A Verifiable Random Function With Short Proofs and Keys

Yevgeniy Dodis* and Aleksandr Yampolskiy**

Abstract. We give a simple and efficient construction of a verifiable random function (VRF) on bilinear groups. Our construction is direct. In contrast to prior VRF constructions [14, 15], it avoids using an inefficient Goldreich-Levin transformation, thereby saving several factors in security. Our proofs of security are based on a decisional bilinear Diffie-Hellman inversion assumption, which seems reasonable given current state of knowledge. For small message spaces, our VRF's proofs and keys have constant size. By utilizing a collision-resistant hash function, our VRF can also be used with arbitrary message spaces. We show that our scheme can be instantiated with an elliptic group of very reasonable size. Furthermore, it can be made distributed and proactive.

1 Introduction

The notion of a **verifiable random function** (VRF) was introduced by Micali, Rabin, and Vadhan [15]. A VRF is a pseudo-random function that provides a non-interactively verifiable proof for the correctness of its output. Given an input value x, the knowledge of the secret key SK enables computing the function value $y = F_{SK}(x)$ together with the proof of correctness π_x. This proof convinces every verifier that the value $y = F_{SK}(x)$ is indeed correct with respect to the public key of the VRF. We can thus view a VRF as a commitment to an exponential number of random-looking bits.

Since their introduction, VRFs have found useful applications in protocol design. To give a few examples, in [16], VRFs were used to reduce the number of rounds for resettable zero-knowledge proofs to three in the bare model. Micali and Rivest [17] used VRFs to construct a non-interactive lottery system employed in micropayments. Recently, Jarecki and Shmatikov [12] constructed a verifiable transaction escrow scheme, which preserves users' anonymity while enabling automatic de-escrow, again with the help of VRFs.

Unfortunately, despite their utility, VRFs are not very well studied. As of this moment, there exist only a handful of constructions in the standard model: [8, 14, 15]. With the exception of [8], these works first construct a **verifiable**

* Department of Computer Science, New York University, 251 Mercer Street, New York, NY 10012, USA. Email: `dodis@cs.nyu.edu`. Supported in part by NSF CAREER award CCR-0133806 and NSF grant CCR-0311095.

** Department of Computer Science, Yale University, 51 Prospect Street, New Haven, CT 06511, USA. Email: `aleksandr.yampolskiy@yale.edu`. Supported by NSF grants CCR-0098078, ANI-0207399, CNS-0305258, and CNS-0435201.

unpredictable function (VUF), whose output is hard to predict but does not necessarily look random. Then, they use an inefficient Goldreich-Levin hardcore bit [10] to convert a VUF into a VRF, thereby losing a factor in security. The size of proofs and keys of VRFs in [8, 14] is linear in the input size, which may be undesirable in resource-constrained environments. Meanwhile, the VRF of Micali-Rabin-Vadhan [15] operates over a large multiplicative group $\mathbb{Z}_n^*$ which has to be very large to achieve reasonable security. Before the VRF value can be computed, it requires inputs to be mapped to primes in a complicated fashion.

In this paper, we construct a simple VRF on groups equipped with bilinear maps. Our construction is direct; it does not use a Goldreich-Levin hardcore bit, saving several factors in security. The inputs need not be primes or codewords of some special encoding. For small inputs, our VRF has constant size proofs and keys. We show that by utilizing a collision-resistant hash function, we can use our VRF with arbitrary inputs as well. Our VRF can be made distributed and proactive.

We begin in Section 2 by formalizing the notions of a VRF and a VUF. We also review the definition of bilinear groups, which are used in our constructions. These groups, recently discovered by Joux and Nguyen [13], have the property that decisional Diffie-Hellman (DDH) assumption (given g, g^a, and g^b, distinguish g^{ab} from random) becomes easy, but computational Diffie-Hellman (CDH) assumption (given g, g^a, and g^b, compute g^{ab}) still remains hard. This fact gives us many useful properties like verifiability.

Our proofs of security rely on two assumptions, which we describe in Section 3. Informally, they are:

- **q-Diffie-Hellman inversion assumption** (q-DHI) states that no efficient algorithm can compute $g^{1/x}$ on input $\left(g, g^x, \ldots, g^{(x^q)}\right)$ [18];
- **q-decisional bilinear Diffie-Hellman inversion assumption** (q-DBDHI) states that no efficient algorithm can distinguish $e(g, g)^{1/x}$ from random even after seeing $\left(g, g^x, \ldots, g^{(x^q)}\right)$ [3]. (Here $e(\cdot, \cdot)$ is a bilinear map, which we define later.)

In Section 4, we give our constructions and analyze their efficiency.

First, in Section 4.1, we consider a signature due to Boneh and Boyen [4]. On input x and a secret key SK, the signature is $\textsc{Sign}_{SK}(x) = g^{1/(x+SK)}$. Boneh and Boyen proved this signature to be existentially unforgeable against **non-adaptive** adversaries. By restricting inputs to have slightly superlogarithmic size (in security parameter), we are able to prove security against **adaptive** adversaries. As a result, our proof is more involved, but necessarily less tight than the proof of [4]. We thus obtain a VUF, which is secure for small inputs. This VUF can then be converted into a VRF using the approach of prior works [14,15]. Specifically, we could use the Goldreich-Levin hardcore bit [10] to convert it into a VRF with output size 1, amplify the output size to match the size of the input, and then follow a tree-based construction to get a VRF with arbitrary input size. Needless, to say this is rather inefficient.

Instead, we prefer to construct a VRF directly (Section 4.2), saving several factors in security. We give a simple direct VRF construction for small inputs, which is secure under the q-DBDHI assumption. On input x and a secret key SK, our VRF computes $(F_{SK}(x), \pi(x))$, where $F_{SK}(x) = e(g,g)^{1/(x+SK)}$ is the VRF value and $\pi(x) = g^{1/(x+SK)}$ is the proof of correctness. We can apply a collision-resistant hash function to large inputs to transform our VRF into a VRF with unrestricted input length. By making the group size sufficiently large, we can construct a VRF with inputs of size roughly 160 bits, which is the length of SHA-1 digests. In theory, we do not have to assume existence of collision-resistant hash functions, and could also apply a variant of a generic tree transformation to amplify the input size. Even though keys and proofs no longer have constant size, they are still shorter than the keys and proofs in constructions of [14,15]. We analyze how large the group has to be and how our VRF compares with other constructions in Section 4.4.

Evaluating the VRF at a single server is a performance bottleneck and a single point of failure. Naturally, in Section 5, we sketch how to make our VRF distributed and proactive.

In Section 6, we analyze the q-DBDHI assumption in the generic group model à la Shoup [21]. We show that if the adversary can distinguish $e(g,g)^{1/x}$ from random with probability $\frac{1}{2} + \epsilon$, he will need to perform (at least) $\Omega(\sqrt{\epsilon p/q})$ generic group operations in a group of size p.

We conclude in Section 7.

2 Definitions

Before presenting our results, we review some basic definitions and assumptions.

Let k be a security parameter. As customary, we model the protocol participants by probabilistic Turing machines whose running time is polynomial in k (abbreviated as PPTs). Hereafter, we use $negl(k)$ to refer to a negligible function in the security parameter k.[1]

2.1 VRFs and VUFs

Let $a : \mathbb{N} \mapsto \mathbb{N} \cup \{*\}$ and $b : \mathbb{N} \mapsto \mathbb{N}$ be any functions for which $a(k)$ and $b(k)$ are computable in poly(k) time (except when a takes the value $*$).[2]

Intuitively, a **verifiable random function (VRF)** behaves like a pseudorandom function, but also provides proofs of its outputs' correctness.

Definition 1. *A function family $F_{(\cdot)}(\cdot) : \{0,1\}^{a(k)} \mapsto \{0,1\}^{b(k)}$ is a family of VRFs if there exists a PPT algorithm* Gen *and deterministic algorithms* Prove *and* Ver *such that* Gen(1^k) *outputs a pair of keys (PK, SK);* Prove$_{SK}(x)$ *computes $\big(F_{SK}(x), \pi_{SK}(x)\big)$, where $\pi_{SK}(x)$ is the proof of correctness; and* Ver$_{PK}(x, y, \pi)$ *verifies that $y = F_{SK}(x)$ using the proof π. Formally, we require:*

[1] A function $negl(k) : \mathbb{N} \mapsto (0,1)$ is **negligible** if for every $c > 0$, for all sufficiently large k, $negl(k) < 1/k^c$. See any standard reference, such as [11], for details.

[2] When $a(k)$ takes the value of $*$, it means the VRF is defined for inputs of all length.

1. ***Uniqueness****: no values* $(PK, x, y_1, y_2, \pi_1, \pi_2)$ *can satisfy* $\textsc{Ver}_{PK}(x, y_1, \pi_1) = \textsc{Ver}_{PK}(x, y_2, \pi_2)$ *when* $y_1 \neq y_2$.
2. ***Provability****: if* $(y, \pi) = \textsc{Prove}_{SK}(x)$*, then* $\textsc{Ver}_{PK}(x, y, \pi) = 1$.
3. ***Pseudorandomness****: for any PPT algorithm* $\mathcal{A} = (A_1, A_2)$*, who does not query its oracle on* x *(see below),*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow \textsc{Gen}(1^k); (x, st) \leftarrow A_1^{\textsc{Prove}(\cdot)}(PK); \\ y_0 = F_{SK}(x); y_1 \leftarrow \{0,1\}^{b(k)}; \\ b \leftarrow \{0,1\}; b' \leftarrow A_2^{\textsc{Prove}(\cdot)}(y_b, st) \end{array} \right] \leq \frac{1}{2} + negl(k)$$

A **verifiable unpredictable function (VUF)** is a close relative of a VRF. Essentially, it is a signature scheme, whose verification algorithm accepts at most one signature for every public key and message.

Definition 2. *A function family* $F_{(\cdot)}(\cdot) : \{0,1\}^{a(k)} \mapsto \{0,1\}^{b(k)}$ *is a family of VUFs, if it satisfies the same syntax, uniqueness and provability properties of the VRFs, except the pseudorandomness property is replaced by the following weaker property:*

3' . ***Unpredictability****: for any PPT algorithm* $\mathcal{A}$*, who does not query its oracle on* x *(see below),*

$$\Pr\left[y = F_{SK}(x) \;\middle|\; (PK, SK) \leftarrow \textsc{Gen}(1^k); (x, y) \leftarrow \mathcal{A}^{\textsc{Prove}(\cdot)}(PK) \right] \leq negl(k)$$

For exact security bounds, we will occasionally say that $F_{(\cdot)}(\cdot)$ is an $(s'(k), \epsilon'(k))$ secure VRF (*resp.*, VUF) if no adversary $\mathcal{A}$, running in time $s'(k)$, can break the pseudorandomness (*resp.*, unpredictability) property with $\epsilon'(k)$ advantage.

2.2 Bilinear Groups

Our constructions utilize bilinear maps. We briefly review their properties below.

Let $\mathbb{G}$ and $\mathbb{G}_1$ be two (multiplicative) cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$. We shall call a mapping **bilinear** if it is linear with respect to each of its variables. Formally:

Definition 3. *An (admissible) bilinear map* $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$ *is a map with the following properties:*

1. **Bilinear:** *for all* $u, v \in \mathbb{G}$ *and* $x, y \in \mathbb{Z}$*, we have* $e(u^x, v^y) = e(u, v)^{xy}$.
2. **Non-degenerate:** $e(g, g) \neq 1$.
3. **Computable:** *there is an efficient algorithm to compute* $e(u, v)$ *for all* $u, v \in \mathbb{G}$.

We say that a group $\mathbb{G}$ is **bilinear** if the group action in $\mathbb{G}$ is efficiently computable and there exists a group $\mathbb{G}_1$ and an admissible bilinear map $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$. Henceforth, we shall use $\mathbb{G}^*$ to stand for $\mathbb{G} \backslash \{1_{\mathbb{G}}\}$.

Bilinear maps provide an algorithm for solving the decisional Diffie-Hellman problem (DDH) in $\mathbb{G}$ [3]; this property comes in handy for constructing a verification algorithm for our VRF. Such maps can be constructed from Weil and Tate pairings on elliptic curves or abelian varieties [5, 9, 13].

3 Complexity Assumptions

We now state the hardness assumptions on which our constructions are based. In what follows, we let $\mathbb{G}$ be a bilinear group of prime order p, and let g be its generator.

3.1 Diffie-Hellman Inversion Assumption

Our VUF construction relies on the Diffie-Hellman inversion (DHI) assumption, which was originally proposed in [18].

The q-DHI problem in $\mathbb{G}$ asks: given the tuple $\left(g, g^x, \ldots, g^{(x^q)}\right) \in (\mathbb{G}^*)^{q+1}$ as input, compute $g^{1/x}$. An algorithm $\mathcal{A}$ has advantage ϵ in solving q-DHI in $\mathbb{G}$ if

$$\Pr\left[\mathcal{A}(g, g^x, \ldots, g^{(x^q)}) = g^{1/x}\right] \geq \epsilon,$$

where probability is taken over the coin tosses of $\mathcal{A}$ and the random choice of $x \in \mathbb{Z}_p^*$.[4]

Definition 4. *(q-DHI assumption) We say that (t, q, ϵ)-DHI assumption holds in $\mathbb{G}$ if, no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in solving the q-DHI problem in $\mathbb{G}$.*

Boneh and Boyen [3] pointed out that the q-DHI assumption implies the $(q+1)$-generalized Diffie-Hellman assumption (GDH), on which many cryptographic constructions are based (*e.g.*, [6, 19, 22] as well as the VUF in [14]). Therefore, security of our VUF rests on an equivalent complexity assumption to the one made before.

3.2 Decisional Bilinear Diffie-Hellman Inversion Assumption

In order to construct a VRF directly, we need to make a decisional bilinear Diffie-Hellman inversion assumption (DBDHI). It was previously used in [3] to construct a selective-ID secure identity based encryption scheme.

The q-DBDHI problem asks: given the tuple $\left(g, g^x, \ldots, g^{(x^q)}\right)$ as input, distinguish $e(g, g)^{1/x}$ from random. Formally, an algorithm $\mathcal{A}$ has advantage ϵ in solving the q-DBDHI problem if

---

[3] Specifically, to determine whether (g, g^x, g^y, g^z) is a DDH tuple, we can check if $e(g^x, g^y) = e(g, g^z)$.

[4] To simplify the notation, from now on, we assume that algorithms implicitly get a description of the bilinear group $(\mathbb{G}, \circ, p)$, on which they operate, as input.

$$\left| \Pr\left[\mathcal{A}(g, g^x, \ldots, g^{(x^q)}, e(g,g)^{1/x}) = 1\right] - \Pr\left[\mathcal{A}(g, g^x, \ldots, g^{(x^q)}, \Gamma) = 1\right]\right| \leq \epsilon,$$

where the probability is taken over the internal coin tosses of $\mathcal{A}$ and choices of $x \in \mathbb{Z}_p^*$ and $\Gamma \in \mathbb{G}_1$.

Definition 5. *(q-DBDHI assumption) We say that the (t, q, ϵ)-DBDHI assumption holds in $\mathbb{G}$ if no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in solving the q-DBDHI problem in $\mathbb{G}$.*

Clearly, q-DBDHI is a stronger assumption than q-DHI. To provide more confidence in its validity, we analyze this assumption in the generic group model in Section 6.

4 Our Constructions

In Section 4.1, we show that a signature scheme due to Boneh and Boyen [4] is in fact a VUF for small inputs. We could then use a Goldreich-Levin hardcore bit to convert the resulting VUF into a VRF. However, the generic transformation is rather inefficient, so we choose to forego it. Instead, in Section 4.2, we construct our VRF directly for inputs of small size. We then show how to extend the VRF input size in Section 4.3. Finally, we evaluate our construction's efficiency in Section 4.4.

Fix input length $a(k)$, output length $b(k)$, and security $s(k)$. For notational convenience, we will usually omit the security parameter k, writing, for example, a or s, instead of $a(k)$ or $s(k)$. Let $\mathbb{G}$ $(|\mathbb{G}| = p)$ be a bilinear group, whose order p is a k-bit prime. Let g be a generator of $\mathbb{G}$. Throughout, we shall assume that messages can be encoded as elements of $\mathbb{Z}_p^*$.

4.1 A Verifiable Unpredictable Function

In order to build the intuition for our next proof, we first describe how to construct a simple VUF (Gen, Sign, Ver), which is secure for small (superlogarithmic) inputs.

Algorithm Gen(1^k): Chooses a secret $s \in_r \mathbb{Z}_p^*$ and sets the secret key to $SK = s$ and a public key to $PK = g^s$.

Algorithm Sign$_{SK}(x)$: Outputs the signature Sign$_{SK}(x) = g^{1/(x+SK)}$. Note that the proof is embedded in the output value so we do not need to include it explicitly.

Algorithm Ver$_{PK}(x, y)$: Outputs 1 if $e(g^x \cdot PK, y) = e(g, g)$; otherwise, outputs 0. Indeed, if the VRF value y was correctly computed, we have:

$$e(g^x \cdot PK, y) = e(g^x g^s, g^{1/(x+s)}) = e(g, g).$$

Boneh and Boyen [4] proved this scheme to be existentially unforgeable against **non-adaptive adversaries** for inputs of arbitrary size. In our proof, we restrict inputs to have slightly superlogarithmic size in k (just like [15] do); that is, we set $a(k) = \log s(k) = \Omega(\log k)$. This enables us to enumerate all possible messages in $s(k)$ time and to respond to adversary's queries **adaptively**. Further, the proof of [4] is based on a q-strong Diffie-Hellman assumption (q-SDH), which is implied by a weaker q-DHI assumption used in our proof. Correspondingly, our proof is more involved but necessarily less tight than the proof of [4].

Theorem 1. *Suppose the* $(s(k), 2^{a(k)}, \epsilon(k))$*-DHI assumption holds in a bilinear group* $\mathbb{G}$ $(|\mathbb{G}| = p)$*. Let the input size be* $a(k)$ *and output size be* $b(k) = \log_2 p$*. Then* $(\textsc{Gen}, \textsc{Sign}, \textsc{Ver})$ *is a* $(s'(k), \epsilon'(k))$ *verifiable unpredictable function, where* $s'(k) = s(k)/(2^{a(k)} \cdot poly(k))$ *and* $\epsilon'(k) = \epsilon(k) \cdot 2^{a(k)}$*.*

Proof. It is easy to see that uniqueness and provability properties of Definition 2 are satisfied. We thus concentrate on residual unpredictability.

We shall use a shortcut and write $q = 2^{a(k)}$. Suppose there exists an adversary $\mathcal{A}$, running in time $s'(k)$, which guesses the value of the function at an unseen point with non-negligible probability $\epsilon'(k)$. We shall construct an algorithm $\mathcal{B}$ that by interacting with $\mathcal{A}$ breaks the q-DHI assumption with non-negligible probability.

Input to the reduction: Algorithm $\mathcal{B}$ is given a tuple $\left(g, g^{\alpha}, \ldots, g^{(\alpha^q)}\right) \in (\mathbb{G}^*)^{q+1}$, for some unknown $\alpha \in \mathbb{Z}_p^*$. Its goal is to compute $g^{1/\alpha}$.

Key generation: We guess that $\mathcal{A}$ will output a forgery on message $x_0 \in_r \{0,1\}^{a(k)}$. We are right with probability $1/2^{a(k)}$; error probability can be decreased by repeating the algorithm sufficiently many times. Let $\beta = \alpha - x_0$.[5] We don't know what β is because α is secret. However, we can use the Binomial Theorem to compute $\left(g^{\beta}, \ldots, g^{(\beta^q)}\right)$ from $\left(g^{\alpha}, \ldots, g^{(\alpha^q)}\right)$. Because $a(k) = \log(s(k))$, we can enumerate all possible inputs in $s(k)$ time. Let $f(z)$ be the polynomial

$$f(z) = \prod_{w \in \{0,1\}^a, w \neq x_0} (z + w) = \sum_{j=0}^{q-1} c_j z^j \text{ (for some coefficients } c_0, \ldots, c_{q-1}).$$

We can compute

$$h = g^{f(\beta)} = \prod_{j=0}^{q-1} \left(g^{(\beta^j)}\right)^{c_j} \text{ and } h^{\beta} = \prod_{j=1}^{q} \left(g^{(\beta_j)}\right)^{c_{j-1}}.$$

Finally, we set h to be the generator and give $PK = h^{\beta}$ to $\mathcal{A}$. The secret key is $SK = \beta$, which we don't know ourselves.

[5] For the sake of readability, we slightly abuse the notation. We should really have written $\beta = \alpha - \psi(x_0)$, where $\psi : \{0,1\}^{a(k)} \mapsto \mathbb{Z}_p^*$.

Responding to oracle queries: Without loss of generality, we assume that $\mathcal{A}$ never repeats a query. Consider the ith query ($1 \leq i < q$) on message x_i. If $x_i = x_0$, then we fail. Otherwise, we must compute $\text{Sign}_{SK}(x_i) = h^{1/(x_i+\beta)}$. Let $f_i(z)$ be the polynomial

$$f_i(z) = f(z)/(z + x_i) = \sum_{j=0}^{q-2} d_j z^j \text{ (for some coefficients } d_0, \ldots, d_{q-2}).$$

We can compute

$$g^{f_i(\beta)} = \prod_{j=0}^{q-2} \left(g^{(\beta^j)}\right)^{d_j} = h^{1/(x_i+\beta)}$$

and return it as the signature.

Outputting the forgery: Eventually, $\mathcal{A}$ outputs a forgery (x^*, σ^*). If $x^* \neq x_0$, then our simulation failed. Because the signature is unique, we must have $\sigma^* = h^{1/(x_0+\beta)} = g^{f(\beta)/(x_0+\beta)}$. Compute

$$f(z)/(z + x_0) = \sum_{j=0}^{q-2} \gamma_j z^j + \frac{\gamma_{-1}}{z + x_0},$$

where $\gamma_{-1} \neq 0$. Hence,

$$\left(\sigma^* \cdot \prod_{j=0}^{q-2} \left(g^{(\beta^i)}\right)^{-\gamma_i}\right)^{1/\gamma_{-1}} = g^{1/(x_0+\beta)} = g^{1/\alpha}.$$

Let $\epsilon'(k) = \epsilon(k) \cdot 2^{a(k)}$ and $s'(k) = s(k)/(2^{a(k)} \cdot \text{poly}(k))$. To finish the proof, note that algorithm $\mathcal{B}$ succeeds with probability $\epsilon'(k)/2^{a(k)} = \epsilon(k)$. Its running time is dominated by answering oracle queries, and each query takes $(2^{a(k)} - 2) \cdot \text{poly}(k)$ time to answer. Therefore, $\mathcal{B}$ will run in roughly $s'(k) \cdot 2^{a(k)}\text{poly}(k) = s(k)$ time.

□

Remark 1. The security reduction of Theorem 1 is not tight. It allows to construct VUFs with input roughly $a(k) = \Omega(\log s(k))$. In theory, this means that the input size we can achieve might be only slightly superlogarithmic in k (similar to [15]). First, it might be reasonable to assume subexponential hardness of the q-DHI assumption which will immediately allow one to support input of size $k^{\Omega(1)}$. Also, by utilizing a collision-resistant hash function, we will anyway only need to construct VUFs with relatively small input size such as 160 bits. Indeed, in Section 4.4, we show that our construction seems to yield a practical and secure VUF for inputs of arbitrary length already when $k = 1,000$ bits.

4.2 A Verifiable Random Function

Our main contribution is a direct construction of a verifiable random function from a slightly stronger q-DBDHI assumption. The VRF $(\text{Gen}, \text{Prove}, \text{Ver})$ is as follows.

Algorithm $\text{Gen}(1^k)$: Chooses a secret $s \in_r \mathbb{Z}_p^*$ and sets the secret key to $SK = s$ and the public key to $PK = g^s$.

Algorithm $\text{Prove}_{SK}(x)$: We let $\text{Prove}_{SK}(x) = \big(F_{SK}(x), \pi_{SK}(x)\big)$ where $F_{SK}(x) = e(g,g)^{1/(x+SK)}$ is the VRF output and $\pi_{SK}(x) = g^{1/(x+SK)}$ is the proof of correctness.

Algorithm $\text{Ver}_{PK}(x, y, \pi)$: To verify whether y was computed correctly, check if $e(g^x \cdot PK, \pi) = e(g,g)$ and whether $y = e(g, \pi)$. If both checks succeed, output 1; otherwise, output 0.

We can prove this scheme to be secure (in the sense of Definition 1) for small inputs (superlogarithmic in k). We then show how to convert it into a VRF with unrestricted input size.

Theorem 2. *Suppose the* $(s(k), 2^{a(k)}, \epsilon(k))$*-decisional BDHI assumption holds in a bilinear group* $\mathbb{G}$ $(|\mathbb{G}| = p)$*. Let the input size be* $a(k)$ *and the output size be* $b(k) = \log_2 p$*. Then* $(\text{Gen}, \text{Prove}, \text{Ver})$*, as defined above, is a* $(s'(k), \epsilon'(k))$ *verifiable random function, where* $s'(k) = s(k)/(2^{a(k)} \cdot poly(k))$ *and* $\epsilon'(k) = \epsilon(k) \cdot 2^{a(k)}$*.*

Proof. It is trivial to show that uniqueness and provability properties of Definition 1 are satisfied. We thus concentrate on the pseudorandomness property.

We shall use $q = 2^{a(k)}$ as a shortcut. For sake of contradiction, suppose there exists an algorithm $\mathcal{A} = (A_1, A_2)$, which runs in time $s'(k)$, and can distinguish between $F_{SK}(x) = e(g,g)^{1/(x+s)}$ (for some x) and a random element in $\mathbb{G}_1$ with probability at least $1/2 + \epsilon'(k)$. We shall construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the q-DBDHI assumption in $\mathbb{G}$.

Input to the reduction: Algorithm $\mathcal{B}$ is given a tuple $(g, g^\alpha, \ldots, g^{(\alpha^q)}, \Gamma) \in (\mathbb{G}^*)^{q+1} \times \mathbb{G}_1$, where Γ is either $e(g,g)^{1/\alpha} \in \mathbb{G}_1$ or a random element in $\mathbb{G}_1$. Its goal is to output 1 if $\Gamma = e(g,g)^{1/\alpha}$ and 0 otherwise.

Key generation: We guess that $\mathcal{A}$ will choose to distinguish the VRF value on message $x_0 \in \{0,1\}^{a(k)}$. Let $\beta = \alpha - x_0$ (see footnote 5). We generate the public and private keys for algorithm $\mathcal{A}$ as in the proof of Theorem 1. Using the Binomial Theorem, we compute the tuple $\big(g^\beta, \ldots, g^{(\beta^q)}\big)$. We define

$$f(z) = \prod_{w \in \{0,1\}^a, w \neq x_0} (z + w) = \sum_{j=0}^{q-1} c_j z^j.$$

This enables us to compute the new base

$$h = g^{f(\beta)} = \prod_{j=0}^{q-1} \left(g^{(\beta^j)}\right)^{c_j}.$$

Finally, we give $PK = h^{\beta} = \prod_{j=1}^{q} \left(g^{(\beta^j)}\right)^{c_{j-1}}$ as the public key to $\mathcal{A}$. The secret key is $SK = \beta$, which we don't know.

Responding to oracle queries: Consider the ith query $(1 \le i < q)$ on message x_i. If $x_i = x_0$, we fail. Otherwise, we must respond with the corresponding proof $\pi_{SK}(x_i)$ and a VRF value $F_{SK}(x_i)$.

As in Theorem 1, we define

$$f_i(z) = f(z)/(z + x_i) = \sum_{j=0}^{q-2} d_j z^j \text{ (for some coefficients } d_0, \ldots, d_{q-2}).$$

We can thus compute

$$\pi_{SK}(x_i) = \prod_{j=0}^{q-2} \left(g^{(\beta^j)}\right)^{d_j} = h^{1/(\beta + x_i)}$$

and

$$F_{SK}(x_i) = e(h, \pi_{SK}(x_i)) = e(h,h)^{1/(\beta + x_i)},$$

and return them to algorithm $\mathcal{A}$.

Challenge: Eventually, $\mathcal{A}$ outputs a message x^* on which it wants to be challenged. If $x^* \neq x_0$, then we fail. Otherwise, $\mathcal{A}$ claims to be able to distinguish $e(h,h)^{1/(\beta + x_0)} = e(h,h)^{1/\alpha}$ from a random element in $\mathbb{G}_1$. Recall that

$$f(z) = \sum_{i=0}^{q-1} c_i z^i.$$

Because $f(z)$ is not divisible by $(z + x_0)$, we have:

$$\begin{aligned} f'(z) &= f(z)/(z + x_0) - \frac{\gamma}{z + x_0} \\ &= \sum_{j=0}^{q-2} \gamma_j z^j \text{ (for some } \gamma \neq 0 \text{ and coefficients } \gamma_0, \ldots, \gamma_{q-2}). \end{aligned}$$

Let Γ_0 be

$$\begin{aligned} \Gamma_0 &= \left(\prod_{i=0}^{q-1} \prod_{j=0}^{q-2} e\left(g^{(\beta^i)}, g^{(\beta^j)}\right)^{c_i \gamma_j} \right) \cdot \left(\prod_{m=0}^{q-2} e\left(g, g^{(\beta^t)}\right)^{\gamma\,\gamma_m} \right) \\ &= e\left(g^{f(\beta)}, g^{f'(\beta)}\right) \cdot e\left(g^{\gamma}, g^{f'(\beta)}\right) \qquad (1) \\ &= e(g,g)^{(f(\beta)^2 - \gamma^2)/\alpha}. \end{aligned}$$

Set $\Gamma^* = \Gamma^{(\gamma^2)} \cdot \Gamma_0$. Notice that if $\Gamma = e(g,g)^{1/\alpha}$, then $\Gamma^* = e(g^{f(\beta)}, g^{f(\beta)/\alpha}) = e(h,h)^{1/\alpha}$. Meanwhile, if Γ is uniformly distributed, then so is Γ^*. We give Γ^* to algorithm $\mathcal{A}$.

Note: It may seem as though computing Γ_0 is very expensive. However, from Equation (1), we see that the computation only takes two bilinear map evaluations.

Guess: Algorithm $\mathcal{A}$ makes some more queries to which we respond as before. Finally, $\mathcal{A}$ outputs a guess $b \in \{0, 1\}$. We return b as our guess as well.

The running time of the reduction is dominated by simulating oracle queries. Per every query, we must perform one bilinear map evaluation (this takes poly(k) time) and $(2^a - 2)$ multiplications and exponentiations (this takes $2^a \cdot \text{poly}(k)$ time). Because $\mathcal{A}$ can make at most $s'(k)$ queries, the running time of $\mathcal{B}$ is altogether $s'(k)(2^{a(k)} \cdot \text{poly}(k))$. The advantage of $\mathcal{B}$ in this experiment is $\epsilon'(k)/2^{a(k)}$. Setting $s'(k) = s(k)/(2^{a(k)} \cdot \text{poly}(k))$ and $\epsilon'(k) = \epsilon(k) \cdot 2^{a(k)}$ completes the proof. □

4.3 Extending the Input Size

We constructed a VRF (Gen, Prove, Ver), which is provably secure for inputs of small size $a(k) = \Omega(\log(k))$. We now explain how to handle inputs of arbitrary size.

Hashing the Input. Notice that if we have a VRF $\text{Prove}_{SK}(\cdot) : \{0, 1\}^{a(k)} \mapsto \{0, 1\}^{b(k)}$ and a collision-resistant hash function $H(\cdot) : \{0, 1\}^* \mapsto \{0, 1\}^{a(k)}$, then their composition $\text{Prove}_{SK}(H(\cdot)) : \{0, 1\}^* \mapsto \{0, 1\}^{b(k)}$ is trivially secure. Although our security reduction is relatively loose, we can make the size of a bilinear group large enough (we give exact numbers in Section 4.4) to have inputs of length roughly $a(k) = 160$ bits, the length of SHA-1 digests. Restriction to small inputs is therefore not limiting because we can always hash longer inputs.

Tree Construction. Although, we recommend using the previous construction (by making the group large enough), in theory, we could always use the (inefficient) generic tree construction to extend the input length. Then, we do not have to assume the existence of a collision-resistant hash function; having a universal hash function suffices.

We shall use the following proposition:

Proposition 1 ([15]). *If there is a VRF with input length $a(k)$, output length 1, and security $s(k)$, then there is a VRF with unrestricted input length, output length 1 and security at least* $\min(s(k)^{1/5}, 2^{a(k)/5})$.

The construction first converts a VRF with output length 1 into a VRF with output length $(a-1)$. This transformation loses a factor of a in security. Because our VRF has output length much larger than 1, we can omit this step. Instead, we apply a universal hash function to VRF's output and let the VRF's value be the first $(a - 1)$ bits of hash function's output (it is easily seen that these bits will be pseudo-random as well).

The rest of the transformation proceeds as usual. We construct a binary trie whose nodes are labeled with strings of length $(a - 1)$. The root is labeled with 0^{a-1} and the children of node y are labeled with VRF values on inputs $(y \circ 0)$

and $(y \circ 1)$. Computing the VRF value on input $x \in \{0,1\}^*$ amounts to tracing a path through the trie to the leaf corresponding to x. The VRF value is the label of the leaf, and the proof of correctness is a tuple of VRF proofs—one proof per each node on the path traced by x.[6]

We also note that both of the aforementioned techniques can be used to convert the VUF in Section 4.1 into a VUF with unrestricted input length.

4.4 Efficiency

We now compare the efficiency of our construction with that of prior VRF constructions. We fix inputs to be $a(k) = 160$ bits, the length of SHA-1 digests, and let $q = 2^{a(k)}$.

Our VRF. According to Theorem 2, if $(s(k), q, \epsilon(k))$-DBDHI holds on $\mathbb{G}$, then our VRF is secure against adversaries running in time $s'(k) = s(k)/(2^{a(k)} \cdot \mathrm{poly}(k))$ that have advantage $\epsilon'(k) = \epsilon(k) \cdot 2^{a(k)}$. To be generous, we instantiate $\epsilon'(k) = 2^{-80}$, $s'(k) = 2^{80}$, and $\mathrm{poly}(k) = 2^{30}$. Then, we have: $\epsilon(k) = 2^{-240}$ and $s(k) = 2^{270}$. Suppose no better algorithm exists for breaking the q-DBDHI assumption than a generic group algorithm. Then, by Theorem 3 (which we prove in Section 6), for these security parameters a bilinear group must have size:

$$\begin{aligned} p &\geq \frac{2(s(k)+q+3)^2 q}{\epsilon(k)} \\ &= \frac{2\left(2^{270}+2^{160}+3\right)^2 2^{160}}{2^{-240}} \\ &\approx 2^{940}. \end{aligned}$$

Therefore, making the group size be a 1,000 bit prime seems sufficient to guarantee security of the VRF that takes 160 bit inputs. Proofs and keys consist of a single group element and will roughly be 125 bytes each. We can generate such groups using the standard parameter generator of [5].

VRF by Micali-Rabin-Vadhan [15]. This VRF operates over a multiplicative group $\mathbb{Z}_n^*$, where $n = pq$ is a k-bit RSA modulus. The fastest general-purpose factoring algorithm today is the number field sieve [7]; it takes approximately $O\left(e^{1.9223(k^{1/3}(\log k)^{2/3})}\right)$ time to factor a k bit number. The RSA based VUF (not even a VRF) constructed in [15] has security $s'(k) = s(k)/(2^{a(k)} \cdot \mathrm{poly}(k))$ where $s(k)$ is hardness of RSA. Letting $s'(k) = 2^{80}$ and $\mathrm{poly}(k) = 2^{30}$ as before, we obtain an RSA security lower bound $s(k) = 2^{80} \cdot (2^{160} \cdot 2^{30}) = 2^{270}$. Because

[6] The inputs have to be prefix-free for this tree construction to work. This can be accomplished using techniques of [15].

RSA is only secure as long as we cannot factor n, to get 270 bits of security, we need n to be a k-bit number, where

$$1.9223k^{1/3}(\log k)^{2/3} = 270.$$

Hence, n must be at least $14,383$ bits long if we want to use this VUF on 160 bit inputs. After following the tree construction, proofs for 160 bit inputs will have size 280 kilobytes.

VRF by Dodis [8] and VUF by Lysyanskaya [14]. These constructions work on elliptic curve groups, whose size is usually a 160 bit prime. At the bare minimum, 160 bit messages yield keys and proofs of size $160 \cdot 160 = 25,600$ bits, which is about 3.2 kilobytes. In fact, they will probably have larger size due to use of error-correcting codes and other encoding expansions.

To summarize, none of the prior VRF constructions come close to the 1,000 bit proofs and keys of our construction. If our VRF is used with the generic tree construction, its keys and proofs consist of $|x|$ group elements (one group element per input bit) when the input is $x \in \{0,1\}^*$. This is less than the $|x|^2$ group elements ($|x|$ group elements per input bit) needed by the VRF of [14].

5 Distributed VRF

We point out that our VUF/VRF constructions can be easily made distributed (or even proactive). Indeed, both of the constructions simply amount to a secure computation of the function $\pi_{SK}(x) = g^{1/(x+SK)}$ when the servers have shares of the secret SK. Because it is well known how to do multiparty addition, inversion, and exponentiation [1, 2], this extension follows immediately. We notice however that unlike the construction of Dodis [8], our distributed VUF/VRF is interactive.

6 Generic Security of the q-DBDHI Assumption

In this section, we examine the q-DBDHI assumption in the generic group model of Shoup [21]. We proceed to derive a lower bound on the computational complexity of a generic adversary who breaks this assumption.

In the generic group model, elements of $\mathbb{G}$ and $\mathbb{G}_1$ are encoded as unique random strings. We define an injective function $\theta : \mathbb{Z}_p \mapsto \{0,1\}^*$, which maps $a \in \mathbb{Z}_p$ to the string representation $\theta(g^a)$ of $g^a \in \mathbb{G}$. Similarly, we define a function $\theta_1 : \mathbb{Z}_p \mapsto \{0,1\}^*$ for $\mathbb{G}_1$. The encodings are such that non-group operations are meaningless. There exist three oracles which compute the group action in $\mathbb{G}$, the group action in $\mathbb{G}_1$, and the bilinear pairing $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$ from elements' encodings.

Theorem 3. *Let $\mathcal{A}$ be an algorithm that solves the q-DBDHI problem. Assume both $x \in \mathbb{Z}_p^*$ and the encoding functions θ, θ_1 are chosen at random. If $\mathcal{A}$ makes at most q_G queries to oracles computing the group action in $\mathbb{G}, \mathbb{G}_1$ and the bilinear mapping $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_1$, then*

$$\left| \Pr \left[\begin{array}{c} \mathcal{A}\left(p, \theta(1), \theta(x), \ldots, \theta(x^q), \right. \\ \left. \theta_1(\Gamma_0), \theta_1(\Gamma_1)\right) = b \end{array} \middle| \begin{array}{c} b \xleftarrow{r} \{0,1\}; \\ \Gamma_b \leftarrow 1/x;\ \Gamma_{1-b} \xleftarrow{r} \mathbb{Z}_p^* \end{array} \right] - \frac{1}{2} \right| \leq \frac{2(q_G + q + 3)^2 q}{p}.$$

Proof. Instead of letting $\mathcal{A}$ interact with the actual oracles, we play the following game.

We maintain two lists: $L = \{ (F_i, s_i) : i = 0, \ldots, t-1\}$ and $L' = \{ (F'_i, s'_i) : i = 0, \ldots, t'-1\}$. Here $s_i, s'_i \in \{0,1\}^*$ are encodings and $F_i, F'_i \in \mathbb{Z}_p[X, \Gamma_0, \Gamma_1]$ are multivariate polynomials in X, Γ_0, and Γ_1. The total length of lists at step $\tau \leq q_G$ in the game must be

$$t + t' = \tau + q + 3. \tag{2}$$

In the beginning of the game, we initialize the lists to $F_0 = 1, F_1 = X, \ldots, F_q = X^q$ and $F'_0 = \Gamma_0, F'_1 = \Gamma_1$. The corresponding encodings are set to arbitrary distinct strings in $\{0,1\}^*$. The lists have length $t = q + 1$ and $t' = 2$.

We start the game by providing $\mathcal{A}$ with encodings $(s_0, \ldots, s_q, s'_0)$. Algorithm $\mathcal{A}$ begins to issue oracle queries. We respond to them in the standard fashion:

Group action: Given a multiply/divide bit and two operands s_i and s_j ($0 \leq i, j < t$), we compute $F_t = F_i \pm F_j$ accordingly. If $F_t = F_l$ for some $l < t$, we set $s_t = s_l$. Otherwise, we set s_t to a random string in $\{0,1\}^* \backslash \{s_0, \ldots, s_{t-1}\}$, and increment t by 1. Group action in G_1 is computed similarly, except we operate on list L'.

Bilinear pairing: Given two operands s_i and s_j ($0 \leq i, j < t$), we compute the product $F_{t'} = F_i F_j$. If $F_{t'} = F_l$ for some $l < t'$, we set $s_{t'} = s_l$. Otherwise we set it to a random string in $\{0,1\}^* \backslash \{s_0, \ldots, s_{t'-1}\}$. We then increment t' by 1.

After making at most q_G queries, $\mathcal{A}$ halts with a guess $\hat{b} \in \{0,1\}$. We now choose $x, y \xleftarrow{r} \mathbb{Z}_p^*$ and consider $\Gamma_b \leftarrow 1/x, \Gamma_{1-b} = y$ for both choices of b. Our simulation is perfect and reveals nothing to $\mathcal{A}$ about b unless the values that we chose for indeterminates give rise to some non-trivial equality relation. Specifically, algorithm $\mathcal{A}$ wins the game if for any $F_i \neq F_j$ or any $F'_i \neq F'_j$, either of these hold:

1. $F_i(x, 1/x, y) - F_j(x, 1/x, y) = 0$
2. $F_i(x, y, 1/x) - F_j(x, y, 1/x) = 0$
3. $F'_i(x, 1/x, y) - F'_j(x, 1/x, y) = 0$
4. $F'_i(x, y, 1/x) - F'_j(x, y, 1/x) = 0$

Notice that $\mathcal{A}$ can never engineer an encoding of an element whose corresponding polynomial would have a $1/X$ term unless he is explicitly given it.

Therefore, we can only get a non-trivial equality relation as a result of numerical cancellation.

For all i, $\deg(F_i) \leq q$ and $\deg(F'_i) \leq 2q$. We can use the Schwartz-Zippel Theorem [20] to bound the probability of a cancellation. It tells us that for all i, j, $\Pr[F_i - F_j = 0] \leq q/p$ and $\Pr[F'_i - F'_j = 0] \leq 2q/p$. Thus $\mathcal{A}$'s advantage is

$$\begin{aligned}\epsilon &\leq 2 \cdot \left(\binom{t}{2}\frac{q}{p} + \binom{t'}{2}\frac{2q}{p}\right) \\ &< 2(q_G + q + 3)^2 \frac{q}{p} \quad \text{(plugging into (2))} \\ &= O\left(\frac{q_G^2 q + q^3}{p}\right).\end{aligned}$$

$\square$

It turns out that in a generic group model algorithm $\mathcal{A}$ that solves the q-DBDHI problem has advantage, which is roughly twice as much as an advantage of an algorithm solving the q-SDH problem (see [4], Section 5). The asymptotic complexities are the same.

The following corollary is immediate.

Corollary 1. *Any adversary that breaks the q-DBDHI assumption with probability $\frac{1}{2} + \epsilon$ $(0 < \epsilon < 1/2)$ in generic groups of order p such that $q < o(\sqrt[3]{p})$ requires $\Omega(\sqrt{\epsilon p/q})$ generic group operations.*

7 Conclusion

We have presented a simple and efficient construction of a verifiable random function. Our VRF's proofs and keys have constant size regardless of the size of the input. Our proofs of security are based on a decisional bilinear Diffie-Hellman inversion assumption, which seems reasonable given current state of knowledge. We also demonstrated that our scheme can be instantiated with elliptic groups of very reasonable size which makes our constructions quite practical.

8 Acknowledgments

The authors would like to thank James Aspnes, Dan Boneh, Salil Vadhan, and the anonymous referees for their helpful comments.

References

1. Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds. In *Proceedings of the ACM Symposium on Principles of Distributed Computation*, pages 201–209, 1989.

2. Michael Ben-or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 1–10, 1988.
3. Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *Advances in Cryptology—EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Berlin: Springer-Verlag, 2004.
4. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology—EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Berlin: Springer-Verlag, 2004.
5. Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. *Lecture Notes in Computer Science*, 2139:213–229, 2001.
6. Dan Boneh and Alice Silverberg. Application of multilinear forms to cryptography. Cryptology ePrint Archive, Report 2002/080, 2002. `http://eprint.iacr.org/2002/080/`.
7. Johannes A. Buchmann, J. Loho, and J. Zayer. An implementation of the general number field sieve. *Lecture Notes in Computer Science*, 773:159–166, 1994.
8. Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In *Proceedings of 6th International Workshop on Theory and Practice in Public Key Cryptography*, pages 1–17, 2003.
9. Steven D. Galbraith. Supersingular curves in cryptography. *Lecture Notes in Computer Science*, 2248:495–513, 2001.
10. Oded Goldreich and Leonid Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21th Annual ACM Symposium on the Theory of Computing*, pages 25–32, 1989.
11. S. Goldwasser and M. Bellare. Lecture notes on cryptography. Summer Course “Cryptography and Computer Security” at MIT, 1996–1999, 1999.
12. Stanislaw Jarecki and Vitaly Shmatikov. Handcuffing big brother : an abuse-resilient transaction escrow scheme. In *Advances in Cryptology - Proceedings of EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 590–608. Springer-Verlag, 2004.
13. Antoine Joux and Kim Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org/2001/003/`.
14. Anna Lysyanskaya. Unique signatures and verifiable random functions from DH-DDH separation. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 597–612, 2002.
15. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 120–130, 1999.
16. Silvio Micali and Leonid Reyzin. Soundness in the public-key model. *Lecture Notes in Computer Science*, 2139:542–565, 2001.
17. Silvio Micali and Ronald L. Rivest. Micropayments revisited. In *CT-RSA*, pages 149–163, 2002.
18. Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. *IEICE Trans. Fundamentals*, pages 481–484, 2002.
19. Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 458–467, 1997.
20. Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27:701–717, 1980.

21. Victor Shoup. Lower bounds for discrete logarithms and related problems. *Lecture Notes in Computer Science*, 1233:256–266, 1997.
22. Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 31–37, 1996.