# Fully secure constrained pseudorandom functions using random oracles

Dennis Hofheinz

May 27, 2014

**Abstract**

A constrained pseudorandom function (CPRF) $\mathsf{PRF}$ allows to derive constrained evaluation keys that only allow to evaluate $\mathsf{PRF}$ on a subset of inputs. CPRFs have only recently been introduced independently by three groups of researchers. However, somewhat curiously, all of them could only achieve a comparatively weak, selective-challenge form of security (except for small input spaces, very limited forms of constrained keys, or with superpolynomial security reductions).

In this paper, we construct the first fully secure CPRF without any of the above restrictions. Concretely, we support "bit-fixing" constrained keys that hardwire an arbitrary subset of the input bits to fixed values, we support exponentially large input spaces, and our security reduction is polynomial. We require very heavyweight tools: we assume multilinear maps, indistinguishability obfuscation, and our proof is in the random oracle model. Still, our analysis is far from tautological, and even with these strong building blocks, we need to develop additional techniques and tools.

As a simple application, we obtain the first adaptively secure non-interactive key exchange protocols for large user groups.

**Keywords:** constrained pseudorandom functions, adaptive security, non-interactive key exchange.

## 1 Introduction

**Motivation.** Pseudorandom functions (PRFs [25]) are a fundamental cryptographic tool. Intuitively, a pseudorandom function $\mathsf{PRF}$ takes a key $K$ and a preimage $X$, and assigns it an image $\mathsf{PRF}_K(X)$, such that it is infeasible to distinguish oracle access to $\mathsf{PRF}_K(\cdot)$ (for unknown $K$) from oracle access to a truly random function. PRFs can be constructed from one-way functions [25, 31] and have found applications in all areas of cryptography.

Quite recently, a useful extension of PRFs has been discovered[1], in which keys $K$ can be constrained to a subset $S$ of preimages, such that a constrained key $K_S$ can be used *only* to evaluate $\mathsf{PRF}_K(X)$ for $X \in S$. In the corresponding security experiment, an adversary can ask for arbitrary constrained keys $K_S$ (for sets $S$ from a class $\mathcal{S}$ of allowed subsets), and must later distinguish a challenge image $\mathsf{PRF}_K(X^*)$ from a random bitstring. Depending on the class $\mathcal{S}$ of allowed subsets, this enables applications such as searchable encryption [33], broadcast encryption [33, 6], and identity-based non-interactive key exchange [6].

However, most existing constructions of such constrained PRFs (CPRFs) achieve only a comparatively weak, selective-challenge form of security, in which the challenge preimage $X^*$ must be chosen by the adversary in advance.[2] In particular, this leads to applications which

[1]The concurrent works [33, 6, 8] all describe the same concept, although under different names and with different constructions; we will mostly adopt the terminology of [6].

[2]This are exceptions for special classes of subsets and preimages. Namely, [6] present a fully secure CPRF for a restricted class of "left-right" subsets $S$ that fix either the left or the right half of the preimage (see also [42]).

only enjoy a selective-challenge (or, non-adaptive) form of security. Below, we will explain the difficulty of constructing fully secure CPRFs, and how to overcome it.

**Our contribution.** We present a new proof strategy for the construction of fully secure CPRFs. In particular, we construct the first fully secure "bit-fixing" CPRF (for constrained keys that hardwire a subset of the preimage bits to arbitrary bit values). Bit-fixing PRFs subsume almost all CPRF examples from [33, 6, 8].[3] In particular, plugging our CPRF into [6, Theorem 6.2] directly enables a broadcast encryption scheme with optimal ciphertext length *and polynomial security reduction.* We also obtain the first adaptively secure non-interactive key exchange protocols for large user groups (both in the identity-based [42, 15, 24, 40, 19, 6, 7] and in the public-key setting [12, 18]).

Our proof strategy requires several heavyweight tools. In particular, we assume multilinear maps (however in a way that is compatible with the recent candidates [21, 14]), indistinguishability obfuscation, and a random oracle. Moreover, even given these powerful tools, the construction and its analysis are far from straightforward. Hence, although efficient and fully secure in the usual polynomial sense, our CPRF should not be considered practical by any means. In fact, we view our scheme as a proof of concept, and as a way to showcase our proof strategy. However, at least for our public-key non-interactive key exchange, we can implement our strategy without indistinguishability obfuscation.

In what follows, we will give some technical details on our results.

**Selectively secure CPRFs.** For concreteness, say that we want to construct a bit-fixing CPRF $\mathsf{PRF}$, i.e., one that allows for constrained keys $K_{S_{\overline{X}}}$ for sets of the form $S_{\overline{X}} = \{X = (x_i)_{i=1}^{\ell} \mid x_i = \overline{x}_i \text{ or } \overline{x}_i = \bot\}$ with $\overline{X} = (\overline{x}_i)_{i=1}^{\ell} \in (\{0,1\} \cup \{\bot\})^{\ell}$. An adversary $\mathcal{A}$ on $\mathsf{PRF}$ may first ask for polynomially many constrained keys $K_{S_{\overline{X}}}$, and then gets challenged on a preimage $X^*$. The goal of a successful simulation is to be able to prepare all $K_{S_{\overline{X}}}$, but *not* to be able to compute $\mathsf{PRF}_K(X^*)$.

Now if $X^* = (x_i^*)_{i=1}^{\ell}$ is known in advance, then the simulation can set up the function $\mathsf{PRF}_K(\cdot)$ in an "all-but-one" way, such that all images except $\mathsf{PRF}_K(X^*)$ can be computed. For instance, the selective-security simulation from [6] (building on ideas from [36, 5]) sets up

$$\mathsf{PRF}_K(X) = e(g, \ldots, g)^{\prod_{i=1}^{\ell} \alpha_{i,x_i}} \qquad (\text{for } K = (\alpha_{i,b})_{i,b}), \tag{1}$$

where $e$ is an $(\ell - 1)$-linear map, and the simulation knows all $\alpha_{i,1-x_i^*}$ (while the $\alpha_{i,x_i^*}$ are only known "in the exponent," as $g^{\alpha_{i,x_i^*}}$). This setup not only allows to compute $\mathsf{PRF}_K(X)$ as soon as there is an $i$ with $x_i \neq x_i^*$ (such that the corresponding $\alpha_{i,x_i} = \alpha_{i,1-x_i^*}$ is known); also, assuming a graded multilinear map (as in [21, 14]), evaluation can be *delegated.* (For instance, a constrained key that allows to evaluate all inputs with $x_1 = 1$ would contain $\alpha_{1,1}$ and $g^{\alpha_{i,b}}$ for all other $i, b$.)

**Why achieving full security is difficult.** However, observe now what happens when $\mathcal{A}$ chooses the challenge preimage $X^*$ only *after* asking for constrained keys. Then, the simulation may be forced to commit to the full function $\mathsf{PRF}_K(\cdot)$ (information-theoretically) before even knowing where "not to be able to evaluate." For instance, for the CPRF from [6] sketched above, already a few suitably chosen constrained keys (for subsets $S_i$) fully determine $\mathsf{PRF}_K(\cdot)$, while the corresponding subsets $S_i$ leave exponentially many potential challenge preimages $X^*$ uncovered. If we assume that the simulation either can or cannot evaluate $\mathsf{PRF}_K$ on a given preimage (at least once $\mathsf{PRF}_K(\cdot)$ is fully determined), we have the following dilemma. Let $\mathcal{C}$ be

Furthermore, [33] construct fully secure CPRFs both for polynomially-small preimage spaces, and for classes of subsets $S$ that cover intervals of preimages, where preimages are interpreted as integers. Finally, [20] prove the full security of a CPRF for subsets $S$ that fix a prefix of the preimage. Besides, [6, 8] also show full security for their CPRFs using a complexity leveraging argument (i.e., by guessing $X^*$ in advance), which leads to an exponential security reduction for standard preimage sizes.

[3]The exception are "circuit PRFs" (constructed in [6] with selective-challenge security), which allow to constrain keys to arbitrary preimage sets recognized by efficient circuits $C$.

set of preimages that the simulation *cannot* evaluate. If $\mathcal{C}$ is too small, then $X^* \in \mathcal{C}$ will not happen sufficiently often, so that the simulation cannot learn anything from $\mathcal{A}$. But if $\mathcal{C}$ is too large, then the simulation will not be able to construct "sufficiently general" constrained keys for $\mathcal{A}$ (because the corresponding sets $S$ would intersect with $\mathcal{C}$).[4]

This argument eliminates not only guessing $X^*$ (at least when aiming at a polynomial reduction), but also the popular class of "partitioning arguments" (e.g., [13, 36, 3, 44, 28]). (Namely, while guessing $X^*$ corresponds to $|\mathcal{C}| = 1$ above, partitioning arguments consider larger sets $\mathcal{C}$. However, the argument above excludes sets $\mathcal{C}$ of *any* size for relevant classes $\mathcal{S}$ of constraining sets and superpolynomial preimage space.) In particular, since the selectively-secure CPRFs from [33, 6, 8] fulfill the assumptions of the argument, it seems hopeless to prove them fully secure, at least for standard preimage sizes.

**Idea 1: adapt the function on the fly.** Hence, to construct a bit-fixing CPRF, we must contradict the assumptions in the negative argument above. Concretely, we will not fully determine $\mathsf{PRF}_K$ initially, but have $\mathsf{PRF}_K(X)$ depend also on a random oracle query $\mathsf{H}(X)$ that must be made explicitly for each $X$. Unfortunately, the most straightforward strategy (simply setting $\mathsf{PRF}_K(X) = \mathsf{PRF}'_K(\mathsf{H}(X))$ for some selectively secure CPRF $\mathsf{PRF}'_K$) does *not* work. Concretely, permuting the preimages arbitrarily destroys their structure, and hence makes delegation (or, the construction of constrained keys) impossible.[5] Hence, we need another idea.

**Idea 2: tagged preimages that enable imperfections of constrained keys.** Our idea is to associate a *tag* $T_X = \mathsf{H}(X)$ to each preimage $X$. The image $\mathsf{PRF}_K(X)$ will then depend on $X$ *and* $T_X$, so we can write $\mathsf{PRF}_K(X) = \mathsf{PRF}'_K(X, T_X)$ for a suitable function $\mathsf{PRF}'_K$. During the simulation, we will be able to evaluate $\mathsf{PRF}'_K(\cdot, \cdot)$ everywhere (even for $X = X^*$), *except* when $T_X = \overline{T}_X$ for a special tag $\overline{T}_X$ (that depends on $X$). We will then adaptively program the random oracle such that $\mathsf{H}(X) = \overline{T}_X$ precisely for the challenge preimage $X^*$. Note that since we do not perturb the preimages themselves with $\mathsf{H}$, this approach does not destroy any structure on preimages, as necessary for key delegation. Furthermore, the $\overline{T}_X$ will be fixed in advance, and so $\mathsf{PRF}'_K$ can be set up in a selectively-secure manner similar to (1).

However, we still need a way to hide the imperfections of constrained keys (i.e., the special tags $\overline{T}_X$ for which evaluation fails). Otherwise, $\mathcal{A}$ may be able to distinguish the simulation (in which $\mathsf{H}(X^*)$ is programmed to $\overline{T}_{X^*}$) from the real security game (in which $\mathsf{H}(X^*)$ is truly random) by somehow "interpolating" $\overline{T}_{X^*}$ from the imperfections of known constrained keys. In fact, since our delegation mechanism will be highly structured, the arising imperfections $\overline{T}_X$ will be highly correlated for different constrained keys. Hence, we need another trick to hide the imperfections of the constrained keys.

**Idea 3: hide key imperfections.** Intuitively, we will hide the imperfections of constrained keys (i.e., the special tags $\overline{T}_X$ for which this key fails) by obfuscating the functionality of these keys. However, since we will employ indistinguishability obfuscation [1, 27, 22], this only guarantees that the obfuscations of two constrained keys with the same imperfections are indistinguishable. To introduce these imperfections in the first place, we cannot rely on obfuscation alone, and hence we need another tool.

We introduce obfuscations with a new tool we call "extensible testers." Intuitively, an extensible tester $\mathcal{T}$ is an obfuscation of a point function [11], such that $\mathcal{T}(\mathsf{Z}) = 1$ precisely for one hidden input $\mathsf{Z}$. (In our case, $\mathsf{Z}$ will be a whole set of bitstrings.) However, unlike with indistinguishability obfuscation, such a tester $\mathcal{T}$ for a random $\mathsf{Z}$ cannot be distinguished from the obfuscation of an all-zero function. Additionally, a tester $\mathcal{T}$ with $\mathcal{T}(\mathsf{Z}) = 1$ is *extensible* in the sense that $\mathcal{T}$ and $\mathsf{Z}'$ allow to derive a tester $\mathcal{T}'$ with $\mathcal{T}(\mathsf{Z} \cup \mathsf{Z}') = 1$. (In other words, the set

[4]In fact, for many classes $\mathcal{S}$ of allowed constraining sets, $\mathcal{A}$ can easily ask for constrained keys that, taken together, allow to evaluate $\mathsf{PRF}_K$ *everywhere* except on $X^*$. For instance, in our case, $\mathcal{A}$ could ask for all keys $K_{S_i}$ with $S_i = \{X = (x_i)_i \mid x_i = 1 - \overline{x}^*_i\}$. Hence, in this case, the simulation *must* fail already whenever $|\mathcal{C}| \geq 2$.

[5]However, for CRPFs with simply structured constraining sets (such as left-right CPRFs, for which the constraining sets $S$ are closed under component-wise hashing), a variation of this strategy may already work [6].

that $\mathcal{T}$ tests for can be extended by another known set $\mathsf{Z}'$.) We instantiate extensible testers using the (non-multilinear) decisional Diffie-Hellman assumption.

We use extensible testers to encode for which tags $\overline{T}_X$ evaluation should fail. (We give details on how this is done below, after presenting our construction.) This way, these imperfections $\overline{T}_X$ are hidden and can be introduced or modified without changing $\mathcal{A}$'s view noticeably.

**Putting things together.** We obtain a CPRF that in fact does not look so different from the bit-fixing CPRF from [6]. Our CPRF is of the form $\mathsf{PRF}_K(X) = \mathsf{PRF}'_K(X, \mathsf{H}(X))$, where we interpret $\mathsf{H}(X)$ as a vector $T = (T_i)_{i=1}^d$ of $k$-bitstrings $T_i$. (Hence, a tag $T$ is actually a vector of $d$ bitstrings, and each bitstring is of length $k$.) The function $\mathsf{PRF}'_K$ is defined through

$$\mathsf{PRF}'_K(X, (T_i)_{i=1}^k) = e(g, \dots, g)^{\prod_{i=1}^{\ell} \prod_{j=1}^{d} \alpha_{i,x_i,j,t_{i,j}}} \qquad (\text{for } K = (\alpha_{i,b,j,c})_{i,b,j,c} \text{ and } T_i = (t_{i,j})_{j=1}^k), \tag{2}$$

where $e$ is an $(\ell d - 1)$-linear map. In the scheme, a constrained key for a subset $S$ of inputs consists of an obfuscation of the function $\mathsf{PRF}'_K(\cdot, \cdot)$ with hardwired $\alpha_{i,b,j,c}$, but restricted to inputs $X \in S$. (That is, on inputs $(X, T)$ with $X \notin S$, the obfuscated function outputs $\perp$.) In the simulation, we will first add imperfections to each generated constrained key. Then, we will replace the obfuscated keys with functionally identical keys that only require knowledge of a subset of the $\alpha_{i,b,j,c}$. This way, we end up with a simulation that does not know the $\alpha_{i,b,j,c}$ necessary to compute $\mathsf{PRF}_K(X^*) = \mathsf{PRF}'_K(X^*, \overline{T}_{X^*})$.

**A more technical explanation.** To get a glimpse of the argument used in the reduction, suppose that for every $i, b, j$, we know only either $\alpha_{i,b,j,0}$ or $\alpha_{i,b,j,1}$ (but not both) in plain. We also know all $\alpha_{i,b,j,c}$ "in the exponent" (i.e., as $g^{\alpha_{i,b,j,c}}$). This allows to evaluate $\mathsf{PRF}'_K(X, T)$ for all $X, T$, *except* when $T = \overline{T}_X$ for a single vector $\overline{T}_X$ that reflects which $\alpha_{i,b,j,c}$ we *don't* know. (In this case, the product $\prod_{i,j} \alpha_{i,x_i,j,t_{i,j}}$ in the exponent of (2) contains *no* known factor.) Observe that the $i$-th component of this vector $\overline{T}_X$ depends only on $x_i$; hence, we can write $\overline{T}_X = (\overline{T}_{i,x_i})_i$ for bitstrings $\overline{T}_{i,b}$ that do not depend on $X$.

Now suppose we want to generate a constrained key that allows to evaluate $\mathsf{PRF}'_K(\cdot, \cdot)$ with some bits of the first input $X$ fixed to certain values. More formally, the key should evaluate $\mathsf{PRF}'_K(X, T)$ whenever $\overline{x}_i \in \{x_i, \perp\}$ for a given partial assignment $\overline{X} = (\overline{x}_i)_i \in \{0, 1, \perp\}^d$. However, by our setup, we can only expect to evaluate $\mathsf{PRF}'_K(X, T)$ whenever $T_i \neq \overline{T}_{i,x_i}$ for all $i$. Consequently, all generated constrained keys will fail on inputs $(X, T)$ with $T = \overline{T}_X = (\overline{T}_{i,x_i})_i$, unlike constrained keys in the original scheme. These failures introduced by our setup are the "imperfections" mentioned above.

We must hence solve two tasks: to bridge our simulation and the original scheme, we must introduce these same imperfections into honestly generated keys. Besides, we must make sure that $\mathsf{H}(X^*) = \overline{T}_{X^*}$ for $\mathcal{A}$'s challenge input $X^*$ to embed a computational challenge into the computation of $\mathsf{PRF}_K(X^*)$. We first describe how to introduce imperfections into honestly generated constrained keys. Namely, we start with implanting an extensible tester $\mathcal{T}$ for the set $\mathsf{Z} = \{\overline{T}_{i,\overline{x}_i}\}_{\overline{x}_i \neq \perp}$ into each constrained key for a bitmask $\overline{X} = (\overline{x}_i)_i$. Furthermore, we program this constrained key such that the evaluation of $\mathsf{PRF}'_K(X, T)$ fails if $\mathcal{T}(\{T_i\}_{\overline{x}_i \neq \perp}) = 1$. In other words, evaluation fails as soon as the input *could be* problematic in case of a setup of the $\alpha_{i,b,j,c}$ as in the simulation. During this step, we assume knowledge of all $\alpha_{i,b,j,c}$, and hide the values $\overline{T}_{i,b}$ even from the simulation. (At this point, we use the extensibility of testers to introduce the $\overline{T}_{i,b}$ consistently across all generated constrained keys.)

However, to embed a computational challenge, we must also program the random oracle $\mathsf{H}$ such that $\mathsf{H}(X^*) = \overline{T}_{X^*}$. This programming in itself is easy to perform, but requires knowledge of $\overline{T}_{X^*} = (\overline{T}_{i,x_i^*})_i$. Hence, introducing imperfections as above becomes much harder with a programmed $\mathsf{H}(X^*)$. We postpone a more technical description of the difficulties and our solution strategy (which heavily involves the extensibility of testers) to the proof of Lemma 4.2.

**More on related techniques.** Our use of obfuscated keys with imperfections resembles the "punctured programs" technique of [41], however with a twist: unlike [41], we also hide *where* the puncturing takes place, even from the simulation itself. Furthermore, tags that introduce isolated "points of failure" have already been used in several works (e.g., [32, 43, 10]). Finally, many works employ random oracles to introduce adaptivity into a simulation, e.g., [17, 26, 4, 39]. We view our contribution primarily as a new *combination* (of course with twists and adaptations, as sketched above) of these concepts.

**Note on a related work of Fuchsbauer et al.** Independently and concurrently, Fuchsbauer et al. [20] have also investigated the adaptive security of constrained PRFs. They provide two main results: first, they show an exponential lower bound for the reduction loss of fully secure bit-fixing CPRFs (and thus provide a formal foundation for our intuitive reasoning given above under "Why achieving full security is difficult"). Our result circumvents their lower bound by the use of a random oracle; indeed, to a certain extent, we believe that their lower bound justifies our use of random oracles. [20] also show that the classic Goldreich-Goldwasser-Micali (GGM) PRF [25] actually is a fully secure prefix-fixing CPRF (i.e., a CPRF with keys that fix an arbitrary prefix of the input bitstring), although with a quasi-polynomial security reduction. Compared to our result, their result provides a weaker notion of CPRFs with an asymptotically worse security reduction, but under a significantly weaker computational assumption, and in the standard model. (Their technique to show adaptive security is entirely different from ours: they use a clever combination of the classic GGM analysis with the prefix-guessing technique of [38, 30, 9].)

**Outline.** In Section 2, we fix some notation and recall some definitions (including a definition of constrained PRFs). In Section 3, we define and construct extensible testers, as discussed above. We present and analyze our constrained PRF in Section 4. In Section 5, we discuss the compatibility of our construction with the recent candidates of "approximate" multilinear maps from [21, 14]. Finally, in Section 6, we give details on our non-interactive key exchange application.

## 2 Preliminaries

**Generic notation.** For $n \in \mathbb{N}$, let $[n] := \{1, \dots, n\}$. Throughout the paper, $k \in \mathbb{N}$ is the security parameter. For a finite set $S$, $s \leftarrow S$ denotes the process of sampling $s$ uniformly from $S$. For a probabilistic algorithm $\mathcal{A}$, $y \leftarrow \mathcal{A}(x)$ denotes the process of running $A$ on input $x$ and with randomness $R$, and assigning $y$ the result. If $\mathcal{A}$'s running time is polynomial in $k$, $\mathcal{A}$ is called probabilistic polynomial-time (PPT). A function $f : \mathbb{N} \to \mathbb{R}$ is negligible if it vanishes faster than the inverse of any polynomial (i.e., if $\forall c \exists k_0 \forall k \geq k_0 : |f(x)| \leq 1/k^c$). Two sequences $X = (X_k)_{k\in\mathbb{N}}$ and $Y = (Y_k)_{k\in\mathbb{N}}$ of random variables are computationally indistinguishable (written $X \overset{c}{\approx} Y$) if $\Pr[\mathcal{A}(X) = 1] - \Pr[\mathcal{A}(Y) = 1]$ is negligible for every PPT $\mathcal{A}$. We write $X \overset{s}{\approx}_\mu Y$ for a function $\mu : \mathbb{N} \to \mathbb{R}$ if the statistical distance between $X_k$ and $Y_k$ is at most $\mu(k)$ for all $k$. We may simply write $X \overset{s}{\approx} Y$ if $X \overset{s}{\approx}_\mu Y$ for some negligible function $\mu$. Finally, if $X_k$ and $Y_k$ are identically distributed for every $k$(i.e., if $X \overset{s}{\approx}_0 Y$), then we write $X \equiv Y$.

**Groups with multilinear maps.** Our constructions make use of multilinear maps. While we formulate our constructions with respect to the standard mathematical notion of multilinear maps (to be defined below), we stress that they can be implemented with the recent candidates [21, 14] of *approximate* multilinear maps. (We will give more details in Section 5.)

Formally, an $D$-linear setting $\mathcal{G} := ((\mathbb{G}_i)_i, g, p, (e_{i,j})_{i,j})$ consists of cyclic groups $\mathbb{G}_1, \dots, \mathbb{G}_D$ of prime order $p$, and a generator $g$ with $\mathbb{G}_1 = \langle g \rangle$. We also assume bilinear maps $e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j}$ for any $i, j \geq 1$ with $i + j \leq D$. We insist that the $e_{i,j}$ are non-degenerate in the sense that $e_{i,j}(g_i, g_j) \neq 1$ for any $g_i \in \mathbb{G}_i, g_j \in \mathbb{G}_j$ with $g_i, g_j \neq 1$. In fact, if we let $g_1 := g$ and $g_{i+1} := e_{1,i-1}(g, g_{i-1}) \in \mathbb{G}_i$, then this implies $\mathbb{G}_i = \langle g_i \rangle$. This setting $\mathcal{G}$ may depend on the security parameter, and we assume that it is publicly known. For instance, $\mathcal{G}$ may be chosen upon key generation in our constrained PRF, and can be implicitly contained in all keys (constrained or not) handed out.

**Bracket notation.** To simplify the exposition, we will write $[x]_i$ instead of $g_i^x \in \mathbb{G}_i$ for any $x \in \mathbb{Z}_p$ and the generators $g_i$ as defined above. We will also write $[x]_i \cdot [y]_j := e_{i,j}([x]_i, [y]_j)$, so that we have $[x]_i \cdot [x]_j = [x \cdot y]_{i+j}$. Occasionally, we will also write $\alpha \cdot [x]_i := [\alpha \cdot x]_i$ for $\alpha \in \mathbb{Z}_p$. Finally, to avoid the overuse of braces in vectors of group elements, we will abbreviate $([x_1]_i, \ldots, [x_n]_i)$ with $[x_1, \ldots, x_n]_i$. This notation will help to express "computations in the exponent" in a more direct and intuitive fashion. (We note that a similar notation has been used in [21, 14, 16].)

**Multilinear decisional Diffie-Hellman problem.** We use the following popular multilinear variant of the decisional Diffie-Hellman problem:

**Definition 2.1** ($D$-MDDH)**.** *We say that the D-linear decisional Diffie-Hellman (short: D-MDDH) assumption holds in $\mathcal{G}$ if the following advantage function is negligible for all PPT adversaries $\mathcal{A}$:*

$$\mathsf{Adv}^{\mathsf{mddh}}_{\mathcal{G},D,\mathcal{A}}(k) = \Pr\left[\mathcal{A}([x_1]_1, \ldots, [x_{D+1}]_1, \left[\prod_{i=1}^{D+1} x_i\right]_D) = 1\right] - \Pr\left[\mathcal{A}([x_1]_1, \ldots, [x_{D+1}]_1, [z]_D) = 1\right].$$

*Here, the probability is over $x_1, \ldots, x_{D+1}, z \leftarrow \mathbb{Z}_p$ and $\mathcal{A}$'s randomness.*

**In groups without multilinear map.** We will also our notation in groups $\mathbb{G}$ without multilinear map. (Since in that case, there is only one group, we will drop the subscript from our bracket notation; that is, we will write $[x] := g^x \in \mathbb{G}$.) The common decisional Diffie-Hellman assumption then arises as a special case of Definition 2.1:

**Definition 2.2** (DDH)**.** *We say that the decisional Diffie-Hellman (short: DDH) assumption holds in $\mathbb{G}$ if the following advantage function is negligible for all PPT adversaries $\mathcal{A}$:*

$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},\mathcal{A}}(k) = \Pr[\mathcal{A}([x, y], [x \cdot y]) = 1] - \Pr[\mathcal{A}([x, y], [z]) = 1].$$

*The probability is over $x, y, z \leftarrow \mathbb{Z}_p$ and $\mathcal{A}$'s randomness.*

**Indistinguishability obfuscation.** Intuitively, an indistinguishability obfuscator transforms a circuit $C$ into one which (up to computational indistinguishability) depends only on the functionality of $C$:

**Definition 2.3** (Indistinguishability obfuscator)**.** *An indistinguishability obfuscator $i\mathcal{O}$ is a PPT algorithm that, on input $1^k$ and a circuit $C$, outputs an obfuscated circuit $i\mathcal{O}(C)$. (We usually drop the first input $1^k$ for readability.) We require that for every PPT adversary $\mathcal{A}$, the advantage function*

$$\mathsf{Adv}^{\mathsf{ind\text{-}obf}}_{i\mathcal{O},\mathcal{A}}(k) = \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}obf}}_{i\mathcal{O},\mathcal{A}}(k) = 1\right] - 1/2$$

*is negligible, where the experiment $\mathsf{Exp}^{\mathsf{ind\text{-}obf}}_{i\mathcal{O},\mathcal{A}}$ is defined as follows:*
*1. $\mathcal{A}$ chooses two functionally identical circuits $C_0, C_1$ of equal size,*
*2. the experiment tosses a coin $b \leftarrow \{0, 1\}$, and sends $i\mathcal{O}(C_b)$ to $\mathcal{A}$,*
*3. $\mathcal{A}$ outputs a guess $b'$ for $b$, and the experiments outputs $1$ if and only if $b' = b$.*

Although formalized already in [1] (and further investigated in [27]), indistinguishability obfuscators could only be constructed very recently [22] (under a case-tailored computational assumption). However, since then, significant research on the applications [22, 41, 29, 23, 7, 34] and limitations [2, 37] of indistinguishability obfuscation has been conducted.

**Constrained and bit-fixing pseudorandom functions.** The concept of constrained pseudorandom functions (CPRFs) has been formalized in several concurrent works [33, 6, 8]. In a nutshell, CPRFs are pseudorandom functions whose keys can be "constrained" to a subset of preimages:

**Definition 2.4** (CPRF). *A constrained pseudorandom function (CPRF)* $\mathsf{PRF}$ *with input length* $\ell = \ell(k)$*, output length* $\ell' = \ell'(k)$*, and for a system* $\mathcal{S} \subseteq \{0,1\}^{\ell'}$ *of sets, comprises three PPT algorithms:*

**Key generation.** $\mathsf{KGen}$ *samples a key* $K$*.*

**Delegation.** $\mathsf{KDel}(K, S)$*, for a key* $K$ *and a set* $S \in \mathcal{S}$*, outputs a constrained key* $K_S$*.*

**Evaluation.** $\mathsf{KEval}(K_S, X)$*, for a constrained key* $K_S$ *and a preimage* $X \in S$*, outputs an image* $\mathsf{PRF}_K(X) \in \{0,1\}^{\ell'}$*. We require that this image only depends on* $K$ *and* $X$ *(but not on* $S$*).*

*We require that for all PPT distinguishers* $\mathcal{A}$*, the advantage function*

$$\mathsf{Adv}^{\mathsf{cprf}}_{\mathsf{PRF},\mathcal{A}}(k) = \Pr\left[\mathsf{Exp}^{\mathsf{cprf}}_{\mathsf{PRF},\mathcal{A}}(k) = 1\right] - 1/2$$

*is negligible, where the experiment* $\mathsf{Exp}^{\mathsf{cprf}}_{\mathsf{PRF},\mathcal{A}}$ *is defined as follows:*

1. *the experiment samples* $K \leftarrow \mathsf{KGen}$ *and gives* $\mathcal{A}$ *oracle access to* $\mathsf{KDel}(K, \cdot)$ *at any point,*
2. $\mathcal{A}$ *decides to be challenged on a preimage* $X^* = (x_i^*)_{i=1}^{\ell} \in \{0,1\}^{\ell}$ *that is not contained in any set* $S$ *for which* $\mathcal{A}$ *has previously queried* $\mathsf{KDel}$*,*
3. *the experiment tosses a coin* $b \leftarrow \{0,1\}$*; if* $b = 0$*, then* $\mathcal{A}$ *gets* $\mathsf{PRF}_K(X^*)$*, and if* $b = 1$*, then* $\mathcal{A}$ *gets a uniformly chosen* $\{0,1\}^{\ell'}$*-element,*
4. *finally, after potentially further querying* $\mathsf{KDel}(K, \cdot)$ *(but only on sets* $S$ *which do not contain* $X^*$*),* $\mathcal{A}$ *outputs a guess* $b'$ *for* $b$*, and the experiment outputs* $1$ *if and only if* $b' = b$*.*

In this work, we will focus on bit-fixing PRFs, for which $\mathcal{S}$ consists of all sets $S_{\overline{X}}$ (for $\overline{X} = (\overline{x}_i)_{i=1}^{\ell} \in (\{0,1\} \cup \{\perp\})^{\ell}$), where

$$S_{\overline{X}} = \left\{ X = (x_i)_{i=1}^{\ell} \mid \forall i : x_i = \overline{x}_i \text{ or } \overline{x}_i = \perp \right\}.$$

In other words, a set $S_{\overline{X}}$ contains all preimages $X$ that match the partial assignment $\overline{X}$. In the following, all CPRFs will be bit-fixing CPRFs, and we will write $\mathsf{KDel}(K, \overline{X})$ instead for $\mathsf{KDel}(K, S_{\overline{X}})$ for brevity.

## 3 Extensible testers

### 3.1 Definition

Intuitively, an extensible tester allows to obfuscate a multiset[6] $\mathsf{Z}$ over some domain $\mathcal{X}$. Slightly abusing language, we will call the obfuscation $\mathcal{T}_{\mathsf{Z}}$ of $\mathsf{Z}$ itself the "tester." We require that it is possible to extend a tester $\mathcal{T}_{\mathsf{Z}}$ for $\mathsf{Z}$ to a tester $\mathcal{T}_{\mathsf{Z} \cup \mathsf{Z}'}$ for a larger multiset $\mathsf{Z} \cup \mathsf{Z}'$, knowing only $\mathcal{T}_{\mathsf{Z}}$ and $\mathsf{Z}'$ (but not $\mathsf{Z}$). Besides, we want that a tester $\mathcal{T}_{\mathsf{Z}}$ does not reveal the multiset $\mathsf{Z}$ it obfuscates, in a sense to be defined. Finally, for technical reasons, we also require the existence of a special tester $\mathcal{T}_{\perp}$ that cannot be extended (in the sense that the extension of $\mathcal{T}_{\perp}$ only yields a re-randomized copy of $\mathcal{T}_{\perp}$ again) and matches no multiset. Formally:

**Definition 3.1** (Extensible tester). *Let* $\mathcal{X}$ *be an efficiently samplable domain (that may depend on the security parameter* $k$*), and let* $\mu_{\mathsf{ET}} = \mu_{\mathsf{ET}}(k)$ *be a real-valued function. An extensible tester* $\mathsf{ET}$ *over* $\mathcal{X}$ *with statistical defect* $\mu_{\mathsf{ET}}$ *consists of the following PPT algorithms:*

- $\mathsf{Mark}$ *takes either* $\perp$ *or a multiset* $\mathsf{Z}$ *over* $\mathcal{X}$ *as input, and outputs a tester* $\mathcal{T}$*.*
- $\mathsf{Test}$ *takes a tester* $\mathcal{T}$ *and a multiset* $\mathsf{Z}$ *over* $\mathcal{X}$ *as input and outputs a verdict* $b \in \{0,1\}$*.*
- $\mathsf{Extend}$ *takes a tester* $\mathcal{T}$ *and a multiset* $\mathsf{Z}'$ *over* $\mathcal{X}$ *as input and outputs a tester* $\mathcal{T}'$*.*

*We require the following:*

**Correctness.** *For all* $k, \mathsf{Z}$*, we have* $\mathsf{Test}(\mathsf{Mark}(\mathsf{Z}), \mathsf{Z}) = 1$ *always.*

[6]Recall that a multiset is a set in which elements can occur more than once. In the following, all multisets will be implicitly assumed to be finite.

**Extensibility.** *For all $k, \mathsf{Z}, \mathsf{Z}'$, we have*

$$\mathsf{Extend}(\mathsf{Mark}(\mathsf{Z}), \mathsf{Z}') \overset{s}{\approx}_{\mu_{\mathsf{ET}}} \mathsf{Mark}(\mathsf{Z} \cup \mathsf{Z}')$$
$$\mathsf{Extend}(\mathsf{Mark}(\perp), \mathsf{Z}) \overset{s}{\approx}_{\mu_{\mathsf{ET}}} \mathsf{Mark}(\perp).$$

**Soundness.** *For all $k$ and all $\mathsf{Z} \neq \emptyset$, we have* $\mathsf{Test}(\mathsf{Mark}(\perp), \mathsf{Z}) = 0$ *always.*
**Indistinguishability.** *For every PPT adversary $\mathcal{A}$ and independently uniform $\mathsf{z} \leftarrow \mathcal{X}$,*

$$\mathsf{Adv}^{\mathsf{et\text{-}ind}}_{\mathsf{ET},\mathcal{A}}(k) := \Pr[\mathcal{A}(\mathsf{Mark}(\perp)) = 1] - \Pr[\mathcal{A}(\mathsf{Mark}(\{\mathsf{z}\})) = 1]$$

*is negligible.*

Note that the soundness property is rather weak: we only exclude false positives for the "empty tester" $\mathcal{T}_\perp$. However, together with our indistinguishability and extensibility requirements, actually false positives for any multiset that contains at least one random element are excluded, at least in a computational sense. Specifically, we will need the following:

**Lemma 3.2.** *An extensible tester* $\mathsf{ET}$ *in the sense of Definition 3.1 also satisfies the following properties for independently uniform* $\mathsf{z}_1, \mathsf{z}_2, \mathsf{z}_3 \leftarrow \mathcal{X}$*:*

$$(\mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{Mark}(\{\mathsf{z}_1\})) \overset{c}{\approx} (\mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{Mark}(\{\mathsf{z}_2\})) \tag{3}$$
$$(\mathsf{Mark}(\{\mathsf{z}_1, \mathsf{z}_2\}), \mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{z}_2) \overset{c}{\approx} (\mathsf{Mark}(\{\mathsf{z}_1, \mathsf{z}_2\}), \mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{z}_3). \tag{4}$$

*Proof.* To show (3), consider

$$\begin{aligned}(\mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{Mark}(\{\mathsf{z}_1\})) &\overset{s}{\approx} (\mathcal{T}_{\{\mathsf{z}_1\}}, \mathsf{Extend}(\mathcal{T}_{\{\mathsf{z}_1\}}, \emptyset)) \overset{c}{\approx} (\mathcal{T}_\perp, \mathsf{Extend}(\mathcal{T}_\perp, \emptyset)) \\ &\overset{s}{\approx} (\mathsf{Mark}(\perp), \mathsf{Mark}(\perp)) \overset{c}{\approx} (\mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{Mark}(\perp)) \overset{c}{\approx} (\mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{Mark}(\{\mathsf{z}_2\}))\end{aligned}$$

for $\mathcal{T}_{\{\mathsf{z}_1\}} \leftarrow \mathsf{Mark}(\{\mathsf{z}_1\})$ and $\mathcal{T}_\perp \leftarrow \mathsf{Mark}(\{\perp\})$. Here, the "$\overset{c}{\approx}$"s follow from indistinguishability, and the "$\overset{s}{\approx}$"s from extensibility of $\mathsf{ET}$. Similarly, (4) can be seen as follows:

$$\begin{aligned}(\mathsf{Mark}(\{\mathsf{z}_1, \mathsf{z}_2\}), \mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{z}_2) &\overset{s}{\approx} (\mathsf{Extend}(\mathcal{T}_{\{\mathsf{z}_1\}}, \{\mathsf{z}_2\}), \mathcal{T}_{\{\mathsf{z}_1\}}, \mathsf{z}_2) \overset{c}{\approx} (\mathsf{Extend}(\mathcal{T}_\perp, \{\mathsf{z}_2\}), \mathcal{T}_\perp, \mathsf{z}_2) \\ &\overset{s}{\approx} (\mathsf{Mark}(\perp), \mathsf{Mark}(\perp), \mathsf{z}_2) \overset{s}{\approx} (\mathsf{Mark}(\perp), \mathsf{Mark}(\perp), \mathsf{z}_3) \overset{s}{\approx} (\mathsf{Extend}(\mathcal{T}_\perp, \{\mathsf{z}_2\}), \mathcal{T}_\perp, \mathsf{z}_3) \\ &\overset{c}{\approx} (\mathsf{Extend}(\mathcal{T}_{\{\mathsf{z}_1\}}, \{\mathsf{z}_2\}), \mathcal{T}_{\{\mathsf{z}_1\}}, \mathsf{z}_3) \overset{s}{\approx} (\mathsf{Mark}(\{\mathsf{z}_1, \mathsf{z}_2\}), \mathsf{Mark}(\{\mathsf{z}_1\}), \mathsf{z}_3),\end{aligned}$$

where again $\mathcal{T}_{\{\mathsf{z}_1\}} \leftarrow \mathsf{Mark}(\{\mathsf{z}_1\})$ and $\mathcal{T}_\perp \leftarrow \mathsf{Mark}(\{\perp\})$. □

## 3.2 Construction from DDH

Our construction of extensible testers is a variant of the DDH-based point function obfuscation of Canetti [11]. (Concretely, we use two instances of the [11]-obfuscation in parallel, to enable the construction of "illegal" testers $\mathcal{T}_\perp$ that do not match anything.) We assume a public group $\mathbb{G} = \langle g \rangle$ of prime order $p$. As already mention in Section 2, we will write $[x]$ for $g^x$ for consistency, and to simplify notation. We now present our extensible tester $\mathsf{ET}$ over $\mathbb{Z}_p^*$.

**Marking and testing.** We start with our marking and testing algorithms:

- $\mathsf{Mark}(\perp)$ outputs $\mathcal{T}_\perp = [r, rx_0, s, sx_1]$ for uniform $r, s, x_0, x_1 \leftarrow \mathbb{Z}_p^*$ subject to $x_0 \neq x_1$.
- $\mathsf{Mark}(\mathsf{Z})$ outputs $\mathcal{T}_\mathsf{Z} = [r, rx, s, sx]$ for uniform $r, s \leftarrow \mathbb{Z}_p^*$ and $x = \prod_{\mathsf{z} \in \mathsf{Z}} \mathsf{z}$.
- $\mathsf{Test}(\mathcal{T}, \mathsf{Z})$, for $\mathcal{T} = [r, u, s, v]$, outputs 1 iff $[u] = x \cdot [r]$ and $[v] = x \cdot [s]$ for $x = \prod_{\mathsf{z} \in \mathsf{Z}} \mathsf{z}$.

Now when trying to extend a tester $\mathcal{T} = [r, u, s, v]$ by a multiset $\mathsf{Z}'$, a first attempt could be to compute $\mathcal{T}' = ([r], x'[u], [s], x'[v])$ for $x' = \prod_{\mathsf{z}' \in \mathsf{Z}'} \mathsf{z}'$. This would indeed yield a tester for $\mathsf{Z} \cup \mathsf{Z}'$ (resp. $\perp$) if $\mathcal{T}$ was a tester for $\mathsf{Z}$ (resp. $\perp$). However, the resulting $\mathcal{T}'$ would not be *fresh* (i.e., independently distributed). Hence, to describe our actual extension algorithm, we need a means to re-randomize testers.

**Re-randomizing testers.** Algorithm $\mathsf{reRand}(\mathcal{T})$, for $\mathcal{T} = [r, u, s, v]$, computes and outputs

$$\mathcal{T}' = (\alpha[r] + \beta[s], \alpha[u] + \beta[v], \gamma[r] + \delta[s], \gamma[u] + \delta[v]) = [\alpha r + \beta s, \alpha u + \beta v, \gamma r + \delta s, \gamma u + \delta v]$$

for uniform $\alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_p$ subject to $\alpha r + \beta s \neq 0$ and $\gamma r + \delta s \neq 0$. (Note that although $\mathsf{reRand}$ does not have access to $r$ and $s$ in plain, these events can be tested for with $[r]$ and $[s]$ only. Hence, $\mathsf{reRand}$ can choose such $\alpha, \beta, \gamma, \delta$.) The following lemma will be instrumental in analyzing $\mathsf{reRand}$:

**Lemma 3.3.** *Fix any $r, s, x_0, x_1 \in \mathbb{Z}_p^*$. For uniform $\alpha, \beta \in \mathbb{Z}_p$ conditioned on $\alpha r + \beta s \neq 0$, consider the elements $(t, w) := (\alpha r + \beta s, \alpha r x_0 + \beta s x_1) \in \mathbb{Z}_p^2$. We have:*
*(a) if $x_0 \neq x_1$, then $t$ and $w$ are independently uniform $\mathbb{Z}_p^*$-elements,*
*(b) if $x_0 = x_1$, then $w = (t \cdot x_0 \bmod p)$ for independently uniform $t \in \mathbb{Z}_p^*$.*

*Proof.* In both cases, $t \in \mathbb{Z}_p^*$ is uniform by assumption. For the first claim, observe that we can write $x_1 = x_0 + \Delta$ (for $\Delta \neq 0$) and thus $w = (\alpha r + \beta s)x_0 + \beta s \Delta$. But even conditioning on $u = \alpha r + \beta s$, the values of $\beta$ and thus also $w$ is uniform. The second claim is immediate. □

**The extension algorithm.** Re-randomizing the simple extension procedure sketched above thus yields:

- $\mathsf{Extend}(\mathcal{T}, \mathsf{Z}')$, for $\mathcal{T} = [r, u, s, v]$, outputs $\mathsf{reRand}([r], x'[u], [s], x'[v])$ for $x' = \prod_{\mathsf{z}' \in \mathsf{Z}'} \mathsf{z}'$.

**Theorem 3.4** (DDH-based extensible tester)**.** *Under the DDH assumption in $\mathbb{G}$, the construction $\mathsf{ET}$ above constitutes an extensible tester over $\mathbb{Z}_p^*$ with statistical defect $1/|\mathbb{G}|$ in the sense of Definition 3.1.*

*Proof.* We need to check the properties from Definition 3.1. Correctness is clear from the construction, and extensibility follows from Lemma 3.3: for any tester $\mathcal{T}$ for $\perp$, Lemma 3.3 (a) implies that $\mathsf{reRand}(\mathsf{Z})$ is uniformly distributed over $(\mathbb{G} \setminus \{1\})^4$. Thus, $\mathsf{reRand}(\mathsf{Z})$ is a fresh tester for $\perp$, except with probability $1/p$. But if $\mathcal{T}$ is a tester for $\mathsf{Z}$, Lemma 3.3 (b) yields that $\mathsf{reRand}(\mathcal{T})$ is of the form $[r, rx, s, sx]$ for fresh $r, s$, and $x = \prod_{\mathsf{z} \in \mathsf{Z}} \mathsf{z}$. Hence, $\mathsf{reRand}(\mathcal{T})$ is a fresh tester for $\mathsf{Z}$.

For soundness, consider a tester $\mathcal{T}_\perp = [r, rx_0, s, sx_1]$ as output by $\mathsf{Mark}(\perp)$. Since $x_0 \neq x_1$, there is no $x$ with $[u] = x \cdot [r]$ and $[v] = x \cdot [s]$. Hence, $\mathsf{Test}(\mathcal{T}_\perp, \mathsf{Z})$ will always output 0, independently of $\mathsf{Z}$.

To show indistinguishability, assume a distinguisher $\mathcal{D}$ between $\mathsf{Mark}(\perp)$ and $\mathsf{Mark}(\{\mathsf{z}\})$ (for uniform $\mathsf{z} \in \mathbb{Z}_p^*$). We construct a DDH distinguisher $\mathcal{D}'$ that distinguishes between $[r, x, rx]$ and $[r, x, y]$ (for uniform $r, x, y \leftarrow \mathbb{Z}_p^*$). Concretely, $\mathcal{D}'([r, x, z])$ sets up

$$\mathcal{T} = \mathsf{reRand}([r], [z], [s], s \cdot [x]) = \mathsf{reRand}([r, z, 1, x]).$$

and outputs whatever $\mathcal{D}(\mathcal{T})$ outputs. Now $\mathcal{T}$ is distributed statistically close to $\mathsf{Mark}(\perp)$ or $\mathsf{Mark}(\{x\})$, depending on whether $z$ is random or $z = rx$. Thus, $\mathcal{D}'$ is a successful DDH-distinguisher whenever $\mathcal{D}$ successfully distinguishes $\mathsf{Mark}(\perp)$ and $\mathsf{Mark}(\{\mathsf{z}\})$. □

## 4 Our constrained PRF

### 4.1 Construction

We are now ready to define our constrained PRF, where input length $\ell = \ell(k)$ and output length $\ell' = \ell'(k)$ may be arbitrary efficiently computable and polynomially bounded functions.

**Ingredients.** We assume the following ingredients:

- We make use of an indistinguishability obfuscator $i\mathcal{O}$ in the sense of Definition 2.3. In what follows, we implicitly assume a suitable circuit representation of obfuscated functions. We also assume a padding of these circuits as necessary to guarantee that the circuits obfuscated in the scheme and in the prove have the same size. (It will be clear that this is always possible.)
- We use an extensible tester $\mathsf{ET}$ over a domain $\mathcal{X}$ in the sense of Definition 3.1. In the following, we will use the bit representation of elements $S \in \mathcal{X}$ and may write $S = (\bar{t}_i)_{i=1}^{d} \in \{0,1\}^d$ for an appropriate $d$ (that may of course depend on the security parameter $k$).
- We assume an $D$-linear setting $\mathcal{G}$ as in Section 2 for $D = \ell d - 1$.
- Finally, we assume a family $\mathcal{H}$ of key derivation functions $h : \mathbb{G}_D \to \{0,1\}^{\ell'}$. We require that $((h, h(g))$ has statistical distance at most $\varepsilon_{\mathcal{H}}$ from $(h, r)$, where $h \leftarrow \mathcal{H}$, $g \leftarrow \mathbb{G}_D$, and $r \leftarrow \{0,1\}^{\ell'}$ are uniformly chosen.

**Key generation.** $\mathsf{KGen}$ chooses $4\ell d$ random exponents $\alpha_{i,b,j,c}$ for $(i,b,j,c) \in [\ell] \times \{0,1\} \times [d] \times \{0,1\}$, as well as $h \leftarrow \mathcal{H}$. Output of $\mathsf{KGen}$ is

$$K = ((\alpha_{i,b,j,c})_{i,b,j,c \in [\ell]\times\{0,1\}\times[d]\times\{0,1\}}, h).$$

**Images.** For a key $K$ as above and an input $X = (x_i)_{i=1}^{\ell} \in \{0,1\}^{\ell}$, we define

$$\mathsf{PRF}_K(X) = h\left(\left[\prod_{(i,j)\in[\ell]\times[d]} \alpha_{i,x_i,j,t_{i,j}}\right]_{\ell d-1}\right)$$

for $\mathsf{H}(X) = (T_i)_{i=1}^{\ell} \in \mathcal{X}^{\ell}$ and the bit representation $T_i = (t_{i,j})_{j=1}^{k} \in \{0,1\}^d$.

**Delegation.** For convenience, we say that a complete assignment $X = (x_i)_{i=1}^{\ell} \in \{0,1\}^{\ell}$ *matches* a partial assignment $\overline{X} = (\overline{x}_i)_{i=1}^{\ell} \in (\{0,1\} \cup \{\perp\})^{\ell}$ if and only if $x_i = \overline{x}_i$ for all $i$ with $\overline{x}_i \neq \perp$. Now $\mathsf{KDel}(K, \overline{X})$, for a partial assignment $\overline{X} = (\overline{x}_i)_{i=1}^{\ell}$, outputs

$$K_{\overline{X}} = i\mathcal{O}(G_{K,\overline{X}}),$$

where the function $G_{K,\overline{X}}$ is defined through

$$G_{K,\overline{X}}(X,T) = \begin{cases} h\left(\left[\prod_{i,j} \alpha_{i,x_i,j,t_{i,j}}\right]_{\ell d-1}\right) & \text{if } X \text{ matches } \overline{X} \\ \perp & \text{else} \end{cases}$$

for $X = (x_i)_i$ and $T = (T_i)_i = ((t_{i,j})_j)_i$ as above. Note that for any complete assignment $X$ that matches $\overline{X}$, we have $G_{K,\overline{X}}(X, \mathsf{H}(X)) = \mathsf{PRF}_K(X)$.

**Evaluation.** $\mathsf{KEval}(i\mathcal{O}(G_{K,\overline{X}}), X)$ outputs $i\mathcal{O}(G_{K,\overline{X}})(X, \mathsf{H}(X)) = G_{K,\overline{X}}(X, \mathsf{H}(X))$.

### 4.2 Security analysis

Correctness of the construction is immediate. We turn to its security analysis.

**Roadmap.** In our (game-based) proof, we start with the interaction of an adversary $\mathcal{A}$ with the constrained PRF $\mathsf{PRF}$ as in Definition 2.4. We follow the strategy outlined in the introduction: our first goal will be to give out "imperfect" partial evaluation keys to $\mathcal{A}$. These keys fail to evaluate a preimage on specific (computationally hidden) "special" tags $T$. We will then program the random oracle such that the tag $T^*$ for the challenge preimage $X^*$ finally selected by $\mathcal{A}$ is special in the above sense. The main difficulty during these steps is to "synchronize" the imperfections in the constrained keys with the programming of $\mathsf{H}(X^*)$. This step (from Game 4 to Game 5) is outsourced into Lemma 4.2.

We then pave the way to a reduction to the MDDH assumption in Game 6. Namely, Game 6 only uses a subset of the key exponents $\alpha_{i,b,j,c}$, as outlined in the introduction. Consequently, this game is only able to hand out imperfect constrained keys, and is not able to compute the challenge image $\mathsf{PRF}_K(X^*)$ on its own. A final reduction to the MDDH assumption then allows to replace $\mathsf{PRF}_K(X^*)$ with a random value. At this point, $\mathcal{A}$ has no advantage in winning the security experiment.

We now turn to a formal proof.

**Theorem 4.1.** *The construction* PRF *from Section 4 is a constrained PRF in the sense of Definition 2.4, assuming that $i\mathcal{O}$ is an indistinguishability obfuscator,* ET *is an extensible tester, and the $(\ell d-1)$-MDDH assumption holds in $\mathcal{G}$. Concretely, for every PPT adversary $\mathcal{A}$, there are adversaries $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, and $\mathcal{F}$ (of roughly the same complexity as $\mathcal{A}$) with*

$$\begin{aligned}\left|\mathsf{Adv}^{\mathsf{cprf}}_{\mathsf{PRF},\mathcal{A}}(k)\right| \;\le\; & q_{\mathsf{key}}\cdot\Big(\left|\mathsf{Adv}^{\mathsf{ind\text{-}obf}}_{i\mathcal{O},\mathcal{B}}(k)\right| + \left|\mathsf{Adv}^{\mathsf{et\text{-}ind}}_{\mathsf{ET},\mathcal{C}}(k)\right| + (1+4\ell(\ell-1)q_{\mathsf{H}}(q_{\mathsf{key}}+5))\cdot\mu_{\mathsf{ET}}\Big) \\ & + q_{\mathsf{H}}\cdot\Big(20\ell(\ell-1)q_{\mathsf{key}}\cdot\left|\mathsf{Adv}^{\mathsf{et\text{-}ind}}_{\mathsf{ET},\mathcal{D}}(k)\right| + q_{\mathsf{key}}\cdot\left|\mathsf{Adv}^{\mathsf{ind\text{-}obf}}_{i\mathcal{O},\mathcal{E}}(k)\right| + \left|\mathsf{Adv}^{\mathsf{mddh}}_{\mathcal{G},\ell d-1,\mathcal{F}}(k)\right| + \varepsilon_{\mathcal{H}}\Big)\;\Big|, \end{aligned} \tag{5}$$

*where $q_{\mathsf{key}} = q_{\mathsf{key}}(k)$ and $q_{\mathsf{H}} = q_{\mathsf{H}}(k)$ bound the number of $\mathcal{A}$'s constrained key and random oracle queries, and $\mu_{\mathsf{ET}} = \mu_{\mathsf{ET}}(k)$ denotes the statistical defect of* ET.

*Proof.* **Game** 1 is the original CPRF game with adversary $\mathcal{A}$. Without loss of generality, we make the following assumptions about $\mathcal{A}$:

- $\mathcal{A}$ always makes exactly $q_{\mathsf{key}} = q_{\mathsf{key}}(k)$ constrained key queries,
- $\mathcal{A}$ always makes exactly $q_{\mathsf{H}} = q_{\mathsf{H}}(k)$ random oracle queries, all of them distinct,
- $\mathcal{A}$ always queries $\mathsf{H}(X^*)$ at some point before asking for a challenge on $X^* = (x_i^*)_{i=1}^{\ell} \in \{0,1\}^{\ell}$.

In the following, let $out_i$ be the final output of Game $i$. (That is, $out_i = 1$ if and only if $\mathcal{A}$ correctly predicts whether the challenge image it receives is real or random.) By construction,

$$\Pr[out_1 = 1] - 1/2 \;=\; \mathsf{Adv}^{\mathsf{cprf}}_{\mathsf{PRF},\mathcal{A}}(k). \tag{6}$$

In **Game** 2, we change the way constrained keys are generated for $\mathcal{A}$. Concretely, instead of $K_{\overline{X}} = i\mathcal{O}(G_{K,\overline{X}})$ (as defined in Section 4 for a query $\overline{X} = (\overline{x}_i)_i$), we compute $K_{\overline{X}} = i\mathcal{O}(G_{K,\overline{X},\mathcal{T}})$) for a freshly chosen $\mathcal{T} \leftarrow \mathsf{Mark}(\perp)$ and the function $G_{K,\overline{X},\mathcal{T}}$ defined as follows:

$$G_{K,\overline{X},\mathcal{T}}(X,T) = \begin{cases} h\left(\left[\prod_{i,j}\alpha_{i,x_i,j,t_{i,j}}\right]_{\ell d-1}\right) & \text{if } X \text{ matches } \overline{X} \textit{ and } \mathsf{Test}(\mathcal{T},\{T_i\}_{\overline{x}_i\neq\perp}) = 0 \\ \perp & \text{else.}\end{cases} \tag{7}$$

Thus, the only difference to $G_{K,\overline{X}}$ is the additional check for $\mathsf{Test}(\mathcal{T},\{T_i\}_{\overline{x}_i\neq\perp}) = 0$. By the soundness property of ET, however, the tester $\mathcal{T}$ will not match anything. Hence, $G_{K,\overline{X}}$ and $G_{K,\overline{X},\mathcal{T}}$ are functionally identical, and the indistinguishability of $i\mathcal{O}$ yields that

$$\Pr[out_2] - \Pr[out_1] \;=\; q_{\mathsf{key}}\cdot\mathsf{Adv}^{\mathsf{ind\text{-}obf}}_{i\mathcal{O},\mathcal{B}}(k) \tag{8}$$

for a suitable $i\mathcal{O}$-distinguisher $\mathcal{B}$, and using a standard hybrid argument.

In **Game** 3, we again change the way constrained keys are generated. Namely, we construct $K_{\overline{X}} = i\mathcal{O}(G_{K,\overline{X},\mathcal{T}})$ as in Game 2, but for

$$\mathcal{T} \leftarrow \mathsf{Mark}(\{R_i\}_{\overline{x}_i\neq\perp})$$

and independently uniform $R_i \leftarrow \mathcal{X}$ chosen freshly for each $\overline{X}$. We claim that a combination of the extensibility and indistinguishability of $\mathsf{ET}$ yields

$$\left|\Pr[out_3] - \Pr[out_2]\right| \;\le\; q_{\mathsf{key}} \cdot \left(\mathsf{Adv}^{\mathsf{et\text{-}ind}}_{\mathsf{ET},\mathcal{C}}(k) + \mu_{\mathsf{ET}}\right). \tag{9}$$

for a suitable $\mathsf{ET}$-distinguisher $\mathcal{C}$. Concretely, $\mathcal{C}$ embeds its own input $\mathcal{T}'$ (where either $\mathcal{T}' \leftarrow \mathsf{Mark}(\perp)$ or $\mathcal{T}' \leftarrow \mathsf{Mark}(\{\mathsf{z}\})$ for random $\mathsf{z}$) into a single constrained key for a partial assignment $\overline{X}$. (We stress that $\overline{X}$ does not have to be known in advance.) Let $i_1, \dots, i_w$ be the indices of the non-$\perp$ positions of $\overline{X}$, i.e., $\{i_1, \dots, i_w\} := \{i \mid \overline{x}_i \neq \perp\}$. Then, $\mathcal{C}$ sets up $K_{\overline{X}} = i\mathcal{O}(G_{K,\overline{X},\mathcal{T}})$ for

$$\mathcal{T} \leftarrow \mathsf{Extend}(\mathcal{T}', \{R_{i_2}, \dots, R_{i_w}\})$$

and independently random $R_{i_j}$. The extensibility of $\mathsf{ET}$ implies that $\mathcal{T}$ has statistical distance at most $\mu_{\mathsf{ET}}$ to $\mathsf{Mark}(\perp)$ (resp. $\mathsf{Mark}(\{R_i\}_{\overline{x}_i \neq \perp})$) if $\mathcal{T}' \leftarrow \mathsf{Mark}(\perp)$ (resp. $\mathcal{T}' \leftarrow \mathsf{Mark}(\{\mathsf{z}\})$, where we implicitly set $\mathsf{z} = R_{i_1}$). Hence, $\mathcal{C}$ can prepare a single constrained key as in Game 2 or Game 3, depending on its own challenge. A standard hybrid argument over all constrained key queries shows (9).

In **Game** 4, we initially choose $2\ell$ random values $\overline{T}_{i,b} \leftarrow \mathcal{X}$ (for $i \in [\ell]$ and $b \in \{0,1\}$). We also guess the index $j^* \in [q_{\mathsf{H}}]$ of $\mathcal{A}$'s $\mathsf{H}$-query to the challenge preimage $X^*$. We now program $\mathsf{H}$ such that $\mathcal{A}$'s $j^*$-th $\mathsf{H}$-query $X = (x_i)_{i=1}^{\ell}$ is answered with $(\overline{T}_{i,x_i})_{i=1}^{\ell}$. Note that this does not change $\mathcal{A}$'s view at all, since all $\overline{T}_{i,b}$ are independently random (and at this point only used to program $\mathsf{H}$). If our guess for $j^*$ is incorrect, we abort (with a uniform bit as output). Note that we guess $j^*$ correctly with probability $1/q_{\mathsf{H}}$, and independently of $\mathcal{A}$'s success. We obtain

$$\Pr[out_4] - \frac{1}{2} \;=\; \frac{1}{q_{\mathsf{H}}}\left(\Pr[out_3] - \frac{1}{2}\right). \tag{10}$$

(In other words, $\mathcal{A}$'s distinguishing advantage just dropped by a polynomial factor of $q_{\mathsf{H}}$.)

In **Game** 5, we again modify the construction of the testers $\mathcal{T}$ used to prepare constrained keys for $\mathcal{A}$. Concretely, instead of setting up $\mathcal{T} \leftarrow \mathsf{Mark}(\{R_i\}_{\overline{x}_i \neq \perp})$ for fresh $R_i \leftarrow \mathcal{X}$ (as in Game 4), we set up

$$\mathcal{T} \leftarrow \mathsf{Mark}(\{\overline{T}_{i,\overline{x}_i}\}_{\overline{x}_i \neq \perp})$$

for the initially chosen $\overline{T}_{i,b} \leftarrow \mathcal{X}$ from Game 4. (In particular, all imperfections of the constrained keys are now "coordinated" as opposed to freshly random as in Game 4.) We postpone a proof of the following lemma until after the main proof.

**Lemma 4.2.** *For a suitable PPT adversary $\mathcal{D}$, we have*

$$\left|\Pr[out_5] - \Pr[out_4]\right| \;\le\; 20\ell(\ell-1)q_{\mathsf{key}} \cdot \mathsf{Adv}^{\mathsf{et\text{-}ind}}_{\mathsf{ET},\mathcal{D}}(k) + (4\ell(\ell-1)q_{\mathsf{key}}(q_{\mathsf{key}}+5)) \cdot \mu_{\mathsf{ET}}. \tag{11}$$

At this point, it is useful to briefly summarize our situation. In Game 5, $\mathcal{A}$ only gets imperfect constrained keys. Specifically, these keys don't work for a preimage $X$ if the corresponding tag $T = \mathsf{H}(X)$ contains $\overline{T}_{i,x_i}$ at the relevant positions. Besides, the "correct" tag $\mathsf{H}(X^*)$ for the challenge preimage $X^*$ is now programmed to $(\overline{T}_{i,x_i^*})_i$. Thus, even if $\mathcal{A}$ *could* ask for constrained keys that match $X^*$, by now all of these keys would fail on $X^*$.

However, the experiment itself still uses the full secret key $K$ to construct all constrained keys, and thus can evaluate even $\mathsf{PRF}_K(X^*)$. Hence, our next step is to put the experiment into a position in which it can *only* produce imperfect constrained keys. Specifically, not even the experiment will be able to compute $\mathsf{PRF}_K(X^*)$ (or even only distinguish $\mathsf{PRF}_K(X^*)$ from a random image).

As a first step, in **Game** 6, we prepare all constrained keys differently. Concretely, recall that in Game 5, we compute constrained keys as $K_{\overline{X}} = i\mathcal{O}(G_{K,\overline{X},\mathcal{T}}))$ for a function $G_{K,\overline{X},\mathcal{T}}$

(as in (7)) that is constructed from $K$ and $\mathcal{T} \leftarrow \mathsf{Mark}(\{\overline{T}_{i,\overline{x}_i}\}_{\overline{x}_i \neq \perp})$. Instead, we now compute $K_{\overline{X}} = i\mathcal{O}(G_{\overline{K},\overline{X},\mathcal{T}})$) for a function $G_{\overline{K},\overline{X},\mathcal{T}}$ (to be defined) that only uses a "weak key"

$$\overline{K} = \Big( \quad \left[\alpha_{i,b,j,\bar{t}_{i,b,j}}\right]_1, \quad \alpha_{i,b,j,1-\bar{t}_{i,b,j}} \quad \Big)_{(i,b,j)\in[\ell]\times\{0,1\}\times[d]},$$

where $\overline{T}_{i,b} = (\bar{t}_{i,b,j})_{j=1}^d$ denotes the bit representation of the $\overline{T}_{i,b} \in \mathcal{X}$. (In other words, $\overline{K}$ contains only some $\alpha_{i,b,j,c}$ "in plain," and others only "in the exponent." The $\alpha_{i,b,j,c}$ given only in the exponent are determined by the $\overline{T}_{i,b}$.)

The obfuscated function $G_{\overline{K},\overline{X},\mathcal{T}}$ is defined exactly like $G_{K,\overline{X},\mathcal{T}}$ (from (7)), but a final non-$\perp$ output is computed as

$$h\Big( \Big[\prod_{i,j} \alpha_{i,x_i,j,t_{i,j}}\Big]_{\ell d-1} \Big) = h\Big( \alpha_{i',x_{i'},j',t_{i',j'}} \cdot \prod_{\substack{(i,j)\in[\ell]\times[d] \\ (i,j)\neq(i',j')}} \left[\alpha_{i,x_i,j,t_{i,j}}\right]_1 \Big),$$

where $(i',j')$ is the lexicographically smallest tuple with $t_{i',j'} \neq \bar{t}_{i',x_{i'},j'}$. We may assume that such a tuple $(i',j')$ exists: indeed, we may assume that $X$ matches $\overline{X}$ (since otherwise, $G_{\overline{K},\overline{X},\mathcal{T}}$ outputs $\perp$). Hence, any $X, T$ with $t_{i,j} = \bar{t}_{i,x_i,j}$ for all $i,j$ would satisfy $\mathsf{Test}(\mathcal{T}, \{T_i\}_{\overline{x}_i \neq \perp}) = 1$ by construction of $\mathcal{T}$, and would thus also lead to a $\perp$-output. (Note that here, we crucially rely on the fact that all of the considered functions $G_{K,\overline{X}}$, $G_{K,\overline{X},\mathcal{T}}$, and $G_{\overline{K},\overline{X},\mathcal{T}}$ output $\perp$ on inputs $X$ that do not match $\overline{X}$.)

Thus, $G_{K,\overline{X},\mathcal{T}}$ and $G_{\overline{K},\overline{X},\mathcal{T}}$ compute the same function. By the indistinguishability of $i\mathcal{O}$,

$$\Pr[out_6] - \Pr[out_5] \ = \ q_{\mathsf{key}} \cdot \mathsf{Adv}^{\mathsf{ind\text{-}obf}}_{i\mathcal{O},\mathcal{E}}(k) \tag{12}$$

for a suitable $i\mathcal{O}$-distinguisher $\mathcal{E}$.

One crucial property we will use next is that Game 6 really only uses $\left[\alpha_{i,b,\bar{t}_{i,b,j}}\right]_1$ (and not the exponents $\alpha_{i,b,\bar{t}_{i,b,j}}$ themselves), *except* for constructing the final challenge key $\mathsf{PRF}_K(X^*)$.

In **Game** 7, we substitute any real image $\mathsf{PRF}_K(X^*)$ handed to $\mathcal{A}$ with $h([z]_D)$ for an independent random group element $[z]_D \in \mathbb{G}_D$. We claim that

$$\Pr[out_6] - \Pr[out_7] \ = \ \mathsf{Adv}^{\mathsf{mddh}}_{\mathcal{G},\ell d-1,\mathcal{F}}(k) \tag{13}$$

for a suitable adversary $\mathcal{F}$. Concretely, $\mathcal{F}$ gets as input $([x_{i,j}]_1)_{(i,j)\in[\ell]\times[k]}$ and a challenge $[z]_{\ell d-1}$, with either $z = \prod_{i,j} x_{i,j}$, or a random $z$. Now $\mathcal{F}$ simulates Game 6, setting up

$$\left[\alpha_{i,b,j,\bar{t}_{i,b,j}}\right]_1 = \alpha'_{i,b,j} \cdot [x_{i,j}]_1 = \left[\alpha'_{i,b,j} \cdot x_{i,j}\right]_1 \quad \text{for uniform } \alpha'_{i,b,j}. \tag{14}$$

All $\alpha_{i,b,j,1-\bar{t}_{i,b,j}}$ are chosen uniformly by, and thus fully known to $\mathcal{F}$, without embedding any of the $[x_{i,j}]_1$. By our remark above, this setup suffices to perfectly simulate Game 6, except for a real challenge image $\mathsf{PRF}_K(X^*)$. If we use that $\mathsf{H}(X^*) = (\overline{T}_{i,x_i^*})_{i=1}^{\ell} = ((\bar{t}_{i,x_i^*,j})_{j=1}^d)_{i=1}^{\ell}$ (by our change in Game 4), we can write

$$\mathsf{PRF}_K(X^*) = \Big[\prod_{i,j} \alpha_{i,x_i^*,j,\bar{t}_{i,x_i^*,j}}\Big]_{\ell d-1} = \Big(\prod_{i,j} \alpha'_{i,x_i^*,j}\Big) \cdot \Big[\prod_{i,j} x_{i,j}\Big]_{\ell d-1}.$$

Hence, if we let $\mathcal{F}$ prepare a real challenge for $\mathcal{A}$ as $(\prod_{i,j} \alpha'_{i,x_i^*,j}) \cdot [z]_{\ell d-1}$, then this induces a view as in Game 6 if $z = \prod_{i,j} x_{i,j}$, and a view as in Game 7 if $z$ is random. This shows (13).

Finally, in **Game** 8, $\mathcal{A}$ is challenged with an independently random $\ell'$-bitstring (instead of $h([z]_D)$) in case $b = 0$. Hence, now $\mathcal{A}$ is always challenged with a random $\ell'$-bitstring, regardless of $b$. By our assumption on $\mathcal{H}$, we have

$$\left|\Pr[out_8] - \Pr[out_7]\right| \ \leq \ \varepsilon_{\mathcal{H}}. \tag{15}$$

Besides, since now $\mathcal{A}$'s view is independent of $b$, we have

$$\Pr[out_8] \;=\; 1/2. \tag{16}$$

Putting (6,8,9,10,11,12,13,15,16) together shows (5) as desired. □

It remains to prove Lemma 4.2.

*Proof of Lemma 4.2.* We construct a series of hybrids between Game 4 and Game 5. For each hybrid step, we construct a distinguisher $\mathcal{D}'$ in the sense of Lemma 3.2.

**The hybrids.** Consider the hybrid game $H_{j',i'}$, in which

- the first $j'-1$ of $\mathcal{A}$'s constrained key queries are answered as in Game 4 (i.e., with a tester $\mathcal{T} \leftarrow \mathsf{Mark}(\{R_i\}_{\overline{x}_i\neq\perp})$ for independently random $R_i$),
- the last $q_{\mathsf{key}}-j'$ of $\mathcal{A}$'s constrained key queries are answered as in Game 5 (i.e., with a tester $\mathcal{T} \leftarrow \mathsf{Mark}(\{\overline{T}_{i,\overline{x}_i}\}_{\overline{x}_i\neq\perp})$ for the initially chosen $\overline{T}_{i,b}$),
- the $j'$-th constrained key query is answered with a tester

$$\mathcal{T} \leftarrow \mathsf{Mark}(\{R_i\}_{\substack{\overline{x}_i\neq\perp\\ i\leq i'}} \cup \{\overline{T}_{i,\overline{x}_i}\}_{\substack{\overline{x}_i\neq\perp\\ i>i'}})$$

  for independently random $R_i$.

Clearly, $H_{q_{\mathsf{key}}+1,0}$ is the same as Game 4, $H_{1,0}$ is Game 5, and $H_{j',q_{\mathsf{H}}}$ is identical to $H_{j'+1,0}$. Hence, we only need to show that any two hybrids $H_{j',i'-1}$ and $H_{j',i'}$ have indistinguishable outputs.

**Intuition for reduction.** Before we describe $\mathcal{D}'$, a little more intuition is in order. The only difference between two adjacent hybrids $H_{j',i'-1}$ and $H_{j',i'}$ is a single value used in the preparation of the tester from $\mathcal{A}$'s $j'$-th constrained key query.[7] Call this tester $\mathcal{T}^*$, and call this value $\mathsf{z}$. In $H_{j',i'-1}$, we have $\mathsf{z}=\overline{T}_{i',\overline{x}_{i'}}$, and in $H_{j',i'}$, the value $\mathsf{z}$ is independently random. Hence, we need to justify replacing (a specific use of) $\overline{T}_{i',\overline{x}_{i'}}$ with a random value. What makes this complicated is the fact that $\overline{T}_{i',\overline{x}_{i'}}$ is also used elsewhere: first, testers prepared upon later constrained key queries may depend on $\overline{T}_{i',\overline{x}_{i'}}$, and, second, $\overline{T}_{i',\overline{x}_{i'}}$ is required *in plain* to program $\mathsf{H}$ in case $\overline{x}^*_{i'}=\overline{x}_{i'}$.

To simplify things, we will consider the cases $\overline{x}^*_{i'}=\overline{x}_{i'}$ and $\overline{x}^*_{i'}\neq\overline{x}_{i'}$ separately. (Our reduction will guess in which case we are and abort if the guess turns out to be wrong.)

First, in case $\overline{x}^*_{i'}\neq\overline{x}_{i'}$, our simulation will not need to know $\overline{T}_{i',\overline{x}_{i'}}$ in plain to program $\mathsf{H}$. Yet, we still need to be able to prepare later testers that involve $\overline{T}_{i',\overline{x}_{i'}}$. Here, the extensibility of testers comes in handy: such later testers can be prepared by extending a single given tester $\mathcal{T}_2$ for $\overline{T}_{i',\overline{x}_{i'}}$. Furthermore, by (3), even given $\mathcal{T}_2$ and another tester $\mathcal{T}_1$ for a value $\mathsf{z}$, we cannot distinguish the cases $\mathsf{z}=\overline{T}_{i',\overline{x}_{i'}}$ and $\mathsf{z}=R_{i'}$ for an independently random $R_{i'}$. Thus, we can use $\mathcal{T}_1$ to prepare the $j'$-th tester $\mathcal{T}^*$. Depending on $\mathsf{z}$, this simulates either $H_{j',i'-1}$ or $H_{j',i'}$ for $\mathcal{A}$ perfectly (but up to $\mathsf{ET}$'s statistical defects).

The case $\overline{x}^*_{i'}=\overline{x}_{i'}$ is a little more involved, since the simulation will need to know $\overline{T}_{i',\overline{x}_{i'}}$ in plain to program $\mathsf{H}$. Recall that we only need to replace one use of $\overline{T}_{i',\overline{x}_{i'}}$ in a single tester $\mathcal{T}^*$. Moreover, observe that there must be at least one index $i^*$ such that $\overline{x}_{i^*}=1-\overline{x}^*_{i^*}$. (In other words, there must be an index $i^*$ in which $\mathcal{A}$'s target preimage $X^*$ differs from $\mathcal{A}$'s $j'$-th constrained key query.) Observe that the corresponding value $\overline{T}_{i^*,\overline{x}_{i^*}}$ will be used in the generation of $\mathcal{T}^*$, but will not have to known in plain to the simulation. (Indeed, $\mathsf{H}$ is programmed with $\overline{T}_{i^*,1-\overline{x}_{i^*}}$, and later constrained key queries can be generated by extending a given tester for $\overline{T}_{i^*,\overline{x}_{i^*}}$ as necessary.) Hence, by (4), we can use $\overline{T}_{i^*,\overline{x}_{i^*}}$ as a "cover" behind which we can replace $\mathsf{z}=\overline{T}_{i',\overline{x}_{i'}}$ with $\mathsf{z}=R_{i'}$ in $\mathcal{T}^*$ (and only there).

[7]Technically, if $\mathcal{A}$'s $j'$-th query satisfies $\overline{x}_{i'}=\perp$, then this value is not even used, and $H_{j',i'-1}$ and $H_{j',i'}$ proceed identically. The interesting case is of course $\overline{x}_{i'}\neq\perp$.

**The reduction.** We can now describe $\mathcal{D}'$ in more detail. For convenience, we will describe a single $\mathcal{D}'$ that behaves either as a distinguisher against (3), or one against (4), depending on an (initial) internal coin toss. First, $\mathcal{D}'$ uniformly guesses $j' \in [q_{\mathsf{key}}]$, as well as $i' \in [\ell]$ and $b' \in \{0,1\}$. If $b' = 1$, then $\mathcal{D}'$ also uniformly chooses $i^* \in [\ell] \setminus \{i'\}$ and a bit $b^*$. Intuitively, $\mathcal{D}'$ will now try to simulate either $H_{j',i'-1}$ or $H_{j',i'}$, depending on its own input. (However, as it will turn out, this simulation can fail, in which case $\mathcal{D}'$ outputs 0. We will make sure that the failure probability will be independent of $i', j'$.)

Specifically, $\mathcal{D}'$ will answer the first $j'-1$ constrained key queries as in Game 4, using testers prepared with independently random values $R_i$. (Observe that this does not require any trapdoor information.) Also as in Game 4, $\mathcal{D}'$ will program $\mathsf{H}(X^*) = (\overline{T}_{i,x_i^*})_{i=1}^{\ell}$ for initially chosen $\overline{T}_{i,x_i^*}$; however, note that so far the values $\overline{T}_{i,1-x_i^*}$ have not been used at all.

For the $(j')$-th constrained key query, say, $\overline{X} = (\overline{x}_i)_{i=1}^{\ell}$, $\mathcal{D}'$ prepares a tester $\mathcal{T}^*$ as follows. First, we may assume that $\overline{x}_{i'} \neq \perp$, since $H_{j',i'-1}$ and $H_{j',i'}$ perform identically for $\overline{x}_{i'} = \perp$. (More specifically, if $\overline{x}_{i'} = \perp$, then $\mathcal{D}'$ can continue and finish the simulation as in $H_{j',i'}$ without embedding its own challenge at all.) Furthermore, we will interpret $\mathcal{D}'$'s initial guess bit $b'$ as follows: if $b' = 0$, we will prepare for $\overline{x}_{i'} = 1 - x_{i'}^*$ (and $\mathcal{D}'$ will act as a (3)-distinguisher); if $b' = 1$, we will prepare for $\overline{x}_{i'} = x_{i'}^*$ (and $\mathcal{D}'$ will act as a (4)-distinguisher).

**Case 1: $\overline{x}_{i'} \neq x_{i'}^*$.** Concretely, in case $b' = 0$, $\mathcal{D}'$ expects a challenge $(\mathcal{T}_1, \mathcal{T}_2)$, where $\mathcal{T}_1 \leftarrow \mathsf{Mark}(\{\mathsf{z}_1\})$ and $\mathcal{T}_2 \leftarrow \mathsf{Mark}(\{\mathsf{z}_2\})$, such that either $\mathsf{z}_1 = \mathsf{z}_2$, or both $\mathsf{z}_1, \mathsf{z}_2$ and independently random. Here, $\mathcal{D}'$ will implicitly set $\overline{T}_{i',\overline{x}_{i'}} = \mathsf{z}_2$. (All other $\overline{T}_{\overline{x}_i,b}$ will be chosen by $\mathcal{D}'$ initially.) $\mathcal{T}_1$ will be used to construct $\mathcal{T}^*$ through

$$\mathcal{T}^* \leftarrow \mathsf{Extend}(\mathcal{T}_1, \{R_i\}_{\substack{\overline{x}_i \neq \perp \\ i<i'}} \cup \{\overline{T}_{i,b}\}_{\substack{\overline{x}_i \neq \perp \\ i>i'}}) \equiv \mathsf{Mark}(\{R_i\}_{\substack{\overline{x}_i \neq \perp \\ i<i'}} \cup \{\mathsf{z}_1\} \cup \{\overline{T}_{i,b}\}_{\substack{\overline{x}_i \neq \perp \\ i>i'}})$$

for freshly random $R_i$. The testers for all remaining constrained key queries are prepared as in Game 5, by extending $\mathcal{T}_2$ (and thus using $\mathsf{z}_2$ as the value of $\overline{T}_{i',\overline{x}_{i'}}$). Concretely, upon $\mathcal{A}$'s $j$-th constrained key query (with $j > j'$) for some $\overline{X}' = (\overline{x}'_i)_i$, $\mathcal{D}'$ prepares a tester $\mathcal{T}'$ for the multiset $\mathsf{Z}' = \{\overline{T}_{i,\overline{x}'_i}\}_{\overline{x}'_i \neq \perp}$. If $\overline{x}'_{i'} \neq \overline{x}_{i'}$, then $\overline{T}_{i',\overline{x}_{i'}} \notin \mathsf{Z}'$, and thus $\mathcal{D}'$ knows all values in $\mathsf{Z}'$ in plain. However, if $\overline{x}'_{i'} = \overline{x}_{i'}$, then $\mathcal{D}'$ prepares

$$\mathcal{T}' \leftarrow \mathsf{Extend}(\mathcal{T}_2, \{\overline{T}_{i,\overline{x}'_i}\}_{\substack{\overline{x}'_i \neq \perp \\ i \neq i'}}).$$

Observe that either way, this yields a tester $\mathcal{T}'$ as prepared in Game 5, with $\overline{T}_{i',\overline{x}_{i'}} = \mathsf{z}_2$.

Hence, unless $\overline{x}_{i'} = x_{i'}^*$ (in which case $\mathcal{D}'$ fails, because it cannot program $\mathsf{H}(X^*)$ with $\overline{T}_{i',\overline{x}_{i'}}$), and up to statistical defects introduced by $\mathsf{Extend}$, this setup yields a perfect view of $H_{j',i'-1}$ if $\mathsf{z}_1 = \mathsf{z}_2$, and of $H_{j',i'}$ if $\mathsf{z}_1, \mathsf{z}_2$ are independent.

**Case 2: $\overline{x}_{i'} = x_{i'}^*$.** In case $b' = 1$, $\mathcal{D}'$ expects a challenge $(\mathcal{T}_1, \mathcal{T}_2, \mathsf{z}_3)$ for $\mathcal{T}_1 \leftarrow \mathsf{Mark}(\{\mathsf{z}_1, \mathsf{z}_2\})$ and $\mathcal{T}_2 \leftarrow \mathsf{Mark}(\{\mathsf{z}_1\})$, with either $\mathsf{z}_3 = \mathsf{z}_2$, or independently random $\mathsf{z}_3$. As already indicated, $\mathcal{D}'$ now prepares for $\overline{x}_{i'} = x_{i'}^*$. Note that in this case, there must be an $i \neq i'$ with $\overline{x}_i = 1 - x_i^*$ (since otherwise, $X^*$ would match $\overline{X}$, which would violate the rules of the experiment). We will interpret the initially chosen $i^*$ as a guess for the smallest such $i$, and $b^*$ as a guess for $\overline{x}_{i'}$. In particular, $\mathcal{D}'$ fails (potentially only later, when $\overline{X}$ and $X^*$ are known) if these guesses turn out to be wrong. (Note that $\mathcal{D}'$ fails except with probability $1/2(\ell-1)$ here.) Otherwise, $\mathcal{D}'$ computes

$$\mathcal{T}^* \leftarrow \mathsf{Extend}(\mathcal{T}_1, \{R_i\}_{\substack{\overline{x}_i \neq \perp \\ i<i' \\ i \neq i^*}} \cup \{\overline{T}_{i,b}\}_{\substack{\overline{x}_i \neq \perp \\ i>i' \\ i \neq i^*}}) \equiv \mathsf{Mark}(\{R_i\}_{\substack{\overline{x}_i \neq \perp \\ i<i' \\ i \neq i^*}} \cup \{\mathsf{z}_1, \mathsf{z}_2\} \cup \{\overline{T}_{i,b}\}_{\substack{\overline{x}_i \neq \perp \\ i>i' \\ i \neq i^*}})$$

for freshly random $R_i$. This implicitly sets $\overline{T}_{i^*,\overline{x}_{i^*}} = \mathsf{z}_1$ (if $i^* > i'$) or $R_{i^*} = \mathsf{z}_1$ (if $i^* < i'$). Since we assumed $\overline{x}_{i^*} = 1 - x_{i^*}^*$, we will never have to unveil $\mathsf{z}_1$ in plain, and further constrained keys involving $\overline{T}_{i^*,\overline{x}_{i^*}}$ can be generated from $\mathcal{T}_2$ as necessary.

Specifically, say that $\mathcal{A}$'s $j$-th constrained key query (for $j > j'$) is $\overline{X}' = (\overline{x}'_i)_i$. $\mathcal{D}'$ must prepare a tester $\mathcal{T}'$ for the multiset $\mathsf{Z}' = \{\overline{T}_{i,\overline{x}'_i}\}_{\overline{x}'_i \neq \perp}$. If $i^* < i'$, then $\mathcal{D}'$ knows all $\overline{T}_{i,b}$ in plain (because the only value unknown to $\mathcal{D}'$ is $\mathsf{z}_1 = R_{i^*}$, which is only used to construct $\mathcal{T}^*$). Hence, in that case, $\mathcal{D}'$ can directly prepare $\mathcal{T}'$ from a known multiset $\mathsf{Z}'$. If, on the other hand, $i^* > i'$, then $\overline{T}_{i^*,\overline{x}_{i^*}} = \mathsf{z}_1$ is unknown to $\mathcal{D}'$. Hence, for $\overline{x}'_{i^*} = \overline{x}_{i^*}$, $\mathcal{D}'$ must compute $\mathcal{T}'$ as

$$\mathcal{T}' \leftarrow \mathsf{Extend}(\mathcal{T}_1, \{\overline{T}_{i,\overline{x}'_i}\}_{\substack{\overline{x}'_i \neq \perp \\ i \neq i^*}}).$$

Again, either way, this yields a tester $\mathcal{T}'$ as prepared in Game 5, with $\overline{T}_{i^*,\overline{x}_{i^*}} = \mathsf{z}_1$.

Moreover, $\mathcal{D}'$ will initially set up $\overline{T}_{i',b^*} = \overline{T}_{i',\overline{x}_{i'}} = \mathsf{z}_3$. This implies that if $\mathsf{z}_2 = \mathsf{z}_3$ (and unless $\mathcal{D}'$ fails), then $\mathcal{D}'$ perfectly simulates $H_{j',i'-1}$ (up to statistical defects of $\mathsf{Extend}$), and if $\mathsf{z}_2, \mathsf{z}_3$ are independent, then $\mathcal{D}'$ perfectly simulates $H_{j',i'}$ (again up to $\mathsf{Extend}$'s statistical defects).

**Wrapping up.** Summarizing, in either case, unless $\mathcal{D}'$ fails, and up to $\mathsf{Extend}$'s defects, $\mathcal{D}'$ provides a perfect view of $H_{j',i'-1}$ or $H_{j',i'-1}$, depending on its own challenge. Now if we let $\mathcal{D}'$ fail artificially (i.e., more often than necessary) in case $b' = 0$, the probability of failure can be fixed to $1/4(\ell - 1)$, independently of $i'$ and $j'$. Furthermore, in either case, $\mathsf{Extend}$ is invoked at most $q_{\mathsf{key}}$ times, creating a statistical defect of at most $q_{\mathsf{key}} \cdot \mu_{\mathsf{ET}}$. Thus, if we write $\varepsilon_{\mathcal{D}'}$ for $\mathcal{D}'$'s advantage in distinguishing the left-hand-side from the right-hand-side of (3), resp. (4), a standard hybrid argument (over $i', j'$) thus yields

$$|\varepsilon_{\mathcal{D}'}| \;\geq\; \frac{1}{4\ell(\ell-1)q_{\mathsf{key}}} \cdot \left|\Pr\left[out_5\right] - \Pr\left[out_4\right]\right| - q_{\mathsf{key}} \cdot \mu_{\mathsf{ET}}. \tag{17}$$

On the other hand, an inspection of the proof of Lemma 3.2 yields the concrete reduction

$$\left|\varepsilon_{\mathcal{D}'}\right| \;\leq\; 5 \cdot \mathsf{Adv}^{\mathsf{et\text{-}ind}}_{\mathsf{ET},\mathcal{D}}(k) + 5 \cdot \mu_{\mathsf{ET}} \tag{18}$$

for a suitable adversary $\mathcal{D}$ on $\mathsf{ET}$'s indistinguishability. Combining (17) and (18) shows (11). □

## 5 On recent candidates for approximate multilinear maps

### 5.1 Current candidates for approximate multilinear maps

We have claimed that our construction can be implemented with the recent candidates [21, 14] of approximate multilinear maps. What makes our claim nontrivial is that the constructions from [21, 14] only provide "approximations" of multilinear maps, in the following sense:

**Randomized encodings.** Instead of group elements, these works define "randomized encodings." Essentially, a randomized encoding is a group element with an additional noise term. This means that there is a whole set of encodings $\mathsf{Enc}(g)$ of a group element $g$. This has several effects which we describe in the following.

**Canonical forms.** It is possible to extract a "canonical form" $\mathsf{Ext}(enc)$ from an encoding $enc$. This canonical form is the same for two given encodings of the same group element $g$, and different for two given encodings of different group elements.[8] Canonical forms can thus also be used to compare group elements (although, e.g., [21] describe a more direct $\mathsf{isZero}$ algorithm for comparison). Moreover, the canonical form of a uniformly random group element is also (statistically close to) uniform.

[8] There is a negligible error in both cases. More specifically, with negligible probability over the implicit noise of the given encodings, we can have false positives and false negatives when comparing encoded group elements by their canonical form. Even worse, it is conceivable that through a clever choice of concrete (completely valid) encodings, such errors can be provoked deliberately. However, if we assume that encodings are re-randomized prior to computing the canonical form (using a suitable $\mathsf{reRand}$ algorithm as described in [21, 14]), this error can at least be made negligible for *all* pairs of (sufficiently low-noise) encodings. (The price to pay here is that the algorithm to compute a canonical form becomes probabilistic.)

**Leakage through noise.** Even though two computations (e.g., $[y]_2 = [x]_1 \cdot [x]_1$ and $[y']_2 = ([x]_1 + [y]_1) \cdot ([x]_1 - [y]_1) + [y^2]_2$) yield the same group element, the computed *encodings* (in this case of $[y]_2$ and $[y']_2$) may not be the same. Specifically, an encoding may leak (through its noise term) the sequence of operations used to construct it. Note, however, that the induced canonical forms of two encodings of the same group element are the same, at least with high probability.

**No exponents.** It is not immediately possible to explicitly use arbitrary "exponents." It *is* possible to choose (almost) uniformly distributed "level-0 encodings," which correspond to exponents. However, these can only be used in a black-box way, and without using their explicit $\mathbb{Z}_p$-representation.

**Noise growth.** Each operation on encodings increases the size of their noise terms. (More specifically, the noise term of the result of an operation is larger than the noise terms of the inputs.) In particular, each encoding can be used only for an a-priori limited number of operations. After that, its noise term becomes too large, and its canonical form may change.

We also note that there is an even more recent third candidate of approximate multilinear maps [35]. This candidate follows the first candidate [21], yet is significantly more efficient. However, this efficiency comes at a cost: unlike [21, 14], the candidate [35] supports only computational (but not decisional) hardness assumptions, and in particular no multilinear DDH assumption. Thus, we do *not* know how to prove our construction secure with this most recent candidate implementation of multilinear maps.

## 5.2 Our constructions with the current candidates

**Necessary adaptations to the scheme.** To use our CPRF from Section 4 with encodings instead of group elements, we make the following changes:

- Key generation chooses uniform level-0 encodings $\alpha_{i,b,j,c}$ instead of exponents.
- Evaluation outputs not the encoding (of a group elements), but its canonical form:

$$\mathsf{PRF}_K(X) = \mathsf{Ext}\left(\left[\prod_{(i,j)\in[\ell]\times[d]} \alpha_{i,x_i,j,t_{i,j}}\right]_{\ell d-1}\right)$$

(Similarly for the output of the function $G_{K,\overline{X}}$ used for constrained keys.)

We emphasize that we do not intend to use encodings instead of group elements for our extensible tester $\mathsf{ET}$. (Recall that $\mathsf{ET}$ only assumes a cyclic group in which the DDH assumption holds, but no multilinear or even only bilinear operation. While it is conceivable to use encodings also for this group, this would seem to unnecessarily complicate things.) As a result, we also do not need to explicitly compare encodings of group elements.

**Dealing with imperfect canonical forms.** However, we emphasize that there is a subtlety here concerning the computation of the canonical form. Concretely, recall that we assume that computing a canonical form first re-randomizes the given encoding (cf. Footnote 8). (Otherwise, it might be feasible to find preimages for which correctness is violated with certainty.) Hence, computing a canonical form requires randomness, although of course the evaluation algorithm of the PRF should be deterministic.

As a solution, we can first of all (polynomially) scale the security parameter of the multilinear maps construction, such that errors become less likely, in the following sense. Namely, any two encodings of the same group element are mapped to the same canonical form, except with probability at most $1/2^{p(k)}$, where $p$ can be an arbitrary a-priori fixed polynomial. We can then add a single string $R$ of random coins (long enough to compute a single canonical form) to $K$ and each derived constrained key. This $R$ is then used as randomness for the final canonical form computation. We can then choose $p$ so large that with overwhelming probability over $R$, the PRF is perfectly correct, in the sense that all constrained keys compute a single function

$\mathsf{PRF}_K(\cdot)$, constrained to the respective subset. (A concrete such polynomial $p$ can be derived from a union bound over all possible constrained keys and preimages.)

We mention that the bit-fixing PRF from [6] faces similar problems (although no explicit discussion of the problem or a solution is provided there).

**Necessary adaptations to the proof.** The modifications from Game 1 to 5 do not affect any computations on encodings, and hence do not require any adaptations. However, in Game 6, we change the way images are computed from constrained keys. In particular, while the functions $G_{\overline{K},\overline{X},\mathcal{T}}$ from Game 6 compute the same *group element* as the functions $G_{K,\overline{X},\mathcal{T}}$ from Game 5, the respective *encodings* may differ. However, since $G_{K,\overline{X},\mathcal{T}}$ and $G_{\overline{K},\overline{X},\mathcal{T}}$ only actually output canonical forms, their output behavior is also identical, except with negligible probability over the setup (i.e., the randomness $R$ above). This observation enables the game hop from Game 5 to Game 6 exactly as in the main proof, but with an additional negligible statistical error term.

The modification in Game 7 is justified using a reduction to the MDDH assumption. This reduction applies equally in a setting with encodings, of course using the encoding-based MDDH assumption from [21]. Finally, the change from Game 8 applies literally, using $\mathsf{Ext}$ as key derivation.

## 6 Application: adaptively secure NIKE for large user groups

As a simple application of fully secure constrained CPRFs, we construct the first adaptively secure identity-based non-interactive key exchange protocol (ID-NIKE, [42, 15, 24, 40, 19, 6, 7]) for large user groups. Both protocol and security proof are very simple, and can be viewed as an extension of a similar result for the two-party case from [42, 6].

Our concrete ID-NIKE construction also gives rise to a public-key non-interactive key exchange (PK-NIKE, [12, 18]) for large user groups. (Perhaps somewhat surprisingly, there is no known general construction of PK-NIKE protocols from ID-NIKE protocols. However, as it turns out, the specific structure of our CPRF also works in a public-key context.) Interestingly, our PK-NIKE does not require indistinguishability obfuscation, and is also the first of its kind (even in the random oracle model).

### 6.1 ID-NIKE security model

We follow the security models presented in [19, 7].

**Definition 6.1** (ID-NIKE). *An $n$-user identity-based non-interactive key exchange (ID-NIKE) consists of the following algorithms:*

**Master key generation.** $\mathsf{Gen}(1^k)$ *generates a master secret key $msk$.*

**User key extraction.** $\mathsf{Extract}(msk, id)$ *(for an identity $id \in \{0,1\}^k$) outputs a user secret key $usk_{id}$.*

**Shared key evaluation.** $\mathsf{Key}(usk_{id}, \mathcal{I})$ *(for a set $\mathcal{I} \subseteq \{0,1\}^k$ of $n$ identities with $id \in \mathcal{I}$) outputs a shared key $K_{\mathcal{I}} \in \{0,1\}^k$.*

*For correctness, we require that the output of* $\mathsf{Key}$ *is deterministic and only depends on $\mathcal{I}$. Formally, for all $msk \leftarrow \mathsf{Gen}(1^k)$ and all $\mathcal{I} \subseteq \{0,1\}^k$ with $|\mathcal{I}| = n$, there is a single value $K_{\mathcal{I}}$ such that $\mathsf{Key}(usk_{id}, \mathcal{I}) = K_{\mathcal{I}}$ for all $id \in \mathcal{I}$ and all $usk_{id} \leftarrow \mathsf{Extract}(msk, id)$.*

**Definition 6.2** (Security of an ID-NIKE). *We say that an $n$-user ID-NIKE* $\mathsf{ID\text{-}NIKE}$ *is adaptively secure if the advantage function*

$$\mathsf{Adv}^{\mathsf{id\text{-}nike}}_{\mathsf{ID\text{-}NIKE},\mathcal{A}}(k) = \Pr\left[\mathsf{Exp}^{\mathsf{id\text{-}nike}}_{\mathsf{ID\text{-}NIKE},\mathcal{A}}(k) = 1\right] - 1/2$$

*is negligible for all PPT adversaries $\mathcal{A}$. Here, experiment $\mathsf{Exp}^{\mathsf{id\text{-}nike}}_{\mathsf{ID\text{-}NIKE},\mathcal{A}}$ is defined as follows:*

1. *the experiment samples $msk \leftarrow \mathsf{Gen}(1^k)$, and gives $\mathcal{A}$ oracle access to $\mathsf{Extract}(msk, \cdot)$; also, $\mathcal{A}$ gets access to an oracle $\mathcal{O}_K$ that, on input a set $\mathcal{I} \subseteq \{0,1\}^k$ of $n$ identities, returns a shared key $K_{\mathcal{I}}$ (as defined in Definition 6.1),*
2. *at some point, $\mathcal{A}$ decides on a target set $\mathcal{I}^* \subseteq \{0,1\}^k$ of $n$ identities,*
3. *the experiment tosses a coin $b \leftarrow \{0,1\}$; if $b = 0$, then $\mathcal{A}$ gets $K_{\mathcal{I}^*}$, and if $b = 1$, then $\mathcal{A}$ gets a uniformly chosen $\{0,1\}^k$-element,*
4. *after potentially more oracle queries, $\mathcal{A}$ outputs a guess $b'$, and the experiment outputs $1$ if $b = b'$.*

*We only quantify over $\mathcal{A}$ that guarantee that (a) $\mathcal{I}^*$ does not contain any identity queried to* $\mathsf{Extract}$*, and (b) $\mathcal{I}^*$ is never queried to $\mathcal{O}_K$.*

## 6.2 Our ID-NIKE protocol

For any polynomial $n = n(k)$, our $n$-user ID-NIKE is readily obtained from a bit-fixing CPRF $\mathsf{PRF} = (\mathsf{KGen}, \mathsf{KDel}, \mathsf{KEval})$ with input length $\ell = nk$ and output length $\ell' = k$:

**Master key generation.** $\mathsf{Gen}(1^k)$ outputs $msk \leftarrow \mathsf{KGen}(1^k)$.

**User key extraction.** $\mathsf{Extract}(msk, id)$ returns $usk_{id} = (K_{id,i})_{i=1}^n$ for $K_{id,i} \leftarrow \mathsf{KDel}(msk, \overline{X}_i)$ with

$$\overline{X}_i = (\perp^{(i-1)k}, id, \perp^{(n-i)k}) \in (\{0,1\} \cup \{\perp\})^{\ell k}. \tag{19}$$

Observe that these constrained keys $K_{id,i}$ allow to evaluate $\mathsf{PRF}$ on all preimages $X$ of the form $X = (id_1, \dots, id_n) \in \{0,1\}^{nk}$ with $id_{i^*} = id$ for some $i^*$.

**Shared key evaluation.** $\mathsf{Key}(usk_{id}, \mathcal{I})$ first writes $\mathcal{I} = \{id_1, \dots, id_n\}$ with $id_i < id_{i+1}$ (in lexicographic order) and $id_{i^*} = id$. Then, $\mathsf{Key}$ outputs

$$K_{\mathcal{I}} = \mathsf{PRF}_K(id_1, \dots, id_n) = \mathsf{KEval}(K_{id_{i^*}}, (id_1, \dots, id_n)). \tag{20}$$

Correctness is obvious from (20). Security can be directly reduced to the security of $\mathsf{PRF}$:

**Theorem 6.3** (Security of $\mathsf{ID\text{-}NIKE}$). $\mathsf{ID\text{-}NIKE}$ *is adaptively secure, assuming* $\mathsf{PRF}$ *is a bit-fixing CPRF. Concretely, for every adversary $\mathcal{A}$ on* $\mathsf{ID\text{-}NIKE}$*'s adaptive security, there is a* $\mathsf{PRF}$ *distinguisher $\mathcal{B}$ (of roughly the same complexity as $\mathcal{A}$) with*

$$\mathsf{Adv}^{\mathsf{cprf}}_{\mathsf{PRF},\mathcal{B}}(k) = \mathsf{Adv}^{\mathsf{id\text{-}nike}}_{\mathsf{ID\text{-}NIKE},\mathcal{A}}(k). \tag{21}$$

*Proof.* $\mathcal{B}$ simulates $\mathcal{A}$, and translates and answers $\mathcal{A}$'s queries as follows:

- an $\mathsf{Extract}(msk, id)$ query is translated into $n$ constrained key queries for $\overline{X}_1, \dots, \overline{X}_n$ (as in (19)), and answered with $usk_{id} = (K_{id,i})_{i=1}^n$,
- an $\mathcal{O}_K(\mathcal{I})$ query (for $\mathcal{I} = \{id_1, \dots, id_n\}$ in lexicographic order) is translated into a constrained key query for $\overline{X} = (id_1, \dots, id_n) \in \{0,1\}^{nk}$, and the resulting $\mathsf{PRF}_K(id_1, \dots, id_n)$ is returned to $\mathcal{A}$,
- similarly, a challenge user set $\mathcal{I}^* = \{id_1^*, \dots, id_n^*\}$ chosen by $\mathcal{A}$ is translated into a challenge preimage $X^* = (id_1^*, \dots, id_n^*)$, and the resulting challenge image is handed to $\mathcal{A}$.

Finally, $\mathcal{B}$ relays $\mathcal{A}$'s guess bit $b'$. This immediately shows (21), assuming that all of $\mathcal{B}$'s queries are allowed (in the sense of the CPRF experiment from Definition 2.4). Hence, we have to check that $\mathcal{A}$'s queries do not lead to $\mathsf{KDel}$ queries from $\mathcal{B}$ which allow to evaluate $\mathsf{PRF}_K(X^*)$. However, this is clear by the rules of the ID-NIKE adaptive security experiment. (Specifically, $\mathcal{A}$ may not query $\mathsf{Extract}$ on any $id_i^*$. Hence, the resulting keys from $\mathsf{KDel}$ queries do not allow to evaluate $\mathsf{PRF}_K(X^*)$. A similar argument holds for $\mathcal{A}$'s $\mathcal{O}_K$ queries.) □

Hence, if we use $\mathsf{ID\text{-}NIKE}$ with our bit-fixing CPRF from Section 4, we obtain an adaptively secure $n$-user ID-NIKE for arbitrary polynomials $n = n(k)$. Of course, this ID-NIKE uses the same heavyweight tools as our CPRF; in particular, it requires indistinguishability obfuscation, multilinear maps, and its security proof is performed only in the random oracle model. We also mention that, although only secure in a static sense, the recent ID-NIKE from [7] enjoys certain interesting properties that our scheme does not have. (Specifically, the scheme from [7] does not require a trusted setup, nor does it require the number $n$ of users to be fixed in advance.)

## 6.3 PK-NIKE security model

Our formalization in the public-key setting follows the "CKS-heavy" model of [18], although of course for a larger number of users. Specifically, our security experiment allows adaptive and active attacks.

**Definition 6.4** (PK-NIKE). *An $n$-user public-key-based non-interactive key exchange (PK-NIKE) consists of the following algorithms:*

**Parameter generation.** $\mathsf{Pars}(1^k)$ *chooses public parameters pars.*

**Key generation.** $\mathsf{Gen}(pars, id)$ *(for an identity $id \in \{0,1\}^k$) outputs a public key $pk_{id}$ and a secret key $sk_{id}$.*

**Shared key evaluation.** $\mathsf{Key}(pars, sk_{id}, \mathcal{I}, (pk_{id})_{id\in\mathcal{I}})$ *(for a set $\mathcal{I} \subseteq \{0,1\}^k$ of $n$ identities with $id \in \mathcal{I}$ and associated public keys $pk_{id}$) outputs a shared key $K \in \{0,1\}^k$.*

*For correctness, we require that the output of* $\mathsf{Key}$ *is deterministic and only depends on $\mathcal{I}$ and the set of public keys. Formally, for all pars $\leftarrow \mathsf{Pars}(1^k)$, all $\mathcal{I} \subseteq \{0,1\}^k$ with $|\mathcal{I}| = n$, and all $(pk_{id}, sk_{id}) \leftarrow \mathsf{Gen}(pars, id)$ (for all $id \in \mathcal{I}$), there is a $K$ with $\mathsf{Key}(pars, sk_{id}, \mathcal{I}, (pk_{id})_{id\in\mathcal{I}}) = K$ for all $id \in \mathcal{I}$, except with negligible probability over the involved $pk_{id}, sk_{id}$.*[9]

**Definition 6.5** (Adaptive PK-NIKE security). *We say that an $n$-user PK-NIKE* $\mathsf{PK\text{-}NIKE}$ *is adaptively secure if the advantage function*

$$\mathsf{Adv}^{\mathsf{id\text{-}nike}}_{\mathsf{PK\text{-}NIKE},\mathcal{A}}(k) = \Pr\left[\mathsf{Exp}^{\mathsf{pk\text{-}nike}}_{\mathsf{PK\text{-}NIKE},\mathcal{A}}(k) = 1\right] - 1/2$$

*is negligible for all PPT adversaries $\mathcal{A}$. Here, experiment $\mathsf{Exp}^{\mathsf{pk\text{-}nike}}_{\mathsf{PK\text{-}NIKE},\mathcal{A}}$ is defined as follows:*

1. *the experiment samples pars $\leftarrow \mathsf{Pars}(1^k)$, hands pars to $\mathcal{A}$, and gives $\mathcal{A}$ oracle access to the following oracles at any point:*
   - *$\mathcal{O}_{pk}$, on input $id \in \{0,1\}^k$, samples $(pk_{id}, sk_{id}) \leftarrow \mathsf{Gen}(pars, id)$ and returns $pk_{id}$,*
   - *$\mathcal{O}_{sk}$, on input $id \in \{0,1\}^k$ and $\overline{pk_{id}}$ (where $\mathcal{O}_{pk}(id)$ has been queried earlier), returns $sk_{id}$, and overwrites the previously generated key $pk_{id}$ for $id$ with $\overline{pk_{id}}$ (for later $\mathcal{O}_K$ queries),*
   - *$\mathcal{O}_K$, on input an ordered set $\mathcal{I} = \{id_1, \dots, id_n\} \subseteq \{0,1\}^k$ (where $\mathcal{O}_{pk}(id)$ has been queried for all $id \in \mathcal{I}$, and no $\mathcal{O}_{sk}$ query has been made for $id_1$), returns $\mathsf{Key}(sk_{id_1}, \mathcal{I}, (pk_{id})_{id\in\mathcal{I}})$,*
2. *at some point, $\mathcal{A}$ decides on a target set $\mathcal{I}^* = \{id_1^*, \dots, id_n^*\} \subseteq \{0,1\}^k$ of $n$ identities (where $\mathcal{O}_{pk}(id_i^*)$ has been queried earlier for all $i$),*
3. *the experiment tosses a coin $b \leftarrow \{0,1\}$; if $b = 0$, then $\mathcal{A}$ gets $\mathsf{Key}(sk_{id_1^*}, \mathcal{I}^*, (pk_{id^*})_{id^*\in\mathcal{I}^*})$, and if $b = 1$, then $\mathcal{A}$ gets a uniformly chosen $\{0,1\}^k$-element,*
4. *after potentially more oracle queries, $\mathcal{A}$ outputs a guess $b'$, and the experiment outputs 1 if $b = b'$.*

*We only quantify over $\mathcal{A}$ that guarantee that (a) $\mathcal{I}^*$ does not contain any identity queried to $\mathcal{O}_{sk}$, (b) $\mathcal{I}^*$ is never queried to $\mathcal{O}_K$, and (c) $\mathcal{O}_{pk}$ is queried at most once for each identity.*

## 6.4 Our PK-NIKE protocol

We now describe our PK-NIKE $\mathsf{PK\text{-}NIKE}$, where we use the same ingredients and assumptions from our CPRF from Section 4, *except* for the indistinguishability obfuscator. (As with our CPRF, we assume that the group setting is externally given for simplicity. To make the group setting explicit, it can be published as part of the public parameters.)

**Parameter generation.** $\mathsf{Pars}(1^k)$ samples a key derivation function $h \leftarrow \mathcal{H}$ and sets $pars = h$.

[9] We allow for a negligible correctness error in view of our upcoming PK-NIKE.

**Key generation.** $\mathsf{Gen}(pars, id)$ chooses $2nk$ random exponents $\alpha_{i,j,b}^{(id)}$ for $(i,j,b) \in [n] \times [k] \times \{0,1\}$, as well as $n$ uniform $k$-bitstrings $S_i^{id} = (s_{i,j}^{id})_{j=1}^k \in \{0,1\}^k$, and returns

$$pk_{id} = \left(id, (\left[\alpha_{i,j,b}^{(id)}\right]_1)_{(i,j,b)\in[n]\times[k]\times\{0,1\}}\right) \qquad sk_{id} = \left((S_i^{id})_{i=1}^n, (\alpha_{i,j,1-s_{i,j}^{id}}^{(id)})_{(i,j)\in[n]\times[k]}\right).$$

**Shared key evaluation.** $\mathsf{Key}(sk_{id}, \mathcal{I}, (pk_{id})_{id\in\mathcal{I}})$ parses $\mathcal{I} = \{id_1, \dots, id_n\}$ (with $id_i < id_{i+1}$), sets

$$(T_i)_{i=1}^n = \mathsf{H}((id_i, pk_i)_{i=1}^n) \in (\{0,1\}^k)^n,$$

and writes $(t_{i,j})_{j=1}^k := T_i$. The shared key $K$ is defined as

$$K = h\left(\left[\prod_{(i,j)\in[n]\times[k]} \alpha_{i,j,t_{i,j}}^{(id_i)}\right]_{nk-1}\right).$$

This key can be computed from $sk_{id_{i^*}}$ as soon as $sk_{id_{i^*}}$ contains one exponent among the $\alpha_{i^*,j,t_{i^*,j}}^{(id_{i^*})}$. This is the case if and only if $T_{i^*} \neq S_{i^*}^{id_{i^*}}$. (Hence, the scheme has a negligible correctness error.)

Again, we stress that we do not use indistinguishability obfuscation in this construction.

**Theorem 6.6** (Security of $\mathsf{PK\text{-}NIKE}$). $\mathsf{PK\text{-}NIKE}$ *is adaptively secure in the random oracle model, provided that the* $(nk-1)$*-MDDH assumption holds in* $\mathcal{G}$.

*Proof sketch.* Since the proof is very similar to the security proof of our CPRF, we only give a sketch. Intuitively, the security reduction knows all involved secret keys $sk_{id}$ during all stages of the proof, but guesses which of $\mathcal{A}$'s random oracle queries refers to the challenge set $\mathcal{I}^* = \{id_1^*, \dots, id_n^*\}$ of identities. The answer to this query is programmed to $(S_i^{id_i^*})_{i=1}^n$. If the guess was correct, then this programming will be perfectly oblivious to $\mathcal{A}$ (since the $S_i^{id_i^*}$ are perfectly hidden until the secret key of some identity $id^* \in \mathcal{I}^*$ is corrupted via an $\mathcal{O}_{sk}$ query). Furthermore, a correct guess implies that the simulation cannot compute the challenge key $\mathsf{Key}(sk_{id_1^*}, \mathcal{I}^*, (pk_{id^*})_{id^*\in\mathcal{I}^*})$. Even more, an $(nk-1)$-MDDH challenge can thus be embedded exactly as in Game 7 of our CPRF proof (or, more specifically, as in (14)). □

Note that this security reduction loses a factor of $q_{\mathsf{H}}$, where $q_{\mathsf{H}}$ denotes the number of $\mathcal{A}$'s random oracle queries. In particular, our reduction only has a polynomial loss.

## Acknowledgements

I am grateful to the Crypto 2014 referees who gave very useful and constructive feedback. Compared to the Crypto submission, this version contains a significantly extended and improved presentation (in particular of the main construction, and of the proof of Lemma 4.2), and correctly mentions a small statistical defect in the analysis of our extensible tester construction.

## References

[1] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. "On the (Im)possibility of Obfuscating Programs". In: *Proceedings of CRYPTO 2001.* Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 1–18.

[2] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. *Indistinguishability Obfuscation vs. Auxiliary-Input Extractable Functions: One Must Fall.* Cryptology ePrint Archive, Report 2013/641. `http://eprint.iacr.org/2013/641`. 2013.

[3] Dan Boneh and Xavier Boyen. “Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles”. In: *Proceedings of EUROCRYPT 2004*. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 223–238.

[4] Dan Boneh and Matthew K. Franklin. “Identity-Based Encryption from the Weil Pairing”. In: *Proceedings of CRYPTO 2001*. Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 213–229.

[5] Dan Boneh and Alice Silverberg. “Applications of Multilinear Forms to Cryptography”. In: *Contemporary Mathematics* 324 (2003), pp. 71–90.

[6] Dan Boneh and Brent Waters. “Constrained Pseudorandom Functions and Their Applications”. In: *Proceedings of ASIACRYPT (2) 2013*. Vol. 8270. Lecture Notes in Computer Science. Springer, 2013, pp. 280–300.

[7] Dan Boneh and Mark Zhandry. *Multiparty Key Exchange, Efficient Traitor Tracing, and More from Indistinguishability Obfuscation*. Cryptology ePrint Archive, Report 2013/642. `http://eprint.iacr.org/2013/642`. 2013.

[8] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. “Functional Signatures and Pseudorandom Functions”. In: *Proceedings of Public Key Cryptography 2014*. Vol. 8383. Lecture Notes in Computer Science. Springer, 2014, pp. 501–519.

[9] Zvika Brakerski and Yael Tauman Kalai. *A Framework for Efficient Signatures, Ring Signatures and Identity Based Encryption in the Standard Model*. Cryptology ePrint Archive, Report 2010/86. `http://eprint.iacr.org/2010/086`. 2010.

[10] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, Jae Hong Seo, and Christoph Striecks. “Practical Signatures from Standard Assumptions”. In: *Proceedings of EUROCRYPT 2013*. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 461–485.

[11] Ran Canetti. “Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information”. In: *Proceedings of CRYPTO 1997*. Vol. 1294. Lecture Notes in Computer Science. Springer, 1997, pp. 455–469.

[12] David Cash, Eike Kiltz, and Victor Shoup. “The Twin Diffie-Hellman Problem and Applications”. In: *Proceedings of EUROCRYPT 2008*. Vol. 4965. Lecture Notes in Computer Science. Springer, 2008, pp. 127–145.

[13] Jean-Sébastien Coron. “On the Exact Security of Full Domain Hash”. In: *Proceedings of CRYPTO 2000*. Vol. 1880. Lecture Notes in Computer Science. Springer, 2000, pp. 229–235.

[14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. “Practical Multilinear Maps over the Integers”. In: *Proceedings of CRYPTO (1) 2013*. Vol. 8042. Lecture Notes in Computer Science. Springer, 2013, pp. 476–493.

[15] Régis Dupont and Andreas Enge. “Provably secure non-interactive key distribution based on pairings”. In: *Discrete Applied Mathematics* 154.2 (2006), pp. 270–276.

[16] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge L. Villar. “An Algebraic Framework for Diffie-Hellman Assumptions”. In: *Proceedings of CRYPTO (2) 2013*. Vol. 8043. Lecture Notes in Computer Science. Springer, 2013, pp. 129–147.

[17] Amos Fiat and Adi Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *Proceedings of CRYPTO 1986*. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 186–194.

[18] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. “Non-Interactive Key Exchange”. In: *Proceedings of Public Key Cryptography 2013*. Vol. 7778. Lecture Notes in Computer Science. Springer, 2013, pp. 254–271.

[19] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. “Programmable Hash Functions in the Multilinear Setting”. In: *Proceedings of CRYPTO (1) 2013*. Vol. 8042. Lecture Notes in Computer Science. Springer, 2013, pp. 513–530.

[20] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. *Adaptive Security of Constrained PRFs*. Unpublished manuscript. 2014.

[21] Sanjam Garg, Craig Gentry, and Shai Halevi. “Candidate Multilinear Maps from Ideal Lattices”. In: *Proceedings of EUROCRYPT 2013*. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 1–17.

[22] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. “Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits”. In: *Proceedings of FOCS 2013*. IEEE Computer Society, 2013, pp. 40–49.

[23] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. “Two-Round Secure MPC from Indistinguishability Obfuscation”. In: *Proceedings of TCC 2014*. Vol. 8349. Lecture Notes in Computer Science. Springer, 2014, pp. 74–94.

[24] Rosario Gennaro, Shai Halevi, Hugo Krawczyk, Tal Rabin, Steffen Reidt, and Stephen D. Wolthusen. “Strongly-Resilient and Non-interactive Hierarchical Key-Agreement in MANETs”. In: *Proceedings of ESORICS 2008*. Vol. 5283. Lecture Notes in Computer Science. Springer, 2008, pp. 49–65.

[25] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. “How to construct random functions”. In: *J. ACM* 33.4 (1986), pp. 792–807.

[26] Shafi Goldwasser and Yael Tauman Kalai. “On the (In)security of the Fiat-Shamir Paradigm”. In: *Proceedings of FOCS 2003*. IEEE Computer Society, 2003, pp. 102–113.

[27] Shafi Goldwasser and Guy N. Rothblum. “On Best-Possible Obfuscation”. In: *Proceedings of TCC 2007*. Vol. 4392. Lecture Notes in Computer Science. Springer, 2007, pp. 194–213.

[28] Dennis Hofheinz and Eike Kiltz. “Programmable Hash Functions and Their Applications”. In: *Proceedings of CRYPTO 2008*. Vol. 5157. Lecture Notes in Computer Science. Springer, 2008, pp. 21–38.

[29] Susan Hohenberger, Amit Sahai, and Brent Waters. “Replacing a Random Oracle: Full Domain Hash from Indistinguishability Obfuscation”. In: *Proceedings of EUROCRYPT 2014*. Vol. 8441. Lecture Notes in Computer Science. Springer, 2014, pp. 201–220.

[30] Susan Hohenberger and Brent Waters. “Short and Stateless Signatures from the RSA Assumption”. In: *Proceedings of CRYPTO 2009*. Vol. 5677. Lecture Notes in Computer Science. Springer, 2009, pp. 654–670.

[31] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. “A Pseudorandom Generator from any One-way Function”. In: *SIAM J. Comput.* 28.4 (1999), pp. 1364–1396.

[32] Jonathan Katz and Nan Wang. “Efficiency improvements for signature schemes with tight security reductions”. In: *Proceedings of ACM Conference on Computer and Communications Security 2003*. ACM, 2003, pp. 155–164.

[33] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. “Delegatable pseudorandom functions and applications”. In: *Proceedings of ACM Conference on Computer and Communications Security 2013*. ACM, 2013, pp. 669–684.

[34] Venkata Koppula, Kim Ramchen, and Brent Waters. *Separations in Circular Security for Arbitrary Length Key Cycles*. Cryptology ePrint Archive, Report 2013/683. `http://eprint.iacr.org/2013/683`. 2013.

[35] Adeline Langlois, Damien Stehlé, and Ron Steinfeld. “GGHLite: More Efficient Multilinear Maps from Ideal Lattices”. In: *Proceedings of EUROCRYPT 2014*. Vol. 8441. Lecture Notes in Computer Science. Springer, 2014, pp. 239–256.

[36] Anna Lysyanskaya. “Unique Signatures and Verifiable Random Functions from the DH-DDH Separation”. In: *Proceedings of CRYPTO 2002*. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 597–612.

[37] Tal Moran and Alon Rosen. *There is no Indistinguishability Obfuscation in Pessiland*. Cryptology ePrint Archive, Report 2013/643. `http://eprint.iacr.org/2013/643`. 2013.

[38] Moni Naor and Moti Yung. “Universal One-Way Hash Functions and their Cryptographic Applications”. In: *Proceedings of STOC 1989*. ACM, 1989, pp. 33–43.

[39] Jesper Buus Nielsen. “Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case”. In: *Proceedings of CRYPTO 2002*. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 111–126.

[40] Kenneth G. Paterson and Sriramkrishnan Srinivasan. “On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups”. In: *Des. Codes Cryptography* 52.2 (2009), pp. 219–241.

[41] Amit Sahai and Brent Waters. *How to Use Indistinguishability Obfuscation: Deniable Encryption, and More.* Cryptology ePrint Archive, Report 2013/454. `http://eprint.iacr.org/2013/454`. 2013.

[42] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. “Cryptosystems based on pairing”. In: *Proceedings of SCIS 2000.* 2000, pp. 26–28.

[43] Brent Waters. “Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions”. In: *Proceedings of CRYPTO 2009.* Vol. 5677. Lecture Notes in Computer Science. Springer, 2009, pp. 619–636.

[44] Brent Waters. “Efficient Identity-Based Encryption Without Random Oracles”. In: *Proceedings of EUROCRYPT 2005.* Vol. 3494. Lecture Notes in Computer Science. Springer, 2005, pp. 114–127.