively secure CPRF is converting selective-challenge secure one into adaptively secure one by guessing a challenge input with a standard technique typically called complexity leveraging. However, this incurs an exponential security loss, and thus we have to rely on sub-exponential assumptions. We would like to avoid this to achieve better security. Hofheinz, Kamath, Koppula, and Waters [HKKW14] constructed an adaptively secure collusion-resistant CPRF for all circuits without relying on complexity leveraging based on indistinguishability obfuscation (IO) [BGI$^+$12, GGH$^+$16] in the random oracle model. However, the random oracle model has been recognized to be problematic [CGH04].

There are a few number of adaptively secure CPRFs in the standard model. Hohenberger, Koppula, and Waters [HKW15] constructed an adaptively secure puncturable PRF based on IO and the subgroup hiding assumption on a composite order group.[5] Very recently, Davidson and Nishimaki [DN18] constructed an adaptively secure CPRF for bit-fixing functions secure against a constant number of collusion under the learning with errors (LWE) assumption, and Katsumata and Yamada [KY18] showed that such a CPRF actually can be constructed solely from one-way functions. However, these schemes only support puncturing functions or bit-fixing functions which are very limited functionalities, and there is no known construction of adaptively secure CPRF for a sufficiently expressive function class (e.g., $\mathbf{NC}^1$ or all polynomial-size circuits) even in the single-key setting and even with IO.

[1] It is also known as delegatable PRF [KPTZ13] and functional PRF [BGI14].

[2] We note that the role of the constraining function $f$ is "reversed" from the definition by Boneh and Waters [BW13], in the sense that the evaluation by a constrained key $\mathsf{sk}_f$ is possible for inputs $x$ with $f(x) = 1$ in their definition, while it is possible for inputs $x$ for $f(x) = 0$ in our paper. Our treatment is the same as Brakerski and Vaikuntanathan [BV15].

[3] A CPRF is called collusion-resistant if it remains secure even if adversaries are given polynomially many constrained keys.

[4] In previous works, both selective-challenge and selective-constraint security are simply called selective security. We use different names for them for clarity.

[5] More precisely, they also generalized their construction to obtain a CPRF for $t$-puncturing functions, which puncture the input space on $t$ points for a polynomial $t$ (rather than a single point).

## 1.2 Our Contribution

In this study, we achieve an adaptively single-key secure CPRF for $\mathbf{NC}^1$ assuming the existence of IO and the subgroup hiding assumption over a (pairing-free) composite order group. This is the first construction of such a CPRF in the standard model without relying on the complexity leveraging technique.

We emphasize that using IO is *not an easy solution* to achieve adaptive security even in the single-key setting, although IO is a strong cryptographic tool (a.k.a. "heavy hammer"). All CPRFs for a sufficiently expressive class based on IO in the standard model do not achieve adaptive security if we do not rely on complexity leveraging [BZ14, BLW17, AFP16, DKW16, DDM17].

## 1.3 Core Technique

The core technique of our construction is using the notion of *partitionable CPRF*, which we introduce in this study. At a high level, a partitionable CPRF has an additional functionality that is based on a combination of the partitioning [BB04, Wat05] and shadow copy techniques in dual system encryption [Wat09, LW10]. A partitionable CPRF enables us to generate a "merged" key in which real and "shadow" master secret keys, as well as a partitioning policy are embedded. According to a partitioning policy, the input space is partitioned into two disjoint spaces, challenge and simulation spaces. The merged key works similarly to the real master secret key on all inputs in the challenge space, and similarly to the shadow master secret key on all inputs in the simulation space. There are two security requirements for partitionable CPRF. First, we require that it satisfies selective-constraint no-evaluation security as a usual CPRF, where an adversary must declare its unique constraining query at the first of the security game and do not make any evaluation queries. Here, it is important that in this security notion, an adversary is allowed to *adaptively* choose a challenge query. Second, we require that a merged key hides the partitioning policy embedded. This property is called partition-hiding.

Our adaptively secure CPRF is actually partitionable CPRF itself except that a constrained key is obfuscated by IO. We give a proof outline of the adaptive single-key security. First, we replace a real master secret key with a merged key in which real and shadow master secret keys and a partitioning policy are embedded, so that all evaluation queries fall in the simulation space and a challenge query falls in the challenge space with noticeable probability. This modification is validated by the partition-hiding property. Then all evaluation queries can be simulated by the shadow master secret key, and the real master secret key is used only for simulating a constrained key and a challenge PRF value. Here, we consider the following two cases.[6] First, if a challenge query is made before a constraining query, then we can replace a challenge PRF value with a random string by a standard puncturing technique [SW14, BZ14] with the security of IO because the challenge input is already determined when a constraining query is made. Second, if a challenge query is made after a constraining query, then we can reduce the pseudorandomness of a challenge PRF value to the selective-constraint no-evaluation security of the partitionable CPRF. (Recall that in this security notion, an adversary is allowed to make a challenge query after its constraining query.) In both cases, we can prove that a challenge PRF value is pseudorandom. This completes the proof of the adaptive single-key security.

We construct a partitionable CPRF for $\mathbf{NC}^1$ by combining ideas from the adaptively secure puncturable PRF of Hohenberger et al. [HKW15] and the selective-constraint no-evaluation secure CPRF for $\mathbf{NC}^1$ recently proposed by Attrapadung, Matsuda, Nishimaki, Yamada, and Yamakawa [AMN+18].

## 1.4 Design Idea and Technical Overview

In this section, we give a more detailed overview of our design idea and technique.

[6]We can assume that an adversary make only one challenge query without loss of generality. (See Remark 2.8.)

**Toward adaptive security: partitioning technique.** Our construction is based on a technique called the partitioning technique, which has been widely used to achieve adaptive security in the context of signature, identity-based encryption, verifiable random function etc. [BB04, Wat05, CHKP12, Jag15, Yam17]. Roughly speaking, in the partitioning technique, a reduction algorithm partitions the input space into two disjoint spaces, the challenge space and the simulation space, so that it can compute PRF values on all inputs in the simulation space whereas it cannot compute it on any input in the challenge space. More specifically, the input space is partitioned via an *admissible hash function* denoted by $h : \{0,1\}^n \to \{0,1\}^m$ and a *partitioning policy* $u \in \{0,1,\perp\}^m$ where $\{0,1\}^n$ is the input space.[7] We partition the input space $\{0,1\}^n$ so that $x \in \{0,1\}^n$ is in the challenge space if $P_u(h(x)) = 0$ and it is in the simulation space if $P_u(h(x)) = 1$, where $P_u$ is defined by

$$P_u(y) = \begin{cases} 0 & \text{If for all } i \in [m],\ u_i = \perp \ \vee \ y_i = u_i \\ 1 & \text{Otherwise} \end{cases},$$

where $y_i$ and $u_i$ are the $i$-th bit of $y$ and $u$, respectively. If we choose $u$ according to an appropriate distribution (depending on the number of evaluation queries), the probability that all evaluation queries fall in the simulation space and a challenge query falls in the challenge space is noticeable, in which case, a reduction algorithm works well. The crucial feature of this technique is that a reduction algorithm need not know a challenge query at the beginning of its simulation.

Though it may seem easy to construct adaptively secure CPRFs based on the above idea, it is not the case because we also have to simulate constrained keys in security proofs of CPRFs. Indeed, Hofheinz et al. [HKKW14] observed that the partitioning technique does not seem to work for constructing collusion-resistant CPRFs. Nonetheless, we show that it works in the case of single-key secure CPRFs by using a *partitionable CPRF* which we introduce in this study.

**Partitionable CPRF.** Intuitively, a partitionable CPRF is a CPRF with an additional functionality that enables us to generate a "merged" key from two independent master keys and a partitioning policy $u$. The behavior of a merged key depends on whether an input is in the challenge space or in the simulation space. Namely, if we merge $\mathsf{msk}_0$ and $\mathsf{msk}_1$ with a partitioning policy $u$ to generate a merged key $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$, then it works similarly to $\mathsf{msk}_0$ for inputs $x$ in the challenge space, and $\mathsf{msk}_1$ for inputs $x$ in the simulation space. We often call $\mathsf{msk}_0$ a real master key, and $\mathsf{msk}_1$ a "shadow" master key because the former is the real master secret key used in actual constructions and the latter is an artificial key that only appears in security proofs.

For a partitionable CPRF, we require two properties. First, we require that it satisfy selective-constraint no-evaluation security as a CPRF, where an adversary must declare its unique constraining query at the beginning of the security game and does not make any evaluation queries. Here, it is important that in this security notion, an adversary is allowed to *adaptively* choose a challenge query. Second, we require a property called the partition-hiding, which means that $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ does not reveal $u$. In particular, $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$, which works exactly the same as $\mathsf{msk}_0$, is computationally indistinguishable from $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$.

**Adaptively secure CPRF from partitionable CPRF.** Now, we take a closer look at how we construct an adaptively single-key secure CPRF based on a partitionable CPRF and IO. Actually, master secret keys and PRF values of the CPRF is defined to be exactly the same as those of the underlying partitionable CPRF. The

[7]Actually, we use an extended notion called a balanced admissible hash function. (See Section 2.2.)

only difference between them is the way of generating constrained keys. In the proposed CPRF, a constrained key for a function $f$ is an obfuscated program that computes PRF values on all inputs $x$ such that $f(x) = 0$ with a real master secret key.

The security proof proceeds as follows. First, we remark that if a challenge query is made before the constraining query, then the proof is easy by the standard puncturing technique [SW14, BZ14]. Thus, in the following, we assume that a challenge query is made after the constraining query. First, we modify the security game so that we use $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ instead of $\mathsf{msk}_0$ where $\mathsf{msk}_1$ is a "shadow" master secret key that is independent from $\mathsf{msk}_0$. This modification causes a negligible difference by the security of IO because $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ works exactly the same as $\mathsf{msk}_0$. Then we replace $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ with $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ for a partitioning policy $u$ chosen from an appropriate distribution. This modification causes a negligible difference by the partition-hiding of the underlying partitionable CPRF. Here, suppose that all evaluation queries are in the simulation space, and the challenge query $x^*$ is in the challenge space. Such an event occurs with noticeable probability by the way we choose $u$. In this case, all evaluation queries can be simulated by using the shadow master secret key $\mathsf{msk}_1$ whereas a challenge value is computed by using the real secret key $\mathsf{msk}_0$. Then we modify a constrained key $\mathsf{sk}_f$ associated with a function $f$ so that we hardwire $\mathsf{sk}_f^{\mathsf{real}}$, which is a constrained key associated with the function $f$ derived from $\mathsf{msk}_0$ by the constraining algorithm of the underlying partitionable CPRF, instead of $\mathsf{msk}_0$. This modification causes a negligible difference by the security of IO since $\mathsf{sk}_f^{\mathsf{real}}$ and $\mathsf{msk}_0$ works similarly on inputs $x$ such that $f(x) = 0$. At this point, a PRF value on $x^*$ such that $f(x^*) = 1$ is pseudorandom by the selective-constraint no-evaluation security of the underlying partitionable CPRF (Recall that $\mathsf{msk}_0$ is not used for simulating the evaluation oracle now). This completes the proof of the adaptive single-key security of the CPRF.

**Partitionable CPRF for puncturing [HKW15].** What is left is a construction of a partitionable CPRF. First, we observe that the construction of adaptively secure puncturable PRF by Hohenberger et al. [HKW15] can be seen as a construction of a partitionable CPRF for puncturing functions. Their construction is a variant of the Naor-Reingold PRF [NR04] on a composite order group $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q$ of an order $N = pq$. Namely, a master secret key $\mathsf{msk}^{\mathsf{hkw}}$ consists of $s_{i,b} \in \mathbb{Z}_N$ for $i \in [m]$ and $b \in \{0, 1\}$, and their PRF $F_{\mathsf{hkw}}$ is defined as

$$F_{\mathsf{hkw}}(\mathsf{msk}^{\mathsf{hkw}}, x) := g^{\prod_{i=1}^{m} s_{i,y_i}}.$$

Here, $g$ is a generator of $\mathbb{G}$ and $y_i$ is the $i$-th bit of $y := h(x)$, where $h$ is an admissible hash function. A punctured key on the challenge input $x^*$ is an obfuscated program that computes $F_{\mathsf{hkw}}(\mathsf{msk}, x)$ on all inputs $x \neq x^*$. They implicitly proved that the above construction is a partitionable CPRF for puncturing if we define $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ to be an obfuscation of a program that computes $F_{\mathsf{hkw}}(\mathsf{msk}_{P_u(x)}, x)$ on an input $x$.

We remark that we cannot directly reduce the partition-hiding property to the security of IO because the functionality of $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ and $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ differ on exponentially many inputs. They overcome this problem by a sophisticated use of the subgroup hiding assumption on a composite order group. Namely, we can prove that this construction satisfies the partition-hiding under the security of IO and the subgroup hiding assumption, which claims that random elements of $\mathbb{G}_p$ and $\mathbb{G}$ are computationally indistinguishable. Then if we can prove the above construction is a selective-constraint no-evaluation secure CPRF for a function class $\mathcal{F}$, then we obtain an adaptively single-key secure CPRF for the function class $\mathcal{F}$ as discussed in the previous paragraph. One may think that it is easy to prove that the above construction is selective-constraint no-evaluation secure for all circuits by using standard puncturing technique with IO [SW14, BZ14]. However, it is not the case because the selective-constraint security requires security against an adversary that makes a challenge query after making a constraining query. Though IO is quite powerful when considering selective-challenge security where an adversary declares a challenge query at the beginning, it is almost useless for selective-constraint security where an adversary may adaptively choose a challenge query. For the case of puncturable PRF, a challenge input is automatically determined when a constraining query is made, and thus selective-constraint security is equivalent to selective-challenge security. This is why they achieved adaptive security only for a puncturable PRF.

**Partitionable CPRF for $\mathbf{NC}^1$.** Finally, we explain how to construct a partitionable CPRF for $\mathbf{NC}^1$. Our idea is to combine Hohenberger et al.'s construction as described above and the selective-constraint no-evaluation secure CPRF for $\mathbf{NC}^1$ recently proposed by Attrapadung et al. [AMN+18]. The construction of Attrapadung et al.'s CPRF $F_{\mathsf{amnyy}}$ (instantiated on a composite order group $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q$) is described as follows.

$$F_{\mathsf{amnyy}}(\mathsf{msk}^{\mathsf{amnyy}}, x) = g^{U(\vec{b},x)/\alpha}$$

where $\mathsf{msk}^{\mathsf{amnyy}} = (\vec{b} \in \mathbb{Z}_N^z, \alpha \in \mathbb{Z}_N)$ is a master secret key and $U(\cdot)$ is a polynomial that works as a universal circuit for $\mathbf{NC}^1$. We omit a description of constrained keys for this CPRF since this is not important in this overview (See Section 3.2 for details). They proved that $F_{\mathsf{amnyy}}$ satisfies selective-constraint no-evaluation security under the $L$-DDHI assumption[8], which can be reduced to the subgroup hiding assumption (See Lemma 2.3). An important fact is that their CPRF is secure against adversaries that adaptively make a challenge query as long as a constraining query is declared at the beginning and they do not make any evaluation queries.

Then we combine $F_{\mathsf{amnyy}}$ and $F_{\mathsf{hkw}}$ to define $F_{\mathsf{ours}}$ as follows:

$$F_{\mathsf{ours}}(\mathsf{msk}^{\mathsf{ours}}, x) = g^{(\prod_{i=1}^m s_{i,y_i})\cdot U(\vec{b},x)/\alpha},$$

where $x$ is an input, $y_i$ is the $i$-th bit of $h(x)$, $h$ is an admissible hash function, and $\mathsf{msk}^{\mathsf{ours}} = (\vec{b}, \alpha, \{s_{i,b}\}_{i\in[m],b\in\{0,1\}})$ is a master secret key. A constrained key for a predicate $f$ consists of that of $F_{\mathsf{amnyy}}$ and $\{s_{i,b}\}_{i\in[m],b\in\{0,1\}}$. It is easy to see that this constrained key can be used to evaluate $F_{\mathsf{ours}}(\mathsf{msk}^{\mathsf{ours}}, x)$ for all $x$ such that $f(x) = 0$ since we have

$$F_{\mathsf{ours}}(\mathsf{msk}^{\mathsf{ours}}, x) = F_{\mathsf{amnyy}}(\mathsf{msk}^{\mathsf{amnyy}}, x)^{\prod_{i=1}^m s_{i,y_i}}$$

where $\mathsf{msk}^{\mathsf{amnyy}} := (\vec{b}, \alpha)$. By this equation, it is also easy to see that the selective-constraint no-evaluation security of $F_{\mathsf{ours}}$ can be reduced to that of $F_{\mathsf{amnyy}}$. A merged key is an obfuscated circuit that computes $\mathsf{Eval}(\mathsf{msk}_{P_u(h(x))}, x)$ where $\mathsf{msk}_0 = (\vec{b}, \alpha, \{s_{i,b}\}_{i\in[m],b\in\{0,1\}})$ and $\mathsf{msk}_1 = (\hat{\vec{b}}, \hat{\alpha}, \{\hat{s}_{i,b}\}_{i\in[m],b\in\{0,1\}})$ are two independent master secret keys and $u$ is a partitioning policy embedded into the merged key.

Now, we look at why the construction satisfies partition-hiding. Intuitively, a partitioning policy $u$ is hidden because it is hardwired in an obfuscated circuit. However, since the functionality of $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ and $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ differ on exponentially many inputs, we cannot directly argue indistinguishability of them based on the security of IO. In the following, we explain how to prove it relying on the subgroup hiding assumption. Roughly speaking, this consists of two parts. In the first part, we modify the way of computing PRF values inside a merged key (which is an obfuscated program) so that it uses a different way to compute them on inputs in the challenge space and on those in the simulation space. In the second step, we make a shadow copy of the real master key by using the Chinese remainder theorem.

[8] It assumes that $\{(\mathcal{G}, g, (g^{\beta^i})_{i\in[L]}, g^{1/\beta})\} \approx_{\mathsf{c}} \{(\mathcal{G}, g, (g^{\beta^i})_{i\in[L]}, \psi_1)\}$ holds, where $\mathcal{G} = (N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2)$, $\mathbb{G}$, $\mathbb{G}_p$, and $\mathbb{G}_q$ are groups of order $N$, $p$, and $q$, respectively, $g$, $g_1$, and $g_2$ are generators of $\mathbb{G}$, $\mathbb{G}_p$, and $\mathbb{G}_q$, respectively, and $\psi_1 \stackrel{\mathsf{R}}{\leftarrow} \mathbb{G}$.

First, to modify the way of computing PRF values inside a merged key, we use the $(m-1)$-DDH assumption, which claims that we have $\{(\mathcal{G}, g, (g^{\beta^i})_{i\in[m-1]}, g^{\beta^m})\} \approx_{\mathsf{c}} \{(\mathcal{G}, g, (g^{\beta^i})_{i\in[m-1]}, \psi_1)\}$, where $\mathcal{G} = (N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2)$, $\mathbb{G}$, $\mathbb{G}_p$, and $\mathbb{G}_q$ are groups of order $N$, $p$, and $q$, respectively, $g$, $g_1$, and $g_2$ are generators of $\mathbb{G}$, $\mathbb{G}_p$, and $\mathbb{G}_q$, respectively, and $\psi_1 \xleftarrow{\mathsf{R}} \mathbb{G}$. As shown in Lemma 2.3, this assumption can be reduced to the subgroup hiding assumption. Recall that the partitioning policy $P_u(y)$ outputs $0$ (i.e., $x$ is in the challenge space) if for all $i$, $u_i = y_i \vee u_i = \perp$. Here, we set $s_{i,\eta} := \beta s'_{i,\eta} \in \mathbb{Z}_N$ for all $(i, \eta)$ such that $u_i = \perp$ or $\eta = u_i$, where $s'_{i,\eta}$ is a uniformly random and $\beta$ comes from the $(m-1)$-DDH instance. The distributions of $s_{i,\eta}$ set as above are statistically close to the original ones. Now, a merged key uses the $(m-1)$-DDH challenge $w \in \mathbb{G}$ (which is $g^{\beta^m}$ or random) for simulating a PRF value on an input $x$ in the challenge space. That is, it computes the PRF value on $x$ as $w^{(\prod_{i=1}^{m} s'_{i,y_i})\cdot U(\vec{b},x)/\alpha}$. On the other hand, on inputs $x$ in the simulation space, it uses the values $(g, g^{\beta}, \ldots, g^{\beta^{m-1}})$ in the $(m-1)$-DDH problem instances as $(g^{\beta^r})^{(\prod_{i=1}^{m} s'_{i,y_i})\cdot U(\vec{b},x)/\alpha}$, where $r := |\{i \in [m] \mid u_i = y_i\}| \leq m-1$. If $w = g^{\beta^m}$, then a merged key as modified above correctly computes PRF values on all inputs. Thus, this modification causes a negligible difference by the security of IO. Then we can replace $w$ with a random element in $\mathbb{G}$ by using the $(m-1)$-DDH assumption.

Now, we use the subgroup hiding assumption to make a shadow copy of the real master key. By the subgroup hiding assumption, we can replace $w \in \mathbb{G}$ and $g \in \mathbb{G}$ with $w \in \mathbb{G}_p$ and $g \in \mathbb{G}_q$, respectively, where $\mathbb{G}$ (resp. $\mathbb{G}_p$, $\mathbb{G}_q$) is a group of order $N = pq$ (resp. $p$, $q$) and $p, q$ are primes.[9]. Then, we can set $\mathsf{msk}_0 := \{s'_{i,b} \bmod p\}_{i,b}$ and $\mathsf{msk}_1 := \{s'_{i,b} \bmod q\}_{i,b}$. Since $w \in \mathbb{G}_p$ and $g^{\beta^j} \in \mathbb{G}_q$ where $j \in \{1, \ldots, m-1\}$, it holds that

$$w^{(\prod s'_{i,y_i})\cdot U(\vec{b},x)/\alpha} = w^{((\prod s'_{i,y_i})\cdot U(\vec{b},x)/\alpha \bmod p)}$$
$$(g^{\beta^j})^{(\prod s'_{i,y_i})\cdot U(\vec{b},x)/\alpha} = (g^{\beta^j})^{((\prod s'_{i,y_i})\cdot U(\vec{b},x)/\alpha \bmod q)}$$

and this change is indistinguishable due to the security of IO. Lastly, by the Chinese remainder theorem, $\mathsf{msk}_0$ and $\mathsf{msk}_1$ are independently and uniformly random (that is, $\mathsf{msk}_1$ can be changed into $\{\hat{s}_{i,b} \bmod q\}_{i,b}$ where $\hat{s}_{i,b}$ are independent of $s'_{i,b}$ and uniformly random). Now, the shadow master secret key is used for evaluating PRF values on inputs in the challenge space whereas the real master secret key is used for evaluating those on inputs in the simulation space as desired.

By these techniques, we can obtain a partitionable CPRF for $\mathbf{NC}^1$ based on IO and the subgroup hiding assumption in pairing-free groups though we omit many details for simplicity in this overview.

In summary, we can obtain an adaptively single-key secure CPRF for $\mathbf{NC}^1$ by combining the above partitionable CPRF for $\mathbf{NC}^1$ based on IO and the subgroup hiding assumption with the transformation from a partitionable CPRF into an adaptively secure CPRF explained in the paragraph of "Adaptively secure CPRF from partitionable CPRF".

## 1.5 Discussion

**Why subgroup-hiding needed?** One may wonder why we need the subgroup hiding assumption as an extra assumption though we rely on IO, which is already a significantly strong assumption. We give two reasons for this below. The first reason is that we do not know how to construct a CPRF with *selective-constraint* security (even in the single-key setting) from IO though we can construct collusion-resistant CRPF with *selective-challenge* security from IO [BZ14]. In the CPRF based on IO, a constrained key is an obfuscated

[9] Note that being given both $g_1 \in \mathbb{G}_p$ and $g_2 \in \mathbb{G}_q$ does not lead to a trivial attack since we use "pairing-free" groups.

program that evaluates the PRF on inputs that satisfy the constraint. In the security proof, we puncture the obfuscated program on the challenge input by using the security of IO. This argument is crucially based on the fact that the challenge is given before all constraining queries, and cannot be used in the selective-constraint setting where the challenge is chosen after a constrained key is given. Since our security definition of partitionable CPRF requires selective-constraint security, it seems difficult to construct it from IO. We note that selective-constraint security (rather than selective-challenge security) of partitionable CPRF is crucial to prove the adaptive security of our final CPRF. The second reason is more specific to the security proof of our partitionable CPRF. Namely, in the proof of the partition-hiding property of our partitionable CPRF, we have to modify outputs of an obfuscated circuit (which is a constrained key) on exponentially many inputs. Since the security of IO only enables us to modify an obfuscated circuit only on one input, it would need exponential number of hybrids to modify outputs on exponentially many inputs if we just use the security of IO. We overcome this issue by a sophisticated use of the subgroup hiding assumption in a similar way to the work by Hohenberger et al. [HKW15]. We note that in this technique, the Chinese remainder theorem is essential, and we cannot replace the assumption with the decisional linear (DLIN) assumption on a prime-order group, though there are some known prime-to-composite-order conversions in some settings [Fre10, SC12, Lew12, HHH$^+$14].

**Why single-key security for $\mathbf{NC}^1$?** One may wonder why our adaptive CPRF only achieves single-key security rather than collusion-resistance and supports $\mathbf{NC}^1$ rather than all polynomial-size circuits (**P/poly**) though there seems to be no obvious attack against our CPRF even if an adversary is given multiple constrained keys for constraints possibly outside $\mathbf{NC}^1$.[10] Actually, we can prove that our CPRF is collusion-resistant and supports **P/poly** in the selective-challenge setting by the puncturing technique similarly to [BZ14]. However, in the security of adaptive security, we crucially rely on the selective-constraint security of the underlying partitionable CPRF, which stems from the CPRF by Attrapadung et al. [AMN$^+$18]. Since their CPRF only achieves single-key security and supports $\mathbf{NC}^1$, our CPRF inherits them. Possible alternatives to their CPRF are lattice-based CPRFs [BV15, BTVW17, PS18] which satisfy selective-constraint single-key security and supports **P/poly**. If we could use these CPRFs instead of Attrapadung et al.'s scheme, we would obtain adaptively single-key secure CPRFs for **P/poly**. However, since we use techniques based on the subgroup-hiding assumption in the proof of the partition-hiding property of our partitionable CPRF, we have to rely on group-based CPRFs for compatibility to the technique, and this is the reason why we cannot use lattice-based CPRFs.

**Relation with private CPRF.** Partitionable CPRF and private CPRF [BLW17] share a similarity that both enable one to modify functionality of a PRF key without revealing inputs on which outputs were manipulated. Actually, a partitionable CPRF can be seen as a private CPRF for the "admissible hash friendly" functionality [GHKW17]. On the other hand, the inverse is not true. Private CPRF does not put any restriction on behaviors of a constrained key on inputs that do not satisfy the constraint except that they look random. On the other hand, partitionable CPRF requires behaviors on these inputs should be consistent in the sense that they are PRF values evaluated on another master secret key. This difference makes it more difficult to construct a partitionable CPRF than constructing a private CPRF.

[10]Though our construction of CPRF in Section 4 only supports $\mathbf{NC}^1$, we can naturally generalize it to support **P/poly** if we do not care about the security.

### 1.6 Other Related Work

Here, we discuss two additional related works that are relevant to adaptively secure CPRFs.

Fuchsbauer, Konstantinov, Pietrazk, and Rao [FKPR14] proved that the classical GGM PRF [GGM86] is an adaptively secure puncturable PRF if the underlying PRG is quasi-polynomially secure. We note that quasi-polynomially-secure PRG is a super-polynomial hardness assumption.

Canetti and Chen [CC17a] proposed a lattice-based construction of (constraint-hiding) single-key secure CPRF for $\mathbf{NC}^1$ that achieves a weaker form of adaptive security where adversaries are allowed to send *logarithmically* many evaluation queries before a constraining query as long as it correctly declares if the evaluation query satisfies the constraint to be queried as a constraining query. We note that in the proceedings version [CC17b], they claimed security against adversaries that make an unbounded number of evaluation queries before a constraining query, but they retracted the claim [CC17a, footnotes 1 and 2]. We remark that the adaptive security defined in this paper does not put any restriction on the number of evaluation queries before a constraining query nor require adversaries to declare if the evaluation query satisfies the constraint to be queried as a constraining query.

**Organization.** The rest of the paper is organized as follows. After introducing notations, security definitions, and building blocks in Section 2, we present the definition of partitionable CPRF, our construction of partitionable CPRF for $\mathbf{NC}^1$, and its security proofs in Section 3, and our adaptively single-key secure CPRFs for $\mathbf{NC}^1$ and its security proofs in Section 4.

## 2 Preliminaries

In this section, we review the basic notation and the definitions for complexity assumptions, tools, and cryptographic primitives.

**Basic notation.** We denote by $\mathbb{N}$ the set of all natural numbers. If $n \in \mathbb{N}$, then "$[n]$" denotes the set $\{1, \ldots, n\}$. We denote by "$x := y$" that $y$ is deterministically assigned to $x$. If $S$ is a finite set, then "$x \xleftarrow{\mathsf{R}} S$" denotes that $x$ is chosen uniformly at random from $S$. If $\mathcal{D}$ and $\mathcal{D}'$ are distributions (over some set), then "$x \xleftarrow{\mathsf{R}} \mathcal{D}$" denotes that $x$ is chosen according to the distribution $\mathcal{D}$, and "$\mathcal{D} \approx_{\mathsf{c}} \mathcal{D}'$" denotes that the two distributions are computationally indistinguishable. If $x$ and $y$ are bit-strings, then we denote by "$x \| y$" the concatenation of $x$ and $y$, and "$(x \stackrel{?}{=} y)$" is defined to be $1$ if $x = y$ and $0$ otherwise. "PPT" stands for *probabilistic polynomial time*. If $\mathcal{A}$ is a probabilistic algorithm, then "$y \xleftarrow{\mathsf{R}} \mathcal{A}(x)$" denotes that $\mathcal{A}$ computes and outputs $y$ by taking $x$ as input and using an internal randomness that is chosen uniformly at random. If furthermore $\mathcal{O}$ is a (possibly probabilistic) function, then "$\mathcal{A}^{\mathcal{O}}$" denotes that $\mathcal{A}$ has oracle access to $\mathcal{O}$. A function $f(\cdot) : \mathbb{N} \to [0, 1]$ is said to be *negligible* if for all polynomials $p(\cdot)$ and all sufficiently large $\lambda \in \mathbb{N}$, we have $f(\lambda) < 1/p(\lambda)$. The function $f$ is *noticeable* when there exists a polynomial $p(\cdot)$ such that we have $f(\lambda) \geq |1/p(\lambda)|$ for all sufficiently large $\lambda$. Throughout the paper, we use "$\lambda$" to denote a security parameter (which is given to algorithms always in the unary form $1^\lambda$). We denote by "$\mathrm{poly}(\cdot)$" an unspecified integer-valued positive polynomial of $\lambda$ and by "$\mathrm{negl}(\lambda)$" an unspecified negligible function of $\lambda$. For sets $\mathcal{D}$ and $\mathcal{R}$, "$\mathsf{Func}(\mathcal{D}, \mathcal{R})$" denotes the set of all functions with domain $\mathcal{D}$ and range $\mathcal{R}$.

### 2.1 Composite Order Group

In this paper, in a similar manner to Hohenberger et al. [HKW15], we will use a group of composite order in which the subgroup hiding assumption holds. We recall it here.

Let $\mathsf{GGen}$ be a PPT algorithm (called the *group generator*) that takes a security parameter $1^\lambda$ as input, and outputs $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2)$, where $p, q \in \Omega(2^\lambda)$, $N = pq$, $\mathbb{G}$ is a cyclic group of order $N$, $\mathbb{G}_p$ and $\mathbb{G}_q$ are the subgroups of $\mathbb{G}$ of orders $p$ and $q$ respectively, and $g_1$ and $g_2$ are generators of $\mathbb{G}_p$ and $\mathbb{G}_q$ respectively. The subgroup hiding assumption with respect to $\mathsf{GGen}$ is defined as follows:

**Definition 2.1 (Subgroup Hiding Assumption).** *Let* $\mathsf{GGen}$ *be a group generator. We say that the* subgroup hiding *assumption holds with respect to* $\mathsf{GGen}$*, if for all PPT adversaries* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{sgh}}_{\mathsf{GGen},\mathcal{A}}(\lambda)$ *defined below is negligible:*

$$\mathsf{Adv}^{\mathsf{sgh}}_{\mathsf{GGen},\mathcal{A}}(\lambda) := \Big|\Pr[\mathcal{A}(\mathcal{G}, \psi_0) = 1] - \Pr[\mathcal{A}(\mathcal{G}, \psi_1) = 1]\Big|,$$

*where* $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{\mathsf{R}} \mathsf{GGen}(1^\lambda)$*,* $\mathcal{G} := (N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2)$*,* $\psi_0 \xleftarrow{\mathsf{R}} \mathbb{G}$*, and* $\psi_1 \xleftarrow{\mathsf{R}} \mathbb{G}_p$*.*

For our purpose in this paper, it is convenient to introduce the following *$L$-DDH*[11] and *$L$-DDHI* assumptions with respect to $\mathsf{GGen}$. These are not additional assumptions since they are implied by the subgroup hiding assumption. For completeness, we give a proof sketch of the implications (using a reduction shown by Hohenberger et al. [HKW14]).

**Definition 2.2 ($L$-DDH & $L$-DDHI Assumptions).** *Let* $\mathsf{GGen}$ *be a group generator and* $L = L(\lambda) = \mathrm{poly}(\lambda)$*. We say that the* $L$-decisional Diffie-Hellman ($L$-DDH) *assumption holds with respect to* $\mathsf{GGen}$*, if for all PPT adversaries* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{L\text{-}\mathsf{ddh}}_{\mathsf{GGen},\mathcal{A}}(\lambda)$ *defined below is negligible:*

$$\mathsf{Adv}^{L\text{-}\mathsf{ddh}}_{\mathsf{GGen},\mathcal{A}}(\lambda) := \Big|\Pr[\mathcal{A}(\mathcal{G}, g, (g^{\alpha^i})_{i\in[L]}, \psi_0) = 1] - \Pr[\mathcal{A}(\mathcal{G}, g, (g^{\alpha^i})_{i\in[L]}, \psi_1) = 1]\Big|,$$

*where* $(N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{\mathsf{R}} \mathsf{GGen}(1^\lambda)$*,* $\mathcal{G} := (N, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2)$*,* $g \xleftarrow{\mathsf{R}} \mathbb{G}$*,* $\alpha \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$*,* $\psi_0 := g^{\alpha^{L+1}}$*, and* $\psi_1 \xleftarrow{\mathsf{R}} \mathbb{G}$*.*

*The* $L$-decisional Diffie-Hellman inversion ($L$-DDHI) *assumption with respect to* $\mathsf{GGen}$ *is defined in the same way as the above, except that "*$\psi_0 := g^{\alpha^{L+1}}$*" is replaced with "*$\psi_0 := g^{1/\alpha}$*".*

**Lemma 2.3.** *Let* $\mathsf{GGen}$ *be a group generator. If the subgroup hiding assumption holds with respect to* $\mathsf{GGen}$*, then the $L$-DDH and $L$-DDHI assumptions hold with respect to* $\mathsf{GGen}$ *for all polynomials* $L = L(\lambda)$*.*

*Proof Sketch of Lemma 2.3.* Let $\mathsf{GGen}$ be a group generator. Firstly, it was shown by Hohenberger et al. [HKW14, Theorem B.1] that the subgroup hiding assumption implies the $L$-DDH assumption for all polynomials $L = L(\lambda)$.

Secondly, it is straightforward to see that for all $L = L(\lambda) = \mathrm{poly}(\lambda)$, the $L$-DDH assumption and the $L$-DDHI assumption are equivalent. Specifically, given an instance of the $L$-DDHI assumption $(\mathcal{G}, g, (g^{\alpha^i})_{i\in[L]}, \psi)$ where $\psi$ is either $g^{1/\alpha}$ or a random element in $\mathbb{G}$, let $h := g^{\alpha^L}$ and $\beta := 1/\alpha$. Then, we have the correspondence $h^{\beta^i} = g^{\alpha^{L-i}}$ for every $i \in [L+1]$. Thus, $(\mathcal{G}, g, (g^{\alpha^i})_{i\in[L]}, \psi)$ can be rearranged as $(\mathcal{G}, h, (h^{\beta^i})_{i\in[L]}, \psi)$ with an appropriate reordering, which is distributed identically to an instance of the $L$-DDH assumption. Note that $\psi$ equals to $g^{1/\alpha}$ or a random group element, and we have $\psi = h^{\beta^{L+1}}$ if the

[11]The $L$-DDH assumption was called *Assumption 2* by Hohenberger et al. [HKW14].

former is the case. Hence, an adversary that can break the $L$-DDHI assumption can be directly used as an adversary that can break the $L$-DDH assumption. The opposite implication can be established in a similar way.

Hence, combining the above two arguments, we can conclude that the subgroup hiding assumption implies the $L$-DDH and $L$-DDHI assumptions for all $L = L(\lambda) = \mathrm{poly}(\lambda)$. ■

## 2.2 Balanced Admissible Hash Functions and Related Facts

Here, we describe the definition of a balanced admissible hash function (AHF) introduced by Jager [Jag15]. A balanced AHF is an extension of an ordinary AHF [BB04, CHKP12], but with some more properties. Similarly to an ordinary AHF, it partitions the input space in a security proof so that the simulation is possible with a noticeable probability. The reason why we use a balanced AHF instead of an ordinary AHF is that the former simplifies our security proof. We note that the following formalization of a balanced AHF is slightly different from that by Jager [Jag15] and corresponds to a special case of the general notion of "a partitioning function" introduced by Yamada [Yam17].

**Definition 2.4 ([Jag15, Yam17]).** *Let $n(\lambda)$ and $m(\lambda)$ be polynomials. Furthermore, for $u \in \{0,1,\perp\}^m$, let $P_u : \{0,1\}^m \to \{0,1\}$ be defined as*

$$P_u(y) = \begin{cases} 0 & \text{If for all } i \in [m],\ u_i = \perp \ \vee\ y_i = u_i \\ 1 & \text{Otherwise} \end{cases},$$

*where $y_i$ and $u_i$ are the $i$-th bit of $y$ and $u$, respectively. We say that an efficiently computable function $h : \{0,1\}^n \to \{0,1\}^m$ is a balanced admissible hash function (balanced AHF), if there exists an efficient algorithm $\mathsf{AdmSample}(1^\lambda, Q, \delta)$, which takes as input $(Q, \delta)$ where $Q = Q(\lambda) \in \mathbb{N}$ is polynomially bounded and $\delta = \delta(\lambda) \in (0,1]$ is noticeable, and outputs $u \in \{0,1,\perp\}^m$ such that:*

1. *There exists $\lambda_0 \in \mathbb{N}$ such that*

$$\Pr\left[\, u \xleftarrow{\mathsf{R}} \mathsf{AdmSample}(1^\lambda, Q(\lambda), \delta(\lambda)) \ :\ u \in \{0,1\}^m \,\right] = 1$$

*for all $\lambda > \lambda_0$. Here, $\lambda_0$ may depend on functions $Q(\lambda)$ and $\delta(\lambda)$.*

2. *For $\lambda > \lambda_0$ (defined in Item 1), there exist $\gamma_{\max}(\lambda)$ and $\gamma_{\min}(\lambda)$ that depend on $Q(\lambda)$ and $\delta(\lambda)$ such that for all $x_1, ..., x_Q, x^* \in \{0,1\}^n$ with $x^* \notin \{x_1, ..., x_Q\}$,*

$$\gamma_{\max}(\lambda) \ge \Pr\left[P_u(h(x_1)) = ... = P_u(h(x_Q)) = 1 \wedge P_u(h(x^*)) = 0\right] \ge \gamma_{\min}(\lambda)$$

*where $\gamma_{\max}(\lambda)$ and $\gamma_{\min}(\lambda)$ satisfy that the function $\tau(\lambda)$ defined as*

$$\tau(\lambda) = \gamma_{\min}(\lambda) \cdot \delta(\lambda) - \frac{\gamma_{\max}(\lambda) - \gamma_{\min}(\lambda)}{2}$$

*is noticeable. We note that the probability is taken over the choice of $u$ where $u \xleftarrow{\mathsf{R}} \mathsf{AdmSample}(1^\lambda, Q(\lambda), \delta(\lambda))$.*

*Remark* 2.5. The term $\tau(\lambda)$ defined above may appear very specific. However, as discussed by Jager [Jag15], such a term appears typically in security analyses that follow the approach of Bellare and Ristenpart [BR09]. See also Lemma 2.6 below.

As shown by Jager [Jag15], who extended previous works that gave simple constructions of AHF [Lys02, FHPS13], a family of codes $h : \{0,1\}^n \to \{0,1\}^m$ with minimal distance $mc$ for a constant $c$ is a balanced AHF. Explicit constructions of such codes are known [SS96, Zém01, Gol08]. The following lemma is adapted from Lemma 8 in the paper by Katsumata et al. [KY16] (see also Lemma 28 in the full version of the paper by Agrawal et al. [ABB10]), and is implicit in many previous works [BR09, Jag15, Yam16]. The lemma encapsulates the change of the advantage of an adversary when there is an abort in the security proof. Note that although the lemma is shown only in the specific case of IBE in the paper by Katsumata et al. [KY16], the same proof works in the following slightly generalized setting as well.

**Lemma 2.6.** *Let us consider a random variable* $\mathsf{coin} \xleftarrow{R} \{0,1\}$ *and a random distribution* $\mathcal{D}$ *that takes as input a bit* $b \in \{0,1\}$ *and outputs* $(X, \widehat{\mathsf{coin}})$ *such that* $X \in \mathcal{X}$ *and* $\widehat{\mathsf{coin}} \in \{0,1\}$*, where* $\mathcal{X}$ *is some domain. For* $\mathcal{D}$*, we define* $\epsilon_0$ *as*

$$\epsilon_0 := \left| \Pr\left[\, \mathsf{coin} \xleftarrow{R} \{0,1\}, (X, \widehat{\mathsf{coin}}) \xleftarrow{R} \mathcal{D}(\mathsf{coin}) \ : \ \widehat{\mathsf{coin}} = \mathsf{coin} \,\right] - \frac{1}{2} \right|$$

*Let us define a map* $\gamma$ *that maps an element in* $\mathcal{X}$ *to a real value in* $[0,1]$*. We then further consider the following modified distribution* $\mathcal{D}'$ *that takes as input a bit* $b \in \{0,1\}$ *and outputs* $(X, \widehat{\mathsf{coin}})$*. To sample from* $\mathcal{D}'(b)$*, we first sample* $(X, \widehat{\mathsf{coin}}) \xleftarrow{R} \mathcal{D}(b)$*. Then, with probability* $1 - \gamma(X)$*, we re-sample* $\widehat{\mathsf{coin}}$ *as* $\widehat{\mathsf{coin}} \xleftarrow{R} \{0,1\}$*. With probability* $\gamma(X)$*, the value of* $\widehat{\mathsf{coin}}$ *is unchanged. The final output of* $\mathcal{D}'(b)$ *is* $(X, \widehat{\mathsf{coin}})$*. Then, the following holds.*

$$\left| \Pr\left[\, \mathsf{coin} \xleftarrow{R} \{0,1\}, (X, \widehat{\mathsf{coin}}) \xleftarrow{R} \mathcal{D}'(\mathsf{coin}) \ : \ \widehat{\mathsf{coin}} = \mathsf{coin} \,\right] - \frac{1}{2} \right| \geq \gamma_{\min} \cdot \epsilon_0 - \frac{\gamma_{\max} - \gamma_{\min}}{2}$$

*where* $\gamma_{\min}$ *and* $\gamma_{\max}$ *are the maximum and the minimum of* $\gamma(X)$ *taken over all possible* $X \in \mathcal{X}$*, respectively.*

## 2.3 Constrained Pseudorandom Functions

Here, we recall the syntax and security definitions for a CPRF. We use the same definitions as Attrapadung et al. [AMN+18].

**Syntax.** Let $\mathcal{F} = \{\mathcal{F}_{\lambda,k}\}_{\lambda,k\in\mathbb{N}}$ be a class of functions[12] where each $\mathcal{F}_{\lambda,k}$ is a set of functions with domain $\{0,1\}^k$ and range $\{0,1\}$, and the description size (when represented by a circuit) of every function in $\mathcal{F}_{\lambda,k}$ is bounded by $\mathrm{poly}(\lambda, k)$.

A CPRF for $\mathcal{F}$ consists of the five PPT algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ with the following interfaces:

$\mathsf{Setup}(1^\lambda) \xrightarrow{R} \mathsf{pp}$**:** This is the setup algorithm that takes a security parameter $1^\lambda$ as input, and outputs a public parameter $\mathsf{pp}$,[13] where $\mathsf{pp}$ specifies the descriptions of the key space $\mathcal{K}$, the input-length $n = n(\lambda) = \mathrm{poly}(\lambda)$ (that defines the domain $\{0,1\}^n$), and the range $\mathcal{R}$.

$\mathsf{KeyGen}(\mathsf{pp}) \xrightarrow{R} \mathsf{msk}$**:** This is the key generation algorithm that takes a public parameter $\mathsf{pp}$ as input, and outputs a master secret key $\mathsf{msk} \in \mathcal{K}$.

[12] In this paper, a "class of functions" is a set of "sets of functions". Each $\mathcal{F}_{\lambda,k}$ in $\mathcal{F}$ considered for a CPRF is a set of functions parameterized by a security parameter $\lambda$ and an input-length $k$.

[13] For clarity, we will define a CPRF as a primitive that has a public parameter. However, this treatment is compatible with the standard syntax in which there is no public parameter, because it can always be contained as part of a master secret key and constrained secret keys.

$\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x) =: y$**:** This is the deterministic evaluation algorithm that takes a public parameter pp, a master secret key msk $\in \mathcal{K}$, and an element $x \in \{0,1\}^n$ as input, and outputs an element $y \in \mathcal{R}$.

$\mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, f) \xrightarrow{\mathsf{R}} \mathsf{sk}_f$**:** This is the constraining algorithm that takes as input a public parameter pp, a master secret key msk, and a function $f \in \mathcal{F}_{\lambda,n}$, where $n = n(\lambda) = \mathrm{poly}(\lambda)$ is the input-length specified by pp. Then, it outputs a constrained key $\mathsf{sk}_f$.

$\mathsf{CEval}(\mathsf{pp}, \mathsf{sk}_f, x) =: y$**:** This is the deterministic constrained evaluation algorithm that takes a public parameter pp, a constrained key $\mathsf{sk}_f$, and an element $x \in \{0,1\}^n$ as input, and outputs an element $y \in \mathcal{R}$.

Whenever clear from the context, we will drop pp from the inputs of Eval, Constrain, and CEval, and the executions of them are denoted as "$\mathsf{Eval}(\mathsf{msk}, x)$", "$\mathsf{Constrain}(\mathsf{msk}, f)$", and "$\mathsf{CEval}(\mathsf{sk}_f, x)$", respectively.

**Correctness.** For correctness of a CPRF for a function class $\mathcal{F} = \{\mathcal{F}_{\lambda,k}\}_{\lambda,k\in\mathbb{N}}$, we require that for all $\lambda \in \mathbb{N}$, $\mathsf{pp} \xleftarrow{\mathsf{R}} \mathsf{Setup}(1^\lambda)$ (which specifies the input length $n = n(\lambda) = \mathrm{poly}(\lambda)$), $\mathsf{msk} \xleftarrow{\mathsf{R}} \mathsf{KeyGen}(\mathsf{pp})$, functions $f \in \mathcal{F}_{\lambda,n}$, and inputs $x \in \{0,1\}^n$ satisfying $f(x) = 0$, we have

$$\mathsf{CEval}\Big(\ \mathsf{Constrain}(\mathsf{msk}, f), x\ \Big) = \mathsf{Eval}(\mathsf{msk}, x).$$

We stress that a constrained key $\mathsf{sk}_f$ can compute the PRF if $f(x) = 0$. (This treatment is reversed from the original definition by Boneh and Waters [BW13].)

**Security.** Here, we give the security definitions for a CPRF. We only consider CPRFs that are secure in the presence of a single constrained key, for which we consider two flavors of security: *adaptive single-key security* and *selective-constraint no-evaluation security*.[14] The former notion captures security against adversaries $\mathcal{A}$ that may decide the constraining function $f$ any time during the experiment. (That is, $\mathcal{A}$ can specify the constraining function $f$ even after seeing some evaluation results of the CPRF.) In contrast, the latter notion captures security against adversaries that declare a constraining query at the beginning of the security game and have no access to the evaluation oracle. The definition below reflects these differences.

Formally, for a CPRF $\mathsf{CPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ (with input-length $n = n(\lambda)$) for a function class $\mathcal{F} = \{\mathcal{F}_{\lambda,k}\}_{\lambda,k\in\mathbb{N}}$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we define the single-key security experiment $\mathsf{Expt}^{\mathsf{cprf}}_{\mathsf{CPRF},\mathcal{F},\mathcal{A}}(\lambda)$ as described in Figure 1.

In the security experiment, the adversary $\mathcal{A}$'s single constraining query is captured by the function $f$ included in the first-stage algorithm $\mathcal{A}_1$'s output. Furthermore, $\mathcal{A}_1$ and $\mathcal{A}_2$ have access to the *challenge* oracle $\mathcal{O}_{\mathsf{Chal}}(\cdot)$ and the *evaluation* oracle $\mathsf{Eval}(\mathsf{msk}, \cdot)$, where the former oracle takes $x^* \in \{0,1\}^n$ as input, and returns either the actual evaluation result $\mathsf{Eval}(\mathsf{msk}, x^*)$ or the output $\mathsf{RF}(x^*)$ of a random function, depending on the challenge bit $\mathsf{coin} \in \{0,1\}$.

We say that an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in the experiment $\mathsf{Expt}^{\mathsf{cprf}}_{\mathsf{CPRF},\mathcal{F},\mathcal{A}}(\lambda)$ is *admissible* if $\mathcal{A}_1$ and $\mathcal{A}_2$ are PPT and respect the following restrictions:

- $f \in \mathcal{F}_{\lambda,n}$.
- $\mathcal{A}_1$ and $\mathcal{A}_2$ never make the same query twice.

[14] selective-constraint no-evaluation security was simply called no-evaluation security in [AMN+18].

$$
\begin{array}{|l|}
\hline
\mathsf{Expt}^{\mathsf{cprf}}_{\mathsf{CPRF},\mathcal{F},\mathcal{A}}(\lambda):\\
\quad \mathsf{coin} \xleftarrow{\mathsf{R}} \{0,1\}\\
\quad \mathsf{pp} \xleftarrow{\mathsf{R}} \mathsf{Setup}(1^\lambda)\\
\quad \mathsf{msk} \xleftarrow{\mathsf{R}} \mathsf{KeyGen}(\mathsf{pp})\\
\quad \mathsf{RF}(\cdot) \xleftarrow{\mathsf{R}} \mathsf{Func}(\{0,1\}^n,\mathcal{R})\\
\quad \mathcal{O}_{\mathsf{Chal}}(\cdot) := \begin{cases} \mathsf{Eval}(\mathsf{msk},\cdot) & \text{if } \mathsf{coin}=1\\ \mathsf{RF}(\cdot) & \text{if } \mathsf{coin}=0 \end{cases}\\
\quad (f,\mathsf{st}_{\mathcal{A}}) \xleftarrow{\mathsf{R}} \mathcal{A}_1^{\mathcal{O}_{\mathsf{Chal}}(\cdot),\mathsf{Eval}(\mathsf{msk},\cdot)}(\mathsf{pp})\\
\quad \mathsf{sk}_f \xleftarrow{\mathsf{R}} \mathsf{Constrain}(\mathsf{msk},f)\\
\quad \widehat{\mathsf{coin}} \xleftarrow{\mathsf{R}} \mathcal{A}_2^{\mathcal{O}_{\mathsf{Chal}}(\cdot),\mathsf{Eval}(\mathsf{msk},\cdot)}(\mathsf{sk}_f,\mathsf{st}_{\mathcal{A}})\\
\quad \text{Return } (\widehat{\mathsf{coin}} \stackrel{?}{=} \mathsf{coin}).\\
\hline
\end{array}
$$

Figure 1: The experiment for defining single-key security for a CPRF.

- All challenge queries $x^*$ made by $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfy $f(x^*) = 1$, and are distinct from any of the evaluation queries $x$ that they submit to the evaluation oracle $\mathsf{Eval}(\mathsf{msk}, \cdot)$.

Furthermore, we say that $\mathcal{A}$ is a *selective-constraint no-evaluation adversary* if $\mathcal{A}_1$ and $\mathcal{A}_2$ are PPT, and they do not make any queries, except that $\mathcal{A}_2$ is allowed to make only a single challenge query $x^*$ such that $f(x^*) = 1$.

**Definition 2.7 (Single-Key Security of CPRF).** *We say that a CPRF* CPRF *for a function class $\mathcal{F}$ is* adaptively single-key secure*, if for all admissible adversaries $\mathcal{A}$, the advantage* $\mathsf{Adv}^{\mathsf{cprf}}_{\mathsf{CPRF},\mathcal{F},\mathcal{A}}(\lambda) := 2 \cdot |\Pr[\mathsf{Expt}^{\mathsf{cprf}}_{\mathsf{CPRF},\mathcal{F},\mathcal{A}}(\lambda) = 1] - 1/2|$ *is negligible.*

*We define* selective-constraint no-evaluation security *of* CPRF *analogously, by replacing the phrase "all admissible adversaries $\mathcal{A}$" in the above definition with "all selective-constraint no-evaluation adversaries $\mathcal{A}$".*

*Remark* 2.8. As noted by Boneh and Waters [BW13], without loss of generality we can assume that $\mathcal{A}$ makes a challenge query only once, because security for a single challenge query can be shown to imply security for multiple challenge queries via a standard hybrid argument. Hence, in the rest of the paper we only use the security experiment with a single challenge query for simplicity.

## 2.4 Indistinguishability Obfuscation

Here, we recall the definition of indistinguishability obfuscation (iO) (for all circuits) [BGI+12, GGH+16].

**Definition 2.9 (Indistinguishability Obfuscation).** *We say that a PPT algorithm* iO *is a secure* indistinguishability obfuscator *(iO), if it satisfies the following properties:*

**Functionality:** iO *takes a security parameter $1^\lambda$ and a circuit $C$ as input, and outputs an obfuscated circuit $\widehat{C}$ that computes the same function as $C$. (We may drop $1^\lambda$ from an input to* iO *when $\lambda$ is clear from the context.)*

**Security:** *For all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the advantage function $\mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},\mathcal{A}}(\lambda)$ defined below is negligible:*

$$\mathsf{Adv}^{\mathsf{io}}_{\mathsf{iO},\mathcal{A}}(\lambda) \coloneqq 2 \cdot \left| \Pr\left[ \begin{array}{l} (C_0, C_1, \mathsf{st}) \xleftarrow{\mathsf{R}} \mathcal{A}_1(1^\lambda);\ \mathsf{coin} \leftarrow \{0,1\}; \\ \widehat{C} \xleftarrow{\mathsf{R}} \mathsf{iO}(1^\lambda, C_b);\ \widehat{\mathsf{coin}} \xleftarrow{\mathsf{R}} \mathcal{A}_2(\mathsf{st}, \widehat{C}) \end{array} : \widehat{\mathsf{coin}} = \mathsf{coin} \right] - \frac{1}{2} \right|.$$

*where it is required that $C_0$ and $C_1$ compute the same function and have the same description size.*

# 3 Partitionable Constrained Pseudorandom Function

In this section, we introduce a concept of Partitionable Constrained Pseudorandom Function (PCPRF), which is used as a building block for constructing our adaptively single-key secure CPRF. Then we construct a PCPRF for $\mathbf{NC}^1$ based on iO and the subgroup hiding assumption.

## 3.1 Definition

A PCPRF for $\mathcal{F}$ w.r.t. a function $h : \{0,1\}^n \to \{0,1\}^m$ consists of ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}, \mathsf{Merge}, \mathsf{MEval}$) where ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}$) forms a CPRF for $\mathcal{F}$. Two additional algorithms ,$\mathsf{Merge}$ and $\mathsf{MEval}$ works as follows.

$\mathsf{Merge}(\mathsf{msk}_0, \mathsf{msk}_1, u)$**:** This is the merging algorithm that takes two master keys $(\mathsf{msk}_0, \mathsf{msk}_1)$ and a partitioning policy $u \in \{0,1,\perp\}^m$, and outputs a merged key $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$.

$\mathsf{MEval}(\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u], x)$**:** This is the evaluation algorithm that takes a merged key $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ and $x \in \{0,1\}^n$ as input, and outputs $y$.

**Correctness.** In addition to the correctness as a CPRF, we require the following. For all $\lambda \in \mathbb{N}$, $\mathsf{pp} \xleftarrow{\mathsf{R}} \mathsf{Setup}(1^\lambda)$ (which specifies the input length $n = n(\lambda) = \mathrm{poly}(\lambda)$), $\mathsf{msk}_0, \mathsf{msk}_1 \xleftarrow{\mathsf{R}} \mathsf{KeyGen}(\mathsf{pp})$, $u \in \{0,1,\perp\}^m$, $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u] \xleftarrow{\mathsf{R}} \mathsf{Merge}(\mathsf{msk}_0, \mathsf{msk}_1, u)$ and inputs $x \in \{0,1\}^n$ we have

$$\mathsf{MEval}(\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u], x) = \mathsf{Eval}(\mathsf{msk}_{P_u(h(x))}, x)$$

where we recall that $P_u$ is as defined in Definition 2.4.

**Security.** We define two security requirements for PCPRFs. The first one is the security as a CPRF, and the second one is partition-hiding, which roughly means that a merged key hides the partition policy $u$ with which the merged key is generated.

**CPRF security.** We say that a PCPRF is selective-constraint no-evaluation secure if ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}$) is selective-constraint no-evaluation secure as a CPRF.[15]

[15]Though it is possible to define the adaptive security for PCPRFs in the similar way, we only define the selective-constraint no-evaluation security since we only need it.

**Partition-hiding.** For all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following advantage $\mathsf{Adv}^{\mathsf{ph}}_{\mathsf{PCPRF},\mathcal{A}}(\lambda)$, defined below, is negligible:

$$\mathsf{Adv}^{\mathsf{ph}}_{\mathsf{PCPRF},\mathcal{A}}(\lambda) :=$$
$$2 \cdot \left| \Pr\left[ \begin{array}{l} \mathsf{pp} \xleftarrow{R} \mathsf{Setup}(1^\lambda); \mathsf{msk}_0, \mathsf{msk}_1 \xleftarrow{R} \mathsf{KeyGen}(\mathsf{pp}); \\ (u, \mathsf{st}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp}); \\ \mathsf{k}_0 \xleftarrow{R} \mathsf{Merge}(\mathsf{msk}_0, \mathsf{msk}_1, \perp^m); \\ \mathsf{k}_1 \xleftarrow{R} \mathsf{Merge}(\mathsf{msk}_0, \mathsf{msk}_1, u); \\ \mathsf{coin} \leftarrow \{0,1\}; \widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}, \mathsf{k}_{\mathsf{coin}}) \end{array} : \widehat{\mathsf{coin}} = \mathsf{coin} \right] - \frac{1}{2} \right|.$$

We note that $\mathsf{k}_0$ generated by $\mathsf{Merge}(\mathsf{msk}_0, \mathsf{msk}_1, \perp^m)$ works completely identically to $\mathsf{msk}_0$, albeit in the sense that $\mathsf{MEval}(\mathsf{k}_0, x) = \mathsf{Eval}(\mathsf{msk}_0, x)$. This is since we have $P_{\perp^m}(h(x)) = 0$ for all $x \in \{0,1\}^n$.

## 3.2 Construction

Here, we construct a partition-hiding and selective-constraint no-evaluation secure PCPRF for $\mathbf{NC}^1$ based on iO and the subgroup hiding assumption. Before describing our scheme, we prepare some notations and describe class of functions our scheme supports. Since the function class our scheme supports is exactly the same as that of [AMN+18], the following two paragraphs are taken from [AMN+18].

### Notations.

In the following, we will sometimes abuse notation and evaluate a boolean circuit $C(\cdot) : \{0,1\}^\ell \to \{0,1\}$ on input $y \in \mathbb{R}^\ell$ for some ring $\mathbb{R}$. The evaluation is done by regarding $C(\cdot)$ as the arithmetic circuit whose AND gates $(y_1, y_2) \mapsto y_1 \wedge y_2$ being changed to the multiplication gates $(y_1, y_2) \mapsto y_1 y_2$, NOT gates $y \mapsto \neg y$ changed to the gates $y \mapsto 1 - y$, and the OR gates $(y_1, y_2) \mapsto y_1 \vee y_2$ changed to the gates $(y_1, y_2) \mapsto y_1 + y_2 - y_1 y_2$. It is easy to observe that if the input is confined within $\{0,1\}^\ell \subseteq \mathbb{R}$, the evaluation of the arithmetized version of $C(\cdot)$ equals to that of the binary version. (Here, we identify ring elements $0, 1 \in \mathbb{R}$ with the binary bit.) In that way, we can regard $C(\cdot)$ as an $\ell$-variate polynomial over $\mathbb{R}$. The degree of $C(\cdot)$ is defined as the maximum of the total degree of all the polynomials that appear during the computation.

### Class of Functions.

Let $n = \mathrm{poly}(\lambda)$, $z(n) = \mathrm{poly}(n)$, and $d(n) = O(\log n)$ be parameters. The function class that will be dealt with by the scheme is denoted by $\mathcal{F}^{\mathbf{NC}^1} = \{\mathcal{F}^{\mathbf{NC}^1}_{\lambda,n(\lambda)}\}_{\lambda \in \mathbb{N}}$, where $\mathcal{F}^{\mathbf{NC}^1}_{\lambda,n}$ consists of (Boolean) circuits $f$ whose input size is $n(\lambda)$, the description size is $z(n)$, and the depth is $d(n)$. We can set the parameters arbitrarily large as long as they do not violate the asymptotic bounds above, and thus the function class corresponds to $\mathbf{NC}^1$ circuits with bounded size. The following lemma will be helpful when describing our scheme.

**Lemma 3.1.** *([CH85, AMN+18]) Let $n = \mathrm{poly}(\lambda)$. There exists a family of universal circuit $\{U_n\}_{n \in \mathbb{N}}$ of degree $D(\lambda) = \mathrm{poly}(\lambda)$ such that $U_n(f, x) = f(x)$ for any $f \in \mathcal{F}^{\mathbf{NC}^1}_{\lambda,n(\lambda)}$ and $x \in \{0,1\}^n$.*

**Construction.**

Let $\mathcal{F}^{\mathbf{NC}^1} = \{\mathcal{F}^{\mathbf{NC}^1}_{\lambda,n}\}_{\lambda,n\in\mathbb{N}}$ be the family of the circuit defined as above and $\{U_n\}_{n\in\mathbb{N}}$ be the family of the universal circuit defined in Lemma 3.1. Let the parameter $D(\lambda)$ be the degree of the universal circuit (chosen as specified in Lemma 3.1). Since we will fix $n$ in the construction, we drop the subscripts and just denote $\mathcal{F}^{\mathbf{NC}^1}$ and $U$ in the following. Let $h : \{0,1\}^n \to \{0,1\}^m$ be any efficiently computable function.[16] The description of our PCPRF $\mathsf{PCPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}, \mathsf{Merge}, \mathsf{MEval})$ is given below.

$\mathsf{Setup}(1^\lambda)$**:** It obtains the group description $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2)$ by running $\mathcal{G} \xleftarrow{\mathsf{R}} \mathsf{GGen}(1^\lambda)$. It then outputs the public parameter $\mathsf{pp} := (N, \mathbb{G})$.

$\mathsf{KeyGen}(\mathsf{pp})$**:** It chooses $g \xleftarrow{\mathsf{R}} \mathbb{G}$, $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^2$, and $(b_1, ..., b_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$, $\alpha \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$.[17] It outputs $\mathsf{msk} := (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.

$\mathsf{Eval}(\mathsf{msk}, x)$**:** Given input $x \in \{0,1\}^n$, it computes $y := h(x)$ and outputs

$$X := g^{\prod_{i=1}^m s_{i,y_i} \cdot U((b_1,\ldots,b_z),(x_1,\ldots,x_n))/\alpha}.$$

$\mathsf{Constrain}(\mathsf{msk}, f)$**:** It first parses $(g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha) \leftarrow \mathsf{msk}$. Then it sets

$$b'_i := (b_i - f_i)\alpha^{-1} \mod N \quad \text{for } i \in [z]$$

where $f_i$ is the $i$-th bit of the binary representation of $f$. It then outputs

$$\mathsf{sk}_f := ((s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), f, b'_1, \ldots, b'_z, g, g^\alpha, \ldots, g^{\alpha^{D-1}}).$$

$\mathsf{CEval}(\mathsf{sk}_f, x)$**:** It parses $((s_{1,0}, s_{1,1}), ..., (s_{n,0}, s_{n,1}), f, b'_1, \ldots, b'_z, g, g^\alpha, \ldots, g^{\alpha^{D-1}}) \leftarrow \mathsf{sk}_f$. It can be shown that, from $(b'_1, ..., b'_z)$, $f$ and $x$, it is possible to efficiently compute $\{c_i\}_{i\in[D]}$ that satisfies

$$U((b_1, \ldots, b_z), (x_1, \ldots, x_n)) = f(x) + \sum_{j=1}^{D} c_j \alpha^j. \tag{1}$$

(We state this as Lemma 3.2 below.)

If $f(x) = 0$, it computes $y = h(x)$ and $X := (\prod_{j=1}^D (g^{\alpha^{j-1}})^{c_j})^{\prod_{i=1}^m s_{i,y_i}}$ and outputs $X$. Otherwise it outputs $\perp$.

$\mathsf{Merge}(\mathsf{msk}_0, \mathsf{msk}_1, u)$**:** Let $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ be a program as described in Figure 2. It computes and outputs

$$\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u] \xleftarrow{\mathsf{R}} i\mathcal{O}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u]).$$

$\mathsf{MEval}(\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u], x)$**:** It computes and outputs $y := \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u](x)$.

[16] The construction will be partition-hiding with respect to $h$. Looking ahead, we will show that PCPRF that is partition-hiding with respect to a balanced AHF is adaptively single-key secure in Section 4. There, we will set $h$ to be a balanced AHF. However, in this section, $h$ can be any efficiently computable function.

[17] This can be done by sampling in $\mathbb{Z}_N$; if it is not in $\mathbb{Z}_N^*$, sampling again until it is. This will succeed with an overwhelming probability since $N$ is a composite with two large prime factors.

| MergedKey[$\mathsf{msk}_0, \mathsf{msk}_1, u$] |
|---|
| Input: $x \in \{0,1\}^n$ |
| Constants: $\mathsf{pp} = (N, \mathbb{G})$ |
| $\mathsf{msk}_0 = (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$ |
| $\mathsf{msk}_1 = (\widehat{g}, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$ |
| $u \in \{0,1,\perp\}^m$ |
| Output $\mathsf{Eval}(\mathsf{msk}_{P_u(h(x))}, x)$ |

Figure 2: Description of Program MergedKey[$\mathsf{msk}_0, \mathsf{msk}_1, u$]

**Correctness.**

Here, we prove the correctness of our PCPRF given in Section 3. Correctness of MEval is easy to see. In the following, we prove the correctness of CEval. For proving the correctness, we rely on the following lemma. Though this can be proven similarly to [AMN+18, Lemma 2], we include the proof for completeness.

**Lemma 3.2.** *(Variant of [AMN+18, Lemma 2]) Given $(b'_1, ..., b'_z)$, $f$ and $x$, one can efficiently compute $\{c_i\}_{i\in[D]}$ satisfying Equation (1).*

*Proof.* The algorithm evaluates the circuit $U(\cdot)$ on input $(b'_1\mathsf{Z} + f_1, \ldots, b'_z\mathsf{Z} + f_z, x_1, \ldots, x_n)$ to obtain $\{c_i\}_{i\in\{0,1,\ldots,D\}}$ such that

$$U(b'_1\mathsf{Z} + f_1, \ldots, b'_z\mathsf{Z} + f_z, x_1, \ldots, x_n) = c_0 + \sum_{i\in[D]} c_i \mathsf{Z}^i \tag{2}$$

where $\mathsf{Z}$ denotes the indeterminant of the polynomial ring $\mathbb{Z}_N[\mathsf{Z}]$. Note that the computation is done over the ring $\mathbb{Z}_N[\mathsf{Z}]$ and can be efficiently performed, since we have $D = \mathrm{poly}(\lambda)$. We prove that $\{c_i\}_{i\in[D]}$ actually satisfies Equation (1). To see this, we first observe that by setting $\mathsf{Z} = 0$ in Equation (2), we obtain $c_0 = U(f_1, \ldots, f_z, x_1 \ldots, x_n) = f(x)$. To conclude, we further observe that by setting $\mathsf{Z} = \alpha$ in Equation (2), we recover Equation (1), since we have $b_j = b'_j\alpha + f_j$ by the definition of $b'_j$. This completes the proof of the lemma. ■

Suppose that we have Equation (1) and $f(x) = 0$. Then we have

$$\begin{aligned}(\prod_{j=1}^{D}(g^{\alpha^{j-1}})^{c_j})^{\prod_{i=1}^m s_{i,y_i}} &= (g^{\sum_{j=1}^D c_j\alpha^{j-1}})^{\prod_{i=1}^m s_{i,y_i}} \\ &= g^{\prod_{i=1}^m s_{i,y_i}\cdot U((b_1,\ldots,b_z),(x_1,\ldots,x_n))/\alpha}.\end{aligned}$$

Therefore CEval correctly evaluate the PRF if $f(x) = 0$.

**Theorem 3.3.** *If* iO *is a secure indistinguishability obfuscator and the subgroup hiding assumption holds for* GGen, *then* PCPRF *is selective-constraint no-evaluation secure PCPRF for $\mathcal{F}$ and partition-hiding with respect to $h$.*

## 3.3 Security of Our Partitionable CPRF

We present the proof of Theorem 3.3 in this section.

*Proof sketch of Theorem 3.3.* We have to prove that the construction satisfies the selective-constraint no-evaluation security and partition-hiding. From high level, the selective-constraint no-evaluation security is proven similarly to [AMN+18], and the partition-hiding is proven similarly to [HKW15].

**CPRF security.** Here, we prove PCPRF is selective-constraint no-evaluation secure. Namely, what we have to prove is that $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval})$ is selective-constraint no-evaluation secure CPRF for $\mathcal{F}$. The proof is very similar to the proof for selective-constraint no-evaluation secure CPRF in [AMN+18]. We note that we only use $(D-1)$-DDHI assumption, which holds under the subgroup hiding assumption, in this part, iO is irrelevant.

*Proof.* Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any selective-constraint no-evaluation adversary that attacks the selective-constraint no-evaluation security of PCPRF. We prove the above theorem by considering the following sequence of games.

Game 0**:** This is the real single-key security experiment $\mathsf{Expt}^{\mathsf{cprf}}_{\mathsf{PCPRF},\mathcal{F}^{\mathbf{NC}^1},\mathcal{A}}(\lambda)$ against the selective-constraint no-evaluation adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Namely,

$\mathsf{coin} \xleftarrow{\mathsf{R}} \{0,1\}$
$\mathsf{pp} \xleftarrow{\mathsf{R}} \mathsf{Setup}(1^\lambda)$
$\mathsf{msk} \xleftarrow{\mathsf{R}} \mathsf{KeyGen}(\mathsf{pp})$
$X^* \xleftarrow{\mathsf{R}} \mathbb{G}$
$(f, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\mathsf{R}} \mathcal{A}_1(\mathsf{pp})$
$\mathsf{sk}_f \xleftarrow{\mathsf{R}} \mathsf{Constrain}(\mathsf{msk}, f)$
$\widehat{\mathsf{coin}} \xleftarrow{\mathsf{R}} \mathcal{A}_2^{\mathcal{O}_{\mathsf{Chal}}(\cdot)}(\mathsf{sk}_f, \mathsf{st}_{\mathcal{A}})$
Return $(\widehat{\mathsf{coin}} \stackrel{?}{=} \mathsf{coin})$

where the challenge oracle $\mathcal{O}_{\mathsf{Chal}}(\cdot)$ is described below.

$\mathcal{O}_{\mathsf{Chal}}(x^*)$**:** Given $x^* \in \{0,1\}^n$ as input, it returns $\mathsf{Eval}(\mathsf{msk}, x^*)$ if $\mathsf{coin} = 1$ and $X^*$ if $\mathsf{coin} = 0$.

We recall that $\mathcal{O}_{\mathsf{Chal}}(\cdot)$ is queried at most once during the game.

Game 1**:** In this game, we change the way $\mathsf{sk}_f$ is sampled. In particular, we change the way of choosing $\{b_i\}_{i\in[z]}$ and $\{b'_i\}_{i\in[z]}$. Namely, given the constraining query $f$ from $\mathcal{A}_1$, the game picks $(b'_1, \ldots, b'_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$, $\alpha \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$, and sets $b_i := b'_i\alpha + f_i \mod N$ for $i \in [z]$.

Game 2 In this game, we change the challenge oracle $\mathcal{O}_{\mathsf{Chal}}(\cdot)$ as follows:

$\mathcal{O}_{\mathsf{Chal}}(x^*)$**:** Given $x^* \in \{0,1\}^n$ as input, if $\mathsf{coin} = 1$, then it does the following. It first computes $\{c_i\}_{i\in[D]}$ that satisfies

$$U((b_1, \ldots, b_z), (x_1^*, \ldots, x_n^*)) = 1 + \sum_{j=1}^{D} c_j \alpha^j$$

from $(b'_1, ..., b'_z)$, $f$ and $x$ by using Lemma 3.2, and returns $(g^{1/\alpha} \cdot \prod_{j=1}^D (g^{\alpha^{j-1}})^{c_j})^{\prod_{i=1}^m s_{i,y_i^*}}$ where $y^* = h(x^*)$. If $\mathsf{coin} = 0$, then it returns $X^*$.

Game 3**:** In this game, the challenge oracle use a uniformly random $\psi \xleftarrow{\mathsf{R}} \mathbb{G}$ instead of $g^{1/\alpha}$. Namely, it works as follows.

$\mathcal{O}_{\mathsf{Chal}}(x^*)$**:** Given $x^* \in \{0,1\}^n$ as input, if $\mathsf{coin} = 1$, then it does the following. It first computes $\{c_i\}_{i\in[D]}$ as in the previous game, picks $\psi \xleftarrow{\mathsf{R}} \mathbb{G}$ and returns $(\psi \cdot \prod_{j=1}^D (g^{\alpha^{j-1}})^{c_j})^{\prod_{i=1}^m s_{i,y_i^*}}$ where $y^* = h(x^*)$. If $\mathsf{coin} = 0$, then it returns $X^*$.

$\mathsf{Game}\ 4$ In this game, the oracle is changed as follows.

$\mathcal{O}_{\mathsf{Chal}}(x^*)$**:** Given $x^* \in \{0,1\}^n$ as input, it returns $X^*$ regardless of the value of coin.

Let $\mathsf{T}_i$ be the event that $\mathsf{Game}\ i$ returns 1.

**Lemma 3.4.** $\Pr[\mathsf{T}_1] = \Pr[\mathsf{T}_0]$

*Proof.* It can be seen that the distributions of $\mathsf{sk}_f$ are exactly the same in these games. Since the change is only conceptual, the lemma follows. ■

**Lemma 3.5.** $\Pr[\mathsf{T}_2] = \Pr[\mathsf{T}_1]$

*Proof.* We have

$$\begin{aligned}(g^{1/\alpha}\prod_{j=1}^{D}(g^{\alpha^{j-1}})^{c_j})^{\prod_{i=1}^{m} s_{i,y_i}} &= (g^{1/\alpha+\sum_{j=1}^{D} c_j\alpha^{j-1}})^{\prod_{i=1}^{m} s_{i,y_i}} \\ &= g^{\prod_{i=1}^{m} s_{i,y_i}\cdot U((b_1,\ldots,b_z),(x_1,\ldots,x_n))/\alpha}.\end{aligned}$$

Therefore outputs by $\mathcal{O}_{\mathsf{Chal}}$ in $\mathsf{Game}\ 1$ and $\mathsf{Game}\ 2$ are identical. Thus the change is only conceptual and thus the lemma follows. ■

**Lemma 3.6.** *If the $(D-1)$-DDHI assumption holds, then* $|\Pr[\mathsf{T}_3] - \Pr[\mathsf{T}_2]| = \mathrm{negl}(\lambda)$.

*Proof.* For the sake of the contradiction, let us assume that $|\Pr[\mathsf{T}_3] - \Pr[\mathsf{T}_2]|$ is non-negligible. We then construct an adversary $\mathcal{B}$ that breaks the $(D-1)$-DDHI assumption using $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

$\mathcal{B}((\mathbb{G}, N), g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^{D-1}}, \psi)$**:** Given the problem instance, $\mathcal{B}$ first gives the group description $\mathsf{pp} := (\mathbb{G}, N)$ to $\mathcal{A}_1$. Then, $\mathcal{A}_1$ outputs a constraining query $f$ along with its state $\mathsf{st}_{\mathcal{A}}$. Then, the adversary $\mathcal{B}$ picks $\mathsf{coin} \xleftarrow{\mathsf{R}} \{0,1\}$, $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^{2m}$, $(b'_1, \ldots, b'_z) \leftarrow \mathbb{Z}_N^z$, and gives $\mathsf{sk}_f := ((s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), f, b'_1, \ldots, b'_z, g, g^{\alpha}, \ldots, g^{\alpha^{D-1}})$ and the state $\mathsf{st}_{\mathcal{A}}$ to $\mathcal{A}_2$. When $\mathcal{A}_2$ makes a challenge query $x^*$ for $\mathcal{O}_{\mathsf{Chal}}(\cdot)$, $\mathcal{B}$ then returns $(\psi\cdot\prod_{j=1}^{D}(g^{\alpha^{j-1}})^{c_j})^{\prod_{i=1}^{m} s_{i,y_i^*}}$ if $\mathsf{coin} = 1$ where $\{c_i\}_{i\in[D]}$ is computed as in $\mathsf{Game}\ 2$ and $3$ and $y^* = h(x^*)$, and $X^*$ if $\mathsf{coin} = 0$ to $\mathcal{A}_2$. Finally, $\mathcal{A}_2$ outputs its guess $\widehat{\mathsf{coin}}$. $\mathcal{B}$ then outputs $(\mathsf{coin} \stackrel{?}{=} \widehat{\mathsf{coin}})$ as its guess.

It can easily be seen that $\mathcal{B}$ simulates $\mathsf{Game}_2$ if $\psi = g^{1/\alpha}$ and $\mathsf{Game}_3$ if $\psi \xleftarrow{\mathsf{R}} \mathbb{G}$. The lemma readily follows. ■

**Lemma 3.7.** $\Pr[\mathsf{T}_3] = \Pr[\mathsf{T}_4]$

*Proof.* In $\mathsf{Game}\ 3$, the response to the challenge query is a random group element of $\mathbb{G}$ regardless of the value of coin. Therefore, the change is only conceptual. ■

**Lemma 3.8.** *We have* $|\Pr[\mathsf{T}_4] - 1/2| = 0$.

*Proof.* In $\mathsf{Game}\ 4$ everything $\mathcal{A}$ sees is independent from coin, and thus there is no way to guess it with non-zero advantage. ■

Therefore, the advantage of $\mathcal{A}$ is $2\cdot|\Pr[\mathsf{T}_0] - 1/2| = \mathrm{negl}(\lambda)$. Finally, we complete the proof by noting that the $(D-1)$-DDHI assumption holds under the subgroup hiding assumption. ■

| $\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$ |
|---|
| Input: $x \in \{0,1\}^n$ |
| Constants: $\mathsf{pp} = (N, \mathbb{G})$ |
| $\mathsf{msk}_0 = (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$ |
| Compute $y := h(x)$ |
| Output $g^{\prod_{i=1}^{m} s_{i,y_i} \cdot U((b_1,\ldots,b_z),(x_1,\ldots,x_n))/\alpha}$. |

Figure 3: Description of Program $\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$

**Partition-hiding.** We want to prove that $\mathsf{k}$ generated by $\mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m])$ and generated by $\mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u])$ are computationally indistinguishable. The difficulty is that $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ and $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ do not have the same functionality, and thus we cannot simply use the security of $\mathsf{iO}$ to conclude it.[18] Actually, this can be proven by using the subgroup hiding assumption in a sophisticated way as in the work by Hohenberger, Koppula and Waters [HKW15]. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT adversary against the partition-hiding property. We prove the above theorem by considering the following sequence of games. We underline modifications from the previous one in descriptions of games.

$\mathsf{Game}\ 0$**:** This game corresponds to the case of $\mathsf{coin} = 0$ in the experiment defining the partition-hiding. More precisely,

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{\mathsf{R}} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_\mathcal{A}) \xleftarrow{\mathsf{R}} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{\mathsf{R}} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$. Then choose $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^{2m}$. Set $\mathsf{msk}_0 := (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
   Choose $\widehat{g} \xleftarrow{\mathsf{R}} \mathbb{G}$, $(\widehat{b}_1, ..., \widehat{b}_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$ and $\widehat{\alpha} \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$. Then choose $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^{2m}$. Set $\mathsf{msk}_1 := (\widehat{g}, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
4. Compute $\mathsf{k} \xleftarrow{\mathsf{R}} \mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m])$
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{\mathsf{R}} \mathcal{A}_2(\mathsf{st}_\mathcal{A}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

$\mathsf{Game}\ 1$**:** In this game, we set $\mathsf{k}$ as an obfuscation of $\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$, which is described in Figure 3.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{\mathsf{R}} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_\mathcal{A}) \xleftarrow{\mathsf{R}} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{\mathsf{R}} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$. Then choose $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^{2m}$. Set $\mathsf{msk}_0 := (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
4. Compute $\underline{\mathsf{k} \xleftarrow{\mathsf{R}} \mathsf{iO}(\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0])}$
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{\mathsf{R}} \mathcal{A}_2(\mathsf{st}_\mathcal{A}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

$\mathsf{Game}\ 2$**:** In this game, we generate $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1})$ in a different way.

[18] If one relies on the technique of "exponential number of hybrids" (e.g., [CLTV15]), then we can prove the indistinguishability of these two cases without relying on subgroup hiding. However, the technique requires sub-exponentially secure iO, which we want to avoid.

MergedKey-Zero$'$[msk$'_0, u, v_0, ..., v_{m-1}, w$]

Input: $x \in \{0,1\}^n$

Constants: pp $= (N, \mathbb{G})$

$v_0, ..., v_{m-1}, w \in \mathbb{G}^{m+1}$

msk$'_0 = ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$

$u \in \{0, 1, \perp\}^m$

Compute $y := h(x)$

If $P_u(y) = 0$

Output $w^{\prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha}$.

Else

Compute $r := |\{i \in [m] | u_i = y_i\}|$

Output $v_r^{\prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha}$.

Figure 4: Description of Program MergedKey-Zero$'$[msk$'_0, u, v_0, ..., v_{m-1}, w$]

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R}$ GGen$(1^\lambda)$, Set pp $:= (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{R} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{R} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{R} \mathbb{Z}_N^*$ and $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$. Set
   $$s_{i,\eta} := \begin{cases} \beta \cdot s'_{i,\eta} & \text{If } u_i = \perp \vee \eta = u_i \\ s'_{i,\eta} & \text{Otherwise} \end{cases}.$$
   Set msk$_0 := (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
4. Compute k $\xleftarrow{R}$ MergedKey-Zero[msk$_0$]
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 3**:** In this game, we set k as an obfuscation of MergedKey-Zero$'$[msk$'_0, u, v_0, ..., v_{m-1}, w$]), which is described in Figure 4.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R}$ GGen$(1^\lambda)$, Set pp $:= (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{R} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{R} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{R} \mathbb{Z}_N^*$ and $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Set $v_j := g^{\beta^j}$ for $j \in \{0, ..., m-1\}$ and $w := g^{\beta^m}$.
   Set msk$'_0 := ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$.
4. Compute k $\xleftarrow{R}$ iO(MergedKey-Zero$'$[msk$'_0, u, v_0, ..., v_{m-1}, w$])
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 4**:** In this game, we randomly choose $w$ from $\mathbb{G}$, which was set to be $g^{\beta^m}$ in the previous game.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R}$ GGen$(1^\lambda)$, Set pp $:= (N, \mathbb{G})$.

2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{R} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{R} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{R} \mathbb{Z}_N^*$ and $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Set $v_j := g^{\beta^j}$ for $j \in \{0, ..., m-1\}$. Choose $w \xleftarrow{R} \mathbb{G}$.
   Set $\mathsf{msk}'_0 := ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{iO}(\mathsf{MergedKey\text{-}Zero}'[\mathsf{msk}'_0, u, v_0, ..., v_{m-1}, w])$
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 5**:** In this game, we randomly choose $g$ and $w$ from $\mathbb{G}_q$ and $\mathbb{G}_p$, respectively, which are randomly chosen from $\mathbb{G}$ in the previous game.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{R} \mathbb{G}_q$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{R} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{R} \mathbb{Z}_N^*$ and $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Set $v_j := g^{\beta^j}$ for $j \in \{0, ..., m-1\}$. Choose $w \xleftarrow{R} \mathbb{G}_p$.
   Set $\mathsf{msk}'_0 := ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{iO}(\mathsf{MergedKey\text{-}Zero}'[\mathsf{msk}'_0, u, v_0, ..., v_{m-1}, w])$
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 6**:** In this game, we set $\mathsf{k}$ as an obfuscation of $\mathsf{MergedKey\text{-}Alt}[\mathsf{msk}'_0, \mathsf{msk}'_1, u, v_0, ..., v_{m-1}, w]$, which is described in Figure 5.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{R} \mathbb{G}_q$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{R} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{R} \mathbb{Z}_N^*$ and $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Set $s'_{i,\eta,p} := s'_{i,\eta} \mod p$ and $s'_{i,\eta,q} := s'_{i,\eta} \mod q$ for $i \in [m]$ and $\eta \in \{0, 1\}$.
   Set $b_{i,p} := b_i \mod p$ and $b_{i,q} := b_i \mod q$ for $i \in [m]$.
   $\alpha_p := \alpha \mod p$ and $\alpha_q := \alpha \mod q$.
   Set $v_j := g^{\beta^j}$ for $j \in \{0, ..., m-1\}$. Choose $w \xleftarrow{R} \mathbb{G}_p$.
   Set $\mathsf{msk}'_0 := ((s'_{1,0,p}, s'_{1,1,p}), ..., (s'_{m,0,p}, s'_{m,1,p}), b_{1,p}, \ldots, b_{z,p}, \alpha_p)$.
   Set $\mathsf{msk}'_1 := ((s'_{1,0,q}, s'_{1,1,q}), ..., (s'_{m,0,q}, s'_{m,1,q}), b_{1,q}, \ldots, b_{z,q}, \alpha_q)$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{iO}(\mathsf{MergedKey\text{-}Alt}[\mathsf{msk}'_0, \mathsf{msk}'_1, u, v_0, ..., v_{m-1}, w])$.
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 7**:** In this game, we modify how to generate $s'_{i,\eta,q}$, $b_{i,q}$ and $\alpha_q$.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.

$\mathsf{MergedKey\text{-}Alt}[\mathsf{msk}'_0, \mathsf{msk}'_1, u, v_0, ..., v_{m-1}, w]$

Input: $x \in \{0,1\}^n$

Constants: $\mathsf{pp} = (N, \mathbb{G})$

$v_0, ..., v_{m-1}, w \in \mathbb{G}^{m+1}$

$\mathsf{msk}'_0 = ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$

$\mathsf{msk}'_1 = ((\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$

$u \in \{0,1,\perp\}^m$

Compute $y := h(x)$

If $P_u(y) = 0$

Output $w^{\prod_{i=1}^m s'_{i,y_i} \cdot U((b_1,\ldots,b_z),(x_1,\ldots,x_n))/\alpha}$.

Else

Compute $r := |\{i \in [m] | u_i = y_i\}|$

Output $v_r^{\prod_{i=1}^m \widehat{s}_{i,y_i} \cdot U((\widehat{b}_1,\ldots,\widehat{b}_z),(x_1,\ldots,x_n))/\widehat{\alpha}}$.

Figure 5: Description of Program $\mathsf{MergedKey\text{-}Alt}[\mathsf{msk}'_0, \mathsf{msk}'_1, u, v_0, ..., v_{m-1}, w]$

3. Choose $g \xleftarrow{R} \mathbb{G}_q$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{R} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{R} \mathbb{Z}_N^*$ and $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Choose $(\widehat{b}_1, ..., \widehat{b}_z) \xleftarrow{R} \mathbb{Z}_N^z$, $\widehat{\alpha} \xleftarrow{R} \mathbb{Z}_N^*$, and $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Set $s'_{i,\eta,p} := s'_{i,\eta} \mod p$ and $s'_{i,\eta,q} := \widehat{s}_{i,\eta} \mod q$ for $i \in [m]$, $\eta \in \{0,1\}$.
   Set $b_{i,p} := b_i \mod p$ and $b_{i,q} := \widehat{b}_i \mod q$ for $i \in [m]$.
   Set $\alpha_p := \alpha \mod p$ and $\alpha_q := \widehat{\alpha} \mod q$.
   Set $v_j := g^{\beta^j}$ for $j \in \{0, ..., m-1\}$. Choose $w \xleftarrow{R} \mathbb{G}_p$.
   Set $\mathsf{msk}'_0 := ((s'_{1,0,p}, s'_{1,1,p}), ..., (s'_{m,0,p}, s'_{m,1,p}), b_{1,p}, \ldots, b_{z,p}, \alpha_p)$.
   Set $\mathsf{msk}'_1 := ((s'_{1,0,q}, s'_{1,1,q}), ..., (s'_{m,0,q}, s'_{m,1,q}), b_{1,q}, \ldots, b_{z,q}, \alpha_q)$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{iO}(\mathsf{MergedKey\text{-}Alt}[\mathsf{msk}'_0, \mathsf{msk}'_1, u, v_0, ..., v_{m-1}, w])$.
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 8**:** In this game, we modify the way to set $\mathsf{msk}'_0$ and $\mathsf{msk}'_1$.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{R} \mathbb{G}_q$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{R} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{R} \mathbb{Z}_N^*$ and $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Choose $(\widehat{b}_1, ..., \widehat{b}_z) \xleftarrow{R} \mathbb{Z}_N^z$, $\widehat{\alpha} \xleftarrow{R} \mathbb{Z}_N^*$, and $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Set $v_j := g^{\beta^j}$ for $j \in \{0, ..., m-1\}$. Choose $w \xleftarrow{R} \mathbb{G}_p$.
   Set $\mathsf{msk}'_0 := ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$.
   Set $\mathsf{msk}'_1 := ((\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{iO}(\mathsf{MergedKey\text{-}Alt}[\mathsf{msk}'_0, \mathsf{msk}'_1, u, v_0, ..., v_{m-1}, w])$.
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

$\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u]$

Input: $x \in \{0,1\}^n$

Constants: $\mathsf{pp} = (N, \mathbb{G})$

$v_0, ..., v_{m-1}, w \in \mathbb{G}^{m+1}$

$\mathsf{msk}'_0 = (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$

$\mathsf{msk}'_1 = (\widehat{g}, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}'_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$

$u \in \{0, 1, \perp\}^m$

Compute $y := h(x)$

If $P_u(y) = 0$

Output $g^{\prod_{i=1}^m s_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha}$.

Else

Output $\widehat{g}^{\prod_{i=1}^m \widehat{s}_{i,y_i} \cdot U((\widehat{b}_1,...,\widehat{b}_z),(x_1,...,x_n))/\widehat{\alpha}}$.

Figure 6: Description of Program $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u]$, more concretely

Game 9**:** In this game, we set $\mathsf{k}$ to be an obfuscation of $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u]$, which is described in Figure 2. For clarity, we give more concrete description of $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ in Figure 6.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{\mathsf{R}} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_\mathcal{A}) \xleftarrow{\mathsf{R}} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{\mathsf{R}} \mathbb{G}_q$, $(b_1, ..., b_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$ and $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^{2m}$.
   Set
   $$s_{i,\eta} := \begin{cases} \beta \cdot s'_{i,\eta} & \text{If } u_i = \perp \vee \eta = u_i \\ s'_{i,\eta} & \text{Otherwise} \end{cases}.$$
   Choose $(\widehat{b}_1, ..., \widehat{b}_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$, $\widehat{\alpha} \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$, and $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^{2m}$.
   Choose $w \xleftarrow{\mathsf{R}} \mathbb{G}_p$.
   Set $\mathsf{msk}_0 := (w, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
   Set $\mathsf{msk}_1 := (g, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
4. Compute $\mathsf{k} \xleftarrow{\mathsf{R}} \mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u])$.
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{\mathsf{R}} \mathcal{A}_2(\mathsf{st}_\mathcal{A}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 10**:** In this game, we modify the way to set $s_{i,\eta}$.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{\mathsf{R}} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_\mathcal{A}) \xleftarrow{\mathsf{R}} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{\mathsf{R}} \mathbb{G}_q$, $(b_1, ..., b_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$.
   Choose $\beta \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$ and $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^{2m}$.
   Choose $(\widehat{b}_1, ..., \widehat{b}_z) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^z$, $\widehat{\alpha} \xleftarrow{\mathsf{R}} \mathbb{Z}_N^*$, and $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \xleftarrow{\mathsf{R}} \mathbb{Z}_N^{2m}$.
   Choose $w \xleftarrow{\mathsf{R}} \mathbb{G}_p$.

Set $\mathsf{msk}_0 := (w, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
Set $\mathsf{msk}_1 := (g, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.

4. Compute $\mathsf{k} \overset{\mathsf{R}}{\leftarrow} \mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u])$.
5. Compute $\widehat{\mathsf{coin}} \overset{\mathsf{R}}{\leftarrow} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 11**:** In this game, we randomly choose $g$ and $w$ from $\mathbb{G}$, which are chosen from $\mathbb{G}_q$ and $\mathbb{G}_p$ in the previous game.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \overset{\mathsf{R}}{\leftarrow} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \overset{\mathsf{R}}{\leftarrow} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $\underline{g \overset{\mathsf{R}}{\leftarrow} \mathbb{G}}$, $(b_1, ..., b_z) \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^z$, and $\alpha \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^*$.
Choose $\beta \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^*$ and $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^{2m}$.
Choose $(\widehat{b}_1, ..., \widehat{b}_z) \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^z$, $\widehat{\alpha} \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^*$, and $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^{2m}$.
Choose $\underline{w \overset{\mathsf{R}}{\leftarrow} \mathbb{G}}$.
Set $\mathsf{msk}_0 := (w, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
Set $\mathsf{msk}_1 := (g, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
4. Compute $\mathsf{k} \overset{\mathsf{R}}{\leftarrow} \mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u])$.
5. Compute $\widehat{\mathsf{coin}} \overset{\mathsf{R}}{\leftarrow} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

Game 12**:** This game is the same as the previous game except that we rename $g$ and $w$ by $\widehat{g}$ and $g$.

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \overset{\mathsf{R}}{\leftarrow} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_{\mathcal{A}}) \overset{\mathsf{R}}{\leftarrow} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $\underline{\widehat{g} \overset{\mathsf{R}}{\leftarrow} \mathbb{G}}$, $(b_1, ..., b_z) \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^z$, and $\alpha \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^*$.
Choose $\beta \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^*$ and $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^{2m}$.
Choose $(\widehat{b}_1, ..., \widehat{b}_z) \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^z$, $\widehat{\alpha} \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^*$, and $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \overset{\mathsf{R}}{\leftarrow} \mathbb{Z}_N^{2m}$.
Choose $\underline{g \overset{\mathsf{R}}{\leftarrow} \mathbb{G}}$.
Set $\mathsf{msk}_0 := (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
Set $\mathsf{msk}_1 := (\widehat{g}, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
4. Compute $\mathsf{k} \overset{\mathsf{R}}{\leftarrow} \mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u])$.
5. Compute $\widehat{\mathsf{coin}} \overset{\mathsf{R}}{\leftarrow} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

This game corresponds to the case of $\mathsf{coin} = 1$ in the experiment defining the partition-hiding.

Let $\mathsf{T}_i$ be the event that Game $i$ returns 1. What we should prove is that $|\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_{12}]| = \mathrm{negl}(\lambda)$. We prove this by the following lemmas.

**Lemma 3.9.** *If* $\mathsf{iO}$ *is a secure indistinguishability obfuscator, then* $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_0]| = \mathrm{negl}(\lambda)$.

*Proof.* Since $P_{\perp^m}(h(x)) = 0$ for all $x \in \{0,1\}^n$, $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ outputs $\mathsf{Eval}(\mathsf{msk}_0, x)$ for all $x$. Therefore, we have that $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ and $\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$ have identical functionality. Therefore $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_0]|$ is negligible by the security of $\mathsf{iO}$. ∎

**Lemma 3.10.** $\Pr[\mathsf{T}_2] = \Pr[\mathsf{T}_1]$.

*Proof.* The only difference between Game 1 and Game 2 is how to choose $s_{i,\eta}$. Namely, we choose $s_{i,\eta}$ uniformly from $\mathbb{Z}_N$ in Game 1, whereas we set them to be $\beta \cdot s'_{i,\eta}$ or $s'_{i,\eta}$ depending on if $\eta = u_i$ by using uniformly chosen $s'_{i,\eta} \xleftarrow{\mathsf{R}} \mathbb{Z}_N$. In Game 2, in both cases, $s_{i,\eta}$ is uniformly distributed because $\beta \in \mathbb{Z}_N^*$. Therefore these games are identical from the view of $\mathcal{A}$. ■

**Lemma 3.11.** *If* $\mathsf{iO}$ *is a secure indistinguishability obfuscator, then* $|\Pr[\mathsf{T}_3] - \Pr[\mathsf{T}_2]| = \mathrm{negl}(\lambda)$.

*Proof.* We claim that $\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$ and $\mathsf{MergedKey\text{-}Zero}'[\mathsf{msk}'_0, u, v_0, ..., v_{m-1}, w]$ have identical functionality. From this claim, the lemma easily follows from the security of $\mathsf{iO}$. We prove the claim as follows. $\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$ computes $g^{\prod_{i=1}^{m} s_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha}$ where $y = h(x)$ for all inputs $x \in \{0,1\}^n$. Since we have

$$s_{i,\eta} := \begin{cases} \beta \cdot s'_{i,\eta} & \text{If } u_i = \perp \vee \eta = u_i \\ s'_{i,\eta} & \text{Otherwise} \end{cases},$$

if $P_u(y) = 0$ (i.e., $u_i = \perp$ or $y_i = u_i$ holds for all $i \in [m]$), then we have

$$\begin{aligned} g^{\prod_{i=1}^{m} s_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha} &= g^{\beta^m \prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha} \\ &= w^{\prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha} \end{aligned}$$

where $w := g^{\beta^m}$ as defined in Game 3. On the other hand, if $P_u(y) = 1$, then we have

$$\begin{aligned} g^{\prod_{i=1}^{m} s_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha} &= g^{\beta^r \prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha} \\ &= v_r^{\prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha} \end{aligned}$$

where $r := |\{i \in [m] | u_i = y_i\}| \leq m-1$ and $v_r := g^{\beta^r}$ as defined in Game 3. This means that an output for input $x$ of $\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$ is identical to that of $\mathsf{MergedKey\text{-}Zero}'[\mathsf{msk}'_0, u, v_0, ..., v_{m-1}, w]$. ■

**Lemma 3.12.** *If the* $(m-1)$*-DDH assumption holds, then* $|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_3]| = \mathrm{negl}(\lambda)$.

*Proof.* The only difference between these games is that how to set $w$. Namely, that is set to be $g^{\beta^m}$ in Game 3 and that is uniformly chosen from $\mathbb{G}$ in Game 4. Since all other parts of these two games can be simulated by using $\mathsf{pp} = (N, \mathbb{G})$, $(g, g^{\beta}, ..., g^{\beta^{m-1}})$ and other elements that are independent from them, if $|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_3]|$ is non-negligible, then we can construct an adversary that breaks the $(m-1)$-DDH assumption. Thus it is negligible under the $(m-1)$-DDH assumption. ■

**Lemma 3.13.** *If the subgroup hiding assumption holds w.r.t.* $\mathsf{GGen}$*, then* $|\Pr[\mathsf{T}_5] - \Pr[\mathsf{T}_4]| = \mathrm{negl}(\lambda)$.

*Proof.* The only difference between these games is that how to choose $g$ and $w$. Namely, they are uniformly chosen from $\mathbb{G}$ in Game 4, and they are uniformly chosen from $\mathbb{G}_q$ and $\mathbb{G}_p$, respectively in Game 5. First, we consider a hybrid game Game 4.5 where $g$ is chosen from $\mathbb{G}$ and $w$ is chosen from $\mathbb{G}_p$. Since all elements used in Game 4 and Game 4.5 except $w$ can be simulated by using $(N, \mathbb{G})$ (especially without knowing $(p, q)$), $\mathcal{A}$ cannot distinguish these two games under the subgroup hiding assumption. Similarly, all elements used in Game 4.5 and Game 5 except $g$ can be simulated by using $(N, \mathbb{G}, g_2 \in \mathbb{G}_q)$ (especially without knowing $(p, q)$ again), $\mathcal{A}$ cannot distinguish these two games under the subgroup hiding assumption. Thus by the triangle inequality, $\mathcal{A}$ cannot distinguish Game 4 and Game 5 under the subgroup hiding assumption. ■

**Lemma 3.14.** *If* iO *is a secure indistinguishability obfuscator, then* $|\Pr[\mathsf{T}_6] - \Pr[\mathsf{T}_5]| = \mathrm{negl}(\lambda)$.

*Proof.* If we can prove that the programs $\mathsf{MergedKey\text{-}Zero}'[\mathsf{msk}'_0, u, v_0, ..., v_{m-1}, w]$ and $\mathsf{MergedKey\text{-}Alt}[\mathsf{msk}'_0, \mathsf{msk}'_1, u, v_0, ..., v_{m-1}, w]$ have identical functionality, then the lemma easily follows from the security of iO. In the following, we prove it. Since we have $w \in \mathbb{G}_p$, we have

$$w^{\prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha} = w^{(\prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha \mod p)}.$$

Therefore $w^{\prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha}$ does not change even if we take modulo $p$ of $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$ before computing it. Similarly, since we have $g \in \mathbb{G}_q$, we have $v_r \in \mathbb{G}_q$, and thus $v_r^{\prod_{i=1}^{m} s'_{i,y_i} \cdot U((b_1,...,b_z),(x_1,...,x_n))/\alpha}$ does not change even if we take modulo $q$ of $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$ before computing it. This means that $\mathsf{MergedKey\text{-}Zero}'[\mathsf{msk}'_0, u, v_0, ..., v_{m-1}, w]$ and $\mathsf{MergedKey\text{-}Alt}[\mathsf{msk}'_0, \mathsf{msk}$ have identical functionality. ■

**Lemma 3.15.** $\Pr[\mathsf{T}_7] = \Pr[\mathsf{T}_6]$.

*Proof.* The difference between these games is how to generate $s'_{i,\eta,q}$, $b_{i,q}$, and $\alpha_q$. Namely, they are derived from $s'_{i,\eta}$, $b_i$ and $\alpha$ that are also used for generating $s'_{i,\eta,p}$, $b_{i,p}$, and $\alpha_p$ in Game 6 whereas they are derived from $\widehat{s}_{i,\eta}$, $\widehat{b}_i$ and $\widehat{\alpha}$ that are independent random values of $s'_{i,\eta}$, $b_i$ and $\alpha$. By the Chinese remainder theorem, $s'_{i,\eta,q}$, $b_{i,q}$, and $\alpha_q$ are uniform on $\mathbb{Z}_q$ and independent from $s'_{i,\eta,p}$, $b_{i,p}$, and $\alpha_p$. Therefore the joint distribution of $s'_{i,\eta,p}$, $b_{i,p}$, $\alpha_p$, $s'_{i,\eta,q}$, $b_{i,q}$, and $\alpha_q$ is identical in these two games. ■

**Lemma 3.16.** *If* iO *is a secure indistinguishability obfuscator, then* $|\Pr[\mathsf{T}_8] - \Pr[\mathsf{T}_7]| = \mathrm{negl}(\lambda)$.

*Proof.* This can be proven similarly to Lemma 3.14. ■

**Lemma 3.17.** *If* iO *is a secure indistinguishability obfuscator, then* $|\Pr[\mathsf{T}_9] - \Pr[\mathsf{T}_8]| = \mathrm{negl}(\lambda)$.

*Proof.* This can be proven similarly to Lemma 3.11. ■

**Lemma 3.18.** $\Pr[\mathsf{T}_{10}] = \Pr[\mathsf{T}_9]$.

*Proof.* This can be proven similarly to Lemma 3.10. ■

**Lemma 3.19.** *If the subgroup hiding assumption holds w.r.t.* GGen*, then* $|\Pr[\mathsf{T}_{11}] - \Pr[\mathsf{T}_{10}]| = \mathrm{negl}(\lambda)$.

*Proof.* This can be proven similarly to Lemma 3.13. ■

**Lemma 3.20.** $\Pr[\mathsf{T}_{11}] = \Pr[\mathsf{T}_{12}]$.

*Proof.* From Game 11 to Game 12, we just renamed $g$ and $w$ by $\hat{g}$ and $g$, respectively. ■

By Lemma 2.3, the $(m-1)$-DDH assumption can be reduced to the subgroup hiding assumption. Therefore if iO is a secure indistinguishability obfuscator and the subgroup hiding assumption holds, then $\Pr[\mathsf{T}_0] - \Pr[\mathsf{T}_{12}] = \mathrm{negl}(\lambda)$. Thus, we complete the proof of the partition-hiding property of PCPRF. This completes the proof of Theorem 3.3. ■

$$\boxed{\begin{array}{l}\mathsf{ConstrainedKey}[\mathsf{msk}, f] \\ \text{Input: } x \in \{0,1\}^n \\ \text{Constants:}\mathsf{pp}, \mathsf{msk}, f \\ \text{If } f(x) = 0 \\ \quad \text{Output } \mathsf{Eval}(\mathsf{msk}, x) \\ \text{Else} \\ \quad \text{Output } \bot \end{array}}$$

Figure 7: Description of Program $\mathsf{ConstrainedKey}[\mathsf{msk}, f]$

# 4 Adaptively Single-key Secure CPRF

In this section, we construct an adaptively single-key secure CPRF based on iO and a partition-hiding no-evaluation secure PCPRF. By instantiating the latter with our construction of PCPRF in Section 3.2, we obtain the first adaptively single-key secure CPRF for $\mathbf{NC}^1$ in the standard model.

## 4.1 Construction

Let $\mathsf{PCPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}, \mathsf{Merge}, \mathsf{MEval})$ be a partition-hiding and selective-constraint no-evaluation secure PCPRF for function class $\mathcal{F}$. Then we construct CPRF $\mathsf{CPRF} = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Eval}', \mathsf{Constrain}', \mathsf{CEval}')$ for the same function class as follows.

$\mathsf{Setup}'(1^\lambda)$**:** This algorithm is completely identical to $\mathsf{Setup}(1^\lambda)$.

$\mathsf{KeyGen}'(\mathsf{pp})$**:** This algorithm is completely identical to $\mathsf{KeyGen}(\mathsf{pp})$.

$\mathsf{Eval}'(\mathsf{msk}, x)$**:** This algorithm is completely identical to $\mathsf{Eval}(\mathsf{msk}, x)$.

$\mathsf{Constrain}'(\mathsf{msk}, f)$**:** It computes and outputs $\mathsf{sk}_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\mathsf{ConstrainedKey}[\mathsf{msk}, f])$ where $\mathsf{ConstrainedKey}[\mathsf{msk}, f]$ is a program described in Figure 7.

$\mathsf{CEval}'(\mathsf{sk}_f, x)$**:** It computes and outputs $\mathsf{sk}_f(x)$.

We note that the program $\mathsf{ConstrainedKey}[\mathsf{msk}, f]$ is padded so that the size of it is the same size as the programs that appear in the security proof. See also Remark 4.11.

The following theorem addresses the security of the above construction. We require $\mathcal{F}$ to contain some basic functions in the theorem. However, this restriction is very mild. Indeed, the requirement for the function class is satisfied in our construction of PCPRF in Section 3.2.

**Theorem 4.1.** *Let $\mathcal{F}$ be a function class that contains constant functions and punctured function $g_y : \{0,1\}^n \to \{0,1\}$ defined as $g_y(x) = (x \stackrel{?}{=} y)$ for all $y \in \{0,1\}^n$. If* iO *is a secure indistinguishability obfuscator and* PCPRF *is both partition-hiding with respect to a balanced AHF $h : \{0,1\}^n \to \{0,1\}^m$ and selective-constraint no-evaluation secure PCPRF for $\mathcal{F}$, then* CPRF *constructed above is an adaptively single-key secure CPRF for $\mathcal{F}$.*

By combining Theorems 3.3 and 4.1, we obtain the following theorem.

**Theorem 4.2.** *If there exists a secure indistinguishability obfuscator and a group generator for which the subgroup hiding assumption holds, then there exists an adaptively single-key secure CPRF for the function class $\mathcal{F}^{\mathbf{NC}^1}$, which is defined in Section 3.*

### 4.2 Security of Our CPRF

We present the proof of Theorem 4.1 in this section.

*Proof.* We prove the theorem by following steps. We denote the master secret key of the scheme by $\mathsf{msk}_0$ for notational convenience. Let $\mathcal{A}$ be a PPT adversary that breaks adaptive single-key security of the scheme. In addition, let $\epsilon = \epsilon(\lambda)$ and $Q = Q(\lambda)$ be its advantage and the upper bound on the number of evaluation queries, respectively. By assumption, $Q(\lambda)$ is polynomially bounded and there exists a noticeable function $\epsilon_0(\lambda)$ such that $\epsilon(\lambda) \geq \epsilon_0(\lambda)$ holds for infinitely many $\lambda$. By the property of the balanced AHF (Definition 2.4, Item 1), $\Pr[u \xleftarrow{R} \mathsf{AdmSample}(1^\lambda, Q(\lambda), \epsilon_0(\lambda)) : u \in \{0,1\}^m] = 1$ for all sufficiently large $\lambda$. Therefore, in the following, we assume that this condition always holds. We show the security of the scheme via the following sequence of games.

Game 0**:** This is the real single-key security experiment $\mathsf{Expt}^{\mathsf{cprf}}_{\mathsf{CPRF},\mathcal{F},\mathcal{A}}(\lambda)$ against an admissible adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Namely,

$\mathsf{coin} \xleftarrow{R} \{0,1\}$
$\mathsf{pp} \xleftarrow{R} \mathsf{Setup}(1^\lambda)$
$\mathsf{msk}_0 \xleftarrow{R} \mathsf{KeyGen}(\mathsf{pp})$
$X^* \xleftarrow{R} \mathcal{R}$
$(f, \mathsf{st}_\mathcal{A}) \xleftarrow{R} \mathcal{A}_1^{\mathcal{O}_{\mathsf{Chal}}(\cdot), \mathsf{Eval}(\mathsf{msk}_0, \cdot)}(\mathsf{pp})$
$\mathsf{sk}_f \xleftarrow{R} \mathsf{iO}(\mathsf{ConstrainedKey}[\mathsf{msk}, f])$
$\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2^{\mathcal{O}_{\mathsf{Chal}}(\cdot), \mathsf{Eval}(\mathsf{msk}_0, \cdot)}(\mathsf{sk}_f, \mathsf{st}_\mathcal{A})$
Return $(\widehat{\mathsf{coin}} \stackrel{?}{=} \mathsf{coin})$

where the challenge oracle $\mathcal{O}_{\mathsf{Chal}}(\cdot)$ is described below.

$\mathcal{O}_{\mathsf{Chal}}(x^*)$**:** Given $x^* \in \{0,1\}^n$ as input, it returns $\mathsf{Eval}(\mathsf{msk}_0, x^*)$ if $\mathsf{coin} = 1$ and $X^*$ if $\mathsf{coin} = 0$.

We recall that $\mathcal{O}_{\mathsf{Chal}}(\cdot)$ is queried at most once during the game.

Game 1**:** In this game, we change Game 0 so that the game performs the following additional step at the end of the experiment. First, the game samples $u \xleftarrow{R} \mathsf{AdmSample}(1^\lambda, Q, \epsilon_0)$ and checks whether the following condition holds:

$$P_u(h(x_1)) = \cdots = P_u(h(x_Q)) = 1 \ \wedge \ P_u(h(x^*)) = 0, \tag{3}$$

where $x_1, \ldots, x_Q$ are inputs to the PRF for which $\mathcal{A}$ called the evaluation oracle $\mathsf{Eval}(\mathsf{msk}_0, \cdot)$. If it does not hold, the game ignores the output $\widehat{\mathsf{coin}}$ of $\mathcal{A}$, and replace it with a fresh random coin $\widehat{\mathsf{coin}} \xleftarrow{R} \{0,1\}$. In this case, we say that the game aborts.

Game 2**:** In this game, we change the way $\mathsf{sk}_f$ is generated and the oracles return answers. At the beginning of the game, we sample $\mathsf{msk}_0 \xleftarrow{R} \mathsf{KeyGen}(\mathsf{pp})$ and $\mathsf{msk}_1 \xleftarrow{R} \mathsf{KeyGen}(\mathsf{pp})$, and compute $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m] \xleftarrow{R} \mathsf{PCPRF.Merge}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$. We then set $C := \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$. Note that $C$ is a circuit such that $C : \{0,1\}^n \to \{0,1\}$. Furthermore, $\mathsf{sk}_f$ given to $\mathcal{A}_2$ is generated as $\mathsf{sk}_f \xleftarrow{R} \mathsf{iO}(\mathsf{ConstrainedKeyAlt}[C, f])$ instead of $\mathsf{sk}_f \xleftarrow{R} \mathsf{iO}(\mathsf{ConstrainedKey}[\mathsf{msk}, f])$, where the circuit $\mathsf{ConstrainedKeyAlt}[C, f]$ is depicted in Figure 8. We also replace the evaluation oracle $\mathsf{Eval}(\mathsf{msk}_0, \cdot)$ and the challenge oracle $\widetilde{\mathcal{O}}_{\mathsf{Chal}}(\cdot)$ with the following oracles.

$\widetilde{\mathsf{Eval}}(C, \cdot)$**:** Given $x \in \{0,1\}^n$ as input, it returns $C(x)$.

$\widetilde{\mathcal{O}}_{\mathsf{Chal}}(C, \cdot)$**:** Given $x^*$ as input, it returns $C(x^*)$ if $\mathsf{coin} = 1$ and $X^*$ if $\mathsf{coin} = 0$.

| |
|---|
| $\mathsf{ConstrainedKeyAlt}[C, f]$ |
| Input: $x \in \{0,1\}^n$ |
| Constants: $\mathsf{pp}$, $C$, and $f$ |
| If $f(x) = 0$ |
|   Output $C(x)$ |
| Else |
|   Output $\perp$ |

Figure 8: Description of Program $\mathsf{ConstrainedKeyAlt}[C, f]$

| |
|---|
| $\widetilde{C}[\mathsf{sk}_{0,g}, \mathsf{msk}_1, f, u]$ |
| Input: $x \in \{0,1\}^n$ |
| Constants: $\mathsf{pp}$, $\mathsf{sk}_{0,g}$, $\mathsf{msk}_1$, $f$, $u$ |
| If $f(x) = 0 \ \wedge \ P_u(h(x)) = 0$ |
|   Output $\mathsf{CEval}(\mathsf{sk}_{0,g}, x)$ |
| If $f(x) = 0 \ \wedge \ P_u(h(x)) = 1$ |
|   Output $\mathsf{Eval}(\mathsf{msk}_1, x)$ |
| Else |
|   Output $\perp$ |

Figure 9: Description of Program $\widetilde{C}[\mathsf{sk}_{0,g}, \mathsf{msk}_1, f, u]$

Game 3**:** Recall that in Game 2, it is checked whether the abort condition Eq. (3) holds or not at the end of the game. In this game, we change the game so that it samples $u$ at the beginning of the game and aborts and outputs a random bit as soon as the abort condition becomes true.

Game 4**:** In this game, we further change the way $C$ is generated. At the beginning of the game, the game samples $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u] \xleftarrow{\mathsf{R}} \mathsf{PCPRF.Merge}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ and then set $C := \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ instead of $C := \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$.

Game 5**:** In this game, we replace $\widetilde{\mathsf{Eval}}(C, \cdot)$ and $\widetilde{\mathcal{O}}_{\mathsf{Chal}}(C, \cdot)$ with the following oracles.

$\mathsf{Eval}(\mathsf{msk}_1, \cdot)$**:** Given $x \in \{0,1\}^n$ as input, it returns $\mathsf{Eval}(\mathsf{msk}_1, x)$.

$\bar{\mathcal{O}}_{\mathsf{Chal}}(\mathsf{msk}_0, \cdot)$**:** Given $x^* \in \{0,1\}^n$ as input, it returns $\mathsf{Eval}(\mathsf{msk}_0, x^*)$ if $\mathsf{coin} = 1$ and $X^*$ if $\mathsf{coin} = 0$.

Game 6**:** In this game, we change the way $\mathsf{sk}_f$ is generated when $\mathcal{A}_1$ makes the call to $\mathcal{O}_{\mathsf{Chal}}$ (namely, the challenge query is made before $f$ is chosen by $\mathcal{A}$). Let $x^*$ be the challenge query made by $\mathcal{A}_1$. We set the function $g_{x^*} : \{0,1\}^n \to \{0,1\}$ as $g_{x^*}(x) = (x \stackrel{?}{=} x^*)$. To generate $\mathsf{sk}_f$, we first sample $\mathsf{sk}_{0,g_{x^*}} \xleftarrow{\mathsf{R}} \mathsf{PCPRF.Constrain}(\mathsf{msk}_0, g_{x^*})$ and set $sk_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\widetilde{C}[\mathsf{sk}_{0,g_{x^*}}, \mathsf{msk}_1, f, u])$, where $\widetilde{C}[\mathsf{sk}_{0,g}, \mathsf{msk}_1, f, u]$ is depicted in Figure 9. Note that if $\mathcal{A}_1$ does not make the challenge query, we do not change the way $\mathsf{sk}_f$ is generated.

Game 7**:** In this game, we change the way $\mathsf{sk}_f$ is generated when $\mathcal{A}_1$ stops without making challenge query (namely, the challenge query will be made after $\mathcal{A}$ chooses $f$). In such a case, we first sample $\mathsf{sk}_{0,f} \xleftarrow{\mathsf{R}} \mathsf{PCPRF.Constrain}(\mathsf{msk}_0, f)$ and set $sk_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\widetilde{C}[\mathsf{sk}_{0,f}, \mathsf{msk}_1, f, u])$.

Let $\mathsf{T}_i$ be the event that Game $i$ returns 1. We prove this by the following lemmas.

**Lemma 4.3.** *If $h$ is a balanced AHF and $|\Pr[\mathsf{T}_0] - 1/2|$ is non-negligible, so is $|\Pr[\mathsf{T}_1] - 1/2|$.*

*Proof.* We apply Lemma 2.6 to evaluate $|\Pr[\mathsf{T}_1] - 1/2|$. To apply the lemma, we set the input to $\mathcal{D}$ and $\mathcal{D}'$ to be coin used in the game, the output of $\mathcal{D}(\mathsf{coin})$ to be $(X = (x^*, x_1, \ldots, x_Q), \widehat{\mathsf{coin}})$ in Game 0, the output of $\mathcal{D}'(\mathsf{coin})$ to be that in Game 1, and $\gamma(X)$ to be the probability that Eq. (3) holds for $X = (x^*, x_1, \ldots, x_Q)$ when randomness is taken over the choice of $u \stackrel{\mathsf{R}}{\leftarrow} \mathsf{AdmSample}(1^\lambda, Q, \epsilon_0)$. Then,

$$\left|\Pr[\mathsf{T}_1] - \frac{1}{2}\right| \geq \gamma_{\min}\epsilon - \frac{\gamma_{\max} - \gamma_{\min}}{2} \geq \underbrace{\gamma_{\min}\epsilon_0 - \frac{\gamma_{\max} - \gamma_{\min}}{2}}_{:=\tau}$$

holds for infinitely many $\lambda$ by the lemma, where $\gamma_{\min} = \min_X \gamma(X)$ and $\gamma_{\max} := \max_X \gamma(X)$. By the property of the balanced AHF $h$ (Definition 2.4, Item 2), we have that $\tau$ is a noticeable function. This implies that $|\Pr[\mathsf{T}_1] - 1/2|$ is noticeable for infinitely many $\lambda$, and thus the term is non-negligible. ■

**Lemma 4.4.** *If* iO *is a secure indistinguishability obfuscator, then $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_1]| = \mathrm{negl}(\lambda)$.*

*Proof.* We first observe that $C(x) = \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m](x) = \mathsf{PCPRF.Eval}(\mathsf{msk}_0, x)$ since $P_{\perp^m}(h(x)) = 0$ for all $x \in \{0,1\}^n$. Therefore, the change for the evaluation and the challenge oracles is only conceptual. Furthermore, $\mathsf{ConstrainedKey}[\mathsf{msk}_0, f]$ and $\mathsf{ConstrainedKeyAlt}[C, f]$ with $C = \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ have identical functionalities. Therefore $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_1]|$ is negligible by the security of iO. ■

**Lemma 4.5.** *We have* $\Pr[\mathsf{T}_3] = \Pr[\mathsf{T}_2]$.

*Proof.* Since the change is only conceptual, the lemma trivially follows. ■

**Lemma 4.6.** *If* PCPRF *is partition-hiding with respect to $h$, we have $|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_3]| = \mathrm{negl}(\lambda)$.*

*Proof.* For the sake of the contradiction, let us assume that $|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_3]|$ is non-negligible. We then construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that breaks the partitioning hiding of PCPRF using $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. In the following, we differentiate two random variables coin and coin$'$. The former denotes the random variable that $\mathcal{A}$ tries to guess in the adaptive single key security game, while the latter denotes the random variable that $\mathcal{B}$ tries to guess in the partition-hiding game.

$\mathcal{B}_1(\mathsf{pp})$**:** Given pp, $\mathcal{B}_1$ first samples $u \stackrel{\mathsf{R}}{\leftarrow} \mathsf{AdmSample}(1^\lambda, Q, \epsilon_0)$, $\mathsf{coin} \stackrel{\mathsf{R}}{\leftarrow} \{0,1\}$, and $X^* \stackrel{\mathsf{R}}{\leftarrow} \mathbb{G}$. It then sets $\mathsf{st}_\mathcal{B} = (\mathsf{pp}, u, X^*, \mathsf{coin})$ and outputs $(u, \mathsf{st}_\mathcal{B})$.

$\mathcal{B}_2(\mathsf{k}, \mathsf{st}_\mathcal{B})$**:** Given k, $\mathcal{B}_2$ first parses $\mathsf{st}_\mathcal{B} \to (\mathsf{pp}, u, X^*, \mathsf{coin})$ and sets $C := \mathsf{k}$. Here, $\mathsf{k} \stackrel{\mathsf{R}}{\leftarrow} \mathsf{iO}[\mathsf{msk}_0, \mathsf{msk}_1, \perp^m]$ if $\mathsf{coin}' = 0$ and $\mathsf{k} \stackrel{\mathsf{R}}{\leftarrow} \mathsf{iO}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ if $\mathsf{coin}' = 1$. It then runs $\mathcal{A}_1$ on input pp. When $\mathcal{A}_1$ makes the query for $\widetilde{\mathcal{O}}_{\mathsf{Chal}}(C, \cdot)$ on input $x^*$, $\mathcal{B}_2$ first checks whether $P_u(h(x^*)) = 0$ holds. If it holds, $\mathcal{B}_2$ returns $C(x)$ if $\mathsf{coin} = 1$ and $X^*$ otherwise. If it does not hold, $\mathcal{B}_2$ stops the experiment and outputs a random coin as its guess. When $\mathcal{A}_1$ makes a query for $\widetilde{\mathsf{Eval}}(C, \cdot)$ on input $x$, $\mathcal{B}_2$ first checks whether $P_u(h(x)) = 1$ holds. If it holds, it returns $C(x)$. Otherwise, $\mathcal{B}_2$ stops the experiment and outputs a random coin as its guess. At some point, $\mathcal{A}_1$ outputs $(f, \mathsf{st}_\mathcal{A})$. $\mathcal{B}_2$ then computes $\mathsf{sk}_f \stackrel{\mathsf{R}}{\leftarrow} \mathsf{iO}(\mathsf{ConstrainedKeyAlt}[C, f])$ and gives $(\mathsf{sk}_f, \mathsf{st}_\mathcal{A})$ to $\mathcal{A}_2$. The oracle queries made by $\mathcal{A}_2$ are handled similarly to the case of $\mathcal{A}_1$. At last, $\mathcal{A}_2$ outputs $\widehat{\mathsf{coin}}$ as its guess for coin. Then, $\mathcal{B}_2$ outputs $(\widehat{\mathsf{coin}} \stackrel{?}{=} \mathsf{coin})$ as its guess.

It can easily be seen that $\mathcal{B}$ simulates $\mathsf{Game}_4$ if $\mathsf{coin}' = 1$ and $\mathsf{Game}_3$ otherwise. The lemma readily follows. ■

**Lemma 4.7.** *We have* $\Pr[\mathsf{T}_5] = \Pr[\mathsf{T}_4]$.

*Proof.* We observe that for any $x \in \{0,1\}^n$, we have $C(x) = \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u](x) = \mathsf{Eval}(\mathsf{msk}_b, x)$ for $b = P_u(h(x))$. By the change made in Game 3, the experiment stops and outputs a random bit as soon as $\mathcal{A}$ makes an evaluation query for $x$ such that $P_u(h(x)) = 0$. Therefore, from the view point of $\mathcal{A}$, the response of $\widetilde{\mathsf{Eval}}(C, \cdot)$ and $\mathsf{Eval}(\mathsf{msk}_1, \cdot)$ are completely the same. Similarly, since the experiment aborts as soon as $\mathcal{A}$ makes the challenge query for $x^*$ such that $P_u(h(x^*)) = 1$, the response of $\widetilde{\mathcal{O}}_{\mathsf{Chal}}(C, \cdot)$ and $\bar{\mathcal{O}}_{\mathsf{Chal}}(\mathsf{msk}_0, \cdot)$ are completely the same. Therefore, the change made in Game 5 is only conceptual and the lemma follows. ■

**Lemma 4.8.** *If* iO *is a secure indistinguishability obfuscator, then* $|\Pr[\mathsf{T}_6] - \Pr[\mathsf{T}_5]| = \mathrm{negl}(\lambda)$.

*Proof.* We first observe that $\mathsf{sk}_f$ is sampled as $\mathsf{sk}_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\mathsf{ConstrainedKeyAlt}[C, f])$ for $C = \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ in Game 5, whereas it is sampled as $sk_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\widetilde{C}[\mathsf{sk}_{0,g_{x^*}}, \mathsf{msk}_1, f, u])$ in Game 6 if $\mathcal{A}_1$ makes the challenge query. We prove that the programs $\mathsf{ConstrainedKeyAlt}[C, f]$ and $\widetilde{C}[\mathsf{sk}_{0,g_{x^*}}, \mathsf{msk}_1, f, u]$ are functionally equivalent. First, it is easy to see that both circuits output $\perp$ for an input $x$ such that $f(x) = 1$. For $x$ such that $f(x) = 0$ and $P_u(h(x)) = 1$, it is also easy to see that both circuits output $\mathsf{Eval}(\mathsf{msk}_1, x)$. In the case of $f(x) = 0$ and $P_u(h(x)) = 0$, $\mathsf{ConstrainedKeyAlt}[C, f]$ outputs $\mathsf{Eval}(\mathsf{msk}_0, x)$ whereas $\widetilde{C}[\mathsf{sk}_{0,g_{x^*}}, \mathsf{msk}_1, f, u]$ outputs $\mathsf{CEval}(\mathsf{sk}_{0,g_{x^*}}, x)$. Since $f(x^*) = 1$, we have $x \neq x^*$ and thus $g_{x^*}(x) = 0$. By the correctness of PCPRF, this implies $\mathsf{CEval}(\mathsf{sk}_{0,g_{x^*}}, x) = \mathsf{Eval}(\mathsf{msk}_0, x)$. Therefore, $|\Pr[\mathsf{T}_6] - \Pr[\mathsf{T}_5]|$ is negligible by the security of iO. ■

**Lemma 4.9.** *If* iO *is a secure indistinguishability obfuscator, then* $|\Pr[\mathsf{T}_7] - \Pr[\mathsf{T}_6]| = \mathrm{negl}(\lambda)$.

*Proof.* We first observe that $\mathsf{sk}_f$ is sampled as $\mathsf{sk}_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\mathsf{ConstrainedKeyAlt}[C, f])$ for $C = \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ in Game 6, whereas it is sampled as $sk_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\widetilde{C}[\mathsf{sk}_{0,f}, \mathsf{msk}_1, f, u])$ in Game 6 if $\mathcal{A}_1$ has not made the challenge query. We prove that $\mathsf{ConstrainedKeyAlt}[C, f]$ and $\widetilde{C}[\mathsf{sk}_{0,f}, \mathsf{msk}_1, f, u]$ are functionally equivalent. First, it is easy to see that both circuits output $\perp$ for an input $x$ such that $f(x) = 1$. For $x$ such that $f(x) = 0$ and $P_u(h(x)) = 1$, it is also easy to see that both output $\mathsf{Eval}(\mathsf{msk}_1, x)$. In the case of $f(x) = 0$ and $P_u(h(x)) = 0$, $\mathsf{ConstrainedKeyAlt}[C, f]$ outputs $\mathsf{Eval}(\mathsf{msk}_0, x)$ whereas $\widetilde{C}[\mathsf{sk}_{0,f}, \mathsf{msk}_1, f, u]$ outputs $\mathsf{CEval}(\mathsf{sk}_{0,f}, x)$. Since $f(x) = 0$, we have $\mathsf{CEval}(\mathsf{sk}_{0,f}, x) = \mathsf{Eval}(\mathsf{msk}_0, x)$ by the correctness of PCPRF. Therefore, $|\Pr[\mathsf{T}_7] - \Pr[\mathsf{T}_6]|$ is negligible by the security of iO. ■

**Lemma 4.10.** *If* PCPRF *is selective-constraint no-evaluation secure,* $|\Pr[\mathsf{T}_7] - 1/2| = \mathrm{negl}(\lambda)$.

*Proof.* For the sake of the contradiction, let us assume that $|\Pr[\mathsf{T}_7] - 1/2|$ is non-negligible. We then construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that breaks the selective-constraint no-evaluation security of PCPRF using $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. In the following, we denote the random coin that $\mathcal{B}$ should guess by coin. Looking ahead, the random coin that $\mathcal{A}$ has to guess in the simulation of Game 7 will be set as the same bit.

$\mathcal{B}_1(\mathsf{pp})$**:** Given pp, $\mathcal{B}_1$ first samples $u \xleftarrow{\mathsf{R}} \mathsf{AdmSample}(1^\lambda, Q, \epsilon_0)$, $\mathsf{msk}_1 \xleftarrow{\mathsf{R}} \mathsf{KeyGen}(\mathsf{pp})$. It then runs $\mathcal{A}_1$ on input pp. When $\mathcal{A}_1$ makes the evaluation query for $\mathsf{Eval}(\mathsf{msk}_1, \cdot)$ on input $x$, $\mathcal{B}_1$ first checks whether $P_u(h(x)) = 1$ holds. If it holds, it returns $\mathsf{Eval}(\mathsf{msk}_1, x)$. Otherwise, $\mathcal{B}_1$ sets $\mathsf{st}_{\mathcal{B}} = \mathtt{abort}$ and outputs $(g_{=1}, \mathsf{st}_{\mathcal{B}})$, where $g_{=1}$ is the constant function that always outputs 1. If $\mathcal{A}_1$ makes the challenge query for $x^*$, $\mathcal{B}_1$ first checks whether $P_u(h(x^*)) = 0$ holds. If it holds, $\mathcal{B}_1$ sets $\mathsf{st}_{\mathcal{B}} := (\mathtt{postchal}, \mathsf{msk}_1, u, x^*)$ and outputs $(g_{x^*}, \mathsf{st}_{\mathcal{B}})$. Otherwise, it sets $\mathsf{st}_{\mathcal{B}} = \mathtt{abort}$ and outputs $(g_{=1}, \mathsf{st}_{\mathcal{B}})$. If $\mathcal{A}_1$ stops and outputs $(f, \mathsf{st}_{\mathcal{A}})$ without having made the challenge query, $\mathcal{B}_1$ sets $\mathsf{st}_{\mathcal{B}} := (\mathtt{prechal}, \mathsf{msk}_1, u, \mathsf{st}_{\mathcal{A}})$ and outputs $(f, \mathsf{st}_{\mathcal{B}})$.

Given the output from $\mathcal{B}_1$, the selective-constraint no-evaluation security game for $\mathcal{B}$ runs $\mathsf{msk}_0 \xleftarrow{\mathsf{R}} \mathsf{KeyGen}(\mathsf{pp})$ and $\mathsf{sk} \xleftarrow{\mathsf{R}} \mathsf{Constrain}(\mathsf{msk}_0, g)$ where $g$ is either $f$ or $g_{=1}$ or $g_{x*}$. Then, $\mathsf{sk}$ is given to $\mathcal{B}_2$.

$\mathcal{B}_2(\mathsf{sk}, \mathsf{st}_\mathcal{B})$**:** It is given a constrained key $\mathsf{sk}$ for some function and the state $\mathsf{st}_\mathcal{B}$ as input. By the way we defined $\mathcal{B}_1$, these should be in the form of ($\mathsf{sk} = \mathsf{sk}_{g_{=1}}, \mathsf{st}_\mathcal{B} = \mathtt{abort}$) or ($\mathsf{sk} = \mathsf{sk}_{0,f}, \mathsf{st}_\mathcal{B} := (\mathtt{prechal}, \mathsf{msk}_1, u, \mathsf{st}_\mathcal{A})$) or ($\mathsf{sk} = \mathsf{sk}_{0,g_{x^*}}, \mathsf{st}_\mathcal{B} := (\mathtt{postchal}, \mathsf{msk}_1, u, x^*)$).

- In the case of ($\mathsf{sk} = \mathsf{sk}_{g_{=1}}, \mathsf{st}_\mathcal{B} = \mathtt{abort}$), $\mathcal{B}_2$ immediately aborts and outputs a random bit.
- In the case of ($\mathsf{sk} = \mathsf{sk}_{0,g_{x^*}}, \mathsf{st}_\mathcal{B} := (\mathtt{postchal}, \mathsf{msk}_1, u, x^*)$), $\mathcal{B}_2$ first makes the challenge query for its own challenge oracle on input $x^*$ and is given the challenge term, which is $X^*$ if $\mathsf{coin} = 0$ and $\mathsf{Eval}(\mathsf{msk}_0, x^*)$ if $\mathsf{coin} = 1$. It then gives the challenge term to $\mathcal{A}_1$. When $\mathcal{A}_1$ makes an evaluation query on input $x$, $\mathcal{B}_2$ first checks whether $P_u(h(x)) = 1$ holds. If it holds, it returns $\mathsf{Eval}(\mathsf{msk}_1, x)$. Otherwise, $\mathcal{B}_2$ aborts and outputs a random bit. At some point, $\mathcal{A}_1$ outputs $(f, \mathsf{st}_\mathcal{A})$. $\mathcal{B}_2$ then sets $\mathsf{sk}_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\widetilde{C}[\mathsf{sk}_{0,g_{x^*}}, \mathsf{msk}_1, f, u])$ and runs $\mathcal{A}_2(\mathsf{sk}_f, \mathsf{st}_\mathcal{A})$. The evaluation queries that $\mathcal{A}_2$ makes are handled as above. Eventually, $\mathcal{A}_2$ outputs its guess $\widehat{\mathsf{coin}}$. Then, $\mathcal{B}_2$ outputs $\widehat{\mathsf{coin}}$ as its guess.
- In the case of ($\mathsf{sk} = \mathsf{sk}_{0,f}, \mathsf{st}_\mathcal{B} := (\mathtt{prechal}, \mathsf{msk}_1, u, \mathsf{st}_\mathcal{A})$), $\mathcal{B}_2$ first computes $sk_f \xleftarrow{\mathsf{R}} \mathsf{iO}(\widetilde{C}[\mathsf{sk}_{0,f}, \mathsf{msk}_1, f, u])$ and runs $\mathcal{A}_2$ on input $(\mathsf{sk}_f, \mathsf{st}_\mathcal{A})$. When $\mathcal{A}_2$ makes an evaluation query for $x$, $\mathcal{B}_2$ first checks whether $P_u(h(x)) = 1$ holds. If it holds, it returns $\mathsf{Eval}(\mathsf{msk}_1, x)$. Otherwise, $\mathcal{B}_1$ aborts and outputs a random bit. When $\mathcal{A}_2$ makes the challenge query for $\mathcal{O}_{\mathsf{Chal}}(\mathsf{msk}_0, \cdot)$ on input $x^*$, it first checks whether $P_u(h(x^*)) = 0$ and aborts and outputs a random bit if it does not hold. On the other hand, if $P_u(h(x^*)) = 0$ holds, it then makes the challenge query for its own challenge oracle on input $x^*$ and is given the challenge term, which is $X^*$ or $\mathsf{Eval}(\mathsf{msk}_0)$ depending on the value of $\mathsf{coin}$. Then it gives the challenge term to $\mathcal{A}_2$. Eventually, $\mathcal{A}_2$ outputs its guess $\widehat{\mathsf{coin}}$. Then, $\mathcal{B}_2$ outputs $\widehat{\mathsf{coin}}$ as its guess.

It can be seen that $\mathcal{B}$ defined above perfectly simulates $\mathsf{Game}\ 7$ for $\mathcal{A}$, where $\mathsf{coin}$ in the game is set as the same bit as the random coin that $\mathcal{B}$ has to guess. Therefore, the lemma readily follows. ■

From Lemma 4.3, we have that $|\Pr[\mathsf{T}_1] - 1/2|$ is non-negligible. Then, by Lemma 4.4, 4.5, 4.6, 4.7, 4.8, and 4.9, we have that $|\Pr[\mathsf{T}_7] - 1/2|$ is non-negligible as well. However, this contradicts Lemma 4.10. This concludes the proof of the theorem. ■

*Remark* 4.11. As one may notice, in the hybrids, we obfuscate a program that contains a merged key $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ that itself is also an obfuscation of some program in our construction. Therefore when generating a constrained key, $\mathsf{ConstrainedKey}[\mathsf{msk}, f]$ should be padded to the maximum size of an obfuscated program that appears in the hybrids, and thus the size of $\mathsf{sk}_f$ is the size of an obfuscation of an obfuscation. Actually, this "obfuscation of obfuscation" blowup could be avoided if we directly construct an adaptively secure CPRF based on iO and the subgroup hiding assumption. However, we believe that the abstraction of PCPRF makes it easier to understand our security proof, and there should be further applications of it.

## Acknowledgments

We would like to thank Yilei Chen for the valuable discussion about adaptive security of the LWE-based constraint-hiding CPRFs. The first, second, and fourth authors were supported by JST CREST Grant Number JPMJCR1688, Japan.

## References


[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May / June 2010. (Cited on page 11.)

[AFP16] Hamza Abusalah, Georg Fuchsbauer, and Krzysztof Pietrzak. Constrained PRFs for unbounded inputs. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 413–428. Springer, Heidelberg, February / March 2016. (Cited on page 1, 2.)

[AMN+18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for $\mathrm{NC}^1$ in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Heidelberg, August 2018. (Cited on page 1, 2, 5, 7, 11, 12, 15, 17, 18.)

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 443–459. Springer, Heidelberg, August 2004. (Cited on page 2, 3, 10.)

[BFP+15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 31–60. Springer, Heidelberg, March 2015. (Cited on page 1.)

[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):6:1–6:48, 2012. (Cited on page 1, 13.)

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014. (Cited on page 1.)

[Bit17] Nir Bitansky. Verifiable random functions from non-interactive witness-indistinguishable proofs. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 567–594. Springer, Heidelberg, November 2017. (Cited on page 1.)

[BKM17] Dan Boneh, Sam Kim, and Hart William Montgomery. Private puncturable PRFs from standard lattice assumptions. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 415–445. Springer, Heidelberg, April / May 2017. (Cited on page 1.)

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 494–524. Springer, Heidelberg, March 2017. (Cited on page 1, 2, 7.)

[BR09] Mihir Bellare and Thomas Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 407–424. Springer, Heidelberg, April 2009. (Cited on page 10, 11.)

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Heidelberg, November 2017. (Cited on page 1, 7.)

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Heidelberg, March 2015. (Cited on page 1, 7.)

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013. (Cited on page 1, 12, 13.)

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Heidelberg, August 2014. (Cited on page 1, 2, 4, 6, 7.)

[CC17a] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC1 from LWE. Cryptology ePrint Archive, Report 2017/143, 2017. `http://eprint.iacr.org/2017/143`. (Cited on page 8.)

[CC17b] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for $NC^1$ from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 446–476. Springer, Heidelberg, April / May 2017. (Cited on page 1, 8.)

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. (Cited on page 1.)

[CH85] Stephen A. Cook and H. James Hoover. A depth-universal circuit. *SIAM J. Comput.*, 14(4):833–839, 1985. (Cited on page 15.)

[CHKP12] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology*, 25(4):601–639, October 2012. (Cited on page 3, 10.)

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015. (Cited on page 20.)

[DDM17] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Constrained pseudorandom functions for unconstrained inputs revisited: Achieving verifiability and key delegation. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 463–493. Springer, Heidelberg, March 2017. (Cited on page 2.)

[DKW16] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 124–153. Springer, Heidelberg, May 2016. (Cited on page 1, 2.)

[DN18] Alex Davidson and Ryo Nishimaki. A bit-fixing prf with o(1) collusion-resistance from lwe. *IACR Cryptology ePrint Archive*, 2018:890, 2018. (Cited on page 1.)

[FHPS13] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 513–530. Springer, Heidelberg, August 2013. (Cited on page 11.)

[FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained PRFs. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 82–101. Springer, Heidelberg, December 2014. (Cited on page 8.)

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 44–61. Springer, Heidelberg, May / June 2010. (Cited on page 7.)

[GGH$^+$16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016. (Cited on page 1, 13.)

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986. (Cited on page 8.)

[GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 537–566. Springer, Heidelberg, November 2017. (Cited on page 1, 7.)

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008. (Cited on page 11.)

[HHH$^+$14] Gottfried Herold, Julia Hesse, Dennis Hofheinz, Carla Ràfols, and Andy Rupp. Polynomial spaces: A new framework for composite-to-prime-order transformations. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 261–279. Springer, Heidelberg, August 2014. (Cited on page 7.)

[HKKW14] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. Cryptology ePrint Archive, Report 2014/720, 2014. `http://eprint.iacr.org/2014/720`. (Cited on page 1, 3.)

[HKW14] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. Cryptology ePrint Archive, Report 2014/521, 2014. `http://eprint.iacr.org/2014/521`. (Cited on page 9.)

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 79–102. Springer, Heidelberg, November / December 2015. (Cited on page 1, 2, 4, 7, 9, 18, 20.)

[Jag15] Tibor Jager. Verifiable random functions from weaker assumptions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 121–143. Springer, Heidelberg, March 2015. (Cited on page 3, 10, 11.)

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 669–684. ACM Press, November 2013. (Cited on page 1.)

[KY16] Shuichi Katsumata and Shota Yamada. Partitioning via non-linear polynomial functions: More compact IBEs from ideal lattices and bilinear maps. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 682–712. Springer, Heidelberg, December 2016. (Cited on page 11.)

[KY18] Shuichi Katsumata and Shota Yamada. Note on constructing constrained prfs from owfs with constant collusion resistance. *IACR Cryptology ePrint Archive*, 2018:914, 2018. (Cited on page 1.)

[Lew12] Allison B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 318–335. Springer, Heidelberg, April 2012. (Cited on page 7.)

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 455–479. Springer, Heidelberg, February 2010. (Cited on page 2.)

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 597–612. Springer, Heidelberg, August 2002. (Cited on page 11.)

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, 51(2):231–262, 2004. (Cited on page 4.)

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 675–701. Springer, Heidelberg, March 2018. (Cited on page 1, 7.)

[SC12] Jae Hong Seo and Jung Hee Cheon. Beyond the limitation of prime-order bilinear groups, and round optimal blind signatures. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 133–150. Springer, Heidelberg, March 2012. (Cited on page 7.)

[SS96] Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Trans. Information Theory*, 42(6):1710–1722, 1996. (Cited on page 11.)

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. (Cited on page 2, 4.)

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Heidelberg, May 2005. (Cited on page 2, 3.)

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 619–636. Springer, Heidelberg, August 2009. (Cited on page 2.)

[Yam16] Shota Yamada. Adaptively secure identity-based encryption from lattices with asymptotically shorter public parameters. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 32–62. Springer, Heidelberg, May 2016. (Cited on page 11.)

[Yam17] Shota Yamada. Asymptotically compact adaptively secure lattice IBEs and verifiable random functions via generalized partitioning techniques. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 161–193. Springer, Heidelberg, August 2017. (Cited on page 3, 10.)

[Zém01] Gilles Zémor. On expander codes. *IEEE Trans. Information Theory*, 47(2):835–837, 2001. (Cited on page 11.)