THE GRAINGER COLLEGE OF ENGINEERING

# CS 521

## Technological Foundations of Blockchain and Cryptocurrency

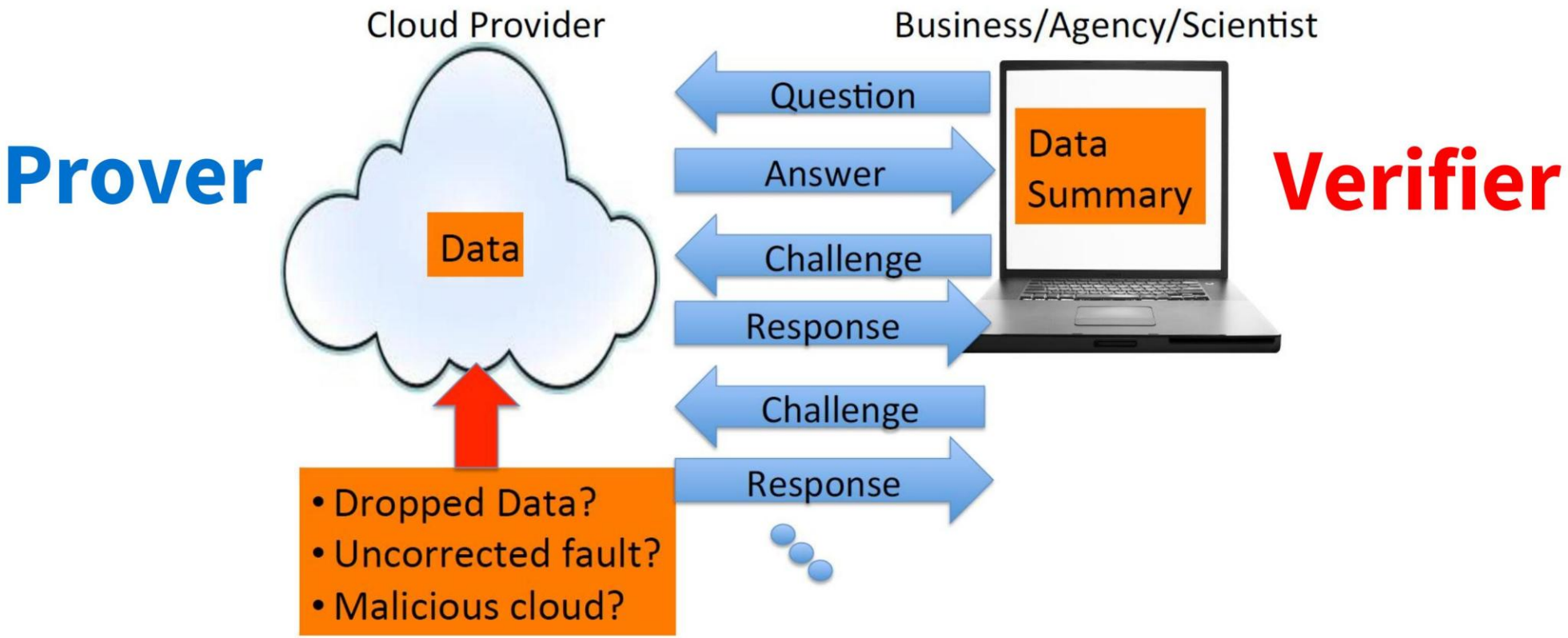
*Grigore Rosu*

Topic 6 – Zero Knowledge Proofs

ILLINOIS

# Interactive Proof

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



# Proving Color Challenge

Given two identical objects (e.g., balls), one green and one red, which Verifier owns but cannot distinguish (Verifier is color-blind)

Goal: Prover to convince Verifier that the two balls are different, without revealing which is which (zero-knowledge proof they are different)

# 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 $n \times n$ 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 Protocol: Proving Knowledge of a Discrete Logarithm

**Common Input:** a prime-order group $\mathbb{G}$ of (exponentially large) order $p$ with 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 $r \in \mathbb{Z}_p$, send $\rho \leftarrow g^r$
2. $\mathcal{V}$: pick $e \in \mathbb{Z}_p$, send $e$
3. $\mathcal{P}$: send $d \leftarrow e \cdot x + r \bmod p$

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

# Schnorr Protocol: Properties

**Completeness:**

If Prover knows *x* and follows the protocol honestly, Verifier always accepts:

$$g^d = g^{e \cdot x + r} = (g^x)^e \cdot g^r = h^e \cdot \rho$$

**Honest-verifier zero-knowledge:**

The transcript (ρ, *e*, *d*) reveals nothing about *x* beyond the fact that Prover knows it

A simulator (who does not know *x*) can produce transcripts indistinguishable from real ones

# Schnorr Protocol: Soundness

**What soundness means:**

A cheating Prover who does *not* know $x$ cannot convince Verifier to accept (except with negligible probability)

**Proof technique — knowledge extraction:**

Suppose a Prover can make Verifier accept. We "rewind" the Prover to the same commitment ρ and give it a different challenge $e'$. If it answers both:

- First run: challenge $e$, response $d$ where $d = e \cdot x + r$
- Second run: challenge $e'$, response $d'$ where $d' = e' \cdot x + r$

Subtracting: $d - d' = (e - e') \cdot x$, so we can recover the secret:

$$x = (d - d') / (e - e') \bmod p$$

Therefore, anyone who can convince Verifier must actually know $x$ (a "knowledge extractor" can recover it)

# Schnorr Signatures via Fiat-Shamir Transform

**Key idea:** make Schnorr non-interactive by replacing Verifier's random challenge with a hash (Fiat-Shamir heuristic)

**Signer** (Prover) signs message $m$ using secret key $x$**:**

1. Pick random $r \in \mathbb{Z}_p$, compute commitment $\rho = g^r$
2. Compute challenge $e = H(\rho \| m)$ ← replaces Verifier's random challenge
3. Compute response $d = e \cdot x + r \bmod p$. Signature = $(e, d)$

**Anyone** (Verifier) verifies $(e, d)$ on message $m$ using public key $h$**:**

1. Reconstruct the commitment: $\rho = g^d \cdot h^{-e}$

(computing $h$ to the power $-e$ is efficient: just invert $e$ mod $p$, then exponentiate)

2. Check $e = H(\rho \| m)$

**Applications:** basis for EdDSA (Ed25519), used in SSH, TLS, cryptocurrency

# Why Is Computing Inverses Easy?

To verify a Schnorr signature, we need $h^{-e}$. Is this hard?

**No — two simple steps:**

1. **Modular inversion**: compute $-e \bmod p = p - e$ (one subtraction)
2. **Modular exponentiation**: compute $h^{p-e}$ via repeated squaring — $O(\log p)$ multiplications

**Why this works:**

By Fermat's Little Theorem, $h^p = h$ in a group of order $p$, so:

$$h^{-e} = h^{p-e}$$

**Key contrast:** computing $h^{-e}$ is easy (polynomial time), but computing the discrete log $x$ from $h = g^x$ is believed to be hard

# 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 $2^v \cdot eval(g)$

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!

# Applications of the Sum-Check Protocol

The Sum-Check protocol is a cornerstone of modern verifiable computation. It enables a weak verifier to check claims made by a powerful (but untrusted) prover.

**Major application areas:**

- **Interactive proofs:** #P-complete problems (e.g., counting satisfying assignments)
- **Verifiable outsourced computation:** delegate computation to the cloud, verify the result cheaply
- **Zero-knowledge proof systems:** building block for SNARKs and STARKs
- **Blockchain scalability:** enabling Layer 2 rollups and succinct on-chain verification

# Sum-Check in Verifiable Computation

**The GKR Protocol (Goldwasser, Kalai, Rothblum, 2008):**

- Uses Sum-Check to verify the output of any arithmetic circuit
- Reduces verification of each layer to verification of the layer below, via Sum-Check at each step
- Verifier cost: nearly linear in the input size, regardless of circuit depth

**Practical impact — verifiable outsourced computation:**

- A client offloads expensive computation to a cloud server
- Server returns the result plus a proof (via Sum-Check)
- Client verifies the proof much faster than re-doing the computation

# Sum-Check in SNARKs, STARKs, and Blockchain

**SNARKs (Succinct Non-interactive Arguments of Knowledge):**

- Systems like Spartan use Sum-Check as the core proof mechanism
- Proof size is logarithmic in computation size — extremely compact

**STARKs (Scalable Transparent Arguments of Knowledge):**

- No trusted setup needed (transparent)
- Rely on Sum-Check combined with FRI (Fast Reed-Solomon IOP) for polynomial commitment

**Blockchain applications:**

- **ZK-Rollups:** batch thousands of transactions off-chain, post a single proof on-chain (e.g., StarkNet, zkSync, Polygon zkEVM)
- **Privacy:** prove transaction validity without revealing amounts or addresses (e.g., Zcash)