THE GRAINGER COLLEGE OF ENGINEERING
SIEBEL SCHOOL OF COMPUTING AND DATA SCIENCE

CS 521

Technological Foundations of Blockchain and Cryptocurrency

Grigore Rosu

Topic 4 – Zero Knowledge Proofs

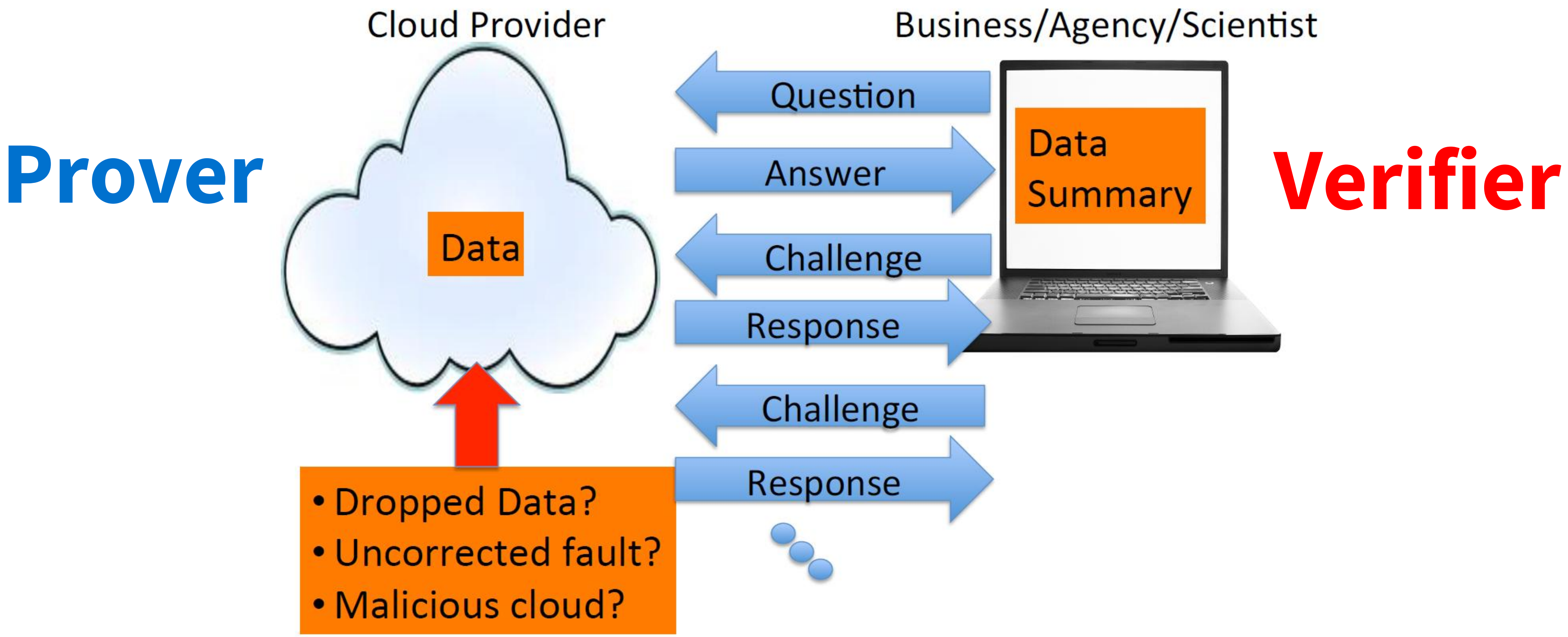
ILLINOIS

Thanks

- Justin Thaler
  - His book: https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html

Interactive Proof

Checking that something you do not have direct access to is true/correct



Proving Color Challenge

Given two objects, ■ and ■, which Verifier owns but cannot distinguish

Goal: Prover to convince Verifier that it knows that ■ ≠ ■

Reed-Solomon Fingerprinting

Goal: Prover to convince Verifier that it knows $(a_1, a_2, \ldots, a_n)$, where $a_i \in F$.

Naïve solution: Prover sends $a_1, a_2, \ldots, a_n$ (expensive, violates privacy, etc)

Good solution:

Verifier sends Prover random element $r \in F$

Prover sends Verifier $h = \sum_{i=1}^{n} a_i \cdot r^{i-1}$

Verifier checks received h against its locally computed $\sum_{i=1}^{n} a_i \cdot r^{i-1}$

Reed-Solomon Fingerprinting

The randomly chosen $r \in F$ needs to be a solution of $p_a(x) - p_b(x) = 0$, where

$$p_a(x) = \sum_{i=1}^{n} a_i \cdot x^{i-1} \qquad p_b(x) = \sum_{i=1}^{n} b_i \cdot x^{i-1}$$

But there are at most n solutions, so probability of r to be solution very small

Freivald’s Algorithm

Verifier has two $n * n$ matrices, A and B, with elements in field F_p.

Goal: Prover to convince Verifier that it knows C such that $C = A * B$

Naïve solution: Prover sends C to Verifier. Verifier computes $A * B$ and checks if it is equal to C or not. This is expensive (n^3), violates privacy, etc.

Good solution:

Verifier sends Prover random element $r \in F_p{}^n$

Prover sends Verifier $h = C * r$

Verifier checks received h against its locally computed $A * (B * r)$

Schnorr: proving knowledge of logarithm

Common Input: the description of a prime-order group $\mathbb{G}$ of (exponentially large) order p with a generator g, and a group element h.

Prover Witness: A value $x \in \mathbb{Z}_p$ such that $g^x = h$.

Protocol:

1. $\mathcal{P}$: pick $r \xleftarrow{\$} \mathbb{Z}_p$, send $\rho \leftarrow g^r$.
2. $\mathcal{V}$: pick $e \xleftarrow{\$} \mathbb{Z}_p$, send e.
3. $\mathcal{P}$: send $d \leftarrow e \cdot x + r \bmod p$

Verification: $\mathcal{V}$ accepts iff $g^d = h^e \rho$.

The Sum-Check Protocol

Given a *v*-variate polynomial *g* over a finite field *F*.

Goal: Prover to provide Verifier with the following sum:

$$H := \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_v \in \{0,1\}} g(b_1, \ldots, b_v)$$

Naïve solution: Verifier can compute *H* in $2^v * eval(g)$

- Considered unacceptable, suppose eval(g) is very expensive

Good solution: $O(v + eval(g))$ for Verifier; $O(2^v)$ for Prover

Round *1*

Prover claims C_1 to Verifier as value of H and sends $g_1(X_1)$ claimed to equal

$$\sum_{(x_2,\ldots,x_v)\in\{0,1\}^{v-1}} g(X_1, x_2, \ldots, x_v)$$

Verifier checks $C_1 = g_1(0) + g_1(1)$

Verifier sends random element $r_1 \in F$ to Prover

Round j $(1 < j < v)$

Prover sends Verifier $g_j(X_j)$ claimed to equal

$$\sum_{(x_{j+1},\ldots,x_v)\in\{0,1\}^{v-j}} g(r_1,\ldots,r_{j-1},X_j,x_{j+1},\ldots,x_v)$$

Verifier checks $g_{j-1}(r_{j-1}) = g_j(0) + g_j(1)$

Verifier sends random element $r_j \in F$ to Prover

Round v

Prover sends Verifier $g_v(X_v)$ claimed to equal

$$g(r_1, \ldots, r_{v-1}, X_v)$$

Verifier checks $g_{v-1}(r_{v-1}) = g_v(0) + g_v(1)$

Verifier chooses random element $r_v \in F$ and checks $g_v(r_v) = g(r_1, \ldots, r_v)$

Verifier evaluated g, which was assumed expensive, only once!