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

Nuttapong Attrapadung[1], Takahiro Matsuda[1], Ryo Nishimaki[2],

Shota Yamada[1], and 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
{ryo.nishimaki.zk,takashi.yamakawa.ga}@hco.ntt.co.jp

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.

1 Introduction

1.1 Background

Constrained pseudorandom function (CPRF) [11][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 sk_f. There are many applications of CPRFs including broadcast encryption [11], attribute-based encryption (ABE) [3], identity-based non-interactive key exchange [11], and policy-based key distribution [11].

[1] It is also known as delegatable PRF [38] and functional PRF [13].
[2] We note that the role of the constraining function f is "reversed" from the definition by Boneh and Waters [11], in the sense that the evaluation by a constrained key 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 [15].

Since the proposal of the concept of CPRF, there have been significant progresses in constructing CPRFs [11,38,13,12,4,15,24,1,7,31,10,9,17,14,42,3]. However, most known collusion-resistant[3] CPRFs (e.g., [11]) only satisfy 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., [15]), 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 choose a challenge input later.[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. In the random oracle model, Hofheinz, Kamath, Koppula, and Waters [33] constructed an adaptively secure collusion-resistant CPRF for all circuits without relying on complexity leveraging based on indistinguishability obfuscation (IO) [5,28], and Attrapadung et al. [3] constructed an adaptively single-key secure CPRF for $\mathbf{NC}^1$ on pairing-free groups. However, the random oracle model has been recognized to be problematic [18].

There are a few number of adaptively secure CPRFs in the standard model. Hohenberger, Koppula, and Waters [35] constructed an adaptively secure puncturable PRF based on IO and the subgroup hiding assumption on a composite order group.[5] Very recently, Davidson et al. [23] constructed an adaptively secure CPRF for bit-fixing functions secure against a constant number of collusion based on 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.

1.2 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.

[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).

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 [12,10,1,24,22].

1.3 Design Idea and Technical Overview

In this section, we give an overview of our design idea and technique.

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. [8,46,20,36,48]. 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* $u \in \{0,1,\bot\}^m$ where $\{0,1\}^n$ is the input space.[6] 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

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

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. [33] 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

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

space. Namely, if we merge msk_0 and 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 msk_0 for inputs x in the challenge space, and msk_1 for inputs x in the simulation space. We often call msk_0 a real master key, and 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, $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$, which works exactly the same as 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. Mmaster secret keys and PRF values of the CPRF is defined to be exactly the same as those of the underlying partitionable CPRF. The 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 [43,12]. 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 $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$ instead of msk_0 where msk_1 is a "shadow" master secret key that is independent from msk_0. This modification causes a negligible difference by the security of IO because $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$ works exactly the same as msk_0. Then we replace $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$ 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 msk_1 whereas a challenge value is computed by using the real secret key msk_0. Then we modify a constrained key 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 msk_0 by the constraining algorithm of the underlying partitionable CPRF, instead of msk_0. This modification causes a negligible difference by the security of IO since $\mathsf{sk}_f^{\mathsf{real}}$ and 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 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 [35]. 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. [35] can be seen as a construction of a partitionable CPRF for puncturing functions. Their construction is a variant of the Naor-Reingold PRF [41] 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_{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 $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$ and $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u]$ differ on exponentially many inputs. They overcome this problem by 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 the standard puncturing technique with IO [43,12]. 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 $\boldsymbol{NC^1}$. 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. [3]. The construction

of Attrapadung et al.'s CPRF F_{amnyy} (instantiated on a composite order group $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q$) is described as follows.

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

where $\mathsf{msk}^{\mathsf{amnyy}} = (\boldsymbol{b} \in \mathbb{Z}_N^z, \alpha \in \mathbb{Z}_N)$ 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_{amnyy} satisfies selective-constraint no-evaluation security under the L-DDHI assumption[7], which can be reduced to the subgroup hiding assumption (See Lemma 2.1). 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_{amnyy} and F_{hkw} to define F_{ours} as follows:

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

where x is an input, y_i is the i-th bit of $h(x)$, h is an admissible hash function, and $\mathsf{msk}^{\mathsf{ours}} = (\boldsymbol{b}, \alpha, \{s_{i,b}\}_{i\in[m],b\in\{0,1\}})$ is a master secret key. A constrained key for a predicate f consists of that of F_{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 $\mathsf{msk}^{\mathsf{amnyy}} := (\boldsymbol{b}, \alpha)$. By this equation, it is also easy to see that the selective-constraint no-evaluation security of F_{ours} can be reduced to that of F_{amnyy}. A merged key is an obfuscated circuit that computes $\mathsf{Eval}(\mathsf{msk}_{P_u(h(x))}, x)$ where $\mathsf{msk}_0 = (\boldsymbol{b}, \alpha, \{s_{i,b}\}_{i\in[m],b\in\{0,1\}})$ and $\mathsf{msk}_1 = (\hat{\boldsymbol{b}}, \hat{\alpha}, \{\hat{s}_{i,b}\}_{i\in[m],b\in\{0,1\}})$ 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 $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$ 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.

[7] 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 $\psi_1 \xleftarrow{\mathsf{R}} \mathbb{G}$.

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 2.1, 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 $s_{i,\eta} \coloneqq \beta s'_{i,\eta} \in \mathbb{Z}_N$ for all (i, η) such that $u_i = \perp$ or $\eta = u_i$, where $s'_{i,\eta}$ is a uniformly random and β 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^{β^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 $w^{(\prod_{i=1}^{m} s'_{i,y_i})\cdot U(\boldsymbol{b},x)/\alpha}$. 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 $(g^{\beta^r})^{(\prod_{i=1}^{m} s'_{i,y_i})\cdot U(\boldsymbol{b},x)/\alpha}$, where $r \coloneqq |\{i \in [m] \mid u_i = y_i\}| \leq m-1$. 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.[8]. Then, we can set $\mathsf{msk}_0 \coloneqq \{s'_{i,b} \bmod p\}_{i,b}$ and $\mathsf{msk}_1 \coloneqq \{s'_{i,b} \bmod q\}_{i,b}$. Since $w \in \mathbb{G}_p$ and $g^{\beta^j} \in \mathbb{G}_q$ where $j \in \{1, \ldots, m-1\}$, it holds that

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

and this change is indistinguishable due to the security of IO. Lastly, by the Chinese remainder theorem, msk_0 and msk_1 are independently and uniformly random (that is, 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

[8] 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.

adaptively secure CPRF explained in the paragraph of "Adaptively secure CPRF from partitionable CPRF".

1.4 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 [12]. In the CPRF based on IO, a constrained key is an obfuscated 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 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 an exponential number of hybrids to modify outputs on exponentially many inputs if we just use the security of IO. We overcome this issue by sophisticated use of the subgroup hiding assumption in a similar way to the work by Hohenberger et al. [35]. 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 [25,44,39,32].

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 ($\mathbf{P/poly}$) though there seems to be no obvious attack against our CPRF even if an adversary is given multiple constrained keys for constraints possibly outside $\mathbf{NC}^1$.[9] In fact, we can prove that our CPRF is collusion-resistant and supports $\mathbf{P/poly}$ in the selective-challenge setting by the puncturing technique similarly to [12]. However, in the security of adaptive security, we crucially rely on the selective-constraint security of the

[9] We note that even if the underlying partitionable CPRF only supports $\mathbf{NC}^1$, we can naturally define a constrained key for a function outside $\mathbf{NC}^1$ in the CPRF given in Section 4 because a function class supported by the partitionable CPRF matters only in the security proof and does not matter for the correctness.

underlying partitionable CPRF, which stems from the CPRF by Attrapadung et al. [3]. 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 [15,14,42] which satisfy selective-constraint single-key security and supports $\mathbf{P/poly}$. If we could use these CPRFs instead of Attrapadung et al.'s scheme, we would obtain adaptively single-key secure CPRFs for $\mathbf{P/poly}$. 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 [10] 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 [31]. 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.

1.5 Other Related Work

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

Fuchsbauer, Konstantinov, Pietrazk, and Rao [27] proved that the classical GGM PRF [30] 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 [16] 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 [17], they claimed security against adversaries that make an unbounded number of evaluation queries before a constraining query, but they retracted the claim [16, 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 definitions for complexity assumptions, tools, and cryptographic primitives.

2.1 Composite Order Group

In this paper, in a similar manner to Hohenberger et al. [35], we will use a group of composite order in which the subgroup hiding assumption holds. We recall it here.

Let GGen be a PPT algorithm (called the *group generator*) that takes a security parameter 1^λ 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 GGen is defined as follows:

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

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

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

For our purpose in this paper, it is convenient to introduce the following *L-DDH*[10] and *L-DDHI* assumptions with respect to GGen. These are not additional assumptions since they are implied by the subgroup hiding assumption.

Definition 2.2 (L-DDH & L-DDHI Assumptions). *Let* 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* 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:*

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

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

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

[10] The L-DDH assumption was called *Assumption 2* by Hohenberger et al. [34].

Lemma 2.1. *Let* GGen *be a group generator. If the subgroup hiding assumption holds with respect to* GGen*, then the L-DDH and L-DDHI assumptions hold with respect to* GGen *for all polynomials* $L = L(\lambda)$*.*

The proof of Lemma 2.1 can be found in the full version.

2.2 Balanced Admissible Hash Functions and Related Facts

Here, we describe the definition of a balanced admissible hash function (AHF) introduced by Jager [36]. A balanced AHF is an extension of an ordinary AHF [8,20], 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 [36] and corresponds to a special case of the general notion of "a partitioning function" introduced by Yamada [48].

Definition 2.3 ([36,48]). *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, δ) *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,* λ_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* $x^* \notin \{x_1, ..., x_Q\}$*,*

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

   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 2.1. The term $\tau(\lambda)$ defined above may appear very specific. However, as discussed by Jager [36], such a term appears typically in security analyses that follow the approach of Bellare and Ristenpart [6].

As shown by Jager [36], who extended previous works that gave simple constructions of AHF [40,26], 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 [45,49,29].

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. [3].

Syntax. Let $\mathcal{F} = \{\mathcal{F}_{\lambda,k}\}_{\lambda,k\in\mathbb{N}}$ be a class of functions[11] 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^λ as input, and outputs a public parameter pp,[12] where 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 pp as input, and outputs a master secret key $\mathsf{msk} \in \mathcal{K}$.

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

$\mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, f) \xrightarrow{R} \mathsf{sk}_f$**:** 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 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 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, $\mathsf{Constrain}$, 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.

[11] 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 λ and an input-length k.

[12] 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.

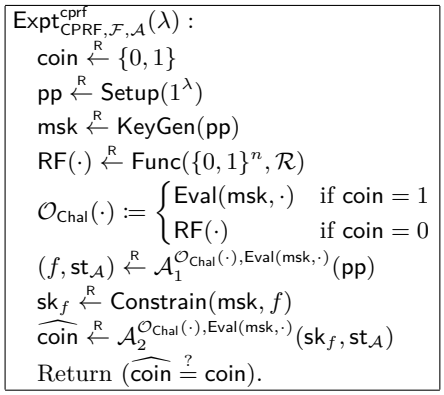

Fig. 1. The experiment for defining single-key security for a CPRF.

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 $\lambda \in \mathbb{N}$, $\mathsf{pp} \xleftarrow{R} \mathsf{Setup}(1^\lambda)$ (which specifies the input length $n = n(\lambda) = \mathrm{poly}(\lambda)$), $\mathsf{msk} \xleftarrow{R} \mathsf{KeyGen}(\mathsf{pp})$, functions $f \in \mathcal{F}_{\lambda,n}$, and inputs $x \in \{0,1\}^n$ satisfying $f(x) = 0$, we have $\mathsf{CEval}(\ \mathsf{Constrain}(\mathsf{msk}, f), x\ ) = \mathsf{Eval}(\mathsf{msk}, x)$. We stress that a constrained key sk_f can compute the PRF if $f(x) = 0$. (This treatment is reversed from the original definition by Boneh and Waters [11].)

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.*[13] 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 where $\mathsf{Func}(\{0,1\}^n, \mathcal{R})$ denotes the set of all functions from $\{0,1\}^n$ to $\mathcal{R}$.

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,

[13] selective-constraint no-evaluation security was simply called no-evaluation security in [3].

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.
- 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 2.4 (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 2.2. As noted by Boneh and Waters [11], 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) [5,28].

Definition 2.5 (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^λ *and a circuit* C *as input, and outputs an obfuscated circuit* $\widehat{C}$ *that computes the same function as* C*. (We may drop* 1^λ *from an input to* iO *when* λ *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:*

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

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 (Setup, KeyGen, Eval, $\mathsf{Constrain}$, CEval, Merge, MEval) where (Setup, KeyGen, Eval, $\mathsf{Constrain}$, CEval) forms a CPRF for $\mathcal{F}$. Two additional algorithms ,Merge and 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 $\lambda \in \mathbb{N}$, $\mathsf{pp} \xleftarrow{R} \mathsf{Setup}(1^\lambda)$ (which specifies the input length $n = n(\lambda) = \mathrm{poly}(\lambda)$), $\mathsf{msk}_0, \mathsf{msk}_1 \xleftarrow{R} \mathsf{KeyGen}(\mathsf{pp})$, $u \in \{0, 1, \perp\}^m$, $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, u] \xleftarrow{R} \mathsf{Merge}(\mathsf{msk}_0, \mathsf{msk}_1, u)$ 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 2.3.

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 (Setup, KeyGen, Eval, $\mathsf{Constrain}$, CEval) is selective-constraint no-evaluation secure as a CPRF.[14]

[14] 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:

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

We note that k_0 generated by $\mathsf{Merge}(\mathsf{msk}_0, \mathsf{msk}_1, \bot^m)$ works completely identically to msk_0, albeit in the sense that $\mathsf{MEval}(\mathsf{k}_0, x) = \mathsf{Eval}(\mathsf{msk}_0, x)$. This is since we have $P_{\bot^m}(h(x)) = 0$ 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 [3], the following two paragraphs are taken from [3].

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 ℓ-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. *([21,3]) 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.[15] 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 $\mathcal{G} \xleftarrow{R} \mathsf{GGen}(1^\lambda)$. It then outputs the public parameter $\mathsf{pp} := (N, \mathbb{G})$.

$\mathsf{KeyGen}(\mathsf{pp})$**:** It chooses $g \xleftarrow{R} \mathbb{G}$, $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \xleftarrow{R} \mathbb{Z}_N^2$, and $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, $\alpha \xleftarrow{R} \mathbb{Z}_N^*$.[16] 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

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

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

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

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

[15] 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.

[16] 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.

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

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

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

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

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

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

Output $\mathsf{Eval}(\mathsf{msk}_{P_u(h(x))}, x)$

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

The proof of correctness of PCPRF can be found in the full version.

Theorem 3.1. *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 3.1 in this section.

Proof sketch of Theorem 3.1. 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 [3], and the partition-hiding is proven similarly to [35]. The selective-constraint no-evaluation security of PCPRF can be reduced to the $(D-1)$-DDHI assumption, which in turn follows from the subgroup hiding assumption similarly to the security proof of the no-evaluation secure CPRF of [3]. Therefore we omit it here, and the proof for this part can be found in the full version. In the following, we give a proof sketch for the partition-hiding.

We want to prove that k generated by $\mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m])$ and generated by $\mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u])$ are computationally indistinguishable. The difficulty is that $\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$ 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 iO to conclude it.[17] Actually, this can be proven by using the subgroup hiding assumption in a sophisticated way as in the work by Hohenberger, Koppula and Waters [35]. 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. In the following, T_i denotes the event that Game i returns 1.

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

[17] If one relies on the technique of "exponential number of hybrids" (e.g., [19]), 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.

$\mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$
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 $y \coloneqq h(x)$
Output $g^{\prod_{i=1}^{m} s_{i,y_i} \cdot U((b_1,\ldots,b_z),(x_1,\ldots,x_n))/\alpha}$.

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

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 $g \xleftarrow{R} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, and $\alpha \xleftarrow{R} \mathbb{Z}_N^*$. Then choose $(s_{1,0}, s_{1,1}), ...,$ $(s_{m,0}, s_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$. Set $\mathsf{msk}_0 \coloneqq (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$. Choose $\widehat{g} \xleftarrow{R} \mathbb{G}$, $(\widehat{b}_1, ..., \widehat{b}_z) \xleftarrow{R} \mathbb{Z}_N^z$ and $\widehat{\alpha} \xleftarrow{R} \mathbb{Z}_N^*$. Then choose $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ...,$ $(\widehat{s}_{m,0}, \widehat{s}_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$. Set $\mathsf{msk}_1 \coloneqq (\widehat{g}, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m])$
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_\mathcal{A}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

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

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

We have $|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_0]| = \mathrm{negl}(\lambda)$ by the security of iO.

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

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 $g \xleftarrow{R} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, 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
   $$s_{i,\eta} \coloneqq \begin{cases} \beta \cdot s'_{i,\eta} & \text{If } u_i = \bot \vee \eta = u_i \\ s'_{i,\eta} & \text{Otherwise} \end{cases}.$$
   Set $\mathsf{msk}_0 \coloneqq (g, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{MergedKey\text{-}Zero}[\mathsf{msk}_0]$
5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_\mathcal{A}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

We have $\Pr[\mathsf{T}_2] = \Pr[\mathsf{T}_1]$ since $\{s_{i,\eta}\}_{i\in[m],\eta\in\{0,1\}}$ is uniformly distributed in $\mathbb{Z}_N^{2m}$ in both games.

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

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

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

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

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

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

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

Compute $y \coloneqq h(x)$

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

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

Fig. 4. Description of Program $\mathsf{MergedKey\text{-}Zero}'[\mathsf{msk}'_0, u, v_0, ..., v_{m-1}, w]$

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_\mathcal{A}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{R} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, 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 $v_j \coloneqq g^{\beta^j}$ for $j \in \{0, ..., m-1\}$ and $w \coloneqq g^{\beta^m}$.
   Set $\mathsf{msk}'_0 \coloneqq ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$.
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 $\widehat{\mathsf{coin}}$.

We have $|\Pr[\mathsf{T}_3] - \Pr[\mathsf{T}_2]| = \mathrm{negl}(\lambda)$ by the security of iO.

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

1. Let $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} := (N, \mathbb{G})$.
2. Compute $(u, \mathsf{st}_\mathcal{A}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
3. Choose $g \xleftarrow{R} \mathbb{G}$, $(b_1, ..., b_z) \xleftarrow{R} \mathbb{Z}_N^z$, 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 $v_j \coloneqq g^{\beta^j}$ for $j \in \{0, ..., m-1\}$. Choose $w \xleftarrow{R} \mathbb{G}$.
   Set $\mathsf{msk}'_0 \coloneqq ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$.
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 $\widehat{\mathsf{coin}}$.

We have $|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_3]| = \mathrm{negl}(\lambda)$ by the $(m-1)$-DDH assumption.

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 $(s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.

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

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

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

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 $\widehat{\mathsf{coin}}$.

We have $|\Pr[\mathsf{T}_5] - \Pr[\mathsf{T}_4]| = \mathrm{negl}(\lambda)$ by the subgroup hiding assumption.

Game 6: In this game, we set 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 $\widehat{\mathsf{coin}}$.

We have $|\Pr[\mathsf{T}_6] - \Pr[\mathsf{T}_5]| = \mathrm{negl}(\lambda)$ by the security of iO.

Game 7: In this game, we modify how to generate $s'_{i,\eta,q}$, $b_{i,q}$ and α_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 $(u, \mathsf{st}_{\mathcal{A}}) \xleftarrow{R} \mathcal{A}_1(\mathsf{pp})$.
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 $s'_{i,\eta,p} \coloneqq s'_{i,\eta} \mod p$ and $s'_{i,\eta,q} \coloneqq \widehat{s}_{i,\eta} \mod q$ for $i \in [m]$, $\eta \in \{0,1\}$.
Set $b_{i,p} \coloneqq b_i \mod p$ and $b_{i,q} \coloneqq \widehat{b}_i \mod q$ for $i \in [m]$.
Set $\alpha_p \coloneqq \alpha \mod p$ and $\alpha_q \coloneqq \widehat{\alpha} \mod q$.
Set $v_j \coloneqq g^{\beta^j}$ for $j \in \{0, ..., m-1\}$. Choose $w \xleftarrow{R} \mathbb{G}_p$.
Set $\mathsf{msk}'_0 \coloneqq ((s'_{1,0,p}, s'_{1,1,p}), ..., (s'_{m,0,p}, s'_{m,1,p}), b_{1,p}, \ldots, b_{z,p}, \alpha_p)$.
Set $\mathsf{msk}'_1 \coloneqq ((s'_{1,0,q}, s'_{1,1,q}), ..., (s'_{m,0,q}, s'_{m,1,q}), b_{1,q}, \ldots, b_{z,q}, \alpha_q)$.

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 $\widehat{\mathsf{coin}}$.

We have $\Pr[\mathsf{T}_7] = \Pr[\mathsf{T}_6]$ by the Chinese remainder theorem.

Game 8: In this game, we modify the way to set msk'_0 and 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 $v_j \coloneqq g^{\beta^j}$ for $j \in \{0, ..., m-1\}$. Choose $w \xleftarrow{R} \mathbb{G}_p$.
Set $\mathsf{msk}'_0 \coloneqq ((s'_{1,0}, s'_{1,1}), ..., (s'_{m,0}, s'_{m,1}), b_1, \ldots, b_z, \alpha)$.
Set $\mathsf{msk}'_1 \coloneqq ((\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
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 $\widehat{\mathsf{coin}}$.

We have $|\Pr[\mathsf{T}_8] - \Pr[\mathsf{T}_7]| = \mathrm{negl}(\lambda)$ by the security of iO.

Game 9: In this game, we set 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 $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R} \mathsf{GGen}(1^\lambda)$, 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

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

Choose $(\widehat{b}_1, ..., \widehat{b}_z) \xleftarrow{R} \mathbb{Z}_N^z$, $\widehat{\alpha} \xleftarrow{R} \mathbb{Z}_N^*$, and $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
Choose $w \xleftarrow{R} \mathbb{G}_p$.
Set $\mathsf{msk}_0 \coloneqq (w, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
Set $\mathsf{msk}_1 \coloneqq (g, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u])$.
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: $\mathsf{pp} = (N, \mathbb{G})$

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

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

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

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

Compute $y \coloneqq h(x)$

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

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

We have $|\Pr[\mathsf{T}_9] - \Pr[\mathsf{T}_8]| = \mathrm{negl}(\lambda)$ by the security of iO.

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

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

We have $\Pr[\mathsf{T}_{10}] = \Pr[\mathsf{T}_9]$ since $\{s_{i,\eta}\}_{i\in[m],\eta\in\{0,1\}}$ is uniformly distributed in $\mathbb{Z}_N^{2m}$ in both games.

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 $\mathcal{G} = (N, p, q, \mathbb{G}, \mathbb{G}_p, \mathbb{G}_q, g_1, g_2) \xleftarrow{R} \mathsf{GGen}(1^\lambda)$, Set $\mathsf{pp} \coloneqq (N, \mathbb{G})$.
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 $(s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Choose $(\widehat{b}_1, ..., \widehat{b}_z) \xleftarrow{R} \mathbb{Z}_N^z$, $\widehat{\alpha} \xleftarrow{R} \mathbb{Z}_N^*$, and $(\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}) \xleftarrow{R} \mathbb{Z}_N^{2m}$.
   Choose $w \xleftarrow{R} \mathbb{G}$.
   Set $\mathsf{msk}_0 \coloneqq (w, (s_{1,0}, s_{1,1}), ..., (s_{m,0}, s_{m,1}), b_1, \ldots, b_z, \alpha)$.
   Set $\mathsf{msk}_1 \coloneqq (g, (\widehat{s}_{1,0}, \widehat{s}_{1,1}), ..., (\widehat{s}_{m,0}, \widehat{s}_{m,1}), \widehat{b}_1, \ldots, \widehat{b}_z, \widehat{\alpha})$.
4. Compute $\mathsf{k} \xleftarrow{R} \mathsf{iO}(\mathsf{MergedKey}[\mathsf{msk}_0, \mathsf{msk}_1, u])$.

5. Compute $\widehat{\mathsf{coin}} \xleftarrow{R} \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{k})$. The game returns $\widehat{\mathsf{coin}}$.

We have $|\Pr[\mathsf{T}_{11}] - \Pr[\mathsf{T}_{10}]| = \mathrm{negl}(\lambda)$ by the subgrup hiding assumption.

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

We have $\Pr[\mathsf{T}_{12}] = \Pr[\mathsf{T}_{11}]$ since we just renamed g and w by $\widehat{g}$ and g. This game corresponds to the case of $\mathsf{coin} = 1$ in the experiment defining the partition-hiding.

Game 0 and Game 12 correspond to the cases of $\mathsf{coin} = 0$ and $\mathsf{coin} = 1$ in the experiment defining the partition-hiding, and we proved $|\Pr[\mathsf{T}_{12}] - \Pr[\mathsf{T}_0]| = \mathrm{negl}(\lambda)$. This completes the proof of the constraint-hiding. More detailed analysis of the above sequence of games can be found in the full version.

This completes the proof of Theorem 3.1. ■

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 $\mathsf{sk}_f \xleftarrow{R} \mathsf{iO}(\mathsf{ConstrainedKey}[\mathsf{msk}, f])$ where $\mathsf{ConstrainedKey}[\mathsf{msk}, f]$ is a program described in Figure 7.

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

Fig. 7. Description of Program $\mathsf{ConstrainedKey}[\mathsf{msk}, f]$

$\mathsf{CEval}'(\mathsf{sk}_f, x)$**:** 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 4.1.

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 3.1 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 $\mathcal{F}^{\boldsymbol{NC^1}}$, which is defined in Section 3.*

Proof. 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 λ. By the property of the balanced AHF (Definition 2.3, 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 λ. Therefore, in the following, we assume that this condition always holds. We show the security of the scheme via the following sequence of games. In the following, T_i denotes the event that $\mathsf{Game}\ i$ returns 1, and we denote the master secret key of the scheme by msk_0 for notational convenience.

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:

$$P_u(h(x_1)) = \cdots = P_u(h(x_Q)) = 1 \ \wedge \ P_u(h(x^*)) = 0, \tag{2}$$

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.

By using the property of AHF, we can prove that the probability that the game does not abort is noticeable. More precisely, if $|\Pr[\mathsf{T}_0] - 1/2|$ is non-negligible, so is $|\Pr[\mathsf{T}_1] - 1/2|$ (See the full version for details).

Game 2: In this game, we change the way 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 $\mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m] \xleftarrow{R} \mathsf{PCPRF.Merge}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$. We then set $C := \mathsf{k}[\mathsf{msk}_0, \mathsf{msk}_1, \bot^m]$. Note that C is a circuit such that $C : \{0,1\}^n \to \{0,1\}$. Furthermore, 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 $\mathsf{coin} = 0$.

We have $|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_1]| = \mathrm{negl}(\lambda)$ by the security of iO.

Game 3: Recall that in Game 2, it is checked whether the abort condition Eq. (2) 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.

We have $\Pr[\mathsf{T}_3] = \Pr[\mathsf{T}_2]$ since the change is conceptual and nothing is changed from the adversary's view.

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

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

Fig. 8. Description of Program $\mathsf{ConstrainedKeyAlt}[C, f]$

We have $|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_3]| = \mathrm{negl}(\lambda)$ by the partition-hiding property of PCPRF.

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 $\mathsf{coin} = 0$.

We have $\Pr[\mathsf{T}_5] = \Pr[\mathsf{T}_4]$ since as soon as $\mathcal{A}$ makes an evaluation or challenge query that makes a difference for the response by the oracles, these games abort.

Game 6: In this game, we change the way 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 sk_f, we first sample $\mathsf{sk}_{0,g_{x^*}} \xleftarrow{R} \mathsf{PCPRF.Constrain}(\mathsf{msk}_0, g_{x^*})$ and set $sk_f \xleftarrow{R} \mathsf{iO}(\widetilde{C}[\mathsf{sk}_{0,g_{x^*}}, \mathsf{msk}_1, f, u])$, 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 sk_f is generated.

We have $|\Pr[\mathsf{T}_6] - \Pr[\mathsf{T}_5]| = \mathrm{negl}(\lambda)$ by the security of iO.

Game 7: In this game, we change the way 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 $\mathsf{sk}_{0,f} \xleftarrow{R} \mathsf{PCPRF.Constrain}(\mathsf{msk}_0, f)$ and set $sk_f \xleftarrow{R} \mathsf{iO}(\widetilde{C}[\mathsf{sk}_{0,f}, \mathsf{msk}_1, f, u])$.

We have $|\Pr[\mathsf{T}_7] - \Pr[\mathsf{T}_6]| = \mathrm{negl}(\lambda)$ by the security of iO.

Finally, we observe that we have $|\Pr[\mathsf{T}_7] - 1/2| = \mathrm{negl}(\lambda)$ by the selective-constraint no-evaluation security of PCPRF. The above completes the proof of Theorem 4.1. More detailed analysis of the above sequence of games can be found in the full version. ■

Remark 4.1. 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 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

$\widetilde{C}[\mathsf{sk}_{0,g}, \mathsf{msk}_1, f, u]$

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

Constants: pp, $\mathsf{sk}_{0,g}$, 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 $\perp$

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

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

1. H. Abusalah, G. Fuchsbauer, and K. Pietrzak. Constrained PRFs for unbounded inputs. In *CT-RSA 2016*, pages 413–428. 2016. 2, 3
2. S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT 2010*, pages 553–572. 2010.
3. N. Attrapadung, T. Matsuda, R. Nishimaki, S. Yamada, and T. Yamakawa. Constrained PRFs for NC^1 in traditional groups. In *CRYPTO 2018, Part II*, pages 543–574. 2018. 1, 2, 5, 9, 12, 13, 16, 18
4. A. Banerjee, G. Fuchsbauer, C. Peikert, K. Pietrzak, and S. Stevens. Key-homomorphic constrained pseudorandom functions. In *TCC 2015, Part II*, pages 31–60. 2015. 2
5. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012. 2, 14
6. M. Bellare and T. Ristenpart. Simulation without the artificial abort: Simplified proof and improved concrete security for Waters' IBE scheme. In *EUROCRYPT 2009*, pages 407–424. 2009. 12
7. N. Bitansky. Verifiable random functions from non-interactive witness-indistinguishable proofs. In *TCC 2017, Part II*, pages 567–594. 2017. 2
8. D. Boneh and X. Boyen. Secure identity based encryption without random oracles. In *CRYPTO 2004*, pages 443–459. 2004. 3, 11
9. D. Boneh, S. Kim, and H. W. Montgomery. Private puncturable PRFs from standard lattice assumptions. In *EUROCRYPT 2017, Part I*, pages 415–445. 2017. 2

10. D. Boneh, K. Lewi, and D. J. Wu. Constraining pseudorandom functions privately. In *PKC 2017, Part II*, pages 494–524. 2017. 2, 3, 9
11. D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT 2013, Part II*, pages 280–300. 2013. 1, 2, 13, 14
12. D. Boneh and M. Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO 2014, Part I*, pages 480–499. 2014. 2, 3, 4, 5, 8
13. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*, pages 501–519. 2014. 1, 2
14. Z. Brakerski, R. Tsabary, V. Vaikuntanathan, and H. Wee. Private constrained PRFs (and more) from LWE. In *TCC 2017, Part I*, pages 264–302. 2017. 2, 9
15. Z. Brakerski and V. Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC 2015, Part II*, pages 1–30. 2015. 1, 2, 9
16. R. Canetti and Y. Chen. Constraint-hiding constrained PRFs for NC1 from LWE. Cryptology ePrint Archive, Report 2017/143, 2017. 9
17. R. Canetti and Y. Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In *EUROCRYPT 2017, Part I*, pages 446–476. 2017. 2, 9
18. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. 2
19. R. Canetti, H. Lin, S. Tessaro, and V. Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *TCC 2015, Part II*, pages 468–497. 2015. 18
20. D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology*, 25(4):601–639, Oct. 2012. 3, 11
21. S. A. Cook and H. J. Hoover. A depth-universal circuit. *SIAM J. Comput.*, 14(4):833–839, 1985. 16
22. P. Datta, R. Dutta, and S. Mukhopadhyay. Constrained pseudorandom functions for unconstrained inputs revisited: Achieving verifiability and key delegation. In *PKC 2017, Part II*, pages 463–493. 2017. 3
23. A. Davidson, S. Katsumata, R. Nishimaki, and S. Yamada. Constrained PRFs for Bit-fixing from OWFs with Constant Collusion Resistance *IACR Cryptology ePrint Archive*, 2018:982, 2018. 2
24. A. Deshpande, V. Koppula, and B. Waters. Constrained pseudorandom functions for unconstrained inputs. In *EUROCRYPT 2016, Part II*, pages 124–153. 2016. 2, 3
25. D. M. Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *EUROCRYPT 2010*, pages 44–61. 2010. 8
26. E. S. V. Freire, D. Hofheinz, K. G. Paterson, and C. Striecks. Programmable hash functions in the multilinear setting. In *CRYPTO 2013, Part I*, pages 513–530. 2013. 12
27. G. Fuchsbauer, M. Konstantinov, K. Pietrzak, and V. Rao. Adaptive security of constrained PRFs. In *ASIACRYPT 2014, Part II*, pages 82–101 2014. 9
28. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016. 2, 14
29. O. Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008. 12
30. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, Oct. 1986. 9

31. R. Goyal, S. Hohenberger, V. Koppula, and B. Waters. A generic approach to constructing and proving verifiable random functions. In *TCC 2017, Part II*, pages 537–566. 2017. 2, 9
32. G. Herold, J. Hesse, D. Hofheinz, C. Ràfols, and A. Rupp. Polynomial spaces: A new framework for composite-to-prime-order transformations. In *CRYPTO 2014, Part I*, pages 261–279. 2014. 8
33. D. Hofheinz, A. Kamath, V. Koppula, and B. Waters. Adaptively secure constrained pseudorandom functions. In *FC 2019* (to appear), 2019. 2, 3
34. S. Hohenberger, V. Koppula, and B. Waters. Adaptively secure puncturable pseudorandom functions in the standard model. Cryptology ePrint Archive, Report 2014/521, 2014. 10
35. S. Hohenberger, V. Koppula, and B. Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In *ASIACRYPT 2015, Part I*, pages 79–102. 2015. 2, 5, 8, 10, 18
36. T. Jager. Verifiable random functions from weaker assumptions. In *TCC 2015, Part II*, pages 121–143. 2015. 3, 11, 12
37. S. Katsumata and S. Yamada. Partitioning via non-linear polynomial functions: More compact IBEs from ideal lattices and bilinear maps. In *ASIACRYPT 2016, Part II*, pages 682–712. 2016.
38. A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS 13*, pages 669–684. 2013. 1, 2
39. A. B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In *EUROCRYPT 2012*, pages 318–335. 2012. 8
40. A. Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *CRYPTO 2002*, pages 597–612. 2002. 12
41. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, 51(2):231–262, 2004. 5
42. C. Peikert and S. Shiehian. Privately constraining and programming PRFs, the LWE way. In *PKC 2018, Part II*, pages 675–701. 2018. 2, 9
43. A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *46th ACM STOC*, pages 475–484. 2014. 4, 5
44. J. H. Seo and J. H. Cheon. Beyond the limitation of prime-order bilinear groups, and round optimal blind signatures. In *TCC 2012*, pages 133–150. 2012. 8
45. M. Sipser and D. A. Spielman. Expander codes. *IEEE Trans. Information Theory*, 42(6):1710–1722, 1996. 12
46. B. R. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT 2005*, pages 114–127. 2005. 3
47. S. Yamada. Adaptively secure identity-based encryption from lattices with asymptotically shorter public parameters. In *EUROCRYPT 2016, Part II*, pages 32–62. 2016.
48. S. Yamada. Asymptotically compact adaptively secure lattice IBEs and verifiable random functions via generalized partitioning techniques. In *CRYPTO 2017, Part III*, pages 161–193. 2017. 3, 11
49. G. Zémor. On expander codes. *IEEE Trans. Information Theory*, 47(2):835–837, 2001. 12