Bandersnatch VRF-AD Specification

Seyed Hosseini Davide Galassi

Draft 2 - 04-04-2024

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 $perp$.

Definition: For an elliptic curve E defined over finite field F with large prime subgroup G with order r generated by the base point B, 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.

Preliminaries

VRF Input

A point on E and generated using msg octet-string via the *Elligator 2* hash-to-curve algorithm described by section 6.8.2 of RFC9380.

The algorithm yields a point in G.

VRF Output

A point generated using VRF input point as:

$$Output \leftarrow sk \cdot Input$$

VRF Hashed Output

An 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 (see Bandersnatch Cipher Suite for details).

IETF VRF

Definition of a VRF based on the IETF RFC9381.

All the details specified by the RFC applies with the additional capability to sign additional data (`ad`) as per definition of VRF-AD we've given.

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

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

Setup

IETF VRF is initiated for prime subgroup G of an elliptic curve E with $K \in G$ defined to be *key base* respectively.

Sign

$$Sign(sk, input, ad) \mapsto (output, proof)$$

Inputs:

- sk: secret key $\in \mathbb{Z}_r^*$.
- $input$: $Input \in G$.
- ad: Additional data octet-string.

Outputs:

- $preout$: $PreOutput \in G$
- $proof$: A *Schnorr-like* proof $\in (\mathbb{Z}_r^*, \mathbb{Z}_r^*)$

Steps:

1. $preout = sk \cdot input$
2. $k \leftarrow Nonce(sk, input)$ (see RFC9381 section 5.4.2)
3. $c \leftarrow Challenge(Y, H, preout, k \cdot B, k \cdot input)$ (see RFC9381 section 5.4.3)
4. $s \leftarrow (k + c \cdot x)$
5. $proof \leftarrow (c, s)$

6. **return** $(preout, proof)$

Verify

Inputs:

- pk: verification key corresponds to $sk\ inG$.
- $input$: $Input \in G$.
- ad: Additional data octet-string
- $output$: $Output \in G$.
- $proof$: as defined for $Sign$ output.

Output:

- True if proof is valid, False otherwise.

Steps:

1. $U = s \cdot K - c \cdot pk$
2. $V = s \cdot H - c \cdot preout$
3. $c' = Challenge(pk, input, output, U, V)$ (see RFC9381 Section 5.4.3)
4. **if** $c \neq c'$ **then return** False
5. **return** True

Bandersnatch Cipher Suite

Suite specification follows RFC9381 section 5.5 guidelines.

- The EC group G is 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 G1 and G2 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`.
- The `ECVRF_nonce_generation` function is as 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.
- The point_to_string function converts a point on E 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 on 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 curve E.
- 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 RFC9380 and using the domain separation tag `DST = "ECVRF_"  h2c_suite_ID_string  suite_string`.

Pedersen VRF

Pedersen VRF resembles EC 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 Pederson 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 unblinded to prove that the output of the VRF is corresponds to the public key of the signer.

Setup

PedersenVRF is initiated for prime subgroup G of an elliptic curve E with $K, B \in G$ defined to be *key base* and *blinding base* respectively.

- K is set equal to point g defined in Bandersnatch Cipher Suite.
- B is defined as:

  - $b.x =$ `0x2039d9bf2ecb2d4433182d4a940ec78d34f9d19ec0d875703d4d04a168ec241e`
  - $b.y =$ `0x54fa7fd5193611992188139d20221028bf03ee23202d9706a46f12b3f3605faa`

In twisted Edwards coordinates.

Sign

Inputs:

- sk: VRF secret key $\in F$.
- sb: Blinding factor $\in F$ (can be random)
- $input$: $VRFInput \in G$.
- ad: Additional data octet-string

Output:

- A quintuple $(preout, compk, KBrand, PORand, ks, bs)$ corresponding to Pedersen VRF signature.

Steps:

1. $preout = sk \cdot input$
2. $krand \leftarrow RandomElement(F)$
3. $brand \leftarrow RandomElement(F)$
4. $KBrand \leftarrow krand \cdot G + brand \cdot B$
5. $POrand \leftarrow krand \cdot input$
6. $compk = sk \cdot K + sb \cdot B$
7. $c \rightarrow Challenge(compk, KBrand, POrand, ad)$
8. $ks \rightarrow krand + c \cdot sk$
9. $bs \rightarrow brand + c \cdot sb$
10. **return** $(preout, compk, KBrand, PORand, ks, bs)$

Verify

Inputs:

- pk: VRF verification key corresponds to $sk \in G$.
- $input$: $VRFInput \in G$.
- $preout$: $VRFPreOutput \in G$.
- ad: Additional data octet-string
- $(compk, KBrand, PORand, ks, bs)$ proof yielded by $Sign$.

Output:

- True if proof is valid, False otherwise.

Steps:

1. $c \rightarrow Challenge(compk, KBrand, POrand, ad)$
2. $z1 \leftarrow POrand + c \cdot preout - input \cdot ks$
3. $z1 \leftarrow ClearCofactor(z1)$
4. **if** $z1 \neq O$ **then return** False
5. $z2 \leftarrow KBrand + c \cdot compk - krand \cdot K - brand \cdot B$
6. $z2 \leftarrow ClearCofactor(z2)$
7. **if** $z2 \neq O$ **then return** False
8. **return** True

NOTE: I don't think step 3 and 6 are necessary, we're working in the prime subgroup.

Challenge

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

Inputs:

- *points*: Sequence of points to include in the challenge.
- *ad*: Additional data octet-string

Output:

- Scalar F.

Steps:

1. str = `"pedersen_vrf"`
2. **for** p **in** *points*: $str = str \| PointToString(obj)$
3. $str = str \| ad \| 0x00$
4. $h = Sha512(str)$
5. $ht = h[0]..h[31]$
6. $c = StringToInt(ht)$
7. **return** c

With *PointToString* and *StringToInto* defined as `point_to_string` and `string_to_int` in RFC9381 with configuration specified in the IETF VRF section of this document.

Pedersen Ring VRF

Anonymized ring VRFs based of Pedersen VRF and ...

Setup

Setup for plain Pedersen VRF applies.

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

Sign

Inputs:

- sk: VRF secret key.
- sb: Blinding factor $\in F$ (can be random)
- $input$: $VRFInput \in G$.
- ad: Additional data octet-string
- P: ring prover key

Output:

- $(preout, pedersen - proof, zk - proof)$

Steps:

1. $(preout, pedersen - proof) \leftarrow PedersenSign(sk, sb, input, ad)$
2. ring-proof: TODO

Verify

Inputs:

- pk: VRF verification key corresponds to $sk \in G$.
- $input$: $VRFInput \in G$.
- $preout$: $VRFPreOutput \in G$.
- ad: Additional data octet-string
- P: ring prover key
- $(pproof, rproof)$ proofs yielded by $Sign$.

Output:

- True if proof is valid, False otherwise.

Steps:

- PedersenVerify(pk, input, preout, ad, pedersen-proof)

- verify ring-proof

References

TODO