cfrg@IETF'102
Montreal, July 2018

draft-irft-cfrg-vrf-02

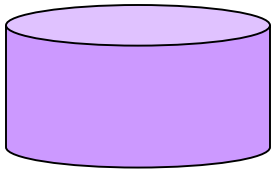
Verifiable Random Functions (VRF)

update on changes + some questions

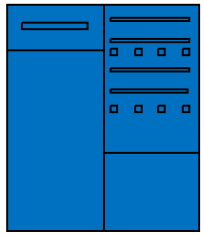
Sharon Goldberg (Boston University)
Leonid Reyzin (Boston University)
Dimitrios Papadopoulos (HKUST)
Jan Vcelak (ns1)

VRF: verifiable random function

Verifier **pk**



Hasher **sk**



input →

proof = **prove(sk**, input**)**

← proof

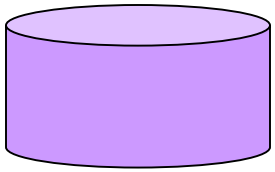
If **verify (pk**, input, proof**)**

hash = **proof2hash(**proof**)**

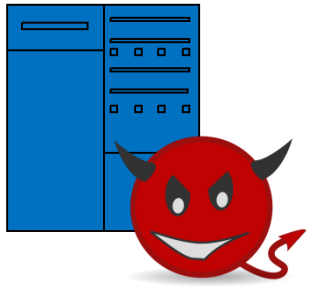
Else INVALID

VRF: verifiable random function

Verifier **pk**



Hasher **sk**



input →

proof = **prove(sk**, input**)**

← proof

If **verify (pk**, input, proof**)**

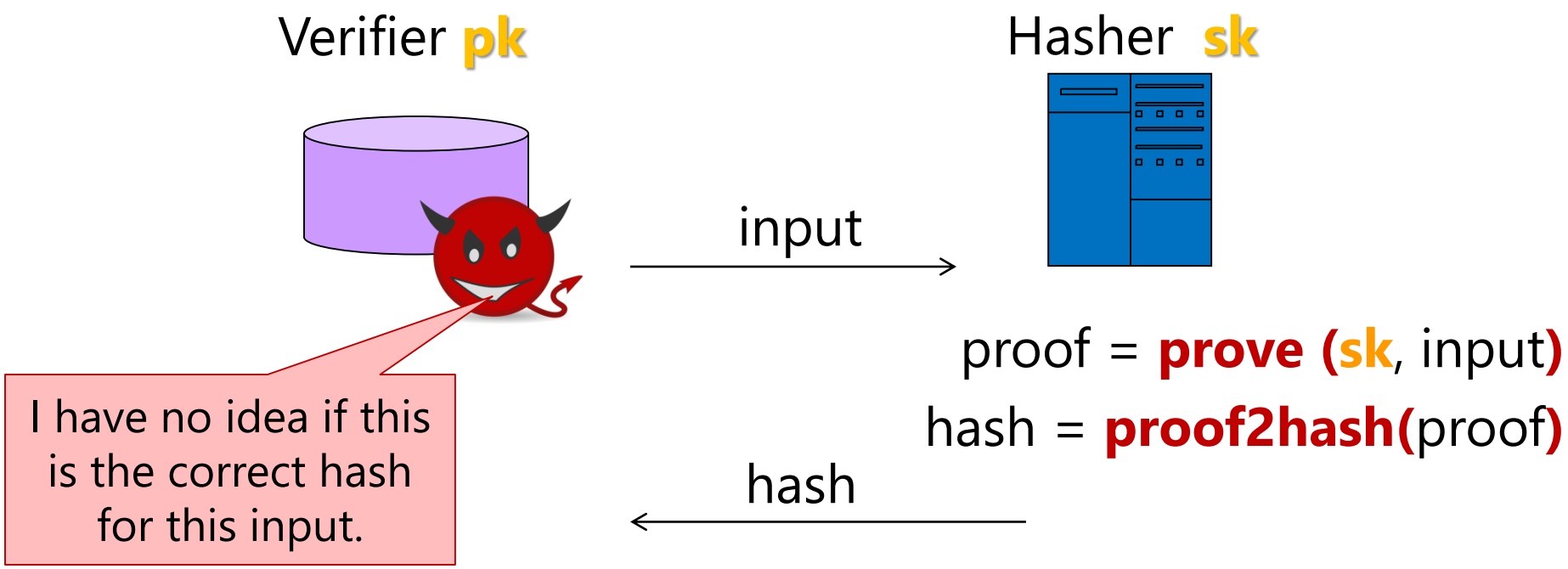
hash = **proof2hash(**proof**)**

Else INVALID

properties:

1. **uniqueness:** 1-to-1 relationship between input and hash.
2. **collision resistance:** hard to find two inputs with same hash

VRF: verifiable random function

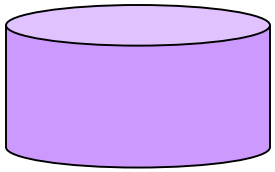


properties:

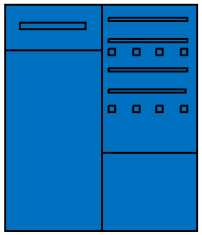
1. **uniqueness:** 1-to-1 relationship between input and hash.
2. **collision resistance:** hard to find two inputs with same hash
3. **pseudorandomness:** only hasher can compute hash from input

EC-VRF (elliptic curve VRF)

Verifier g^x



Hasher $sk=(x,z)$



input →

$\mathbf{h}$ = hash_to_curve (suite, $\mathbf{g^x}$, input)

nonce $\mathbf{k}$ = hash(…, $\mathbf{z}$, $\mathbf{h}$)

$\mathbf{c} = \mathbf{H}(\mathbf{h}, \mathbf{h^x}, \mathbf{g^k}, \mathbf{h^k})$

$\mathbf{s} = \mathbf{k} + \mathbf{cx} \bmod \mathbf{q}$

← proof: $(\mathbf{h^x}, \mathbf{c}, \mathbf{s})$

$\mathbf{u} = \mathbf{g^s} / (\mathbf{g^x})^{\mathbf{c}}$

$\mathbf{h}$ = hash_to_curve (input)

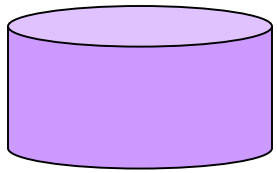
$\mathbf{v} = \mathbf{h^s} / (\mathbf{h^x})^{\mathbf{c}}$

If $\mathbf{c} = \mathbf{H}(\mathbf{h}, \mathbf{h^x}, \mathbf{u}, \mathbf{v})$

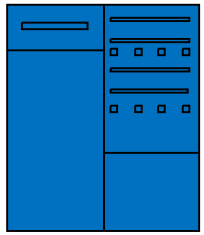
return hash($\mathbf{h^x}$)

Else return INVALID

EC-VRF features



Verifier g^x



Hasher $sk=(x,z)$

input →

h = hash_to_curve (suite, $\mathbf{g^x}$, input)

nonce **k** = hash(…, **z, h**)

c = **H(h**, $\mathbf{h^x}$, $\mathbf{g^k}$, $\mathbf{h^k}$ **)**

s = **k + cx mod q**

NEW Future proofing!

Short! Just 128 bits!

← proof: ($\mathbf{h^x}$, **c**, **s**)

u = $\mathbf{g^s / (g^x)^c}$

h = hash_to_curve (input)

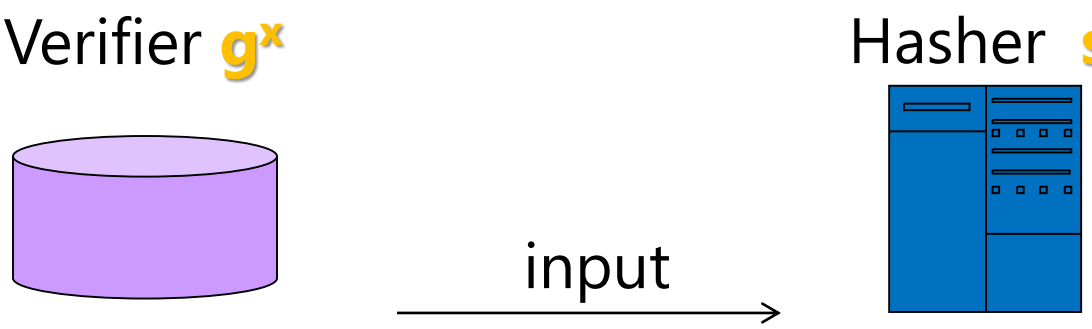
v = $\mathbf{h^s / \gamma^c}$

If **c** = **H(h**, **γ**, **u**, **v)**

return hash(**γ**)

Else return INVALID

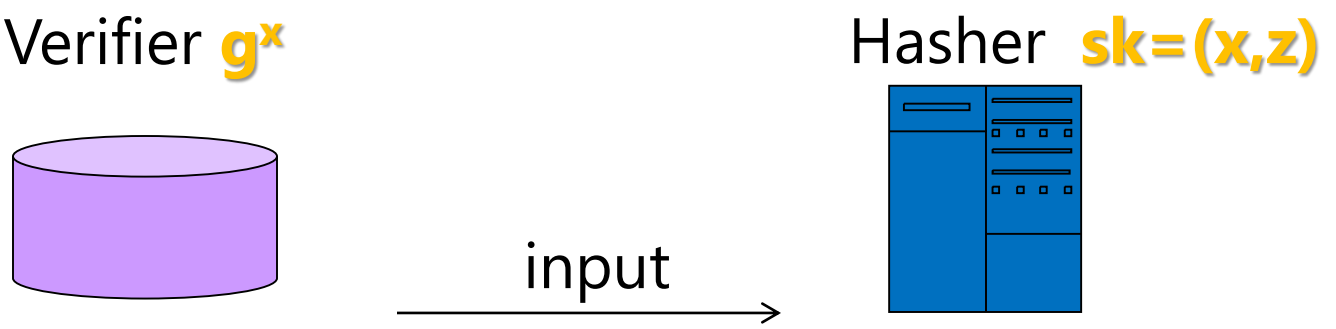
Ciphersuite EC-VRF-P256-SHA256



Ciphersuite choices:

- **Curve:** NIST P-256
- **Hash:** SHA256
- NEW **Nonce:** Deterministic, identical to RFC 6979 [Determist'c ECDSA]
- **Hash-to-curve:** Try-and-increment with SHA256
- **Key generation:** Same as SECG1

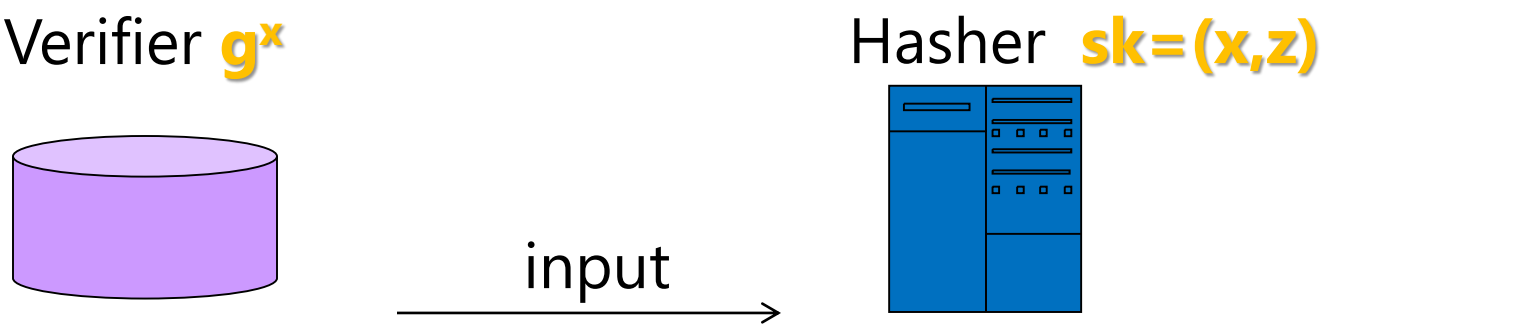
Ciphersuite EC-VRF-ED25519-SHA512



Ciphersuite choices:

- **Curve:** Ed25519
- NEW **Hash:** SHA512
- NEW **Nonce:** Deterministic, based on RFC 8032 [EdDSA]
- **Hash-to-curve:** Try-and-increment with SHA256
- **Key generation:** Same as RFC 8032 [EdDSA]

NEW Ciphersuite EC-VRF-ED25519-SHA512-Elligator2



Ciphersuite choices:

- **Curve:** Ed25519

NEW **Hash:** SHA512

NEW **Nonce:** Deterministic, based on RFC 8032 [EdDSA]

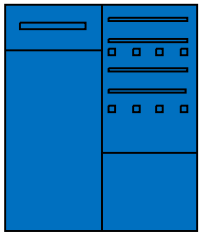
NEW **Hash-to-curve:** Elligator2

- **Key generation:** Same as RFC 8032 [EdDSA]

decision: domain separation strategy

nonce generation is domain-separated because it uses z; for the other three hashes, domain separation via one-octet prefix 0x01, 0x02, or 0x03

Hasher **sk=(x,z)**



h = hash_to_curve (**suite, 1,** $\mathbf{g^x}$, input)

nonce **k** = hash(…, **z,** **h**)

c = **H(suite, 2,** **h**, $\mathbf{h^x}$, $\mathbf{g^k}$, $\mathbf{h^k}$ **)**

s = **k + cx mod q**

proof: ($\mathbf{h^x}$, **c**, **s**)

$\mathbf{u = g^s / (g^x)^c}$

h = hash_to_curve (input)

$\mathbf{v = h^s / \gamma^c}$

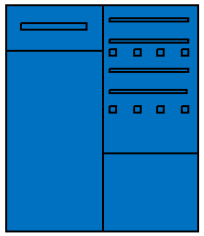
If **c** = **H(suite, 3,** **h**, **γ**, **u**, **v** **)**

return hash(**γ**)

Else return INVALID

decision: no prehash ciphersuites

Current design



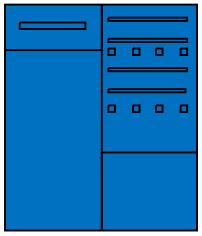
h = hash_to_curve (… , input)

nonce **k** = hash(…,z, **h**)

c = **H(h**, $\mathbf{h^x}$, $\mathbf{g^k}$, $\mathbf{h^k}$ **)**

s = **k + cx mod q**

Possible prehash suite



prehash = prehash (input)

h = hash_to_curve (…, **prehash**)

nonce **k =** hash(…. , z, **prehash**)

c = **H(prehash**, $\mathbf{h^x}$, $\mathbf{g^k}$, $\mathbf{h^k}$ **)**

s = **k + cx mod q**

our claim: hash_to_curve already acts like a prehash!

seeking feedback: ciphersuites

Specified ciphersuites:

1. EC-VRF-P256-SHA256
2. EC-VRF-ED25519-SHA512

} non-constant time

3. EC-VRF-ED25519-SHA512-Elligator NEW

Q: Do we need all three ciphersuites?

We could easily kill #2.

seeking feedback: ED25519-SHA512-x nonce gen

Q: Do we "copy" ED25519 nonce generation from RFC8032?

Nonce generation in RFC 8032 [EdDSA]

nonce = hash("sigEd25519 no Ed25519 collisions", **0**, 0, hash(z), input) mod 2^{255}-19
nonce = hash("sigEd25519 no Ed25519 collisions", **1**, 0, hash(z), prehash(input)) mo

Nonce generation in our draft:

nonce = hash(hash(z), **h**) mod 2^{255}-19

seeking feedback: P256-SHA256 nonce gen

Q: Do we copy P256 nonce generation from RFC6979?

Nonce generation in RFC 6979 uses HMAC_DRBG

pros: already implemented for deterministic ECDSA
cons: needs at least 10 applications of a hash. (slower!)
very small probability of a timing side channel

$K_1 = HMAC_0(1, 0, z, h)$
$V_1 = HMAC_{K1}(1)$
$K_2 = HMAC_{K1}(V_1, 1, z, h)$
$V_2 = HMAC_{K2}(V_1)$
If $V_3 = HMAC_{K2}(V_2) <$ prime, output V_3; else repeat this step.

timing sidechannel

ALTERNATIVE:
Use SHA512 in this suite with ED25519-style nonce gen.

seeking feedback: nits

Q: We use exponential notation. Switch to multiplicative?

h^x **vs** xH

Q: We do not support "contexts". Should we?

h = hash_to_curve (suite, g^x, input)

vs

h = hash_to_curve (suite, g^x, **contextlen, context,** input,)

Q: Take the "first n octets" or the "last n octets" of a hash?
Also: terminology: "First-most" vs "leftmost"?

Q: Do we add domain separation, context to the RSA VRF?
Easy for us to copy from EC VRF.