Constrained Pseudorandom Functions: Verifiable and Delegatable

Nishanth Chandran* Srinivasan Raghuraman† Dhinakaran Vinayagamurthy‡

Abstract

Constrained pseudorandom functions (introduced independently by Boneh and Waters (CCS 2013), Boyle, Goldwasser, and Ivan (PKC 2014), and Kiayias, Papadopoulos, Triandopoulos, and Zacharias (CCS 2013)), are pseudorandom functions (PRFs) that allow the owner of the secret key k to compute a constrained key k_f, such that anyone who possesses k_f can compute the output of the PRF on any input x such that $f(x) = 1$ for some predicate f. The security requirement of constrained PRFs state that the PRF output must still look indistinguishable from random for any x such that $f(x) = 0$.

Boneh and Waters show how to construct constrained PRFs for the class of bit-fixing as well as circuit predicates. They explicitly left open the question of constructing constrained PRFs that are delegatable - i.e., constrained PRFs where the owner of k_f can compute a constrained key $k_{f'}$ for a further restrictive predicate f'. Boyle, Goldwasser, and Ivan left open the question of constructing constrained PRFs that are also verifiable. Verifiable random functions (VRFs), introduced by Micali, Rabin, and Vadhan (FOCS 1999), are PRFs that allow the owner of the secret key k to prove, for any input x, that y indeed is the output of the PRF on x; the security requirement of VRFs state that the PRF output must still look indistinguishable from random, for any x for which a proof is not given.

In this work, we solve both the above open questions by constructing constrained pseudorandom functions that are simultaneously verifiable and delegatable.

1 Introduction

Pseudorandom functions were introduced by Goldreich, Goldwasser, and Micali, in their seminal paper [GGM86]. A PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ takes in a key $k \in \mathcal{K}$ and an input $x \in \mathcal{X}$. A user who has the key k can compute $F(k, x) \in \mathcal{Y}$ deterministically in polynomial time for any $x \in \mathcal{X}$. But for those who do not have the key k, the PRF output $F(k, x)$ is indistinguishable from random, even given oracle access to the function.

1.1 Verifiable and Delegatable Constrained PRFs

Constrained PRFs. Recently, the notion of constrained PRFs was proposed independently by Boneh and Waters [BW13], Boyle, Goldwasser and Ivan [BGI14] and Kiayias, Papadopoulos, Triandopoulos and Zacharias [KPTZ13], where a user with the key k can generate a constrained key k_f for some predicate (represented as a circuit) f such that the user possessing the constrained key k_f can compute $F(k, x)$ (by running a different $\mathsf{Eval}(k_f, x)$ algorithm) only when $f(x) = 1$. Otherwise, $F(k, x)$ is indistinguishable from random. Even when users possessing $k_{f_1}, \cdots, k_{f_m}$ collude, they should be able to obtain $F(k, x)$ only when $f_i(x) = 1$ for some $i \in [m]$, and for all other x in the domain, $F(k, x)$ should be indistinguishable from random. This way of partially delegating the PRF evaluation, results in interesting applications like broadcast encryption with optimal ciphertext length, identity based key exchange and so on. Boneh and Waters [BW13] construct constrained PRFs for both bit-fixing predicates and for arbitrary circuit predicates, relying on multilinear maps. Boyle *et al.* [BGI14] and Kiayias *et al.* [KPTZ13] construct constrained PRFs for prefix predicates based on [GGM86] and also achieve predicate privacy where the predicate is hidden given the constrained key.

Verifiable PRFs. PRFs inherently rely on the honesty of the evaluator for the correctness of $F(k, x)$. That is, the user who receives $F(k, x)$ has no way of checking whether the value he got is the correct PRF output or not. To address this, Micali, Rabin and Vadhan [MRV99] proposed a notion called verifiable random functions (VRFs). VRFs are extensions of PRFs where the user possessing the key k can *non-interactively* generate a proof π along

*Microsoft Research, India; Email: nichandr@microsoft.com

†Indian Institute of Technology, Madras; Email: rsrini@cse.iitm.ac.in. Work done while at Microsoft Research, India.

‡University of Toronto; Email: dhinakaran5@cs.toronto.edu. Work done while at Microsoft Research, India.

with the PRF output $F(k, x)$, so that the user receiving (y, π) will be able to verify whether the value y he received is indeed the correct PRF output $F(k, x)$. But on points $x \in \mathcal{X}$ for which π is not received, the PRF security still holds i.e the output $F(k, x)$ is indistinguishable from random. At first glance, it might seem trivial to generate such a proof through standard zero-knowledge protocols. However, such an approach has the following drawbacks. First, the interactive version of zero-knowledge cannot be used here as we would like the PRF evaluation and proof generation to be non-interactive. Second, using non-interactive zero-knowedge would require a trusted setup such as a common reference string (CRS) which would defeat the purpose of not trusting the evaluator to provide the right output of the PRF. The works of [Lys02, DY05, CL07, ACF14, BGRV09, HW10] give various constructions of VRFs under different assumptions and security guarantees.

Our Results. In this work, we construct a verifiable constrained random function (VCRF) for arbitrary constraints that can be represented as a polynomial size levelled circuit with any (fixed) polynomial depth, thus solving the main open question posed by Boyle, Goldwasser, and Ivan [BGI14]. We also extend our scheme to provide key delegation, where a constrained key k_f obtained for a constraint f can be used to generate a further constrained key $k_{f \wedge f'}$ for a constraint $f \wedge f'$. More generally, when provided with constrained keys $k_{f_1}, \cdots, k_{f_m}$ for some m, our key delegation mechanism enables us to generate a constrained key for any constraint which is of the form $(g_1(f_1, \ldots, f_m) \wedge f') \vee g_2(f_1, \ldots, f_m)$ for any two monotone boolean formulae g_1, g_2 and any constraint f'[1]. In fact, our scheme allows for multiple levels of delegation. Thus we also solve an open problem posed by [BW13] on achieving key delegation for constrained PRFs. Our construction achieves selective security based on the κ-Multilinear Decisional Diffie-Hellman assumption. Adaptive security of our scheme can be achieved through standard complexity leveraging.

Very recently, Hofheinz [Hof14] constructed a fully secure constrained PRF scheme for the "bit-fixing" constraints using multilinear maps and indistinguishability obfuscation in the random oracle model. Fuchsbauer et al. [FKPR14] give a new security reduction for constrained PRFs for prefix predicates based on PRFs in [GGM86] which incurs only quasi-polynomial factor loss.

Our results on VCRFs can be extended to functional signatures. Functional signatures were introduced recently by [BGI14, BF13, BMS13] where there is a master secret key k using which a user can sign any message m. There is also a way to generate a constrained key k_f for some f from a family $\mathcal{F}$ of constraints, and a user possessing k_f can sign only the messages m' in the range of f i.e $m' = f(m)$ for some m. One can see that a VCRF scheme for a constraint family $\mathcal{F}$ implies a functional signature scheme for the same family $\mathcal{F}$. In fact, a VCRF scheme for some $\mathcal{F}$ imply a functional unique signature scheme for the same $\mathcal{F}$ (a unique signature scheme [GO92, Lys02] is a signature scheme such that there exists only one valid signature for every message, even for adversarially generated (pk, sk) pairs). Thus our results on VCRFs imply that there also exists a functional unique signature scheme for a family of polynomial size predicate circuits of (fixed) polynomial depth. This functional unique signature scheme also allows for multiple levels of delegation in the same way as described for our VCRF scheme.

1.2 Techniques

[BW13] overview. The starting point of our construction is the constrained PRF constructed by Boneh and Waters [BW13] based on multilinear maps (we will refer to it as the BW scheme). In the main construction of BW where the constraints are circuit predicates (represented by f), they assume the existence of multilinear maps with $n + \ell$ levels, where n is the number of input wires in the circuit and ℓ is the depth of the output wire in the circuit (with the input wire at depth 1). A pair of secret values $(d_{i,0}, d_{i,1}) \leftarrow \mathbb{Z}_q^2$ are assigned for each input wire in the circuit (and the corresponding $D_{i,0} = g^{d_{i,0}}, D_{i,1} = g^{d_{i,1}}$ are added as part of the public key). The PRF output is,

$$F(k, x) = g_{n+\ell}^{u \prod_{i \in [n]} d_{i,x_i}}$$

where $u \leftarrow \mathbb{Z}_q$ is a random component exponent and is part of the secret key k. When a user obtains a constrained key k_f for some circuit f, for any wire w in f, he is somehow allowed to compute the component

$$K_w = g_{n+j}^{r_w \prod_{i \in [n]} d_{i,x_i}}$$

whenever $f_w(x) = 1$ i.e when the value on the wire w is 1 on evaluating the circuit f with input x. Here, j is the depth of wire w in circuit f. Intuitively, the security guarantees that the user will not get anything for the wire w

[1] We note that any constraint f'' such that $\forall x, f''(x) = 1 \Rightarrow f_i(x) = 1$ for some $i \in [m]$ can be represented in the given form and hence we allow the delegation to any further restrictive predicate.

when $f_w(x) = 0$. With the user receiving the K_w components for the input as a part of the constrained key k_f, by induction, he will be able to compute $g_{n+\ell}^{u\prod_{i\in[n]} d_{i,x_i}}$ when $f(x) = 1$.

How to make it verifiable? Now our goal is to provide a way to generate a proof π along with the PRF output y enabling the receiver to check whether y is the correct PRF output or not. Our first idea is to make the BW PRF output itself the proof. Now, to make this proof verifiable, we must add something related to u as part of the public key. In our construction, $U = g_{\ell+1}^u$ is given as a part of the public key. We next assume the existence of multilinear maps upto $n + \ell + 1$ levels (one more level than BW). Unfortunately, now we cannot use the BW PRF output (which is also our proof) as our PRF output because this value, if output to an adversary, is now publicly verifiable, and one can just use the public parameters to distinguish this from a randomly generated element from the same group. To overcome this issue, we first use one more level of pairing to generate the proof as

$$\pi = e\left(g_{n+\ell}^{u\prod_{i\in[n]} d_{i,x_i}}, V\right) = g_{n+\ell+1}^{uv\prod_{i\in[n]} d_{i,x_i}}$$

where $V = g^v$ is provided as a part of the public key with $v \leftarrow \mathbb{Z}_q$ being part of the secret key k (this is similar in spirit to the techniques used by Hohenberger and Waters [HW10] to construct verifiable random functions). Now the actual PRF output is obtained by using one more level of pairing, i.e

$$F(k, x) = e\left(g_{n+\ell+1}^{uv\prod_{i\in[n]} d_{i,x_i}}, W\right) = g_{n+\ell+2}^{uvw\prod_{i\in[n]} d_{i,x_i}}$$

where again $W = g^w$ is provided as a part of the public key with $w \leftarrow \mathbb{Z}_q$ being part of the secret key k. Now, a user receiving (y, π) can check whether y is the correct output or not as follows: the user first computes $\tilde{\pi} = g_{n+\ell+2}^{uv\prod_{i\in[n]} d_{i,x_i}}$ by using pairing (multiple times) the components D_{i,x_i}, U, V (which are all part of the public key), then checks the following two equations:

$$\tilde{\pi} \stackrel{?}{=} e(\pi, g)$$
$$e(\pi, W) \stackrel{?}{=} y$$

Now, one can show that it is impossible for an adversary to generate a (y, π) pair which is not valid and also satisfies these two constraints. Now, we can also show our PRF output is indistinguishable from random on any input such that $f(x) = 0$. Next, we show how a user with a constrained key k_f, can generate a proof π along with $y = \mathsf{Eval}(k_f, x) = F(k, x)$ such that y is the correct PRF output for x. To do this, we first use the constrained key k_f to compute $g_{n+\ell}^{u\prod_{i\in[n]} d_{i,x_i}}$ in the same way that the PRF output of BW is computed and then pair this with V to get $\pi = g_{n+\ell+1}^{uv\prod_{i\in[n]} d_{i,x_i}}$.

Delegating the PRF. We next show that the above construction supports key delegation. We give a mechanism through which a user possessing a constrained key k_f can delegate the evaluation of the PRF by generating a further constrained key $k_{f\wedge f'}$ for any f' where $k_{f\wedge f'}$ allows to compute $F(k, x)$ only when both $f(x) = 1$ and $f'(x) = 1$. We do this as follows. We first note, that the only secret component that prevents a user having just the public parameters, from generating a constrained key k_f for some circuit f is u. Hence any user can pick a u' of his choice and provide a $\tilde{k}_{f'}$ which would allow one to compute

$$g_{n+\ell+2}^{u'vw\prod_{i\in[n]} d_{i,x_i}}$$

whenever $f'(x) = 1$. Now, for key delegation the user possessing the constrained key k_f picks a random $u' \leftarrow \mathbb{Z}_q$ and modifies just the key component for the last gate so that given this modified key one could compute

$$g_{n+\ell+2}^{(u+u')vw\prod_{i\in[n]} d_{i,x_i}}$$

whenever $f(x) = 1$. Next, he generates the key $\tilde{k}_{f'}$ for f' using u'. The key $\tilde{k}_{f'}$ and the modified form of k_f form the key $k_{f\wedge f'}$. When a user wishes to evaluate the PRF with this key on some input x, he first uses $\tilde{k}_{f'}$ to compute $g_{n+\ell+2}^{u'vw\prod_{i\in[n]} d_{i,x_i}}$ which he can obtain only when $f'(x) = 1$. Then he uses the modified form of k_f to obtain $g_{n+\ell+2}^{(u+u')vw\prod_{i\in[n]} d_{i,x_i}}$ which he can do only when $f(x) = 1$. From these two values, he can obtain the PRF

output $F(k,x) = g_{n+\ell+2}^{uvw\prod_{i\in[n]} d_{i,x_i}}$. We can similarly do multiple levels of delegation just by "secret sharing" the component u further. In general, if the user has secret keys $k_{f_1}, \ldots, k_{f_m}$ for some m, he can provide a secret key for any constraint of the form $g_1(f_1, \ldots, f_m) \wedge f' \vee g_2(f_1, \ldots, f_m)$ where g_1 and g_2 are some monotone boolean formulae and any f'.

1.3 Organization

We briefly overview the κ-Multilinear Decisional Diffie-Hellman (MDDH) assumption and define verifiable constrained random functions (with key delegation) in Section 2. As a warmup, we present a construction of a verifiable constrained random function (VCRF) for bit-fixing predicates, in Section 3. Our VCRF construction for circuit predicates is provided in Section 4. Next, in Section 5, we show how our scheme supports key delegation. Finally, in Appendix B, we show how to transform our scheme into the setting of graded encoding systems [GGH13a] (as opposed to multilinear maps).

2 Preliminaries

Leveled multilinear groups. We assume the existence of a group generator $\mathcal{G}$, which takes as input a security paramter 1^λ and a positive integer κ to indicate the number of levels. $\mathcal{G}(1^\lambda, \kappa)$ outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ each of large prime order $p > 2^\lambda$. In addition, we let g_i be a canonical generator of $\mathbb{G}_i$ that is known from the group's description. We let $g = g_1$.

We assume the existence of a set of multilinear maps $\{e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j} | i, j \geq 1; i + j \leq \kappa\}$. The map $e_{i,j}$ satisfies the following relation:

$$e_{i,j}(g_i^a, g_j^b) = g_{i+j}^{ab} \forall a, b \in \mathbb{Z}_p.$$

When the context is obvious, we will drop the subscripts i, j. For example, we may simply write

$$e(g_i^a, g_j^b) = g_{i+j}^{ab}.$$

We define the κ-Multilinear Decisional Diffie-Hellman (κ-MDDH) assumption [GGH13a] as follows:

Assumption 2.1. (κ-Multilinear Decisional Diffie-Hellman: κ-MDDH) The κ-Multilinear Decisional Diffie-Hellman (κ-MDDH) problem is as follows: A challenger runs $\mathcal{G}(1^\lambda, \kappa)$ to generate groups and generators of order p. Then it picks random $c_1, \ldots, c_{\kappa+1} \in \mathbb{Z}_p$. The assumption states that given $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$, it is hard to distinguish the element $T = g_\kappa^{\prod_{j\in[\kappa+1]} c_j}$ from a random group element in $\mathbb{G}_\kappa$ with better than negligible advantage in the security parameter λ.

2.1 Definitions

Verifiable Constrained Random Functions. We begin by defining a verifiable constrained random function (VCRF).

Definition 2.1. Let $F : \{0,1\}^{\mathsf{seed}(\lambda)} \times \{0,1\}^{\mathsf{in}(\lambda)} \to \{0,1\}^{\mathsf{out}(\lambda)}$, where seed, in and out are all polynomials in the security parameter 1^λ, be an efficient function. We say that F is a verifiable constrained random function with respect to a set system $\mathcal{S} \subseteq 2^{\{0,1\}^{\mathsf{in}(\lambda)}}$ if there exist algorithms (Setup, $\mathsf{Constrain}$, $\mathsf{Evaluate}$, Prove, Verify) such that

- $\mathsf{Setup}(1^\lambda, 1^{\mathsf{in}(\lambda)})$ outputs a pair of keys (pk, sk);
- $\mathsf{Constrain}(sk, \mathcal{S})$ outputs a constrained key $k_\mathcal{S}$ which enables the evaluation of $F(sk, x)$ for all $x \in \mathcal{S}$ and no other x;
- $\mathsf{Evaluate}(k_\mathcal{S}, x)$ outputs $F(sk, x)$ if $x \in \mathcal{S}$ and $\perp$ otherwise, where $\perp \notin \{0,1\}^{\mathsf{out}(\lambda)}$, we will use the shorthand $F(k_\mathcal{S}, x)$ for $F.\mathsf{Evaluate}(k_\mathcal{S}, x)$;
- $\mathsf{Prove}(k_\mathcal{S}, x)$ outputs a pair $(F(k_\mathcal{S}, x), \pi_{k_\mathcal{S}}(x))$, where $\pi_{k_\mathcal{S}}(x)$ is the proof of correctness of $F(sk, x)$; and
- $\mathsf{Verify}(pk, x, y, \pi)$ verifies that $y = F(sk, x)$ using the proof π and public key pk.

Formally, the following security properties are to be satisfied:

1. **Provability:** For all $(pk, sk) \in \mathsf{Setup}(1^\lambda, 1^{\mathsf{in}(\lambda)})$ and inputs $x \in \{0,1\}^{\mathsf{in}(\lambda)}$, and for all keys output by $\mathsf{Constrain}(sk, \mathcal{S})$ (for all $\mathcal{S}$) if $x \in \mathcal{S}$ and $(y, \pi) = \mathsf{Prove}(k_{\mathcal{S}}, x)$, then $\mathsf{Verify}(pk, x, y, \pi) = 1$[2].

2. **Uniqueness:** For all $(pk, sk) \in \mathsf{Setup}(1^\lambda, 1^{\mathsf{in}(\lambda)})$ and inputs $x \in \{0,1\}^{\mathsf{in}(\lambda)}$, there does not exist a tuple (y_1, y_2, π_1, π_2) such that $y_1 \neq y_2$, $\mathsf{Verify}(pk, x, y_1, \pi_1) = 1$ and $\mathsf{Verify}(pk, x, y_2, \pi_2) = 1$.

3. **Pseudorandomness:** For all p.p.t. distinguishers $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function μ such that:

$$\Pr[(pk, sk) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{in}(\lambda)}); (x, s) \leftarrow \mathcal{A}_1^{\mathsf{Constrain}(\cdot), \mathsf{Prove}(\cdot,\cdot)}(1^\lambda, pk); y_0 = F(sk, x);$$
$$y_1 \leftarrow \{0,1\}^{\mathsf{out}(\lambda)}; b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}_2^{\mathsf{Constrain}(\cdot), \mathsf{Prove}(\cdot,\cdot)}(y_b, s) :$$
$$b = b' \wedge x \notin S] \leq \frac{1}{2} + \mu(\lambda),$$

   where $S = S_1 \cup S_2$, $S_1 = \cup_i \mathcal{S}_i$ where the $\mathcal{S}_i$'s are the queries given by $\mathcal{A}$ to its oracle $\mathsf{Constrain}$ and S_2 is the set of queries given by $\mathcal{A}$ to its oracle Prove.

Remarks. When constructing verifiable constrained random functions, it will be more convenient to work with the definition where the adversary is allowed to issue only a single challenge query. A standard hybrid argument shows that this definition is equivalent to the one where an adversary is allowed to issue multiple challenge queries. In the pseudorandomness property, we do not provide explicit oracle access to $\mathsf{Evaluate}$ as he can always obtain access to $\mathsf{Evaluate}$ through $\mathsf{Constrain}$. Finally, while the definition allows for verification of the final function value being computed, it provides no guarantee that the constrained key obtained is correct. A proof that the constrained key provided is correct can be obtained by considering the $\mathcal{NP}$ statement that proves this and one can then give an interactive zero-knowledge proof for the correctness of the constrained key produced. One could potentially have two extra algorithms $\mathsf{ProveKey}$ and $\mathsf{VerifyKey}$ which would model non-interactive proofs (and verification) that the constrained key was constructed honestly. In the case of bit-fixing, we have provide such well-defined algorithms itself which would suffice and we do not need an interactive zero-knowledge proof, as described in the construction. It would be an interesting problem to come up with such algorithms in the case of general circuits.

Key Delegation. We now proceed to the notion of key delegation, which we describe as follows. A user $\mathcal{U}$ requests the owner of the secret key sk for a constrained key corresponding to $\mathcal{S}$ and gains possession of $k_{\mathcal{S}}$. $\mathcal{U}$ then makes public the information that it possesses $k_{\mathcal{S}}$. A second user $\mathcal{U}'$ can request $\mathcal{U}$ for a constrained key corresponding to $\mathcal{S}'$. In the ideal case, we would expect that $\mathcal{S}' \subseteq \mathcal{S}$, but the definition below is general enough to accommodate for any $\mathcal{S}'$ so that $\mathcal{U}$ does not enable $\mathcal{U}'$ to compute anything more than what he himself can compute.

Definition 2.2. Let $F : \{0,1\}^{\mathsf{seed}(\lambda)} \times \{0,1\}^{\mathsf{in}(\lambda)} \to \{0,1\}^{\mathsf{out}(\lambda)}$, where seed, in and out are all polynomials in the security parameter 1^λ, be an efficient function. We say that F is a verifiable constrained random function with respect to a set system $\mathcal{S} \subseteq 2^{\{0,1\}^{\mathsf{in}(\lambda)}}$ which supports key delegation if there exist algorithms (Setup, $\mathsf{Constrain}$, KeyDel, $\mathsf{Evaluate}$, Prove, Verify) such that Setup, $\mathsf{Constrain}$, $\mathsf{Evaluate}$, Prove and Verify are as specified in Definition 2.1 and $\mathsf{KeyDel}(k_{\mathcal{S}}, \mathcal{S}')$ outputs a constrained key $k_{\mathcal{S} \cap \mathcal{S}'}$ which enables the evaluation of $F(sk, x)$ for all $x \in \mathcal{S} \cap \mathcal{S}'$ and no other x. The same properties listed in Definition 2.1 need to be satisfied.

Note that in the case that $\mathcal{S}' \subseteq \mathcal{S}$, $k_{\mathcal{S} \cap \mathcal{S}'} = k_{\mathcal{S}'}$ in the sense defined earlier, i.e., it enables the evaluation of $F(sk, x)$ for all $x \in \mathcal{S}'$ and no other x.

3 A Bit-Fixing Construction

In this section, we show how to construct a verifiable constrained random function for bit-fixing predicates. We base our construction on multilinear maps and on the κ-MDDH assumption; we later show how to transform our construction into the language of graded encodings for which a candidate construction due to [GGH13a] is known. As described in the introduction, the starting point of our construction is the constrained PRF construction of Boneh and Waters [BW13]. The "proof" for our verifiable constrained random function will take a form very similar to the constrained PRF of [BW13], thereby ensuring that anyone with a constrained key for a set $\mathcal{S}$ can

[2]This property must also hold for the secret key sk, but it is implicitly captured by the key $k_{\mathcal{S}}$ when $\mathcal{S}$ is the set of all inputs.

generate proofs for the correct evaluation of $F(k, x)$ for $x \in \mathcal{S}$. We would now need to provide a new value as the PRF output value. We do this by pairing the proof with a random element (present in the public key) to get the final PRF output. When we do this, we must be careful that the proof and PRF value are both not computable by an adversary on the challenge input x^*, but proofs must still be verifiable for all other inputs. For this, we must carefuly select the values that will go into the public key. We now describe the verifiable constrained random function construction for bit-fixing predicates.

3.1 Construction

[3] $F.\mathsf{Setup}(1^\lambda, 1^n)$:
The setup algorithm takes as input the security parameter λ and the bit length, n, of PRF inputs. The algorithm runs $\mathcal{G}(1^\lambda, \kappa = n + 2)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ of prime order p with canonical generators $g_1, \ldots, g_\kappa$, where $g = g_1$. It chooses random exponents $u, v, w \in \mathbb{Z}_p$ and $(d_{1,0}, d_{1,1}), \ldots, (d_{n,0}, d_{n,1}) \in \mathbb{Z}_p^2$ and computes $D_{i,\beta} = g^{d_{i,\beta}}$ for $i \in [n]$ and $\beta \in \{0, 1\}$. It computes $U = g^u$, $V = g^v$ and $W = g^w$. It then sets the keys as:

$$pk = (\vec{\mathbb{G}}, p, g_1, \ldots, g_\kappa, U, V, W, D_{1,0}, D_{1,1}, \ldots, D_{n,0}, D_{n,1}); sk = (\vec{\mathbb{G}}, p, g_1, \ldots, g_\kappa, u, v, w, d_{1,0}, d_{1,1}, \ldots, d_{n,0}, d_{n,1})$$

Letting x_i denote the i-th bit of $x \in \{0, 1\}^n$, the keyed pseudo-random function is defined as

$$F(sk, x) = g_\kappa^{uvw \prod_{i \in [n]} d_{i,x_i}}$$

$F.\mathsf{Constrain}(sk, \mathbf{v})$:
The constrain algorithm takes as input the secret key sk and a vector $\mathbf{v} \in \{0, 1, ?\}^n$. Let $\mathbf{V}$ be the set of indices $i \in [n]$ such that $\mathbf{v}_i \neq ?$, i.e., the indices for which the bit is fixed to either 0 or 1. The constrained key is computed as:

$$k_{\mathbf{v}} = g_{1+|\mathbf{V}|}^{uv \prod_{i \in \mathbf{V}} d_{i,\mathbf{v}_i}}$$

Note that if $\mathbf{V}$ is empty, then we interpret the product to be 1.

$F.\mathsf{Evaluate}(k_{\mathbf{v}}, x)$:
The evaluate algorithm takes as input a constrained key $k_{\mathbf{v}}$ and an input $x \in \{0, 1\}^n$. If $\exists\, i \in \mathbf{V}$ such that $x_i \neq \mathbf{v}_i$, the algorithm aborts. If $|\mathbf{V}| = n$, then all bits are fixed and the output of the function is $e(k_{\mathbf{v}}, W)$. Otherwise, through repeated application of pairing of $D_{i,x_i}, \forall i \notin \mathbf{V}$, the algorithm can compute the intermediate value

$$S = g_{n-|\mathbf{V}|}^{\prod_{i \in [n] \setminus \mathbf{V}} d_{i,x_i}}$$

and finally computes $e(e(k_{\mathbf{v}}, S), W) = e\left(g_{n+1}^{uv \prod_{i \in [n]} d_{i,x_i}}, W\right)$.

$F.\mathsf{Prove}(k_{\mathbf{v}}, x)$:
The prove algorithm takes as input a constrained key $k_{\mathbf{v}}$ and an input $x \in \{0, 1\}^n$. The algorithm outputs $F(k_{\mathbf{v}}, x)$ together with a proof $\pi = g_{n+1}^{uv \prod_{i \in [n]} d_{i,x_i}}$. Note that this value can be computed from $k_{\mathbf{v}}$, x, and the public key pk. If the evaluate algorithm aborts, then this algorithm aborts as well. There is non-$\perp$ output only if the evaluate algorithm does not abort.

$F.\mathsf{Verify}(pk, x, y, \pi)$:
The verify algorithm takes as input the public key pk, an input $x \in \{0, 1\}^n$, a value y and a proof π. It computes $\tilde{\pi} = g_{n+1}^{u \prod_{i \in [n]} d_{i,x_i}}$ through repeated use of pairing of $D_{i,x_i} \forall i \in [n]$ and U, and checks that:

$$e(\pi, g) \stackrel{?}{=} e(\tilde{\pi}, V)$$
$$e(\pi, W) \stackrel{?}{=} y$$

[3] Henceforth, we let $\mathsf{in}(\lambda) = n$ denote the length of inputs to our VCRF. $\mathsf{out}(\lambda)$ would be dictated by the function definition.

and outputs 1 if and only if both checks verify.

$F.\mathsf{ProveKey}(\mathbf{v})$**:**
The algorithm takes as input a vector $\mathbf{v} \in \{0,1,?\}^n$. The algorithm outputs $F.\mathsf{Constrain}(sk, \mathbf{v})$ together with a proof $\pi_{\mathbf{v}} = g_{1+|\mathbf{V}|}^{uv \prod_{i \in \mathbf{V}} d_{i,x_i}}$. Note that if $\mathbf{V}$ is empty, then we interpret the product to be 1.

$F.\mathsf{VerifyKey}(pk, \mathbf{v}, k, \pi_{\mathbf{v}})$**:**
The algorithm takes as input the public key pk, a vector $\mathbf{v} \in \{0,1,?\}^n$, a key k and a proof $\pi_{\mathbf{v}}$. The algorithm computes $\tilde{\pi} = g_{1+|\mathbf{V}|}^{u \prod_{i \in \mathbf{V}} d_{i,x_i}}$ through repeated use of pairing of $D_{i,x_i} \forall i \in \mathbf{V}$ and U (if $\mathbf{V}$ is empty, then $\tilde{\pi} = U$), and checks that $e(\pi_{\mathbf{v}}, g) = e(\tilde{\pi}, V)$ and outputs 1 if and only if the check verifies.

3.2 Proof of VCRF

The provability property is verifiable in a straightforward manner from the construction. The uniqueness property also follows easily from the group structure; that is, for any input, there is only one group element in $\mathbb{G}_\kappa$ that is the valid output and it is not possible to give a valid proof for another element. More formally, assume the existence of an x and a tuple (y_1, y_2, π_1, π_2) such that $y_1 \neq y_2$, $\mathsf{Verify}(pk, x, y_1, \pi_1) = 1$ and $\mathsf{Verify}(pk, x, y_2, \pi_2) = 1$. From the description of $F.\mathsf{Verify}$, we have

$$e(\pi_1, g) = e(\pi_2, g) = e(\tilde{\pi}, V) = g_{n+2}^{uv \prod_{i \in [n]} d_{i,x_i}}$$

which is fixed for a given x. So, it must be the case that $\pi_1 = \pi_2$. However, we also have that $e(\pi_1, W) = y_1$ and $e(\pi_2, W) = y_2$. Since $\pi_1 = \pi_2$, $y_1 = y_2$, which is a contradiction.

The pseudorandomness property of the VCRF is proved ahead. The proof will use the standard complexity leveraging technique of guessing the challenge x^* to prove adaptive security. This guess will cause a loss of a $1/2^n$-factor in the reduction.

Theorem 3.1. *If there exists a PPT adversary $\mathcal{A}$ that breaks the pseudorandomness property of our bit-fixing construction for n-bit inputs with advantage $\epsilon(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the $\kappa = n+2$-Multilinear Decisional Diffie-Hellman assumption with advantage $\epsilon(\lambda)/2^n$.*

Proof. The algorithm $\mathcal{B}$ first receives a $\kappa = n+2$-MDDH challenge consisting of the group sequence description $\vec{\mathbb{G}}$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ along with T, where T is either $g_\kappa^{\prod_{i \in [\kappa+1]} c_i}$ or a random group element in $\mathbb{G}_\kappa$.

Setup:
It chooses an $x^* \in \{0,1\}^n$ uniformly at random. Next, it chooses random $z_1, \ldots, z_n \in \mathbb{Z}_p$ and sets

$$D_{i,\beta} = \begin{cases} g^{c_i} & x_i^* = \beta \\ g^{z_i} & x_i^* \neq \beta \end{cases}$$

for $i \in [n]$ and $\beta \in \{0,1\}$. This corresponds to setting

$$d_{i,\beta} = \begin{cases} c_i & x_i^* = \beta \\ z_i & x_i^* \neq \beta \end{cases}$$

It then sets $u = c_{\kappa-1}$, $v = c_\kappa$ and $w = c_{\kappa+1}$. The keys are computed as before.

Constrain:
Suppose a query is made for a secret key for $\mathbf{v} \in \{0,1,?\}^n$. Let $\mathbf{V}$ be the set of indices $i \in [n]$ such that $\mathbf{v}_i \neq ?$, i.e., the indices for which the bit is fixed to either 0 or 1. If $\mathbf{v}_i = x_i^*$ for all $i \in \mathbf{V}$, then $\mathcal{B}$ aborts (note that the this happens with probability at most $1 - \frac{1}{2^n}$). Otherwise, $\mathcal{B}$ identifies an arbitrary $i \in \mathbf{V}$ such that $\mathbf{v}_i \neq x_i^*$. Through pairing of $D_{j,x_j} \forall j \in \mathbf{V} \backslash \{i\}$, U and V, it computes $g_{1+|\mathbf{V}|}^{uv \prod_{j \in \mathbf{V} \backslash \{i\}} d_{j,\mathbf{v}_j}}$ and raises it to $d_{i,\mathbf{v}_i} = z_i$ to get $k_{\mathbf{v}} = g_{1+|\mathbf{V}|}^{uv \prod_{i \in \mathbf{V}} d_{i,\mathbf{v}_i}}$ and outputs it.

Evaluate:
Suppose a query is made for a secret key for an input $x \in \{0,1\}^n$. If $x = x^*$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ identifies an arbitrary i such that $x_i \neq x_i^*$. Through pairing of $D_{j,x_j} \forall j \in [n] \backslash \{i\}$, it computes $g_{n-1}^{\prod_{j \in [n] \backslash \{i\}} d_{j,x_j}}$ and raises it to $d_{i,x_i} = z_i$ to get $H' = g_{n-1}^{\prod_{i \in [n]} d_{i,x_i}}$. Finally it computes

$$e(e(e(H', U), V), W) = F(sk, x)$$

and outputs it.

Prove:
On query input x, $\mathcal{B}$ first checks if $x = x^*$ and aborts if this is true, outputting a random bit as its guess. Otherwise, it identifies an arbitrary i such that $x_i \neq x_i^*$. Through pairing of $D_{j,x_j} \forall j \in [n] \backslash \{i\}$, U and V, it computes $g_{n+1}^{uv \prod_{j \in [n] \backslash \{i\}} d_{j,x_j}}$ and raises it to $d_{i,x_i} = z_i$ to get $\pi = g_{n+1}^{uv \prod_{i \in [n]} d_{i,x_i}}$ and computes $y = e(\pi, W)$. It outputs y and π.

[4]ProveKey:
If $\mathbf{v}_i = x_i^*$ for all $i \in \mathbf{V}$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ identifies an arbitrary $i \in \mathbf{V}$ such that $\mathbf{v}_i \neq x_i^*$. Through pairing of $D_{j,x_j} \forall j \in \mathbf{V} \backslash \{i\}$, U and V, it computes $g_{1+|\mathbf{V}|}^{uv \prod_{j \in \mathbf{V} \backslash \{i\}} d_{j,\mathbf{v}_j}}$ and raises it to $d_{i,\mathbf{v}_i} = z_i$ to get $\pi_{\mathbf{v}} = g_{1+|\mathbf{V}|}^{uv \prod_{i \in \mathbf{V}} d_{i,\mathbf{v}_i}}$. On query input x, if $\exists i \in \mathbf{V}$ such that $x_i \neq \mathbf{v}_i$, $\mathcal{B}$ aborts. If $|\mathbf{V}| = n$, then all bits are fixed and $\mathcal{B}$ sets $y = e(k_{\mathbf{v}}, W)$. Otherwise, through repeated application of pairing of $D_{i,x_i} \forall i \notin \mathbf{V}$, $\mathcal{B}$ computes the intermediate value

$$S = g_{n-|\mathbf{V}|}^{\prod_{i \in [n] \backslash \mathbf{V}} d_{i,x_i}}$$

Then it computes $e(S, k_{\mathbf{v}}) = g_{n+1}^{uv \prod_{i \in [n]} d_{i,x_i}}$ and finally computes $y = e\left(g_{n+1}^{uv \prod_{i \in [n]} d_{i,x_i}}, W\right)$. Note that if the set of indices under consideration in any product is empty, then we interpret the product to be 1. $\mathcal{B}$ finally outputs y and $\pi_{\mathbf{v}}$.

Eventually, $\mathcal{A}$ will issue a challenge input $\tilde{x}$. If $\tilde{x} = x^*$, $\mathcal{B}$ will return the value T and output the same bit as $\mathcal{A}$ does as its guess. If $\tilde{x} \neq x^*$, $\mathcal{B}$ outputs a random bit as its guess. This completes the description of the adversary $\mathcal{B}$.

We first begin by noting that when T comes from an MDDH tuple, as long as $\mathcal{B}$ does not abort, the distribution of the real game and the game executed by $\mathcal{B}$ are identical. We now analyze the probability that $\mathcal{B}$'s guess was correct. Let δ' denote $\mathcal{B}$'s output and let δ denote whether T is an MDDH tuple or not, $\delta, \delta' \in \{0,1\}$. Now

$$\begin{aligned}
\Pr[\delta' = \delta] &= \Pr[\delta' = \delta | \mathsf{abort}] \Pr[\mathsf{abort}] + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \Pr[\overline{\mathsf{abort}}] \\
&= \frac{1}{2}(1 - 2^{-n}) + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \cdot (2^{-n}) \\
&= \frac{1}{2}(1 - 2^{-n}) + \left(\frac{1}{2} + \epsilon\right) \cdot (2^{-n}) \\
&= \frac{1}{2} + \epsilon \cdot (2^{-n})
\end{aligned}$$

The set of equations shows that the advantage of $\mathcal{B}$ is $\epsilon(\lambda)/2^n$. The second equation is true since the probability of $\mathcal{B}$ not aborting is 2^{-n}. The third equation comes from the fact that the probability of the adversary winning conditioned on not aborting is the same as the original probability of the attacker winning. The reason is that the attacker's success is independent of whether $\mathcal{B}$ guessed x^*.

This completes the proof of the theorem, which establishes the pseudorandomness property of the construction. Hence, VCRF construction for the bit-fixing case is secure under the κ-MDDH assumption. □

4 A Circuit-predicate Construction

In this section, we now show how to construct a verifiable constrained random function for arbitrary polynomial size circuit predicates. Once again, we base our construction on multilinear maps and on the κ-MDDH assumption.

[4]This part of the simulation is not required for the proof since it is not part of the definition. But since the ProveKey algorithm has been presented in the construction, we present its simulation here for completeness.

Again, the starting point of our construction is the constrained PRF construction of Boneh and Waters [BW13] which is based on the attribute-based encryption construction for circuits [GGH+13b]. We will follow the same overall strategy that we did while constructing our verifiable constrained random function for bit-fixing predicates. We will use the same notation found in [BW13, GGH+13b] when describing the circuits; for details, please refer Appendix A.

4.1 Construction

$F.\mathsf{Setup}(1^\lambda, 1^n, 1^\ell)$:
The setup algorithm takes as input the security parameter λ, the bit length, n, of PRF inputs and ℓ, the maximum depth of the circuit. The algorithm runs $\mathcal{G}(1^\lambda, \kappa = n + \ell + 2)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ of prime order p with canonical generators $g_1, \ldots, g_\kappa$, where $g = g_1$. It chooses random exponents $u, v, w \in \mathbb{Z}_p$ and $(d_{1,0}, d_{1,1}), \ldots, (d_{n,0}, d_{n,1}) \in \mathbb{Z}_p^2$ and computes $D_{i,\beta} = g^{d_{i,\beta}}$ for $i \in [n]$ and $\beta \in \{0,1\}$. It computes $U = g_{\ell+1}^u$, $V = g^v$ and $W = g^w$. It then sets the keys as:

$$pk = (\vec{\mathbb{G}}, p, g_1, \ldots, g_\kappa, U, V, W, D_{1,0}, D_{1,1}, \ldots, D_{n,0}, D_{n,1}); sk = (\vec{\mathbb{G}}, p, g_1, \ldots, g_\kappa, u, v, w, d_{1,0}, d_{1,1}, \ldots, d_{n,0}, d_{n,1})$$

Letting x_i denote the i-th bit of $x \in \{0,1\}^n$, the keyed pseudo-random function is defined as

$$F(sk, x) = g_\kappa^{uvw \prod_{i \in [n]} d_{i,x_i}}$$

$F.\mathsf{Constrain}(sk, f = (n, q, A, B, \mathtt{GateType}))$:
The constrain algorithm takes as input the secret key sk and a circuit description f. The circuit has $n + q$ wires with n input wires, q gates and the wire $n + q$ designated as the output wire.

To generate a constrained key k_f, the key generation algorithm chooses random $r_1, \ldots, r_{n+q-1} \in \mathbb{Z}_p$, where we think of the the random value r_w as being associated with the wire w. It sets $r_{n+q} = u$.

Next, the algorithm generates key components for every wire w. The structure of the key components depends on whether w is an input wire, an output of an OR gate or an output of an AND gate. The key components in each case are described below.

- *Input wire*
  By convention, if $w \in [n]$, then it corresponds to the w-th input. The key component is:

$$K_w = g_2^{r_w d_{w,1}}$$

- *OR gate*
  Suppose that $w \in$ Gates and that $\mathtt{GateType}(w) =$ OR. In addition, let $j = \mathtt{depth}(w)$ be the depth of the wire w. The algorithm chooses random $a_w, b_w \in \mathbb{Z}_p$. Then, the algorithm creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)}}, K_{w,4} = g_j^{r_w - b_w \cdot r_{B(w)}}$$

- *AND gate*
  Suppose that $w \in$ Gates and that $\mathtt{GateType}(w) =$ AND. In addition, let $j = \mathtt{depth}(w)$ be the depth of the wire w. The algorithm chooses random $a_w, b_w \in \mathbb{Z}_p$. Then, the algorithm creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

The constrained key k_f consists of all these $n + q$ key components.

$F.\mathsf{Evaluate}(k_f, x)$:
The evaluate algorithm takes as input a constrained key k_f for the circuit $f = (n, q, A, B, \mathtt{GateType})$ and an input $x \in \{0,1\}^n$. The algorithm first checks that $f(x) = 1$, and if not, it aborts.

Consider the wire w at depth j. If $f_w(x) = 1$, then, the algorithm computes $E_w = g_{n+j}^{r_w \prod_{i \in [n]} d_{i,x_i}}$. If $f_w(x) = 0$, then nothing needs to be computed for that wire. The algorithm proceeds iteratively starting with computing E_1 and proceeds, in order, to compute E_{n+q}. Computing these values in order ensures that the computation on a depth $j - 1$ wire that evaluates to 1 will be defined before computing for a depth j wire. Since $r_{n+q} = u$,

$E_{n+q} = g_{n+\ell}^{u \prod_{i\in[n]} d_{i,x_i}}$.

We show how to compute E_w for all w where $f_w(x) = 1$, case-wise, according to whether the wire is an input, an OR gate or an AND gate. Define $D(x) = g_n^{\prod_{i\in[n]} d_{i,x_i}}$, which is computable through pairing operations.

- *Input wire*
  By convention, if $w \in [n]$, then it corresponds to the w-th input. Suppose $f_w(x) = 1$. Through pairing operations, it computes $g_{n-1}^{\prod_{j\in[n]\setminus\{i\}} d_{j,x_j}}$. It then computes:

$$E_w = e\left(K_w, g_{n-1}^{\prod_{j\in[n]\setminus\{i\}} d_{j,x_j}}\right) = g_{n+1}^{r_w \prod_{i\in[n]} d_{i,x_i}}$$

- *OR gate*
  Consider a wire $w \in$ Gates with $\texttt{GateType}(w) = \text{OR}$. In addition, let $j = \texttt{depth}(w)$ be the depth of the wire w. The computation is performed if $f_w(x) = 1$. Note that in this case, at least one of $f_{A(w)}(x)$ and $f_{B(w)}(x)$ must be 1. If $f_{A(w)}(x) = 1$, the algorithm computes:

$$\begin{aligned} E_w &= e(E_{A(w)}, K_{w,1}) \cdot e(K_{w,3}, D) \\ &= e\left(g_{n+j-1}^{r_{A(w)} \prod_{i\in[n]} d_{i,x_i}}, g^{a_w}\right) \cdot e\left(g_j^{r_{A(w)} - a_w \cdot r_{A(w)}}, g_n^{u \prod_{i\in[n]} d_{i,x_i}}\right) = g_{n+j}^{r_w \prod_{i\in[n]} d_{i,x_i}} \end{aligned}$$

  Otherwise, $f_{B(w)}(x) = 1$ and the algorithm computes:

$$\begin{aligned} E_w &= e(E_{B(w)}, K_{w,2}) \cdot e(K_{w,4}, D) \\ &= e\left(g_{n+j-1}^{r_{B(w)} \prod_{i\in[n]} d_{i,x_i}}, g^{b_w}\right) \cdot e\left(g_j^{r_{B(w)} - b_w \cdot r_{B(w)}}, g_n^{u \prod_{i\in[n]} d_{i,x_i}}\right) = g_{n+j}^{r_w \prod_{i\in[n]} d_{i,x_i}} \end{aligned}$$

- *AND gate*
  Consider a wire $w \in$ Gates with $\texttt{GateType}(w) = \text{AND}$. In addition, let $j = \texttt{depth}(w)$ be the depth of the wire w. The computation is performed if $f_w(x) = 1$. Note that in this case, $f_{A(w)}(x) = f_{B(w)}(x) = 1$. The algorithm computes:

$$\begin{aligned} E_w &= e(E_{A(w)}, K_{w,1}) \cdot e(E_{B(w)}, K_{w,2}) \cdot e(K_{w,3}, D) \\ &= e\left(g_{n+j-1}^{r_{A(w)} \prod_{i\in[n]} d_{i,x_i}}, g^{a_w}\right) \cdot e\left(g_{n+j-1}^{r_{B(w)} \prod_{i\in[n]} d_{i,x_i}}, g^{b_w}\right) \cdot e\left(g_j^{r_{A(w)} - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}, g_n^{u \prod_{i\in[n]} d_{i,x_i}}\right) \\ &= g_{n+j}^{r_w \prod_{i\in[n]} d_{i,x_i}} \end{aligned}$$

The procedures above are evaluated in order for all w for which $f_w(x) = 1$. Then, the algorithm computes $S = e(E_{n+q}, V)$ and finally computes $e(S, W) = F(sk, x)$.

$F.\mathsf{Prove}(k_f, x)$**:**
The prove algorithm takes as input a constrained key k_f and an input $x \in \{0,1\}^n$. The algorithm outputs $F(k_f, x)$ together with a proof $\pi = g_{n+\ell+1}^{uv \prod_{i\in[n]} d_{i,x_i}}$. Note that if the evaluate algorithm aborts, then this algorithm aborts as well. There is non-$\perp$ output only if the evaluate algorithm does not abort.

$F.\mathsf{Verify}(pk, x, y, \pi)$**:**
The verify algorithm takes as input the public key pk, an input $x \in \{0,1\}^n$, a value y and a proof π. It computes $\tilde{\pi} = g_{n+\ell+1}^{u \prod_{i\in[n]} d_{i,x_i}}$ through repeated use of pairing of $D_{i,x_i} \forall i \in [n]$ and U, and checks that:

$$e(\pi, g) \stackrel{?}{=} e(\tilde{\pi}, V)$$
$$e(\pi, W) \stackrel{?}{=} y$$

and outputs 1 if and only if both checks verify.

ProveKey and VerifyKey for this construction could be protocols as decribed earlier. We do not provide their descriptions here.

4.2 Proof of VCRF

The provability property is verifiable in a straightforward manner from the construction. The uniqueness property also follows easily from the group structure; that is, for any input, there is only one group element in $\mathbb{G}_\kappa$ that is the valid output and it is not possible to devise a valid proof for another element. More formally, assume the existence of an x and a tuple (y_1, y_2, π_1, π_2) such that $y_1 \neq y_2$, $\mathsf{Verify}(pk, x, y_1, \pi_1) = 1$ and $\mathsf{Verify}(pk, x, y_2, \pi_2) = 1$. From the description of $F.\mathsf{Verify}$, we have

$$e(\pi_1, g) = e(\pi_2, g) = e(\tilde{\pi}, V) = g_{n+2}^{uv \prod_{i\in[n]} d_{i,x_i}}$$

which is fixed for a given x. So, it must be the case that $\pi_1 = \pi_2$. However, we also have that $e(\pi_1, W) = y_1$ and $e(\pi_2, W) = y_2$. Since $\pi_1 = \pi_2$, $y_1 = y_2$, which is a contradiction.

The pseudorandomness property of the VCRF is proved ahead. The proof will use the standard complexity leveraging technique of guessing the challenge x^* to prove adaptive security. This guess will cause a loss of a $1/2^n$-factor in the reduction.

Theorem 4.1. *If there exists a PPT adversary $\mathcal{A}$ that breaks the pseudorandomness property of our circuit-predicate construction for n-bit inputs with advantage $\epsilon(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the $\kappa = n + \ell + 2$- Multilinear Decisional Diffie-Hellman assumption with advantage $\epsilon(\lambda)/2^n$.*

Proof. The algorithm $\mathcal{B}$ first receives a $\kappa = n + \ell + 2$-MDDH challenge consisting of the group sequence description $\vec{\mathbb{G}}$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ along with T, where T is either $g_\kappa^{\prod_{i\in[\kappa+1]} c_i}$ or a random group element in $\mathbb{G}_\kappa$.

Setup:
It chooses an $x^* \in \{0,1\}^n$ uniformly at random. Next, it chooses random $z_1, \ldots, z_n \in \mathbb{Z}_p$ and sets

$$D_{i,\beta} = \begin{cases} g^{c_i} & x_i^* = \beta \\ g^{z_i} & x_i^* \neq \beta \end{cases}$$

for $i \in [n]$ and $\beta \in \{0,1\}$. This corresponds to setting

$$d_{i,\beta} = \begin{cases} c_i & x_i^* = \beta \\ z_i & x_i^* \neq \beta \end{cases}$$

It then sets $u = c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+\ell+1}$, $v = c_\kappa$ and $w = c_{\kappa+1}$. The keys are computed as before.

Constrain:
Suppose a query is made for a secret key for a circuit $f = (n, q, A, B, \mathtt{GateType}$. If $f(x^*) = 1$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ generates key components for every wire w, case-wise, according to whether w is an input wire, an OR gate or an AND gate as described below.

- *Input wire*
  By convention, if $w \in [n]$, then it corresponds to the w-th input. If $x_w^* = 1$, then $\mathcal{B}$ chooses r_w at random. The key component is:
  $$K_w = e(D_{w,1}, g^{r_w}) = g_2^{r_w d_{w,1}}$$
  If $x_w^* = 0$, then $\mathcal{B}$ sets $r_w = c_{n+1}c_{n+2} + \eta_w$, where $\eta_w \in \mathbb{Z}_p$ is a randomly chosen element. The key component is:
  $$K_w = (e(g^{c_{n+1}}, g^{c_{n+2}}) \cdot g_2^{\eta_w})^{z_w} = g_2^{r_w d_{w,1}}$$
- *OR gate*
  Suppose that $w \in$ Gates and that $\mathtt{GateType}(w) = \mathrm{OR}$. In addition, let $j = \mathtt{depth}(w)$ be the depth of the wire w. If $f_w(x^*) = 1$, then $\mathcal{B}$ chooses a_w, b_w and r_w at random. Then, $\mathcal{B}$ creates key components:
  $$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)}}, K_{w,4} = g_j^{r_w - b_w \cdot r_{B(w)}}$$
  If $f_w(x^*) = 0$, then $\mathcal{B}$ sets $a_w = c_{n+j+1} + \psi_w$, $b_w = c_{n+j+1} + \phi_w$ and $r_w = c_{n+1} \cdot \ldots \cdot c_{n+j+1} + \eta_w$, where $\psi_w, \phi_w, \eta_w \in \mathbb{Z}_p$ are a randomly chosen elements. Then, $\mathcal{B}$ creates key components:
  $$K_{w,1} = g^{c_{n+j+1}+\psi_w} = g^{a_w}, K_{w,2} = g^{c_{n+j+1}+\phi_w} = g^{b_w},$$

$$K_{w,3} = g_j^{\eta_w - c_{n+j+1}\cdot\eta_{A(w)} - \psi_w(c_{n+1}\cdot\ldots\cdot c_{n+j} + \eta_{A(w)})} = g_j^{r_w - a_w \cdot r_{A(w)}},$$
$$K_{w,4} = g_j^{\eta_w - c_{n+j+1}\cdot\eta_{B(w)} - \psi_w(c_{n+1}\cdot\ldots\cdot c_{n+j} + \eta_{B(w)})} = g_j^{r_w - b_w \cdot r_{B(w)}}$$

$\mathcal{B}$ is able to create the last two key components due to a cancellation. Since $f_{A(w)}(x^*) = f_{B(w)}(x^*) = 0$, $\mathcal{B}$ would have set $r_{A(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j} + \eta_{A(w)}$ and $r_{B(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j} + \eta_{B(w)}$. Further, $g_j^{c_{n+1}\cdot\ldots\cdot c_{n+j}}$ can be computed using pairing of g^{c_i}, $n+1 \leq i \leq n+j$.

- *AND gate*
  Suppose that $w \in$ Gates and that $\texttt{GateType}(w) =$ AND. In addition, let $j = \texttt{depth}(w)$ be the depth of the wire w. If $f_w(x^*) = 1$, then $\mathcal{B}$ chooses a_w, b_w and r_w at random. Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{a_w}, K_{w,2} = g^{b_w}, K_{w,3} = g_j^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

  If $f_w(x^*) = 0$, then at least one of $f_{A(w)}(x^*)$ and $f_{B(w)}(x^*)$ must be zero. If $f_{A(w)}(x^*) = 0$, then $\mathcal{B}$ sets $a_w = c_{n+j+1} + \psi_w$, $b_w = \phi_w$ and $r_w = c_{n+1} \cdot \ldots \cdot c_{n+j+1} + \eta_w$, where $\psi_w, \phi_w, \eta_w \in \mathbb{Z}_p$ are a randomly chosen elements. Then, $\mathcal{B}$ creates key components:

$$K_{w,1} = g^{c_{n+j+1}+\psi_w} = g^{a_w}, K_{w,2} = g^{\phi_w} = g^{b_w},$$
$$K_{w,3} = g_j^{\eta_w - \psi_w \cdot c_{n+1}\cdot\ldots\cdot c_{n+j} - (c_{n+j+1}+\psi_w)\eta_{A(w)} - \phi_w \cdot r_{B(w)}} = g_j^{r_w - a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

  $\mathcal{B}$ is able to create the last key component due to a cancellation. Since $f_{A(w)}(x^*) = 0$, $\mathcal{B}$ would have set $r_{A(w)} = c_{n+1} \cdot \ldots \cdot c_{n+j} + \eta_{A(w)}$. Further, $g_j^{r_{B(w)}}$ can be directly computed and $g_j^{c_{n+1}\cdot\ldots\cdot c_{n+j}}$ can be computed using pairing of g^{c_i}, $n+1 \leq i \leq n+j$.
  The case where $f_{B(w)}(x^*) = 0$ and $f_{A(w)}(x^*) = 1$ is performed in a symmetric way to the above, with the roles of a_w and b_w interchanged.

We set, for the output wire $w = n+q$, $\eta_w = 0$, so that $r_w = u$ in our internal view. It is easy to see that a_w and b_w have the same distribution in the real game and game executed by $\mathcal{B}$, since in the real game, they are chosen at random and in the game executed by $\mathcal{B}$, they are either chosen at random or are values offset by some random values ψ_w and ϕ_w, respectively. For $w \in [n+q-1]$, r_w also has the same distribution in the real game and game executed by $\mathcal{B}$, since in the real game, they are chosen at random and in the game executed by $\mathcal{B}$, they are either chosen at random or are values offset by some random values η_w. Now, we look at r_{n+q}. In the real game, it is a fixed value u, and in the game executed by $\mathcal{B}$, by setting $\eta_{n+q} = 0$, $r_{n+q} = c_{n+1} \cdot c_{n+2} \cdot \ldots \cdot c_{n+\ell+1} = u$ internally. Hence, they too have the same distribution. Hence all the parameters in the real game and game executed by $\mathcal{B}$ have the identical distribution.

The constrained key k_f consists of all these $n+q$ key components.

Evaluate:
Suppose a query is made for a secret key for an input $x \in \{0,1\}^n$. If $x = x^*$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ identifies an arbitrary i such that $x_i \neq x_i^*$. Through pairing of g^{c_i}, $n+1 \leq i \leq n+\ell+1$, it computes $H = g_{\ell+1}^u = g_{\ell+1}^{c_{n+1}\cdot\ldots\cdot c_{n+\ell+1}}$. Then, through pairing of $D_{j,x_j} \forall j \in [n]\backslash\{i\}$, it computes $g_{n-1}^{\prod_{j\in[n]\backslash\{i\}} d_{j,x_j}}$ and raises it to $d_{i,x_i} = z_i$ to get $H' = g_{n-1}^{\prod_{i\in[n]} d_{i,x_i}}$. Then, it computes $H'' = e(H, H') = g_{n+l}^{u\prod_{i\in[n]} d_{i,x_i}}$. Finally it computes

$$e(e(H'', V), W) = F(sk, x)$$

and outputs it.

Prove:
On query input x, $\mathcal{B}$ first checks if $x = x^*$ and aborts if this is true, outputting a random bit as its guess. Otherwise, it identifies an arbitrary i such that $x_i \neq x_i^*$. Through pairing operations, it computes $g_{n+1}^{uv\prod_{j\in[n]\backslash\{i\}} d_{j,x_j}}$ and raises it to $d_{i,x_i} = z_i$ to get $\pi = g_{n+1}^{uv\prod_{i\in[n]} d_{i,x_i}}$ and computes $y = e(\pi, W)$. It outputs y and π.

Eventually, $\mathcal{A}$ will issue a challenge input $\tilde{x}$. If $\tilde{x} = x^*$, $\mathcal{B}$ will return the value T and output the same bit as $\mathcal{A}$ does as its guess. If $\tilde{x} \neq x^*$, $\mathcal{B}$ outputs a random bit as its guess.

This completes the description of the adversary $\mathcal{B}$. We first note that in the case where T is part of a MDDH tuple, the real game and game executed by $\mathcal{B}$ have the identical distribution. Secondly, in both cases (i.e., whether or not T is part of the MDDH tuple), as long as $\mathcal{B}$ does not abort, once again, the real game and game executed by $\mathcal{B}$ have the identical distribution, except for the output of $\mathcal{B}$ on the challenge query x^*. We now analyze the probability that $\mathcal{B}$'s guess was correct. Let δ' denote $\mathcal{B}$'s output and let δ denote whether T is an MDDH tuple or not, $\delta, \delta' \in \{0,1\}$. Now

$$\begin{aligned}
\Pr[\delta' = \delta] &= \Pr[\delta' = \delta | \mathsf{abort}] \Pr[\mathsf{abort}] + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \Pr[\overline{\mathsf{abort}}] \\
&= \frac{1}{2}(1 - 2^{-n}) + \Pr[\delta' = \delta | \overline{\mathsf{abort}}] \cdot (2^{-n}) \\
&= \frac{1}{2}(1 - 2^{-n}) + \left(\frac{1}{2} + \epsilon\right) \cdot (2^{-n}) \\
&= \frac{1}{2} + \epsilon \cdot (2^{-n})
\end{aligned}$$

The set of equations shows that the advantage of $\mathcal{B}$ is $\epsilon(\lambda)/2^n$. The second equation is true since the probability of $\mathcal{B}$ not aborting is 2^{-n}. The third equation comes from the fact that the probability of the attacker winning conditioned on not aborting is the same as the original probability of the attacker winning. The reason is that the attacker's success is independent of whether $\mathcal{B}$ guessed x^*.

This completes the proof of the theorem, which establishes the pseudorandomness property of the construction. Hence, VCRF construction for the circuit-predicate case is secure under the κ-MDDH assumption. □

5 Key Delegation

For bit-fixing predicates, it is trivial to delegate keys as noted in [BW13]. On obtaining $k_{\mathbf{v}} = g_{1+|\mathbf{V}|}^{uv \prod_{i \in \mathbf{V}} d_{i,\mathbf{v}_i}}$, we can generate a key $k_{\mathbf{v}'}$ (for any set $\mathbf{v}'$ formed by replacing some ? in $\mathbf{v}$ with 0 or 1) as follows:

$$k_{\mathbf{v}'} = e\left(k_{\mathbf{v}}, g_{|\mathbf{V}' \setminus \mathbf{V}|}^{\prod_{i \in \mathbf{V}' \setminus \mathbf{V}} d_{i,x_i}}\right) = g_{1+|\mathbf{V}'|}^{uv \prod_{i \in \mathbf{V}'} d_{i,\mathbf{v}'_i}}$$

Also this way we can delegate keys for multiple levels. Now, in this section, we extend the constructions for circuit-predicates to support key delegation.

5.1 Construction

We only describe KeyDel here. The other algorithms are as defined in the earlier construction.

$F.\mathsf{KeyDel}(k_f, f')$**:**
The key delegation algorithm takes as input a constrained key k_f and a circuit f'. Without loss of generality, we may assume that $w = n + q$, the output wire of the circuit f has depth greater than 1, i.e., the output wire of f is that of an AND or an OR gate. We choose a u' at random and alter the keys for the output wire as under, depending on whether it is an AND or an OR gate.

- *OR gate*

$$\tilde{K}_{w,3} = K_{w,3} \cdot g_\ell^{u'} = g_\ell^{u+u'-a_w \cdot r_{A(w)}}, \tilde{K}_{w,4} = K_{w,4} \cdot g_\ell^{u'} = g_\ell^{u+u'-b_w \cdot r_{B(w)}}$$

- *AND gate*

$$\tilde{K}_{w,3} = K_{w,3} \cdot g_\ell^{u'} = g_\ell^{u+u'-a_w \cdot r_{A(w)} - b_w \cdot r_{B(w)}}$$

In essence, we have replaced the randomness for the output wire, r_{n+q}, with $u + u'$, instead of u which was used earlier. Let $\tilde{k}_f$ be k_f with the keys for the output wire replaced suitably by the ones specified above.

Next, we obtain a "constrained key", $\tilde{k}_{f'}$ for f' with randomness for the output wire as u', instead of u. This can be done by emulating $F.\mathsf{Constrain}(sk, f')$ setting $r_{n+q} = u'$. We set the delegated key $k_{f \wedge f'} = \left(\tilde{k}_f, \tilde{k}_{f'}\right)$.

The evaluation now requires some additional work. If we proceed to evaluate as earlier, the two sets of keys now yield $g_{n+\ell}^{(u+u')\prod_{i\in[n]} d_{i,x_i}}$ and $g_{n+\ell}^{u'\prod_{i\in[n]} d_{i,x_i}}$. Dividing the first by the second gives us the required value.

Remark. For multilevel delegation, the key would be of the form $k_{f\wedge f'\wedge f''\wedge\cdots\wedge f^{(t)}} = \left(\tilde{k}_f, \tilde{k}_{f'}, \tilde{k}_{f''}, \ldots, \tilde{k}_{f^{(t)}}\right)$. Given, $k_{f\wedge f'\wedge f''\wedge\cdots\wedge f^{(t-1)}}$, the alteration of keys for the output wire would be done on $\tilde{k}_{f^{(t-1)}}$ and we compute $\tilde{k}_{f^{(t)}}$ as described for the computation of $\tilde{k}_{f'}$ in the single level delegation case above.

The evaluation again has additional steps. Suppose the randomness u was split as $u+u', u'+u'', u''+u''', \ldots, u^{(t)}$. We would now end up evaluating $g_{n+\ell}^{(u+u')\prod_{i\in[n]} d_{i,x_i}}$, $g_{n+\ell}^{(u'+u'')\prod_{i\in[n]} d_{i,x_i}}$, $g_{n+\ell}^{(u''+u''')\prod_{i\in[n]} d_{i,x_i}}$, $\ldots$, $g_{n+\ell}^{u^{(n)}\prod_{i\in[n]} d_{i,x_i}}$. Dividing one by the other in sequence will give the required value in $t-1$ steps.

5.2 Proof of VCRF with key delegation

The provability property is verifiable in a straightforward manner from the construction. The uniqueness property also follows easily from the group structure; that is, for any input, there is only one group element in $\mathbb{G}_\kappa$ that is the valid output and it is not possible to devise a valid proof for another element. The formal proof can be given as before.

The pseudorandomness property of the VCRF is shown ahead. The proof will use the standard complexity leveraging technique of guessing the challenge x^* to prove adaptive security. This guess will cause a loss of a $1/2^n$-factor in the reduction.

Theorem 5.1. *If there exists a PPT adversary $\mathcal{A}$ that breaks the pseudorandomness property of our circuit-predicate construction which supports key delegation for n-bit inputs with advantage $\epsilon(\lambda)$, then there exists a PPT algorithm $\mathcal{B}$ that breaks the $\kappa = n+\ell+2$- Multilinear Decisional Diffie-Hellman assumption with advantage $\epsilon(\lambda)/2^n$.*

Proof. The algorithm $\mathcal{B}$ first receives a $\kappa = n+\ell+2$-MDDH challenge consisting of the group sequence description $\vec{\mathbb{G}}$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ along with T, where T is either $g_\kappa^{\prod_{i\in[\kappa+1]} c_i}$ or a random group element in $\mathbb{G}_\kappa$.

We only describe how $\mathcal{B}$ emulates KeyDel here. The other algorithms are emulated as in the earlier proof.

KeyDel:
Suppose a query is made for a secret key for f' by constraining k_f. We have two cases here. Suppose that a query to obtain a contrained key for f, i.e., for k_f has already been made[5], either by virtue of Constrain or KeyDel, then we assume that $\mathcal{B}$ has k_f and it proceeds. Otherwise, $\mathcal{B}$ aborts, claiming that it does not have k_f.

If $\mathcal{B}$ does not abort, without loss of generality, we may assume that $w = n+q$, the output wire of the circuit f has depth greater than 1, i.e., the output wire of f is that of an AND or an OR gate. $\mathcal{B}$ chooses a u' at random and alter the keys for the output wire as under, case-wise, depending on whether it is an AND or an OR gate.

- *OR gate*
$$\tilde{K}_{w,3} = K_{w,3}\cdot g_\ell^{u'} = g_\ell^{u+u'-a_w\cdot r_{A(w)}}, \tilde{K}_{w,4} = K_{w,4}\cdot g_\ell^{u'} = g_\ell^{u+u'-b_w\cdot r_{B(w)}}$$
- *AND gate*
$$\tilde{K}_{w,3} = K_{w,3}\cdot g_\ell^{u'} = g_\ell^{u+u'-a_w\cdot r_{A(w)}-b_w\cdot r_{B(w)}}$$

Let $\tilde{k}_f$ be k_f with the keys for the output wire replaced suitably by the ones specified above.

Next, we obtain a "constrained key", $\tilde{k}_{f'}$ for f' with randomness for the output wire as u', instead of u. This can be done by emulating $F.\mathsf{Constrain}(sk, f')$ setting $r_{n+q} = u'$.

$\mathcal{B}$ outputs the delegated key $k_{f\wedge f'} = \left(\tilde{k}_f, \tilde{k}_{f'}\right)$.

[5]This can always be kept track of. For instance, assume that $\mathcal{B}$ maintains a table with tuples of the form (f, k_f), and it updates this table after every Constrain or KeyDel query for which it returns a non-$\perp$ output.

The evaluation again requires some additional work. The two sets of keys can be used to obtain $g_{n+\ell}^{(u+u')\prod_{i\in[n]} d_{i,x_i}}$ and $g_{n+\ell}^{u'\prod_{i\in[n]} d_{i,x_i}}$. Dividing the first by the second gives the required value.

This completes the description of the adversary $\mathcal{B}$. The analysis of the probability that $\mathcal{B}$'s guess was correct is as before. This completes the proof of the theorem, which establishes the pseudorandomness property of the construction. Hence, VCRF construction for the circuit-predicate case supporting key delegation is secure under the κ-MDDH assumption. □

5.3 More general forms of delegation

Our scheme can in fact support delegations much more than getting the constrained key $k_{f\wedge f'}$ from k_f. Consider a user who has constrained keys $k_{f_1}, \ldots, k_{f_m}$ for some m which is polynomial in the security parameter λ. This user can now generate a secret key for any constraint of the form $g = g_1(f_1, \ldots, f_m) \wedge f' \vee g_2(f_1, \ldots, f_m)$ for any two monotone boolean formulae g_1, g_2 and for any constraint f'. One simple way to answer a query of the above form using our scheme is to convert the formula g into its Disjunctive Normal Form (DNF). Note that when g is represented in DNF, it will be of the form $\bigvee(f_i \wedge h)$ for some circuit h which is a conjunction involving some of $f_1, \ldots, f_m, f'$ and for $i \in [m]$. The user can now generate keys for each clause separately by using our KeyDel scheme (through multiple levels of delegation) described in the previous section because the clause has one of $f_1, \ldots, f_m$ for which the constrained key is known. Now the constrained key k_g is set of the constrained keys for each of the individual clauses.

References

[ACF14] Michel Abdalla, Dario Catalano, and Dario Fiore. Verifiable random functions: Relations to identity-based key encapsulation and new constructions. *J. Cryptology*, 27(3):544–593, 2014.

[BF13] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. Cryptology ePrint Archive, Report 2013/413, 2013.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public Key Cryptography*, pages 501–519, 2014.

[BGRV09] Zvika Brakerski, Shafi Goldwasser, Guy N. Rothblum, and Vinod Vaikuntanathan. Weak verifiable random functions. In *TCC*, pages 558–576, 2009.

[BMS13] Michael Backes, Sebastian Meiser, and Dominique Schrder. Delegatable functional signatures. Cryptology ePrint Archive, Report 2013/408, 2013.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT (2)*, pages 280–300, 2013.

[CL07] Melissa Chase and Anna Lysyanskaya. Simulatable vrfs with applications to multi-theorem nizk. In *CRYPTO*, pages 303–322, 2007.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography*, pages 416–431, 2005.

[FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained prfs. Cryptology ePrint Archive, Report 2014/416, 2014.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO (2)*, pages 479–499, 2013.

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

[GO92] Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In *CRYPTO*, pages 228–245, 1992.

[Hof14] Dennis Hofheinz. Fully secure constrained pseudorandom functions using random oracles. Cryptology ePrint Archive, Report 2014/372, 2014.

[HW10] Susan Hohenberger and Brent Waters. Constructing verifiable random functions with large input spaces. In *EUROCRYPT*, pages 656–672, 2010.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM Conference on Computer and Communications Security*, pages 669–684, 2013.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the dh-ddh separation. In *CRYPTO*, pages 597–612, 2002.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

A Circuit Notation

We restrict our consideration to certain classes of boolean circuits. First, our circuits will have a single output gate. Next, we will consider layered circuits. In a layered circuit, a gate at depth j will receive both of its inputs from wires at depth $j-1$. Finally, we restrict ourselves to monotonic circuits where gates are either AND or OR gates of two inputs.[6]

Our circuits will be a five tuple $f = (n, q, A, B, \texttt{GateType})$. We let n be the number of inputs and q be the number of gates. We define inputs $= [n]$, Wires $= [n+q]$ and Gates $= [n+q] \backslash [n]$. The wire $n+q$ is designated as the output wire, outputwire. A : Gates $\rightarrow$ Wires$\backslash\{$outputwire$\}$ is a function where $A(w)$ identifies w's first incoming wire and B : Gates $\rightarrow$ Wires$\backslash\{$outputwire$\}$ is a function where $B(w)$ identifies w's second incoming wire. Finally, $\texttt{GateType}$: Gates $\rightarrow \{\text{AND}, \text{OR}\}$ is a function that identifies a gate as either an AND gate or an OR gate.

We require that $w > B(w) > A(w)$. We also define a function $\texttt{depth}(w)$ where if $w \in$ inputs, $\texttt{depth}(w) = 1$ and in general $\texttt{depth}(w)$ of a wire w is equal to the length of the shortest path to an input wire plus one. Since our circuit is layered, we require that for all $w \in$ Gates, if $\texttt{depth}(w) = j$, then $\texttt{depth}(A(w)) = \texttt{depth}(B(w)) = j-1$.

We will abuse notation and let $f(x)$ be the evaluation of the circuit f on the input $x \in \{0,1\}^n$. On addition, we let $f_w(x)$ be the value of the wire w of the circuit on the input x.

B VCRFs using graded encoding systems

In this section, we describe how to map our contructions for VCRFs using multilinear maps to the graded encoding realization of Garg, Gentry and Halevi [GGH13a]. We will describe the translation of our bit-fixing[7] explicity. The translation of our circuit construction follows in a similar manner. For simplicity, we drop the **params** argument given as input to the algorithms.

B.1 A Bit-Fixing Construction

B.1.1 Construction

$F.\mathsf{Setup}(1^\lambda, 1^n)$**:**
The setup algorithm takes as input the security parameter λ and the bit length, n, of PRF inputs. The algorithm runs $\mathbf{InstGen}(1^\lambda, n+1)$ to obtain the graded encoding system, $\mathbf{G}$. Next, it runs the algorithm **samp** $(2n+3)$ times to generate random level-0 encodings t_g, t_{uv}, t_w and $D_{i,\beta}$ for $i \in [n]$ and $\beta \in \{0,1\}$. It computes $T_{uv} = \mathbf{encode}(1, t_{uv})$. It then sets the keys as:

$$pk = (\mathbf{G}, t_g, T_{uv}, t_w, D_{1,0}, D_{1,1}, \ldots, D_{n,0}, D_{n,1}), sk = (\mathbf{G}, t_g, t_{uv}, t_w, D_{1,0}, D_{1,1}, \ldots, D_{n,0}, D_{n,1})$$

[6] These restrictions are mostly useful for exposition and do not implact functionality. General circuits can be built from non-monotonic circuits. In addition, given a circuit, an equivalent layered circuit exists that is larger by atmost a polynomial factor.

[7] We omit the descriptions of $F.\mathsf{ProveKey}$ and $F.\mathsf{VerifyKey}$ here, since they are not required. However, they can be translated in a way similar to the descriptions given for the other algorithms.

Letting x_i denote the i-th bit of $x \in \{0,1\}^n$, the keyed function is defined as

$$F(sk, x) = \mathbf{extract}\left(p_{zt}, t_{uv} \cdot t_w \cdot \prod_{i \in [n]} D_{i,x_i}\right)$$

where the multiplication is interpreted as repeated application of the multiplication algorithm in the graded encoding system.

$F.\mathsf{Constrain}(sk, \mathbf{v})$**:**
The constrain algorithm takes as input the secret key sk and a vector $\mathbf{v} \in \{0,1,?\}^n$. Let $\mathbf{V}$ be the set of indices $i \in [n]$ such that $\mathbf{v}_i \neq ?$, i.e., the indices for which the bit is fixed to either 0 or 1. The constrained key is computed as:

$$k_{\mathbf{v}} = \mathbf{rerand}\left(t_{uv} \cdot \prod_{i \in \mathbf{V}} D_{i,x_i}\right)$$

Note that if $\mathbf{V}$ is empty, then we interpret the product to be 1.

$F.\mathsf{Evaluate}(k_{\mathbf{v}}, x)$**:**
The evaluate algorithm takes as input a constrained key $k_{\mathbf{v}}$ and an input $x \in \{0,1\}^n$. If $\exists i \in \mathbf{V}$ such that $x_i \neq \mathbf{v}_i$, the algorithm aborts. If $|\mathbf{V}| = n$, then all bits are fixed and the output of the function is $\mathbf{extract}(p_{zt}, t_w \cdot k_{\mathbf{v}})$. Otherwise, through repeated application of the mutliplication algorithm on $D_{i,x_i} \forall i \notin \mathbf{V}$, the algorithm can compute the intermediate value

$$S = k_{\mathbf{v}} \cdot \prod_{i \in [n] \setminus \mathbf{V}} D_{i,x_i}$$

and finally computes $\mathbf{extract}(p_{zt}, t_w \cdot k_{\mathbf{v}} \cdot S)$.

$F.\mathsf{Prove}(k_{\mathbf{v}}, x)$**:**
The prove algorithm takes as input a constrained key $k_{\mathbf{v}}$ and an input $x \in \{0,1\}^n$. The algorithm outputs $F(k_{\mathbf{v}}, x)$ together with a proof $\pi = t_{uv} \cdot \prod_{i \in [n]} D_{i,x_i}$. Note that if the evaluate algorithm aborts, then this algorithm aborts as well. There is non-$\perp$ output only if the evaluate algorithm does not abort.

$F.\mathsf{Verify}(pk, x, y, \pi)$**:**
The verify algorithm takes as input the public key pk, an input $x \in \{0,1\}^n$, a value y and a proof π. It computes $\tilde{\pi} = T_{uv} \cdot \prod_{i \in [n]} D_{i,x_i}$ through repeated use of multiplication of $D_{i,x_i} \forall i \in [n]$ and T_{uv}, and checks that:

$$\mathbf{extract}(p_{zt}, t_g \cdot \pi) \stackrel{?}{=} \mathbf{extract}(p_{zt}, \tilde{\pi})$$
$$\mathbf{extract}(p_{zt}, t_w \cdot \pi) \stackrel{?}{=} y$$

and outputs 1 if and only if both checks verify.

B.1.2 Proof of VCRF

The provability property is verifiable in a straightforward manner from the construction. The uniqueness property also follows easily from the group structure; that is, for any input, there is only one group element in $\mathbb{G}_\kappa$ that is the valid output and it is not possible to devise a valid proof for another element. More formally, assume the existence of an x and a tuple (y_1, y_2, π_1, π_2) such that $y_1 \neq y_2$, $\mathsf{Verify}(pk, x, y_1, \pi_1) = 1$ and $\mathsf{Verify}(pk, x, y_2, \pi_2) = 1$. From the description of $F.\mathsf{Verify}$, we have

$$\mathbf{extract}(p_{zt}, t_g \cdot \pi_1) = \mathbf{extract}(p_{zt}, t_g \cdot \pi_2) = \mathbf{extract}(p_{zt}, \tilde{\pi})$$

So, $t_g \cdot \pi_1$, $t_g \cdot \pi_2$ and $\tilde{\pi}$ must be level-n encodings of the same α, for some α. So, it must be the case that π_1 and π_2 are level-$(n-1)$ encodings of the same α', for some α' and hence $t_w \cdot \pi_1$ and $t_w \cdot \pi_2$ are level-n encodings of the same α'', for some α''. Since $\mathbf{extract}(p_{zt}, t_w \cdot \pi_1) = y_1$ and $\mathbf{extract}(p_{zt}, t_w \cdot \pi_2) = y_2$, $y_1 = y_2$, which is a contradiction.

The pseudorandomness property/security of the VCRF can be proved as before using an analogue of the κ-Multilinear Decisional Diffie-Hellman stated in the language of graded encodings given in [GGH13a].

Remark. The subscript uv for t is used since we have combined the u and v of our scheme here. The other subscripts are self-explanatory and reflect the relevant parameters of our scheme. As such, this construction mirrors the previous one. Another point to note is that in our construction as well, the public and secret keys can be made to look similar to the ones used here with all evaluations being done by pairings.