How to prove with zero knowledge

Zero-knowledge proof primer **part II**

Michał Zając

Clearmatics Ltd / University of Tartu

Recall

Interactive proof is **complete** if an honest verifier V accepts a proof from an honest prover P

Interactive proof is **sound** if no dishonest prover P^* can make verifier V accept (with non negl. prob.)

Interactive proof is **zero-knowledge** if there exists a **simulator** S that for every $x \in \mathcal{L}$ can produce a transcript of proof between honest prover P and some verifier V^*

Non-interactive zero knowledge

Protocol is **zero-knowledge** if $\forall V^* \; \exists S^V \forall x$:

$$S^V(x) \approx (P, V^*)(x)$$

Assume we have a NIZK protocol Π, can it be sound?

1-round protocol $\Longrightarrow$
Nothing to rewind for S

ACCEPT!

Non-interactive zero knowledge

Protocol is **zero-knowledge** if $\forall V^* \; \exists S^V \forall x$:

$$S^V(x) = (P, V^*)(x)$$

Non-interactive negligible-sound ZK proof is **impossible** in the standard model

Thank you!

Non-interactive zero knowledge

Protocol is **zero-knowledge** if $\forall V^* \; \exists S^V \forall x$:

$$S^V(x) = (P, V^*)(x)$$

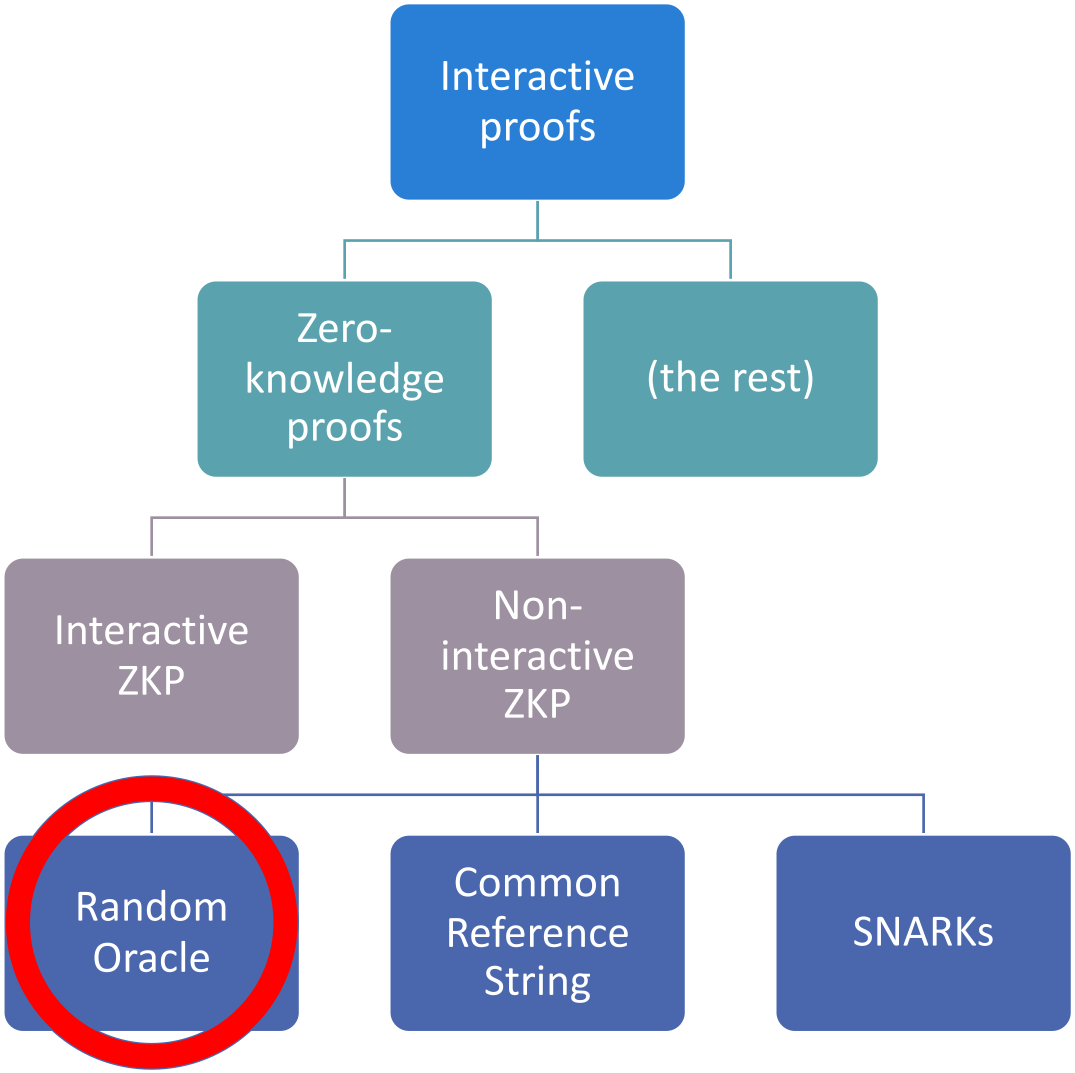
Non-interactive negligible-sound ZK proof is **impossible** in the standard model

Now: Consider **non-standard models** and get around the impossibility result

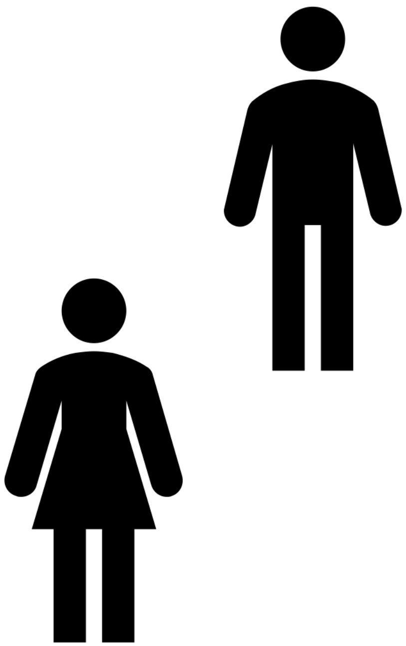
Random Oracle Model

Common Reference String Model

Random Oracle Model



Random Oracle



$a \in \{0,1\}^k$

If a not queried before:

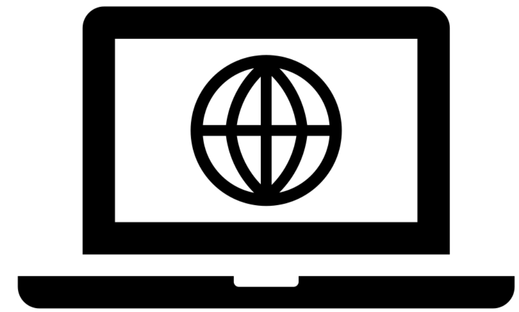
$b \in_r \{0,1\}^n, H(a) = b$

Return b, record (a, b)

If for a **there** is (a, b),

Return b

k, n

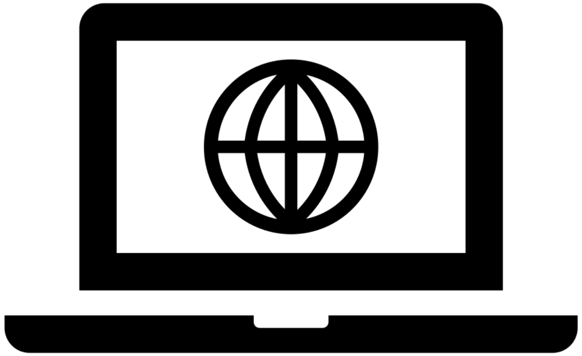


H

H responses are random (impossible to predict)

Infeasible to find x and x' such that $H(x) = H(x')$

Random Oracle - instantiation



$$H_{n,k}\colon \{0,1\}^k \to \{0,1\}^n$$

From a set $\mathcal{F}_{n,k}$ of all functions from $\{0,1\}^k$ to $\{0,1\}^n$ pick a **random function** f

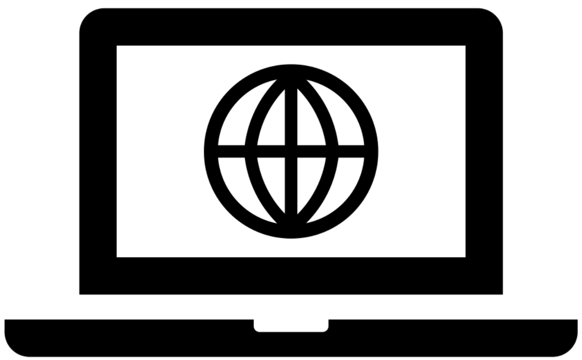
$$H = f$$

$\mathcal{F}$ contains $(2^n)^{2^k}$ elements

Problems:

- How to pick efficiently from such a vast set?
- With a great probability f is infeasible to describe (we cannot describe it in polynomial time)

Random Oracle - instantiation



$$H_{k,n}\colon \{0,1\}^k \to \{0,1\}^n$$

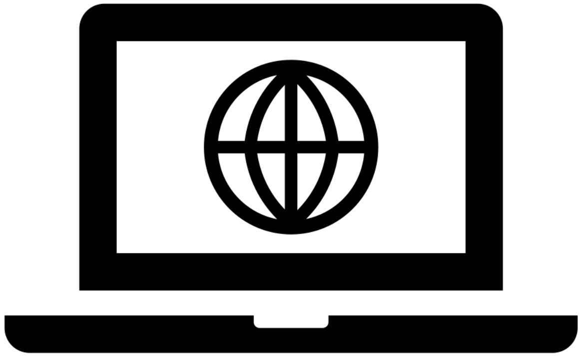
From a set $\mathcal{F}_{k,n}$ of all functions from $\{0,1\}^k$ to $\{0,1\}^n$ pick a **random function** f

$$H = f$$

No collection of deterministic function can instantiate a **random oracle** ☹

There are protocols secure in ROM, that become insecure if the oracle is instantiated by any function

Random Oracle - instantiation



$$H_{k,n}: \{0,1\}^k \to \{0,1\}^n$$

From a set $\mathcal{F}_{k,n}$ of all functions from $\{0,1\}^k$ to $\{0,1\}^n$ pick a **random function** f

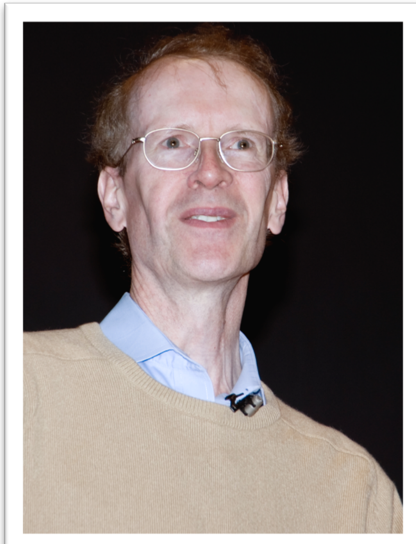
$$H = f$$

But security proof with a random oracle is better than **no proof** at all

In practice random oracle is substituted by a **hash function** like sha256, sha-3, previously md-5

Fiat-Shamir transformation

x, w, H



$\beta = H(a)$

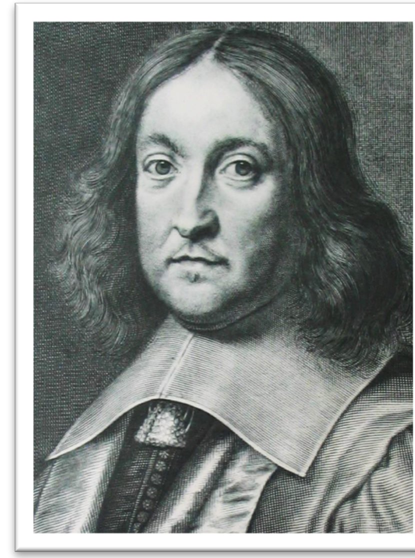
$$\xrightarrow{\alpha}$$

$$\xleftarrow{\beta}$$

$$\xrightarrow{\gamma}$$

$$\xrightarrow{\alpha, \gamma(\alpha, H(\alpha))}$$

x, H



$\beta = H(a)$

Instead of waiting for Pierre's input,
Andrew computes challenges himself

Requirements: public-coin, constant-round argument

Public-coin vs private-coin

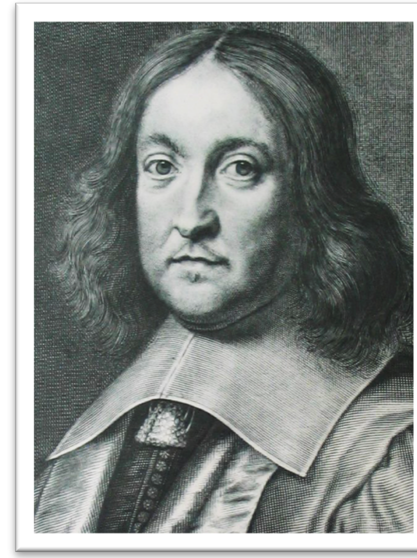
x, w

α →

β ←

γ →

x



Public coin - V picks randomly his challenges and sends it

Private coin - V may not reveal his randomness

Public-coin vs private-coin

Public coin:

- Graph Hamiltonicity
- Graph isomorphism
- Schnorr identification scheme

…

Private coin:

- Graph Non-isomorphism
- Any proof where V sends a commitment

…

Constant number of rounds

There exists multi-round protocol that is secure when executed interactively, but not secure with FS applied

Take any constant-sound protocol and execute it sequentially

(P, V)

$\xrightarrow{\alpha_1}$

$\xleftarrow{\beta_1}$

$\xrightarrow{\gamma_1}$

$\ldots$

$\xrightarrow{\alpha_k}$

$\xleftarrow{\beta_k}$

$\xrightarrow{\gamma_k}$

(P_{FS}, V_{FS})

$\xrightarrow{\alpha_1, \beta_1, \gamma(\alpha_1, H(\alpha_1))}$

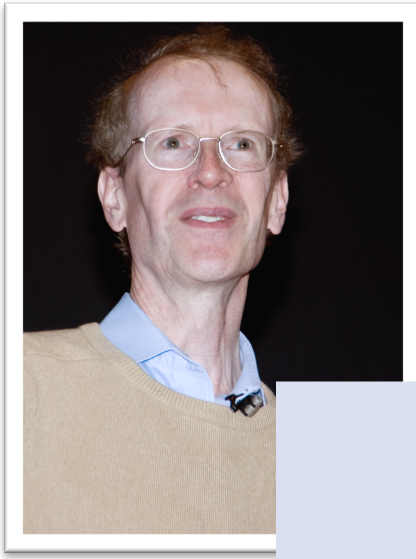
Note: Probability of finding good β is constant

$\ldots$

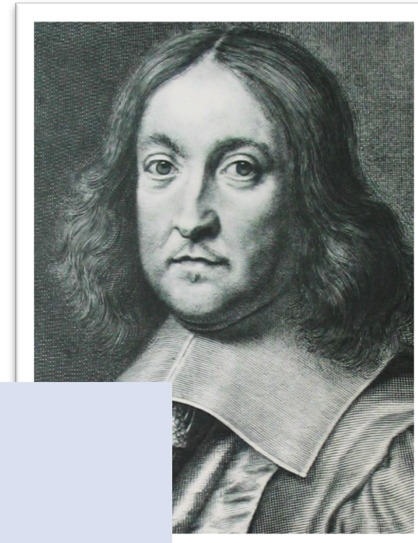
$\xrightarrow{\alpha_k, \beta_k, \gamma(\alpha_k, H(\alpha_k))}$

Fiat-Shamir transformation - soundness

x, w



x



$$\xrightarrow{\alpha, \gamma(\alpha, H(\alpha))}$$

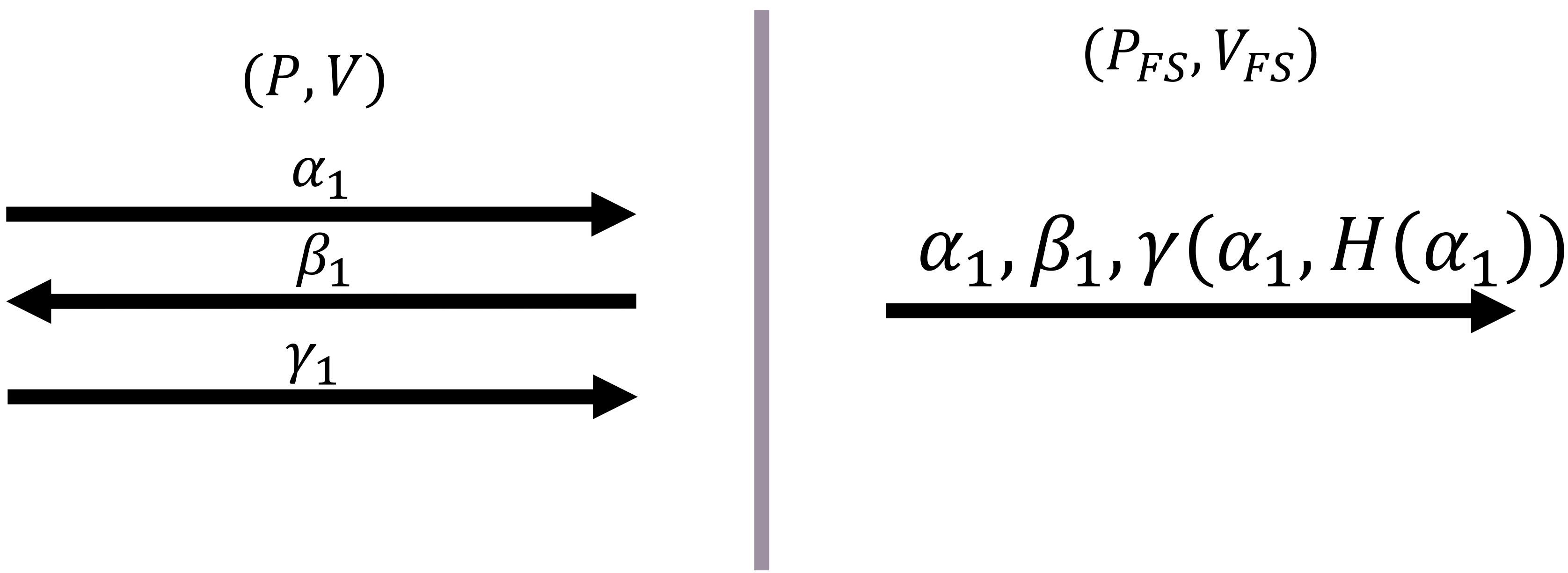
$\beta = H(\alpha),$

Soundness (intuition):

1. Only very few $H(\alpha_i)$ suits Andrew
2. It is hard to Andrew to pick α_i such that he knows $\alpha_{i+1}(\alpha_1, H(\alpha_1) \ldots, \alpha_i, H(\alpha_i))$ unless $x \in \mathcal{L}$

Fiat-Shamir security

Let (P, V) be an interactive, constant-round, public-coin, sound proving system. Then (P_{FS}, V_{FS}) is sound as well



Fiat-Shamir security proof

Useful fact:

Suppose X, Y are jointly distributed RVs s.t.

$$\Pr[\, A(X,Y) = 1] \geq \epsilon$$

Then, for at least $\frac{\epsilon}{2}$ fraction of x-s it holds that

$$(*)\Pr_{Y|x}[A(x,Y) = 1] \geq \frac{\epsilon}{2}\ .$$

Proof

Markov's inequality.

Suppose not. Call x good if (*) holds, then

$$\Pr[A(X,Y)] = \Pr[X\ good]\Pr[A(X,Y)|\ X\ good] +$$
$$+ \Pr[X\ bad]\Pr[A(X,Y)|X\ bad] < \frac{\epsilon}{2} \cdot 1 + 1 \cdot \frac{\epsilon}{2} = \epsilon$$

Fiat-Shamir security proof

Suppose $\exists x \notin \mathcal{L}$ and P_{FS}^* that runs in time T and makes V_{FS} accept x with probability $\geq \epsilon$

Construct P^* such that V accepts x with probability $\mathrm{poly}(\epsilon, \frac{1}{T})$
Denote oracle queries by $Q_1, \ldots, Q_T$, wlog all Q_i distinct and $\alpha \in \{Q_1, \ldots, Q_T\}$
CLAIM: $\exists i^* \in [T]$ s.t. P_{FS}^* wins with prob. $\frac{\epsilon}{T}$ conditioned on $Q_{i^*} = \alpha$
PROOF: by contradiction

Fiat-Shamir security proof

Forking lemma

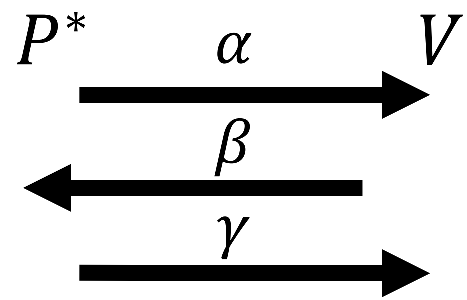
For $\frac{\epsilon}{2T}$ fraction of $(q_1, \dots, q_{i^*})$ it holds that P_{FS}^* wins w.p. $\frac{\epsilon}{2T}$ conditioned on $Q_{i^*} = \alpha$ and $Q_i = q_i$ for all $i \leq i^*$

PROOF: by the useful fact

Fiat-Shamir security proof

Breaking soundness

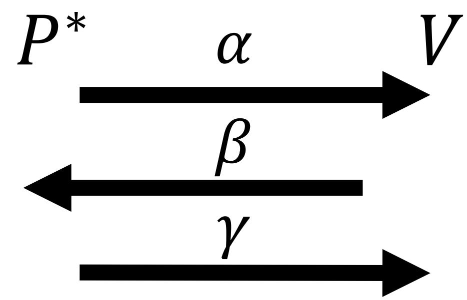
- Start running P_{FS}^* up to its i^* query using random answers
- Let $\alpha = Q_{i^*}$ be the i^*-th query. Send α and get β
- Continue running P_{FS}^* while answering Q_i^* with β and other queries uniformly at random
- Eventually P_{FS}^* outputs $(\alpha', \beta', \gamma')$
- If $\alpha = \alpha', \beta = \beta'$ send $\gamma = \gamma'$



Fiat-Shamir security proof

Use **Forking lemma**

- For $\frac{\epsilon}{2T}$ fraction of $(q_1, \ldots, q_{i^*})$ it holds that P_{FS}^* wins w.p. $\frac{\epsilon}{2T}$ conditioned on $Q_{i^*} = \alpha$ and $Q_i = q_i$ for $i \leq i^*$.
- Wp $\frac{\epsilon}{2T}$ over choice of $(Q_1, \ldots, Q_{i^*})$ it holds that wp $\frac{\epsilon}{2T}$ over all remaining coin tosses P_{FS}^* wins and $\alpha' = \alpha$
- Hence, P^* wins with probability $\left(\frac{\epsilon}{2T}\right)^2$, which is non-negligible



Fiat-Shamir transformation - zk

$$\xrightarrow{\alpha, H(\alpha), \gamma(\alpha, H(\alpha))}$$

Zero knowledge

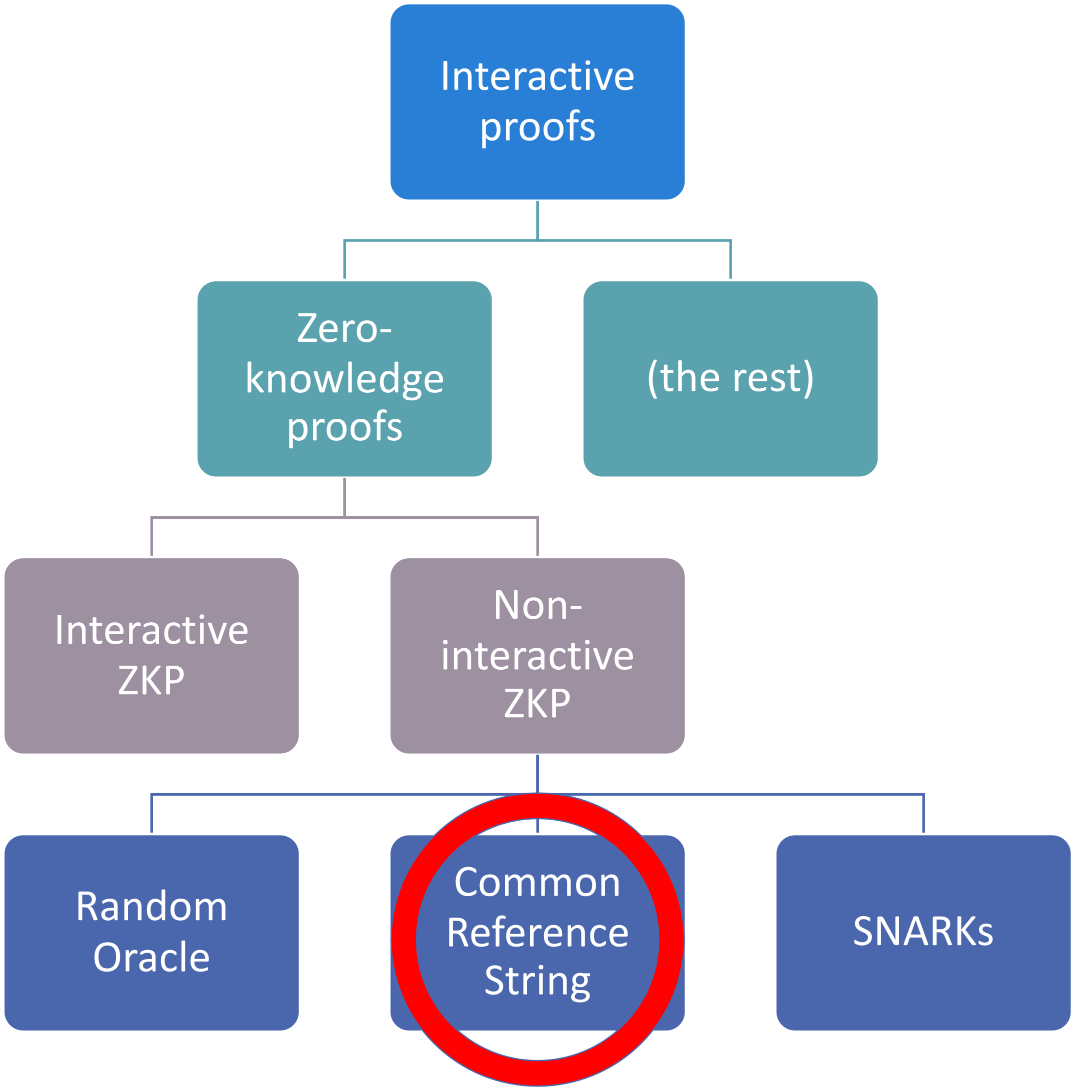
S equipped with a **superpower**: S <u>can change</u> H

Simulation (intuition)
Pick α,
Compute β such that S knows how to answer $\gamma(\alpha, \beta)$
Set $H(\alpha) = \beta$
Send α, β, γ

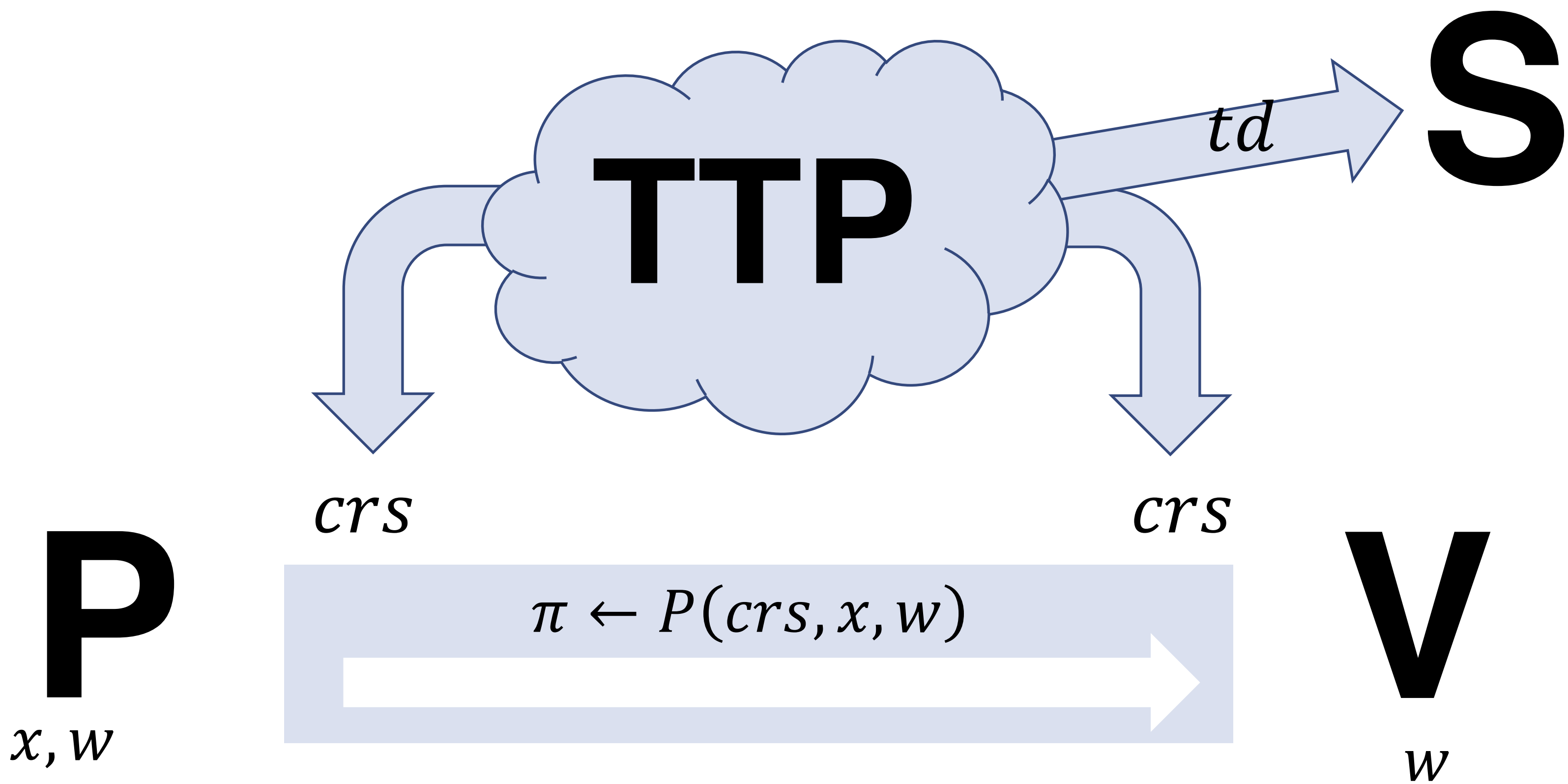
$$\Pr_{R,H}\left[S^{V^*}(x) = a\right] = \Pr_{R,H}\left[(P, V^*)\,(x) = a\right]$$

(picking random oracle is part of the randomness)

Common Reference String Model



Common Reference String Model



Zero-knowledge

$P(crs, x, w) \approx S(crs, td, x)$

ACCEPT

$if\ V(crs, x) = 1$

Non-interactive proofs – problems

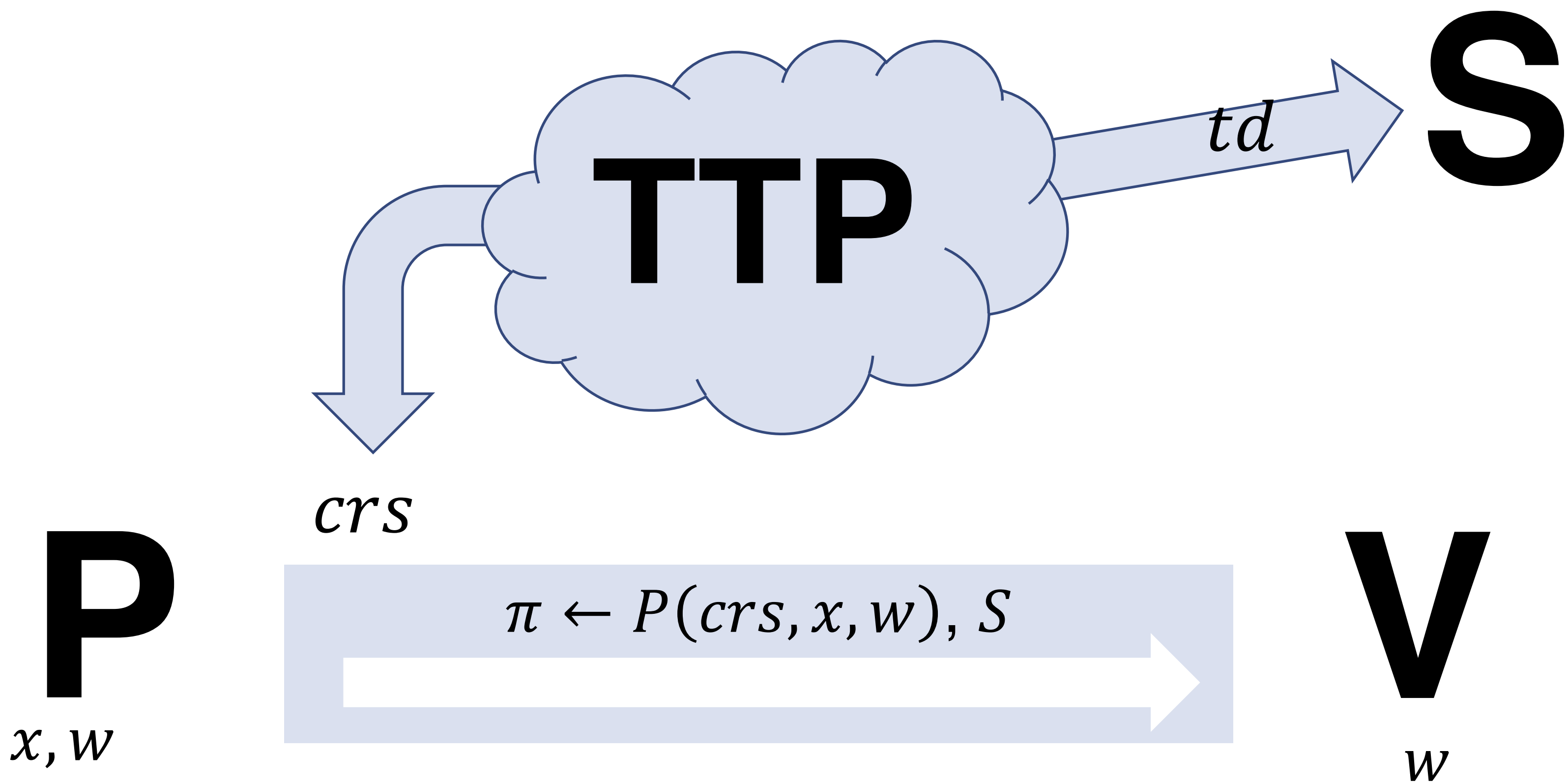
Till 2006 only theoretical result

Jens Groth, Rafail Ostrovsky, Amit Sahai
Perfect Non-interactive Zero Knowledge for NP

Jens Groth, Amit Sahai
Efficient Non-interactive Proof Systems for Bilinear Groups

For long statements proofs are still very long
But we want to proof long statements!

Hidden Bits Model



Picks S – a part of the CRS that will be disclosed to V

Feige-Lapidot-Shamir Paradigm

How to get **NIZK**?

Fiat-Lapidor-Shamir Paradigm

Get a NIZK in a **Hidden Bits Model**

Compile to the CRS model

Feige-Lapidot-Shamir Paradigm

How to get **NIZK**?

Fiat-Lapidor-Shamir Paradigm

Get a NIZK in a **Hidden Bits Model**

Compile to the CRS model

NIZK in Hidden Bits Model

NIZK in HBM for Hamiltonicity

Recall: Hamiltonicity is $\mathbb{NP}$-complete

Common input $G = (V, E)$
Witness: u – Hamiltonian path
CRS: random cycle graph
$C = (V_C, E_C)$ on $|V|$

C

P

f, S $\longrightarrow$

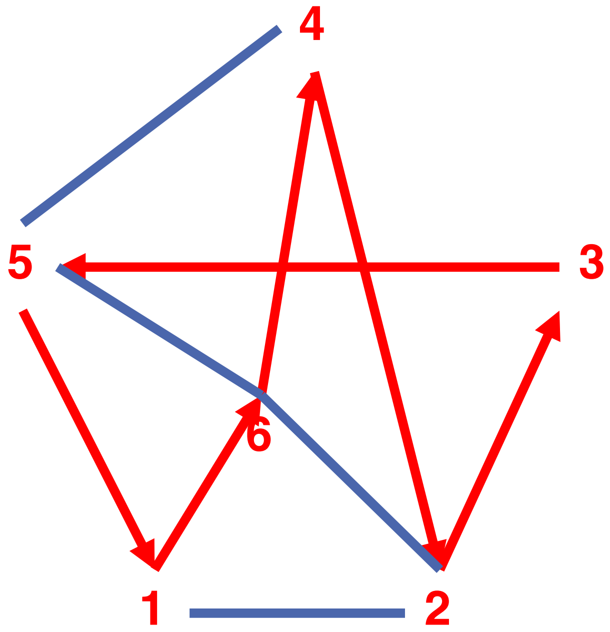
V

Find injective map
$f: V \to V_C$
that preserves cycle

Reveal $S \subset V_C \times V_c$ st
$S = f(V^2 \setminus E)$

Check that f injective
$\forall e \notin E,\ f(e)$ was revealed

HAM in HBM



$f(1) = 6$

$f(2) = 3$

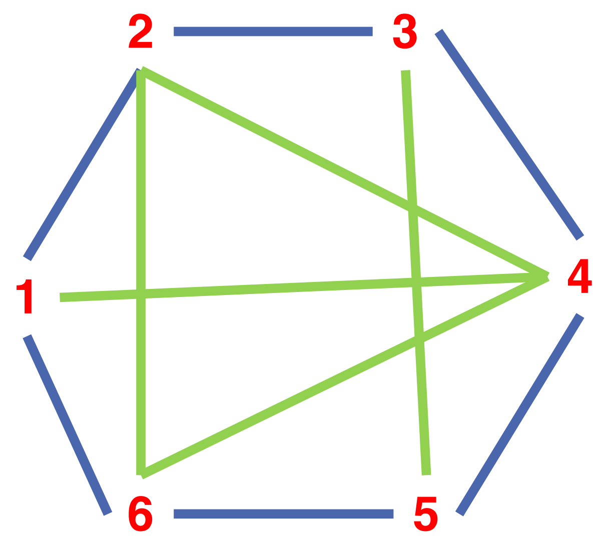
$f(3) = 4$

$f(4) = 2$

$f(5) = 5$

$f(6) = 1$

Reveal $S \subset V_C \times V_C$ st
$S = f(V^2 \setminus E)$



Perfect soundness

Assume V accepts: f is injective, all non edges E revealed

Consider E' the inverse of the cycle edges of C (e.g $\left(f^{-1}(6), f^{-1}(1)\right)$)

1. $E' \subset E$ (i.e. contains only actual edges)
2. E' forms a Hamiltonian cycle (as an inverse of a Ham. cycle)

Thus G is Hamiltonian

HAM in HBM

Zero knowledge

Simulator picks random injective f to $[n]$

Output (f, S, CRS_S), where $S = f(V^2 \setminus E)$ and $CRS_S = 0..00$

Claim 1: for every fixed choice of f the simulated view is identical to the real

Claim 2: mapping in the real execution is a random injective function

Feige-Lapidot-Shamir Paradigm

How to get **NIZK**?

Fiat-Lapidor-Shamir Paradigm

Get a NIZK in a **Hidden Bits Model**

Compile to the CRS model

From HBM to CRS

Idealised trapdoor permutation

$p_k: \{0,1\}^n \to \{0,1\}^n$

- Given p_k it is easy to compute $p_k(x)$ for every x
- Given $y = p_k(x)$ infeasible to compute x
- There exists trapdoor τ that given y, τ easy to compute $x = p_k^{-1}(y)$

Hardcore bit

There exists function $h: \{0,1\}^n \to \{0,1\}$ st
given $p_k(x)$ infeasible to compute bit $h(x)$

Perfect trapdoor permutations **do not exist**!
But we have public-key encryption schemes (RSA, ElGamal, etc.)
(k – public key, τ – secret key)

From HBM to CRS – bird's eye

CRS consists of $y_1, \ldots, y_\ell \in \{0,1\}^n$

P chooses k, τ
Hidden bits: $h(x_i)$
To reveal a bit P sends x_i

$y_1 \ldots y_\ell$

P $\xrightarrow{\alpha, \pi, S}$ **V**

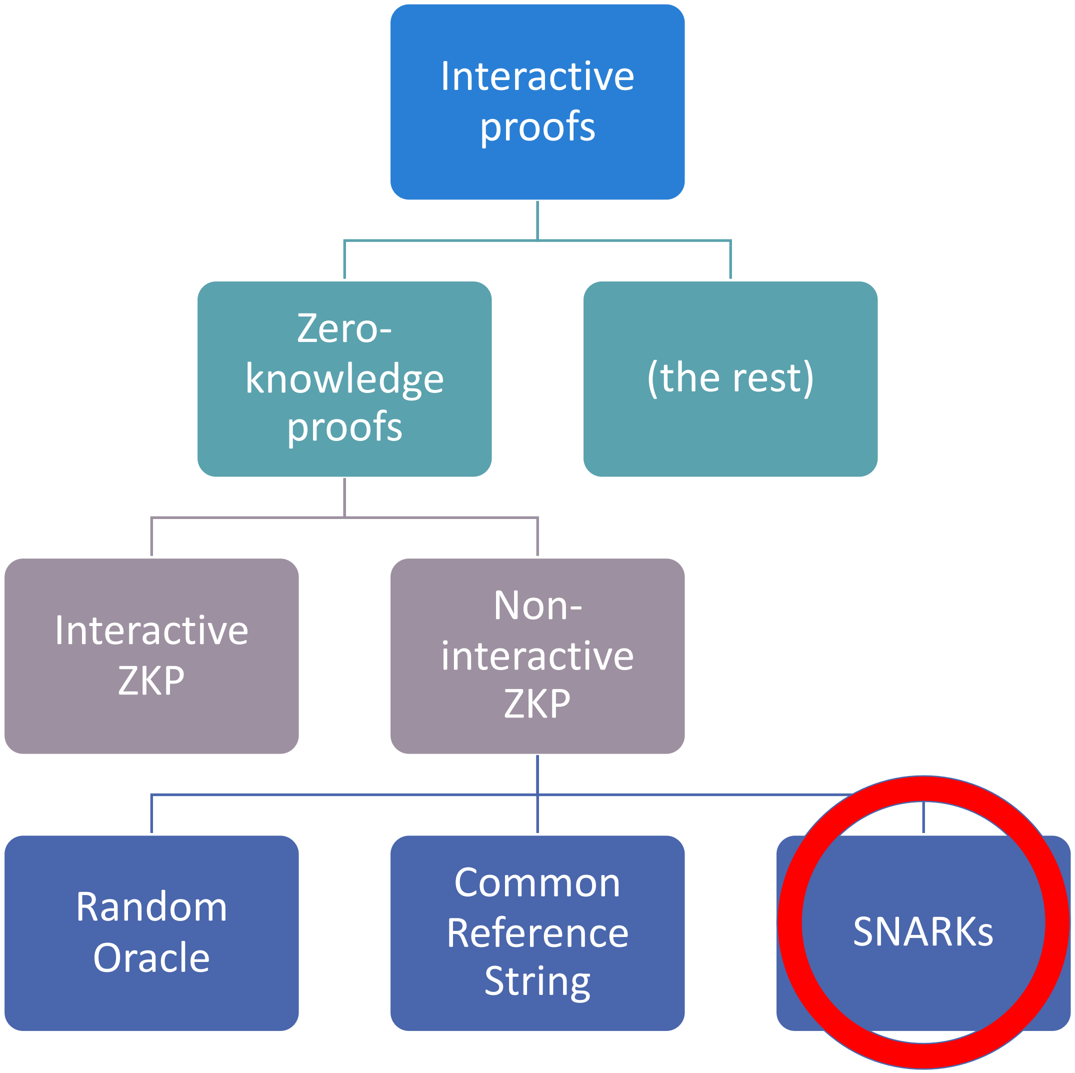
P:

x, w
Choose k, τ
Define $x_i = p_k^{-1}(y_i)$
Let $b_i = h(x_i)$
Run HB protocol $(x, w, (b_1 \ldots b_\ell))$
Get proof π and set S

V:

x
For $i \in S$ check $p_\alpha(x_i) = y_i$
Define $b_i = h(x_i)$
Check that HB verifier accepts $(x, \pi, \{b_i\}_{i \in S})$

SNARKs


Interactive proofs
Zero-knowledge proofs
(the rest)
Interactive ZKP
Non-interactive ZKP
Random Oracle
Common Reference String
SNARKs

SNARKs

Proofs vs Arguments

We call protocol (P, V) **proof** if it holds for all possible P^*

(no one can cheat V)

We call protocol (P, V) **argument** if no PPT P^* can cheat V

Compare:
$\forall x \; \forall V^* \; \exists S$
$S^{V^*}(x)$
$= (P, V^*)(x)$

NIZK $\Pi = (K, P, V, S)$

- additional element responsible for CRS generation
- S given explicitly– one symulator **for all** verifiers

$\pi \leftarrow P(crs, x, w)$

If $|\pi|$ sublinear compared to $|x|$ and $|w|$ we call Π **succinct**

SNARK – Succinct Non-interactive ARgument of Knowledge

Verifiable computation

The client may send P once and evaluate it on many inputs

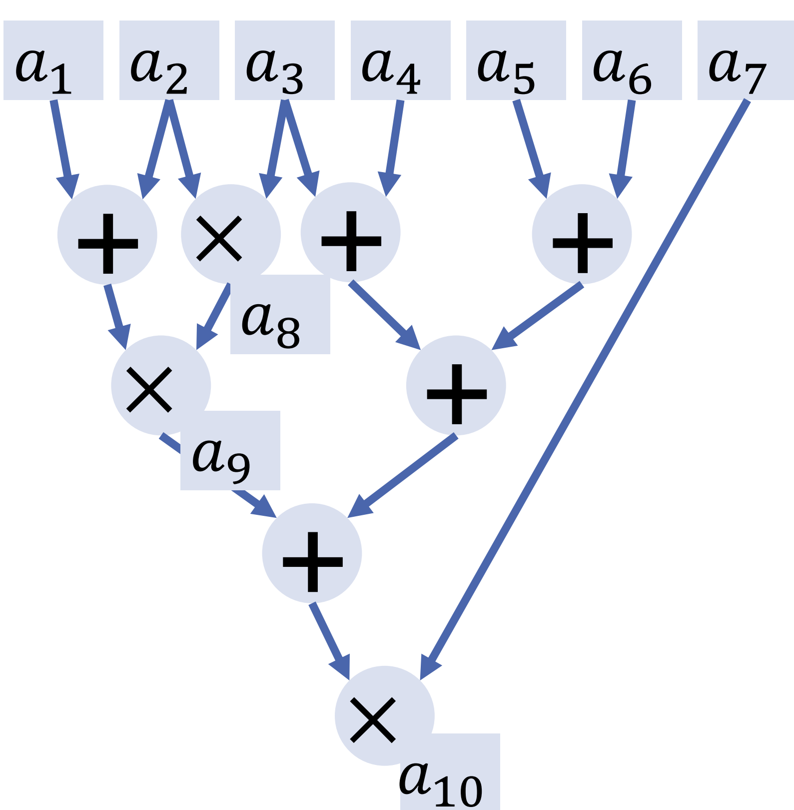
sends a program P along with its (public) input x

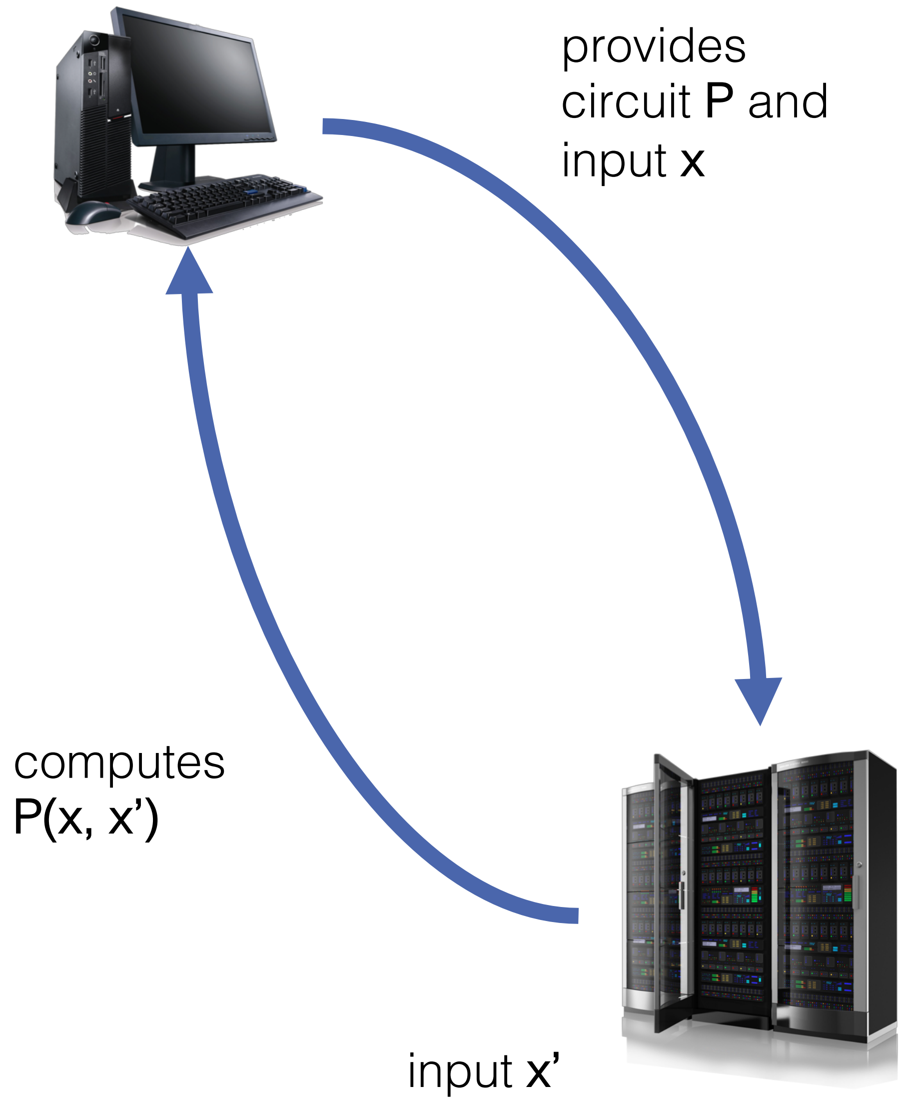
responds with the output $y = P(x, x')$ along with **a proof** π that the result is correct

- program P that needs a lot of computational power
- input x

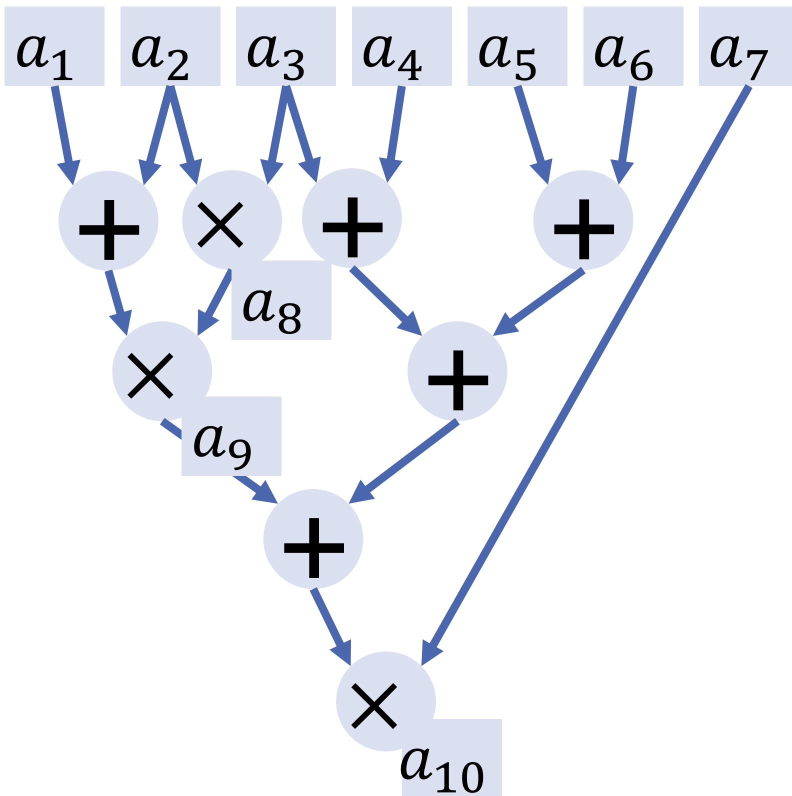
- has required power
- additional input x'

Program as a circuit





How to prove correctness



Server could just reveal values on all wires:
a_1 , a_2, … and let the **client** compute P(x, x') on his own

Client does not need server

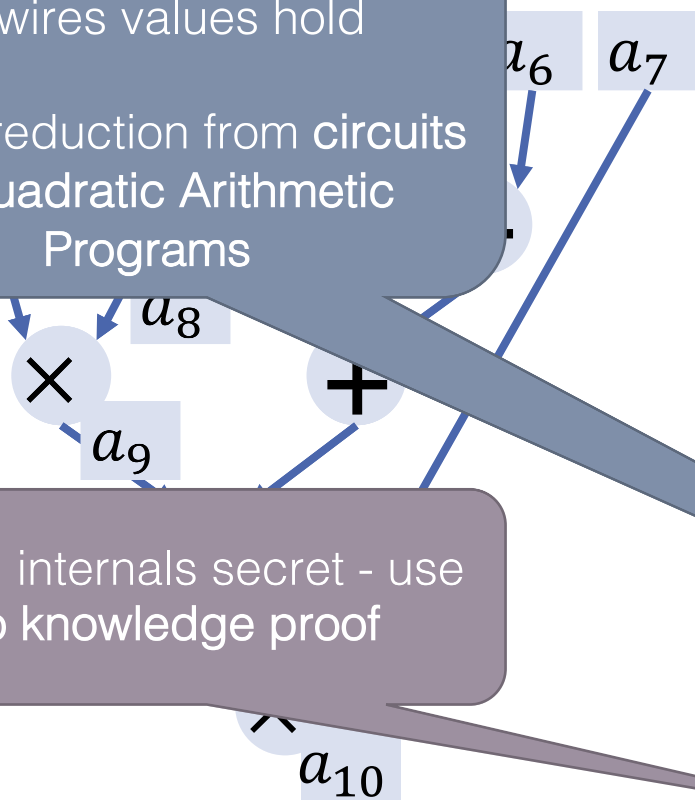
What if **x'** is secret?

Prove the correctness
without revealing intermediate values

How to prove correctness

correctness = some relation on wires values hold

natural reduction from **circuits** to **Quadratic Arithmetic Programs**



keeping internals secret - use **zero knowledge proof**

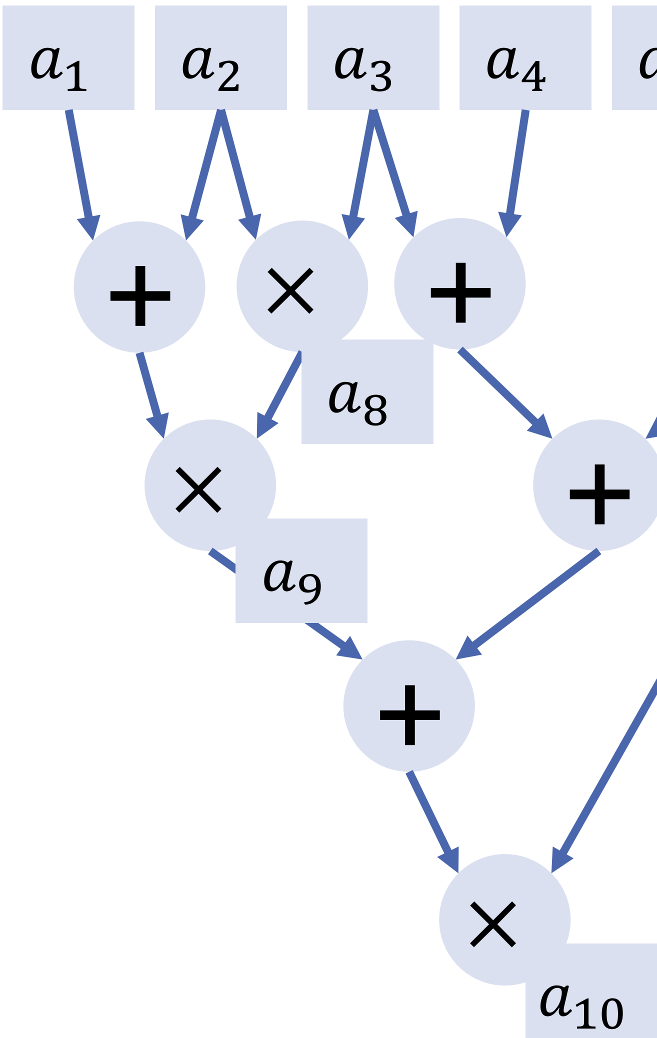
Server could just reveal values on all wires: a_1 , a_2, ... and let the **client** compute P(x, x') on his own

Client does not need server

What if **x'** is secret?

Prove the correctness **without** revealing intermediate values

Verifiable computation

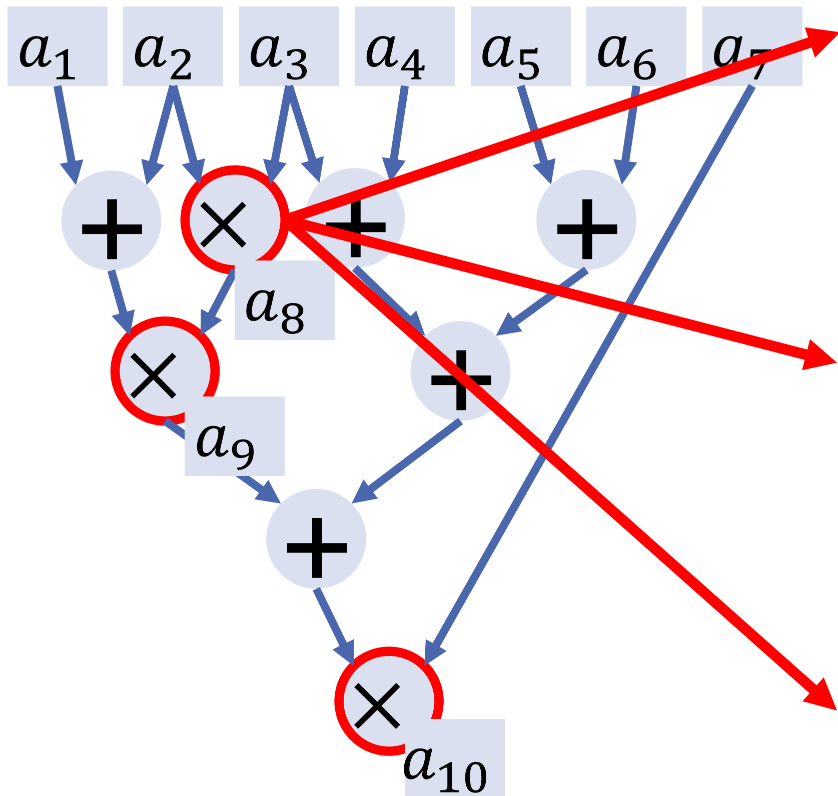


Statement

for given circuit C and public $a_1, a_2 \dots a_5$, I know $a_6, \dots a_{10}$ such that C is computed correctly

How to represent circuit C to
- show validity of computation
- verify the computation

Circuit representation



$\mathcal{U}$

1	2	3	4	5	6	7	8	9	10
	1								
1	1								
		1	1	1	1			1	

$\mathcal{V}$

1	2	3	4	5	6	7	8	9	10
		1							
							1		
						1			

$\mathcal{W}$

1	2	3	4	5	6	7	8	9	10
							1		
								2	
									5

Statement

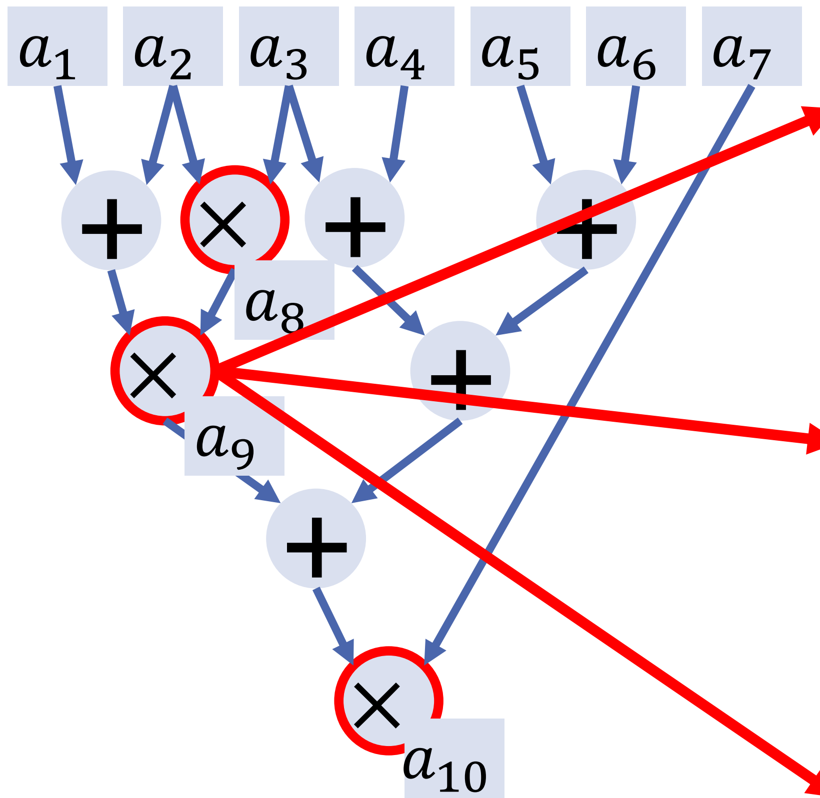
for given circuit C and public $a_1, a_2 \dots a_5$, I know $a_6, \dots a_{10}$ such that C is computed correctly

Matrix dimension

$$n = |rows| = |MUL\ gates|$$

$$m = |columns| = |inputs| + |MUL\ gates|$$

Circuit representation



$\mathcal{U}$

1	2	3	4	5	6	7	8	9	10
	1								
1	1								
		1	1	1	1			1	

$\mathcal{V}$

1	2	3	4	5	6	7	8	9	10
		1							
							1		
						1			

$\mathcal{W}$

1	2	3	4	5	6	7	8	9	10
							1		
								2	
									5

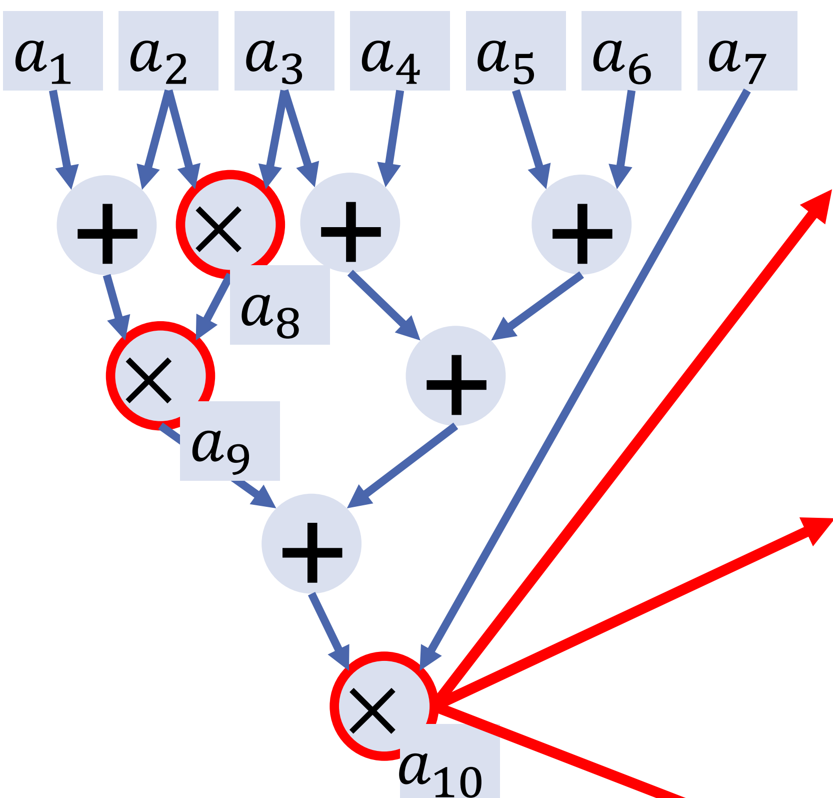
Statement

for given circuit C and public $a_1, a_2 \ldots a_5$, I know $a_6, \ldots a_{10}$ such that C is computed correctly

Matrix dimension

$$n = |rows| = |MUL\ gates|$$
$$m = |columns| = |inputs| + |MUL\ gates|$$

Circuit representation



1	2	3	4	5	6	7	8	9	10
	1								
1	1								
		1	1	1	1			1	

$\mathcal{U}$

1	2	3	4	5	6	7	8	9	10
		1							
							1		
						1			

$\mathcal{V}$

1	2	3	4	5	6	7	8	9	10
							1		
								2	
									5

$\mathcal{W}$

Statement

for given circuit C and public $a_1, a_2 \dots a_5$, I know $a_6, \dots a_{10}$ such that C is computed correctly

Matrix dimension

$$n = |rows| = |MUL\ gates|$$
$$m = |columns| = |inputs| + |MUL\ gates|$$

Matrix circuit representation

$\mathcal{U}$

1	2	3	4	5	6	7	8	9	10
	1								
1	1								
		1	1	1	1			1	

$\mathcal{V}$

1	2	3	4	5	6	7	8	9	10
		1							
							1		
						1			

$\mathcal{W}$

1	2	3	4	5	6	7	8	9	10
							1		
								2	
									5

Let a_i be a circuit input or multiplication gate output, $u_{i,q}, v_{i,q}, w_{i,q}$ be i-th element of q-th equation, then

$$\sum_{i=0\ldots m} a_i u_{i,q} \cdot \sum_{i=0..m} a_i v_{i,q} = \sum_{i=0..m} a_i w_{i,q}$$

Problem:
Operations on matrices are usually very inefficient ☹

Idea:
We know how to make polynomial operations efficiently!
If we only could represent the matrices as polynomials…

Polynomial representation

Let a_i be a circuit input or multiplication gate output, $u_{i,q}, v_{i,q}, w_{i,q}$ be i-th element of q-th equation, then

$$\sum_{i=0\ldots m} a_i u_{i,q} \cdot \sum_{i=0..m} a_i v_{i,q} = \sum_{i=0..m} a_i w_{i,q}$$

Define polynomials u_i, v_i, w_i such that

- $u_i(r_q) = u_{i,q}$,
- $v_i(r_q) = v_{i,q}$,
- $w_i(r_q) = w_{i,q}$,

Then the constraint above can be expressed as

$$\sum_{i=0\ldots m} a_i u_i(r_q) \cdot \sum_{i=0..m} a_i v_i(r_q) = \sum_{i=0..m} a_i w_i(r_q)$$

Polynomial representation

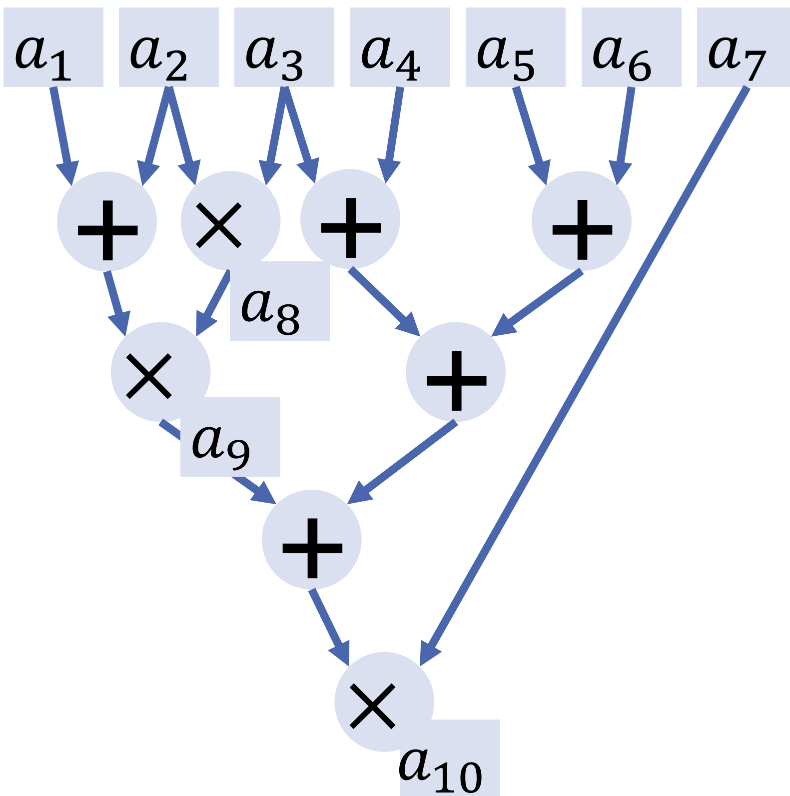
$$\sum_{i=0\ldots m} a_i u_i(r_q) \cdot \sum_{i=0..m} a_i v_i(r_q) = \sum_{i=0..m} a_i w_i(r_q)$$

Define $t(X) = \prod_{i=1..n}(X - r_q)$

$$\sum_{i=0\ldots m} a_i u_i(X) \cdot \sum_{i=0..m} a_i v_i(X) = \sum_{i=0..m} a_i w_i(X)\, mod\; t(X)$$

$$R = \left\{ (\phi, w) \,\middle|\, \begin{array}{l} \phi = (a_1, \ldots, a_\ell) \in \mathbb{F}^\ell \\ w = (a_{\ell+1}, \ldots, a_m) \in \mathbb{F}^{m-\ell} \\ \\ \quad \sum_{i=0}^m a_i u_i(X) \cdot \sum_{i=0}^m a_i v_i(X) \equiv \sum_{i=0}^m a_i w_i(X) \mod t(X) \end{array} \right\}.$$

Getting things together



1	2	3	4	5	6	7	8	9	10
	1								
1	1								
		1	1	1	1			1	

1	2	3	4	5	6	7	8	9	10
		1							
							1		
						1			

1	2	3	4	5	6	7	8	9	10
							1		
								2	
									5

$$R = \left\{ (\phi, w) \middle| \begin{array}{l} \phi = (a_1, \ldots, a_\ell) \in \mathbb{F}^\ell \\ w = (a_{\ell+1}, \ldots, a_m) \in \mathbb{F}^{m-\ell} \\ \\ \quad \sum_{i=0}^m a_i u_i(X) \cdot \sum_{i=0}^m a_i v_i(X) \equiv \sum_{i=0}^m a_i w_i(X) \mod t(X) \end{array} \right\}.$$

One more step

1	2	3	4	5	6	7	8	9	10
	1								
1	1								
		1	1	1	1			1	

Lagrange interpolation

- n points defines $(n-1)$-degree polynomial $p(X)$
- $(x_1, y_1) \ldots (x_n, y_n)$
- Lagrange basis:

$$\ell_i(X) = \prod_{0 \le k \le n, k \ne i} \frac{X - x_k}{x_i - x_k}$$

- $p(X) = \sum y_i \, \ell_i(X)$

$$u_1(X) = 1 \cdot \ell_1(X) + 1 \cdot \ell_2(X) + 0 \cdot \ell_3(X)$$

One more trick:
Instead of using polynomials, evaluate them at random point and use Schwartz-Zippel

SNARK for QAP

$$\mathsf{crs_P} \leftarrow \left(\begin{array}{c} \left[\alpha, \beta, \delta, \left(\frac{u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)}{\delta}\right)_{j=\ell+1}^{m}\right]_1, \\ \left[(\chi^i \ell(\chi)/\delta)_{i=0}^{n-2}, (u_j(\chi), v_j(\chi))_{j=0}^{m}\right]_1, \left[\beta, \delta, (v_j(\chi))_{j=0}^{m}\right]_2 \end{array} \right)$$

$\mathsf{P}(\mathbf{R}, \mathsf{z_R}, \mathsf{crs_P}, \mathsf{x} = (A_1, \dots, A_\ell), \mathsf{w} = (A_{\ell+1}, \dots, A_m))$**:**

1. Let $a^\dagger(X_\chi) \leftarrow \sum_{j=0}^{m} A_j u_j(X_\chi)$, $b^\dagger(X_\chi) \leftarrow \sum_{j=0}^{m} A_j v_j(X_\chi)$, $c^\dagger(X_\chi) \leftarrow \sum_{j=0}^{m} A_j w_j(X_\chi)$,
2. Set $h(X_\chi) = \sum_{i=0}^{n-2} h_i X_\chi^i \leftarrow (a^\dagger(X_\chi) b^\dagger(X_\chi) - c^\dagger(X_\chi))/\ell(X_\chi)$,
3. Set $[h(\chi)\ell(\chi)/\delta]_1 \leftarrow \sum_{i=0}^{n-2} h_i \left[\chi^i \ell(\chi)/\delta\right]_1$,
4. Set $r_a \leftarrow_r \mathbb{Z}_p$; Set $\mathfrak{a} \leftarrow \sum_{j=0}^{m} A_j [u_j(\chi)]_1 + [\alpha]_1 + r_a [\delta]_1$,
5. Set $r_b \leftarrow_r \mathbb{Z}_p$; Set $\mathfrak{b} \leftarrow \sum_{j=0}^{m} A_j [v_j(\chi)]_2 + [\beta]_2 + r_b [\delta]_2$,
6. Set $\mathfrak{c} \leftarrow r_b \mathfrak{a} + r_a \left(\sum_{j=0}^{m} A_j [v_j(\chi)]_1 + [\beta]_1\right) + \sum_{j=\ell+1}^{m} A_j [(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\delta]_1 + [h(\chi)\ell(\chi)/\delta]_1$,
7. Return $\pi \leftarrow (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$.

$$[\alpha]_i = \alpha \mathfrak{g}_i$$

$\mathsf{V}(\mathbf{R}, \mathsf{z_R}, \mathsf{crs_V}, \mathsf{x} = (A_1, \dots, A_\ell), \pi = (\mathfrak{a}, \mathfrak{b}, \mathfrak{c}))$**:** assuming $A_0 = 1$, check that

$$\mathfrak{a} \bullet \mathfrak{b} = \mathfrak{c} \bullet [\delta]_2 + \left(\sum_{j=0}^{\ell} A_j \left[\frac{u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)}{\gamma}\right]_1\right) \bullet [\gamma]_2 + [\alpha\beta]_T \quad .$$

Common Reference String – important questions

How to instantiate the CRS generator?
What if the generator **colludes** with the prover?
What if the generator **colludes** with the verifier?
What security guarantees can we get then?
Is it better to use Random Oracle?

Philosophical question

Is non-interactive
zero-knowledge
zero-knowledge?

When P gives proof π to V, V learns not only that $x \in \mathcal{L}$, but also a convincing proof for that

Key idea: Proof π holds in respect to

- Particular CRS
- Particular Random Oracle

Thank you!

Zero-knowledge in modern life

Part III

Blockchain

- Alice pays Bob 2€
- Charlie pays Alice 3€
- Alice pays Debbie 1€
- Debbie pays Bob 3€

- Frank pays Bob 2€
- Debbie pays Alice 3€
- Robert pays Alice 4€
- Charlie pays Debbie 1,5€

- Alice pays Bob 3€
- Bob pays Charlie 3€
- Alice pays Bob 2€

time

Blockchain

Alice pays Bob 2€

time t

Only Alice can post such transaction

Charlie pays Alice 4€

time $t' < t$

Everybody checks that Alice has sufficient funds to perform the transaction

Everybody knows that Charlie paid Alice, and Alice paid Bob
Everybody knows the value of transactions

Privacy preserving blockchains

Idea: Show in ZK that the transaction is correct

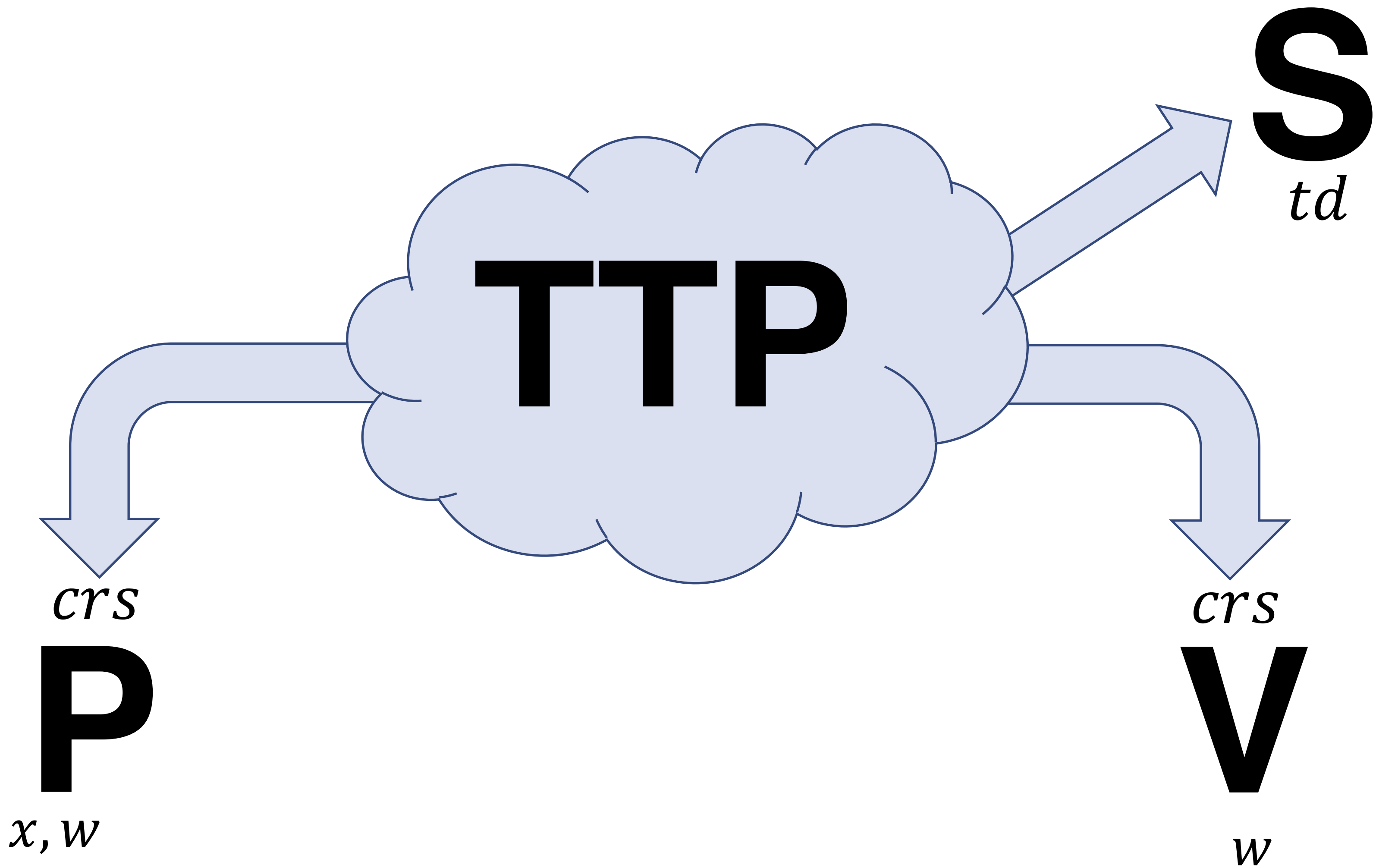
- Alice has sufficient funds
- Bob can spend the funds transferred by Alice

Problem: There is billions transactions on blockchain. How to show possession of funds efficiently?

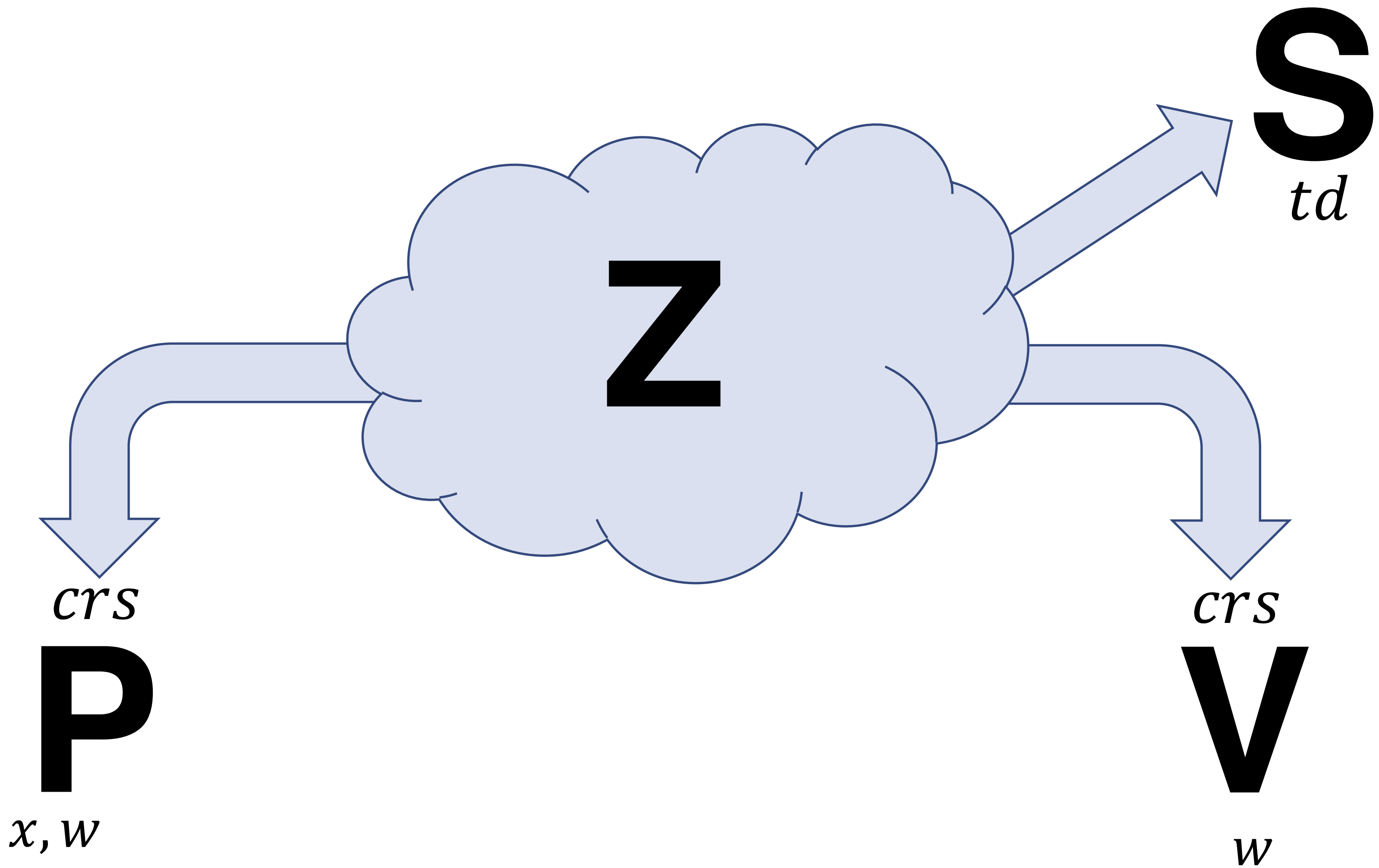
Solution: Use SNARKs!

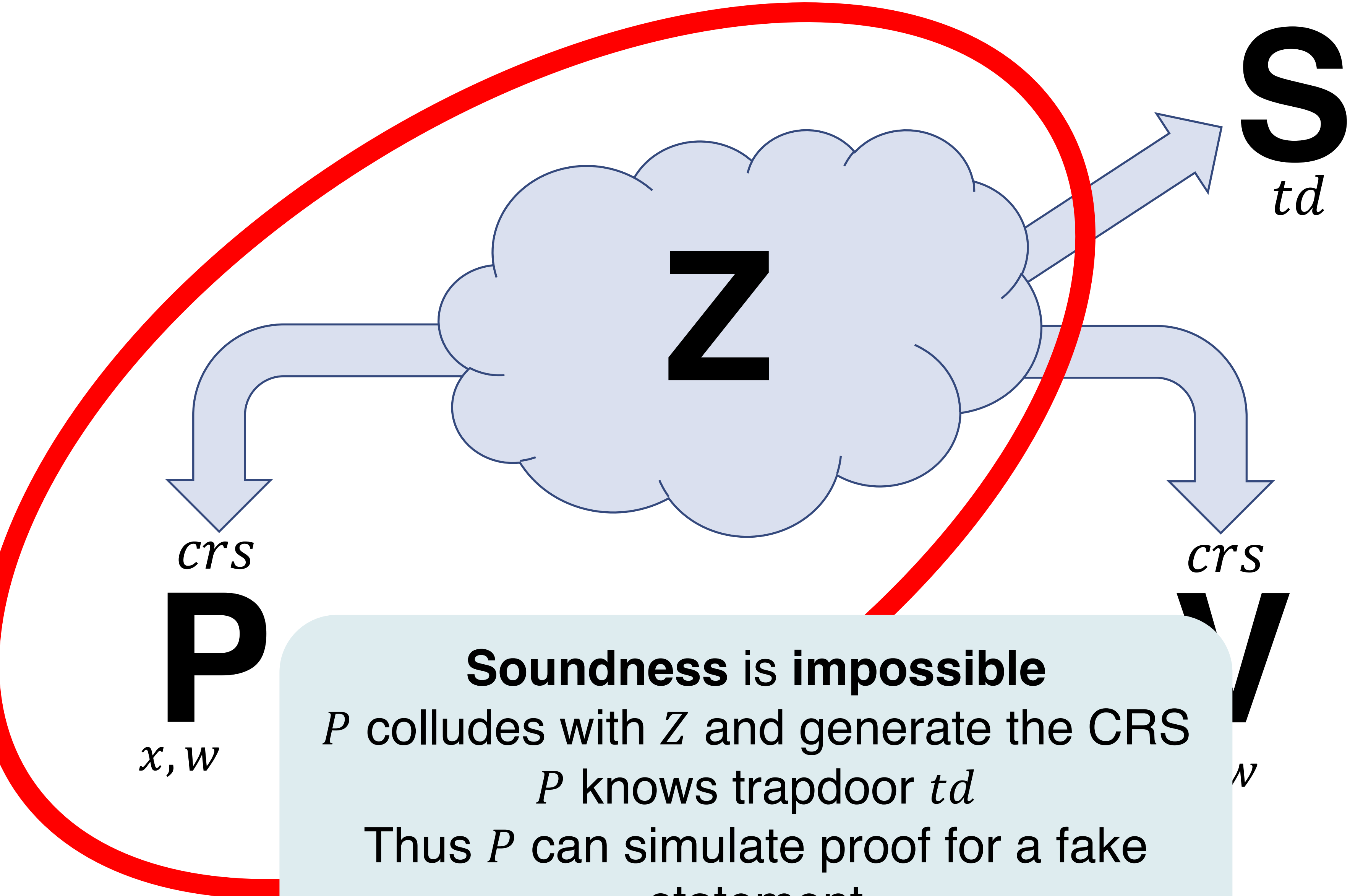
ZCASH

Subversive CRS generation



S
td
TTP
crs
P
x, w
crs
V
w


S
td
Z
crs
P
x, w
crs
V
w

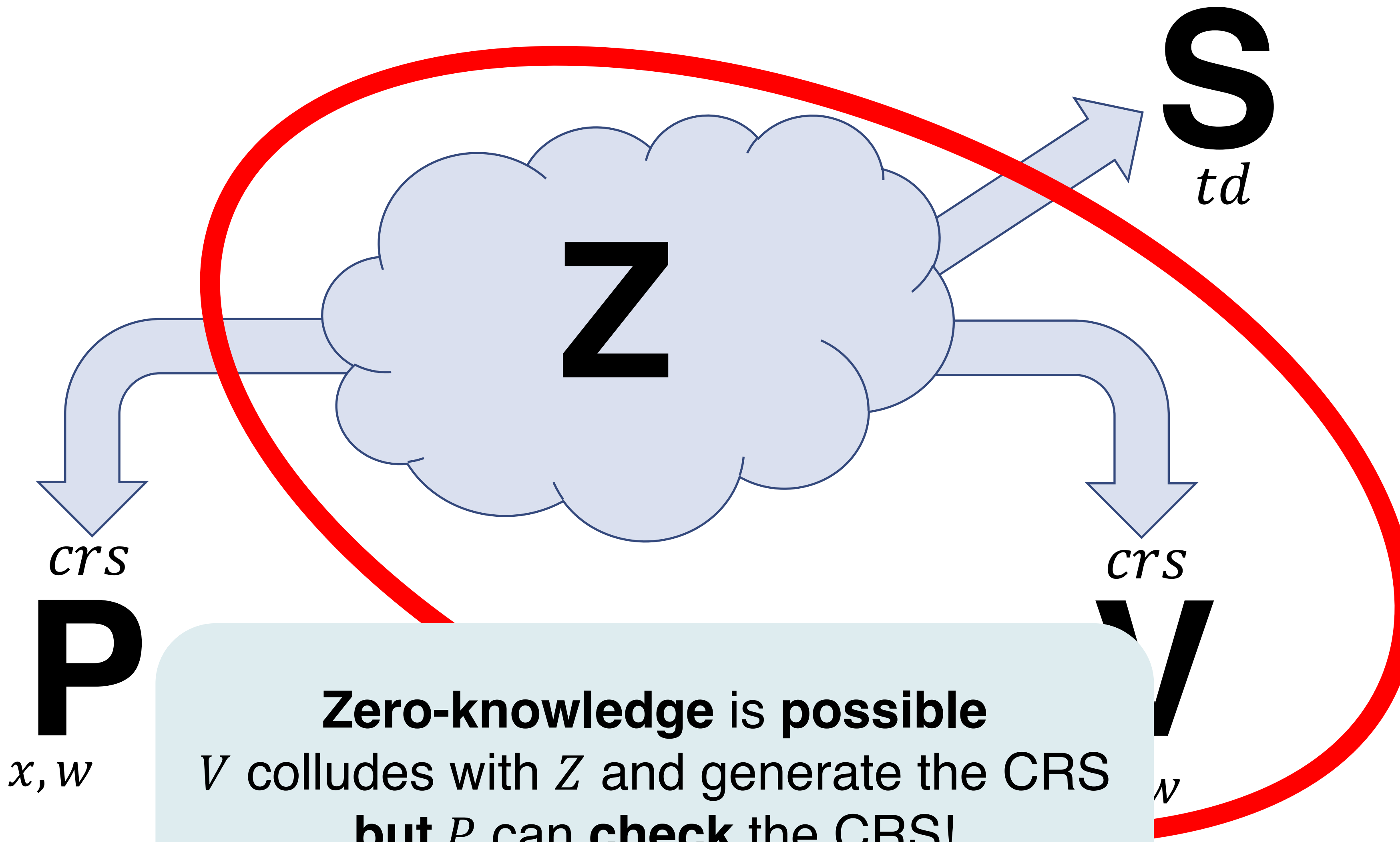


Soundness is **impossible**

P colludes with Z and generate the CRS

P knows trapdoor td

Thus P can simulate proof for a fake statement



Zero-knowledge is **possible**

V colludes with Z and generate the CRS

but P can **check** the CRS!

How to assure S gets the trapdoor?

How to provide S with trapdoor?

Diffie-Hellman knowledge assumption
$h, g^a, h^a \leftarrow \mathcal{A}(g; r)$ then there exists extractor E for $\mathcal{A}$ that $E(g; r)$ outputs a

Idea: Use knowledge assumption on subverter Z.
Intuitively: $crs = f(td)$ and since Z produce crs it knows td

Provide S with the extracted td
Since now S can simulate $\Longrightarrow$ zero-knowledge holds

Falsifiability in cryptography

Knowledge assumptions are not falsifiable ☹

Falsifiability defined as a **GAME** between adversary A and challenger C
C setups the game and answers on A's queries
In the end C returns 0 (A loses) or 1 (A wins)

Assumption (C, c) is **falsifiable** if for all PPT A

$$\Pr[(A, C) = 1] \leq c + negl$$

Fun fact: SNARKs cannot be sound without non-falsifiable assumptions