VRF Specification

Ann Weine

June 30, 2017

1 Introduction

A Verifiable Random function (VRF) is the public-key version of a keyed cryptographic hash. Only the person who has a private key could generate a hash, while all the owners of corresponding public key could verify its correctness.
For our purposes we use the VRF over P256 Curve and SHA256 as the hash function.
A VRF consists of two different functions. The first one ($ECVRF - prove$) is used to generate a hash, the second function ($ECVRF - verify$) is used for the verification of the provided proof.

A prover generates a proof using function VRF-Proof.
A verifier is given the proof. The verifier computes $hash'$ using the given proof and public key. The hash is accepted as valid if and only if the computed $hash'$ is equal to the hash given by the prover.

1.1 Used crypto primitives

The function is built based on the NIST OpenSSL Elliptic Curve and uses the following operations over the curve:

- EC point multiplication
- EC point addition
- Check whether the specified point belongs to the curve

The function is using SHA256 as a hash function.

1.2 Other used primitives

The following primitives are used to make a casting between data types.

- OS2ECP/ECP2OS.
  The first procedure takes as an input a bytes representation of a point and returns the corresponding EC point :

  ```
  val _OS2ECP : bytes -> Tot(serialized_point)
  ```

  The second procedure does the opposite algorithm. It takes a EC point and returns the corresponding bytes representation of the point:

  ```
  val _ECP2OS : gamma: serialized_point -> Tot(r: bytes)
  ```

  The algorithms are implemented according to the specification described in Standards for Efficient Cryptography Group (SECG).

- I2OSP/OS2IP.
  The first procedure takes as an input an integer number and returns the same integer as in byte representation. The second procedure does the opposite algorithm.

  ```
  val _I2OSP: value1: int -> n: int{n > 0} -> Tot(r: bytes{Seq.length r = n})
  val _OS2IP: s: bytes{Seq.length s > 0} -> int
  ```

1.3 Proof generation function

The proof generation function ($ECVRF - prove$) takes

- input of type bytes
- pair of public/private key of type bytes each

and returns the proof of type bytes that is used to verify the correctness of computed hash.
Steps:

- $h = ECVRF - hash - to - curve(input, g^x)$, where g^x is a public key and $ECVRF - hash - to - curve$ is a subroutine used to convert an input to a point of the curve
- $gamma = h^x$
- $k = random(0(q - 1))$, where q is the prime order of the EC group
- $c = ECVRF - hash - points(g, h, publickey, gamma, g^k, h^k)$, where $ECVRF - hash - points$ is a subroutine used to convert points to hash value (SHA256 hash function is used)
- $c = k - c * qmodq$
- $pi = ECP2OS(gamma)||I2OSP(c, n)||I2OSP(s, 2n)$
- return pi

```
val _ECVRF_prove:
        input: bytes {Seq.length input < pow2 61 - (op_Multiply 2 n) - 5 } ->
        public_key: serialized_point -> private_key : bytes ->
        generator : serialized_point->
        Tot(proof: option bytes {Some?proof ==> Seq.length
        (Some?.v proof) = (op_Multiply 5 n) + 1})
```

1.4 Hash generation function

The hash generation function ($_ECVRF_proof2hash$) takes a proof as an input and returns the corresponding hash.

```
val _ECVRF_proof2hash: pi: bytes{Seq.length pi = op_Multiply 5 n + 1} ->
Tot(hash: bytes)
```

1.5 Proof verification function

The proof verification function ($ECVRF - verify$) takes

- public key as bytes
- proof as bytes
- input as bytes

and returns the result whether the proof is valid or not.
Steps:

- $gamma, c, s = ECVRF - decode - proof(pi)$
- $ifnotisValidPointgammathenreturnfalseelse$
- $u = (g^x)^c * g^s$
- $h = ECVRF - hash - to - curve(alpha, g^x)$, where $ECVRF - hash - to - curve$ is a subroutine used to convert an input to a point of the curve
- $v = gamma^c * h^s$

- $c' = ECVRF - hash - points(g, h, g^x, gamma, u, v)$, where $ECVRF - hash - points$ is a subroutine used to convert points to hash value (SHA256 hash function is used)
- return $c == c'$

```
val _ECVRF_verify : generator: serialized_point ->
        public_key : serialized_point ->
        pi: bytes {Seq.length pi = op_Multiply 5 n +1} ->
        input : bytes ->
        Tot(bool)
```