Verifiable Random Functions

Silvio Micali[*] Michael Rabin[†] Salil Vadhan[‡]

Abstract

We efficiently combine unpredictability and verifiability by extending the Goldreich–Goldwasser–Micali construction of pseudorandom functions f_s from a secret seed s, so that knowledge of s not only enables one to evaluate f_s at any point x, but also to provide an $\mathbf{NP}$-proof that the value $f_s(x)$ is indeed correct without compromising the unpredictability of f_s at any other point for which no such a proof was provided.

1 Introduction

PSEUDORANDOM ORACLES. Goldreich, Goldwasser, and Micali [GGM86] show how to simulate a random oracle from a-bit strings to b-bit strings by means of a construction using a *seed*, that is, a secret and short random string. They show that, if pseudorandom generators exist [BM84, Yao82], then there exists a polynomial-time algorithm $F(\cdot,\cdot)$ such that, letting s denote the seed, the function $f_s \stackrel{\text{def}}{=} F(s,\cdot) : \{0,1\}^a \to \{0,1\}^b$ *passes all efficient statistical tests for oracles*. That is, to an observer with sufficiently limited computational resources, accessing a random oracle from $\{0,1\}^a$ to $\{0,1\}^b$ is provably indistinguishable from accessing (as an oracle) f_s, even if algorithm F is publicly known (provided that s is still kept secret).

THE PROBLEM OF CONSTRUCTING VERIFIABLE PSEUDORANDOM FUNCTIONS. By its very definition, a pseudorandom oracle à la [GGM86] is not *verifiable*: without knowledge of the seed (or any other additional information), upon receiving the value z of a pseudorandom oracle f_s at point x, one cannot distinguish it from an independently selected random string of the proper length. The possibility thus exists that, if it so suits him, the party knowing the seed s may declare that the value of his pseudorandom oracle at some point x is other than $f_s(x)$ without fear of being detected. It is for this reason that we refer to these objects as "pseudorandom oracles" rather than using the standard terminology "pseudorandom functions" — the values $f_s(x)$ come "out of the blue," as if from an oracle, and the receiver must simply trust that they are computed correctly from the seed s.

Therefore, though quite large, the applicability of pseudorandom oracles is limited: for instance, to settings in which (1) the "seed owner", and thus the one evaluating the pseudorandom oracle, is totally trusted; or (2) it is to the seed-owner's advantage to evaluate his pseudorandom oracle correctly; or (3) there is absolutely nothing for the seed-owner to gain from being dishonest.

One efficient way of enabling anyone to verify that $f_s(x)$ really is the value of pseudorandom oracle f_s at point x clearly consists of publicizing the seed s. However, this will also destroy the unpredictability of f_s: anyone could easily compute the value of f_s at any point.

We instead wish to provide a new type of pseudorandom oracle. Informally, we want one in which the owner of the seed s can, as usual, evaluate f_s at any point, but also prove (with an $\mathbf{NP}$ proof[1]) that the so obtained values are indeed correct without compromising the unpredictability of the value of f_s at any point x for which no proof of correctness for $f_s(x)$ is given. That is, given an input x, the seed-owner should be able to produce in polynomial time the value $v = f_s(x)$ together with a string proof_x efficiently proving that v is correct. The scheme should have the property that a *unique* value v is provable as the value of $f_s(x)$. We call such a mathematical object a *verifiable (pseudo-)random function*, VRF for brevity.

A WEAKER SOLUTION: PSEUDORANDOM ORACLES + ZERO-KNOWLEDGE PROOFS. If interaction were allowed, VRFs could be constructed from GGM pseudorandom oracles via zero-knowledge proofs [GMR89] and a commitment scheme. Indeed, as suggested in a signature scheme of Bellare and Goldwasser [BG89], the owner of the seed

[*] Laboratory for Computer Science, MIT, Cambridge, MA 02139.

[†] Department of Applied Science, Harvard University, Cambridge, MA 02138. Work supported in part by NSF Contract CCR-9877138.

[‡]MIT Laboratory for Computer Science. 545 Technology Square. Cambridge, MA 02139. E-mail: `salil@theory.lcs.mit.edu`. URL: `http://theory.lcs.mit.edu/~salil`. Supported by a DOD/NDSEG fellowship and partially by DARPA grant DABT63-96-C-0018.

[1]Strictly speaking, we actually allow "$\mathbf{MA}$ proofs", since their verification may be probabilistic.

s to a pseudorandom oracle f_s can publish a commitment c to s. Whenever he wishes to prove that v is the value of his oracle at a point x to a verifier V, he proves in zero knowledge to V that $v = f_s(x)$ and that c is a commitment to s. Such a statement is provable in zero knowledge because all NP statements are provable in zero knowledge [GMW91]. The trouble with such an approach is that it requires interaction. A very efficient incarnation of this idea is given by Naor and Reingold [NR97], but it still suffers from the need for interaction.

Such interaction could be removed by using noninteractive zero-knowledge proofs (NIZK) [BFM88, BDMP91], as done by Bellare and Goldwasser [BG89]. This approach however suffers from another drawback: noninteractive zero-knowledge proofs presuppose that the prover and verifier share a bit-string that is *guaranteed* to be random. So the question is who is to select this shared random string R. Each of the possibilities has a deficiency that we wish to avoid in defining VRFs:

1. The seed owner selects R: If the seed owner selects the shared random string improperly, the soundness of the NIZK proof system is no longer guaranteed, so there may be many values v that are "provable" as $f_s(x)$.
2. The verifier selects R: If the verifier selects the shared random string improperly, the zero-knowledge property of the NIZK proof system is no longer guaranteed. Thus, by proving $f_s(x) = v$ with respect to such an improperly chosen R, the prover may leak knowledge about the seed s and f_s will "lose" its pseudorandomness.
3. The seed owner and verifier jointly select R by a "coin-flipping" protocol.: This requires interaction, which we wish to avoid.
4. A trusted third party selects R: We do not want to assume the existence of such a trusted third party.

OUR SOLUTION. We propose a notion of VRF's which needs *neither interaction nor sharing a guaranteed random string*. Rather, we only require that the owner of the function f publish a public key PK, which can be viewed as a commitment to the function f. (Indeed, something must bind the owner to the function in order for "proving the statement $f(x) = v$" to make sense.) The crucial way in which our notion differs from what the NIZK-based approach discussed above achieves is:

For any public key PK, *even an improperly chosen one*, a unique value v is provable as the value of $f(x)$.

Thus, we may safely have the owner of the function unilaterally select and publish the public key. The most obvious scenario in which this applies is when the public key can be published once and for all, in a location where it cannot be changed. But VRFs may also be useful in settings where the public key is provided "on the fly" to prove that various function values (given previously or at the same time) are indeed consistent with one single VRF. In case the VRF outputs strings longer than the public key, it may even be useful to provide the public key on the fly to prove that a *single* value is consistent with some VRF, as this would limit the owner to relatively few choices.

In addition to introducing this notion, we provide an explicit VRF construction, based on a variant of the RSA assumption. Informally stated, we prove:

> **Main Theorem:** Assume that the RSA function with large prime exponents cannot be inverted in polynomial time. Then, there exists a VRF from $\{0,1\}^*$ into $\{0,1\}$.

OVERVIEW OF THE CONSTRUCTION. We motivate our construction by first discussing the relationship between VRFs and secure signature schemes. In a signature scheme that is existentially unforgeable against a chosen message attack [GMR88], the signature of a message x, denoted $SIG(x)$, is a value that is unpredictable (even given signatures of chosen other messages), but verifiable (given the proper public key). However, such schemes do not directly give rise to VRFs by setting $f(x)$ to be $SIG(x)$, for two reasons:

1. There may be many valid signatures for a given string x (violating the unique provability requirement).
2. $SIG(x)$ is only unpredictable, not necessarily pseudorandom.

We begin by discussing the first deficiency, as it is the more serious one. Even though the definition does not guarantee the uniqueness of signatures, one might hope that existing signature schemes happen to have this property. However, most known secure signature schemes are either *probabilistic* or *history dependent.* Either property violates the the unique provability requirement: if we define $f(x)$ to be $SIG(x)$, there may be a multiplicity of signatures of x and thus a multiplicity of $f(x)$ values, all duly provable. One can transform a probabilistic signature scheme, such as the scheme in [GMR88], into a deterministic one if the signer uses a GGM pseudorandom oracle to replace the randomness used. Even so, this does not yield a VRF because one cannot be certain that the signer used the proper GGM oracle when producing $SIG(x)$, and hence unique provability is NOT guaranteed.

More generally, it is not enough that the specified signing algorithm produces a unique signature for every message. Rather, it should be the case that the verification algorithm accepts a unique (or at most one) signature for every message (given any fixed, but even improperly chosen, public key). A signature scheme that satisfies this latter property can be thought of as a *verifiable unpredictable function*; that is, a verifiable unpredictable function is defined analogously to a verifiable pseudorandom function except the pseudorandomness requirement is replaced with unpredictability.

So the two questions that remain are (1) do verifiable unpredictable functions imply verifiable pseudorandom functions?, and (2) can we construct verifiable unpredictable functions? The natural approach to answering the first question is to use the hardcore bit construction of Goldreich and Levin [GL89], which is a general tool for converting unpredictability to pseudorandomness. That is, we replace the verifiable unpredictable function $f(x)$ with $f'(x) = \langle f(x), r\rangle$, where r is a randomly chosen binary string of the same length as $f(x)$ and $\langle\cdot,\cdot\rangle$ denotes mod-2 inner product. Note that for this construction to preserve verifiability, r should be placed in the public key (the proof that $f'(x) = b$ is a string v such that $\langle r, v\rangle = b$ together with a proof that $f(x) = v$). Unfortunately, it has been shown by Naor and Reingold [NR98] that using a public Goldreich–Levin vector r does not work in general for converting unpredictable functions into pseudorandom functions.[2] The way we get around this obstacle is by noting that, a public r *can* be used if we restrict to functions whose input length is logarithmically related to the security. Then, we remove this restriction on the input length via a tree-based construction which converts any VRF with a fixed input length into one whose domain is $\{0,1\}^*$.

Thus, we are left with the task of finding a verifiable unpredictable function. Our construction builds upon an RSA-based unpredictable number generator of Shamir [Sha83], adapted to secure signature schemes by [GMY83, DN94, CD96, GHR99, CS99]. Shamir shows that seeing $r^{1/e_i} \bmod m$ for different exponents $e_{i_1}, \ldots, e_{i_k}$ does does not help one predict $r^{1/e_{i_{k+1}}} \bmod m$ as long as all of these $k+1$ exponents are relatively prime to each other and to $\phi(m)$. This suggests constructing a verifiable unpredictable function by placing m and r in the public key, and defining $f(i)$ to be $v = r^{1/e_i} \bmod m$. This can be verified simply by checking that $v^{e_i} = r \bmod m$; the solution $v \in \mathbb{Z}_m^*$ to this equation will be unique as long as e_i is guaranteed to be relatively prime to $\phi(m)$. Thus, we obtain all the desired properties as long as we can efficiently index into a set of such e_i's which are guaranteed to be all relatively prime to each other and to $\phi(m)$. We accomplish this by restricting to exponents which are distinct primes larger than m, and we index into such a set by using the prime sequence generator of Cachin, Micali, and Stadler [CMS99]. This turns out to yield a verifiable unpredictable function whose input length is logarithmically related to the security. This restriction on input length is of no concern, because we increase the input length after converting it into a VRF using the tree-based construction mentioned above.

[2] Interestingly, they show that using a *private* r does in fact work. This is the only known application of the Goldreich–Levin hardcore bit where keeping the vector private is necessary.

2 Preliminaries[3]

If $A(\cdot)$ is a probabilistic algorithm, then for any input x, the notation "$A(x)$" refers to the probability space that assigns to the string σ the probability that A, on input x, outputs σ. If S is a probability space, then "$x \stackrel{R}{\leftarrow} S$" denotes the algorithm which assigns to x an element randomly selected according to S, and "$x_1, \ldots, x_n \stackrel{R}{\leftarrow} S$" denotes the algorithm that respectively assigns to, $x_1, \ldots, x_n$, n elements randomly and independently selected according to S. If F is a finite set, then the notation "$x \stackrel{R}{\leftarrow} F$" denotes the algorithm that chooses x uniformly from F. If $p(\cdot,\cdot,\cdots)$ is a predicate, the notation $\mathrm{PROB}[x \stackrel{R}{\leftarrow} S; y_1, \ldots, y_n \stackrel{R}{\leftarrow} A(x); \cdots : p(x, y_1, \ldots, y_n, \cdots)]$ denotes the probability that $p(x, y_1, \ldots, y_n, \cdots)$ will be true after the ordered execution of the algorithms $x \stackrel{R}{\leftarrow} S$; $y_1, \ldots, y_n \stackrel{R}{\leftarrow} A(x); \cdots$.

3 The Notion of a VRF

3.1 An Informal Exposition

VRF GENERATION. To be a VRF, a function f must possess both

1. a compact, *implicit representation*, which does not enable one to evaluate f efficiently, and
2. a compact, *explicit representation*, which enables anyone to evaluate f efficiently.

The first representation can be viewed as f's *public key*, PK_f, and the second as its corresponding *secret key*, SK_f. Of course, SK_f will be hard to compute from PK_f. Accordingly, to formalize our notion of a VRF we make use of a probabilistic *generating* algorithm G outputting a public key with its matching secret key from a sequence of coin tosses.

VRF COMPUTATION AND VERIFICATION. Knowledge of SK_f enables one both to evaluate f and to prove the correctness of such evaluations. We actually envisage that $f(x)$ is always computed together with, $proof_x$, a string "proving its correctness", by running an efficient algorithm F on inputs x and SK_f. The function f proper is thus evaluated by running F, so as to obtain a function value and its proof, and then "stripping out" the proof. The correctness of $proof_x$ is instead verified by running an efficient algorithm V on inputs PK_f, x, $f(x)$, and $proof_x$. For convenience,

[3] Verbatim from [BDMP91] and [GMR88].

we denote the two components of $F(SK, x)$ by $F_1(SK, x)$ and $F_2(SK, x)$ (corresponding to the $f(x)$ and $proof_x$, respectively).

Because a proof of correctness for $f(x)$ is only checked against f's public key, we require that it is impossible to find a public key (even a "fake" one) of a VRF for which one can "prove" the correctness of two different VRF outputs for the same VRF input.

VRF PSEUDORANDOMNESS. Our VRFs are unpredictable in a very strong sense, that suitably generalizes to our context the original notion of [GGM86]. Informally, VRFs pass all *efficient statistical tests for functions* at those values for which no proof of correctness was provided. In essence, an efficient statistical test for verifiable functions is an efficient algorithm T that is given the public key of one of our functions f, and then "experiments with f" by asking and receiving both the function value and its corresponding proof of correctness at any input of its choice. After this experimentation phase, T outputs a string x in the domain of f, the *exam*, which is supposed to be different from any input on which it has already queried the function. At this point, T is provided with a value v that, with equal probability, consists of either (a) f evaluated at the exam or (b) a random value in f's range. Then T enters a "judgement phase," in which it attempts to guess whether (a) or (b) is the case (after obtaining additional function values and proofs at points of its choice other than x). We say that our VRFs pass statistical test T if the probability of T guessing correctly is not substantially better than $1/2$.

We find it convenient to think of T as comprising two components: T_E and T_J. T_E is the experimental component that queries f and computes the exam, while T_J the judging component that, given the exam and v, tries to distinguishes whether v is the value of f at the exam or a random value. To enable coordination between T_E and T_J, we let T_E pass on to T_J not only the exam, but also any piece of "state" information that it may deem useful.

3.2 A Formal Definition

Definition (VRFs): Let G, F, and V be polynomial-time algorithms, where

- G (the *function generator*) is probabilistic; receives as input a unary string (the *security parameter* k); and outputs two binary strings (the *public key* PK and *secret key* SK) ;
- $F = (F_1, F_2)$ (the *function evaluator*) is deterministic; receives as input two binary strings (SK and an input x to the VRF); and outputs two binary strings (the *value* $F_1(SK, x)$ of the VRF on x and the corresponding *proof* $= F_2(SK, x)$); and
- V (the *function verifier*) is probabilistic; receives as input four binary strings (PK, x, v, and *proof*); and outputs either YES or NO.

Let $a\colon \mathbb{N} \to \mathbb{N} \cup \{*\}$ and $b, s\colon \mathbb{N} \to \mathbb{N}$ be any three functions such that $a(k), b(k), s(k)$ are all computable in time poly(k) and $a(k)$ and $b(k)$ are both bounded by a polynomial in k (except when a takes on the value $*$). We say that (G, F, V) is a *verifiable pseudorandom function (VRF)* with *input length* $a(k)$,[4] *output length* $b(k)$, and *security* $s(k)$ if the following properties hold:

1. The following conditions hold with probability $1 - 2^{-\Omega(k)}$ over $(PK, SK) \stackrel{R}{\leftarrow} G(1^k)$:
   (a) (Domain-Range Correctness):
   for all $x \in \{0,1\}^{a(k)}$, $F_1(SK, x) \in \{0,1\}^{b(k)}$.
   (b) (Complete Provability): for all $x \in \{0,1\}^{a(k)}$, if $(v, proof) = F(SK, x)$,
   $$\text{PROB}[(V(PK, x, v, proof) = \text{YES}] > 1 - 2^{-\Omega(k)}$$
   (this probability is over the coin tosses of V).
2. (Unique Provability): For every $PK, x, v_1, v_2, proof_1$, and $proof_2$ such that $v_1 \neq v_2$, the following holds for either $i = 1$ or $i = 2$:
   $$\text{PROB}[V(PK, x, v_i, proof_i) = \text{YES}] < 2^{-\Omega(k)}$$
   (this probability over the coin tosses of V).
3. (Residual Pseudorandomness): Let $T = (T_E, T_J)$ be any pair of algorithms such that $T_E(\cdot,\cdot)$ and $T_J(\cdot,\cdot,\cdot)$ run for a total of at most $s(k)$ steps when their first input is 1^k. Then the probability that T succeeds in the following experiment is at most $1/2 + 1/s(k)$:
   (a) Run $G(1^k)$ to obtain (PK, SK).
   (b) Run $T_E^{F(SK,\cdot)}(1^k, PK)$ to obtain $(x, state)$.
   (c) Choose $r \stackrel{R}{\leftarrow} \{0,1\}$.
      i. if $r = 0$, let $v = F_1(SK, x)$.
      ii. if $r = 1$, choose $v \stackrel{R}{\leftarrow} \{0,1\}^{b(k)}$.
   (d) Run $T_J^{F(SK,\cdot)}(1^k, v, state)$ to obtain *guess*.
   (e) $T = (T_E, T_J)$ *succeeds* if $x \in \{0,1\}^{a(k)}$, $guess = r$, and x was not asked as a query to $F(SK, \cdot)$ by either T_E or T_J.

If $(PK, SK) \stackrel{R}{\leftarrow} G(1^k)$, we shall refer to $f(\cdot) = F_1(SK, \cdot)$ as an *individual VRF*. If $a(k) = *$ for all k, we say that the VRF has *unrestricted input length.*

[4]When $a(k)$ takes the value $*$, it means that the VRF is defined for inputs of all lengths. Specifically, if $a(k) = *$, then $\{0,1\}^{a(k)}$ is to be interpreted as the set of all binary strings, as usual.

Remarks.

1. Note the adversary $T = (T_E, T_J)$ is given $F(SK, \cdot)$ as an oracle, and thus gets answers that include function values and proofs of their correctness.
2. A VRF with input length $a(k)$ and output length $b(k)$ can and security $s(k)$ can be converted into one with input length $a'(k) = a(k) - \lceil \log_2 \ell(k) \rceil$, output length $b'(k) = b(k) \cdot \ell(k)$, and security $s'(k) = s(k)/\ell(k)$. Simply define $f'(x) = f(x \circ u_1) \circ f(x \circ u_2) \circ \cdots f(x \circ u_{\ell(k)})$, where $u_1, \ldots, u_{\ell(k)}$ are the first $\ell(k)$ strings of length $\lceil \log_2 \ell(k) \rceil$. (A factor of ℓ is lost in the security because it takes ℓ queries to f to simulate a single query to f', and because a factor of ℓ is lost in the adversary's success probability in the "hybrid argument" based security reduction.)

   Hence, to construct a VRF it is sufficient to fix $b = 1$ (i.e., to construct a "verifiable pseudorandom predicate"), and vice versa. In this case, residual unpredictability can be so simplified:

   3′. (Residual Pseudorandomness for Predicates): Let $T(\cdot, \cdot)$ be any algorithm that runs in time $s(k)$ when its first input is 1^k. Then the probability that T succeeds in the following experiment is at most $1/2 + 1/s(k)$:
   (a) Run $G(1^k)$ to obtain (PK, SK).
   (b) Run $T^{F(SK,\cdot)}(1^k, PK)$ to obtain $(x, guess)$.
   (c) T *succeeds* if $x \in \{0,1\}^{a(k)}$, $guess = F_1(SK, x)$, and x was not asked as a query to $F(SK, \cdot)$ by T.

   The reasons the "judgement" component T_J can be eliminated for predicates are: (a) there are only two possible values for v, so all the oracle queries that $T_J^{F(SK,\cdot)}(1^k, v, state)$ would make in case $v = 0$ or $v = 1$ can be asked before actually receiving v. (b) distinguishing a predicate $f(x)$ from a random bit with probability $1/2 + 1/s(k)$ is equivalent to guessing $f(x)$ with the same probability (cf., [Yao82]).

In order to construct a VRF, we will first construct a verifiable unpredictable function, which can also be thought of as a signature scheme in which a unique (or at most one) signature is accepted by the verification algorithm for every message and public key.

Definition (VUFs): A *verifiable unpredictable function (VUF)* (or *unique signature scheme*[5]) (G, F, V) with input length $a(k)$, output length $b(k)$, and security $s(k)$ is defined in the same way as a VRF, except that the Residual Pseudorandomness requirement is replaced with the following:

3. (Residual Unpredictability) Let $T(\cdot, \cdot)$ be any algorithm that runs in time $s(k)$ when its first input is 1^k. Then the probability that T succeeds in the following experiment is at most $1/s(k)$:

1. Run $G(1^k)$ to obtain (PK, SK).
2. Run $T^{F(SK,\cdot)}(1^k, PK)$ to obtain $(x, guess)$.
3. T *succeeds* if $x \in \{0,1\}^{a(k)}$, $guess = F_1(SK, x)$, and x was not asked as a query to $F(SK, \cdot)$ by T.

4 Formal statement of results

First, we exhibit general techniques for converting VUFs to VRFs and increasing the input length for VRFs.

Proposition 1 (from VUF to VRF) *If there is a VUF with input length $a(k)$, output length $b(k)$, and security $s(k)$, then, for any $a'(k) \le a(k)$, there is a VRF with input length $a'(k)$, output length $b(k) = 1$, and security $s'(k) = s(k)^{1/3}/(\mathrm{poly}(k) \cdot 2^{a'(k)})$.*

Proposition 2 (increasing the input length) *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 $b(k) = 1$, and security at least $\min\{s(k)^{1/5}, 2^{a(k)/5}\}/\mathrm{poly}(k)$.*

These two propositions reduce the problem of constructing VRFs to constructing VUFs. We do the latter based on a variant of the RSA assumption. We denote by PRIMES_k the set of the k-bit primes, and by RSA_k the set of composite integers that are the product of two primes of length $\lfloor (k-1)/2 \rfloor$. (For k large, RSA_k contains the hardest k-bit inputs to any known factoring algorithm.) We make the following assumption on the hardness of RSA, where the exponents are primes (1-bit) bigger than the modulus. For any function $s(k)$ computable in time poly(k):

The RSA' $s(k)$-Hardness Assumption: Let A be any probabilistic algorithm which runs in time $s(k)$ when its first input is 1^k. Then the probability that A succeeds in the following experiment is at most $1/s(k)$:

1. Select $m \xleftarrow{R} \mathrm{RSA}_k; x \xleftarrow{R} \mathbb{Z}_m^*; p \xleftarrow{R} \mathrm{PRIMES}_{k+1}$.
2. Let $y \xleftarrow{R} A(1^k, m, x, p)$.
3. A *succeeds* if $y^p = x \pmod m$.

Given the state-of-the-art in computational number theory, it seems reasonable to take $s(k) = 2^{k^\delta}$ for a small constant $\delta > 0$, though we will be able to construct VRFs as long as $s(k) = k^{\omega(1)}$.

[5] The terminology "unique signature scheme" was suggested to us by Moni Naor and Omer Reingold.

Proposition 3 (RSA-based VUFs) *Let $a(k) \leq \text{poly}(k)$ and $s(k)$ be any functions (both computable in time $\text{poly}(k)$). Under the RSA' $s(k)$-Hardness Assumption, there is a VUF with input length $a(k)$, output length $b(k) = 1$, and security $s'(k) = s(k)/\left(2^{a(k)} \cdot \text{poly}(k)\right)$.*

Putting all the above together, we conclude:

Theorem 4 *Under the RSA' $s(k)$-Hardness Assumption, there is a VRF with unrestricted input length, output length $b(k) = 1$, and security $s(k)^{1/35}/\text{poly}(k)$. In particular, if $s(k) = k^{\omega(1)}$ (i.e., RSA' cannot be inverted in polynomial time), then the VRF also has security $k^{\omega(1)}$.*

To deduce Theorem 4 we apply the above Propositions with $a(k) = a'(k) = (\log s(k))/7$. Note that this requires knowing an *a priori* lower bound $s(k)$ on the security of RSA'. However, this drawback can be removed. That is, we can build VRFs whose *construction* is independent of the hardness of RSA', while the security remains polynomially related to that of RSA'. This can be done using a standard trick, which we describe in the full version of the paper.

5 From Unpredictability to Pseudorandomness

In this section, we sketch how to prove Proposition 1, using the Goldreich–Levin [GL89] hardcore bit to convert verifiable unpredictable functions to verifiable pseudorandom function. The construction and proof will be given in more detail in the full version of the paper. Given a VUF $f(\cdot)$, the VRF $f'(\cdot)$ is defined by $f'(x) = \langle f(x), r\rangle$, where r is a binary vector chosen uniformly and placed in the public key and $\langle \cdot, \cdot \rangle$ denotes inner product mod 2. The proof that $f'(x) = \sigma$ consists of a value v such that $\langle v, r\rangle = \sigma$ and a proof that $f(x) = v$. The Domain-Range Correctness, Complete Provability, and Unique Provability of f' all follow immediately from the same properties of f.

We now outline the steps in the proof of the residual unpredictability of f'. Suppose, for sake of contradiction that there is an adversary T' running in time s' that predicts $f'(x) = \langle f(x), r\rangle$ at an unseen value with probability at least $1/2 + 1/s'$. Then,

1. T' can actually be used to guess $\langle f(x), r\rangle$ for a random, prespecified x rather than one that T' chooses its own. This can be done at the price of reducing T''s success probability to $1/2 + \varepsilon'$ for $\varepsilon' = 1/(2^{a'} \cdot s')$, because a random x will equal the exam T' chooses with probability $1/2^{a'}$. (Recall that a' is the input length for f'.)
2. By a Markov argument, at least an $\varepsilon'/2$ fraction of the x's,[6] the marginal probability that T' correctly guesses $\langle f(x), r\rangle$ taken just over the choice of r is at least $1/2 + \epsilon'/2$.
3. The Goldreich–Levin reconstruction algorithm then implies that for the same $\varepsilon'/2$ fraction x's, $f(x)$ itself can be computed with probability at least $\Omega((\varepsilon')^2)$ at a cost of increasing the running time of T by a factor of $\text{poly}(k)/(\varepsilon')^2$.
4. All together this gives an adversary T running in time $s' \cdot \text{poly}(k)/(\varepsilon')^2 \leq s$ which guesses $f(x)$ correctly at an unseen point with probability at least $(\varepsilon'/2) \cdot \Omega((\varepsilon')^2) > 1/s$, contradicting the fact that f has security s.

[6] Actually, the choice of f and the coin tosses of T should also be included and fixed with x in this $\varepsilon'/2$ probability.

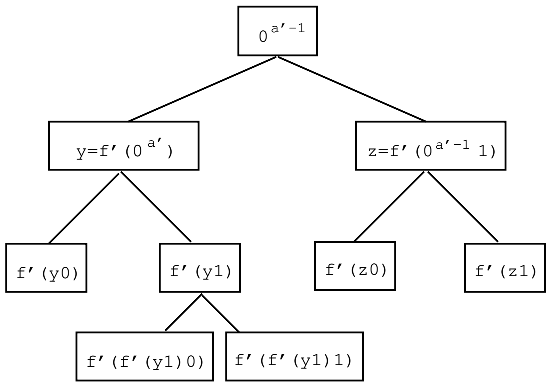

Figure 1. The tree construction

6 Increasing the input length

In this section, we sketch the proof of Proposition 2, which takes a VRF with small (but super-logarithmic) input length and converts it into a VRF with unrestricted input length. The construction and its analysis will be given in more detail in the full version of the paper. Let f be any VRF with input length a, output length 1, and security s. By Remark 2 after the definition of VRFs, we can easily transform f into a VRF f' with input length $a' = a - O(\log a)$, output length $b' = a' - 1$, and security $s' = s/b' = s/\text{poly}(k)$.

From this VRF f' which shrinks an a'-bit input by one bit, we will construct a VRF f'' which can take inputs x of arbitrary lengths. We view f' as defining an infinite binary tree whose nodes are labelled by strings of length $a' - 1$. The root of the tree is labelled $0^{a'-1}$, and the two children of a node labelled y are labelled $f'(y0)$ and $f'(y1)$ (see Figure 1). Now, to evaluate f'' on a string x, we view the bits of $x \in \{0,1\}^t$ as defining a path of length t from the root of the tree. We define $f''(x)$ to be the label of the node at the last point on this path. Now, a proof for the value of

f'' can be obtained by giving the labels of all nodes on this path together with f'-proofs for each label.

One small problem with the construction as described so far is that the path corresponding to a string x contains the path corresponding to all prefices of x, so having seen the proof for $f''(x)$, one knows the value of f'' on all prefices of x. To avoid this problem, we work with a *prefix-free encoding* of strings, which is a map $x \mapsto [x]$ from $\{0,1\}^*$ to $\{0,1\}^*$ such that there is no pair $x \neq y$ where $[x]$ is a prefix of $[y]$ and furthermore, $|[x]| = O(|x|)$ for all x. (It is easy to construct such a map which is efficiently computable, e.g., $[b_1 b_2 \cdots b_t] = b_1 b_1 b_2 b_2 \cdots b_t b_t 01$.)

So, in the actual construction, $f''(x)$ is computed as follows: Let $[x] = b_1 \cdots b_t$ and $y_0 = 0^{a'-1}$, and recursively compute $y_i = f'(y_{i-1} b_i)$. $f''(x)$ is defined to be y_t. The proof that $f''(x) = y$ is a sequence $(y_1, \ldots, y_t)$ such that $y_t = y$ together with proofs that $y_i = f'(y_{i-1} b_i)$. The Domain–Range Correctness, Complete Provability, and Unique Provability of f'' follow from the same properties of f'. The proof of Residual Pseudorandomness proceeds as follows:

1. As long as the subtree of labels seen by the adversary does not contain a repetition (i.e. two different nodes in the tree that have the same label), the value of f'' at a new point x is equal to the value of f' at a new point y (namely $y = y_{t-1} b_t$, where $[x] = b_1 b_2 \cdots b_t$). Hence, it is not be distinguishable from random.
2. The subtree of labels seen by the adversary does not contain a repetition: This follows from the residual pseudorandomness of f' and the fact that f' has a reasonably large output length b'. Suppose an efficient adversary does find a repetition with noticeable probability. Then, one can predict f' by randomly guessing which of the two nodes in the subtree form the first repetition, and using the label of the first node to predict the label of the second node. Being able to predict the value of f' at a new point with probability noticeably more than $1/2^{b'}$ distinguishes it from a random value, violating the residual pseudorandomness of f'.

7 A Verifiable Unpredictable Function

In this section we construct a VUF based on the RSA$'$ hardness assumption, proving Proposition 3. First we recall some basic number theory.

NUMBER THEORY. We write (a, b) denote the greatest common divisor of positive integers a and b. For a positive integer m, *Euler's totient function*, $\phi(m)$, is defined as the number of positive integers $< m$ that are relatively prime to m. Under multiplication modulo m, the set of all such integers form a group, denoted by $\mathbb{Z}_m^*$. In our VRF construction, we shall use the following two well-known facts about $\phi(m)$:

Fact 1: If m is the product of two distinct primes q_1 and q_2, then $\phi(m) = (q_1 - 1) \cdot (q_2 - 1)$.

Fact 2: If $(e, \phi(m)) = 1$, then the map $x \mapsto x^e \pmod{m}$ is a permutation on $\mathbb{Z}_m^*$. In particular, for any integer r, there is at most one $x \in \mathbb{Z}_m^*$ such that $x^e = r \pmod{m}$ (there will be none if $(r, m) \neq 1$). This x (if it exists) is denoted $r^{1/e}$ and one can compute it in polynomial time given inputs m, e, x, and $\phi(m)$.

As outlined in the introduction, our VUF construction is based on the unpredictable number generator of Shamir [Sha83]. The value of $f(x)$ will be defined as $r^{1/p_x} \pmod{m}$, where m and $r \in \mathbb{Z}_m^*$ are public and p_x is a prime 1-bit larger than m. To define the indexing $x \mapsto p_x$ into a "random" set of large primes, we use a prime sequence generator of Cachin et al. [CMS99], which we describe first.

THE PRIME-SEQUENCE GENERATOR. Ideally, a prime-sequence generator is a 1-1 mapping $x \mapsto p_x$ from a-bit strings to $(k+1)$-bit primes. Based on currently known results on the distribution of primes, such a mapping certainly exists, but might not be efficiently computable, unless one uses some unproven assumption — such as Cramer's conjecture. To avoid making such assumptions, we use a a construction of [CMS99], which probabilistically constructs such a mapping as follows: First, a $2k^2$-wise independent function Q from $\{0,1\}^a \times \{1, \ldots, 2k^2\}$ to the set of $(k+1)$-bit integers is randomly selected and fixed. Then, p_x is defined to be the first prime among $Q(x, 1), Q(x, 2), \ldots, Q(x, 2k^2)$. Primes are sufficiently dense so that this sequence of independent $(k+1)$-bit numbers will contain a prime with high probability, and even just the pairwise independence of Q guarantees that all the p_x's will be distinct with high probability.

To implement this idea, we need a polynomial-time primality tester `PrimalityTest`, e.g. one of the algorithms given in [SS77, Rab80]. Such an algorithm `PrimalityTest` takes a a $(k+1)$-bit integer n and $\ell = \ell(k) \leq \text{poly}(k)$ random bits and outputs 1 with high probability if n is prime and outputs 0 with high probability if n is composite. We assume that the error probability of this algorithm is at most 2^{-2k} on $(k+1)$-bit inputs. In order for the final mapping to be deterministic, the random coins of `PrimalityTest` will be externally chosen and fixed and given as input to `PrimeSeq`. Another technicality is that the $2k^2$-wise mapping Q will be defined by a polynomial over GF(2^k), so a representation of this field (i.e., an irreducible polynomial of degree k over GF(2)) must be included with Q.

Now we formally describe the prime-sequence generator `PrimeSeq`. The only modification to the construction of [CMS99] is that we force its outputs to be "truly $(k+1)$-bit" integers (i.e., without leading 0's).

Description of $\texttt{PrimeSeq}(\cdot,\cdot,\cdot)$

Inputs: an a-bit string x, a polynomial Q of degree at most $2k^2-1$ over GF(2^k) (together with a representation of the field GF(2^k)), and an ℓ-bit string *coins*.

Output: a $(k+1)$-bit integer p_x (a prime with overwhelming probability over the choice of Q and *coins*).

Code for $\texttt{PrimeSeq}(x, Q, coins)$:

1. For $j = 1,\ldots,2k^2$, let y_j be the $(k+1)$-bit string $1 \circ Q(x \circ \bar{j})$, where $\bar{j}$ denotes the j'th string in $\{0,1\}^{k-a}$ under the lexicographic order and we associate GF$(()2^k)$ with $\{0,1\}^k$.
2. Use `PrimalityTest` with random coins *coins* to test each y_j (viewed as a $(k+1)$-bit integer) for primality, and let p_x be the first (probable) prime in the sequence $y_1, y_2, \ldots, y_{2k^2}$. Output p_x.

The main property of this generator that we will use is the following.

Proposition 5 ([CMS99]) *Let $a \leq k/2$. Then, with probability at least $1-2^{-\Omega(k)}$ over Q and coins selected uniformly, $\{\texttt{PrimeSeq}(x, Q, coins) : x \in \{0,1\}^a\}$ is a set of 2^a distinct $(k+1)$-bit primes.*

THE VUF. We now decribe the VUF construction. Fix $a(k)$, the input length, and $s(k)$, the assumed hardness of RSA'; we may assume that $s(k) < 2^{\sqrt{k}}$, as known factoring algorithms (cf., [Pom90]) can be break RSA' in that much time. For notational convenience, we will usually hide the dependence of the parameters k, writing, for example, a or s instead of $a(k)$ or $s(k)$. The generation algorithm $G(\cdot)$ chooses the RSA modulus m, the public $r \in \mathbb{Z}_m^*$ whose roots will be the values of the function, and the randomization needed to fix the prime sequence (the polynomial Q and the coin tosses for `PrimalityTest`).

Description of $G(\cdot)$

Inputs: a security parameter 1^k.

Outputs: a public key $PK = (m, r, Q, coins)$ and a secret key $SK = (PK, \phi(m))$, where $m \in \mathrm{RSA}_k$; $r \in \mathbb{Z}_m^*$; $coins \in \{0,1\}^\ell$; and Q is a polynomial of degree at most $2k^2-1$ over GF(2^k) (together with a representation of GF(2^k)).

Code for $G(1^k)$*:*

1. Use `PrimalityTest` to compute (by trial and error) two random primes q_1 and q_2 (of length $\lfloor (k-1)/2 \rfloor$). Compute $m = q_1 q_2 \in \mathrm{RSA}_k$, and then compute $\phi(m) = (q_1 - 1)\cdot(q_2 - 1)$.
2. $r \xleftarrow{R} \mathbb{Z}_m^*$, $coins \xleftarrow{R} \{0,1\}^\ell$.
3. Choose a representation for GF$(()2^k)$ (by randomly picking degree k polynomials over GF$(()2)$ and testing for irreducibiility) and let Q be selected uniformly from the set of all polynomials of degree at most $2k^2-1$ over GF(2^k).
4. Output $(m, r, Q, coins)$ and $\phi(m)$.

When given $x \in \{0,1\}^a$, the evaluation algorithm F uses x to index into the prime sequence, obtaining a prime p_x, and outputs the p_x'th root of $r \in \mathbb{Z}_m^*$ as the value of the VUF at x. This value is its own proof, so we do not include a separate proof in the output. Strictly speaking, the output should be a bit-string of a fixed length $b(k)$, so elements of $\mathbb{Z}_m^*$ should be written with leading zeroes to make them of length exactly $k+1$ as strings. (Recall that m is the product of two primes of length $\lfloor (k-1)/2 \rfloor$, so $m < \left(2^{(k-1)/2+1}\right)^2 = 2^{k+1}$.)

DESCRIPTION OF $F(\cdot,\cdot)$

Inputs: A secret key $SK = (PK, \phi(m))$, where $PK = (m, r, Q, coins)$ and $x \in \{0,1\}^a$.

Output: a value $v \in \mathbb{Z}_m^*$ (which is its own proof).

Code for $F((m, r, Q, coins), \phi(m), x)$*:*

1. Compute $p_x = \texttt{PrimeSeq}(x, Q, coins)$. (We expect p_x to be a $(k+1)$-bit prime.)
2. Compute and output $v = r^{1/p_x} \pmod m$. (easily done due to knowledge of $\phi(m)$).

To check that the value of the VUF at point x is v, the main thing the verification algorithm needs to do is make sure that v is a p_x'th root of r mod m, i.e., $v^{p_x} = r \pmod m$. However, to guarantee that this value is unique, it also should check that p_x is in fact a prime larger than m and that $v \in \mathbb{Z}_m^*$.

Description of $V(\cdot,\cdot,\cdot)$

Inputs: A public key $PK = (m, r, Q, coins)$, a point x, and a value v.

Output: YES or NO.

Code for $V(1^k, (m, r, Q, coins), x, v)$*:*

1. Compute $p_x = \texttt{PrimeSeq}(x, Q, coins)$.
2. Check that p_x is greater than m and is prime (by running `PrimalityTest` using fresh random coin tosses, not those from the public key).
3. Check that $v \in \mathbb{Z}_m^*$ and $v^{p_x} = r \pmod m$.
4. If all checks pass, output YES. Otherwise, output NO.

7.1 Correctness of the VUF construction

In this section, we prove that (G, F, V) described in the previous section is in fact a VRF with security $s'(k) = s(k)^{1/7}$, establishing Proposition 3. The efficiency of the algorithms G, F, and V is apparent, so we proceed to the other conditions.

DOMAIN–RANGE CORRECTNESS & COMPLETE PROVABILITY. By Proposition 5, it is true that with probability $1 - 2^{-\Omega(k)}$ over the generation of the keys $PK = (m, r, Q, \mathit{coins})$ and $SK = (\phi(m))$, that all the values $p_x = \texttt{PrimeSeq}(x, Y, y, z)$ are primes of length $k + 1$. Since $\phi(m) < m < 2^{k+1}$, it follows that all of these p_x's are relatively prime with $\phi(m)$, and hence r has a p_x'th root modulo $\phi(m)$. Given that these roots exist, it is immediate that F will successfully compute them, establishing Domain–Range Correctness. Complete Provability also follows immediately; the only reason V would reject a correct proof is a faulty execution of the primality testing algorithm `PrimalityTest` (which occurs with exponentially small probability).

UNIQUE PROVABILITY. Assume that an adversary chooses a (good-looking but illegitimate) public key $(m, r, Q, \mathit{coins})$ and consider any input x. If $p_x \stackrel{\text{def}}{=} \texttt{PrimeSeq}(x, Q, \mathit{coins})$ is not prime or is not larger than m, then the verification algorithm will detect this and reject with high probability. If p_x is a prime larger than m, then p_x must be relatively prime to $\phi(m)$, so r has a unique p_x'th root mod m, and this is the only value that the verification algorithm will accept.

RESIDUAL UNPREDICTABILITY. Suppose, for sake of contradiction, (G, F, V) is not an $s'(k)$-secure VUF and let T be the adversary running in time $s'(k)$ that guesses the value of the function at an unseen point with probability at least $1/s'(k)$.

We will use T to construct an algorithm A that contradicts the RSA$'$ $s(k)$-Hardness Assumption. A will be given a modulus m, a prime p, and $u \in \mathbb{Z}_m^*$ as input, from which it will construct a public key PK which it will give to T. Thus, we first concentrate on how the public key $PK = (m, r, Q, \mathit{coins})$ will be constructed. Q will be chosen in such a way that $\texttt{PrimeSeq}(x_0, Q, \mathit{coins}) = p$ for a specified $x_0 \in \{0,1\}^a$. This means that $1 \circ Q(x \circ \bar{j}_0)$ should equal p for some $j_0 \in \{1, \ldots, 2k^2\}$, while $1 \circ Q(x \circ \bar{j})$ should be composite for $j < j_0$. We want the distribution of Q obtained in this way (when p is a random $(k+1)$-bit prime) to be close to its distribution in the actual scheme, which is uniform. This is done using the following procedure:

Description of $\texttt{ChoosePoly}(\cdot, \cdot)$

Inputs: a prime p of length $k + 1$, and $x_0 \in \{0,1\}^a$.

Output: a polynomial Q of degree at most $2k^2 - 1$ over GF(2^k) and a ℓ-bit string *coins* (such that $\texttt{PrimeSeq}(x_0, Q, \mathit{coins}) = p$)

Code for $\texttt{ChoosePoly}(p, x_0)$:

1. $w_1, \ldots, w_{2k^2} \stackrel{R}{\leftarrow} \{0,1\}^k$
2. Let j_0 be the smallest j such that $1 \circ w_j$ is prime (by running `PrimalityTest` on each of them).
3. Choose and fix a representation for GF(2^k) (exactly as done in the generation algorithm G).
4. Let Q be the unique polynomial of degree at most $2k^2 - 1$ over GF(2^k) subject to the conditions $Q(x_0 \circ \bar{j}_0) = p$ and $Q(x_0 \circ \bar{j}) = w_j$ for all $j \neq j_0$ (where $\bar{j}$ denotes the $(k-a)$-bit representation of j, with possible leading zeroes). This step can be implemented using standard polynomial interpolation.
5. $\mathit{coins} \stackrel{R}{\leftarrow} \{0,1\}^\ell$.
6. Output (Q, coins).

Claim 6 *For every* $x_0 \in \{0,1\}^a$, *the distribution on* (Q, coins) *obtained by running* $\texttt{ChoosePoly}(p, x_0)$ *for a random prime* p *of length* $k + 1$ *has statistical difference*[7] $2^{-\Omega(k)}$ *from the uniform distribution on* (Q, coins).

It is straightforward to verify this claim using Proposition 5 and the fact that the error probability of `PrimalityTest` is 2^{-2k}. Of course, (Q, coins) is only part of the public key. We now describe how the remainder of the public key is generated. On input (m, p, u), the following algorithm G' will "guess" which point x_0 the adversary T will choose as its exam; use `ChoosePoly` to guarantee that $p_{x_0} = p$; and, following [Sha83], prepare $r \in \mathbb{Z}_m^*$ so that the p_x'th root of r can be easily computed for all $x \neq x_0$, while the p_{x_0}'th root of r can be used to compute the p'th root of u. (This will all be proven in more detail shortly.)

Description of $G'(\cdot, \cdot, \cdot)$

Inputs: a modulus $m \in \text{RSA}_k$, a prime p of length $k + 1$, and $u \in \mathbb{Z}_m^*$.

Output: $(m, r, Q, \mathit{coins})$ and $x_0 \in \{0,1\}^a$.

Code for $G'(m, p, u)$:

1. $x_0 \stackrel{R}{\leftarrow} \{0,1\}^a$.
2. $(Q, \mathit{coins}) \stackrel{R}{\leftarrow} \texttt{ChoosePoly}(p, x_0)$.

[7] The *statistical difference* between two random variables X and Y is defined to be $\max_S |\text{PROB}[X \in S] - \text{PROB}[Y \in S]|$.

3. Set $e = \prod_{x \neq x_0} \mathtt{PrimeSeq}(x, Q, \mathit{coins})$ and $r = u^e \pmod{m}$.
4. Output $(m, r, Q, \mathit{coins})$ and x_0.

Claim 7 *The distribution on $((m, r, Q, \mathit{coins}), x_0)$ obtained by running G' on $m \xleftarrow{R} \mathrm{RSA}_k$, $p \xleftarrow{R} \mathrm{PRIMES}_{k+1}$, $u \xleftarrow{R} \mathbb{Z}_m^*$ has statistical difference at most $2^{-\Omega(k)}$ from the distribution obtained by running $G(1^k)$ to select $(m, r, Q, \mathit{coins})$ and independently selecting x_0 uniformly in $\{0,1\}^a$.*

Claim 7 is easily deduced from Claim 6 and the fact that the map $u \mapsto u^e$ is a permutation on $\mathbb{Z}_m^*$ as long as $(e, \phi(m)) = 1$ (which is the case, since e is the product of primes greater than $\phi(m)$ with high probability). By Claim 7, if T is presented with a public key generated by G', it's success probability will be reduced to by only an exponentially small amount to $1/s'(k) - 2^{-\Omega(k)}$. In addition, since x_0 is independent from the public key produced by G' (up to statistical difference $2^{-\Omega(k)}$), the event that T chooses x_0 as its exam is also independent of T's success. Hence, additionally requiring that T's success be at x_0 only decreases the success probability by a factor of $1/2^a$. To formalize this, we consider the following experiment.

Experiment A:

1. $m \xleftarrow{R} \mathrm{RSA}_k$; $p \xleftarrow{R} \mathrm{PRIMES}_{k+1}$; $u \xleftarrow{R} \mathbb{Z}_m^*$
2. $((m, r, Q, \mathit{coins}), x_0) \xleftarrow{R} G'(m, p, u)$
3. Set $PK = (m, r, Q, \mathit{coins})$, $SK = (PK, \phi(m))$
4. $(x, \mathit{guess}) = T^{F(SK, \cdot)}$
5. T *succeeds* if $x = x_0$, $\mathit{guess} = F(SK, x)$ (i.e., $\mathit{guess}^p = r \pmod{m}$), and x was not asked to the oracle $F(SK, \cdot)$.

By Claim 7 and the above discussion, it follows that the probability that T succeeds in Experiment A is at least $\varepsilon' \stackrel{\text{def}}{=} 1/(2^a \cdot s'(k)) - 2^{-\Omega(k)} > 1/s$.

Now we use the analysis of Shamir [Sha83], which shows that since $r = u^e$ where $e = \prod_{x' \neq x_0} p_{x'}$, it is easy to answer all of T's queries for $F(SK, x')$ (for $x' \neq x_0$) without using $\phi(m)$. In addition, from $F(SK, x_0) = r^{1/p}$, it is easy to compute $u^{1/p}$. In more detail, we consider the following algorithm A.

Description of $A(\cdot, \cdot, \cdot)$

Inputs: a k-bit modulus m, a prime p of length $k+1$, and $u \in \mathbb{Z}_m^*$.

Output: $u^{1/p}$ (hopefully)

Code for $A(m, p, u)$:

1. $((m, r, Q, \mathit{coins}), x_0) \xleftarrow{R} G'(m, p, u)$
2. Set $PK = (m, r, Q, \mathit{coins})$ and $e = \prod_{x \neq x_0} p_x$, where $p_x \stackrel{\text{def}}{=} \mathtt{PrimeSeq}(x, Q, \mathit{coins})$.
3. Simulate $T(1^k, PK)$. Respond to an oracle query y as follows:
   (a) If $y = x_0$, abort with output FAIL.
   (b) If $y \neq x_0$, respond with $r^{1/p_y} = u^{e_y} \pmod{m}$, where $e_y = e/p_y$.
4. Obtain output (x, guess) from T.
5. If $\mathit{guess}^p \neq r \pmod{m}$, then output FAIL.
6. If $\mathit{guess}^p = r$, use the GCD algorithm to calculate $\alpha, \beta \in \mathbb{Z}$ such that $\alpha e + \beta p = 1$, and output $\mathit{guess}^\alpha u^\beta$.

Claim 8 $A(m, p, u) = u^{1/p} \pmod{m}$ *with probability at least $\varepsilon' > 1/s$ (over the choice of $m \xleftarrow{R} \mathrm{RSA}_k$, $p \xleftarrow{R} \mathrm{PRIMES}_{k+1}$, $u \xleftarrow{R} \mathbb{Z}_m^*$, and the coins of A).*

We now quickly justify this claim. A straightforward calculation shows that the responses to T's oracle queries are computed correctly (when $y \neq x_0$). Thus, as long as T does not ask oracle query x_0, everything proceeds exactly as in Experiment A. Our analysis of Experiment A tells us that with probability at least ε', T' does not ask query x_0 and $\mathit{guess} = r^{1/p_{x_0}} = r^{1/p}$. The GCD algorithm will succeed as long as all the p_x's are distinct, and this is the case with overwhelming probability by Proposition 5 and Claim 6. Assuming $\mathit{guess} = r^{1/p}$ and the GCD algorithm succeeds, it follows that

$$\begin{aligned} \mathit{guess}^\alpha u^\beta &= r^{\alpha/p} u^\beta \\ &= (u^e)^{\alpha/p} u^\beta \\ &= u^{(\alpha e + \beta p)/p} = u^{1/p} \pmod{m}. \end{aligned}$$

We now just need to analyze the running time of A. A's running time is dominated by simulating the oracle queries of T. For every oracle query of T, A must compute $u^{e_y} \bmod m$, where e_y is an integer of length $O(2^a \cdot k)$ (since e is the product of $2^a - 1$ primes of length $k+1$). This modular exponenation takes time $O(2^a \cdot k) \cdot \mathrm{poly}(k) = 2^a \cdot \mathrm{poly}(k)$. Since there T makes at most s' oracle queries, the total running time is at most $s' \cdot 2^a \cdot \mathrm{poly}(k) \leq s$, violating the RSA$'$ $s(k)$-Hardness Assumption. ■

Acknowledgments

We thank Oded Goldreich, Shafi Goldwasser, Shai Halevi, Joe Kilian, and David Mazieres, Moni Naor, and Omer Reingold for their insightful comments and suggestions.

References

[BG89] Mihir Bellare and Shafi Goldwasser. New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 194–211. Springer-Verlag, 1990, 20–24 August 1989.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, December 1991.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 103–112, Chicago, Illinois, 2–4 May 1988.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Advances in Cryptology—EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer-Verlag, 1999, 2–6 May 1999.

[CD96] Ronald Cramer and Ivan Damgård. New generation of secure and practical RSA-based signatures. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 173–185. Springer-Verlag, 18–22 August 1996.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. Technical Report 99-01, Theory of Cryptography Library, January 1999. See also revision, July 1999.

[DN94] Cynthia Dwork and Moni Naor. An efficient existentially unforgeable signature scheme and its applications. In Yvo G. Desmedt, editor, *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 234–246. Springer-Verlag, 21–25 August 1994.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In J. Stern, editor, *Advances in Cryptology—EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer-Verlag, 1999, 2–6 May 1999.

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

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMY83] Shafi Goldwasser, Silvio Micali, and Andy Yao. Strong signature schemes. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 431–439, Boston, Massachusetts, 25–27 April 1983.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions (extended abstract). In *38th Annual Symposium on Foundations of Computer Science*, pages 458–467, Miami Beach, Florida, 20–22 October 1997. IEEE.

[NR98] Moni Naor and Omer Reingold. From unpredictability to indistinguishability: A simple construction of pseudorandom functions from macs (extended abstract). In Hugo Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 267–282. Springer-Verlag, 23–27 August 1998.

[Pom90] Carl Pomerance. Factoring. In Carl Pomerance, editor, *Cryptology and Computational Number Theory*, volume 42 of *Proceedings of Symposia in Applied Mathematics*, pages 27–47. American Mathematical Society, 1990.

[Rab80] Michael O. Rabin. Probabilistic algorithms for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems*, 1(1):38–44, February 1983.

[SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, March 1977.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 November 1982. IEEE.