Bandersnatch VRF-AD Specification

Davide Galassi Seyed Hosseini

21 June 2024 - Draft 6

Abstract

This specification delineates the framework for a Verifiable Random Function with Additional Data (VRF-AD), a cryptographic construct that augments a standard VRF by incorporating auxiliary information into its signature. We're going to first provide a specification to extend IETF's ECVRF as outlined in RFC-9381 [1], then we describe a variant of the Pedersen VRF originally introduced by BCHSV23 [2], which serves as a fundamental component for implementing anonymized ring signatures as further elaborated by Vasilyev [3]. This specification provides detailed insights into the usage of these primitives with Bandersnatch, an elliptic curve constructed over the BLS12-381 scalar field specified in MSZ21 [4].

1. Preliminaries

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

- $Prove(sk, in, ad) \mapsto (out, \pi)$: from secret key sk, input in, and additional data ad returns a verifiable output out and proof π.
- $Verify(pk, in, ad, out, \pi) \mapsto (0|1)$: for public key pk, input in, additional data ad, output out and proof π returns either 1 on success or 0 on failure.

1.1. VRF Input

An arbitrary length octet-string provided by the user and used to generate some unbiasable verifiable random output.

1.2. VRF Input Point

A point in $\langle G \rangle$ generated from VRF input octet-string using the *Elligator 2 hash-to-curve* algorithm as described by section 6.8.2 of RFC-9380 [5].

1.3. VRF Output Point

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

1.4. VRF Output

A fixed length octet-string generated from VRF output point using the proof-to-hash procedure defined in section 5.2 of RFC-9381.

1.5 Additional Data

An arbitrary length octet-string provided by the user to be signed together with the generated VRF output. This data doesn't influence the produced VRF output.

2. IETF VRF

Based on IETF RFC-9381 which is extended with the capability to sign additional user data (`ad`).

2.1. Configuration

Configuration is given by following the *"cipher suite"* guidelines defined in section 5.5 of RFC-9381.

- `suite_string` = `"Bandersnatch_SHA-512_ELL2"`.
- The EC group $\langle G \rangle$ is the prime subgroup of the Bandersnatch elliptic curve, in Twisted Edwards form, with finite field and curve parameters as specified in MSZ21. For this group, `fLen` = `qLen` = 32 and `cofactor` = 4.
- The prime subgroup generator $G \in \langle G \rangle$ is defined as follows:

$$G.x = 0x29c132cc2c0b34c5743711777bbe42f32b79c022ad998465e1e71866a252ae18$$

$$G.y = 0x2a6c669eda123e0f157d8b50badcd586358cad81eee464605e3167b6cc974166$$

- `cLen` = 32.
- The public key generation primitive is $pk = sk \cdot G$, with sk the secret key scalar and G the group generator. In this cipher suite, the secret scalar `x` is equal to the secret key `sk`.
- `encode_to_curve_salt` = `pk_string` (i.e. `point_to_string(pk)`).
- The `ECVRF_nonce_generation` function is specified in section 5.4.2.1 of RFC-9381.
- 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 eventually reducing modulo the prime field order.
- 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 x sign. This implies that `ptLen = flen = 32`.
- The `string_to_point` function converts an octet-string to a point on E. The string most significant bit is removed to recover the x coordinate as function of y, which is first decoded from the rest of the string using `int_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 RFC-6234 [6], with `hLen` = 64.
- The `ECVRF_encode_to_curve` function uses *Elligator2* method described in section 6.8.2 of RFC-9380 and is described in section 5.4.1.2 of RFC-9381, with `h2c_suite_ID_string` = `"Bandersnatch_XMD:SHA-512_ELL2_RO_"` and domain separation tag `DST = "ECVRF_"  h2c_suite_ID_string  suite_string`.

2.2. Prove

Input:

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

Outputs:

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

Steps:

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

Externals:

- *nonce*: refer to section 5.4.2.1 of RFC-9381.
- *challenge*: refer to section 5.4.3 of RFC-9381 and section 2.4 of this specification.

2.3. Verify

Inputs:

- $Y \in \langle G \rangle$: Public key
- $I \in \langle G \rangle$: VRF input point
- *ad*: Additional data octet-string
- $O \in \langle G \rangle$: VRF output point
- $\pi \in (\mathbb{Z}_r^*, \mathbb{Z}_r^*)$: Schnorr-like proof

Outputs:

- True if proof is valid, False otherwise.

Steps:

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

Externals:

- *challenge*: as defined for *Sign*

2.3.1 Validity Argument

TODO

2.4. Challenge

Challenge construction mostly follows the procedure given in section 5.4.3 of RFC-9381 [1] with some tweaks to add additional data.

Input:

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

Output:

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

Steps:

```
1. $str = \texttt{suite\_string} \parallel \texttt{0x02}$
2. **for each** $P$ **in** $Points$: $str = str \parallel \texttt{point\_to\_string}(P)$$
3. $str = str \| ad \| 0x00$
4. $h = \texttt{hash}(str)$
5. $h_t = h[0] \| .. \| h[cLen - 1]$
6. $c = \texttt{string\_to\_int}(h_t)$
7. **return** $c$
```

With `point_to_string`, `string_to_int` and `hash` as defined in section 2.1.

3. Pedersen VRF

Pedersen VRF resembles IETF EC-VRF but replaces the public key with a Pedersen commitment to the secret key, which makes this VRF useful in anonymized ring proofs.

The scheme proves that the output has been generated with a secret key associated with a blinded public key (instead of the public key). The blinded public key is a cryptographic commitment to the public key, and it can be unblinded to prove that the output of the VRF corresponds to the public key of the signer.

This specification mostly follows the design proposed by BCHSV23 [2] in section 4 with some details about blinding base point value and challenge generation procedure.

3.1. Configuration

Pedersen VRF is initiated for prime subgroup $\langle G \rangle$ of Bandersnatch elliptic curve E defined in MSZ21 [4] with *blinding base* $B \in \langle G \rangle$ defined as follows:

$$B.x = 0x2039d9bf2ecb2d4433182d4a940ec78d34f9d19ec0d875703d4d04a168ec241e$$

$$B.y = 0x54fa7fd5193611992188139d20221028bf03ee23202d9706a46f12b3f3605faa$$

For all the other configurable parameters and external functions we adhere as much as possible to the Bandersnatch cipher suite for IETF VRF described in section 2.1 of this specification.

3.2. Prove

Inputs:

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

Output:

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

Steps:

1. $O \leftarrow x \cdot I$
2. $(k, k_b) \leftarrow random()$
3. $\bar{Y} \leftarrow x \cdot G + b \cdot B$
4. $R \leftarrow k \cdot G + k_b \cdot B$
5. $O_k \leftarrow k \cdot I$
6. $c \leftarrow challenge(\bar{Y}, I, O, R, O_k, ad)$
7. $s \leftarrow k + c \cdot x$
8. $s_b \leftarrow k_b + c \cdot b$
9. $\pi \leftarrow (\bar{Y}, R, O_k, s, s_b)$
10. **return** (O, π)

Externals:

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

3.3. 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. $(\bar{Y}, R, O_k, s, s_b) \leftarrow \pi$
2. $c \leftarrow challenge(\bar{Y}, I, O, R, O_k, ad)$
3. $z_1 \leftarrow O_k + c \cdot O - I \cdot s$
4. **if** $z_1 \neq 0$ **then return** False
5. $z_2 \leftarrow R + c \cdot \bar{Y} - s \cdot G - s_b \cdot B$
6. **if** $z_2 \neq 0$ **then return** False
7. **return** True

Externals:

- *challenge*: see [Challenge] section

4. Ring VRF

Anonymized ring VRFs based of [Pedersen VRF] and …

4.1. Setup

Setup for plain [Pedersen VRF] applies.

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

4.2. Sign

Inputs:

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

Output:

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

Steps:

1. $(O, \pi_p) \leftarrow Pedersen.Sign(x, I, ad)$

2. $\pi_r \leftarrow Ring.Prove(P, ...)$ (TODO)

4.3. Verify

Inputs:

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

Output:

- True if proof is valid, False otherwise.

Steps:

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

5. References

1.

Internet Engineering Task Force *Verifiable Random Functions*; RFC Editor, 2023;

2.

Burdges, J.; Ciobotaru, O.; Alper, H.K.; Stewart, A.; Vasilyev, S. Ring Verifiable Random Functions and Zero-Knowledge Continuations 2023.

3.

Vasilyev, S. Ring Proof Technical Specification 2024.

4.

Masson, S.; Sanso, A.; Zhang, Z. Bandersnatch: A Fast Elliptic Curve Built over the Bls12-381 Scalar Field 2021.

5.

Internet Engineering Task Force *Hashing to Elliptic Curves*; RFC Editor, 2023;

6.

Internet Engineering Task Force *US Secure Hash Algorithms*; RFC Editor, 2011;