Bandersnatch VRFs

Introduction

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

- $VRF.KeyGen : () \mapsto (pk, sk)$ where pk is a public key and sk is its corresponding secret key.
- $VRF.Sign : (sk, msg, ad) \mapsto \pi$ takes a secret key sk, an input msg, and additional data ad, and returns a VRF signature π.
- $VRF.Eval : (sk, msg) \mapsto Out$ takes a secret key sk and an input msg, and returns a VRF output out.
- $VRF.Verify : (pk, msg, aux, \pi) \mapsto (out|prep)$ for a public key pk, an input msg, and additional data ad, and then returns either an output out or else failure $perp$.

Definition: For an elliptic curve E defined over finite field F with large prime subgroup G generated by 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.

For input msg and ad additional data first we compute the $VRFInput$ which is a point on elliptic curve E as follows:

$$t \leftarrow Transcript(msg)$$

$$VRFInput := H2C(challange(t, "vrf-input")$$

Where: - *transcript* function is described in [[ark-transcript]] section. - $H2C : B \rightarrow G$ is a hash to curve function correspond to curve E specified in Section [[hash-to-curve]] for the specific choice of E

VRF Input

The VRF Input is a point on the elliptic curve E and generated as output of the Elligator 2 hash-to-curve algorithm as described by section 6.8.2 of RFC9380. The algorithm yields a point which is inside the prime order subgroup of E.

VRF Preoutput and Output

Definition: *VRF pre-output* is generated using VRF input point as:

$$PreOutput \leftarrow sk \cdot VrfInput$$

Definition: *VRF output* is generated using *VRF pre-output* point as:

$$VrfOutput \leftarrow Hash("vrfoutput", Encode(PreOutput))$$

IETF VRF

Definition of a VRF based on the IETF RFC-9381.

All the details specified by the RFC applies with the additional capability to add additional data (`ad`) as per definition of EC-VRF we've given. In particular the step 5 of section 5.4.3 is defined as:

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

Bandersnatch Cipher Suite Configuration

Configuration follows the RFC-9381 suite specification 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 · 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.

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 commitement 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.

Sign

Inputs:

- sk: VRF secret key.
- sb: Blinding factor $\in F$
- $Input$: $VRFInput \in G$.
- 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. $c \rightarrow Challenge("pedersen", compk, KBrand, POrand, ad)$
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.
- $input$: $VRFInput$.
- $preout$: $VRFPreOutput$.
- ad: Additional data octet-string
- $(compk, KBrand, PORand, ks, bs)$ the output of Pedersen VRF Sign.

Output:

- True if Pedersen VRF is valid False otherwise.

Steps:

1. $c \rightarrow Challenge("pedersen", compk, KBrand, POrand, ad)$
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.