Network Working Group S. Goldberg
Internet-Draft Boston University
Intended status: Standards Track D. Papadopoulos
Expires: September 14, 2017 University of Maryland
J. Vcelak
NS1
March 13, 2017

Verifiable Random Functions (VRFs)
draft-goldbe-vrf-00

Abstract

A Verifiable Random Function (VRF) is the public-key version of a keyed cryptographic hash. Only the holder of the private key can compute the hash, but anyone with public key can verify the correctness of the hash. VRFs are useful for preventing enumeration of hash-based data structures. This document specifies several VRF constructions that are secure in the cryptographic random oracle model. One VRF uses RSA and the other VRF uses Eliptic Curves (EC).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. Rationale

A Verifiable Random Function (VRF) [MRV99] is the public-key version of a keyed cryptographic hash. Only the holder of the private VRF key can compute the hash, but anyone with corresponding public key can verify the correctness of the hash.

The main application of the VRF is to protect the privacy of data records stored in a hash-based data structure against a querying adversary. In this application, a prover holds the VRF secret key and uses the VRF hashing to construct a hash-based data structure on the input data. Due to the nature of the VRF hashing, only the prover can answer queries about whether or not some data is stored in the data structure. Anyone who knows the public VRF key can verify that the prover has answered the queries correctly. However no offline inferences (i.e. inferences without querying the prover) can be made about the data stored in the data strucuture.

1.2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.3. Terminology

The following terminology is used through this document:

SK: The private key for the VRF.

PK: The public key for the VRF.

alpha: The input to be hashed by the VRF.

beta: The VRF hash output.

pi: The VRF proof.

2. VRF Algorithms

A VRF comes with a key generation algorithm that generates a public VRF key PK and private VRF key SK.

A VRF hashes an input alpha using the private VRF key SK to obtain a VRF hash output beta:

```
beta = VRF_hash(SK, alpha)
```

The VRF_hash algorithm MUST be deterministic, in the sense that it will always produce the same output beta given a pair of inputs (SK, alpha). The private key SK is also used to construct a proof pi that beta is the correct hash output

```
pi = VRF_prove(SK, alpha)
```

The VRFs defined in this document allow anyone to deterministically obtain the VRF hash output beta directly from the proof value pi as

```
beta = VRF_proof2hash(pi)
```

Notice that this means that

```
VRF_hash(SK, alpha) = VRF_proof2hash(VRF_prove(SK, alpha))
```

The proof pi allows anyone holding the public key PK to verify that beta is the correct VRF hash of input alpha under key PK. Thus, the VRF also comes with an algorithm

```
VRF_verify(PK, alpha, pi)
```

that outputs VALID if beta=VRF_proof2hash(pi) is correct VRF hash of alpha under key PK, and outputs INVALID otherwise.

3. VRF Security Properties

VRFs are designed to ensure the following security properties.

3.1. Full Uniqueness or Trusted Uniqueness

Uniqueness states that, for any fixed public VRF key and for any input alpha, there is a unique VRF output beta that can be proved to be valid, even for a computationally-bounded adversary that knows the VRF secret key SK.

More precisely, full uniqueness states that a computationally bounded adversary cannot choose a VRF public key PK, a VR input alpha, two different VRF hash outputs beta1 and beta2, and two proofs pi1 and pi2 such that VRF_verify(PK, alpha, pi1) and VRF_verify(PK, alpha, pi2) both output VALID.

A slightly weaker security property called "trusted uniqness" sufficies for many applications. Trusted uniqueness is the same as full uniqueness, but it must hold only if the VRF keys PK and SK were

generated in a trustworthy manner. In otherwords, uniqueness might not hold if keys were generated in an invalid manner.

3.2. Full Pseudorandomness or Selective Pseudorandomness

Suppose the public and private VRF keys (PK, SK) were generated in a trustworthy manner.

Pseudorandomness ensures that the VRF hash output beta (without its corresponding VRF proof pi) on any adversarially-chosen "target" VRF input alpha looks indistinguishable from random for any computationally bounded adversary who does not know the private VRF key SK. This holds even if the adversary also gets to choose other VRF inputs alpha' and observe their corresponding VRF hash outputs beta' and proofs pi'.

With "full pseudorandomness", the adversary is allowed to choose the target VRF input alpha at any time, even after it observes VRF outputs beta' and proofs pi' on a variety of chosen inputs alpha'.

"Selective pseudorandomness" is a weaker security property which suffices in many applications. Here, the adversary must choose the target VRF input alpha independently of the public VRF key PK, and before it observes VRF outputs beta' and proofs pi' on inputs alpha' of its choice.

It is important to remember that the VRF output beta does not look random to a party that knows the private VRF key SK! Such a party can easily distinguish beta from a random value by comparing it to the result of VRF_hash(SK, alpha).

Also, the VRF output beta does not look random to any party that knows valid VRF proof pi corresponding to the VRF input alpha, even if this party does not know the private VRF key SK. Such a party can easily distinguish beta from a random value by checking whether VRF_verify(PK, alpha, pi) returns "VALID" and beta = VRF_proof2hash(pi).

Finally, the VRF output beta may not look random if VRF key generation was not done in a trustworthy fashion. (For example, if VRF keys were not generated with good randomness.)

3.3. Full Collison Resistance or Trusted Collision Resistance

Finally, like any cryprographic hash function, VRFs need to be collision resistant. Specifically, it should be computationally infeasible for an adversary to find two distinct VRF inputs alpha1

and alpha2 that have the same VRF hash beta, even if that adversary knows the secret VRF key SK.

For most applications, a slightly weaker security property called "trusted collision resistance" suffices. Trusted collision resistance is the same as collision resistance, but it holds only if PK and SK were generated in a trustworthy manner.

4. RSA Full Domain Hash VRF (RSA-FDH-VRF)

The RSA Full Domain Hash VRF (RSA-FDH-VRF) is VRF that satisfies the trusted uniqueness, full pseudorandomness, and trusted collision resistance properties defined in Section 3. Its security follows from the standard RSA assumption in the random oracle model. Formal security proofs are in [nsec5ecc].

The VRF computes the proof pi as a deterministic RSA signature on input alpha using the RSA Full Domain Hash Algorithm [RFC8017] parametrized with the selected hash algorithm. RSA signature verification is used to verify the correctness of the proof. The VRF hash output beta is simply obtained by hashing the proof pi with the selected hash algorithm.

The key pair for RSA-FDH-VRF MUST be generated in a way that it satisfies the conditions specified in Section 3 of [RFC8017].

In this document, the notation from [RFC8017] is used.

Used parameters:

(n, e) - RSA public key

K - RSA private key

k - length in octets of the RSA modulus n

Fixed options:

Hash - cryptographic hash function

hLen - output length in octets of hash function Hash

Options constraints:

Cryptographic security of Hash is at least as high as the cryptographic security level of the RSA key

Used primitives:

I2OSP - Coversion of a nonnegative integer to an octet string as defined in Section 4.1 of [RFC8017]

OS2IP - Coversion of an octet string to a nonnegative integer as defined in Section 4.2 of [RFC8017]

RSASP1 - RSA signature primitive as defined in Section 5.2.1 of [RFC8017]

RSAVP1 - RSA verification primitive as defined in Section 5.2.2 of [RFC8017]

MGF1 - Mask Generation Function based on a hash function as defined in Section B.2.1 of [RFC8017]

4.1. RSA-FDH-VRF Proving

RSAFDHVRF_prove(K, alpha)

Input:

K - RSA private key

alpha - VRF hash input, an octet string

Output:

pi - proof, an octet string of length k

Steps:

1. EM = MGF1(alpha, k - 1)

2. m = OS2IP(EM)

3. s = RSASP1(K, m)

4. pi = I2OSP(s, k)

5. Output pi

4.2. RSA-FDH-VRF Proof To Hash

RSAFDHVRF_proof2hash(pi)

Input:

pi - proof, an octet string of length k

Output:

beta - VRF hash output, an octet string of length hLen

Steps:

1. beta = Hash(pi)

2. Output beta

4.3. RSA-FDH-VRF Verifying

RSAFDHVRF_verify((n, e), alpha, pi)

Input:

(n, e) - RSA public key

alpha - VRF hash input, an octet string

pi - proof to be verified, an octet string of length n

Output:

"VALID" or "INVALID"

Steps:

1. s = OS2IP(pi)

2. m = RSAVP1((n, e), s)

3. EM = I2OSP(m, k - 1)

4. EM' = MGF1(alpha, k - 1)

5. If EM and EM' are the same, output "VALID"; else output "INVALID".

5. Elliptic Curve VRF (EC-VRF)

The Elliptic Curve Verifiable Random Function (EC-VRF) is VRF that satisfies the trusted uniqueness, full pseudorandomness, and trusted collision resistance properties defined in Section 3. The security of this VRF follows from the decisional Diffie-Hellman (DDH) assumption in the cyclic group in the random oracle model. Formal security proofs are in [nsec5ecc].

The key pair generation primitive is specified in Section 3.2.1 of [SECG1].

Fixed options:

G - EC group

q - prime order of group G

g - generator of group G

2n - ceil(log2(q)/8); where log2(x) is the binary logarithm of x and ceil(x) is the smallest integer larger than or equal to the real number x.

Hash - cryptographic hash function

hLen - output length in octets of function Hash

Options constraints:

Cryptographic security of Hash is at least as high as the cryptographic security of G

hLen is equal to 2n

Used parameters:

g^x - EC public key

x - EC private key

Used primitives:

"" - empty octet string

|| - octet string concatenation

p^k - EC point multiplication

p1*p2 - EC point addition

h[i] - the i'th octet of octet string h

ECP2OS - EC point to octet string conversion with point compression as specified in Section 2.3.3 of [SECG1]

OS2ECP - octet string to EC point conversion with point compression as specified in Section 2.3.4 of [SECG1]

5.1. EC-VRF Proving

ECVRF_prove(g^x, x, alpha)

Input:

g^x - EC public key

x - EC private key

alpha - VRF input, octet string

Output:

pi - VRF proof, octet string of length 5n+1

Steps:

```
1.  h = ECVRF_hash_to_curve(alpha, g^x)

2.  gamma = h^x

3.  choose a random nonce k from [0, q-1]

4.  c = ECVRF_hash_points(g, h, g^x, h^x, g^k, h^k)

5.  s = k - c*q mod q

6.  pi = ECP2OS(gamma) || I2OSP(c, n) || I2OSP(s, 2n)

7.  Output pi
```

5.2. EC-VRF Proof To Hash

ECVRF_proof2hash(pi)

Input:

pi - VRF proof, octet string of length 5n+1

Output:

beta - VRF hash output, octet string of length 2n

Steps:

```
1.  beta = pi[2] || pi[3] || ... pi[2n+1]

2.  Output beta
```

5.3. EC-VRF Verifying

```
ECVRF_verify(g^x, pi, alpha)

Input:

   g^x - EC public key

   pi - VRF proof, octet string of length 5n+1

   alpha - VRF input, octet string

Output:

   "VALID" or "INVALID"

Steps:

1.  gamma, c, s = ECVRF_decode_proof(pi)

2.  If gamma is not a valid EC point in G, output "INVALID" and stop.

3.  u = (g^x)^c * g^s

4.  h = ECVRF_hash_to_curve(alpha, g^x)

5.  v = gamma^c * h^s

6.  c' = ECVRF_hash_points(g, h, g^x, gamma, u, v)

7.  If c and c' are the same, output "VALID"; else output "INVALID".
```

5.4. EC-VRF Auxiliary Functions

5.4.1. EC-VRF Hash To Curve

The ECVRF_hash_to_curve algorithm takes in an octet string alpha and converts it to h, an EC point in G.

5.4.1.1. ECVRF_hash_to_curve1

The following ECVRF_hash_to_curve1(alpha, g^x) algorithm implements ECVRF_hash_to_curve in a simple and generic way that works for any elliptic curve that supports point compression.

However, this algorithm MUST NOT be used in applications where the VRF input alpha must be kept secret. This is because the running time of the hashing algorithm depends on alpha, and so it is susceptible to timing attacks. That said, the amount of information obtained from such a timing attack is likely to be small, since the algorithm is expected to find a valid curve point after only two attempts (i.e., when ctr=1) on average (see [Icart09]).

ECVRF_hash_to_curve1(alpha, g^x)

Input:

   alpha - value to be hashed, an octet string

   g^x - EC public key

Output:

   h - hashed value, EC point in G

Steps:

```
1.  ctr = 0

2.  pk = ECP2OS(g^x)

3.  Repeat:

    A.  CTR = I2OSP(ctr, 4)

    B.  p = 0x02 || Hash(alpha || pk || CTR)

    C.  Goto step 3 if OS2ECP(p) is valid EC point in G

    D.  p = 0x03 || Hash(alpha || pk || CTR)

    E.  Goto step 3 if OS2ECP(p) is valid EC point in G

    F.  ctr = ctr + 1

4.  h = OS2ECP(p)

5.  Output h
```

The initial octet 0x02 in the octet string created in step B represents that the point in compressed form has positive y-coefficient [SECG1]. Similarly, the 0x03 octet in step D represents negative y-coefficient.

5.4.1.2. ECVRF_hash_to_curve2

For applications where VRF input alpha must be kept secret, the following ECVRF_hash_to_curve algorithm MAY be used to used as generic way to hash an octet string onto any elliptic curve.

[TODO: If there interest, we could look into specifying the generic deterministic time hash_to_curve algorithm from [Icart09].]

5.4.2. EC-VRF Hash Points

```
ECVRF_hash_points(p_1, p_2, ..., p_j)

Input:

   p_i - EC point in G

Output:

   h - hash value, integer between 0 and 2^(8n)-1

Steps:

1.  P = ""

2.  for p_i in [p_1, p_2, ... p_j]:
    P = P || ECP2OS(p_i)

3.  h' = Hash(P)

4.  h = OS2IP(h'[1] || h'[2] || ... h'[n])

5.  Output h
```

5.4.3. EC-VRF Decode Proof

```
ECVRF_decode_proof(pi)

Input:

   pi - VRF proof, octet string (5n+1 octets)

Output:

   gamma - EC point

   c - integer between 0 and 2^(8n)-1
```

s - integer between 0 and 2^(16n)-1

Steps:

1. let gamma', c', s' be pi split after (2n+1)-th and (3n+1)-th octet

2. gamma = OS2ECP(gamma')

3. c = OS2IP(c')

4. s = OS2IP(s')

5. Output gamma, c, and s

5.5. EC-VRF Ciphersuites

[Seeking feedback on this section!]

This document defines EC-VRF-P256-SHA256 as follows:

o The EC group G is the NIST-P256 elliptic curve, with curve parameters as specified in [FIPS-186-3] (Section D.1.2.3) and [RFC5114] (Section 2.6). For this group, the length in octets of a single coordinate of an EC point is 2n = 32.

o The hash function Hash is SHA-256 as specified in [RFC6234].

o The ECVRF_hash_to_curve function is ECVRF_hash_to_curve1, as specified in Section 5.4.1.1.

This document defines EC-VRF-ED25519-SHA256 as follows:

o The EC group G is the Ed25519 elliptic curve with parameters defined in [RFC7748] (Section 4.1). For this group, the length in octets of a single coordinate of an EC point is 2n = 32.

o The hash function Hash is SHA-256 as specified in [RFC6234].

o The ECVRF_hash_to_curve function is as specified in Section 5.4.1.1.

[TODO: Should we add an EC-VRF-ED25519-SHA256-Elligator ciphersuite where the Elligator hash function is used for ECVRF_hash-to-curve?]

[TODO: Add an Ed448 ciphersuite?]

6. Implementation Status

An implementation of the RSA-FDH-VRF (SHA-256) and EC-VRF-P256-SHA256 was developed as a part of the NSEC5 project [I-D.vcelak-nsec5] and is available at <http://github.com/fcelda/nsec5-crypto>.

The Key Transparency project at Google uses a VRF implemention that is almost identical to the EC-VRF-P256-SHA256 specified here, with a few minor changes including the use of SHA-512 instead of SHA-256. Its implementation is available <https://github.com/google/keytransparency/blob/master/core/vrf/vrf.go>

Open Whisper Systems also uses a VRF very similar to EC-VRF-ED25519-SHA512-Elligator, called VXEdDSA, and specified here: <https://whispersystems.org/docs/specifications/xeddsa/>

7. Security Considerations

7.1. Key Generation

Applications that use the VRFs defined in this document MUST ensure that that the VRF key is generated correctly, using good randomness. Without good randomness, pseudorandomness properties of the VRF may not hold. Also, trusted uniqueness and trusted collision-resistance may also not hold if the keys are generated adversarially (e.g., the RSA modulus is not a product of two primes for the RSA-FDH-VRF or the public key g^x is not valid point in the prime-order group G for the EC).

Full uniqueness and full collision-resistance (as opposed to trusted uniqueness and trusted collision-resistance) are properties that hold even if VRF keys are generated by an adversary. The VRFs defined in this document do not have these properties. However, they may be modifed to have these properties if adversarial key generation is a concern. The modification consists of additional cryptographic proofs that keys have of the correct form. These modifications are left for future specification.

Note that for the RSA-FDH-VRF, it might be possible to construct such a proof using the [GQ88] identification protocol made non-interactive using the Fiat-Shamir heuristic in the random oracle model.

However, it is not possible to guarantee pseudorandomness in the face of adversarially generated VRF keys. This is because an adversary can always use bad randomness to generate the VRF keys, and thus, the VRF output may not be pseudorandom.

7.2. Proper randomness for EC-VRF

Applications that use the EC-VRF defined in this document MUST ensure that the random nonce k used in the ECVRF_prove algorithm is chosen with proper randomness. Otherwise, an adversary may be able to recover the private VRF key x (and thus break pseudorandomness of the VRF) after observing several valid VRF proofs pi.

7.3. Timing attacks

The EC-VRF_hash_to_curve algorithm defined in Section 5.4.1.1 should not be used in applications where the VRF input alpha is secret and is hashed by the VRF on-the-fly. This is because the EC-VRF_hash_to_curve algorithm's running time depends on the VRF input alpha, and thus creates a timing channel that can be used to learn information about alpha.

7.4. Selective vs Full Pseudorandomness

[nsec5ecc] presents cryptographic reductions to an underlying hard problem (e.g. Decisional Diffie Hellman, or the standard RSA assumption) that prove the VRFs specificied in this document possess full pseudorandomness as well as selective pseudorandomness. However, the cryptographic reductions are tighter for selective pseudorandomness than for full pseudorandomness. This means the the VRFs have quantitavely stronger security guarentees for selective pseudorandomness.

Applications that are concerned about tightness of cryptographic reductions therefor have two options.

- They may choose to ensure that selective pseudorandomness is sufficient for the application. That is, that pseudorandomness of outputs matters only for inputs that are chosen independently of the VRF key.

- If full pseudorandomness is required for the application, the application may increase security parameters to make up for the loose security reduction. For RSA-FDH-VRF, this means increasing the RSA key length. For EC-VRF, this means increasing the cryptographic strength of the EC group G. For both RSA-FDH-VRF and EC-VRF the cryptographic strength of the hash function Hash may also potentially need to be increased.

8. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

00 - Forked this document from draft-vcelak-nsec5-04. Cleaned up the definitions of VRF algorithms. Added security definitions for VRF and security considerations. Parameterized EC-VRF so it could support curves other than P-256 and Ed25519.

9. Contributors

Leonid Reyzin (Boston University) made major contributions to this document. This document also would not be possible without the work of Moni Naor (Weizmann Institute), Sachin Vasant (Cisco Systems), and Asaf Ziv (Facebook). Shumon Huque (Salesforce) and David C. Lawerence (Akamai) provided valuable input to this draft.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <http://www.rfc-editor.org/info/rfc8017>.

[RFC5114] Lepinski, M. and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards", RFC 5114, DOI 10.17487/RFC5114, January 2008, <http://www.rfc-editor.org/info/rfc5114>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <http://www.rfc-editor.org/info/rfc7748>.

[I-D.vcelak-nsec5]
Vcelak, J., Goldberg, S., Papadopoulos, D., Huque, S., and D. Lawrence, "NSEC5, DNSSEC Authenticated Denial of Existence", draft-vcelak-nsec5-04 (work in progress), March 2017.

[FIPS-186-3]
National Institute for Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-3, June 2009.

[SECG1] Standards for Efficient Cryptography Group (SECG), "SEC 1: Elliptic Curve Cryptography", Version 2.0, May 2009, <http://www.secg.org/sec1-v2.pdf>.

10.2. Informative References

[nsec5ecc]
Papadopoulos, D., Wessels, D., Huque, S., Vcelak, J., Naor, M., Reyzin, L., and S. Goldberg, "NSEC5 from Elliptic Curves", in ePrint Cryptology Archive 2017/099, February 2017, <https://eprint.iacr.org/2017/099.pdf>.

[MRV99] Michali, S., Rabin, M., and S. Vadhan, "Verifiable Random Functions", in FOCS, 1999.

[CP92] Chaum, D. and C. Pederson, "Wallet databases with observers", in FOCS, 1992.

[Icart09] Icart, T., "How to Hash into Elliptic Curves", in CRYPTO, 2009.

[GQ88] Guillou, L. and JJ. Quisquater, "A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory", in Advances in Cryptology - EUROCRYPT '88, 1988.

Appendix A. Open Issues

Note to RFC Editor: please remove this appendix before publication as an RFC.

1. Open issues

Authors' Addresses

Sharon Goldberg
Boston University
111 Cummington St, MCS135
Boston, MA 02215
USA

EMail: goldbe@cs.bu.edu

Dimitrios Papadopoulos
University of Maryland
8223 Paint Branch Dr
College Park, MD 20740
USA

EMail: dipapado@bu.edu

Jan Vcelak
NS1
16 Beaver St
New York, NY 10004
USA

EMail: jvcelak@ns1.com