Note on Constructing Constrained PRFs from OWFs with Constant Collusion Resistance

Shuichi Katsumata *  Shota Yamada †

September 26, 2018

Abstract

Constrained pseudorandom functions (CPRFs) are a type of PRFs that allows one to derive a constrained key $K_C$ from the master key $K$. While the master key $K$ allows one to evaluate on any input as a standard PRF, the constrained key $K_C$ only allows one to evaluate on inputs $x$ such that $C(x) = 1$. Since the introduction of CPRFs by Boneh and Waters (ASIACRYPT’13), Kiayias et al. (CCS’13), and Boyle et al. (PKC’14), there have been various constructions of CPRFs. However, thus far, almost all constructions (from standard assumptions and non-trivial constraints) are only proven to be secure if at most one constrained key $K_C$ is known to the adversary, excluding the very recent work of Davidson and Nishimaki (EPRINT’18). Considering the interesting applications of CPRFs such as ID-based non-interactive key exchange, we desire CPRFs that are collusion resistance with respect to the constrained keys. In this work, we make progress in this direction and construct a CPRF for the bit-fixing predicates that are collusion resistance for a constant number of constrained keys. Surprisingly, compared to the heavy machinery that was used by previous CPRF constructions, our construction only relies on the existence of one-way functions.

1 Introduction

Constrained pseudorandom functions (CPRFs), introduced simultaneously by Boneh and Waters [BW13], Kiayias et al. [KPTZ13], and Boyle et al. [BG114], are a generalization of standard pseudorandom functions (PRF) [GGM84] which allows one to derive a constrained key that can be used to evaluate the PRF on a subset of the inputs satisfied by the constraint. So far there have been numerous constructions of CPRFs from various assumptions [BW13, HKKW14, BV15, BLW17, CC17, BTVW17, PS18, CVW18, AMN+18, DN18]. However, beside the CPRFs for trivial constraints such as constraints represented by singleton sets: $F = \{x\mid x \in \{0,1\}^n\}$ or prefix-fixing predicates which are satisfied by the classical Goldreich-Goldwasser-Micali PRF [GGM84], much is left to be desired. For instance, the constructions that satisfy full collusion-resistance of the constrained keys, i.e., pseudorandomness can be argued even when the adversary obtains polynomially many constrained keys that are constrained on the challenge points, all rely on the random oracle model or assume the existence of indistinguishable obfuscation or multi-linear maps [BW13, HKKW14, BLW17]. Very recently, [DN18] made great progress in this direction and showed a CPRF construction for the bit-fixing predicate that achieves constant key

*The University of Tokyo, National Institute of Advanced Industrial Science and Technology (AIST). E-mail: shuichi_katsumata@it.k.u-tokyo.ac.jp
†National Institute of Advanced Industrial Science and Technology (AIST). E-mail: yamada-shota@aist.go.jp
collusion-resistance based on standard lattice-based assumptions in the standard model. Considering some of the interesting applications of CPRFs such as non-interactive key exchange [BLW17], it is desirable to have CPRFs that are secure even if the adversary is allowed to obtain multiple constrained keys. Moreover, other than [HKKW14, DN18], all constructions are only proven to be selectively-secure, where the adversary must commit to the constraint of the constrained key it receives from the challenger at the beginning of the game.

In this paper, we provide an adaptive and collusion-resistant CPRF for the bit-fixing predicate, where the scheme can tolerate up to any constant $Q$-collusions of constrained keys. Our construction achieves the same security requirement as [DN18]. Compared to previous CPRFs that required heavy machinery, our construction is very simple and can be proven solely from the existence of one-way functions.

2 Preliminaries

2.1 Pseudorandom Functions

We first define the standard notion of pseudorandom functions (PRFs).

Syntax. Let $n_{\text{in}} = n_{\text{in}}(\kappa)$, and $n_{\text{out}} = n_{\text{out}}(\kappa)$ be integer-valued positive polynomials of the security parameter $\kappa$. A pseudorandom function is defined by a pair of PPT algorithms $\Pi_{\text{PRF}} = (\text{PRF.Gen, PRF.Eval})$ where:

- $\text{PRF.Gen}(1^\kappa) \rightarrow K$: The key generation algorithm takes as input the security parameter $1^\kappa$ and outputs a key $K \in \{0, 1\}^k$.
- $\text{PRF.Eval}(K, x) :\rightarrow y$: The evaluation algorithm takes as input $x \in \{0, 1\}^{n_{\text{in}}}$ and outputs $y \in \{0, 1\}^{n_{\text{out}}}$.

Pseudorandomness. We define the notion of (adaptive) pseudorandomness for PRFs $\Pi_{\text{PRF}} = (\text{PRF.Gen, PRF.Eval})$. The security notion is defined by the following game between an adversary $A$ and a challenger:

Setup: At the beginning of the game, the challenger prepares the key $K \leftarrow \text{PRF.Gen}(1^\kappa)$ and a set $S$ initially set to be empty.

Evaluation Queries: During the game, $A$ can adaptively query an evaluation on any input. When $A$ submits $x \in \{0, 1\}^{n_{\text{in}}}$ to the challenger, the challenger evaluates $y \leftarrow \text{PRF.Eval}(K, x)$ and returns $y \in \{0, 1\}^{n_{\text{out}}}$ to $A$. It then updates $S \leftarrow S \cup \{x\}$.

Challenge Phase: At some point, $A$ chooses its target input $x^* \in \{0, 1\}^{n_{\text{in}}}$ such that $x^* \notin S$ and submits it to the challenger. The challenger chooses a random coin $\text{coin} \leftarrow \{0, 1\}$. If $\text{coin} = 0$, it evaluates $y^* \leftarrow \text{PRF.Eval}(K, x^*)$. If $\text{coin} = 1$, it samples a random value $y^* \leftarrow \{0, 1\}^{n_{\text{out}}}$. Finally, it returns $y^*$ to $A$.

Evaluation Queries: After the challenge phase, $A$ may continue to make evaluation queries with the added restriction that it cannot query $x^*$.

Guess: Eventually, $A$ outputs $\widehat{\text{coin}}$ as a guess for $\text{coin}$.

We say the adversary $A$ wins the game if $\widehat{\text{coin}} = \text{coin}$.
**Definition 1.** A PRF $\Pi_{\text{PRF}}$ is said to be (adaptive) pseudorandom if for all PPT adversary $A$, the probability of $A$ winning the above game is negligible.

It is a well known fact that PRFs are implied from one-way functions [GGM84, HILL99].

### 2.2 Constrained Pseudorandom Functions

We now define constrained pseudorandom functions (CPRFs).

**Syntax.** Let $n_{\text{in}} = n_{\text{in}}(\kappa)$, and $n_{\text{out}} = n_{\text{out}}(\kappa)$ be integer-valued positive polynomials of the security parameter $\kappa$. Let $\mathcal{C} = \{C_\kappa\}_{\kappa \in \mathbb{N}}$ be a family of circuits, where $C_\kappa$ is a set of circuits with domain $\{0, 1\}^{n_{\text{in}}}$ and range $\{0, 1\}$ whose sizes are polynomially bounded. In the following we drop the subscript for clarity.

A constrained pseudorandom function for $\mathcal{C}$ is defined by the four PPT algorithms $\Pi_{\text{CPRF}} = (\text{CPRF.Gen}, \text{CPRF.Eval}, \text{CPRF.Constrain}, \text{CPRF.ConstrainEval})$ where:

- \text{CPRF.Gen}(1^\kappa) \rightarrow K$: The key generation algorithm takes as input the security parameter $1^\kappa$ and outputs a master key $K \in \{0, 1\}^\kappa$.

- \text{CPRF.Eval}(K, x) \rightarrow y$: The evaluation algorithm takes as input the master key $K$ and input $x \in \{0, 1\}^{n_{\text{in}}}$ and outputs $y \in \{0, 1\}^{n_{\text{out}}}$.

- \text{CPRF.Constrain}(K, C) \rightarrow K_C$: The constrained key generation algorithm takes as input the master key $K$ and a circuit $C \in \mathcal{C}$ specifying the constraint and outputs a constrained key $K_C$.

- \text{CPRF.ConstrainEval}(K_C, x) \rightarrow y$: The constrained evaluation algorithm takes as input the constrained key $K_C$ and an input $x \in \{0, 1\}^{n_{\text{in}}}$ and outputs either $y \in \{0, 1\}^{n_{\text{out}}}$ or $\perp$.

**Correctness.** We define the notion of correctness for CPRFs. We say a CPRF $\Pi_{\text{CPRF}}$ is correct if for all $\kappa \in \mathbb{N}$, $n_{\text{in}}, n_{\text{out}} \in \text{poly}(\kappa)$, $K \in \text{CPRF.Gen}(1^\kappa)$, $C \in C_\kappa$, $K_C \in \text{CPRF.Constrain}(K, C)$, $x \in \{0, 1\}^{n_{\text{in}}}$ such that $C(x) = 1$, we have $\text{CPRF.Eval}(K, x) = \text{CPRF.ConstrainEval}(K_C, x)$.

**Pseudorandomness on Constrained Points.** We define the notion of (adaptive) pseudorandomness on constrained points for CPRFs. Informally, we require it infeasible to evaluate on a point when only given constrained keys that are constrained on that particular point. For any $C : \{0, 1\}^{n_{\text{in}}} \rightarrow \{0, 1\}^{n_{\text{out}}}$, let $\text{ConPoint} : \mathcal{C} \rightarrow \{0, 1\}^{n_{\text{in}}}$ be a function which outputs the set of all constrained points $x \mid C(x) = 0$. Here $\text{ConPoint}$ is not necessarily required to be efficiently computable.

Formally, this security notion is defined by the following game between an adversary $A$ and a challenger:

**Setup:** At the beginning of the game, the challenger prepares the master key $K \leftarrow \text{CPRF.Gen}(1^\kappa)$ and two sets $S_{\text{eval}}, S_{\text{con}}$ initially set to be empty.

**Queries:** During the game, $A$ can adaptively make the following two types of queries:

- **Evaluation Queries:** Upon a query $x \in \{0, 1\}^{n_{\text{in}}}$, the challenger evaluates $y \leftarrow \text{CPRF.Eval}(K, x)$ and returns $y \in \{0, 1\}^{n_{\text{out}}}$ to $A$. It then updates $S_{\text{eval}} \leftarrow S_{\text{eval}} \cup \{x\}$.

- **Constrained Key Queries:** Upon a query $C \in \mathcal{C}$, the challenger runs $K_C \leftarrow \text{CPRF.Constrain}(K, C)$ and returns $K_C$ to $A$. It then updates $S_{\text{con}} \leftarrow S_{\text{con}} \cup \{C\}$. 


**Challenge Phase:** At some point, A chooses its target input $x^* \in \{0,1\}^{n_{in}}$ such that $x^* \not\in S_{eval}$ and $x^* \in \text{ConPoint}(C)$ for all $C \in S_{\text{con}}$. The challenger chooses a random coin $\text{coin} \leftarrow \{0,1\}$. If $\text{coin} = 0$, it evaluates $y^* \leftarrow \text{PRF.Eval}(K, x^*)$. If $\text{coin} = 1$, it samples a random value $y^* \leftarrow \{0,1\}^{n_{out}}$. Finally, it returns $y^*$ to A.

**Queries:** After the challenge phase, A may continue to make evaluation queries with the added restriction that it cannot query $x^*$ as the evaluation query and cannot query any circuit $C$ such that $C(x^*) = 1$ as the constrained key query.

**Guess:** Eventually, A outputs $\tilde{\text{coin}}$ as a guess for $\text{coin}$.

We say the adversary A wins the game if $\tilde{\text{coin}} = \text{coin}$.

**Definition 2.** A CPRF $\Pi_{\text{CPRF}}$ is said to be (adaptive) pseudorandom on constrained points if for all PPT adversary A, $|\text{Pr}[A \text{ wins}] - 1/2| = \text{negl}(\kappa)$ holds.

**Remark 1** (Selective Security). In case all the constrained key queries made by the adversary must be provided before the Setup phase, we say it is selective pseudorandom on constrained points.

**Remark 2** (Collusion Resistance). We can adjust the strength of the above notion by imposing a restriction on the number of constrained keys an adversary can query. In case the adversary can query at most one constrained key, it is called single-key secure. In case we can tolerate up to $Q$ constrained key queries, we say it is $Q$-collusion resistance.

### 3 Constructing CPRFs from Standard PRFs

In this section, we provide a construction of an adaptive pseudorandom on constrained points, $Q$-collusion resistant CPRFs for the bit-fixing predicate from any PRF, where $Q$ can be set to be any constant independent of the security parameter. In particular, the result implies the existence of such CPRFs from one-way functions. CPRFs with the same property were very recently constructed by Davidson and Nishimaki [DN18], however, their results require the LWE assumption. Other than ours and [DN18], no other CPRFs are known to be adaptive and/or to achieve $Q$-collusion resistance for any $Q > 1$ in the standard model, excluding the CPRFs where the constraints are the trivial singleton sets: $\mathcal{F} = \{ \{x\} \mid x \in \{0,1\}^{n_{in}} \}$ or prefix-fixing predicates [GGM84, BW13, BLW17].

#### 3.1 Preparation: Bit-Fixing Predicates

Here, we provide the constraint class we will be considering: bit-fixing predicates. Formally, for a vector $v \in \{0,1,*\}^{\ell}$, define the circuit $C_{v}^{\text{BF}} : \{0,1\}^{\ell} \rightarrow \{0,1\}$ associated with $v$ as

$$C_{v}^{\text{BF}}(x) = 1 \iff \bigwedge_{i=1}^{\ell} \left( (v_{i} \neq x_{i}) \vee (v_{i} = \ast) \right) = 1,$$

where $v_{i}$ and $x_{i}$ denotes the $i$-th bit of the string $v$ and $x$, respectively. Then, the bit-fixing predicate (for length $\ell$ inputs) is defined as

$$C_{\ell}^{\text{BF}} := \{ C_{v}^{\text{BF}} \mid v \in \{0,1,*\}^{\ell} \}.$$
Since we can consider a canonical representation of the circuit $C^\text{BF}_{v^*}$ given the string $v \in \{0, 1, *\}^\ell$, with an abuse of notation, we may occasionally write $v \in C^\text{BF}_v$ and view $v$ as $C^\text{BF}_v$ when the meaning is clear.

Moreover, for any $v \in \{0, 1, *\}^\ell$ and $T = (t_1, \ldots, t_Q) \in [\ell]^Q$ such that $Q \leq \ell$, let us define $v_T \in \{0, 1, *\}^Q$ as the string $v_{t_1}v_{t_2}\cdots v_{t_Q}$, where $v_i$ is the $i$-th symbol of $v$. In addition, let $G_{\text{aut}}$ be a function defined as

$$G_{\text{aut}}(v_T) = \{ w \in \{0, 1\}^Q \mid C^\text{BF}_{v_T}(w) = 1 \}.$$

Namely, it is the set of all points with the same length as $v_T$ that equals to $v_T$ on the non-wild cared entries. For example, if $\ell = 8, Q = 5, v = 011*011$, and $T = (4, 1, 2, 6, 1)$, then $v_T = *0110$ and the authorized set of points would be $G_{\text{aut}}(v_T) = \{00110, 10110\}$. Here, with an abuse of notation, we define the function $G_{\text{aut}}$ for all input lengths.

### 3.2 Construction

Let $n_{\text{in}} = n_{\text{in}}(\kappa)$, and $n_{\text{out}} = n_{\text{out}}(\kappa)$ be integer-valued positive polynomials of the security parameter $\kappa$ and $Q$ be any constant positive integer smaller than $n_{\text{in}}$. Let $\mathcal{C}^\text{BF} := \{ C_{\kappa} \}_{\kappa \in \mathbb{N}} := \{ C^\text{BF}_{n_{\text{in}}(\kappa)} \}_{\kappa \in \mathbb{N}}$ be a family of circuits representing the class of constraints. Let $\Pi_{\text{PRF}} = (\text{PRF.Gen}, \text{PRF.Eval})$ be any PRF with input length $n_{\text{in}}$ and output length $n_{\text{out}}$.

Our $Q$-collusion resistance CPRF $\Pi_{\text{CPRF}}$ for the constrained class $\mathcal{C}^\text{BF}$ is provided as follows:

**CPRF.Gen($1^\kappa$) $\rightarrow$ $K$:** On input the security parameter $1^\kappa$, it runs $\tilde{K}_{T,w} \leftarrow \text{PRF.Gen}(1^\kappa)$ for all $T \in [n_{\text{in}}]^Q$ and $w \in \{0, 1\}^Q$. Then it outputs the master key as

$$K = (\tilde{K}_{T,w})_{T \in [n_{\text{in}}]^Q, w \in \{0, 1\}^Q}.$$

**CPRF.Eval($K, x$) $\rightarrow$ $y$:** On input the master key $K$ and input $x \in \{0, 1\}^{n_{\text{in}}}$, it first parses $(\tilde{K}_{T,w})_{T \in [n_{\text{in}}]^Q, w \in \{0, 1\}^Q} \leftarrow K$. It then computes

$$y = \bigoplus_{T \in [n_{\text{in}}]^Q} \text{PRF.Eval}(\tilde{K}_{T,x_T}, x),$$

where recall $x_T \in \{0, 1\}^Q$ is defined as the string $x_{t_1}x_{t_2}\cdots x_{t_Q}$ and $T = (t_1, \ldots, t_Q)$. Finally, it outputs $y \in \{0, 1\}^{n_{\text{out}}}$.

**CPRF.Constrain($K, C^\text{BF}_v$) $\rightarrow$ $K_v$:** On input the master key $K$ and a circuit $C^\text{BF}_v \in \mathcal{C}^\text{BF}_v$, it first parses $(\tilde{K}_{T,w})_{T \in [n_{\text{in}}]^Q, w \in \{0, 1\}^Q} \leftarrow K$ and sets $v \in \{0, 1, *\}^{n_{\text{in}}}$ as the representation of $C^\text{BF}_v$. Then it outputs the constrained key

$$K_v = \left( (\tilde{K}_{T,w})_{w \in G_{\text{aut}}(v_T)} \right)_{T \in [n_{\text{in}}]^Q},$$

where recall $G_{\text{aut}}(v_T) = \{ w \in \{0, 1\}^Q \mid C^\text{BF}_{v_T}(w) = 1 \}$.

**CPRF.ConstrainEval($K_v, x$) $\rightarrow$ $y$:** On input the constrained key $K_v$ and an input $x \in \{0, 1\}^{n_{\text{in}}}$, it first parses $(\tilde{K}_{T,w})_{w \in G_{\text{aut}}(v_T)} \leftarrow K_v$. It then uses the PRF keys included in the constrained key and computes

$$y = \bigoplus_{T \in [n_{\text{in}}]^Q} \text{PRF.Eval}(\tilde{K}_{T,x_T}, x).$$

Finally, it outputs $y \in \{0, 1\}^{n_{\text{out}}}$.
3.3 Correctness

We check correctness of our CPRF. Let \( C^\text{BF}_v \) be any bit-fixing predicate in \( \mathcal{C}^\text{BF} \). Put differently, let us fix an arbitrary \( v \in \{0, 1, \ast \}^n \). Then, by construction we have

\[
K_v = \left( \left( \tilde{K}_{T,w} \right)_{w \in G_{\text{aut}}(v_T)} \right)_{T \in [n]_Q} \leftrightarrow \text{CPRF.Constrain}(K, C^\text{BF}_v).
\]

Now, for any \( x \in \{0, 1\}^n \) such that \( C^\text{BF}_v(x) = 1 \), by definition of the bit-fixing predicate, we have

\[
\bigwedge_{i=1}^n \left( \left( v_i = x_i \right) \lor \left( v_i = \ast \right) \right) = 1.
\]

Then, by definition of function \( G_{\text{aut}} \), we have \( x_T \in G_{\text{aut}}(v_T) \) for any \( T \in [n]_Q \) since we have \( C^\text{BF}(x_T) = 1 \) if \( C^\text{BF}_v(x) = 1 \). In particular, we have

\[
\tilde{K}_{T,x_T} \in K_v \text{ for all } T \in [n]_Q.
\]

Therefore, since CPRF.Eval and CPRF.ConstrainEval are computed exactly in the same way using the same PRF keys, correctness holds.

3.4 Pseudorandomness on Constrained Points

We show the following lemma.

**Theorem 1.** If the underlying PRF \( \Pi_{\text{PRF}} \) is adaptive pseudorandom, then our above CPRF \( \Pi_{\text{CPRF}} \) for the bit-fixing predicate \( C^\text{BF} \) is adaptively pseudorandom on constrained points and \( Q \)-collusion resistant.

**Proof.** We show the theorem by considering the following sequence of games between an adversary \( A \) against the pseudorandomness on constrained points security game and the challenger. In the following, for simplicity, we say an adversary \( A \) against the CPRF pseudorandomness game. Below, let \( E_i \) denote the probability that \( \text{coin} = \text{coin} \) holds in \( \text{Game}_i \). Recall that \( A \) makes at most \( Q \)-constrained key queries, where \( Q \) is a constant.

**Game_1:** This is defined as the ordinary CPRF pseudorandomness game played between \( A \) and the challenger. In particular, at the beginning of the game the challenger prepares the empty sets \( S_{\text{eval}}, S_{\text{con}} \). In this game, the challenger responds to the queries made by \( A \) as follows:

- When \( A \) submits \( x \in \{0, 1\}^n \) as the evaluation query, the challenger returns \( y \leftarrow \text{CPRF.Eval}(K, x) \) to \( A \) and updates \( S_{\text{eval}} \leftarrow S_{\text{eval}} \cup \{x\} \).
- When \( A \) submits \( C^\text{BF}_{v(j)} \in \mathcal{C}^\text{BF}_v \) as the \( j \)-th \((j \in [Q]) \) constrained key query, the challenger returns \( K_{v(j)} \leftarrow \text{CPRF.Constrain}(K, C^\text{BF}_{v(j)}) \) to \( A \) and updates \( S_{\text{con}} \leftarrow S_{\text{con}} \cup \{C^\text{BF}_{v(j)}\} \).

Furthermore, recall that when \( A \) submits the target input \( x^* \in \{0, 1\}^n \) as the challenge query, we have the restriction \( x^* \notin S_{\text{eval}} \) and \( x^* \in \text{ConPoint}(C^\text{BF}_{v(j)}) \) for all \( C^\text{BF}_{v(j)} \in S_{\text{con}} \). Here, the latter condition is equivalent to

\[
\bigwedge_{i=1}^n \left( \left( v^{(j)}_i = x^*_i \right) \lor \left( v^{(j)}_i = \ast \right) \right) = 0 \quad \text{for all } C^\text{BF}_{v(j)} \in S_{\text{con}}, \tag{1}
\]

By definition, we have \( |\Pr[E_0] - 1/2| = \epsilon \).
Game$_1$: In this game, we add an extra abort condition for the challenger. Specifically, at the end of the game, the challenger samples a random set $T^* \overset{\$}{\leftarrow} [n_{in}]^Q$. Let us set $T^* = (t_1, \cdots, t_Q)$. The challenger further samples $b^*_j \overset{\$}{\leftarrow} \{0,1\}$ for all $j \in [Q]$. Let $b^*_T := b_{t_1} b_{t_2} \cdots b_{t_Q} \in \{0,1\}^Q$. Then, the challenger checks whether the following equation holds with respect to the constrained key queries and the challenge query made by the adversary.

1. The challenger aborts if there exists $j \in [Q]$ such that
   \[ (v_{t_j}^{(j)} \neq b^*_j) \vee (v_{t_j}^{(j)} = *) = 0. \]  

2. The challenger aborts if $x^*$ does not satisfy
   \[ (b^*_T \neq x^*) = \bigwedge_{j \in [Q]} (b^*_j \neq x^*_j) = 1. \]

3. The challenger aborts if $(T^*, b^*_T)$ chosen by the challenger does not equal to the first pair (with respect to some pre-defined order over $[n_{in}]^Q \times \{0,1\}^Q$ such as the lexicographic order) that satisfies Equation (2) for all $j \in [Q]$ and Equation (3). Note that it is possible to efficiently find such a pair by enumerating over $[n_{in}]^Q \times \{0,1\}^Q$ since $Q = O(1)$.

When the challenger aborts, it substitutes the guess $\hat{\text{coin}}$ outputted by A with a random bit. We call this event abort.

As we will show in Lemma 1, there exists at least single pair $(T^*, b^*_T) \in [n_{in}]^Q \times \{0,1\}^Q$ that satisfies Equation (2) for all $j \in [Q]$ and Equation (3). Therefore, the event abort occurs with probability $1 - 1/(2n)^Q$. Furthermore, it can be seen that abort occurs independently from the view of A. Therefore, we have

\[ |\Pr[E_1] - 1/2| = |\Pr[E_0] \cdot \Pr[-\text{abort}] + (1/2) \cdot \Pr[\text{abort}] - 1/2| = |\Pr[E_0] \cdot (1/(2n)^Q) + (1/2) \cdot (1 - 1/(2n)^Q) - 1/2| = \epsilon/((2n)^Q), \]

where we used the fact that $\hat{\text{coin}}$ is randomly chosen and thus equals to coin with probability 1/2 when abort occurs.

Game$_2$: Recall that in the previous game, the challenger aborts at the end of the game, if the abort condition is satisfied. In this game, we change the game so that the challenger chooses $T^*$ and $b^*_T$, at the beginning of the game and aborts as soon as the challenger makes a constrained key query $C^B_{(\phi)} \in \mathcal{C}^B_{n_{in}}$ that does not satisfy Equation (2) or a challenge query for $x^*$ that does not satisfy Equation (3). Furthermore, it aborts if $(T^*, b^*_T)$ is not the first pair that

\footnote{One may wonder why the final condition for the abort is necessary, because the reduction in the proof of Lemma 2 works even without it. This additional abort step is introduced to make the probability of abort to occur independently of the choice of the constrained key queries and the challenge query made by the adversary. Without this step, we cannot lower bound $|\Pr[E_1] - 1/2|$. Similar problem was identified by Waters [Wat05], who introduced “the artificial abort step” to resolve it. Our analysis here is much simpler because we can compute the abort probability exactly in our case.}
satisfies Equation (2) for all \( j \in [Q] \) and Equation (3). Since it is only a conceptual change, we have

\[
Pr[E_2] = Pr[E_1].
\]

**Game3:** In this game, we change how the challenger responds to the challenge query when \( \text{coin} = 0 \). For all the evaluation query and constrained key query, the challenger acts exactly the same way as in the previous game. In the previous game \Game_2, when the adversary submits the target input \( x^* \in \{0,1\}^{n_{\text{in}}} \) as the challenge query, the challenger first checks whether the condition in Equation (3) holds. If not it aborts. Otherwise, it samples \( \text{coin} \overset{\$}{\leftarrow} \{0,1\} \). In case \( \text{coin} = 0 \), it computes \( \text{CPRF.Eval}(K, x^*) \) as

\[
y^* = \bigoplus_{T \in [n_{\text{in}}]^Q} \text{PRF.Eval}(\bar{K}_{T,x^*}, x^*)
\]

using the master key \( K = (\bar{K}_{T,w})_{T \in [n_{\text{in}}]^Q, w \in \{0,1\}^{Q}} \) it constructed at the beginning of the game, where \( \bar{K}_{T,w} \leftarrow \text{PRF.Gen}(1^\kappa) \) for all \( T \in [n_{\text{in}}]^Q \) and \( w \in \{0,1\}^{Q} \). Due to the condition in Equation (3), i.e., \( b_{T^*}^* = x_{T^*}^* \in \{0,1\}^{Q} \), we can rewrite Equation (4) as

\[
y^* = \text{PRF.Eval}(\bar{K}_{T^*,b_{T^*}^*}, x^*) \oplus \left( \bigoplus_{T \in [n_{\text{in}}]^Q \backslash T^*} \text{PRF.Eval}(\bar{K}_{T,x^*}, x^*) \right).
\] (5)

In this game \Game_3, when \( \text{coin} = 0 \), the challenger instead samples a random \( \bar{y}^* \overset{\$}{\leftarrow} \{0,1\}^{n_{\text{out}}} \) and returns the following to \( A \) instead of returning \( y^* \) to \( A \) as in Equation (5):

\[
y^* = \bar{y}^* \oplus \left( \bigoplus_{T \in [n_{\text{in}}]^Q \backslash T^*} \text{PRF.Eval}(\bar{K}_{T,x^*}, x^*) \right).
\] (6)

We show in Lemma 2 that

\[
|Pr[E_2] - Pr[E_3]| = \text{negl}(\kappa)
\]

assuming pseudorandomness of the underlying \( \Pi_{\text{PRF}} \). In this game \Game_3, the distribution of \( y^* \) for \( \text{coin} = 0 \) and \( \text{coin} = 1 \) are exactly the same since \( A \) has not made an evaluation query on \( x^* \) and \( \bar{K}_{T^*,b_{T^*}^*} \) is not given through any of the constrained key query. Concretely, \( \bar{y}^* \) is distributed uniform random regardless of whether \( \text{coin} = 0 \) or \( \text{coin} = 1 \) and thus the value of \( \text{coin} \) is information theoretically hidden to \( A \). Therefore, we have

\[
Pr[E_3] = 1/2.
\]

Combining everything together with Lemma 1 and Lemma 2, we have

\[
\epsilon = |Pr[E_0] - 1/2| \leq (2n_{\text{in}})^Q \cdot (|Pr[E_3] - 1/2| + \text{negl}(\kappa)) = \text{negl}(\kappa),
\]

where the last equality follows by recalling that \( n_{\text{in}} = \text{poly}(\kappa) \) and \( Q \) a constant.

**Lemma 1.** In Game\(_1\), we have

\[
\{(T^*, b_{T^*}^*) \in [n_{\text{in}}]^Q \times \{0,1\}^Q \mid (T^*, b_{T^*}^*) \text{ satisfies Equation (2) for all } j \in [Q] \text{ and Equation (3)}\} \neq \emptyset.
\]
Proof. By the restriction posed on \( A \) in the game, for all \( j \in [Q] \), there exists \( t^{(j)} \in [n_{in}] \) such that

\[ v^{(j)}_{t^{(j)}} = 1 - x^{*}_{t^{(j)}}. \]

Let us denote \( T := (t^{(1)} , \ldots , t^{(Q)}) \in [n_{in}]^{Q} \) and \( b_{T} := x^{*}_{T} \in \{0,1\}^{Q} \). It is easy to check that Equation (2) for all \( j \in [Q] \) and Equation (3) hold if \( T^{*} = T \) and \( b_{T^{*}}^{*} = b_{T} \). \( \square \)

**Lemma 2.** We have \( |Pr[\mathsf{E}_{2}] - Pr[\mathsf{E}_{3}]| = \text{negl}(\kappa) \) assuming that the underlying PRF \( \Pi_{\text{PRF}} \) satisfies adaptive pseudorandomness.

Proof. For the sake of contradiction, let us assume an adversary \( A \) that distinguishes \( \text{Game}_{2} \) and \( \text{Game}_{3} \) non-negligible probability \( \epsilon' \). We then construct an adversary \( B \) that breaks the pseudorandomness of \( \Pi_{\text{PRF}} \) with the same probability. The adversary \( B \) proceeds as follows.

At the beginning of the game \( B \) samples a random tuple \( T^{*} = (t_{1}, \ldots , t_{Q}) \overset{\$}{\leftarrow} [n_{in}]^{Q} \) and \( b_{T^{*}_{j}} \overset{\$}{\leftarrow} \{0,1\} \) for all \( j \in [Q] \) as in the \( \text{Game}_{2} \)-challenger. Let \( b_{T^{*}} := b_{t_{1}}^{*} b_{t_{2}}^{*} \cdots b_{t_{Q}}^{*} \in \{0,1\}^{Q} \). Then, it further samples \( K_{T,w} \leftarrow \text{PRF.Gen}(\mathsf{1}^{*}) \) for all \( T \in [n_{in}]^{Q} \) and \( w \in \{0,1\}^{Q} \) except for \( (T^{*}, b_{T^{*}}^{*}) \). It then sets the (simulated) master key \( K^{*} \) as

\[ K^{*} = (K_{T,w})_{(T,w) \in [n_{in}]^{Q} \times \{0,1\}^{Q} \setminus (T^{*}, b_{T^{*}}^{*})}. \]

Here, \( B \) implicitly sets \( K_{T^{*},b_{T^{*}}^{*}} \) as the PRF key used by its PRF challenger. Finally, \( B \) prepares two empty sets \( S_{\text{eval}} , S_{\text{con}} \). \( B \) then simulates the response to the queries made by \( A \) as follows:

- When \( A \) submits \( x \in \{0,1\}^{n_{in}} \) as the evaluation query, \( B \) checks whether \( x_{T^{*}} = b_{T^{*}}^{*} \). If not, then it can use the simulated master key \( K^{*} \) to compute

\[ y = \bigoplus_{T \in [n_{in}]^{Q}} \text{PRF.Eval}(K_{T,x_{T}} , x). \]

Otherwise, it makes an evaluation query to its PRF challenger on the input \( x \). When it receives back \( \tilde{y} \) from the PRF challenger, \( B \) computes the output as

\[ y = \tilde{y} \oplus \bigoplus_{T \in [n_{in}]^{Q} \setminus T^{*}} \text{PRF.Eval}(K_{T,x_{T}} , x). \]

Finally, \( B \) returns \( y \) to \( A \) and updates \( S_{\text{eval}} \leftarrow S_{\text{eval}} \cup \{x\} \). Note that by the specification of the PRF challenger, we have \( \tilde{y} = \text{PRF.Eval}(K_{T^{*},b_{T^{*}}^{*}} , x) \).

- When \( A \) submits \( C_{v_{j}}^{\text{BF}} \in C_{n_{in}}^{\text{BF}} \) as the \( j \)-th \( (j \in [Q]) \) constrained key query, \( B \) checks whether the condition in Equation (2) holds. If not it aborts and outputs a random bit. Otherwise, it returns the following constrained key \( K_{v_{j}}^{*} \) to \( A \):

\[ K_{v_{j}}^{*} = \left( (K_{T,w})_{w \in G_{\text{aut}}(v_{j})} \right)_{T \in [n_{in}]^{Q}}, \]

where recall \( G_{\text{aut}}(v_{j}) = \{w \in \{0,1\}^{Q} \mid C_{v_{j}}^{\text{BF}}(w) = 1\} \). Here, \( B \) can prepare all the PRF keys since condition in Equation (2) guarantees us that we have \( b_{T^{*}}^{*} \not\in G_{\text{aut}}(v_{j}) \), or equivalently, \( C_{v_{j}}^{\text{BF}}(b_{T^{*}}^{*}) = 0 \). Namely, \( K_{T^{*},b_{T^{*}}^{*}} \) is not included in \( K_{v_{j}}^{*} \).
• When A submits the target input $x^* \in \{0,1\}^{n_{in}}$ as the challenge query, B checks whether the condition in Equation (3) holds. If not it aborts and outputs a random bit. Otherwise, B queries its PRF challenger on $x^*$ as its challenge query and receives back $\tilde{y}^*$. It then computes $y^*$ as in Equation (6) and returns $y^*$ to A. Here, since Equation (3) holds, $K_{T^*,b_{T^*}}$ must be required to compute on input $x^*$.

Finally, A outputs its guess $\text{coin}$. B then checks whether $(T^*, b_{T^*})$ is the first pair that satisfies Equation (2) for all $j \in [Q]$ and Equation (3). If it does not hold, B outputs a random bit. Otherwise, B outputs $\text{coin}$ as its guess.

This completes the description of B. It is easy to check that in case $\text{coin} = 0$, B receives $\tilde{y}^* \leftarrow \text{PRF.Eval}(K_{T^*,b_{T^*}}, x^*)$, hence B simulates Game$_2$ perfectly. Otherwise in case $\text{coin} = 1$, B receives $\tilde{y}^* \leftarrow (0,1)^{n_{out}}$, hence B simulates Game$_3$ perfectly. Therefore, we conclude that B wins the PRF pseudorandomness game with probability exactly $\epsilon'$. Assuming that $\Pi_{\text{PRF}}$ is pseudorandom, this is a contradiction, hence, $\epsilon'$ must be negligible.

This completes the proof.

References


