EUROCRYPT'17
Paris, May 2017

an IETF standard for

Verifiable Random Functions (VRF)

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

Contributors: 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**)**

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**

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

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

First proposed by
Micali-Rabin-Vadhan
1999

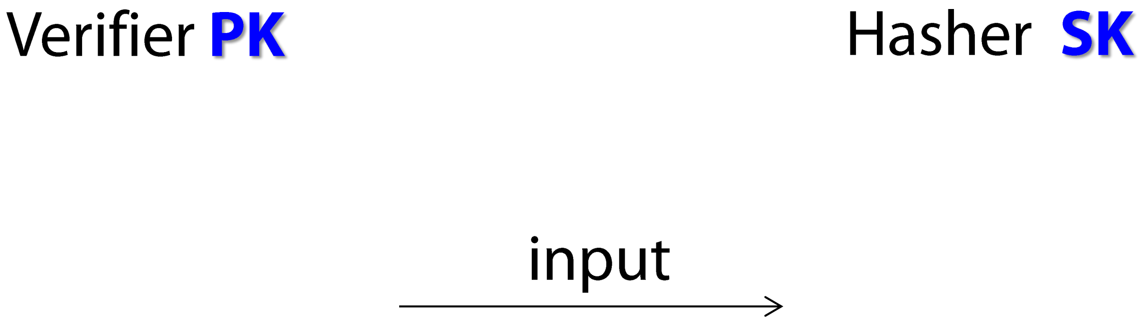
VRF: verifiable random function

Verifier **PK**

Hasher

SK

VRF: verifiable random function



Verifier **PK**

Hasher **SK**

input

VRF: verifiable random function

Verifier **PK** Hasher **SK**

$$\xrightarrow{\text{input}}$$

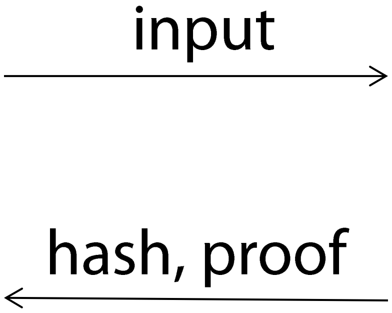
hash = **hash(SK**, input**)**

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

VRF: verifiable random function

Verifier **PK**

Hasher **SK**



input

hash = **hash(SK**, input**)**

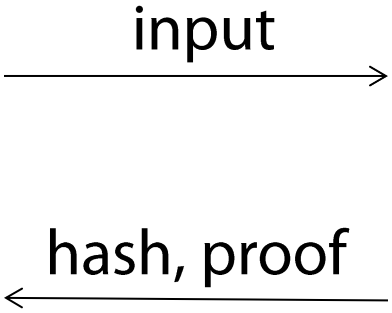
proof = **prove(SK**, input**)**

hash, proof

VRF: verifiable random function

Verifier **PK**

Hasher **SK**



input

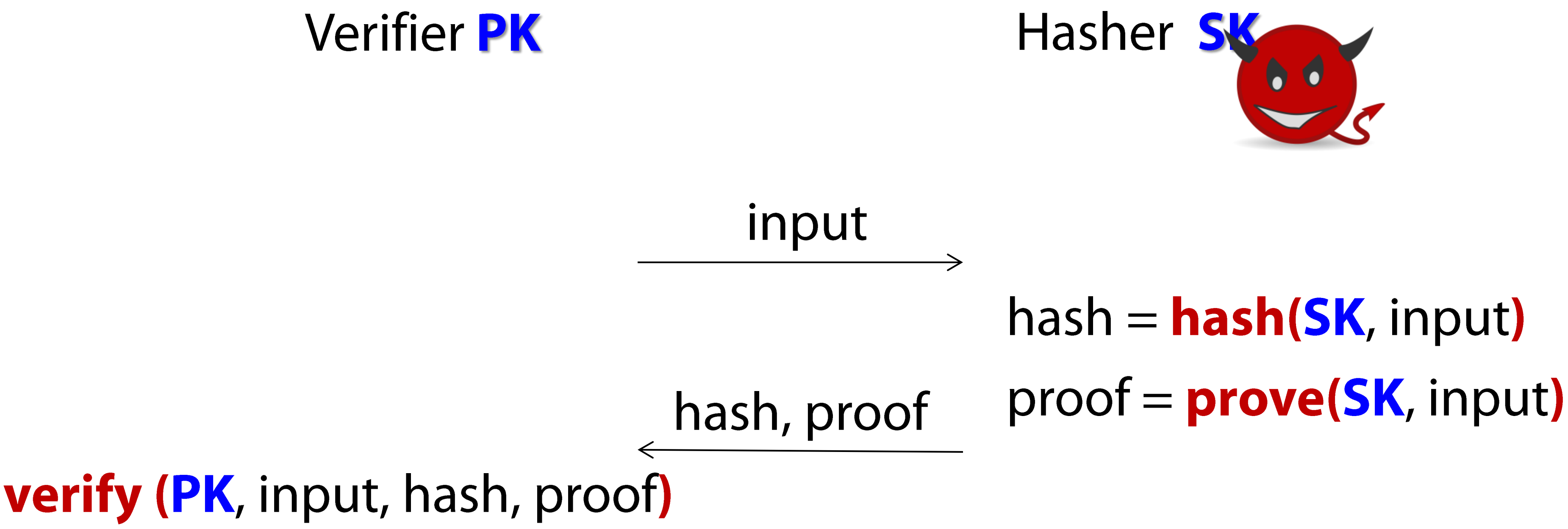
hash = **hash(SK**, input**)**

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

hash, proof

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

VRF security: uniqueness and collision-resistance



Verifier **PK**

Hasher **SK**

input

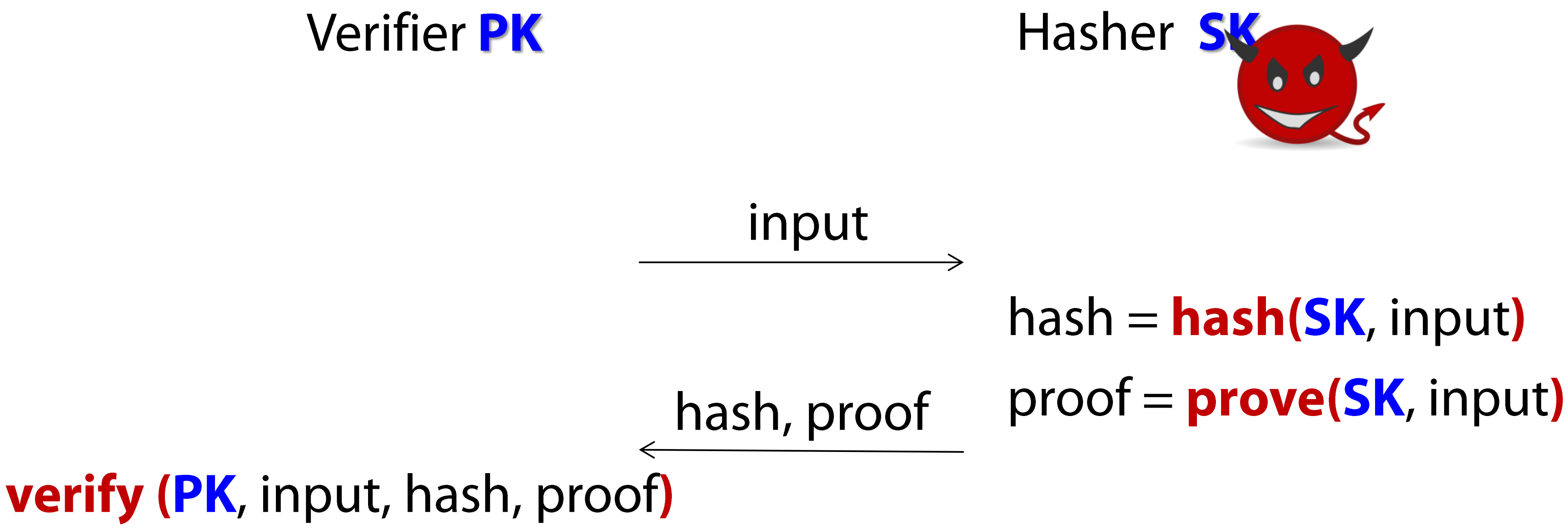
hash = **hash(SK**, input**)**

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

hash, proof

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

VRF security: uniqueness and collision-resistance



Verifier **PK**

Hasher **SK**

input

hash = **hash(SK**, input**)**

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

hash, proof

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

Like any public hash function:

output verifiably determined by input

and collisions hard to find

VRF security: pseudorandomness

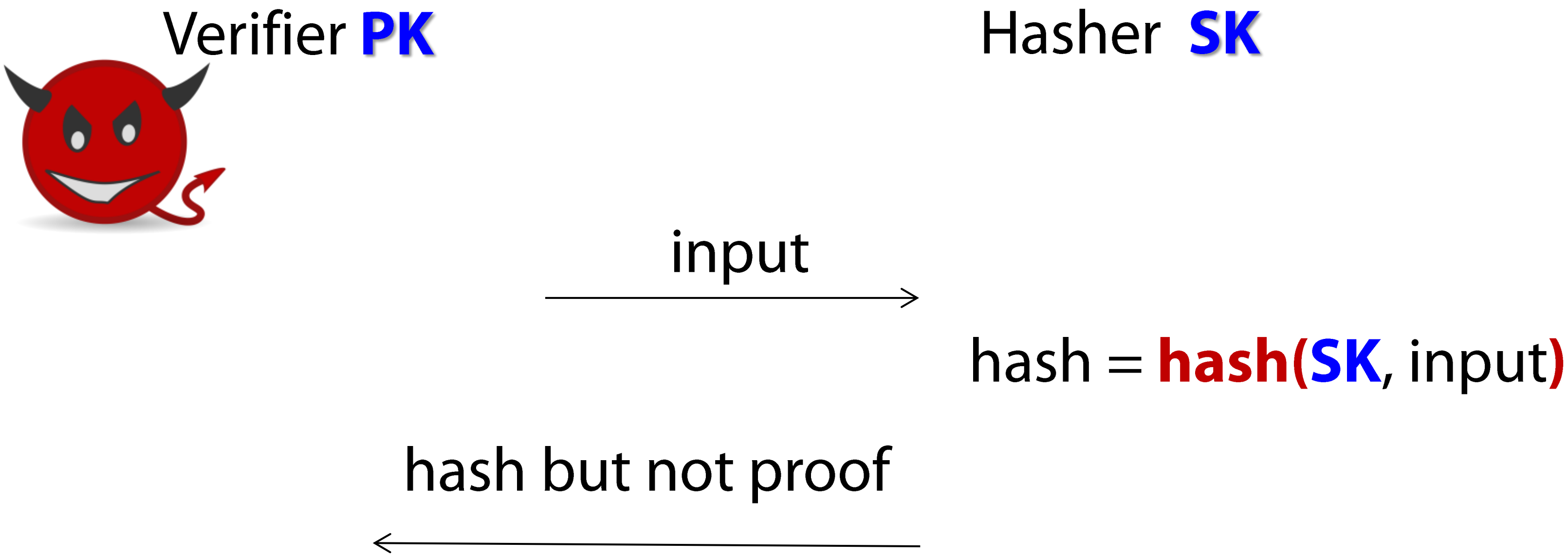
Verifier **PK**

Hasher **SK**

$\xrightarrow{\text{input}}$

hash = **hash(SK**, input**)**

VRF security: pseudorandomness



Verifier **PK**

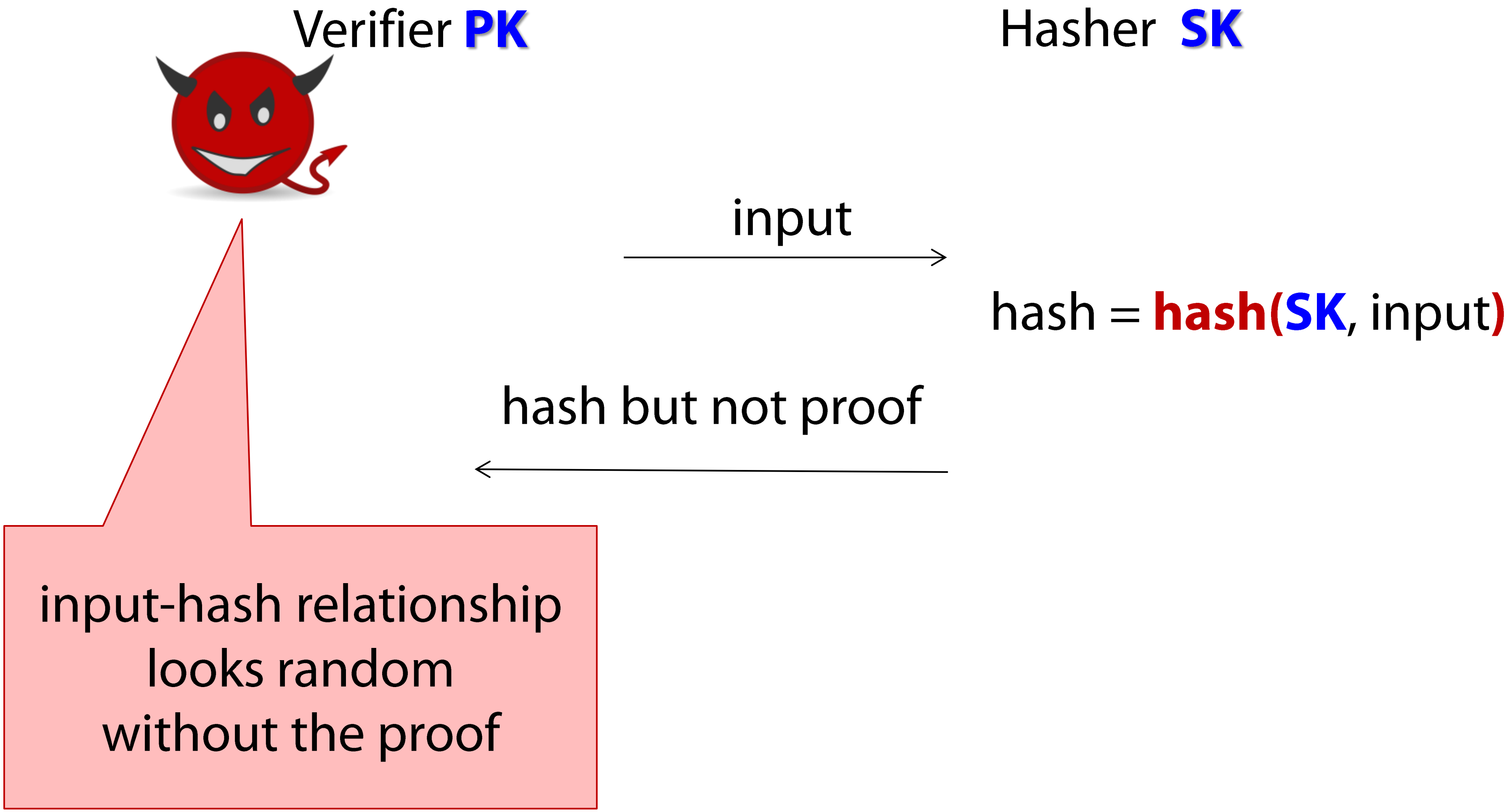
Hasher **SK**

input

hash = **hash(SK**, input**)**

hash but not proof

VRF security: pseudorandomness



Verifier **PK**

Hasher **SK**

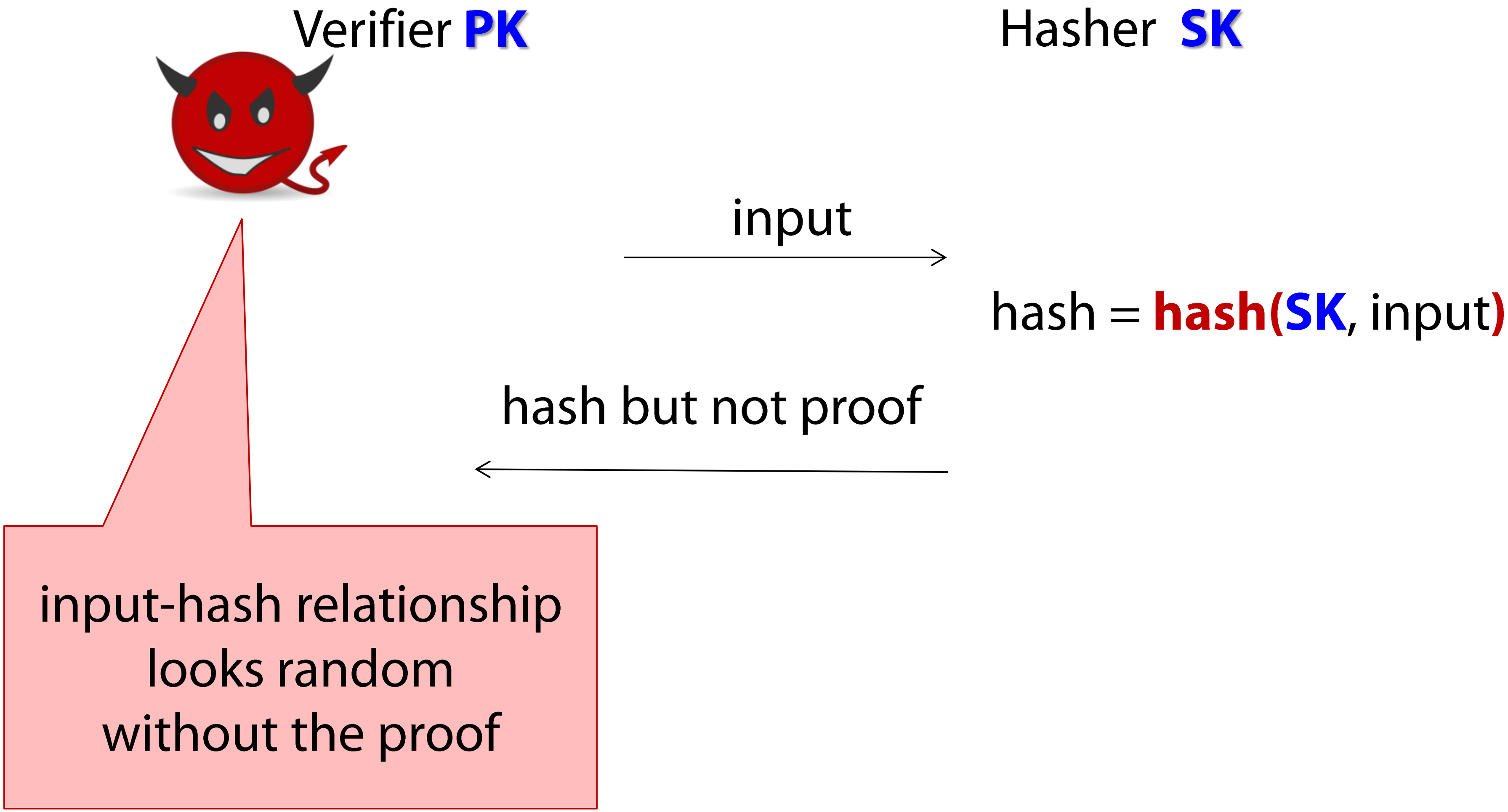
input

hash = **hash(SK**, input**)**

hash but not proof

input-hash relationship
looks random
without the proof

VRF security: pseudorandomness



Verifier **PK**

Hasher **SK**

input

hash = **hash(SK**, input**)**

hash but not proof

input-hash relationship
looks random
without the proof

Like a keyed hash function whose key you don't know

Why should you care?

VRFs Have:

- Collision Resistance (like hash functions)
- Uniqueness/Verifiability (like public hash functions)
- Pseudorandomness (like keyed hash functions)

But why should we want
to standardize them?

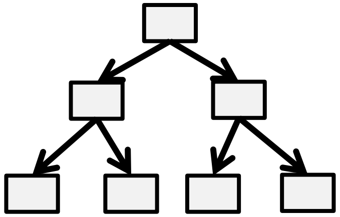
VRFs stop dictionary attacks on data structures

Verifier

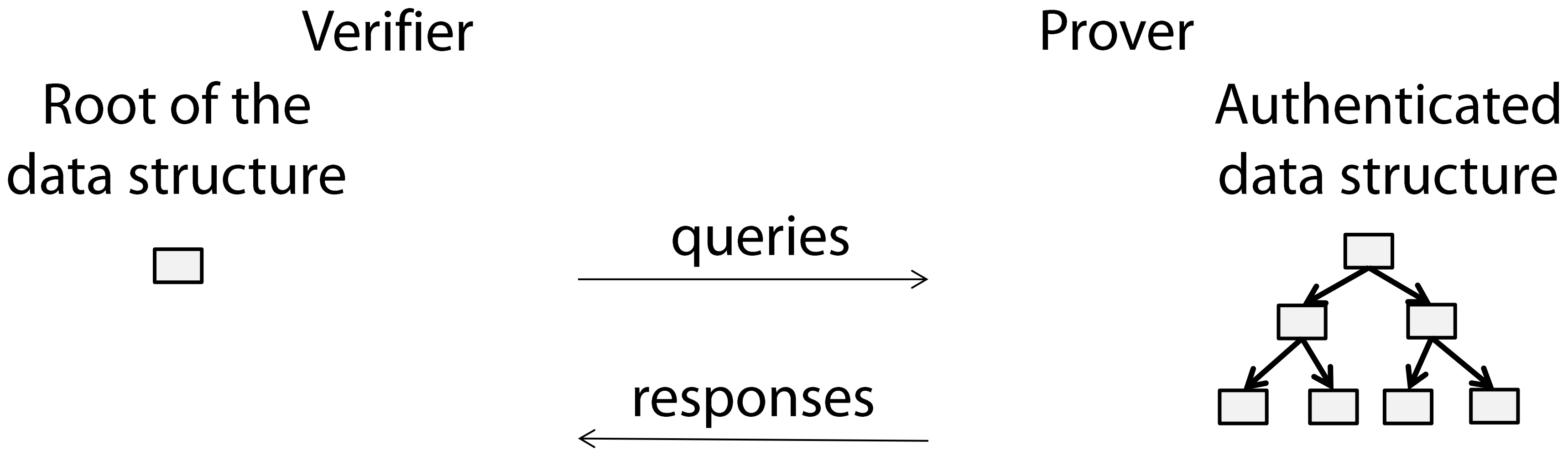
Root of the
data structure

Prover

Authenticated
data structure



VRFs stop dictionary attacks on data structures



Verifier

Prover

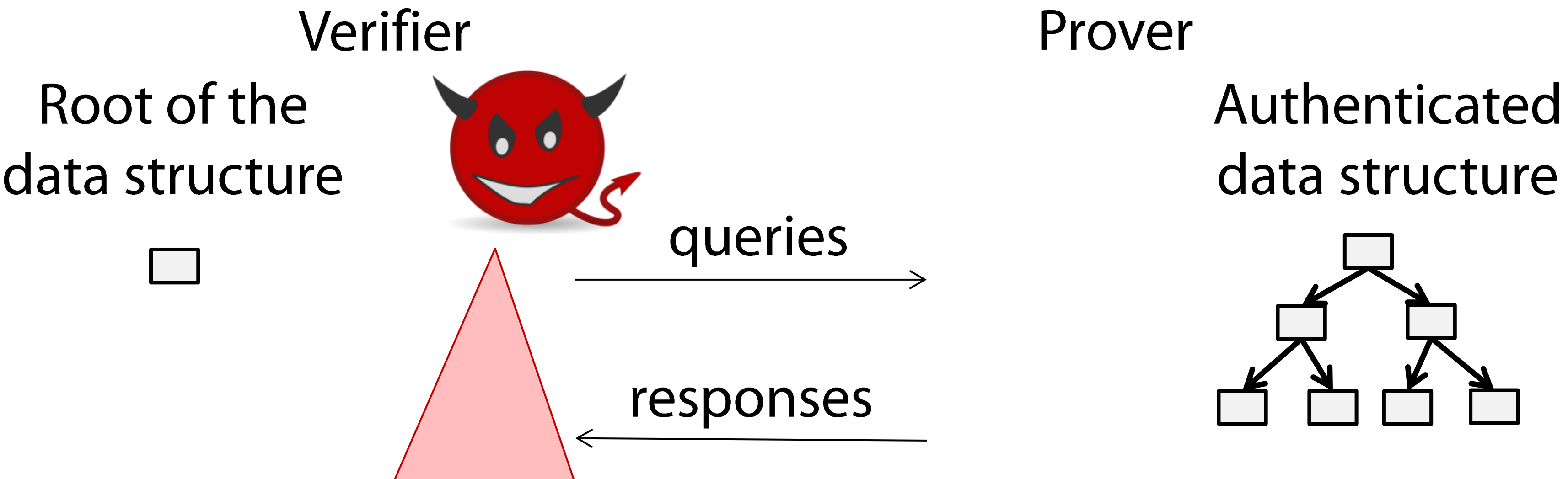
Root of the data structure

Authenticated data structure

queries

responses

VRFs stop dictionary attacks on data structures



Verifier

Prover

Root of the data structure

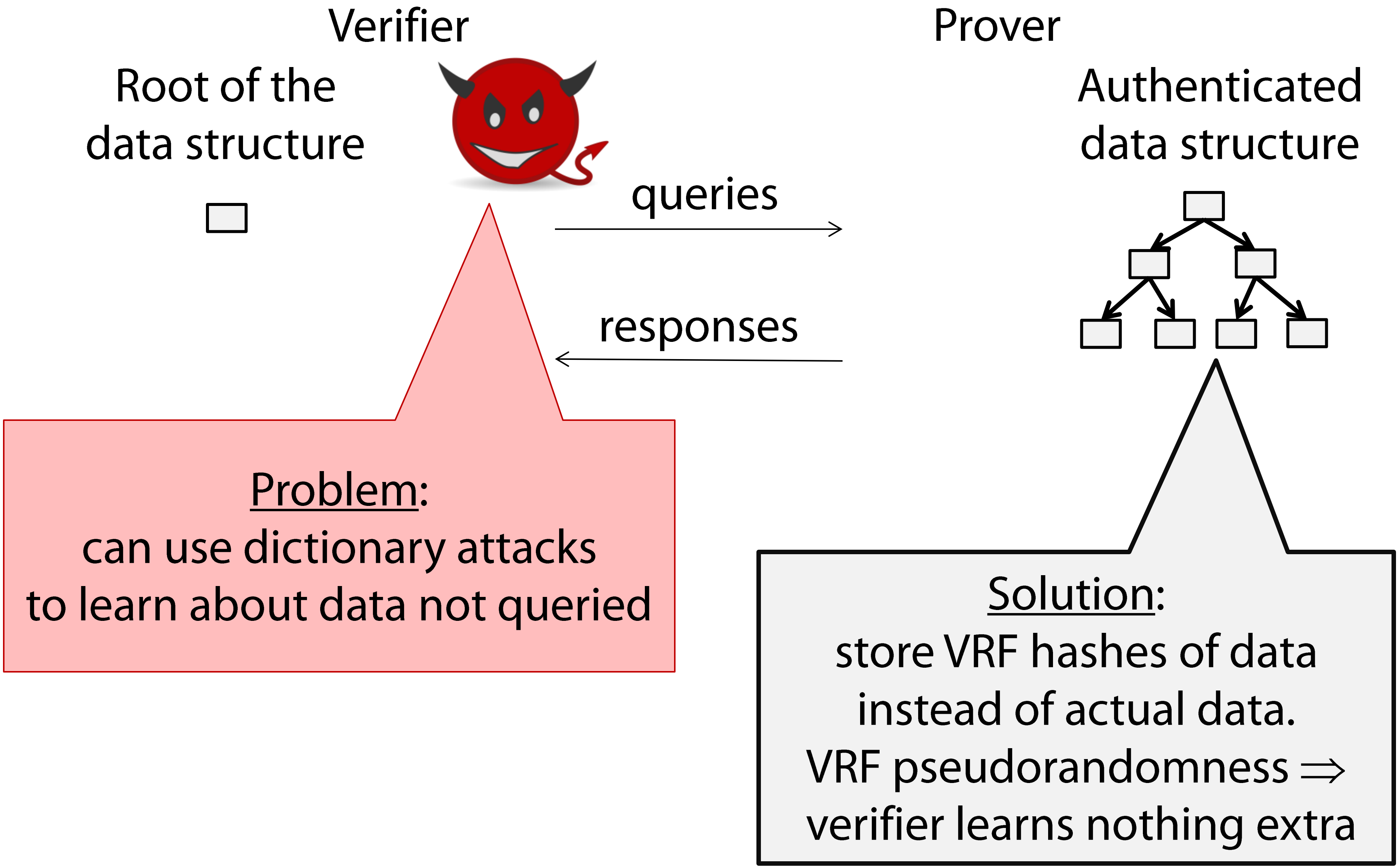
Authenticated data structure

queries

responses

Problem:
can use dictionary attacks
to learn about data not queried

VRFs stop dictionary attacks on data structures



Verifier

Root of the
data structure

queries

responses

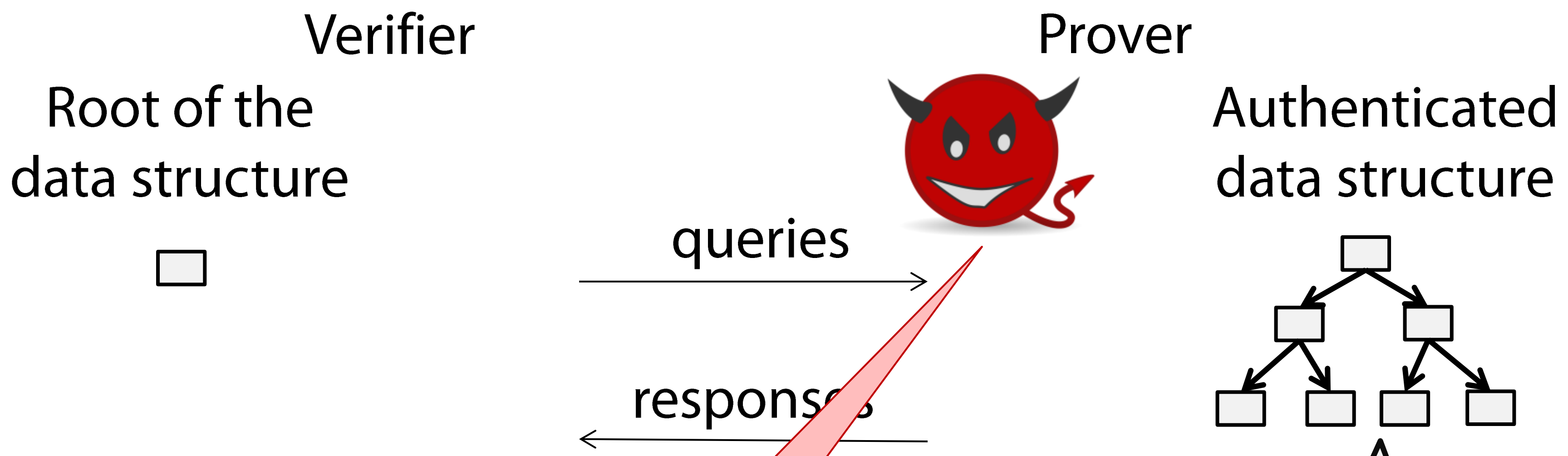
Prover

Authenticated
data structure

Problem:
can use dictionary attacks
to learn about data not queried

Solution:
store VRF hashes of data
instead of actual data.
VRF pseudorandomness ⇒
verifier learns nothing extra

VRFs stop dictionary attacks on data structures



Verifier

Prover

Root of the data structure

Authenticated data structure

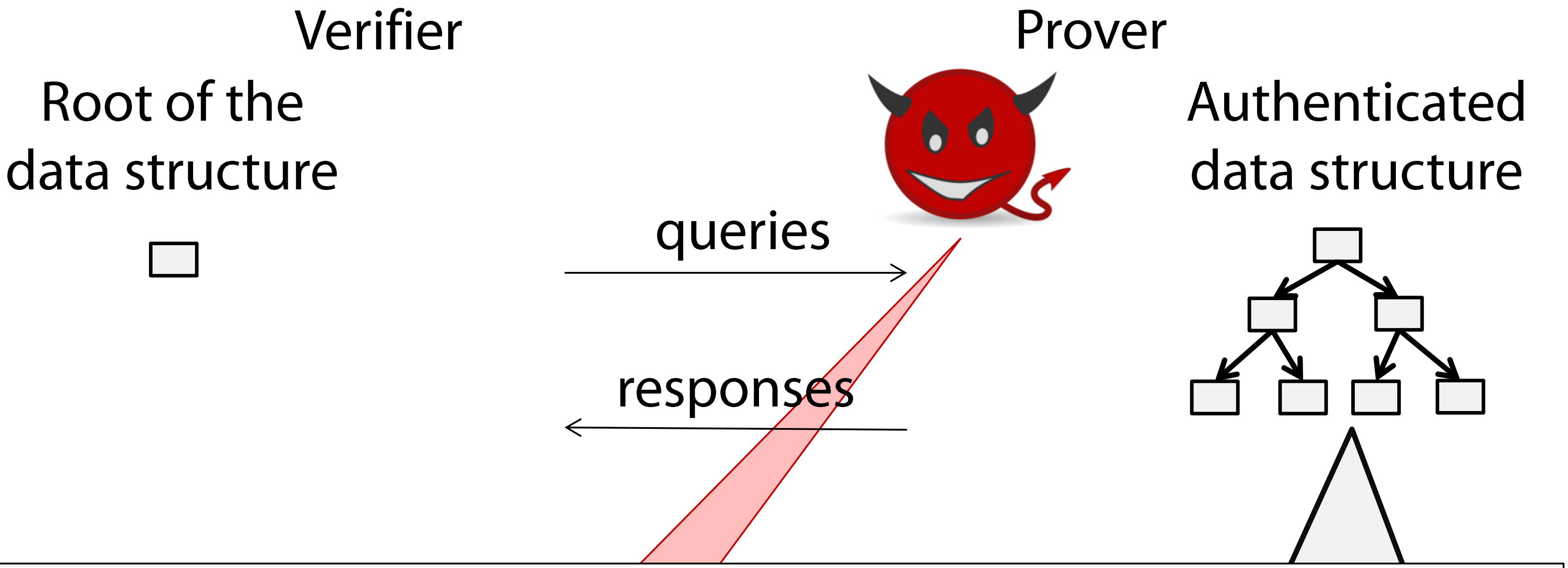
queries

responses

Because of VRF uniqueness, prover still can't lie about the data.

<u>Solution</u>: store VRF hashes of data instead of actual data. VRF pseudorandomness ⇒ verifier learns nothing extra

VRFs stop dictionary attacks on data structures



Useful for:

- **DNSSEC (NSEC5 denial of existence)**
- **Certificate transparency**
- **End-user key verification (CONIKS)**
- **Cryptocurrencies**

If interested in standardizing VRFs,

please talk to us!

If you use VRFs,

please talk to us!

Draft is out at

https://tools.ietf.org/html/draft-goldbe-vrf-00

(working through saag and cfrg)

reyzin@bu.edu