KU LEUVEN

The ISC Pre-School Workshop
(22 Feb 2023)

NIZK Arguments and MPC-in-the-Head Paradigm

Karim Baghery
Postdoctoral Researcher, KU Leuven, Belgium
karim.baghery@kuleuven.be

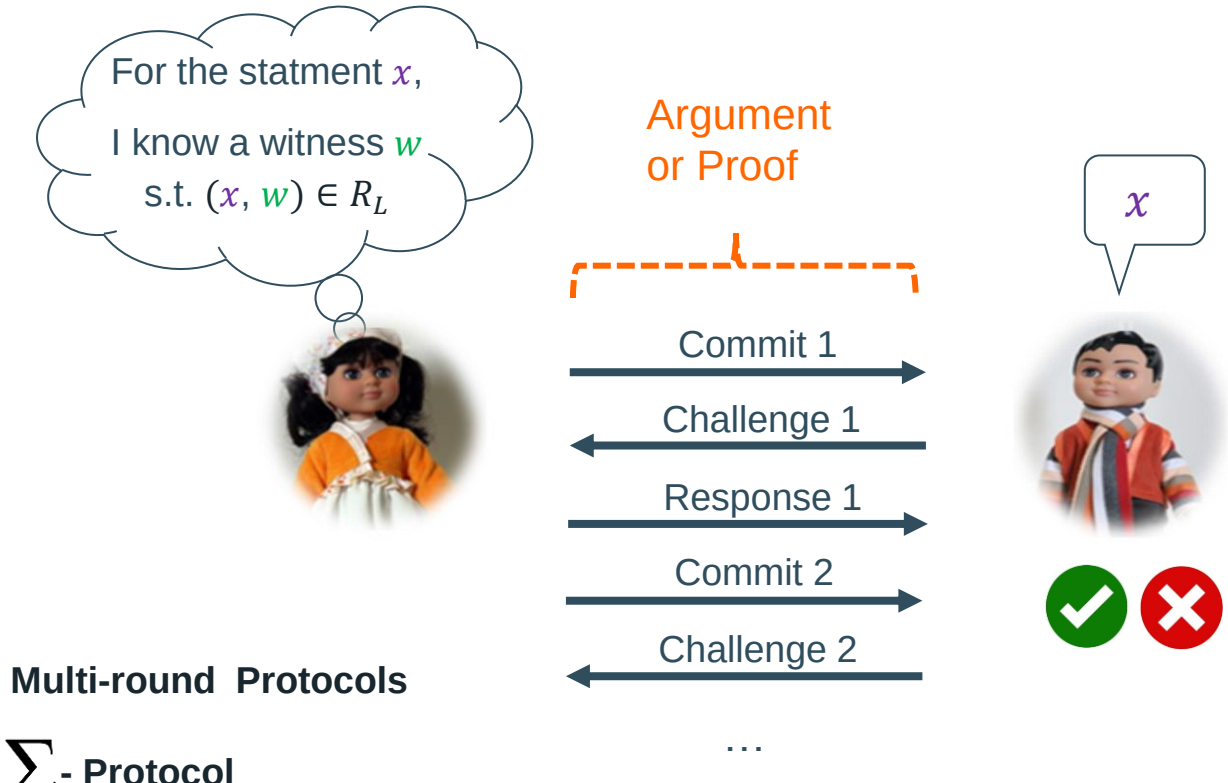
Outline:

- Introduction and NIZK Arguments
  - Proof Systems, Interactive and Non-Interactive
  - Requirements for Proof Systems
  - Constructing NIZK Arguments in the RO and CRS Models
  - NIZKs with Subverted/Updated/Trusted CRS
  - NIZKs with Public/Private Verifiers
  - Two Sample NIZK Arguments in the RO and CRS Models
- MPC-in-the-Head Paradigm
  - Building NIZK Argumenten from MPC Protocols
  - MPC-in-the-Head in the Pre-processing Model
  - A Sample Construction (KKW18)



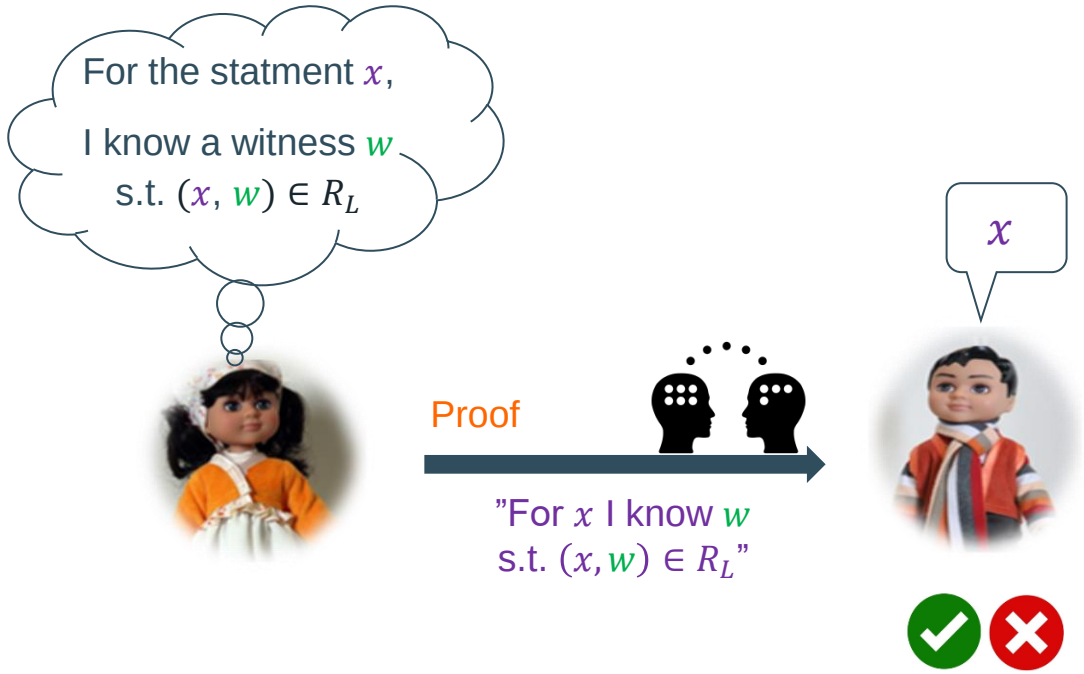
Proof Systems: Interactive or Non-Interactive

- Prover wants to convince Verifier that statement x has some property
  - "x is in language L" or "for (public) x, I know (secret) w s.t. $(x, w) \in R_L$"



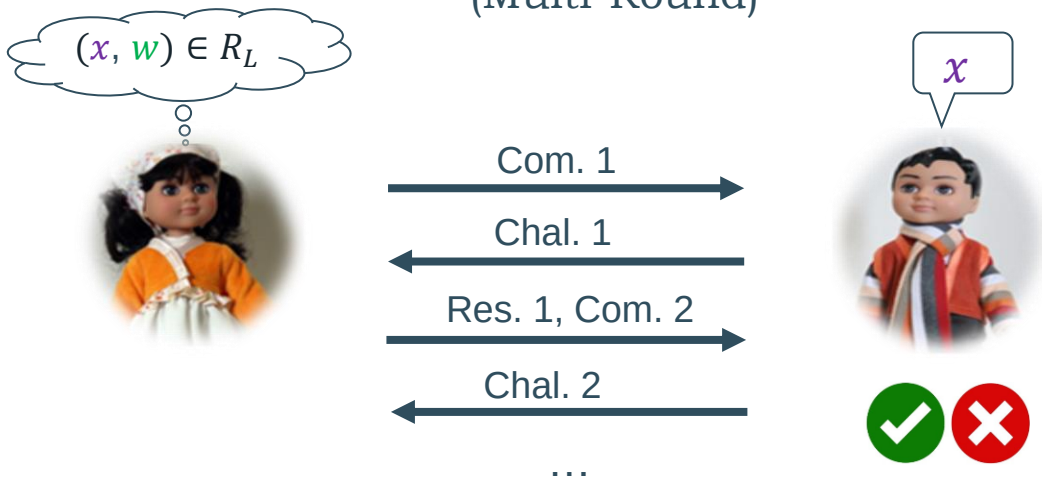
Multi-round Protocols

$\sum$- Protocol



Interactive or Non-Interactive ZK Proofs: Requirements

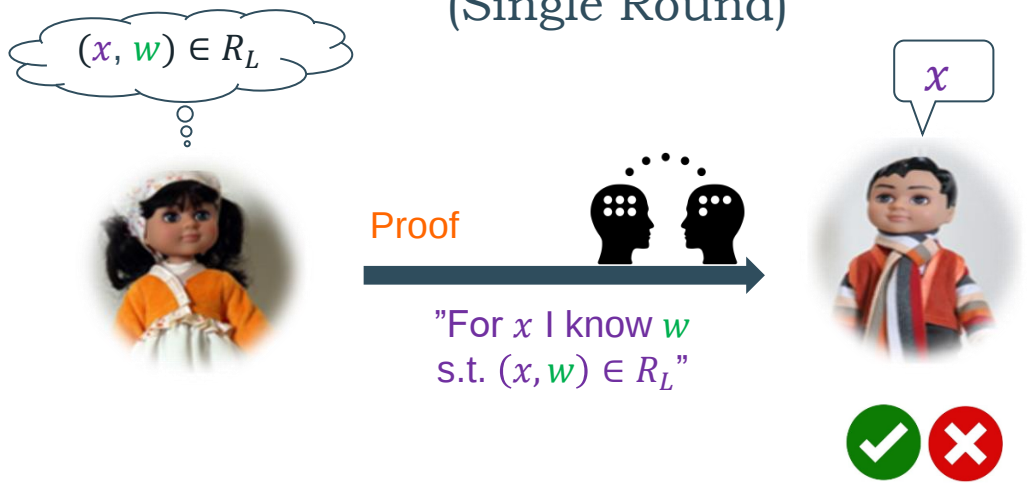
Interactive
(Multi-Round)



- **Completeness**
- **Honest-Verifier Zero-Knowledge (HVZK)**
- **Soundness**
- **K-Special Soundness** (K=2 in ∑-protocols)
- **Simulation Soundness**
- **Simulation Knowledge Soundness (Sim. Extr.)**

Stronger Security

Non-Interactive
(Single Round)



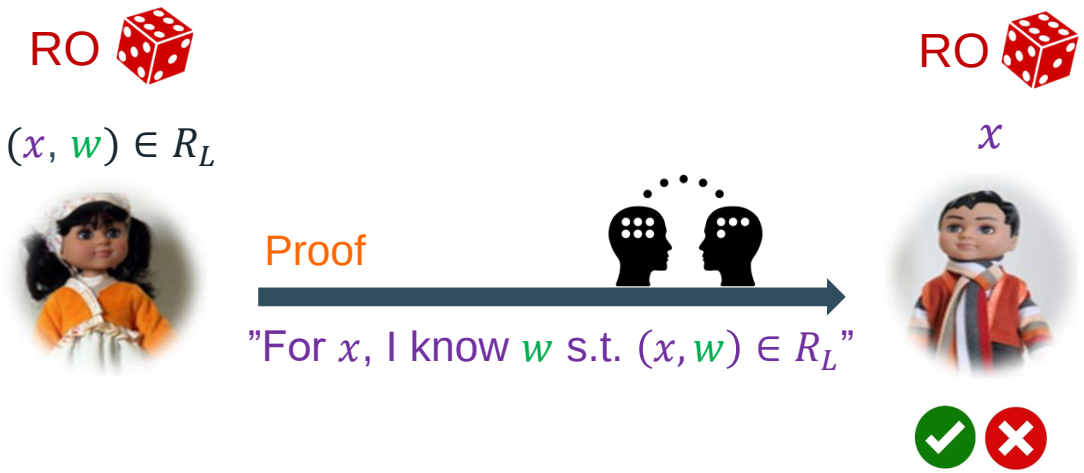
- **Completeness**
- **Zero-Knowledge (ZK) → Witness Ind. (WI)**
- **Soundness**
- **Knowledge Soundness**
- **Simulation Soundness**
- **Simulation Knowledge Soundness**

Constructing Non-Interactive Proof Systems: RO & CRS

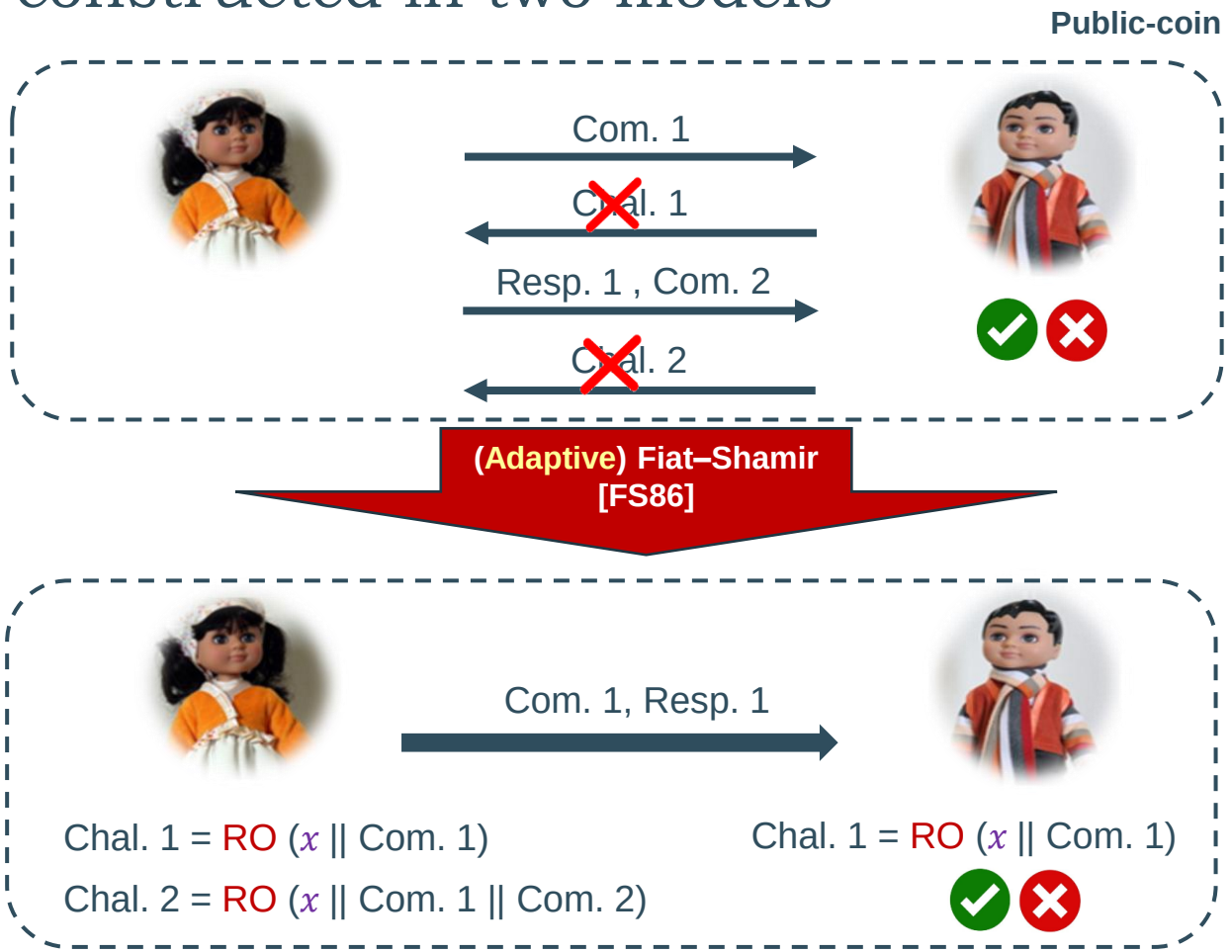
- Non-Interactive proof systems are constructed in two models

Random Oracle (RO) Model

- Parties have access to an RO [BG93, Mic94]



- The ROs do not exist, but in practice they are instantiated with hash functions, e.g. SHA-256, …



Fiat-Shamir Transformation:

- Fiat–Shamir Transformation [FS86]: Parties have access to an RO [BG93, Mic94]

Let $\Pi = (\mathcal{P}, \mathcal{V})$ be a $(2\mu+1)$-move public-coin interactive proof, where the challenge from the i-th round is sampled from set $\mathcal{C}_i$. For simplicity, we consider μ random oracles $\mathsf{RO}_i \colon \{0,1\}^{\leq u} \to \mathcal{C}_i$ that map into the respective challenge spaces.

Fiat-Shamir Transformation. *The static Fiat-Shamir transformation* $\mathsf{FS}[\Pi] = (\mathcal{P}_{\mathsf{fs}}, \mathcal{V}_{\mathsf{fs}})$ *is the NIROP where* $\mathcal{P}_{\mathsf{fs}}^{\mathsf{RO}_1,\ldots,\mathsf{RO}_\mu}(x;w)$ *runs* $\mathcal{P}(x;w)$ *but instead of asking the verifier for the challenge* c_i *on message* a_i*, the challenges are computed as*

$$c_i = \mathsf{RO}_i(a_1, \ldots, a_{i-1}, a_i)\,;$$

the output is then the proof $\pi = (a_1, \ldots, a_{\mu+1})$*. On input a statement* x *and a proof* $\pi = (a_1, \ldots, a_{\mu+1})$*,* $\mathcal{V}_{\mathsf{fs}}^{\mathsf{RO}_1,\ldots,\mathsf{RO}_\mu}(x,\pi)$ *accepts if, for* c_i *as above* $\mathcal{V}$ *accepts the transcript* $(a_1, c_1, \ldots, a_\mu, c_\mu, a_{\mu+1})$ *on input* x*.*

If the challenges are computed as

$$c_i = \mathsf{RO}_i(x, a_1, \ldots, a_{i-1}, a_i)\,;$$

the resulting NIROP is referred to as the adaptive *Fiat-Shamir transformation.*

Remarks on Fiat-Shamir Transformation [FS86]:

- A secure hash-function H is shared with all parties.
- Efficient NIZK arguments that work well in practice
  - E.g., Schnarr's ID protocol
- They are supposed to be secure …
  - Can argue heuristically that they can achieve computational soundness in the RO model [BR93], where the function H is a truly random function
  - But in real life H is a deterministic function and there are instantiations of the Fiat-Shamir heuristic [GK03] that yields insecure real-life schemes.

Security Proofs in Quantum RO Model:

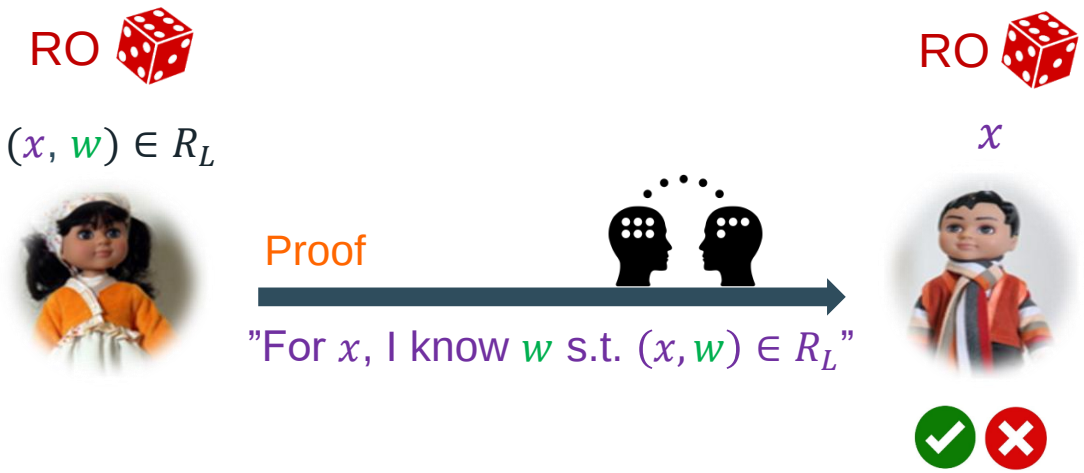
- Truly random function that is public → …
  - **Superposition access**
- Usually the list of input/output queries to RO is needed in the security proof
  - **Is not straightforward because of no-cloning (measurement might disturb the state)**
- Usually adaptively program RO in the proof
  - **A superposition query may contain all the inputs/outputs**

Constructing Non-Interactive Proof Systems: RO & CRS

- Non-Interactive proof systems are constructed in two models

Random Oracle (RO) Model

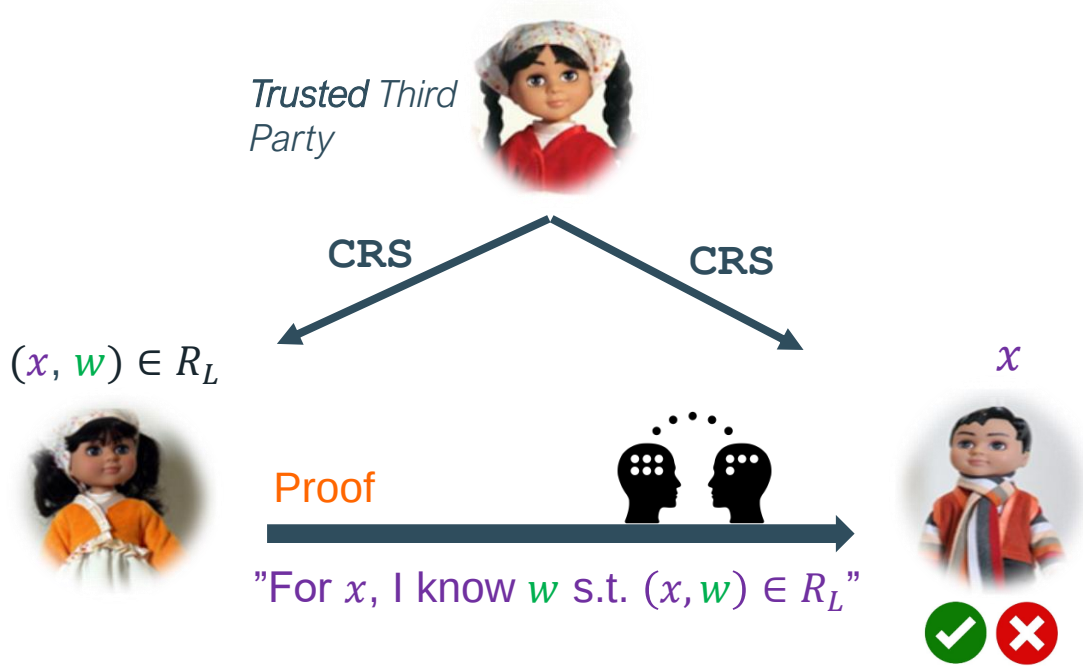
- Parties have access to an RO [BG93, Mic94]



- The ROs do not exist, but in practice they are instantiated with hash functions, e.g. SHA-256, …

Common Reference String (CRS) Model

- Trusted Third Party generates a CRS [BFM88]



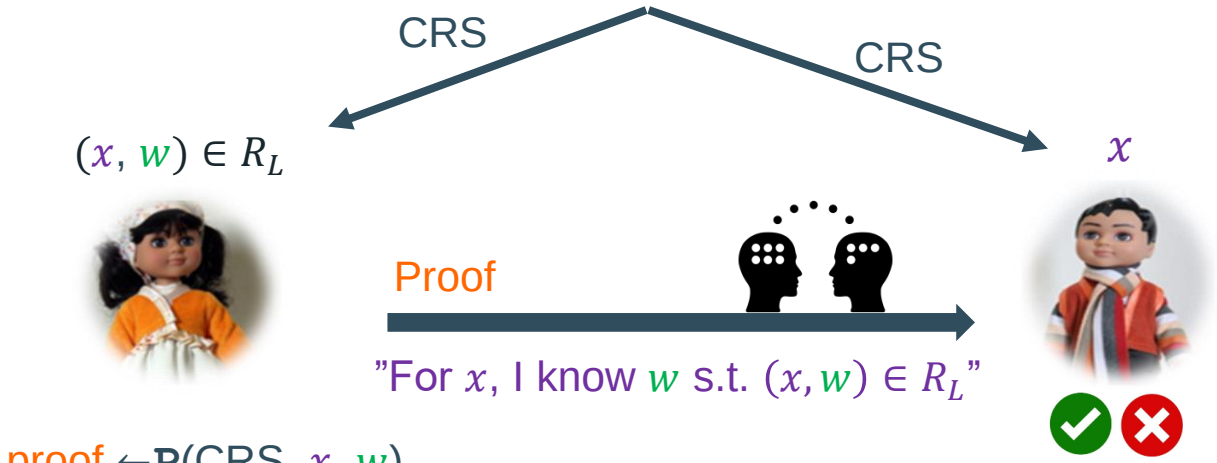
NIZKs in the CRS Model: Constructions

- **CRS:** Can be uniformly random or specific distribution
  - e.g., CRS := 10011101010101011
  - or non-universal CRS := $\left\{\left(g_1^{x^i}, g_2^{x^i}\right)_{i=1}^{d}, g_1^{\frac{\alpha u(x)+\beta v(x)+w(x)}{\gamma}}\right\}$
  - or universal (SRS or) CRS := (g, g^x) or $\left(g_1^{x^i}, g_2^{x^i}\right)_{i=1}^{d}$
- **CRS Generation:** Can be
  - done by a Trusted Third Party (TTP) [Gro16, …]
  - done by a secure MPC protocol [BCG+14, ABL+19]
  - in multi-string model (where majority of strings are honest) [GO07]
  - done by a non-trusted party (e.g., by Verifier) [BFS16, ABLZ17, Bag19]
  - updatable (MPC with dynamic parties) [GKM+18, ARS20, BS21, BB22]
  - Trusted hardware (e.g., Intel SGX)
  - …

- Common Reference String (CRS) Model
  - A trusted CRS is shared among the parties [BFM88]

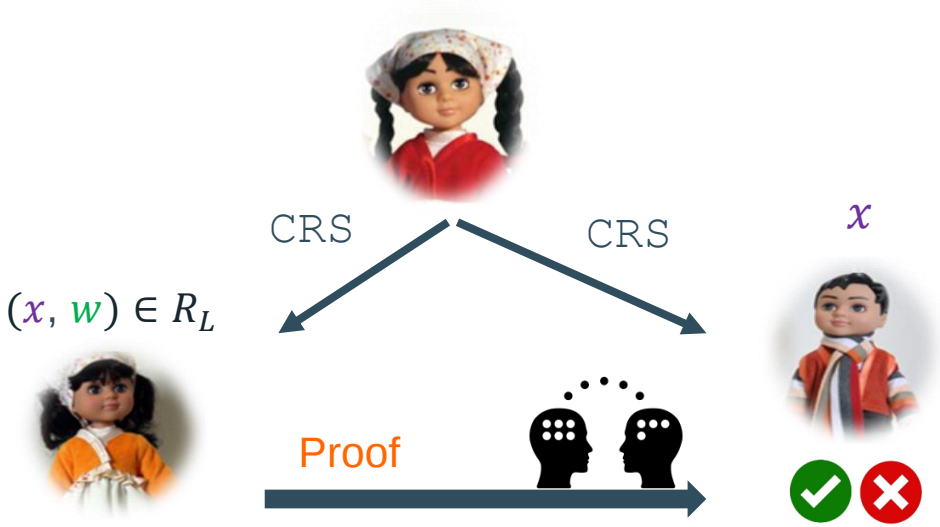


CRS ←**CRSGen**($1^\lambda, R_L$)



NIZKs with (Subverted/Updated/Trusted) CRS: Requirements

- **CRS Generation:** Can be
  - done by a Trusted Third Party (TTP) [Gro16, …]
  - done by a secure MPC protocol [BCG+14, ABL+19]
  - **done by a non-trusted party (e.g., by Verifier)** [BFS16, ABLZ17, Bag19]
  - **updatable (MPC with dynamic parties)** [GKM+18, ARS20, BS21, BB22]
  - …



- ✓ **Sub-Completeness**
- ✓ **Sub-Zero-Knowledge (Sub-ZK)**
- ✓ **Sub-Soundness**
- ✓ **Sub-Knowledge Soundness**
- ✓ **Sub-Simulation Soundness**
- ✓ **Sub-Simulation Extractability**

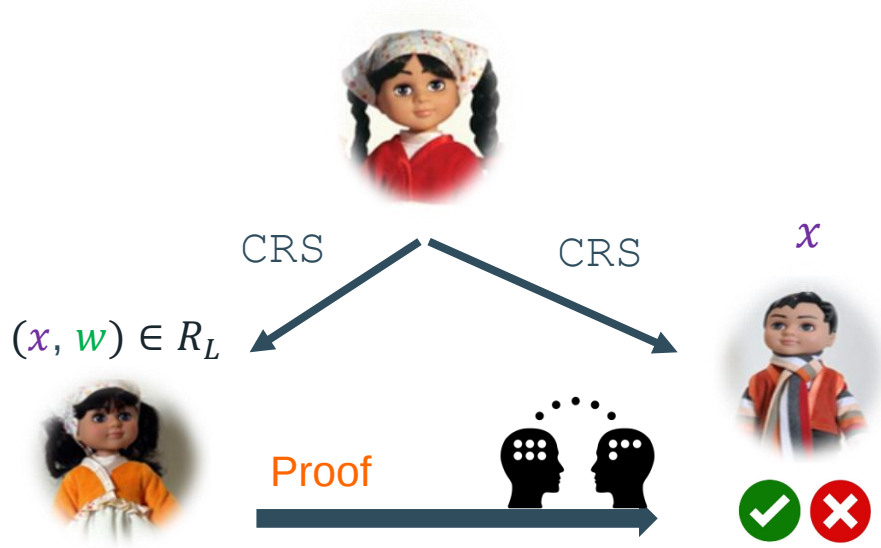
- ➢ **Upd-Completeness**
- ➢ **Upd-Zero-Knowledge (Upd-ZK)**
- ➢ **Upd-Soundness**
- ➢ **Upd-Knowledge Soundness**
- ➢ **Upd-Simulation Soundness**
- ➢ **Upd-Simulation Extractability**

- **Completeness**
- **Zero-Knowledge (ZK)**
- **Soundness**
- **Knowledge Soundness**
- **Simulation Soundness**
- **Simulation Extractability**

NIZK Arguments with (Public/Designated) Verifiers:

- **NIZKs with Public Verifiers**
  - **CRSGen** generates CRS using secret information $trap$
  - Verifier **V** checks proof given CRS, statement and Proof
  - E.g., CRS= (g_1, g_2^a) , $x = g_1^b$ and Proof = g_2^{ab} V checks if $e(g_1, g_2^{ab}) =? e(g_1^b, g_2^a)$

- **NIZKs with Designated Verifiers (or Private Verifiers)**
  - **CRSGen** generates CRS using secret information $trap$
  - Verifier **V** checks proof given $trap$, statement and Proof
  - E.g., for $trap = sk$, Proof = (A, B, C), check if Dec(A, sk) = Dec(B, sk) Dec (C, sk)

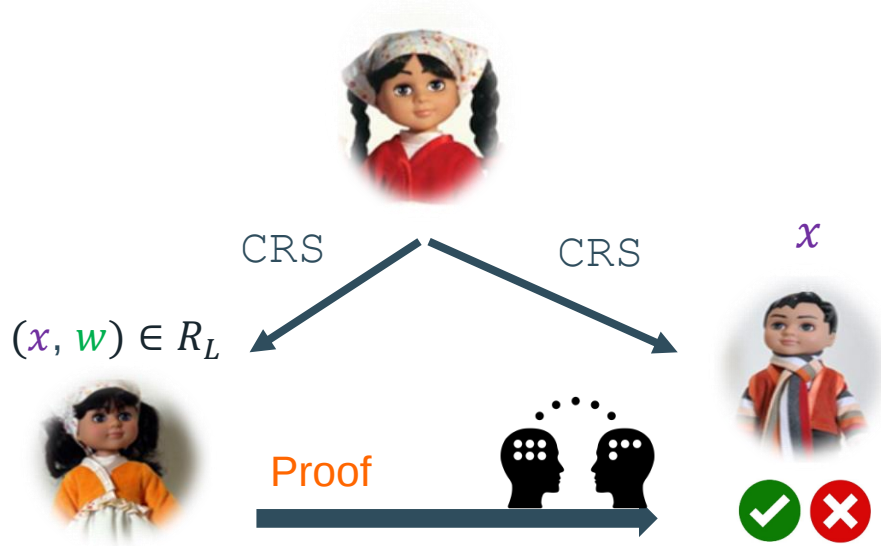


- **Public Verifiability**
  - **Sometimes required:** Signatures, Privacy-preserving coins, …
  - **Reusability:** Prover only needs to generate a single proof, Proof can be sent to different verifiers, … ☺
  - **Hard to construct** 😐

- **Private Verifiability**
  - **Sometimes suffices:** Outsourcing computation, …
  - **Cannot be transferred:** Prover needs to generate separate proofs for each verifier, … ☹
  - **Easier to construct:** Verifier gets additional secret information, … ☺

ZK Proofs vs ZK Arguments:

- **Proofs**
  - Perfect soundness
  - No unbounded adversary can prove a false statement
  - A proof system cannot achieve Perfect Soundness and Perfect ZK at the same time! Therefore, Proofs can only have computational ZK.
  - E.g., Groth-Ostrovsky-Sahai 2012 (2006) NIZK **Proofs** can achieve Perfect Soundness and Computational ZK (& Perfect Completeness).

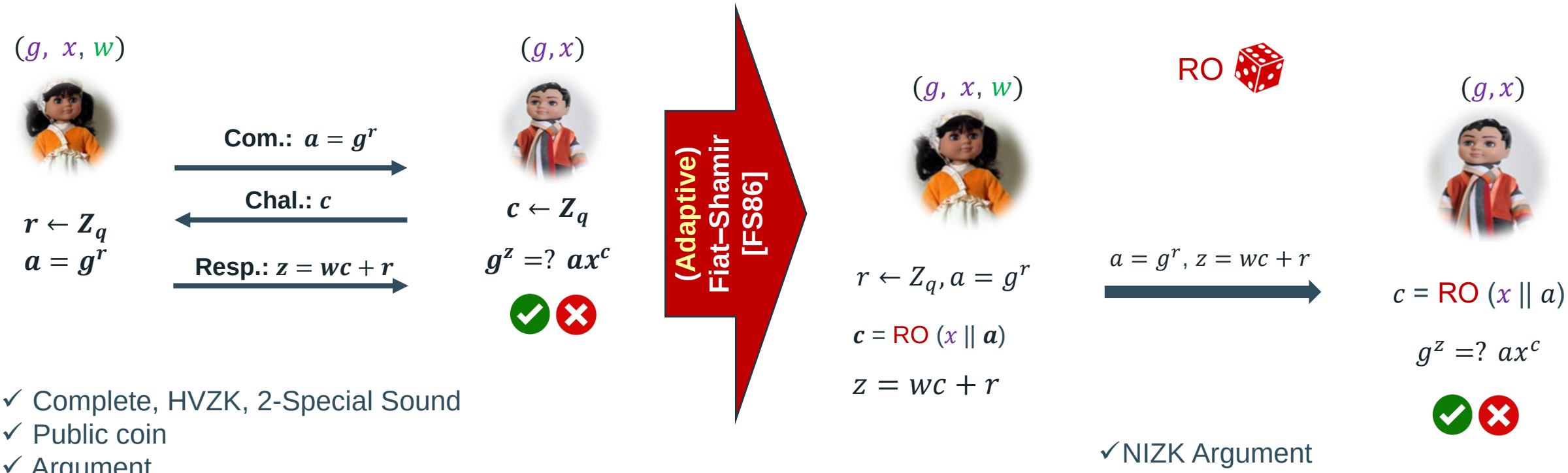


- **Arguments**
  - Computational soundness
  - No probabilistic polynomial time adversary can prove a false statement
  - Arguments may achieve perfect ZK.
  - E.g., pairing-based zero-knowledge Succinct Non-interactive **Arguments** of Knowledge (zk-SN**A**RK), e.g. [Gro16], can achieve Perfect ZK and Computational Soundness.

Will see more about NIZK Arguments in the main event!

A NIZK Argument in the RO Model: Schnarr's ID Protocol

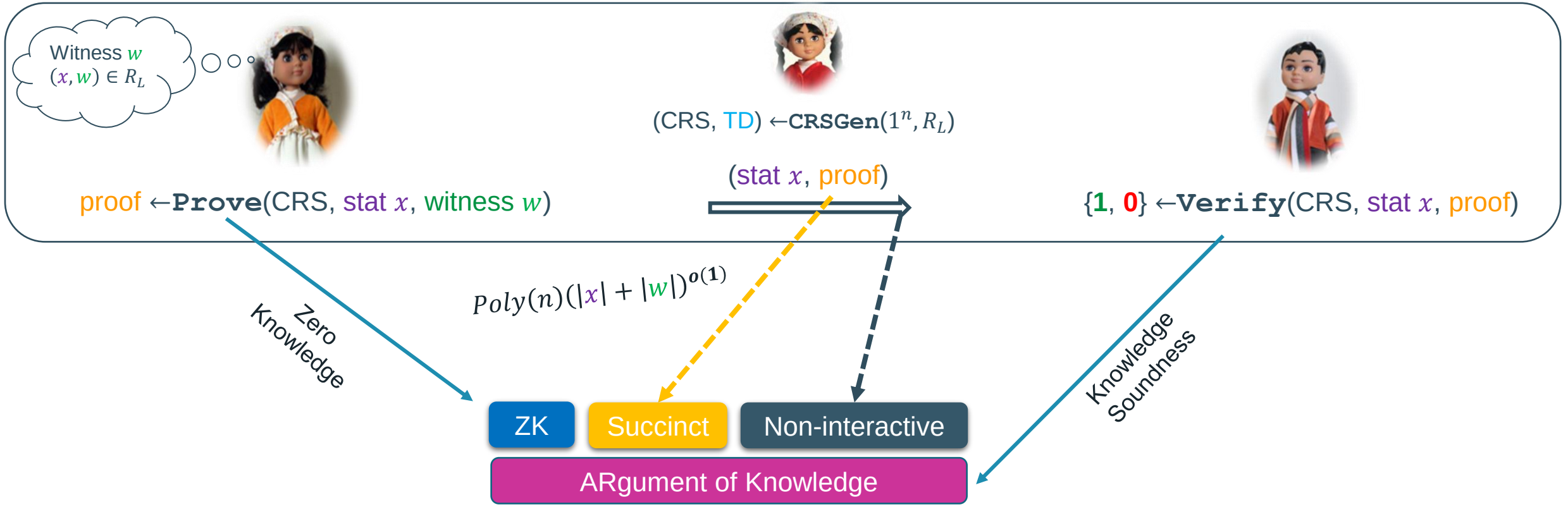
- Let p be some prime number, and g be a generator of a cyclic group of prime-order q which solving the DL problem is computationally hard.
- Given g, P wants to convince a V that she knows a witness w s.t. $x = g^w$.



✓ Complete, HVZK, 2-Special Sound
✓ Public coin
✓ Argument

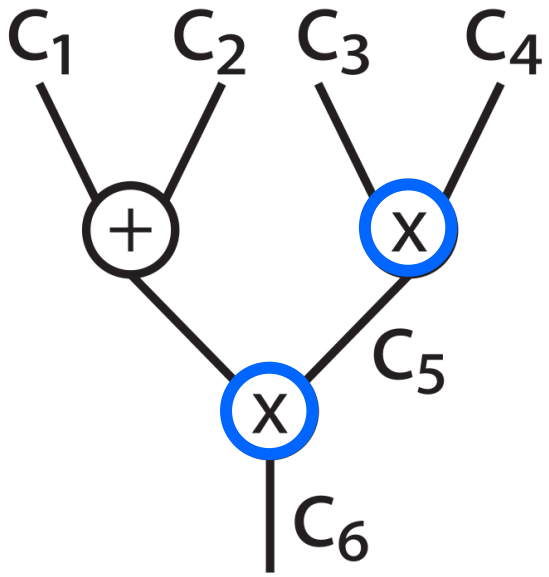
✓NIZK Argument

NIZKs in the CRS Model: zk-SNARKs



ZK-SNARK: [Gro10, Lip12, PHGR13, BCTV14] → **[Gro16]**

Quadratic Arithmetic Programs (QAPs): Main Intuition



$C_3 * C_4 == C_5$

$(C_1 + C_2) * C_5 == C_6$

⋮

- Construct polynomials $T(z) = (z - r_5)(z - r_6)$ and $P(z) = (\sum c_i v_i(z))(\sum c_i w_i(z)) - (\sum c_i y_i(z))$ that encode gate equations and wire values $\{c_i\}$
- $(c_1, \ldots, c_m)$ is a valid set of wire values iff:

$$T(z) \text{ divides } P(z)$$

$$\equiv$$

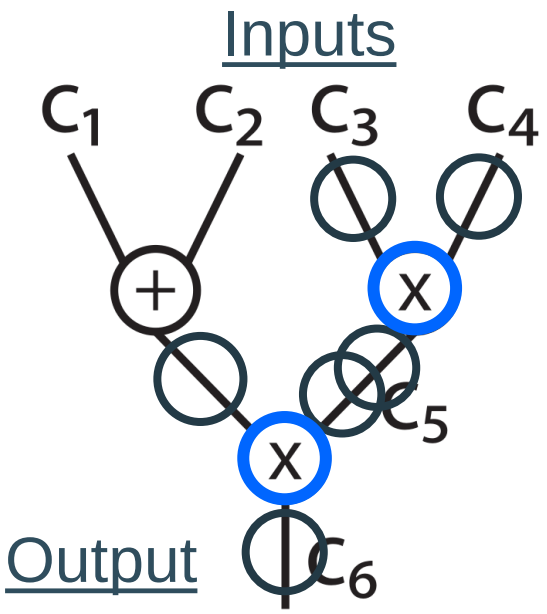
$$\exists\, H(z):\ H(z) \cdot T(z) == P(z)$$

$$\equiv$$

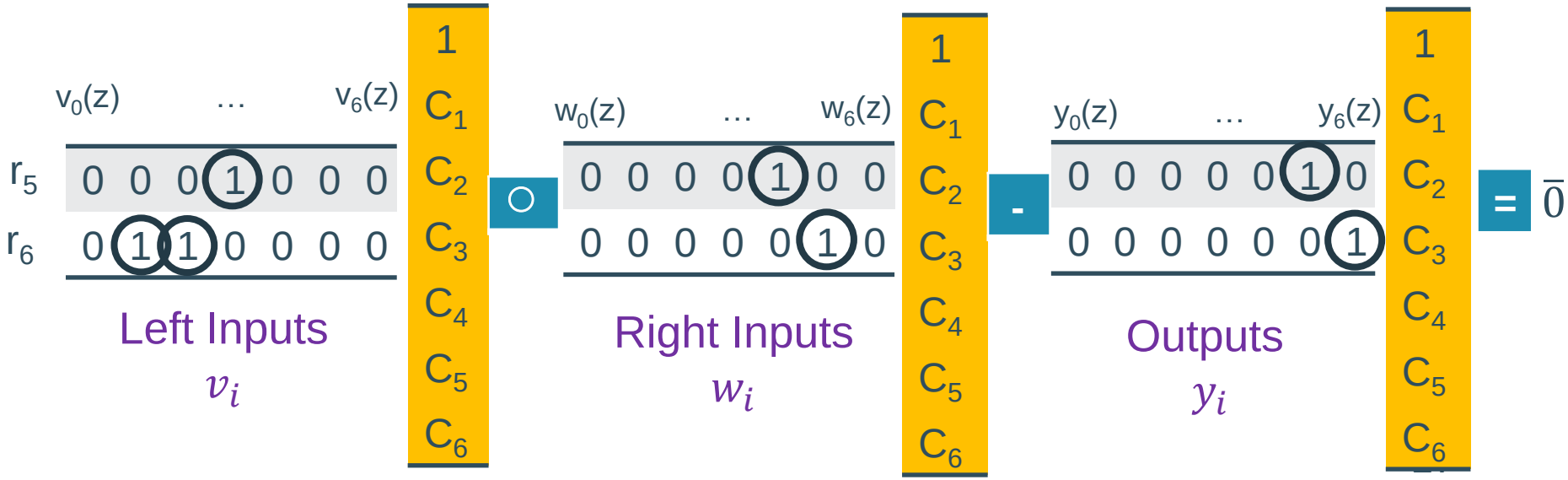
$$\forall\, r_i : T(r_i) == 0 \ \Rightarrow\ P(r_i) == 0$$

QAP-based protocols check divisibility at a random point, and hence cheaply check the correctness

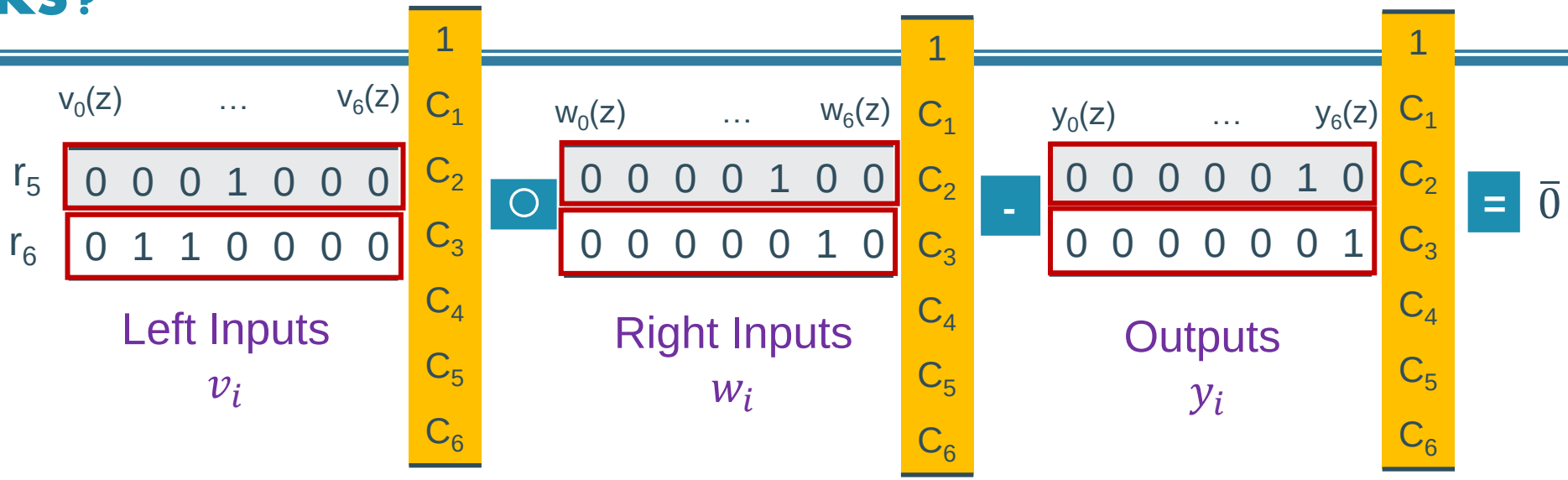
Converting Arithmetic Circuit to QAPs:

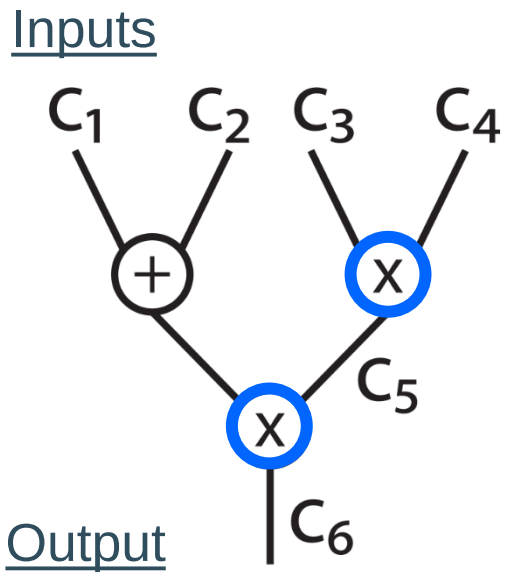


- Pick arbitrary root for each (x) : r_5 , r_6 from $\mathbb{F}$
- Define: $T(z) = (z - r_5)(z - r_6)$
- Define P(z) via three sets of polynomials: $\{v_0(z), ..., v_6(z)\}$ $\{w_0(z), ..., w_6(z)\}$ $\{y_0(z), ..., y_6(z)\}$ where $P(z) = (\sum c_i v_i(z))(\sum c_i w_i(z)) - (\sum c_i y_i(z))$



Why it works?





- Based on definitions:
  $T(z) = (z - r_5)(z - r_6)$ & $P(z) = (\sum c_i v_i(z))(\sum c_i w_i(z)) - (\sum c_i y_i(z))$
- $T(z)$ divides $P(z)$ means:
  $\forall\, r_i : T(r_i) == 0 \;\Rightarrow\; P(r_i) == 0$

$T(r_5) = 0$ $\quad$ $P(r_5) = (c_3)(c_4) - (c_5)$

$T(r_6) = 0$ $\quad$ $P(r_6) = (c_1 + c_2)(c_5) - (c_6)$

Proving an Instance of QAPs: Using Pinocchio zk-SNARK



CRSGen(F) ➡ CRS_P , CRS_V

1. Generate the QAP for y $= F(x)$,
   namely, $v_1(z)$, …, $y_1(z)$, ..., $w_1(z)$, ... and $T(z) = \prod(z - r_i)$
2. Pick random value s
3. Compute CRS_P = {$g^{v_1(s)}$, …, $g^{v_m(s)}$, $g^{w_1(s)}$, …, $g^{w_m(s)}$, $g^{y_1(s)}$, …, $g^{y_m(s)}$, g^{s^i}}
4. Compute CRS_V = {$g^{T(s)}$}
5. Set CRS := (CRS_P,CRS_V)



Prove(CRS_P, x, y) ➡ Π_y

1. Evaluate circuit. Get wire values c_1,…,c_m
2. Compute: $g^{v(s)} = \prod(g^{v_i(s)})^{c_i}$,
   $g^{w(s)} = \prod(g^{w_i(s)})^{c_i}$, $g^{y(s)} = \prod(g^{y_i(s)})^{c_i}$
3. Find H(z) s.t. H(z)*T(z) = V(z)*W(z)-Y(z)
   $= \sum_{i=1}^{d} h_i \,.\, z^i$
4. Compute $g^{H(s)} = \prod \left(g^{s^i}\right)^{h_i}$
5. Proof is ($g^{v(s)}$, $g^{w(s)}$, $g^{y(s)}$, $g^{H(s)}$)



Verify(CRS_V, y, Π_y) ➡ {Yes, No}

Check: $e(g^{v(s)}, g^{w(s)})/e(g^{y(s)}, g) =?= e(g^{H(s)}, g^{T(s)})$

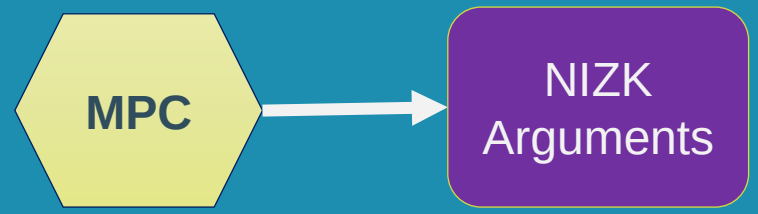
e(·, ·) is a pairing: $e(g^a, g^b) == e(g, g)^{ab}$



MPC-in-the-Head:

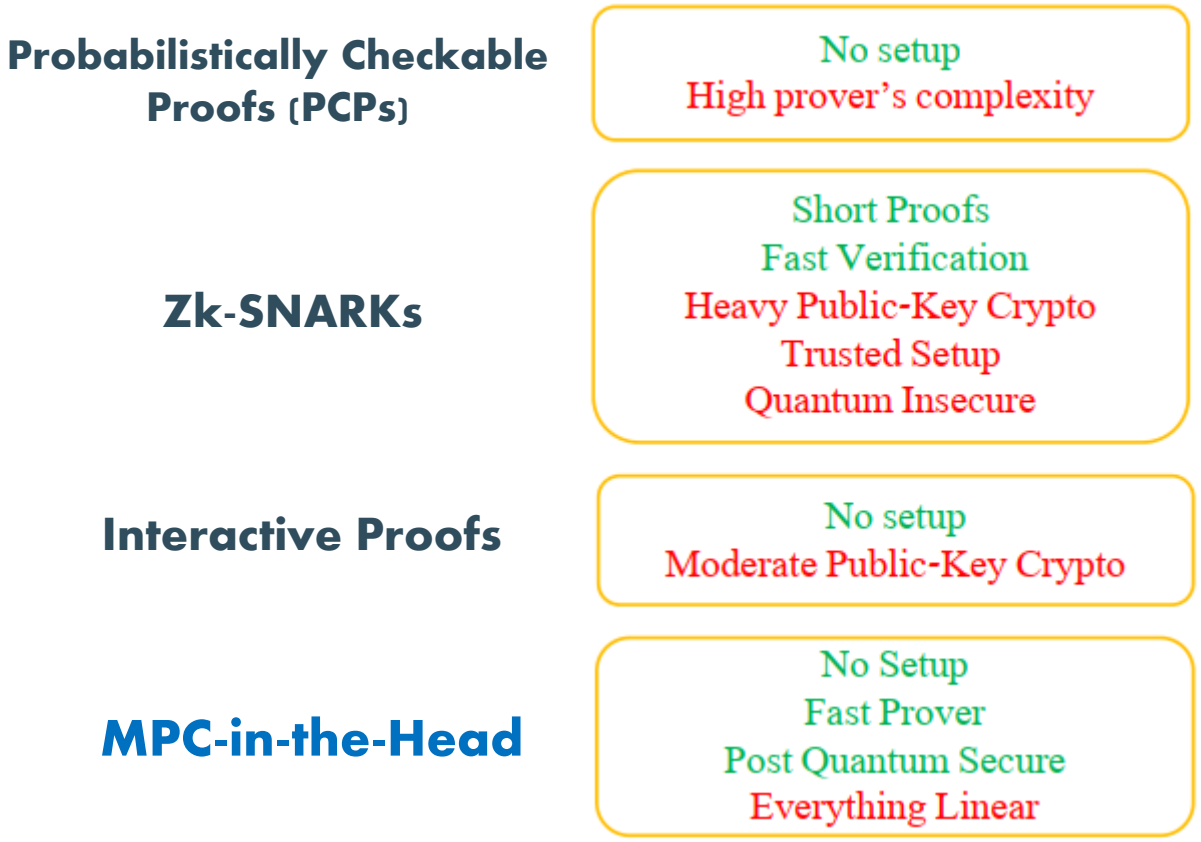
NIZK Arguments from MPC Protocols

[Ishai, Kushilevitz, Ostrovsky, Sahai, 2007]
[GMO16, CDG+17, AHIV17, KKW18]



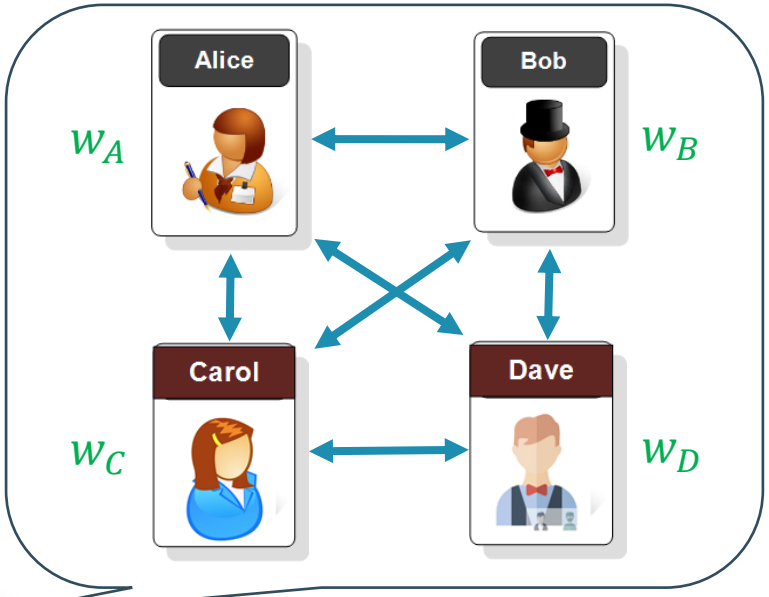
Taxonomy of Proofs:

- Interactive vs Non-interactive
- Trusted setup vs Updatable Setup vs No setup (transparent)
- ZK vs (only) Soundness
- Succinct vs Non-succinct Proofs
- Prover: O(n) vs O(n log n)
- Public-Key Crypto vs (only) Symmetric-Key Crypto



MPC-in-the-Head:

- A compiler which allows to build zk-PCP from MPC
  - Honest majority, information-theoretic secure MPC
  - Completeness, Soundness and ZK derived from security of MPC protocol









Efficient MPCitH-based Protocols:

- ZKBoo (Giacomelli, Madsen, Orlandi, USENIX Security 2016)
- ZKB++ (Chase, Derler, Goldfeder, Orlandi, Ramacher, Slamanig, Zaverucha, ACM CCS 2017)
- KKW (Katz, Kolesnikov, Wang, ACM CCS 2018)
- BN (Baum, Nof, PKC 2020)
- Ligero, Ligero ++ (Ames, Hazay, Ishai, Venkitasubramaniam, ACM CCS 2017, 2020)
- BooLigero (Yaron Gvili, Sarah Scheffler, Mayank Varia, FC 2021):
- Limbo (Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, Titouan Tanguy, ACM 2021)
- …

Picnic Signature

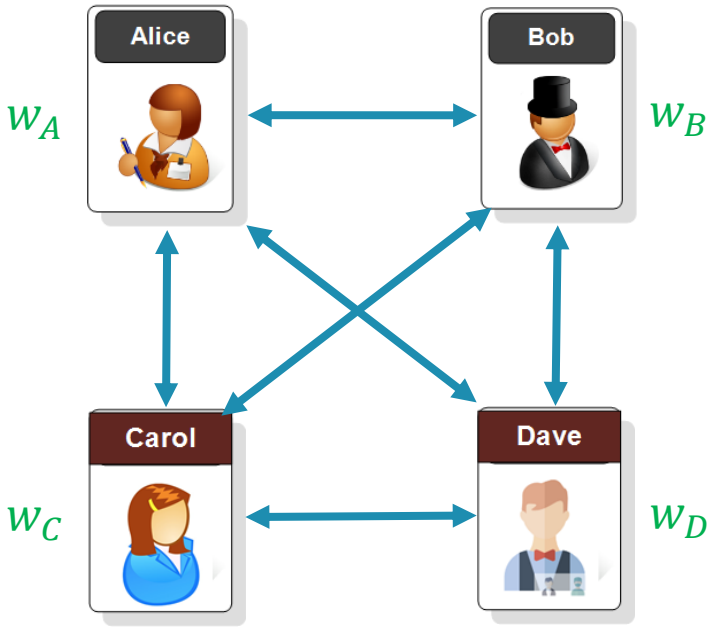
MPC-in-the-Head: Building NIZKs from MPC [IKOS07]

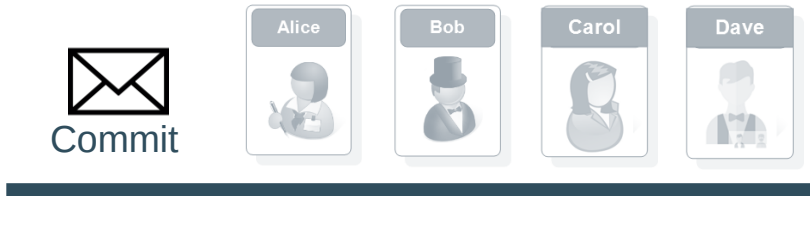
"I know w s.t. C(w)=1"

C(w)=1



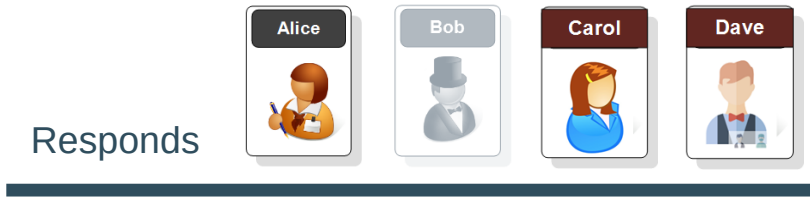
$w = w_A + w_B + w_C + w_D$







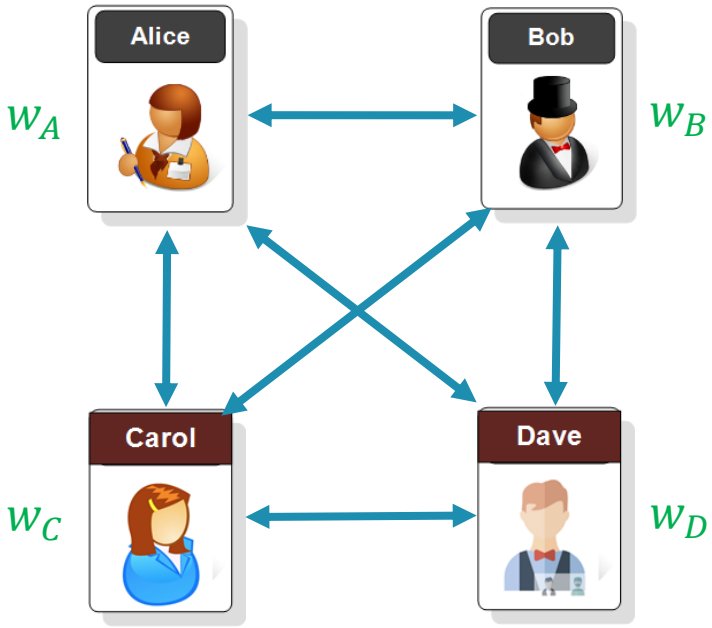




- Verifier decides to accept/reject
  - Openings are consistent with the correct execution of the protocol
  - Views of the parties are consistent with each other

MPC-in-the-Head: Building NIZKs from MPC Protocols

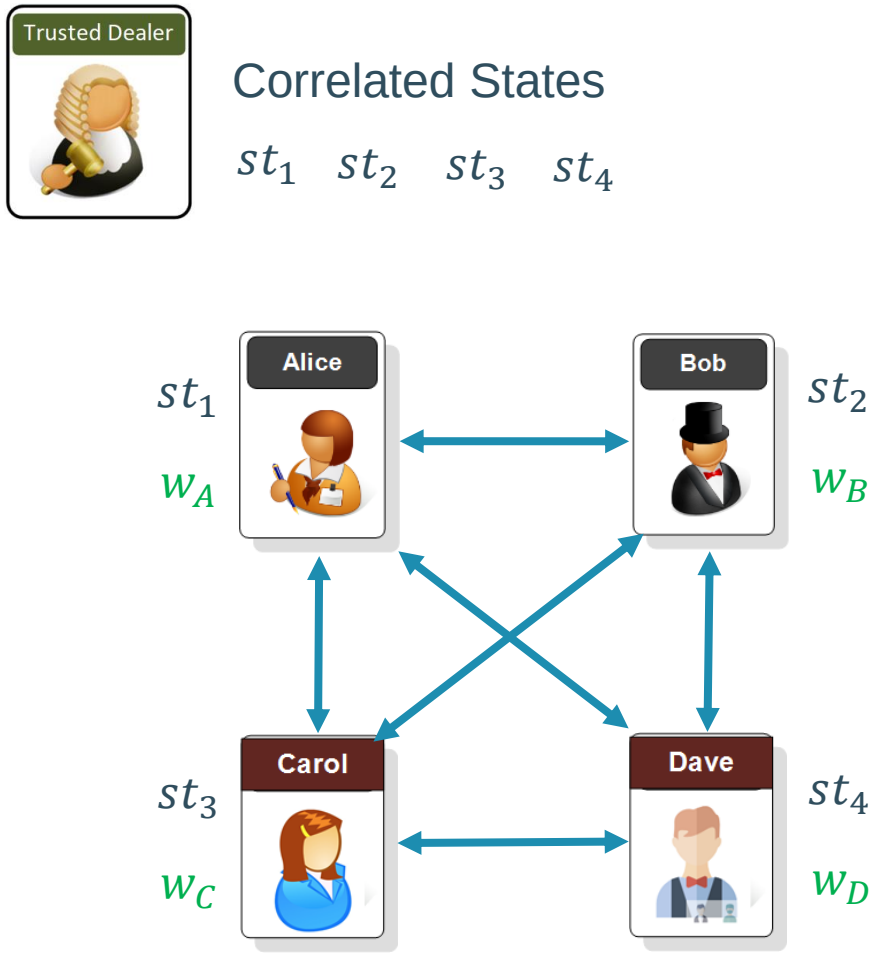
$w = w_A + w_B + w_C + \mathrm{w_D}$



- Semi-honest security is sufficient
- Privacy against t corrupted parties → ZK if $\leq t$ parties opened
- Soundness:
  - Prover needs to cheat on views of at least a single pair of parties
  - Probability of cheating for $t = n - 1$ is about $\approx \frac{2}{n}$
  - Seems *larger* n gives better soundness error rate for *each repetition* (in ZKBoo, and ZKB++ $n = 3$, so soundness error is $\frac{2}{3}$ per each execution)

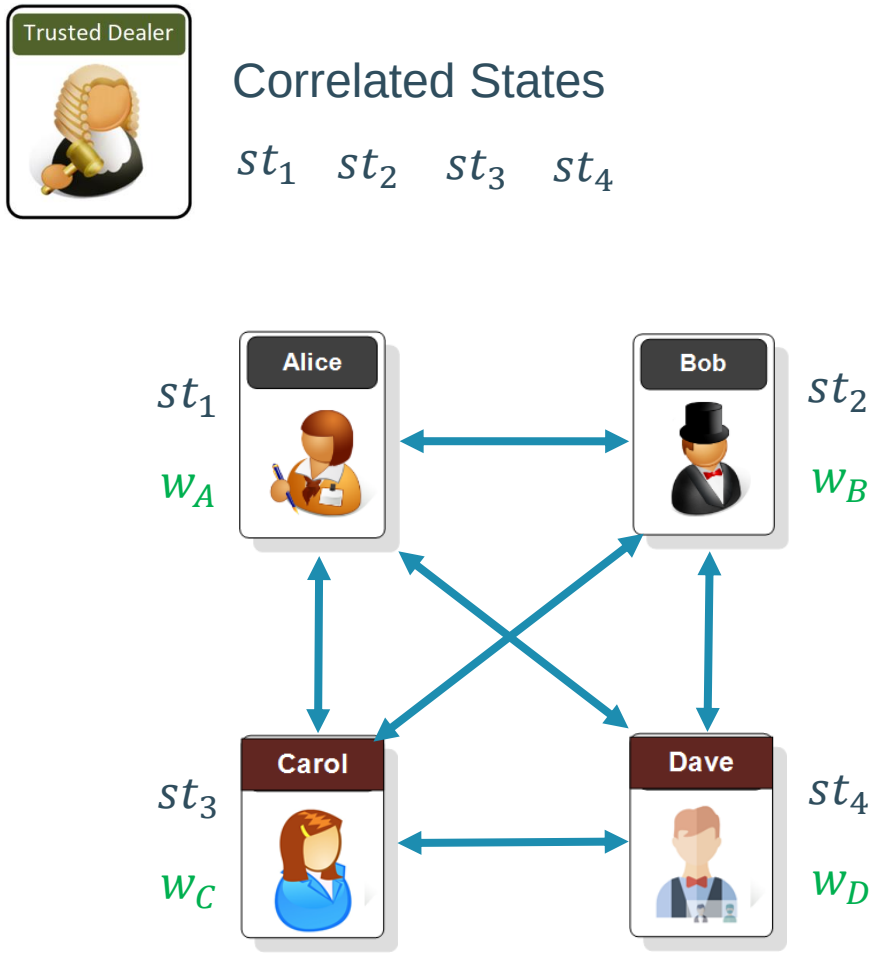
MPC-in-the-Head in the Pre-Processing Model: KKW18

- **New approach for instantiating the "MPC-in-the-Head" paradigm**
  - Using MPC Protocols in the pre-processing model
    - There is a trusted dealer who initialized all the parties with some correlated state that they can later use in the online phase of the secure computation protocol
  - Allows to achieve better soundness per execution → better efficiency overall
  - Optimize the efficacy of the approach and minimize the proof size (commitments, …)
  - Application to post-quantum secure signatures
    - Standard signature as in ZKB++
    - Group/ring signatures



MPC-in-the-Head in the Pre-Processing Model: KKW18

- **Need to check if MPC protocols with the pre-processing model are compatible with the "MPC-in-the-Head" approach**
  - Prover might cheat by generating incorrect states during the pre-processing phase
  - If you open all, then will violate the ZK
  - Opening t statment might not be enough to determine if the provier is cheated or not
- **They revise the approach of MPC-in-the-Head paradigm into a two-stage protocol now**



KKW18: Two Stages MPC-in-the-Head



Correlated States

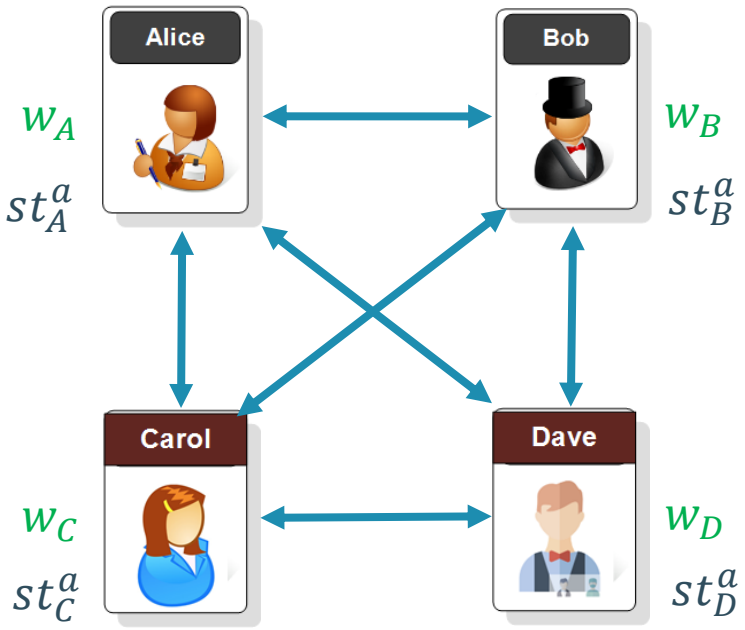
$st_i^a \quad st_i^b \quad st_i^c \quad st_i^d$

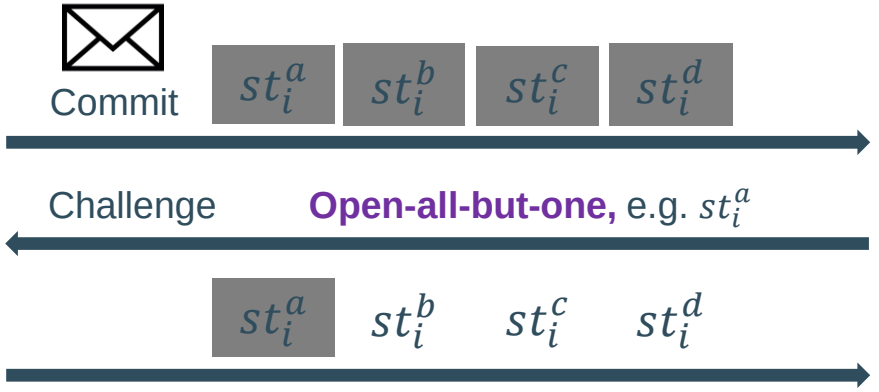
"I know w s.t. C(w)=1"

C(w)=1



$w = w_A + w_B + w_C + w_D$









- Verifier decides to accept/reject

KKW18: ZK & Soundness

- Zero-knowledge is achieved as
  - Opening all parties' state from pre-processing reveals nothing about the input
  - As before, opening t parties in the second stage is okay
- (Special) Soundness
  - Assuming opening all-but-one, a malicious prover can cheat in the first stage with probability at most $\frac{1}{M}$
  - If pre-processing used is *correct*, soundness is as before

KKW18: 5-round protocol

- **Round 1:** For each $j \in [m]$, the prover emulates the preprocessing phase
- **Round 2:** The verifier chooses uniform $c \in [m]$ and sends it to the prover
- **Round 3:** The prover sends $\{\text{seed}_j^*\}_{j \neq c}$ to the verifier and also simulates the online phase of the n-party protocol using the states generated with seed_c^*, …
- **Round 4:** The verifier chooses uniform $p \in [n]$ and sends it to the prover
- **Round 5:** The prover sends openings and final response based on p

KKW18: By Now

”I know *w* s.t. C(w)=1”

C(w)=1



1st Stage

Commit st_i^a st_i^b st_i^c st_i^d

Challenge **Open-all-but-one,** e.g. st_i^a

st_i^a st_i^b st_i^c st_i^d



2st Stage

Commit st_i^c Bob st_i^d Dave

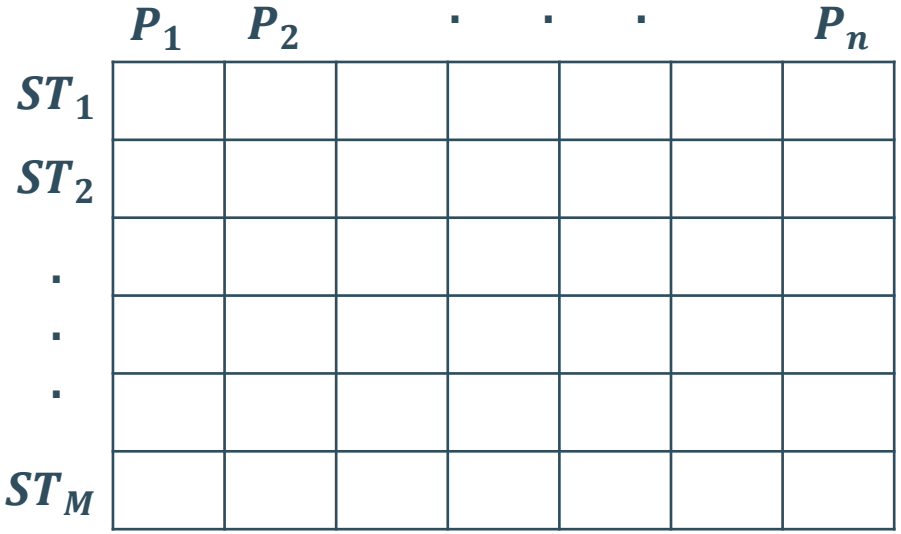
Challenge **Dave**, **Alice**, **Carle**

Responds Alice Bob Carol Dave $st_{A,C,D}^a$

- Verifier decides to accept/reject

KKW18: Reducing Rounds

- The main protocol is a 5-round protocol
  - Fiat-Shamir can be applied on 0(1)-round public coin protocols (but concreate security can suffer, due to using the forking lemma, …)
- Can compress to a 3-round protocol
  - Roughly the same communication; constant-factor increase in prover computation



KKW18: 3-round protocol

- **Round 1:** For each $j \in [m]$, the prover emulates the preprocessing phase and simulates the online phase of the n-party protocol using the states $\{state_{i,j}\}_i$, …
- **Round 2:** The verifier chooses uniform τ-size $\vec{c} \in [m]$ and a uniform $\vec{P} = \{p_j\}_{j\in[\tau]}$ where each $p_j \in [n]$ and sends $(\vec{c}, \vec{P})$
- **Round 3:** For each $j \in [M]\backslash\vec{c}$ the prover does opening of pre-processing and for each $j \in \vec{c}$ does the opening for parties except p_j

A 3-round HVZK proof

Inputs: Both parties have a circuit C; the prover also holds w with $C(w) = 1$. Values M, n, τ are parameters of the protocol.

Round 1 For each $j \in [M]$, the prover does:

1. Choose uniform $\mathsf{seed}_j^* \in \{0,1\}^\kappa$ and use it to generate values $\mathsf{seed}_{j,1}, r_{j,1}, \ldots, \mathsf{seed}_{j,n}, r_{j,n}$. Also compute $\mathsf{aux}_j \in \{0,1\}^{|C|}$ as described in the text. For $i = 1, \ldots, n-1$, let $\mathsf{state}_{j,i} := \mathsf{seed}_{j,i}$; let $\mathsf{state}_{j,n} := \mathsf{seed}_{j,n} \| \mathsf{aux}_j$.
2. For $i \in [n]$, compute $\mathsf{com}_{j,i} := \mathsf{Com}(\mathsf{state}_{j,i}; r_{j,i})$.
3. The prover simulates the online phase of the n-party protocol Π (as described in the text) using $\{\mathsf{state}_{j,i}\}_i$, beginning by computing the masked inputs $\{\hat{z}_{j,\alpha}\}$ (based on w and the $\{\lambda_{j,\alpha}\}$ defined by the preprocessing). Let $\mathsf{msgs}_{j,i}$ denote the messages broadcast by S_i in this protocol execution.
4. Let $h_j := H(\mathsf{com}_{j,1}, \ldots, \mathsf{com}_{j,n})$ and $h'_j := H(\{\hat{z}_{j,\alpha}\}, \mathsf{msgs}_{j,1}, \ldots, \mathsf{msgs}_{j,n})$.

Compute $h := H(h_1, \ldots, h_M)$ and $h' := H(h'_1, \ldots, h'_M)$ and send $h^* := H(h, h')$ to the verifier.

Round 2 The verifier chooses a uniform τ-sized set $\mathcal{C} \subset [M]$ and $\mathcal{P} = \{p_j\}_{j\in[\tau]}$ where each $p_j \in [n]$ is uniform. Send $(\mathcal{C}, \mathcal{P})$ to the prover.

Round 3 For each $j \in [M] \setminus \mathcal{C}$, the prover sends seed_j^*, h'_j to the verifier.

For each $j \in \mathcal{C}$, the prover sends $\{\mathsf{state}_{j,i}, r_{j,i}\}_{i\neq p_j}$, com_{j,p_j}, $\{\hat{z}_{j,\alpha}\}$, and msgs_{j,p_j} to the verifier.

Verification The verifier accepts iff all the following checks succeed:

1. For every $j \in \mathcal{C}$, $i \neq p_j$, the verifier uses $\mathsf{state}_{j,i}$ and $r_{j,i}$ to compute $\mathsf{com}_{j,i}$. It then computes $h_j := H(\mathsf{com}_{j,1}, \ldots, \mathsf{com}_{j,n})$.
2. For $j \in [M] \setminus \mathcal{C}$, the verifier uses seed_j^* to compute h_j as an honest prover would. It then computes $h := H(h_1, \ldots, h_M)$.
3. For each $j \in \mathcal{C}$, the verifier simulates an execution of Π among the $\{S_i\}_{i\neq p_j}$ using $\{\mathsf{state}_{j,i}\}_{i\neq p_j}$, masked input-wire values $\{\hat{z}_\alpha\}$, and msgs_{j,p_j}. This yields $\{\mathsf{msgs}_i\}_{i\neq p_j}$ and an output bit b. The verifier checks that $b \stackrel{?}{=} 1$ and computes $h'_j := H(\{\hat{z}_{j,\alpha}\}, \mathsf{msgs}_{j,1}, \ldots, \mathsf{msgs}_{j,n})$ as well as $h' := H(h'_1, \ldots, h'_m)$.
4. The verifier checks that $H(h, h') \stackrel{?}{=} h^*$.

KKW18: Performance of the proposed NIZK PoK

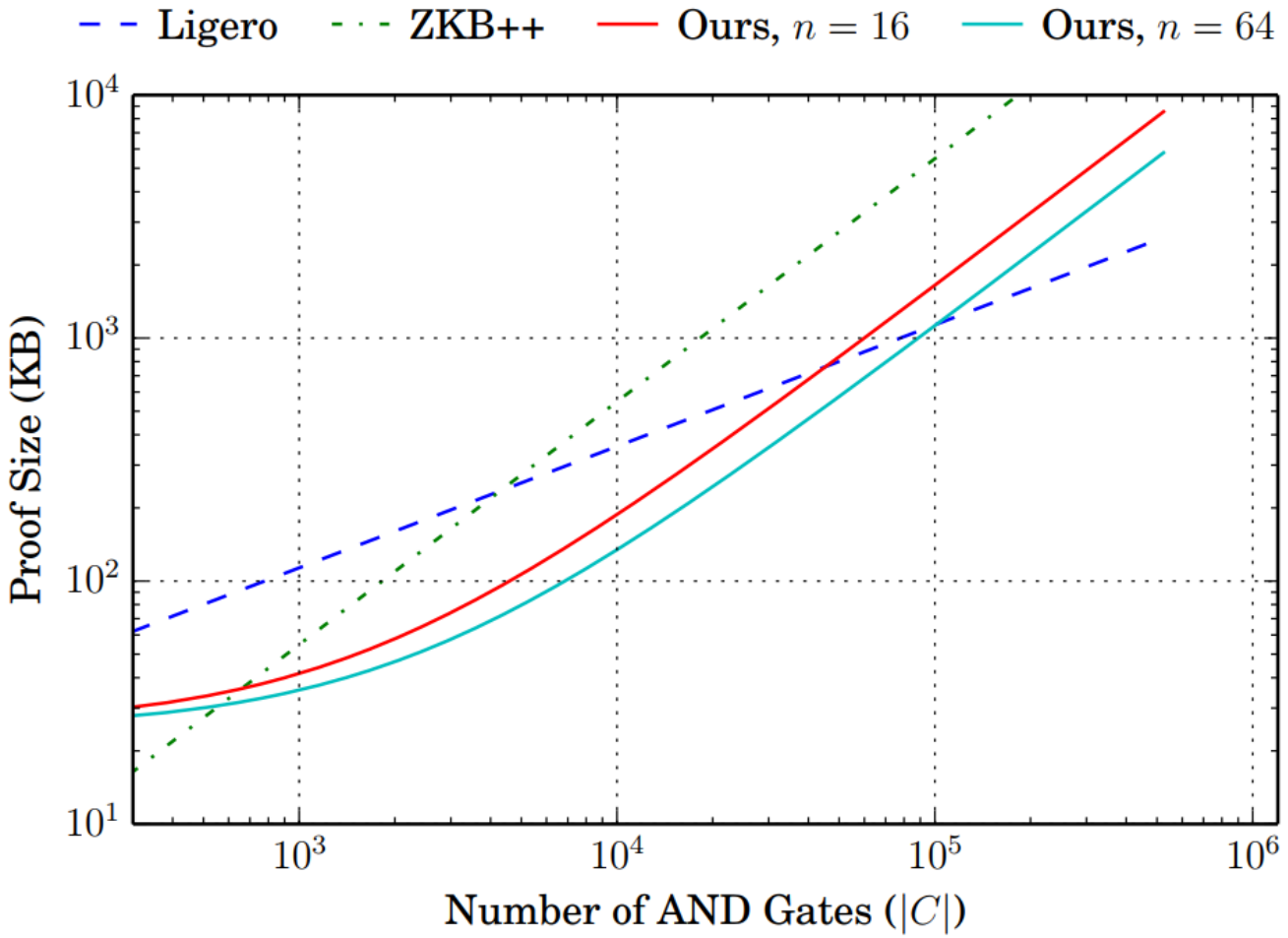
- For circuits with various number of AND gates
  - $|\sigma|$: Proof size
  - t_{total}: Prover's total computation time $\approx$ Verifier's total computation time
  - t_{online}: Prover's *online* computation time

Very good flexibility

Circuit size:	1,000			10,000			100,000		
	$\|\sigma\|$	t_{total}	t_{online}	$\|\sigma\|$	t_{total}	t_{online}	$\|\sigma\|$	t_{total}	t_{online}
$n = 64$	37 KB	128 ms	10 ms	136 KB	851 ms	220 ms	1126 KB	7865 ms	2207 ms
$n = 32$	39 KB	51 ms	7 ms	159 KB	420 ms	149 ms	1351 KB	4015 ms	1498 ms
$n = 16$	44 KB	28 ms	7 ms	190 KB	250 ms	105 ms	1652 KB	2408 ms	1039 ms
$n = 8$	50 KB	16 ms	4 ms	246KB	190 ms	87 ms	2203 KB	1886 ms	883 ms

KKW18: Comparison to prior work

- The proof size still is $O(n)$
  - Same asymptotic communication as ZKBoo, ZKB++
  - Better concrete factors
  - Similar prover computations
  - Worse asymptotic communication than Ligero
    - Still better for “medium-size” circuits
    - Asymptotically better prover computation
  - Smallest proof size for circuits with 300-100.000 AND gates



KKW18: Building Signatures

- The proposed NIZK is based on PRFs and Hash functions
  - Not known quantum attacks
  - Classical security proof based on “quantum-secure assumptions”
- **To Construct a Signature Scheme**
  - Private key x
  - Public key is $y = F_x(0)$ for block cipher F
  - Signature is a NIZK proof of knowledge of x
- Circuit size of F is critical
  - They use LowMC (as in ZKB++)
- The constructed NIZK is used to build post-quantum secure ring/group signatures

KU LEUVEN

Thank You!



karim.baghery@kuleuven.be