Selected Areas in Cryptology
Cryptanalysis
Week 7

Marc Stevens

stevens@cwi.nl

https://homepages.cwi.nl/~stevens/mastermath/

Asymmetric from symmetric cryptography

- Can we build asymmetric cryptography from symmetric cryptography?

- Benefits:
  - Symmetric cryptography seems generally to resist quantum cryptanalysis
  - No number-theoretic assumptions needed

- This week:
  - MPC-in-the-head on symmetric cryptography (continued)

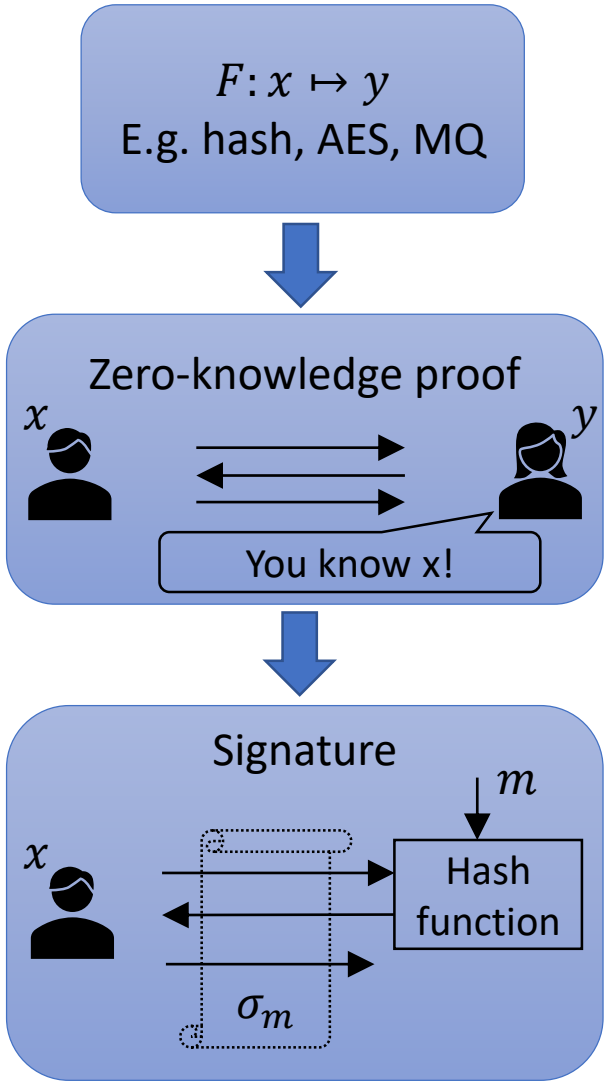
Recall: Signatures from Symmetric Crypto

- Consider any one-way function F
  - Hash function: $F(x) = f(x)$
  - Block cipher: $F(x) = Enc_x(0)$
  - MQ system: $F(\vec{x}) = \left(p_1(\vec{x}), \dots, p_k(\vec{x})\right)$, where $p_i(\vec{x}) \in F_q[x_1, \dots, x_n]$

- Consider a protocol P for a circuit C_y:
  - Protocol to prove a Prover knows a secret witness x such that $C_y(x) = 1$
  - Instantiation $C_y(x) = F(x) =? y$ for secret x and public $y = F(x)$
  - Protocols can be made non-interactive using hash function call

- Combine F and P into a signature scheme:
  - Add message to hash function call
  - ⇒ binds non-interactive proof to message
  - The non-interactive proof proves signer knows secret x for public key y

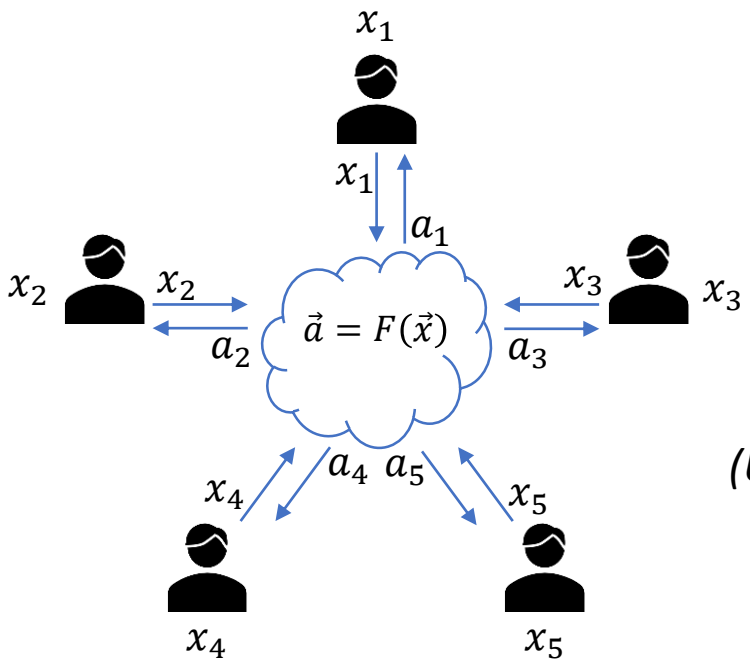
- This week: how can we transform any F into a zero-knowledge proof?



Secure Multi-Party Computation (MPC)

Secure Multi-Party Computation (MPC)

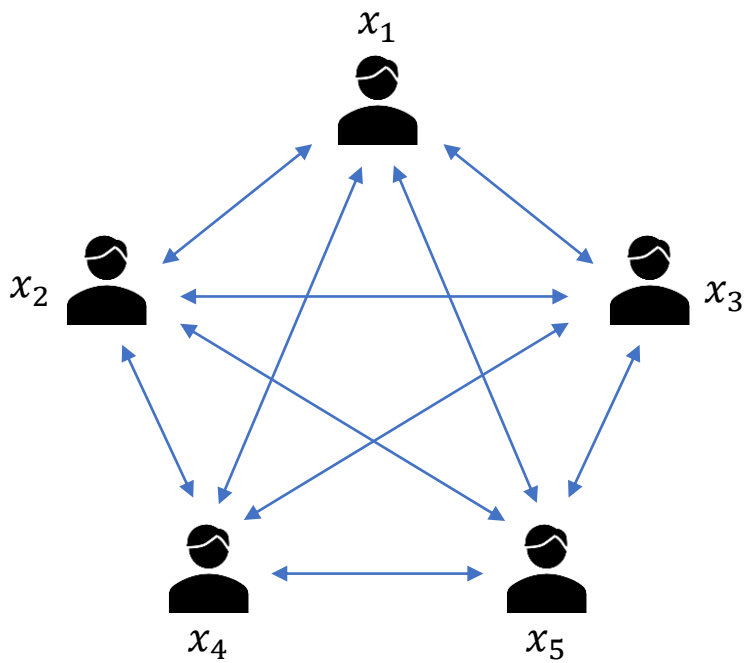
- Interactive ZK Proofs can be seen as a special case of MPC
- n parties $P_1, \ldots, P_n$ want to compute F on their secret inputs $x_1, \ldots, x_n$



Ideal Functionality
A trusted third party ("God")
that computes F for them
without leaking anything else

$\approx$

(Universal Composability Framework:
MPC Protocol is secure
if it's indistinguishable
from Ideal Functionality)



MPC Protocol
Parties jointly compute F
and learn its output
without learning anything else

Linear Secret Sharing Schemes (LSSS)

Cryptographic scheme to share a secret $x \in \mathbb{F}$ over n parties

- Notation : $[x] = (x_1, \dots, x_n)$ i-th share is also written as $[x]_i$
- Sharing : $[x] \leftarrow \mathrm{share}(x)$
- Reconstruction : $x = \mathrm{reconstruct}([x])$ (some shares are allowed to be missing/corrupted)
- ℓ-privacy:
  - If and only if: any set of ℓ shares $x_{i_1}, \dots, x_{i_\ell}$ is statistically independent of x
- t-reconstruction:
  - If and only if: x is correctly reconstructed given any subset of t correct shares $x_{i_1}, \dots, x_{i_t}$

- Linear:
  - Given secret sharings $[x], [y]$ and public scalar $a \in \mathbb{F}$
  - Addition : $[x + y] = (x_1 + y_1, \dots, x_n + y_n)$
  - Scalar multiplication : $[a \cdot x] = (a \cdot x_1, \dots, a \cdot x_n)$
  - $\Rightarrow$ any linear function on secret sharings $[s_1], \dots, [s_m]$ can be computed *locally*

Linear Secret Sharing Schemes (LSSS)

Example: *additive secret sharing*: $x = \sum x_i$

- Sharing:
  - $x_1, \dots, x_{n-1} \leftarrow \mathbb{F}$
  - $x_n = x - x_1 - \cdots - x_{n-1}$
  - Note: each individual share x_i is uniformly distributed over $\mathbb{F}$
- n-reconstruction:
  - $x = x_1 + x_2 + \cdots + x_n$
- Linear:
  - $\text{reconstruct}\big([x + y] = (x_1 + y_1, \dots, x_n + y_n)\big) = \sum(x_i + y_i) = (\sum x_i) + (\sum y_i) = x + y$
  - $\text{reconstruct}\big([a \cdot x] = (a \cdot x_1, \dots, a \cdot x_n)\big) = \sum(a \cdot x_i) = a \cdot (\sum x_i) = a \cdot x$

- $(n-1)$-privacy:
  - Any set of $(n-1)$ shares (say missing x_i) is statistically independent of x
  - Since we can rewrite sampling order to have $x_i = x - \sum_{j \neq i} x_j$ without changing distribution

Linear Secret Sharing Schemes (LSSS)

Example: (t, n)-Shamir's secret sharing over $\mathbb{F}$

- Select $n + 1$ pair-wise distinct field elements: $e_1, \dots, e_n,\ e_0 = 0$
- Sharing:
  - $[x] = \big(P(e_1), \dots, P(e_n)\big)$ for randomly sampled $P(X) \leftarrow \mathbb{F}[X]$ with $P(0) = x$ and $\deg(P) \leq t$
- $(t + 1)$-Reconstruction using Interpolation Theorem:
  - $P(X) = \text{interpolate}\Big(\big(e_{i_1}, x_{i_1}\big), \dots, \big(e_{i_{t+1}}, x_{i_{t+1}}\big)\Big)$ for any subset of $(t + 1)$ shares
  - $x = P(0)$
- Linearity follows linearity of polynomials:
  - A linear combination of share polynomials $a\ P_1(X) + b\ P_2(X)$ still has degree $\leq t$
  - Evaluations are also linear: $\big(a\ P_1(e) + b\ P_2(e)\big) = a \cdot \big(P_1(e)\big) + b \cdot \big(P_2(e)\big)$

- t-privacy follows from using $(0, x')$ as $(t + 1)$-th interpolation point:
  - Given any t shares, obtain bijection: $x' \leftrightarrow P'(X)$ with $\deg(P') \leq t$
  - Each value for $P'(X)$ has the same probability, and thus so does each value for x'

Multiplying Shares

- Any public linear function on secret shared values can be computed locally using linearity of secret sharing scheme
- Can we also do multiplications of secret shared values?
  I.e. obtain $[x \cdot y]$ from $[x]$ and $[y]$.
- Assume we have a Beaver-Triple Oracle BTO
  - I.e., that provides random sharings $[a], [b], [c = a \cdot b]$ for uniform random a, b
  - There are various protocols to achieve this. Recent work: Correlated Pseudorandom Generators!
- MPC Protocol to compute $[x \cdot y]$:
  1. Query $[a], [b], [c = a \cdot b] \leftarrow BTO$
  2. Compute $[\alpha] = [x + a]$ and $[\beta] = [y + b]$ locally
  3. Broadcast shares of $[\alpha], [\beta]$ and reconstruct α, β publicly
  4. Compute $[x \cdot y] = \alpha \cdot \beta - \beta \cdot [a] - \alpha \cdot [b] + [c]$ locally
  - Indeed $(x + a)(y + b) - (y + b)a - (x + a)b + ab = xy + \cancel{ay} + \cancel{bx} + \cancel{ab} - \cancel{ya} - \cancel{ba} - \cancel{xb} - \cancel{ab} + \cancel{ab}$
  - Note that as a, b are uniform random, α, β are as well, thus do no leak information about x, y

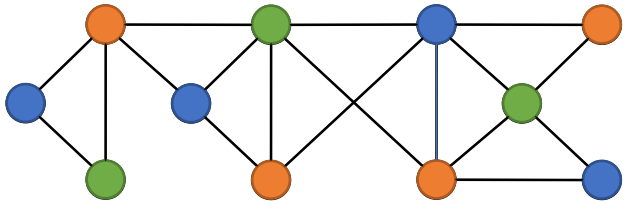
Basic MPC Protocol

- Given n *semihonest* parties $P_1, \ldots, P_n$ with secret inputs $x_1, \ldots, x_n \in \mathbb{F}$
- Given Circuit C of additions and multiplications in $\mathbb{F}$ that implements F
- MPC Protocol for C
  1. Each party P_i secret shares its secret input as $[x_i]$
  2. Evaluate circuit C
     - Every addition gate of C can be computed locally
     - Every multiplication gate of C with multiplication protocol using BeaverTriple Oracle (BTO)
     - (Note many multiplication gates can be computed in parallel)
  3. Results in secret shares of $[y_i]$ for $(y_1, \ldots, y_m) = C(x_1, \ldots, x_n)$
  4. Broadcast all shares of $[y_i]$ and reconstruct output y_i
- Correctness is straightforward
  - Assuming BTO is correct & all parties correctly follow the protocol
- Privacy of secret inputs & gate outputs: guaranteed by secret sharing & BTO
- ℓ-privacy allows up to ℓ colluding semihonest parties
  (which correctly follow the protocol, but try to learn more information together)
- Various techniques to protect against actively malicious parties (won't cover here)

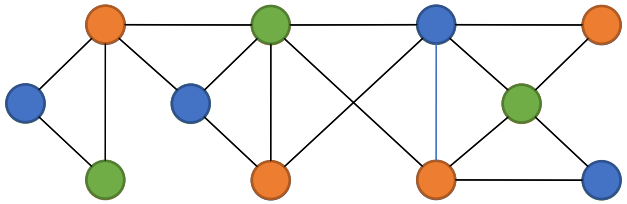
MPC-in-the-head

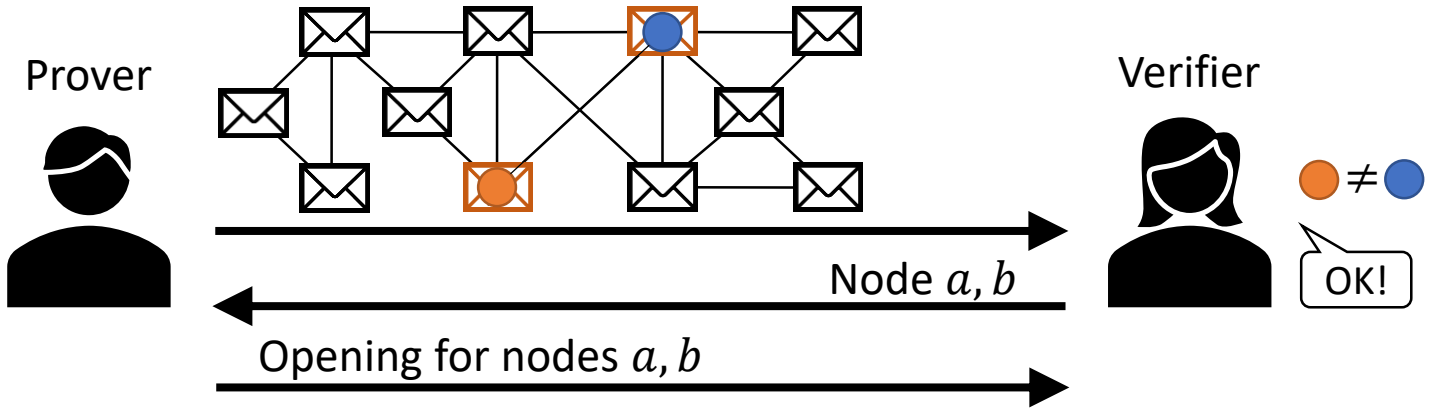
ZK-PoK for 3-Coloring

- A nice simple example towards MPC-in-the-Head:
  - 3-Coloring for a graph G
    - Each node is colored red, green or blue
    - Two adjacent nodes must have different color
  - Determining if a graph G admits a 3-Coloring is NP-Hard



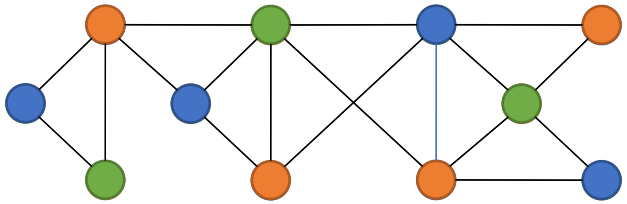
- A ZK-PoK to prove Prover knows a 3-Coloring for a graph G
  1. Prover randomly permutes colors on solution
  2. Prover sends a commitment for the coloring of each node
  3. Verifier randomly selects 2 neighboring nodes
  4. Prover opens commitments for those 2 nodes
  5. Verifier checks opening and that nodes have different color

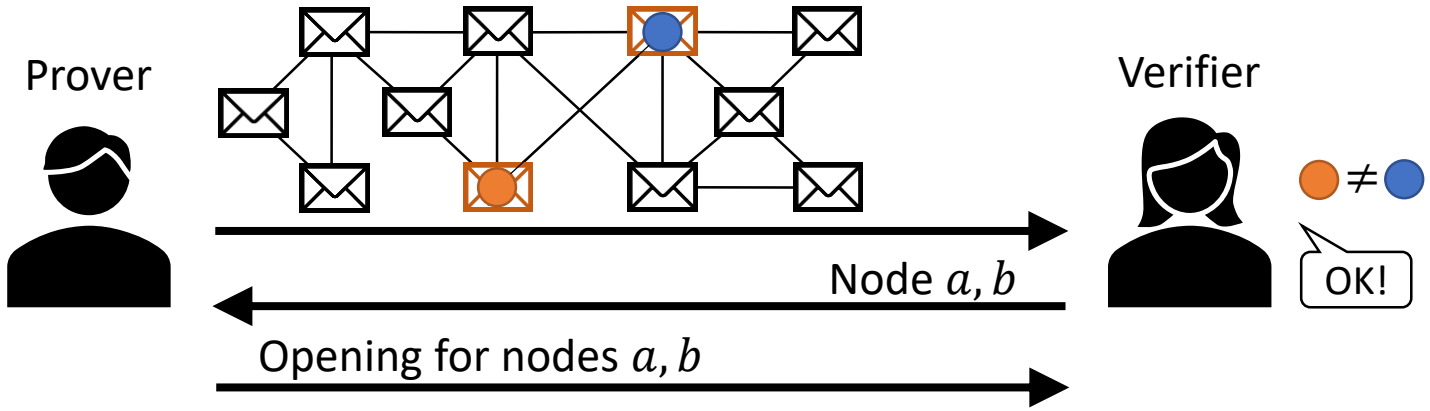




ZK-PoK for 3-Coloring

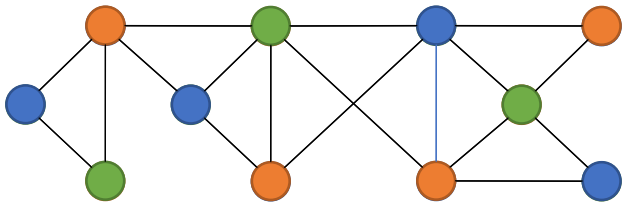
- Zero-knowledge: Simulator transcript generation
  - Can “cheat” by selecting a, b randomly before commitments
  - One can show transcript output distribution is statistically identical to valid transcripts (in the ROM)
- Special Knowledge Soundness: Extractor
  - Chooses 2 fixed neighbours a, b: note that coloring of a, b fixes prover’s color permutation on solution
  - After commitment, clone Prover to obtain related openings for 2 times 2 nodes: a, b, i, j
  - $\Rightarrow$ obtains prover’s solution for i, j under alternate fixed 3-coloring scheme (a-color, b-color, not-a-b-color)
  - Repeat until entire solution is recovered
- Soundness error: invalid solution $\Rightarrow$ at least 1 out of all N pairs is wrong $\Rightarrow \Pr[\text{"OK"}] \leq 1 - 1/N$

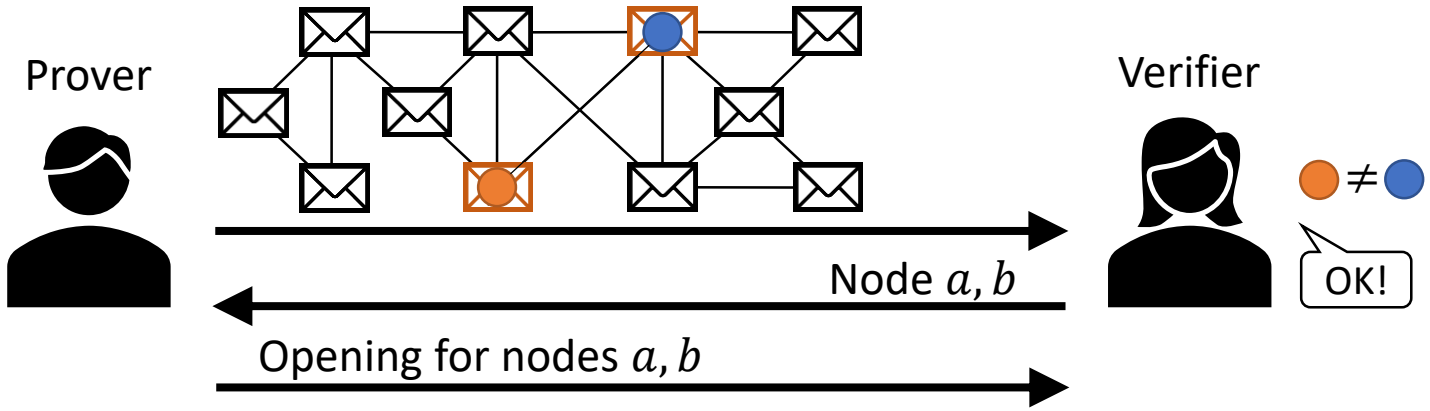




ZK-PoK for 3-Coloring

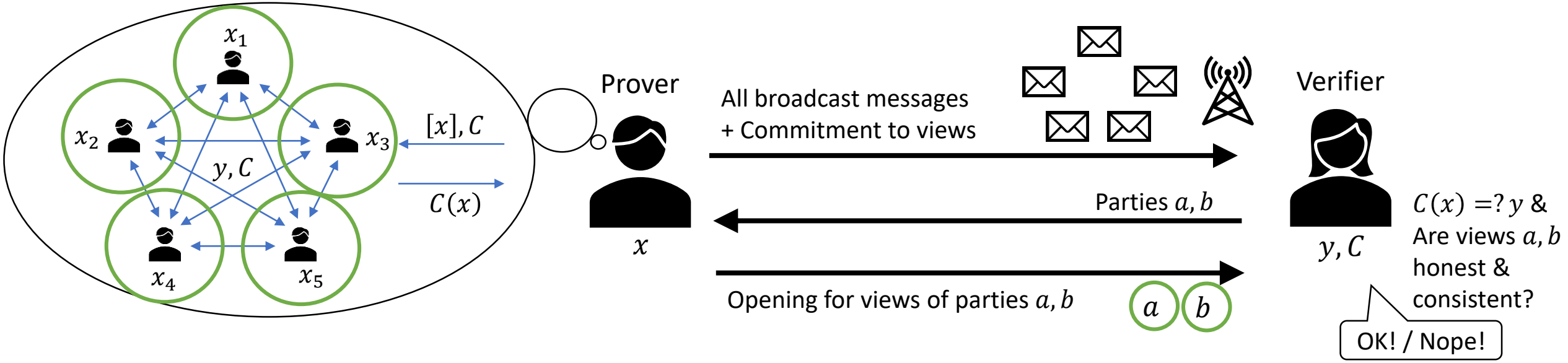
- Take-away concept
  - There is no 3-coloring computation in the interactive proof
  - Prover generates random ‘local’ views on solution & sends commitment
  - Verifier is allowed to verify a random view which verifies solution ‘locally’
- How to do this for arbitrary computations?





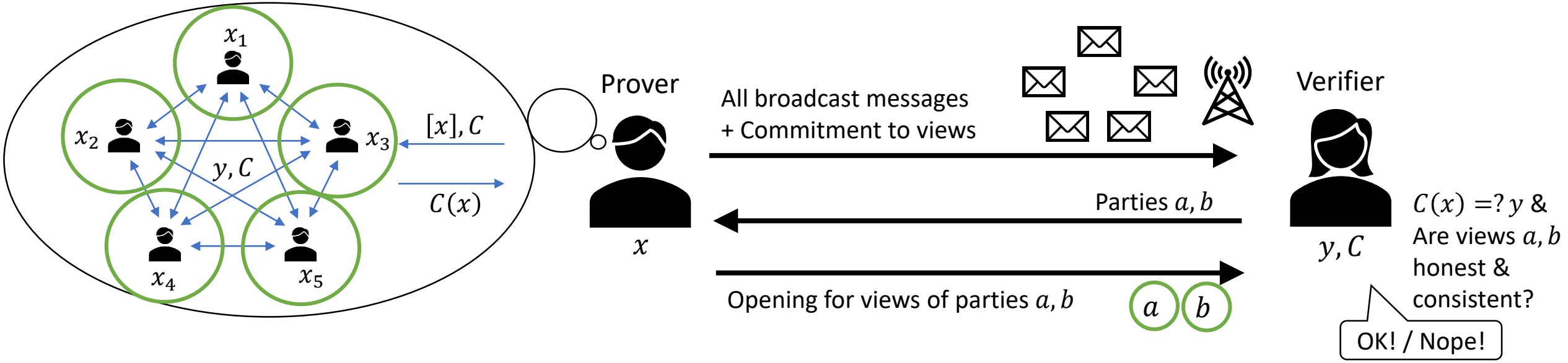
MPC-in-the-Head

- Generic ZK-PoK for knowledge of a computation, say $C(x)$
  1. Prover simulates MPC protocol run on secret shared x
  2. Prover generates local views for each party: all inputs/outputs & all messages to/from party
  3. Prover sends all broadcast messages and the commitment for each view
  4. Verifier randomly selects 2 parties a, b
  5. Prover opens views for parties a, b
  6. Verifier verifies MPC output $C(x) =? y$ and local views of a, b for correctness, i.e., that parties a, b acted honestly and have consistent views



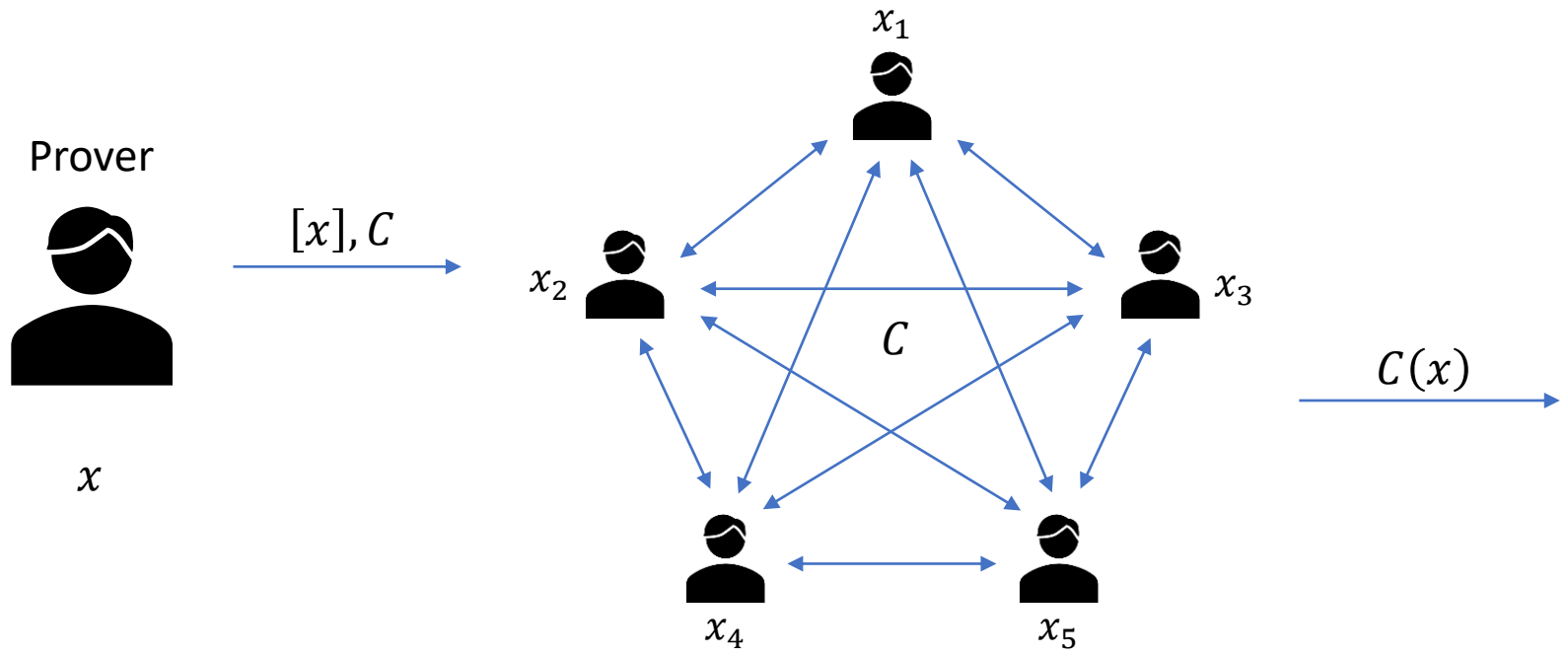
MPC-in-the-Head

- Zero-knowledge: follows from definition of MPC security
  - Recall: LSSS with ℓ-privacy ⇒ any subset of ℓ shares of x is statistically independent of x
- Special Knowledge Soundness:
  - Reconstruction of x possible by recovering *sufficiently many* related views through cloning of Prover
- Soundness error:
  - Invalid solution ⇒ ≥ 1 party dishonest, or inconsistency between views of ≥ 1 pair of parties
  - When opening ℓ views: $\Pr[\text{"OK"} \mid \text{bad solution}] \leq 1 - \binom{\ell}{2}/\binom{n}{2}$



MPC-in-the-Head

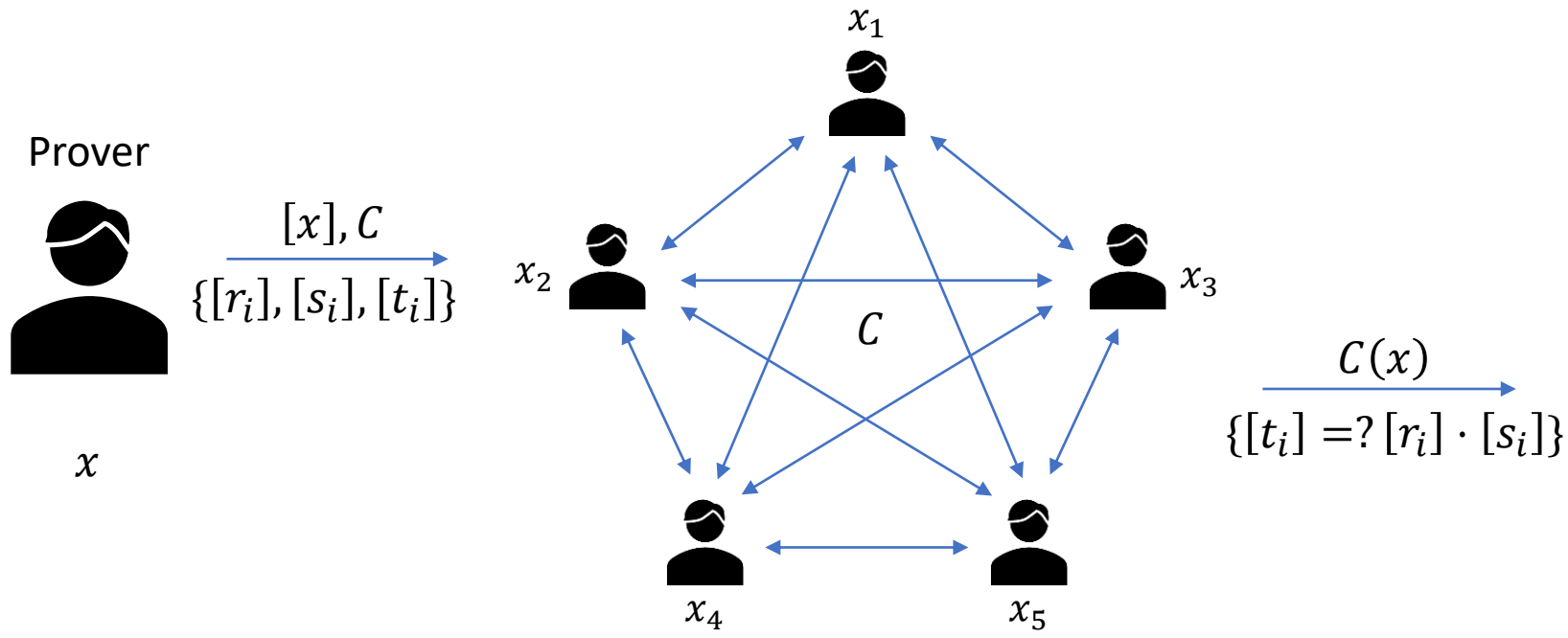
- Prover simulates n parties running a MPC protocol



- For any circuit C of addition & multiplication gates over $\mathbb{F}$
  - E.g. AES blockcipher over $\mathbb{F}_{2^8}$
    - All linear operations, except S-Box
    - Note AES S-Box is defined using multiplicative inverse
    - In MPC implemented as $[s] = [r]^{254}$ using square-and-multiply
    - But for MPC-in-the-Head we can do better using the fact that $[r] \cdot [s] = 1$!

MPC-in-the-Head

- Prover simulates n parties running a MPC protocol



- Number of rounds of parallel multiplications depends on *multiplication-depth* of C
- Improvement: Prover knows evaluation of $C(x)$
  - Prover can provide sharings of all multiplication gates in $C(x)$: $[r_i], [s_i], [t_i = r_i \cdot s_i]$
  - MPC computation becomes entirely linear!
  - But need to verify correctness of $[r_i], [s_i], [t_i]$ to verify correctness of $C(x)$!
  - Moreover! Verifying inverse $[s_i] = [r_i^{-1}]$ is simply checking 1 product that $[r_i] \cdot [s_i] = [t_i] = 1$

Multiplication Verification Protocol

- Prover wants to prove that $r \cdot s = t$ for secret shares $[r], [s], [t]$ to Verifier
- MPC-in-the-Head Protocol with inputs from Prover *and* Verifier:
  1. Prover shares random BeaverTriple $[a], [b], [c]$ to the n parties
  2. Verifier sends randomly selected scalar $\rho \in \mathbb{F}\backslash\{0\}$
  3. Parties compute $[\alpha] = \rho[r] + [a], \quad [\beta] = [s] + [b]$
  4. Parties broadcast shares & reconstruct α, β
  5. Parties compute $[v] = \rho[t] - [c] + \alpha[b] + \beta[a] - \alpha\beta$
  6. Parties broadcast shares & reconstruct v
  7. Verifier checks $v = 0$

  - Note that if MPC protocol was simulated correctly then
    $v = \rho t - c + (\rho r b + ab) + (sa + ba) - (\rho rs + \rho rb + as + ab)$
    $= \rho(t - rs) + (ba - c)$
  - If $r \cdot s = t$ and $a \cdot b = c$ then $v = 0$
  - If exactly 1 inequality $r \cdot s \neq t$ *OR* $a \cdot b \neq c$ then $v \neq 0$
  - If $r \cdot s \neq t$ and $a \cdot b \neq c$ then $\Pr[v = 0] = \Pr[\rho = (t - rs)^{-1}(c - ab)] = 1/|\mathbb{F}^*|$

  - Note that this omits the necessary steps to verify the MPC protocol was simulated correctly (i.e., P: commitments to views, V: random view selection, P: opening of selected views, V: check)

MPC-in-the-Head Signature Scheme

- Bringing all components together:
  - Circuit $C(\cdot)$ implementing one-way function $F(\cdot)$
  - Prover knows pre-image x for $y = F(x)$
  - Prover simulates MPC protocol for n parties
    - Prover shares x and the outputs t_i of all multiplicative gates $(r_i, s_i, t_i = r_i \cdot s_i)$ in $C(x)$
    - MPC Parties compute all $[r_i]$, $[s_i]$ locally (fully linear in inputs $[x]$ and all $[t_i]$)
    - Run multiplication verification protocol for all $[r_i]$, $[s_i]$, $[t_i]$ in parallel
  - Prover proves simulated MPC protocol is correct
    - Using commitments for each party's view, and opening a random selection of views
  - Fiat-Shamir
    - Verifier's messages are generated by Random Oracle
    - Transcript of messages between Prover & Verifier is non-interactive ZK-PoK
  - Signature scheme
    - pk and m are also input to Random Oracle ⇒ non-interactive ZK-PoK bound to pk & m
    - Random Oracle is replaced by cryptographic hash function
  - Parallel Soundness amplification (multiple MPC simulations in parallel), but omitting in next slides

MPC-in-the-Head Signature Scheme

- Sign(m):
  1. Generate random sharings $[x]$, $\{[t_i]\}$ and random BeaverTriples $[a_i], [b_i], [c_i]$ for n parties
  2. Compute $[y], \{[r_i]\}, \{[s_i]\}$ as linear combinations of $[x], \{[t_i]\}$
  3. Generate commitments of the view of each party $p = 1, \dots, n$:
     - $\left(c_p^{(1)}, o_p^{(1)}\right) \leftarrow \mathrm{Commit}\left([x]_p, \left\{\left([r_i]_p, [s_i]_p, [t_i]_p, [a_i]_p, [b_i]_p, [c_i]_p\right)\right\}, [y]_p\right)$
  4. Query random scalars: $\{\rho_i\} \leftarrow \mathrm{RO}\left(pk, m, \left\{c_p^{(1)}\right\}\right)$
  5. Linearly compute: $\{[\alpha_i] = \rho_i[r_i] + [a_i]\}$, $\{[\beta_i] = [s_i] + [b_i]\}$
  6. "Broadcast" all $[\alpha_i]$ and $[\beta_i]$ shares and compute $\{\alpha_i\}, \{\beta_i\}$
  7. Linearly compute: $[v_i] = \rho_i[t_i] - [c_i] + \alpha_i[b_i] + \beta_i[a_i] - \alpha_i\beta_i$
  8. "Broadcast" all $[v_i]$ shares and compute $\{v_i\}$

  Parallel Multiplication Verification Protocol

  9. Generate commitments of the view of each party $p = 1, \dots, n$:
     - $\left(c_p^{(2)}, o_p^{(2)}\right) \leftarrow \mathrm{Commit}\left(c_p^{(1)}, \left\{\left(\rho_i, [\alpha_i]_p, [\beta_i]_p, [v_i]_p\right)\right\}\right)$
  10. Query to-open views: $u, v \leftarrow \mathrm{RO}\left(pk, m, \left\{c_p^2\right\}\right)$

  Commit to views
  Randomly select to-open views

  11. "Broadcast" all $[y]$ shares and compute y
  12. $\sigma_m = \left(\left\{c_p^{(1)}\right\}, \left\{c_p^{(2)}\right\}, \left\{[\alpha_i]_p\right\}, \left\{[\beta_i]_p\right\}, \left\{[v_i]_p\right\}, \left\{[y]_p\right\}, o_u^{(1)}, o_u^{(2)}, o_v^{(1)}, o_v^{(2)}\right)$

  All commitments + all broadcasts + view openings

MPC-in-the-Head Signature Scheme

- Verify(pk, m, σ):
  1. Let $\sigma = \left(\left\{c_p^{(1)}\right\}, \left\{c_p^{(2)}\right\}, \{[\alpha_i]_p\}, \{[\beta_i]_p\}, \{[v_i]_p\}, \{[y]_p\}, o_u^{(1)}, o_u^{(2)}, o_v^{(1)}, o_v^{(2)}\right)$
  2. Query random scalars: $\{\rho_i\} \leftarrow \mathrm{RO}\left(pk, m, \left\{c_p^{(1)}\right\}\right)$
  3. Query to-open views: $u', v' \leftarrow \mathrm{RO}\left(pk, m, \left\{c_p^2\right\}\right)$
  4. Reconstruct broadcasts: $\{\alpha_i\}, \{\beta_i\}, \{v_i\}, y$
  5. Check if y is correct: $y =? pk$
  6. Check if all multiplication gates are correct: all $v_i =? 0$
  7. Check MPC Simulation was correct:
     1. Verify openings $o_u^{(1)}, o_u^{(2)}, o_v^{(1)}, o_v^{(2)}$ to commitments $c_{u'}^{(1)}, c_{u'}^{(2)}, c_{v'}^{(1)}, c_{v'}^{(2)}$ belonging to parties u', v'
     2. Verify view of each party $p \in \{u', v'\}$:
        1. Extract view $\left([x]_p, \left\{\left([r_i]_p, [s_i]_p, [t_i]_p, [a_i]_p, [b_i]_p, [c_i]_p\right)\right\}, [y]_p\right)$ from openings
        2. Recompute shares $[y]_p, \left\{[r_i]_p\right\}, \left\{[s_i]_p\right\}, \left\{[\alpha_i]_p\right\}, \left\{[\beta_i]_p\right\}, \left\{[v_i]_p\right\}$ linearly as in signature algorithm
        3. Check consistency of recomputed shares with view-opening shares / broadcasted shares

MPC-in-the-Head schemes

- 2017 NIST PQC competition (1 out of 82 submissions):
  - Picnic: LowMC blockcipher (didn't make it beyond round 3)
- 2023 NIST PQC "on-ramp" signature competition (9 out of 40):
  - 1st Round: 9 out of 40
    - FAEST : AES
    - AIMer : AIM tweakable one-way function
    - Biscuit : MQ
    - MQOM : MQ
    - MIRA : MinRank
    - MiRitH : MinRank
    - PERK : Permuted Kernel Problem
    - RYDE : Syndrome Decoding
    - SDitH : Syndrome Decoding
  - 2nd Round: 6 out of 14
    - Merge: MIRA & MiRitH ⇒ Mirath
    - FAEST, MQOM, PERK, RYDE, SDitH
- For example: FAEST (with many optimizations not covered here!)
  - pk,sk: 32B, sig: 4.5KiB

Summary

- MPC
  - Linear Secret Sharing Schemes
  - Multiplication Protocol using BeaverTriples
  - Basic MPC protocol

- MPC-in-the-Head
  - Paradigm of committing to random views of local computations & open selection
  - ZK-PoK for any Circuit evaluation $C(x)$ on secret x
  - Share all multiplication gates → Multiplication Verification Protocol

- Basic MPC-in-the-Head signature scheme description