saag@IETF'98
Chicago, March 2017

draft-goldbe-vrf-00

Verifiable Random Functions (VRF)

Sharon Goldberg (Boston University)
Dimitrios Papadopoulos (University of Maryland)
Jan Vcelak (ns1)

Contributors: Leonid Reyzin (Boston University), Shumon Huque (Salesforce), David C. Lawrence (Akamai), Moni Naor & Asaf Ziv (Weizmann Institute)

hash function zoo

hash function: SHA256 BLAKE

- no key
- hash = **H(**input**)**
- Verify: Check hash = **H(**input**)**

pseudorandom function: HMAC

- symmetric key **k**
- hash = **H(k**, input**)**
- Verify: Cannot without **k**

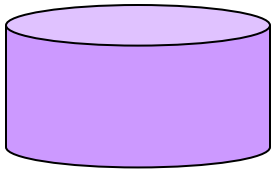
verifiable random function (VRF):

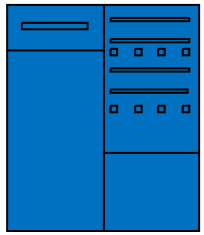
- asymmetric key **(SK**, **PK)**
- hash = **VRF_hash(SK**, input**)**
- Verify: Use **PK**

VRF: verifiable random function

Verifier **PK**

Hasher **SK**





input →

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

← proof

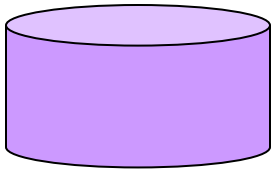
If **verify (PK**, input, proof**)**

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

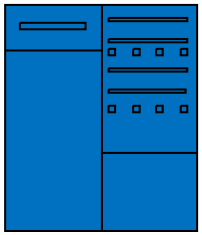
Else INVALID

VRF security: trusted uniqueness

Verifier PK



Hasher SK



input →

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

Like a deterministic digital signature

← proof

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

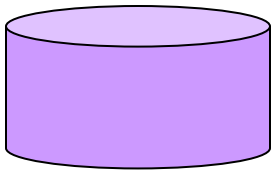
hash = **proof2hash(**proof**)**

Else INVALID

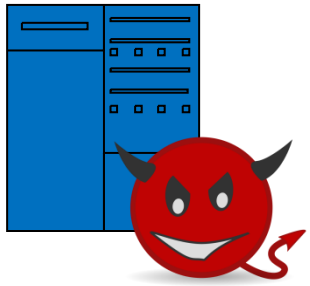
Like a regular hash function (eg SHA256)

1-to-1 relationship between input and hash. (As with SHA-256!)

VRF security: trusted uniqueness



Verifier **PK**



Hasher **SK**

input →

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

← proof

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

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

Else INVALID

Like a deterministic digital signature

Like a regular hash function (eg SHA256)

1-to-1 relationship between input and hash. (As with SHA-256!)

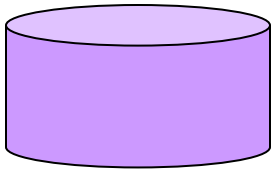
Trusted uniqueness:

Suppose the VRF keys (**PK,SK**) are generated in a trusted way.

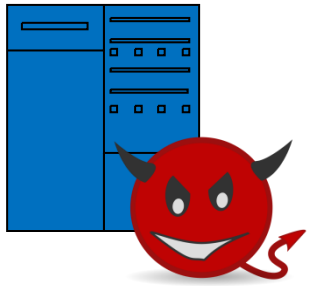
- If **PK** is fixed, then even an adversary that knows **SK** can't find
- …two distinct VRF hash values that are valid for same input

VRF security: trusted collision resistance

Verifier **PK**



Hasher **SK**



input →

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

← proof

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

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

Else INVALID

Like a deterministic digital signature

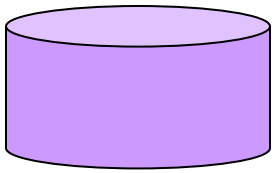
Like a regular hash function (eg SHA256)

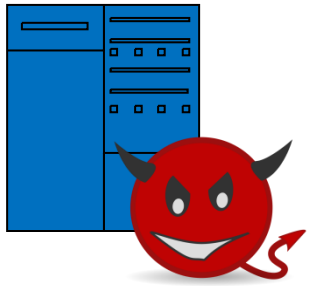
Collision resistance. (As with SHA-256!)

VRF security: trusted collision resistance

Verifier **PK**

Hasher **SK**





input →

← proof

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

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

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

Else INVALID

Like a regular hash function (eg SHA256)

Like a deterministic digital signature

Collision resistance. (As with SHA-256!)

Trusted collision resistance:

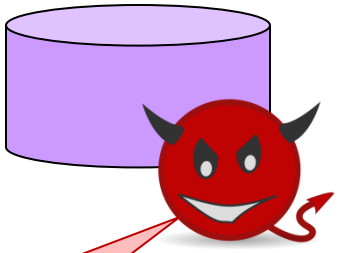
Suppose the VRF keys (**PK,SK**) are generated in a trusted way.

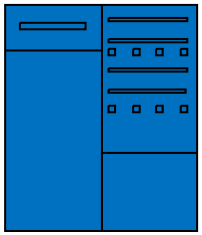
- If **PK** is fixed, then even an adversary that knows **SK** can't find
- …two distinct inputs that have the same valid VRF hash

VRF security: pseudorandomness

Verifier **PK**

Hasher **SK**



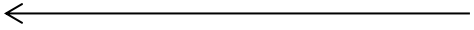


I have no idea what input this hash corresponds to.

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

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

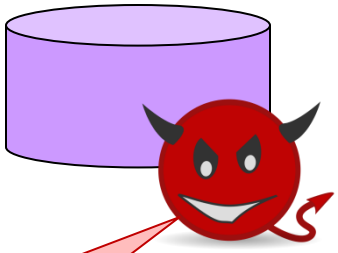
hash



Only the Hasher can compute the hash. (No dictionary attacks!)

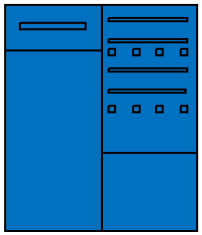
VRF security: pseudorandomness

Verifier **PK**



I have no idea what input this hash corresponds to.

Hasher **SK**



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

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

hash ←

Only the Hasher can compute the hash. (No dictionary attacks!)

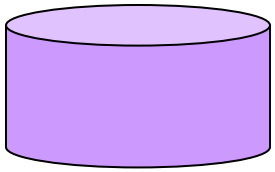
Pseudorandomness:

Suppose the VRF keys (**PK,SK**) are generated in a trusted way.

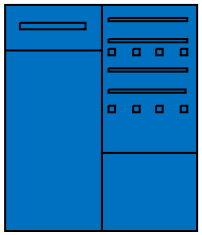
- Given an input, its VRF hash output looks pseudorandom
- … to any adversary that does not know its proof or **SK**.

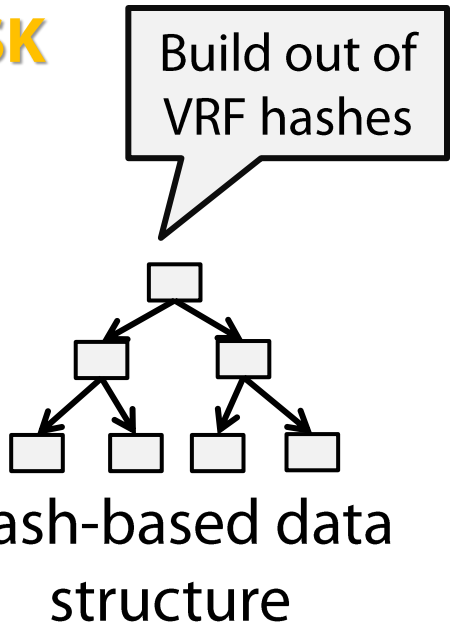
VRFs stop dictionary attacks on hash-based structures

Verifier PK



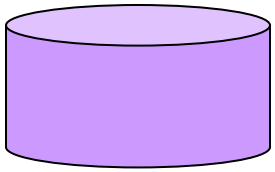
Hasher SK

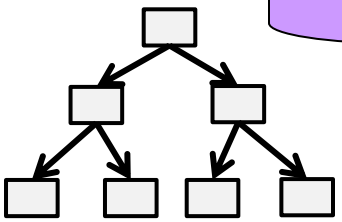




VRFs stop dictionary attacks on hash-based structures

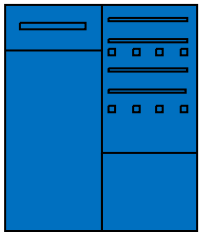
Verifier PK



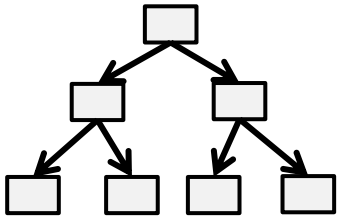


Hash-based data structure

Hasher SK

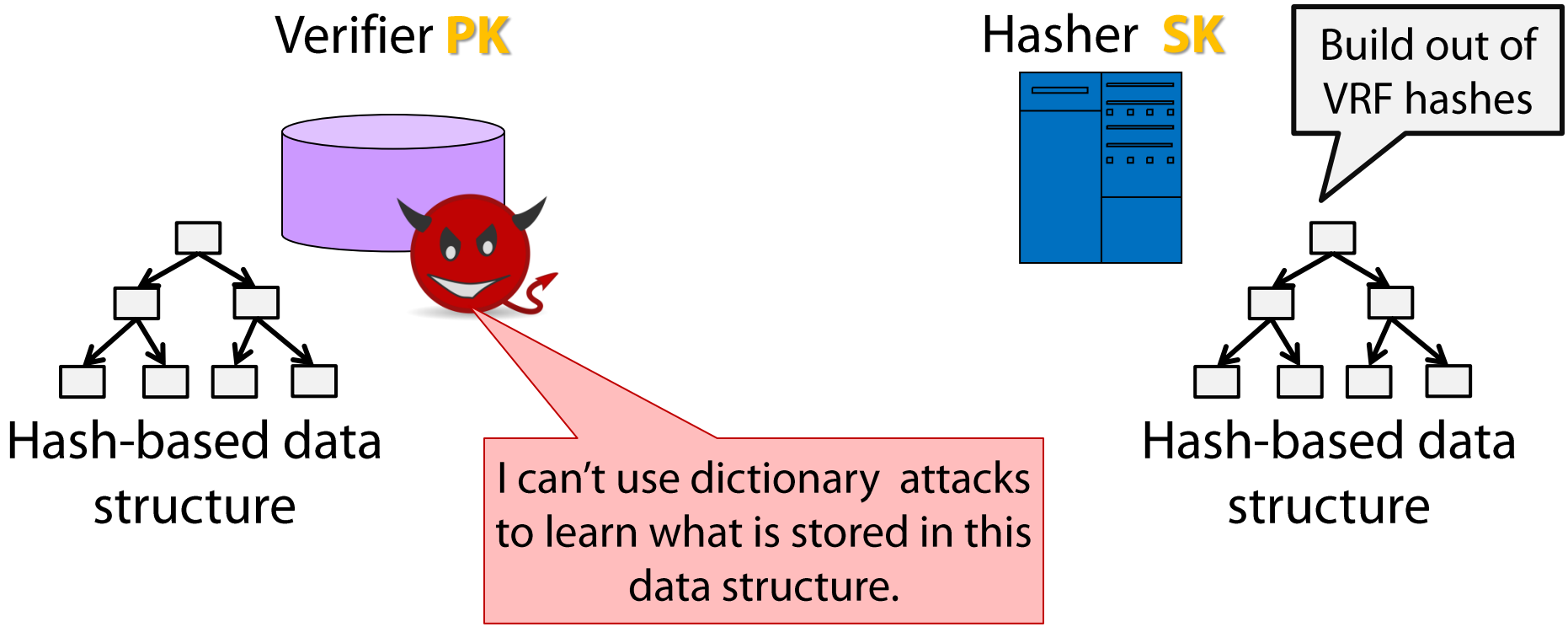


Build out of VRF hashes

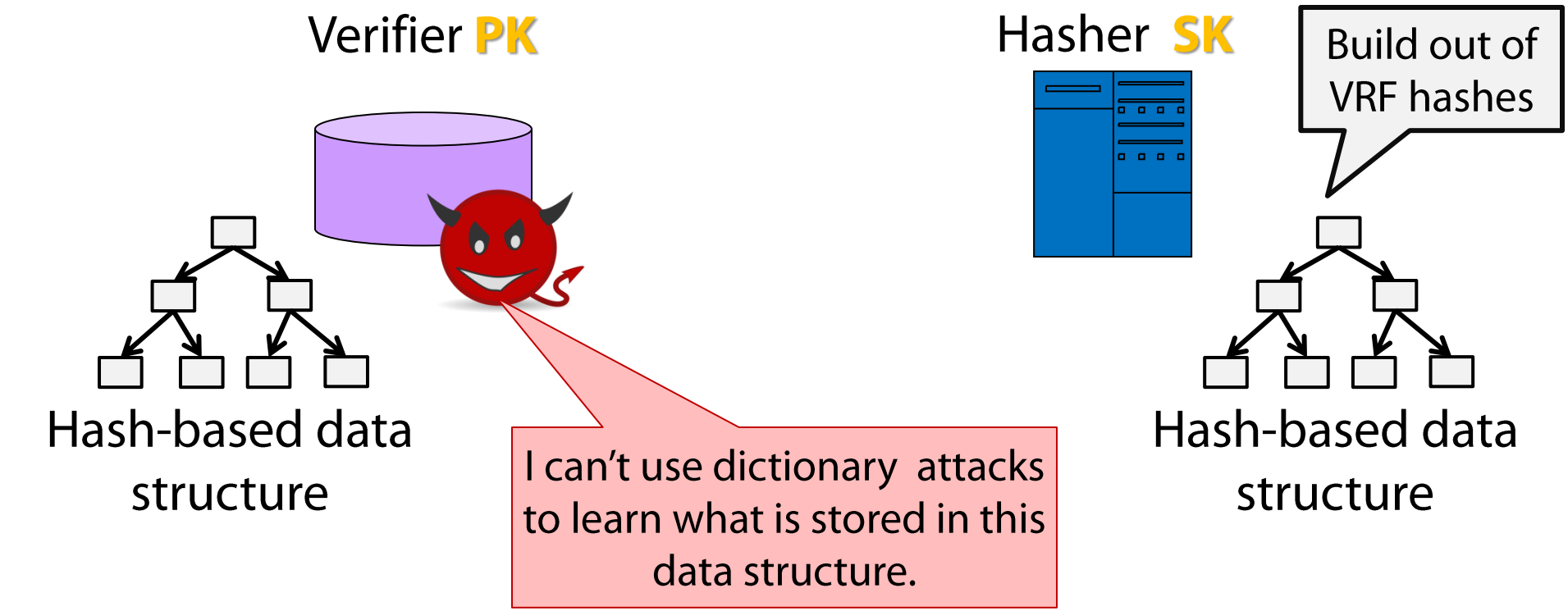


Hash-based data structure

VRFs stop dictionary attacks on hash-based structures

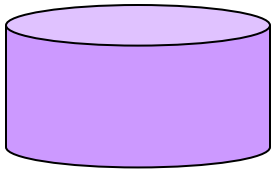


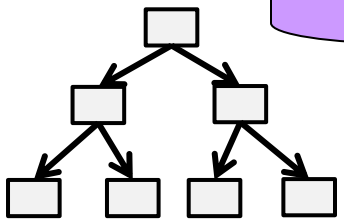
VRFs stop dictionary attacks on hash-based structures



VRFs stop dictionary attacks on hash-based structures

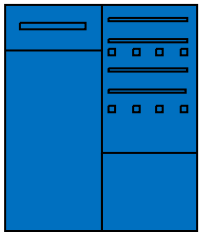
Verifier **PK**



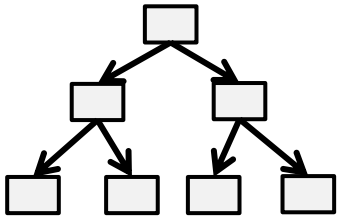


Hash-based data structure

Hasher **SK**



Build out of VRF hashes



Hash-based data structure

Is input in the data structure?

input →

← proof

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

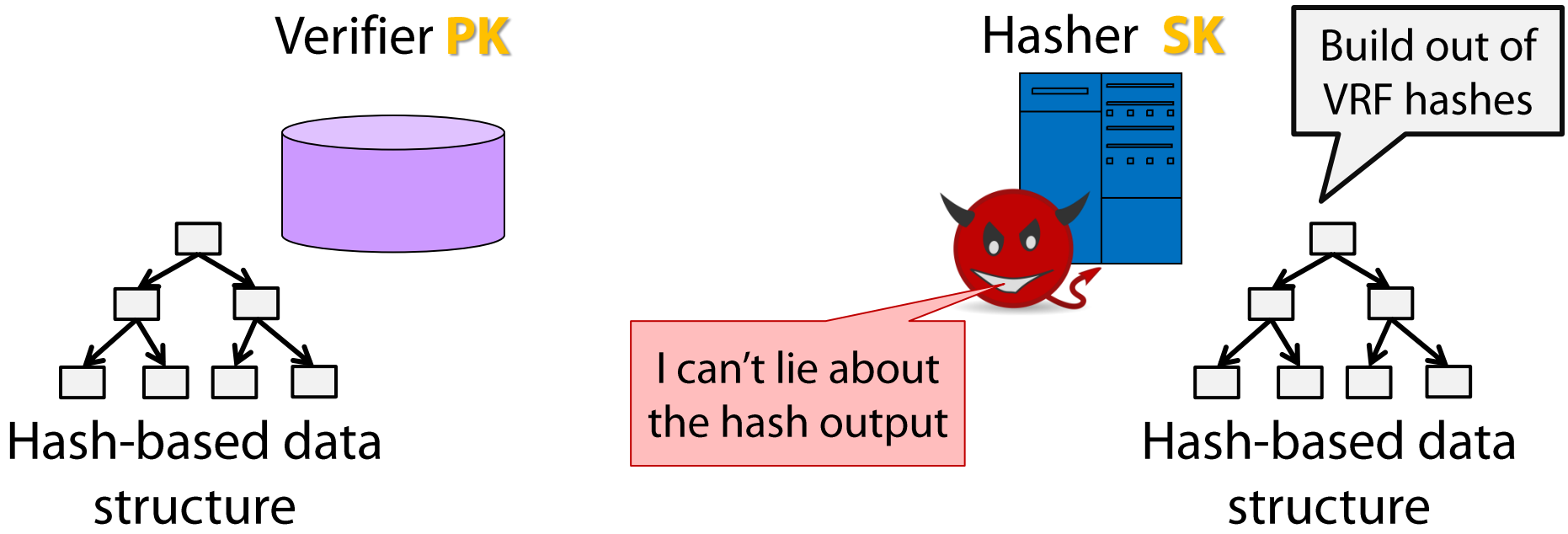
If **verify (PK**, input, proof**)**

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

Is hash in data structure?

Else INVALID

VRFs stop dictionary attacks on hash-based structures



Is input in the data structure?

input →

← proof

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

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

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

Is hash in data structure?

Else INVALID

-00 draft includes

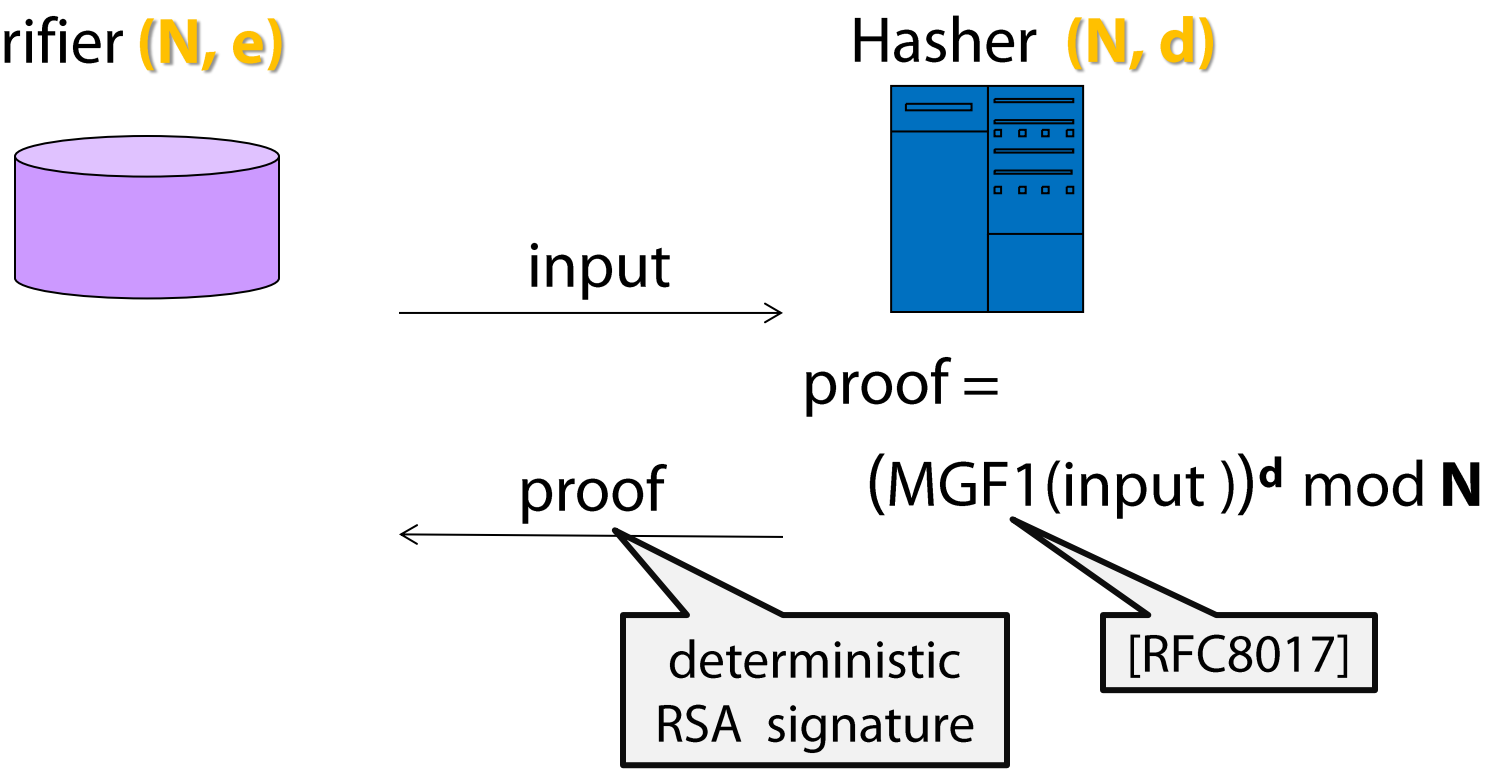
- VRF Security Definitions and Security Considerations

- Elliptic Curve VRF (**EC-VRF**)
  - Works with any cyclic group **G** of prime order **q** with generator **g**
  - Ciphersuites for NIST P-256 curve and Ed25519 curve
  - Algorithm is generic. Could support other curves

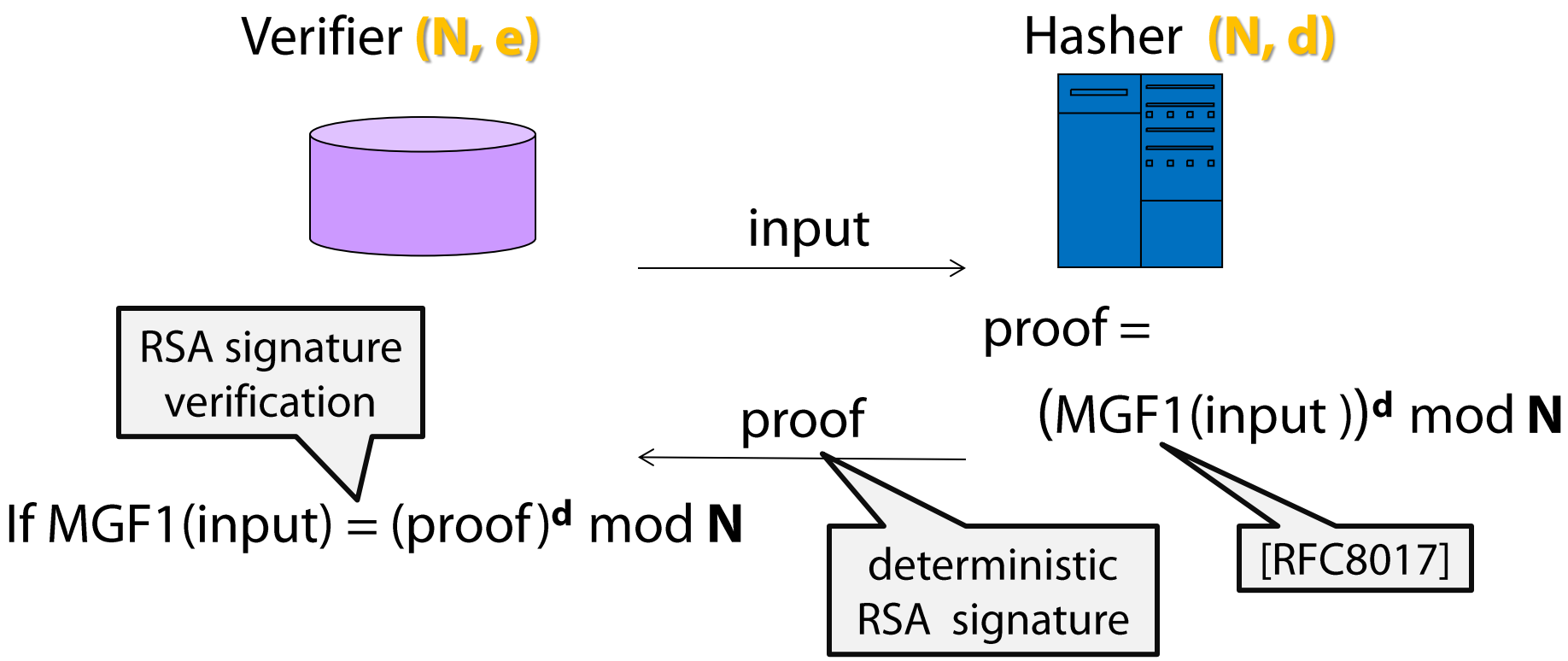
- RSA Full-Domain-Hash VRF (**RSA-FDH-VRF**)

- Also, we have:
  - Formal cryptographic security proofs: **http://ia.cr/2017/099**
  - Implementations: **https://github.com/fcelda/nsec5-crypto**

RSA-FDH-VRF (RSA full domain hash VRF)



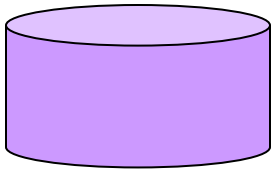
RSA-FDH-VRF (RSA full domain hash VRF)

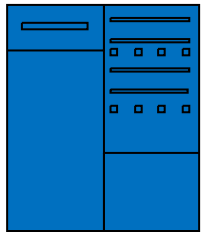


RSA-FDH-VRF (RSA full domain hash VRF)

Verifier (N, e)

Hasher (N, d)





input →

proof =

(MGF1(input))^{d} mod **N**

[RFC8017]

← proof

deterministic RSA signature

RSA signature verification

If MGF1(input) = (proof)^{d} mod **N**

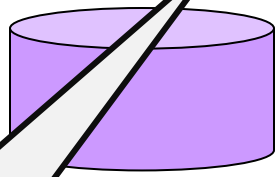
hash = **H(** proof **)**

Else INVALID

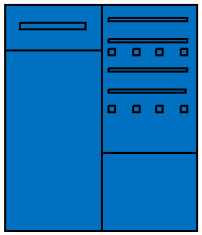
regular hash function (eg SHA256)

EC-VRF (elliptic curve VRF)

Verifier g^x



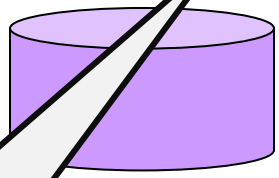
Hasher x



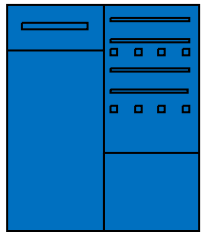
Cyclic group **G** of prime order **q** with generator **g** (eg Ed25519)

EC-VRF (elliptic curve VRF)

Verifier g^x



Hasher x



input

Cyclic group **G** of prime order **q** with generator **g** (eg Ed25519)

regular hash function (eg SHA256)

$\mathbf{h} = \text{hash\_to\_curve (input)}$

$\boldsymbol{\gamma} = \mathbf{h}^{\mathbf{x}}$

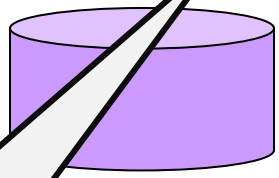
choose random nonce **k**

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

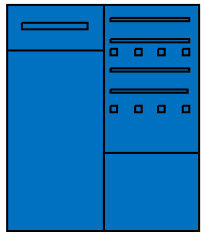
$\mathbf{s} = \mathbf{k - cx \bmod q}$

EC-VRF (elliptic curve VRF)

Verifier g^x



Hasher x



input →

$\mathbf{h} = \text{hash\_to\_curve (input)}$

$\boldsymbol{\gamma} = \mathbf{h}^{\mathbf{x}}$

choose random nonce $\mathbf{k}$

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

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

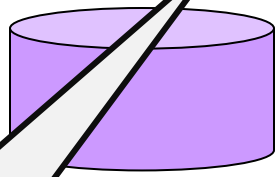
← proof: $(\boldsymbol{\gamma}, \mathbf{c}, \mathbf{s})$

Cyclic group **G** of prime order **q** with generator **g** (eg Ed25519)

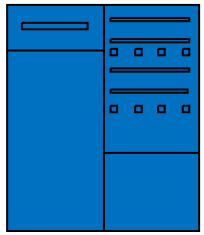
regular hash function (eg SHA256)

EC-VRF (elliptic curve VRF)

Verifier g^x



Hasher x



Cyclic group **G** of prime order **q** with generator **g** (eg Ed25519)

input →

h = hash_to_curve (input)

γ = $\mathbf{h^x}$

choose random nonce **k**

regular hash function (eg SHA256)

c = **H(g, g^x, h, h^x, g^k, h^k)**

← proof: **(γ, c, s)**

s = **k – cx mod q**

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

h = hash_to_curve (input)

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

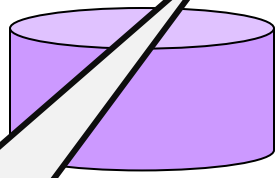
If **c** = **H(g, g^x, h, γ, u, v)**

hash = x-coordinate of **γ**

Else INVALID

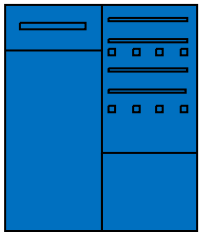
EC-VRF (elliptic curve VRF)

Verifier g^x



Cyclic group **G** of prime order **q** with generator **g** (eg Ed25519)

Hasher **x**



input →

h = hash_to_curve (input)

γ = $\mathbf{h}^{\mathbf{x}}$

choose random nonce **k**

regular hash function (eg SHA256)

c = **H(g, g^x, h, h^x, g^k, h^k)**

← proof: **(γ, c, s)**

s = **k – cx mod q**

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

h = hash_to_curve (input)

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

If **c** = **H(g, g^x, h, γ, u, v)**

hash = x-coordinate of **γ**

Else INVALID

ciphersuites

- NIST P-256 curve with SHA256
- Ed25519 curve with SHA256
- Could add other curves (eg Ed448)