

Bandersnatch VRF-AD Specification

Seyed Hosseini

Davide Galassi

Draft 2 - 04-04-2024

Abstract

This technical specification introduces the Verifiable Random Function with Additional Data (VRF-AD), a cryptographic construct that extends the capabilities of the IETF's Elliptic Curve Verifiable Random Functions (ECVRF) as defined in RFC9381 [1]. The document also discusses the Pedersen VRF, a novel variation proposed by Burdges [2], which serves as a fundamental component for implementing anonymized ring signatures. Additionally, the specification provides detailed insights into the usage of these primitives within the Bandersnatch [3], an elliptic curve constructed over the BLS12-381 scalar field.

1. Introduction

Definition: A *verifiable random function with additional data (VRF-AD)* can be described with two functions:

- $Sign(sk, msg, ad) \mapsto \pi$: from a secret key sk , an input msg , and additional data ad , and returns a signature π .
- $Verify(pk, msg, ad, \pi) \mapsto (out|prep)$: for a public key pk , an input msg , additional data ad , and VRF signature π returns either an output out or else a failure $prep$.

Definition: For an elliptic curve E defined over finite field \mathbb{F}_p with large subgroup $\langle G \rangle$ with prime order r generated by the base point G , an EC-VRF is VRF-AD where $pk = sk \cdot G$ and VRF *Sign* is based on an elliptic curve signature scheme.

All VRFs described in this specification are EC-VRF.

2. Preliminaries

2.1. VRF Input

A point in $\langle G \rangle$ and generated using a *msg* octet-string via the *Elligator 2* *hash-to-curve* algorithm described by section 6.8.2 of RFC9380 [4].

Refer to [Bandersnatch Cipher Suite] for configuration details.

2.2. VRF Output

A point in G generated using VRF input point as: $Output \leftarrow sk \cdot Input$.

2.3. VRF Hashed Output

A fixed length octet-string generated using VRF output point.

The generation procedure details are specified by the proof-to-hash procedure in section 5.2 of RFC9381 and the output length depends on the used hasher

Refer to [Bandersnatch Cipher Suite] for configuration details.

3. IETF VRF

Definition of a VRF based on the IETF RFC9381 specification.

This VRF faithfully follows the RFC but extends it with the capability to sign additional user data (*ad*) as per our definition of VRF-AD.

In particular, *step 5* of RFC section 5.4.3 is defined as:

```
str = str || ad || challenge_generation_domain_separator_back
```

3.1. Setup

Setup follows from the “*cipher suite*” specification defined by faithfully following the RFC9381 section 5.5 guidelines and naming conventions.

- The EC group $\langle G \rangle$ is the prime subgroup of the Bandersnatch elliptic curve, in Twisted Edwards form, with the finite field and curve parameters as specified in the [neuromancer] standard curves database. For this group, $fLen = qLen = 32$ and $cofactor = 4$.
- The prime subgroup generator G is constructed following *Zcash*’s guidelines: “*The generators of G_1 and G_2 are computed by finding the lexicographically smallest valid x -coordinate, and its lexicographically smallest y -coordinate and scaling it by the cofactor such that the result is not the point at infinity.*”
 - $G.x := 0x29c132cc2c0b34c5743711777bbe42f32b79c022ad998465e1e71866a252ae18$
 - $G.y := 0x2a6c669eda123e0f157d8b50badcd586358cad81eee464605e3167b6cc974166$

- The public key generation primitive is $pk = sk \cdot G$, with sk the secret key scalar and G the group generator. In this ciphersuite, the secret scalar x is equal to the secret key sk .
- `suite_string` = 0x33.
- `cLen` = 32.
- `encode_to_curve_salt` = `pk_string` (i.e. $Encode(pk)$).
- The `ECVRF_nonce_generation` function is specified in Section 5.4.2.1 of RFC9381.
- The `int_to_string` function encodes into the 32 bytes little endian representation.
- The `string_to_int` function decodes from the 32 bytes little endian representation.
- The `point_to_string` function converts a point in $\langle G \rangle$ to an octet string using compressed form. The y coordinate is encoded using `int_to_string` function and the most significant bit of the last octet is used to keep track of the x 's sign. This implies that the point is encoded in 32 bytes.
- The `string_to_point` function tries to decompress the point encoded according to `point_to_string` procedure. This function MUST outputs "INVALID" if the octet string does not decode to a point on the prime subgroup $\langle G \rangle$.
- The hash function `Hash` is SHA-512 as specified in RFC6234, with `hLen` = 64.
- The `ECVRF_encode_to_curve` function (*Elligator2*) is as specified in Section 5.4.1.2, with `h2c_suite_ID_string` = "BANDERSNATCH_XMD:SHA-512_ELL2_RO_". The suite must be interpreted as defined by Section 8.5 of [4] and using the domain separation tag `DST` = "ECVRF_" `h2c_suite_ID_string` `suite_string`.

3.2. Sign

Inputs:

- x : Secret key $\in \mathbb{Z}_r^*$
- I : VRF Input $\in \langle G \rangle$
- ad : Additional data octet-string

Outputs:

- O : VRF Output $\in \langle G \rangle$
- π : Schnorr-like proof $\in (\mathbb{Z}_r^*, \mathbb{Z}_r^*)$

Steps:

1. $O \leftarrow x \cdot I$
2. $Y \leftarrow x \cdot G$
3. $k \leftarrow \text{nonce}(x, I)$
4. $c \leftarrow \text{challenge}(Y, I, O, k \cdot G, k \cdot I, ad)$
5. $s \leftarrow (k + c \cdot x)$
6. $\pi \leftarrow (c, s)$
7. **return** (O, π)

Externals:

- *nonce*: refer to RFC9381 section 5.4.2
- *challenge*: refer to RFC9381 section 5.4.3

3.3. Verify

Inputs:

- Y : Public key $\in \langle G \rangle$
- I : VRF Input $\in \langle G \rangle$
- ad : Additional data octet-string
- O : VRF Output $\in \langle G \rangle$
- π : As defined for *Sign* output.

Outputs:

- True if proof is valid, False otherwise.

Steps:

1. $(c, s) \leftarrow \pi$
2. $U \leftarrow s \cdot K - c \cdot Y$
3. $V \leftarrow s \cdot H - c \cdot O$
4. $c' \leftarrow \text{challenge}(Y, I, O, U, V, ad)$
5. **if** $c \neq c'$ **then return** False
6. **return** True

Externals:

- *challenge*: as defined for *Sign*

4. Pedersen VRF

Pedersen VRF resembles IETF VRF but replaces the public key by a Pedersen commitment to the secret key, which makes the Pedersen VRF useful in anonymized ring VRFs (see [Pedersen Ring VRF]).

Strictly speaking Pedersen VRF is not a VRF. Instead, it proves that the output has been generated with a secret key associated with a blinded public (instead of public key). The blinded public key is a cryptographic commitment to the public key. And it could be unblinded to prove that the output of the VRF corresponds to the public key of the signer.

4.1. Setup

Pedersen VRF is initiated for prime subgroup $\langle G \rangle$ of an elliptic curve E with *blinding base* $B \in \langle G \rangle$ defined as:

- $B.x := 0x2039d9bf2ecb2d4433182d4a940ec78d34f9d19ec0d875703d4d04a168ec241e$
- $B.y := 0x54fa7fd5193611992188139d20221028bf03ee23202d9706a46f12b3f3605faa$

In twisted Edwards coordinates.

For all the other configurable parameters and external functions we'll pick as much as possible from the [Bandersnatch Cipher Suite] specification for IETF VRF.

4.2. Sign

Inputs:

- x : Secret key $\in \mathbb{Z}_r^*$
- I : VRF Input $\in \langle G \rangle$
- ad : Additional data octet-string

Output:

- O : VRF Output $\in \langle G \rangle$
- π : Pedersen proof $\in (\langle G \rangle, \langle G \rangle, \langle G \rangle, \mathbb{Z}_r^*, \mathbb{Z}_r^*)$

Steps:

1. $O \leftarrow x \cdot I$
2. $(b, k, k_b) \leftarrow random()$
3. $Y_c \leftarrow x \cdot G + b \cdot B$
4. $KB_r \leftarrow k \cdot G + k_b \cdot B$
5. $O_k \leftarrow k \cdot I$

6. $c \leftarrow \text{challenge}(Y_c, KB_r, O_k, ad)$
7. $s \leftarrow k + c \cdot x$
8. $s_b \leftarrow k_b + c \cdot b$
9. $\pi \leftarrow (Y_c, KB_r, O_k, s, s_b)$
10. **return** (O, π)

Externals:

- *challenge*: see [Challenge] section
- *random*: generates random scalars in \mathbb{Z}_r^*

4.3. Verify

Inputs:

- I : VRF Input $\in \langle G \rangle$.
- O : VRF Output $\in \langle G \rangle$.
- ad : Additional data octet-string
- π : Pedersen proof as defined for *Sign*.

Output:

- True if proof is valid, False otherwise.

Steps:

1. $(Y_c, KB_r, O_k, s, s_b) \leftarrow \pi$
2. $c \leftarrow \text{challenge}(Y_c, KB_r, O_k, ad)$
3. $z_1 \leftarrow O_k + c \cdot O - I \cdot s$
4. **if** $z_1 \neq O$ **then return** False
5. $z_2 \leftarrow KB_r + c \cdot Y_c - s \cdot G - s_b \cdot B$
6. **if** $z_2 \neq O$ **then return** False
7. **return** True

Externals:

- *challenge*: see [Challenge] section

4.4. Challenge

Defined similarly to the challenge procedure specified by section 5.4.3 of RFC9381.

Inputs:

- *Points*: Sequence of points $\in \langle G \rangle$.
- *ad*: Additional data octet-string

Output:

- *c*: Challenge $\in \mathbb{Z}_r^*$.

Steps:

1. *str* = "pedersen_vrf" (ASCII encoded octet-string)
2. **for** *P* **in** *Points*: *str* = *str* || *PointToString*(*P*)
3. *str* = *str* || *ad* || 0x00
4. *h* = *Sha512*(*str*)
5. *h_t* = *h*[0] || .. || *h*[31]
6. *c* = *StringToInt*(*h_t*)
7. **return** *c*

With *PointToString* and *StringToInt* defined as `point_to_string` and `string_to_int` from RFC9381 respectively.

5. Pedersen Ring VRF

Anonymized ring VRFs based of [Pedersen VRF] and ...

5.1. Setup

Setup for plain [Pedersen VRF] applies.

TODO: - SRS for zk-SNARK definition - All the details

5.2. Sign

Inputs:

- *x*: Secret key $\in \mathbb{Z}_r^*$.
- *I*: VRF Input $\in \langle G \rangle$.
- *ad*: Additional data octet-string
- *P*: Ring prover key

Output:

- *O*: VRF Output $\in \langle G \rangle$.
- π_p : Pedersen proof as specified in [Pedersen VRF].
- π_r : Ring proof as specified in Sergey

Steps:

1. $(O, \pi_p) \leftarrow \text{Pedersen.Sign}(x, I, ad)$
2. $\pi_r \leftarrow \text{Ring.Prove}(P, \dots)$ (TODO)

5.3. Verify

Inputs:

- I : VRF Input $\in \langle G \rangle$.
- O : VRF Output $\in \langle G \rangle$.
- ad : Additional data octet-string
- V : ring verifier key $\in ?$
- π_p : Pedersen proof as defined in Pedersen VRF.
- π_r : Ring proof as defined in Sergey

Output:

- True if proof is valid, False otherwise.

Steps:

- 1. $r = \text{Pedersen.Verify}(I, O, ad, \pi_p)$
- 1. **if** $r \neq \text{True}$ **return** False
- 1. $r = \text{Ring.Verify}(V, \pi_r, \dots)$ (TODO)
- 1. **if** $r \neq \text{True}$ **return** False
- 1. **return** True

6. References

1.

Internet Engineering Task Force (2023) Verifiable Random Functions. RFC Editor

2.

Burdges J, Ciobotaru O, Alper HK, et al (2023) Ring verifiable random functions and zero-knowledge continuations

3.

Masson S, Sanso A, Zhang Z (2021) Bandersnatch: A fast elliptic curve built over the BLS12-381 scalar field

4.

Internet Engineering Task Force (2023) Hashing to Elliptic Curves. RFC Editor