INDOCRYPT 2016

Faster Zero-Knowledge and Applications

Claudio Orlandi – Aarhus University

Based on joint work with:

- David Derler (TU Graz)
- Tore Frederiksen (BIU)
- Irene Giacomelli (UW-Madison)
- Marek Jawurek (SAP)
- Florian Kerschbaum (SAP)
- Jesper Madsen (AU)
- Jesper Buus Nielsen (AU)
- Sebastian Ramacher (TU Graz)
- Christian Rechberger (TU Graz, DTU)
- Daniel Slamanig (TU Graz)

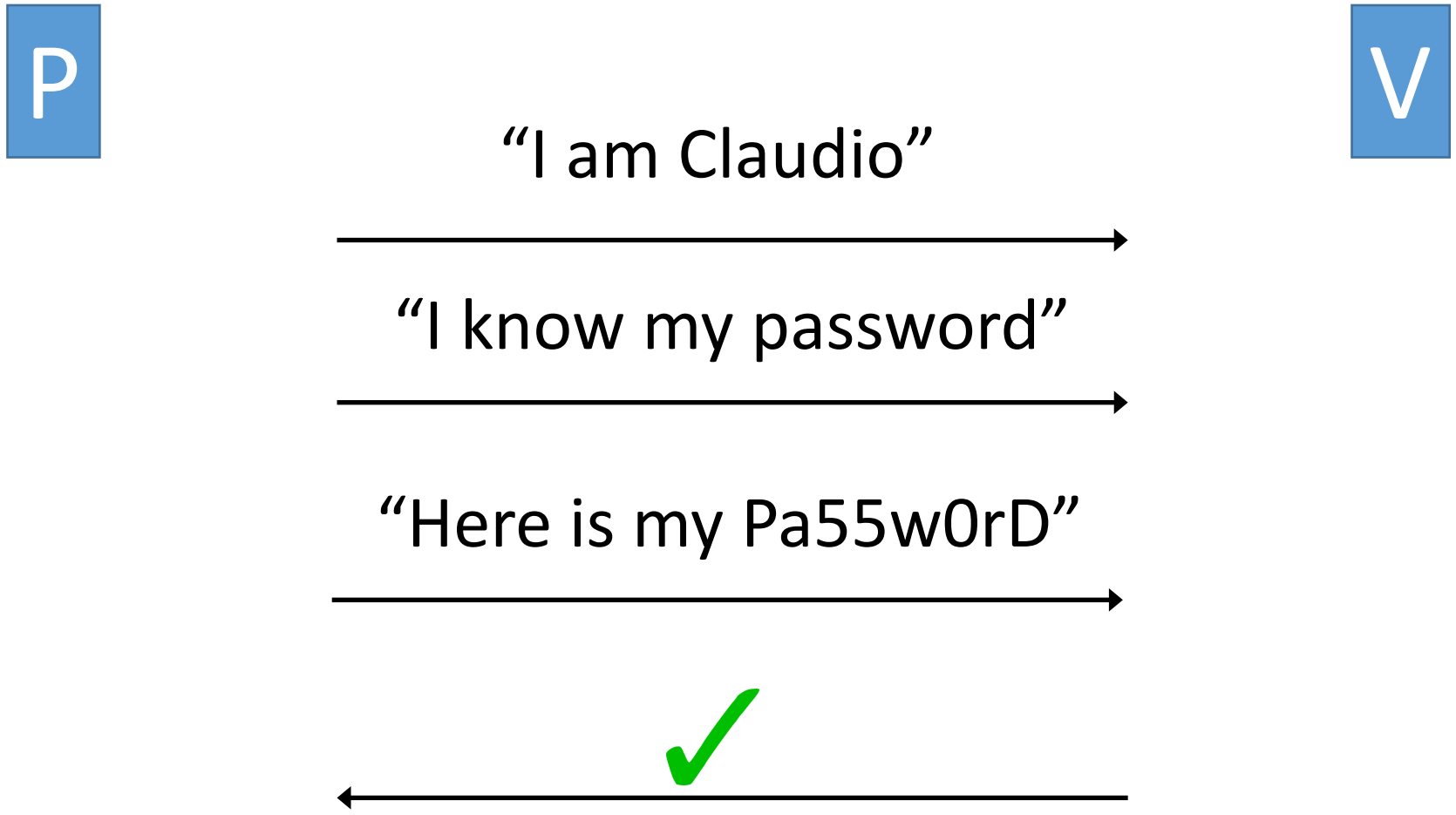




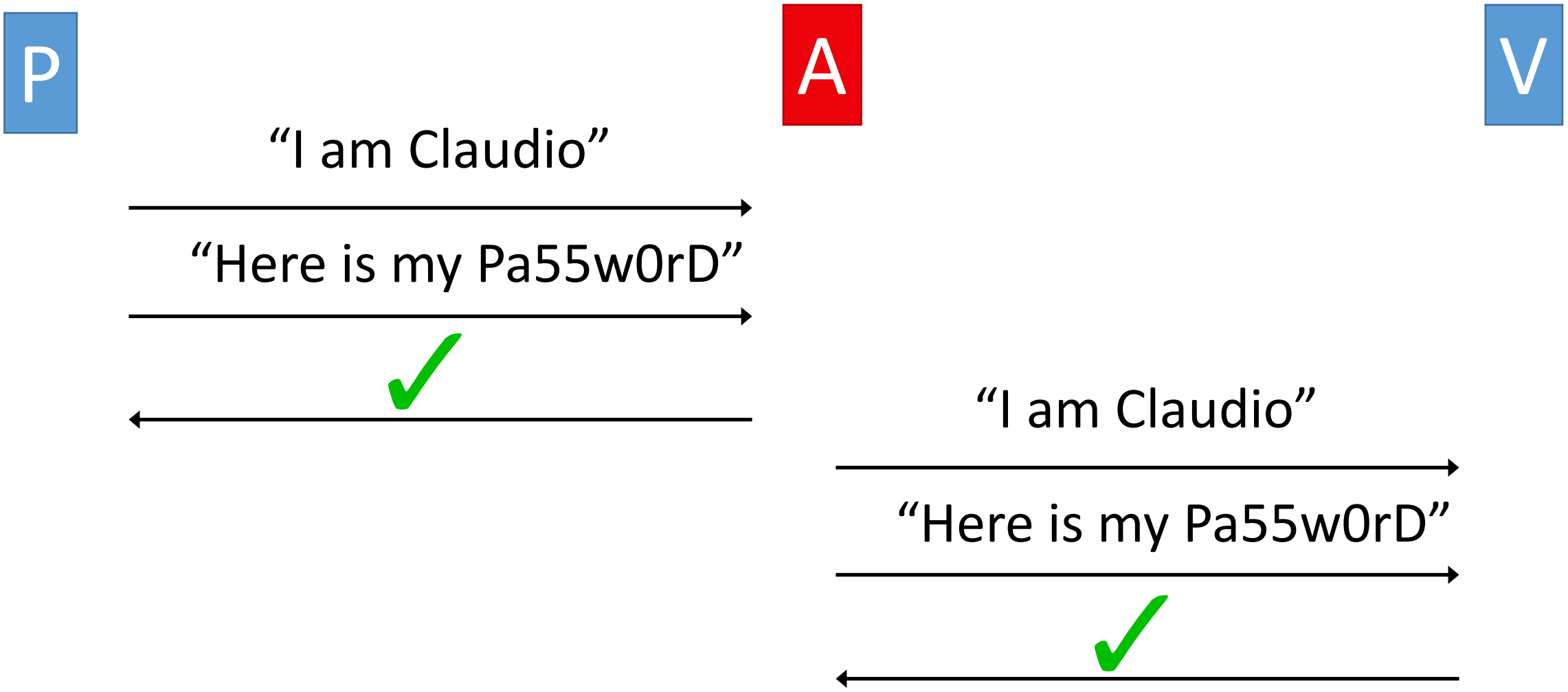




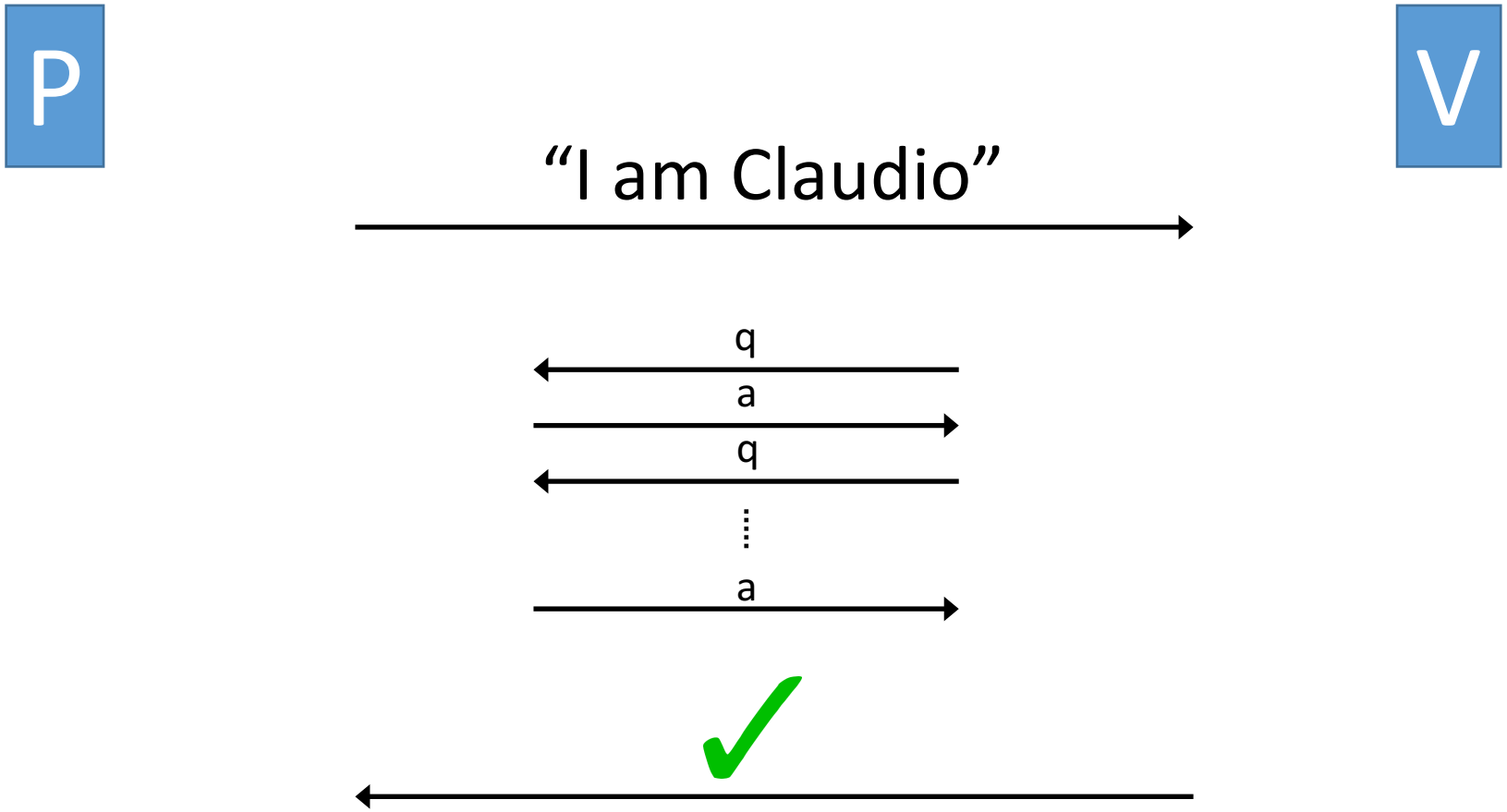

Motivation: Authentication



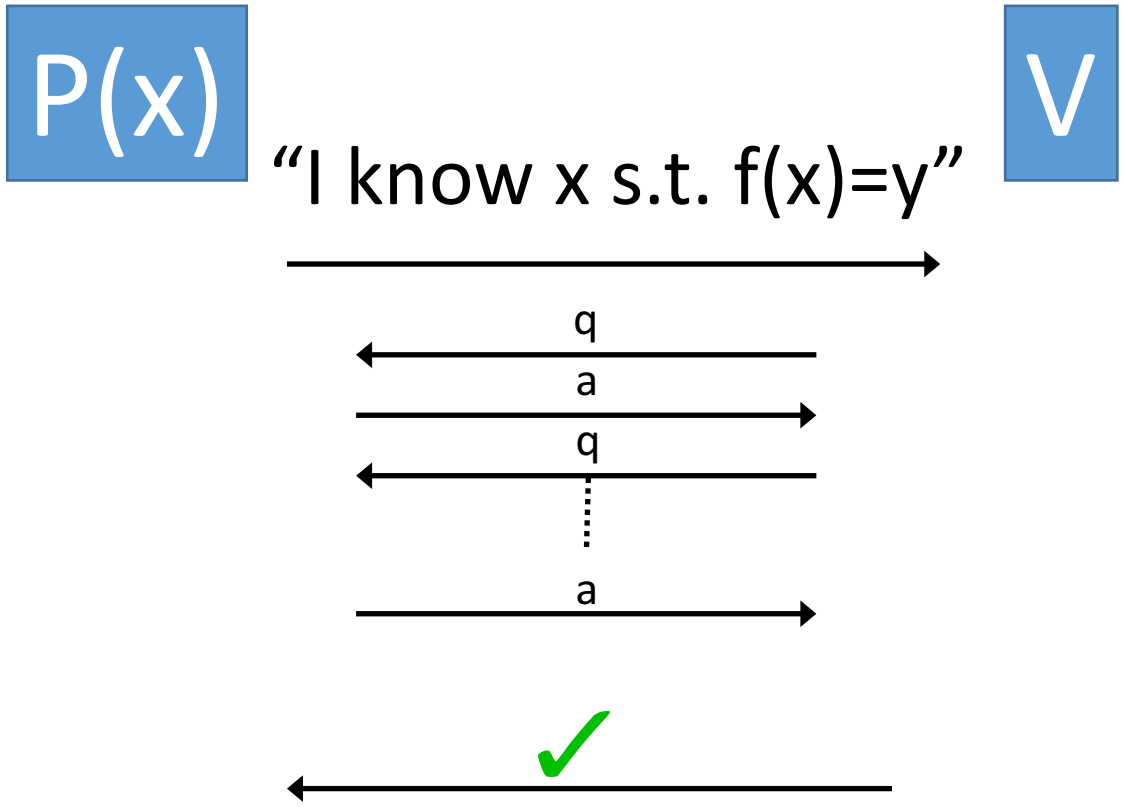
Motivation: Authentication



Motivation: Zero-Knoweldge Authentication



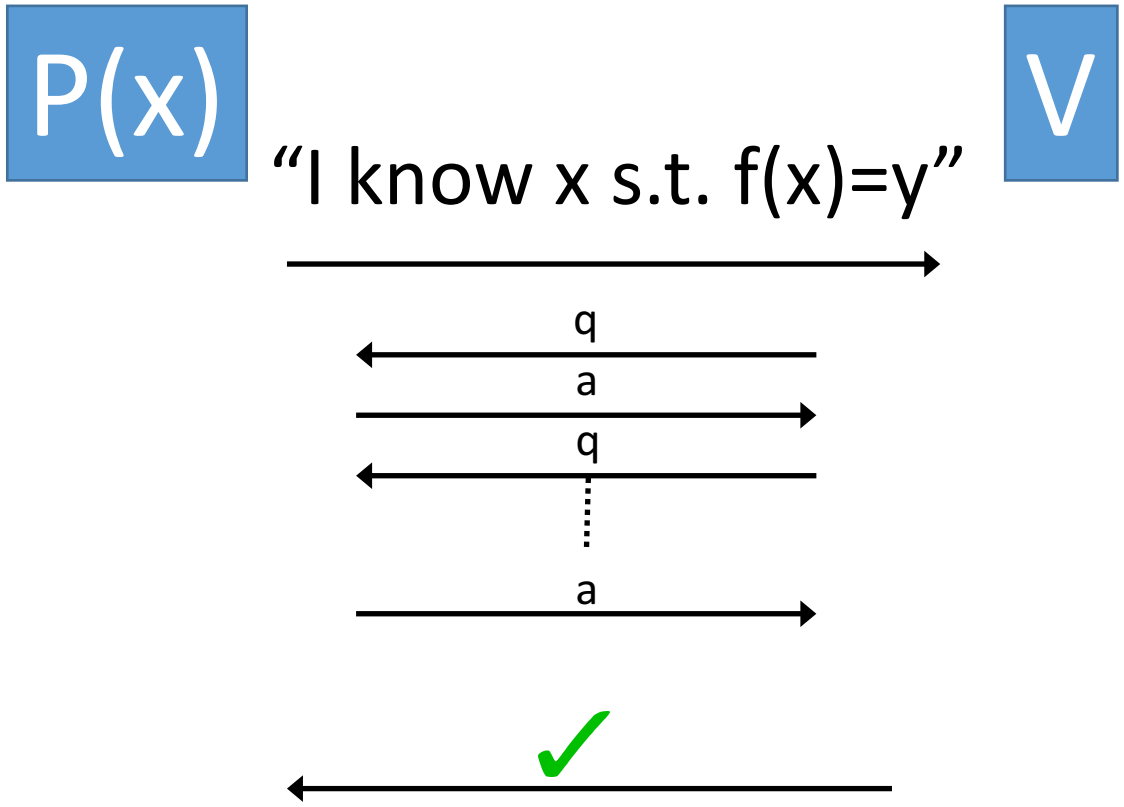
ZK: Definitions



Only P knows x

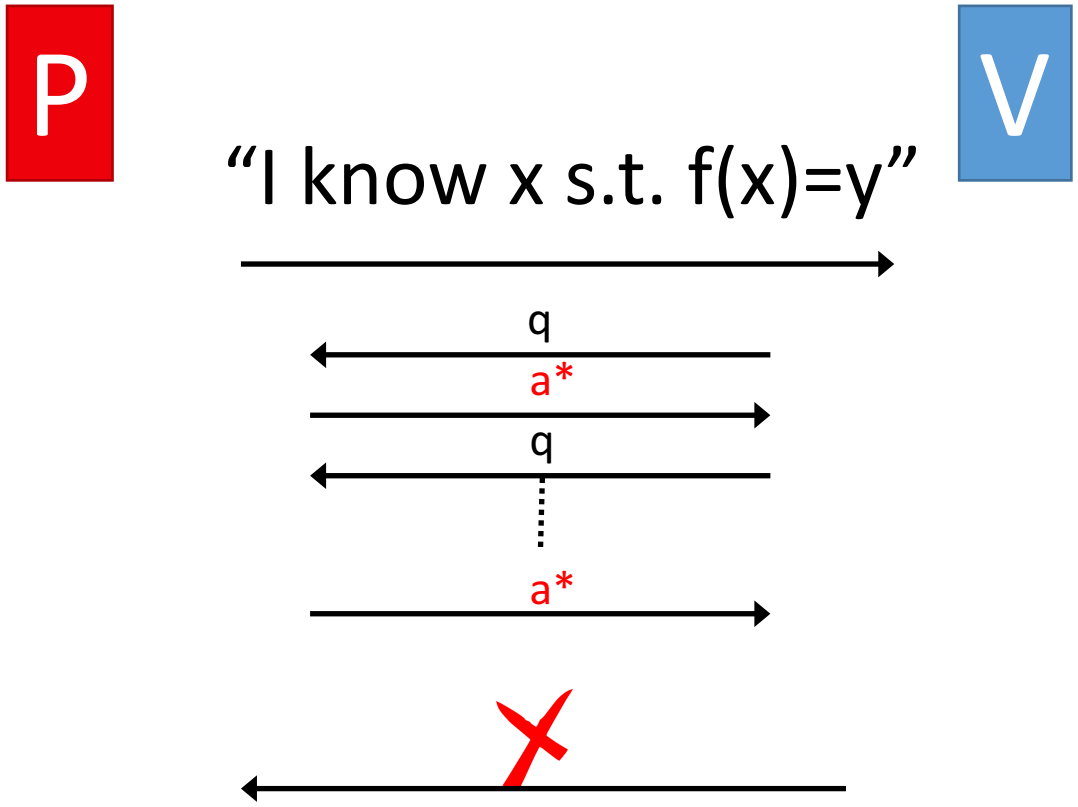
P,V know f,y

ZK: Definitions



- **Completeness**
  - P,V honest → V accepts

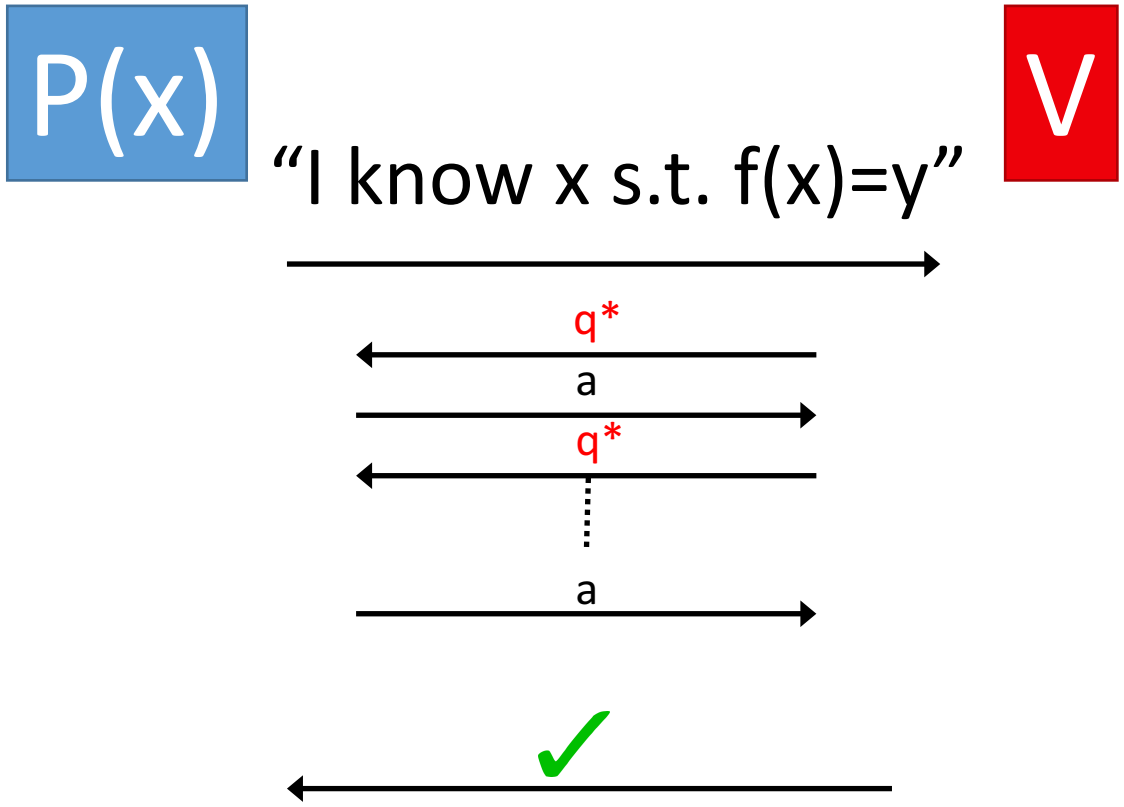
ZK: Definitions



- **Completeness**
  - P,V honest → V accepts

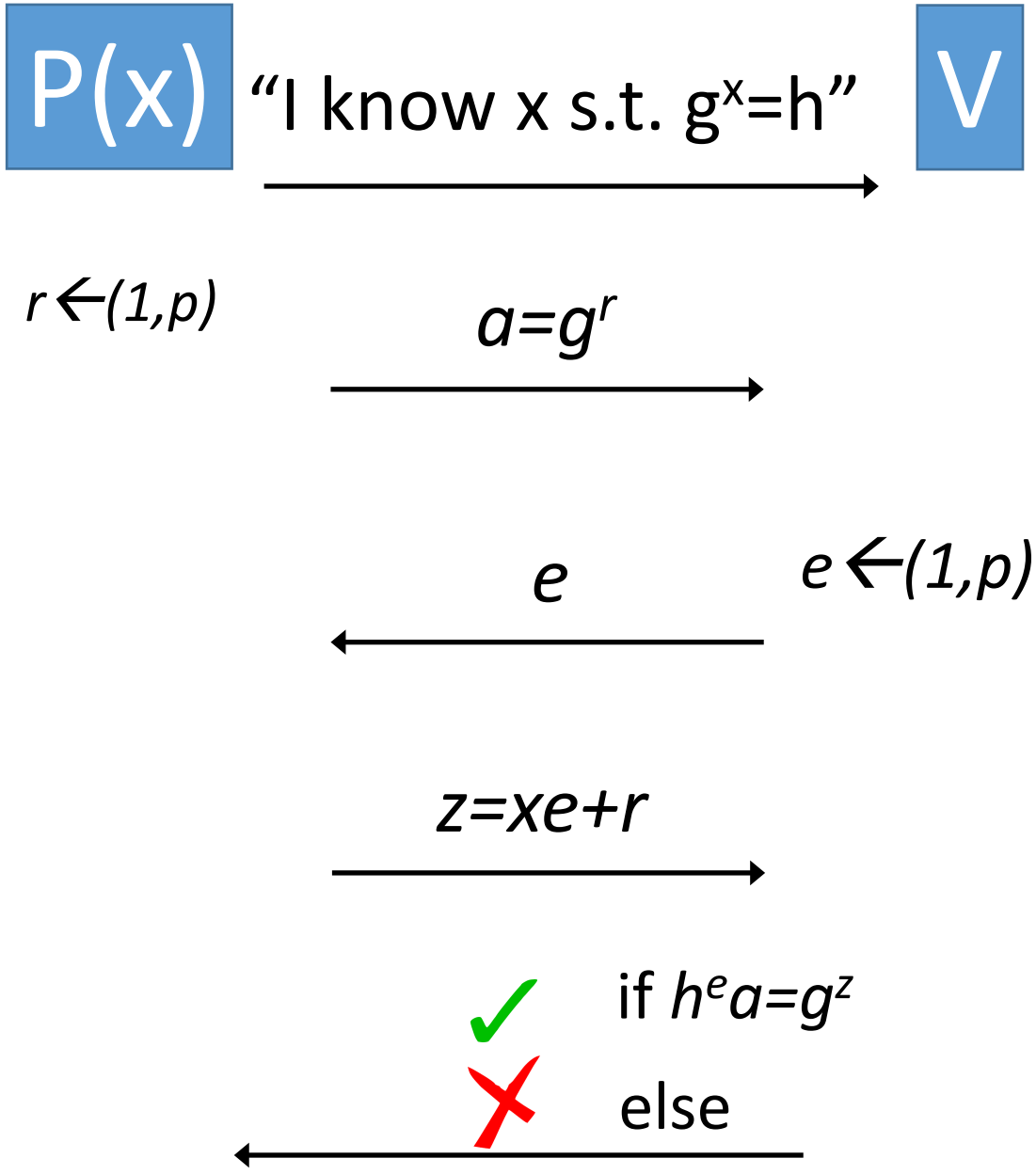
- **Proof-of-Knowledge**
  - If P does not know x → V rejects

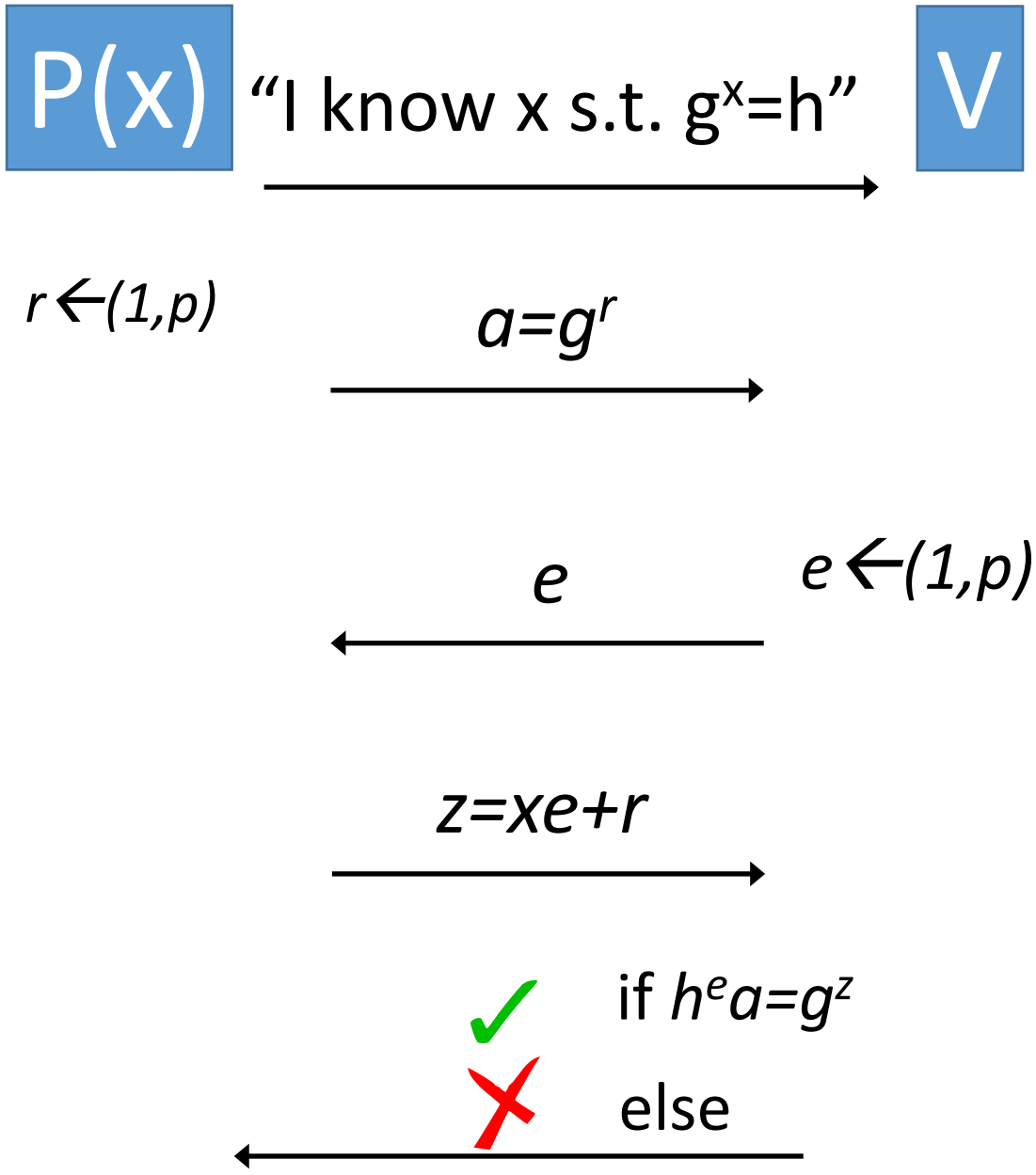
ZK: Definitions



- **Completeness**
  - P,V honest → V accepts
- **Proof-of-Knowledge**
  - If P does not know x → V rejects
- **Zero-Knowledge**
  - V learns nothing about x

Example: Schnorr Protocol

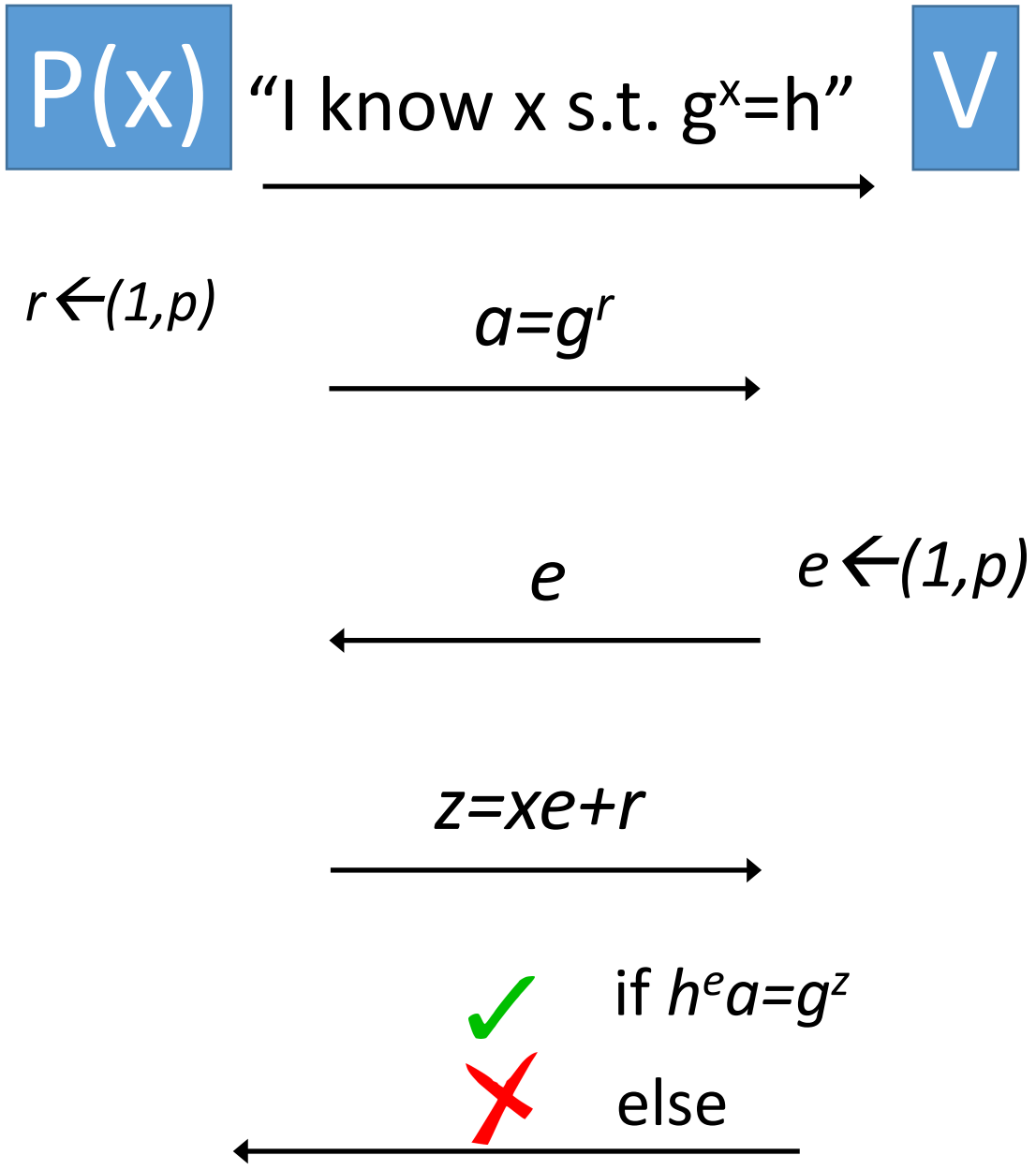




Example: Schnorr Protocol

Completeness

$$g^z = g^{xe+r} = h^e g^r = h^e a$$



Example: Schnorr Protocol

~~**Proof-of-Knowledge**~~

Special Soundness:

From two accepting transcripts

$$(a_1, e_1, z_1), (a_2, e_2, z_2)$$

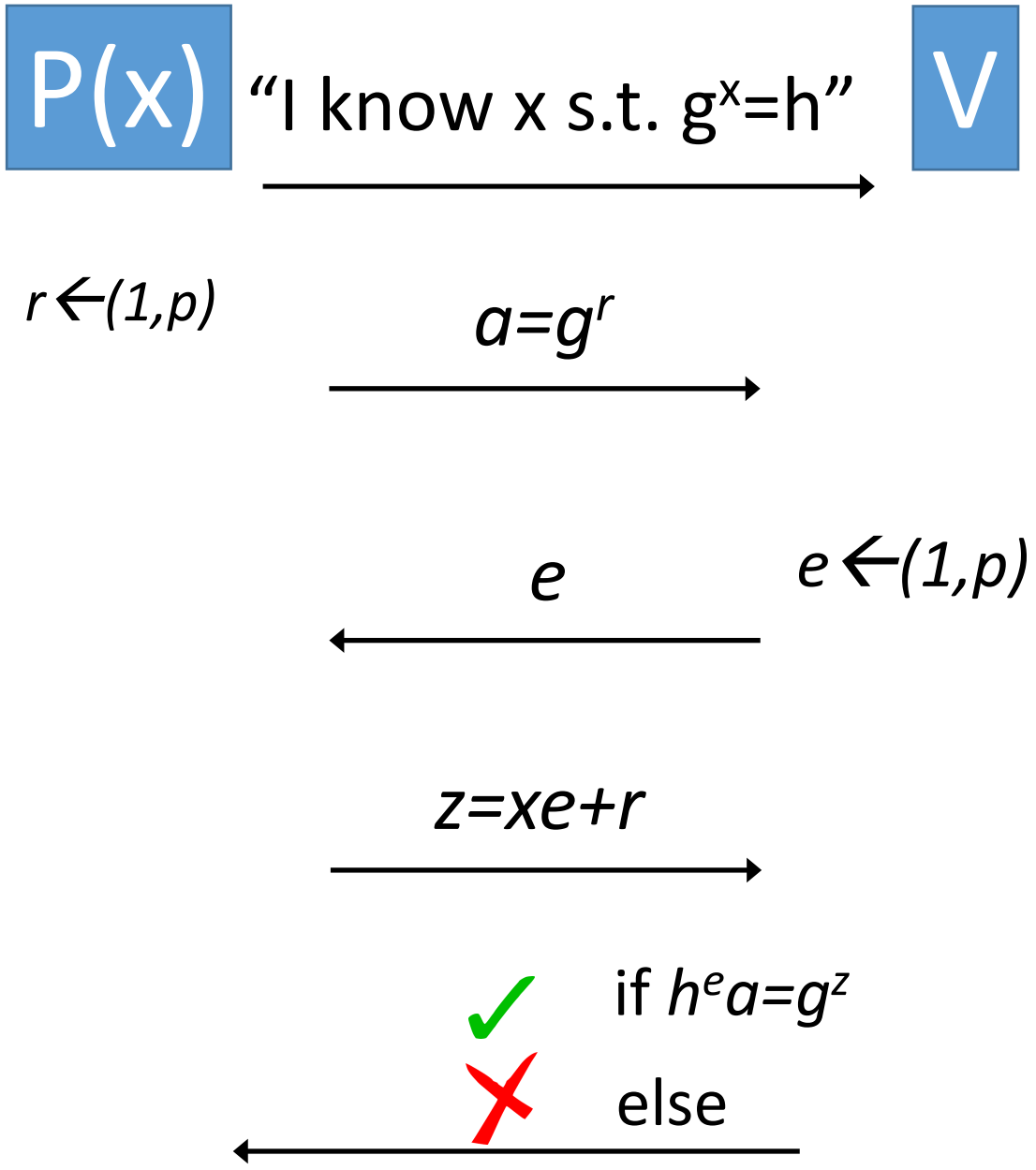
with $a_1=a_2$ we can **extract** x.

Solve:

$$z_1 = xe_1 + r,$$

$$z_2 = xe_2 + r$$

(P can answer 2 different challenges → P knows x)



Example: Schnorr Protocol

(Honest Verifier) Zero-Knowledge

The transcript can be **simulated** without knowing *x*
(hence, it contains no informaiton about x)

Simulator

1. Pick e ← (1,p)
2. Pick z ← (1,p)
3. Compute a = h^e/g^z
4. Output (a,e,z)

What can be proven in ZK?

Feasability: NP, even PSPACE!

Efficiently: algebraic languages
(Schnorr, …, Groth-Sahai, …)

SNARKS (generic)

- **Short proofs, efficient verification ☺**
- **Slow prover ☹**
- Implementations: Pinocchio, libsnark,

This talk:

Can we construct efficient proofs for non-algebraic languages such as

"I know x such that SHA(x)=y"?

Two protocols:

- *ZKGC (from Garbled Circuits)*
- *ZKBoo (from MPC)*

One application:

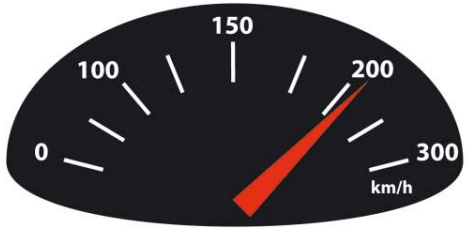
- *Generic (post-quantum) signatures*

The Crypto Toolbox

Weaker assumption

Stronger assumption

OTP >> SKE >> PKE >> FHE >> Obfuscation



More efficient

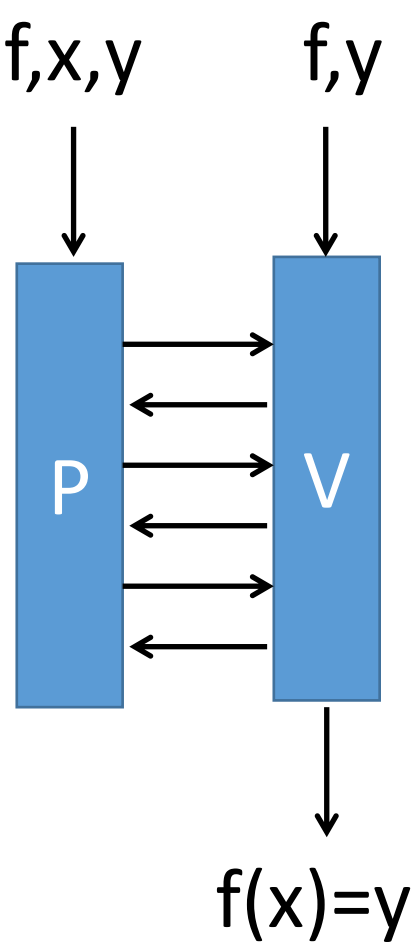
Less efficient

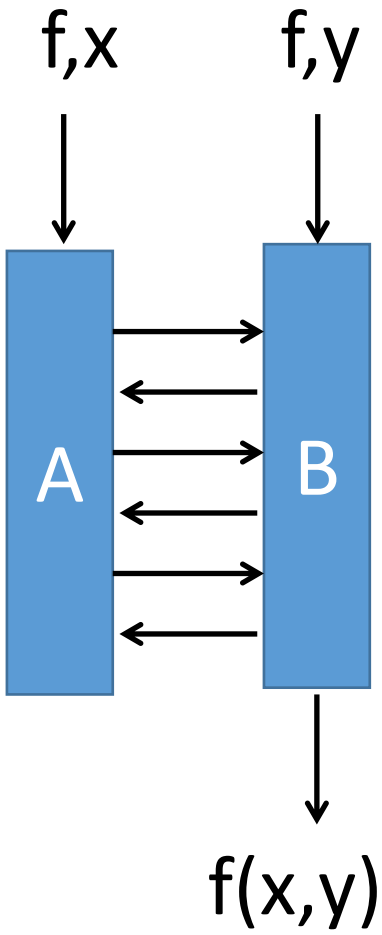
Zero-Knowledge from Garbled Circuits

Jawurek, Ferschbaum, Orlandi

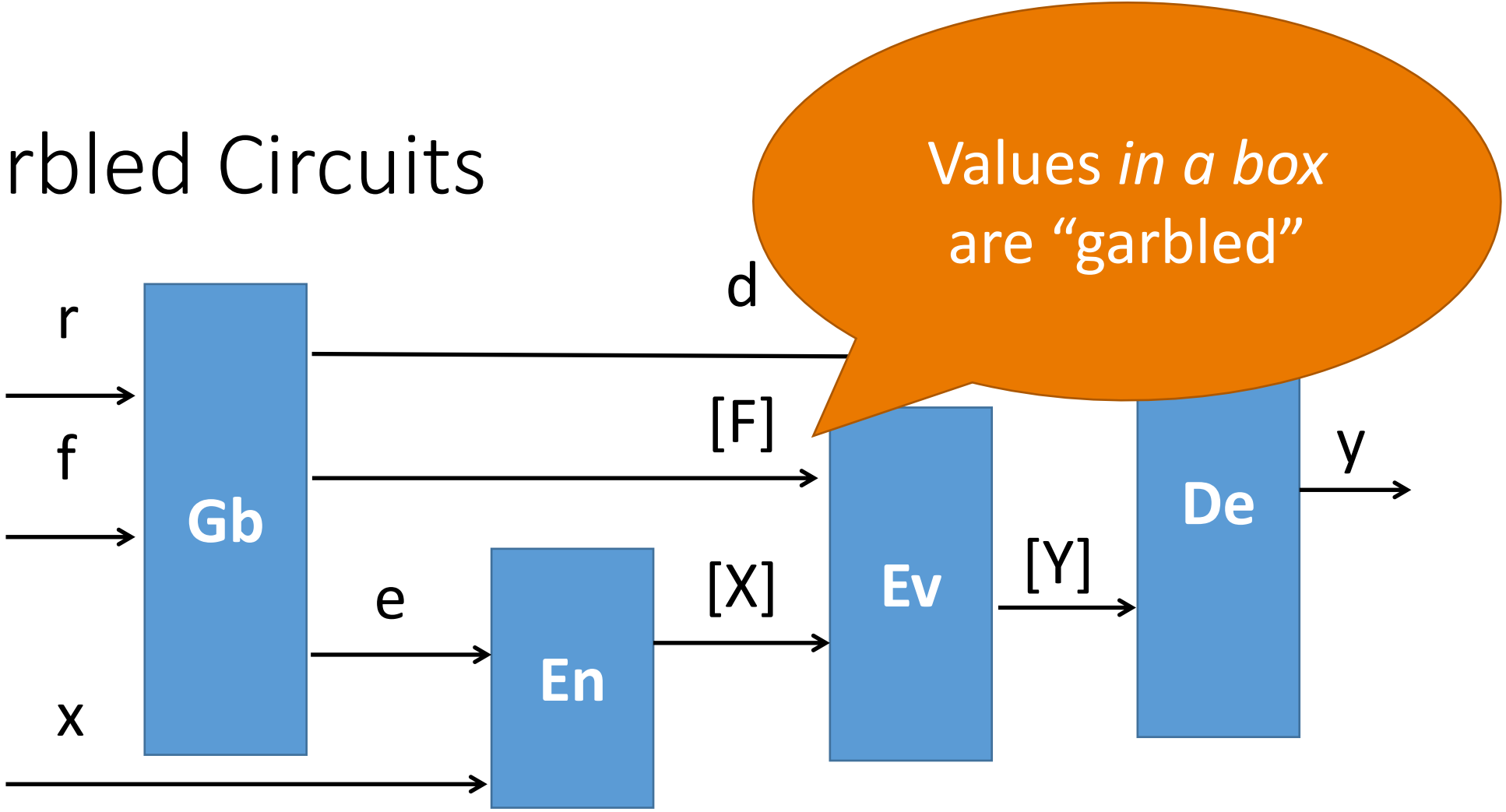
CCS 2013

Zero-Knowledge vs Secure 2PC



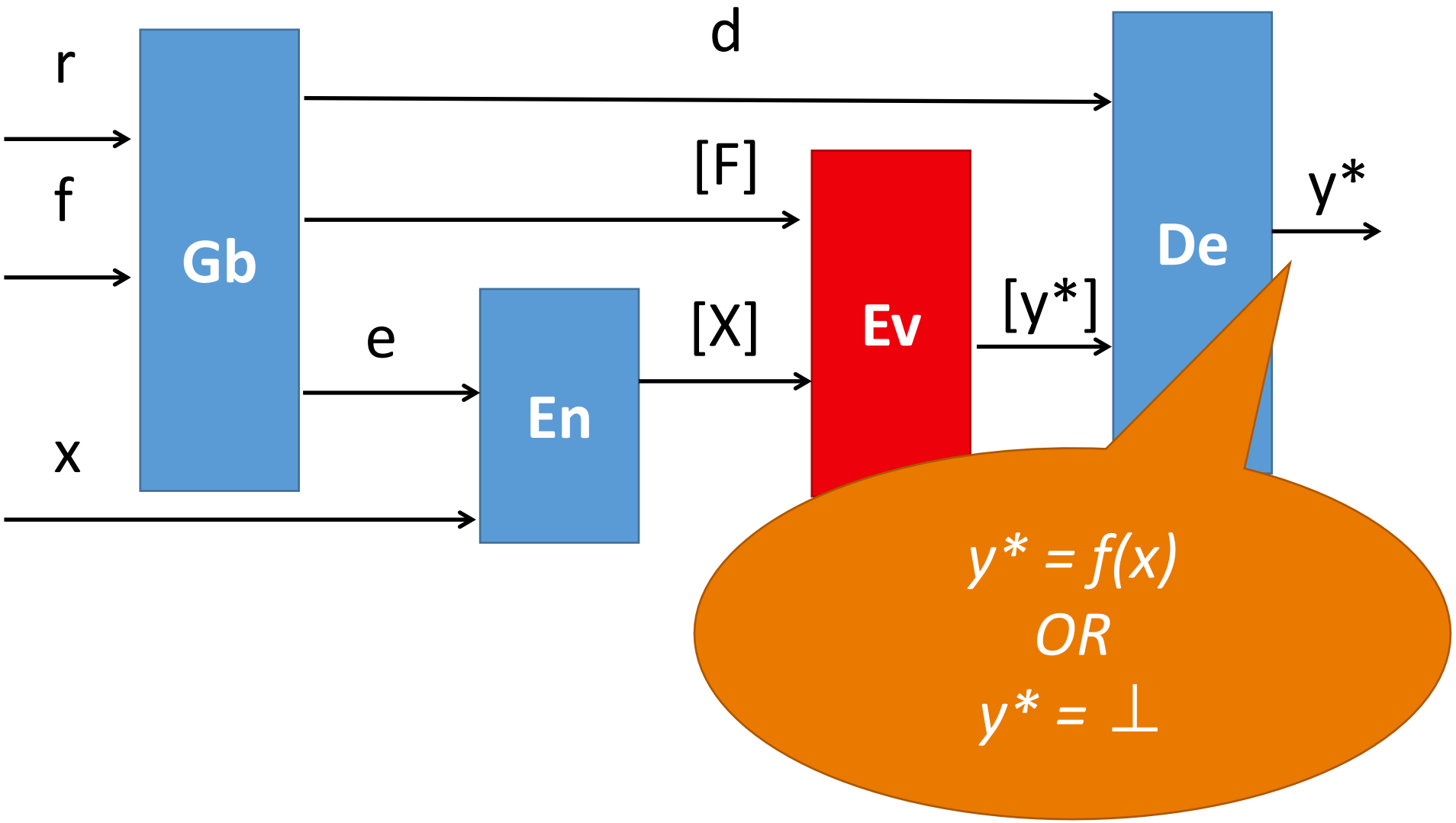


Garbled Circuits

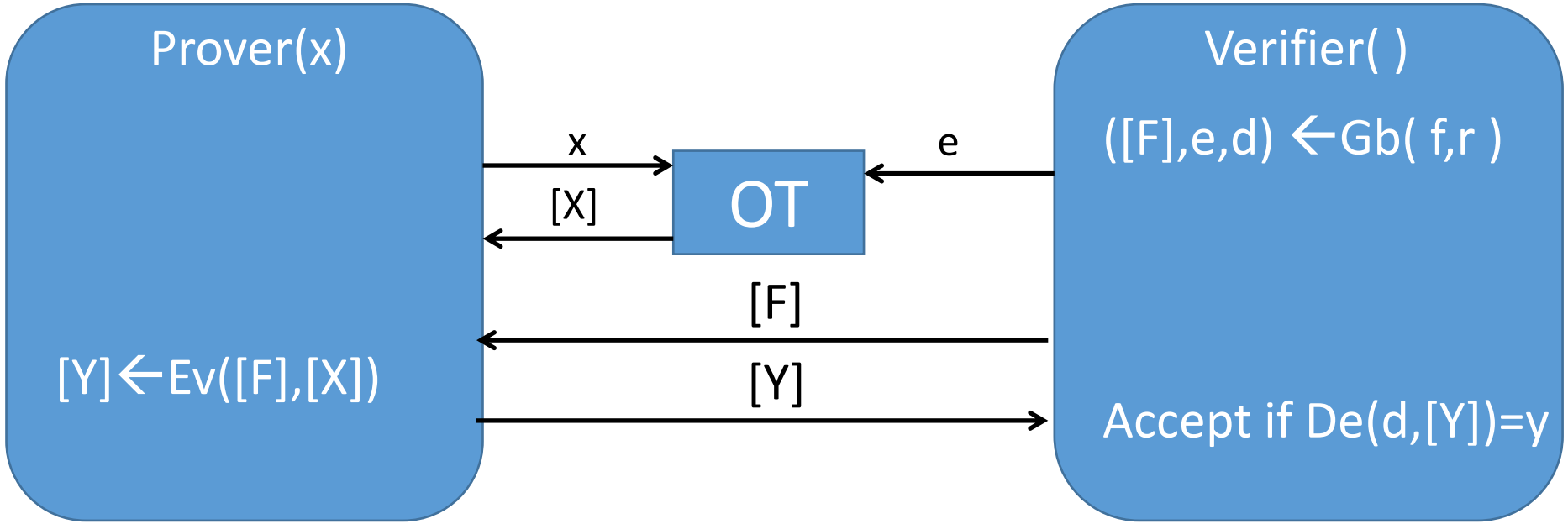


Correct if y=f(x)

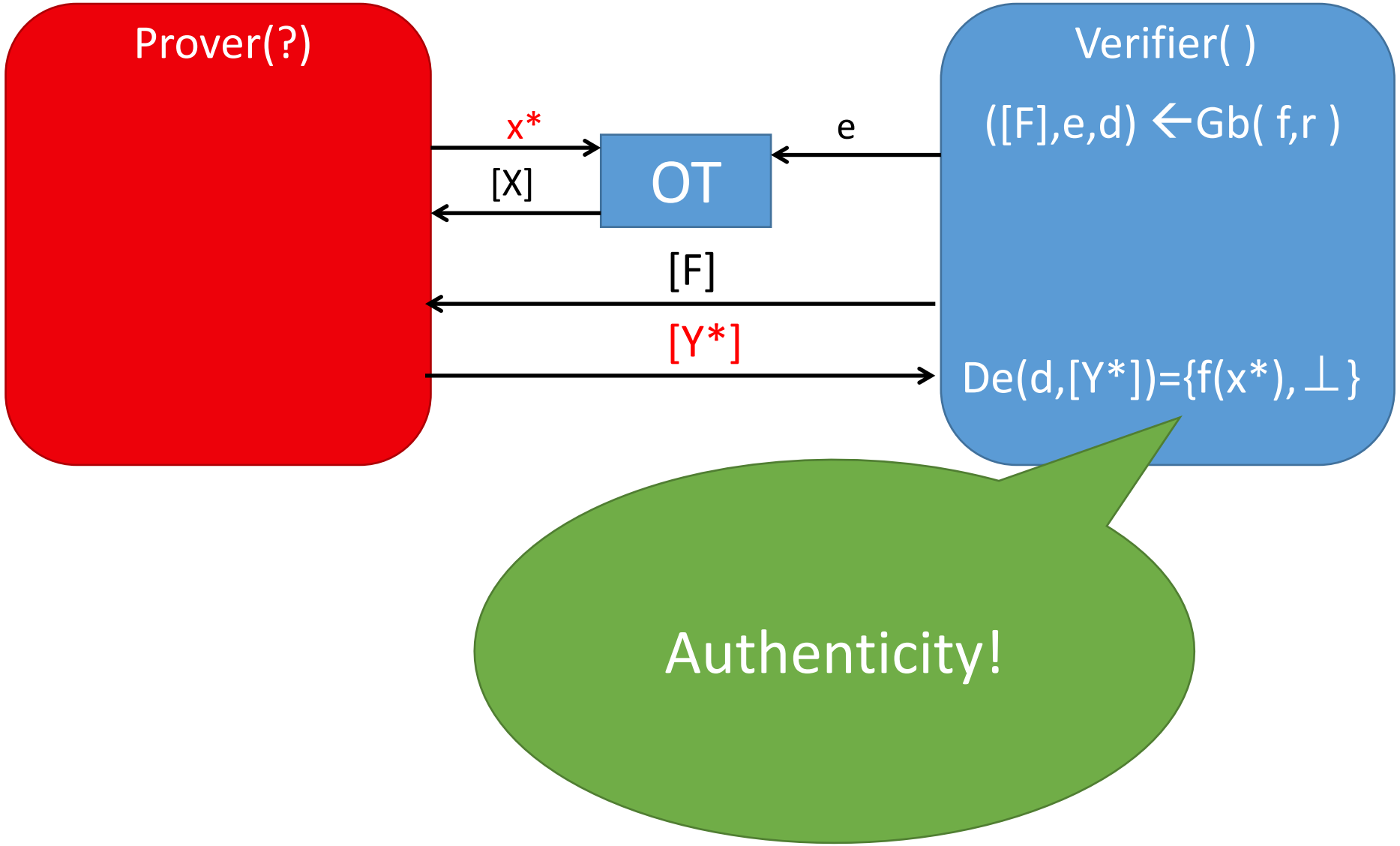
Garbled Circuits: Authenticity



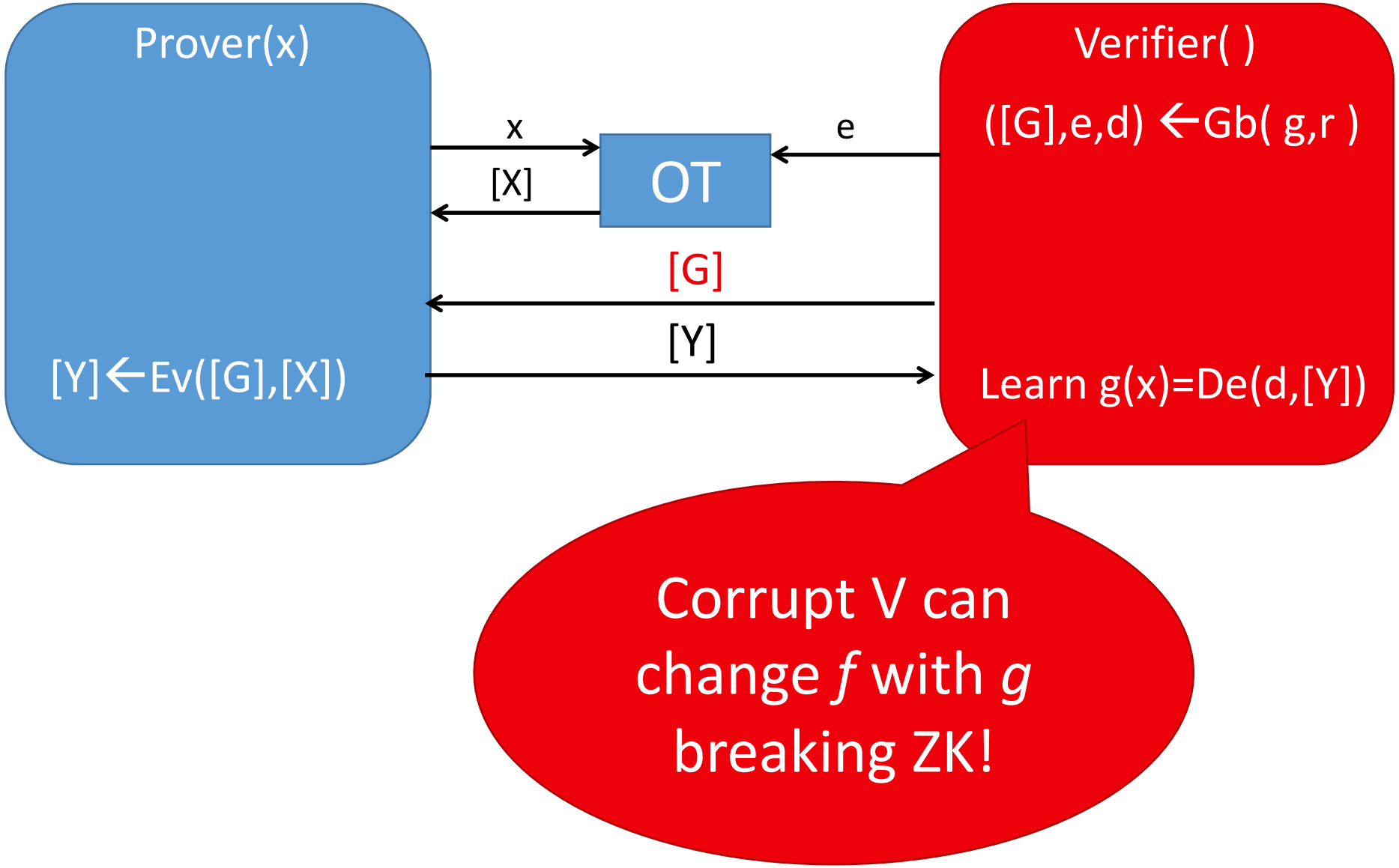
(HV)ZKGC to prove f(x)=y



(HV)ZKGC to prove f(x)=y



(HV)ZKGC to prove f(x)=y



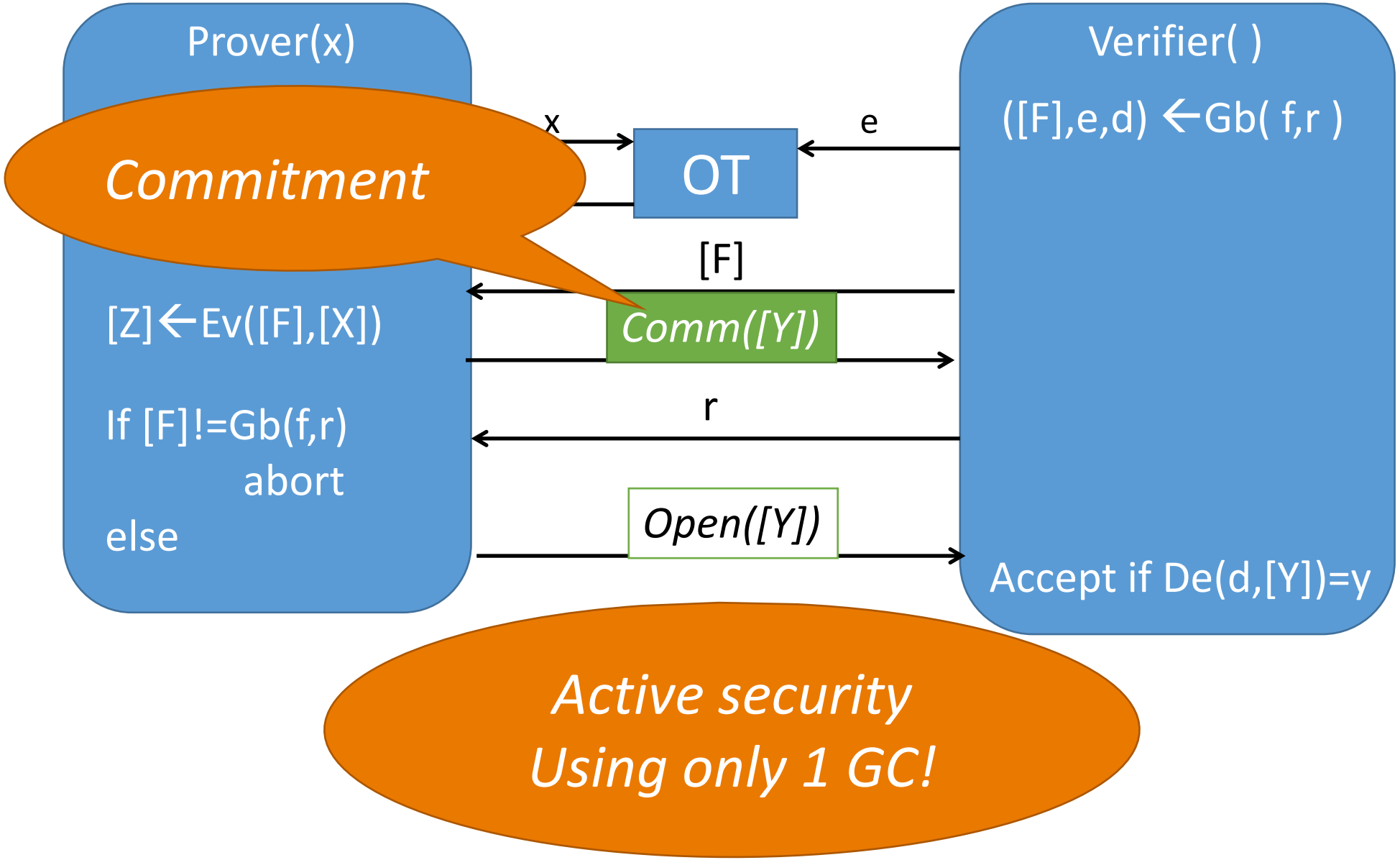
Garbled circuits with active security?

How can the verifier prove that f was garbled correctly

(without breaking soundness)?

- Plenty of (costly) solutions are known for 2PC
  - Zero-Knowledge
  - Cut-and-choose
  - Etc.
- **Can we do better for ZK?**

ZKGC to prove f(x)=y



Recap: ZK based on GC

- **The main idea:**
  - In ZK the verifier (Bob) has no secrets!
  - After the protocol, Bob can reveal all his randomness.
  - Alice can simply check that Bob behaved honestly

    by *redoing his entire computation*.

Privacy-Free Garbled Circuits

Frederiksen, Nielsen, Orlandi

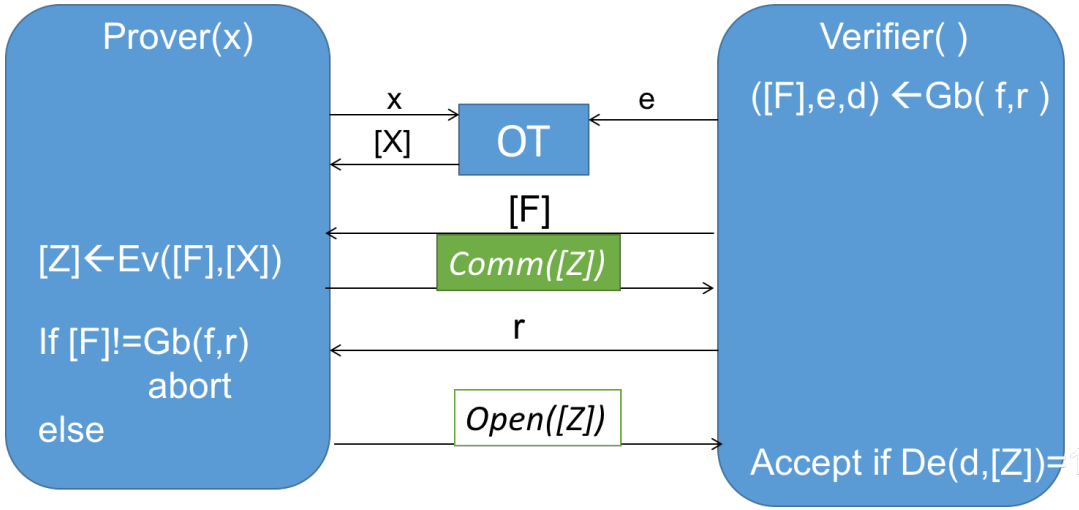
EUROCRYPT 2015

Main idea

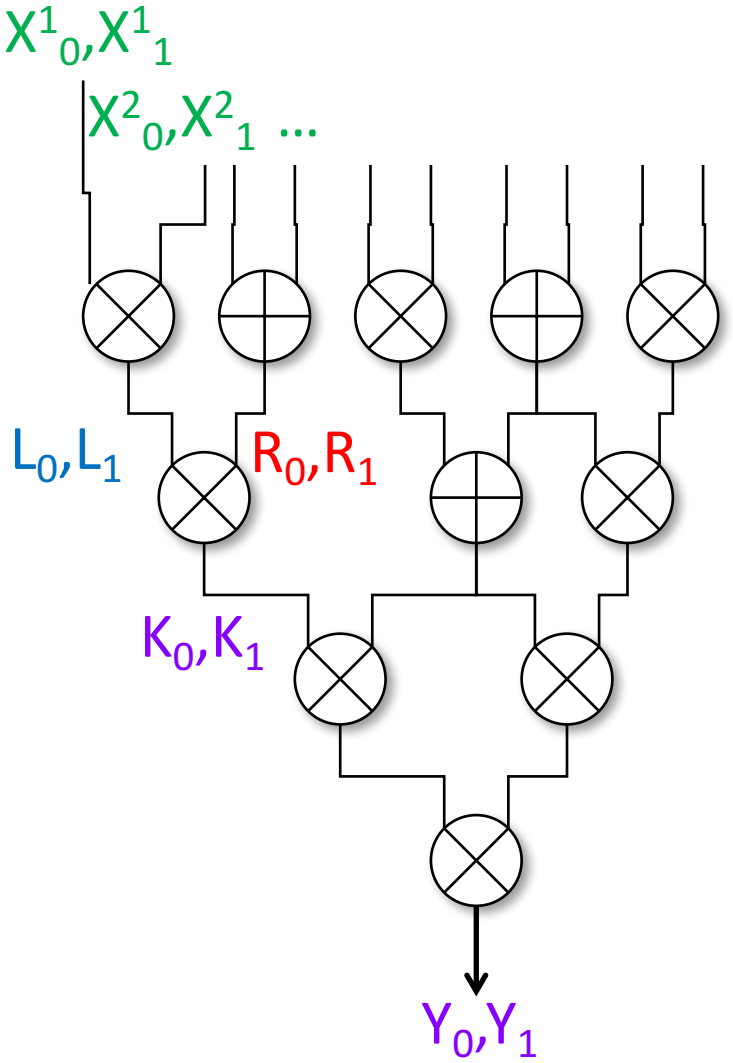
- In 2PC the garbler has **secret input**
- GC privacy --> privacy of input

- In ZK V has **no input to protect**
- Can we get more efficient GC without privacy?

Yes!



Garbling a Circuit : ([F],e,d)← Gb(f)



- Choose 2 random keys X^i_0, X^i_1 for each input wire
- For each gate g compute
  - $(gg, K_0, K_1) \leftarrow Gb(g, L_0, L_1, R_0, R_1)$
- Output
  - e=(X^i_0, X^i_1) for all input wires
  - d=(Y_0, Y_1)
  - [F]=(gg^i) for all gates i

Encoding and Decoding

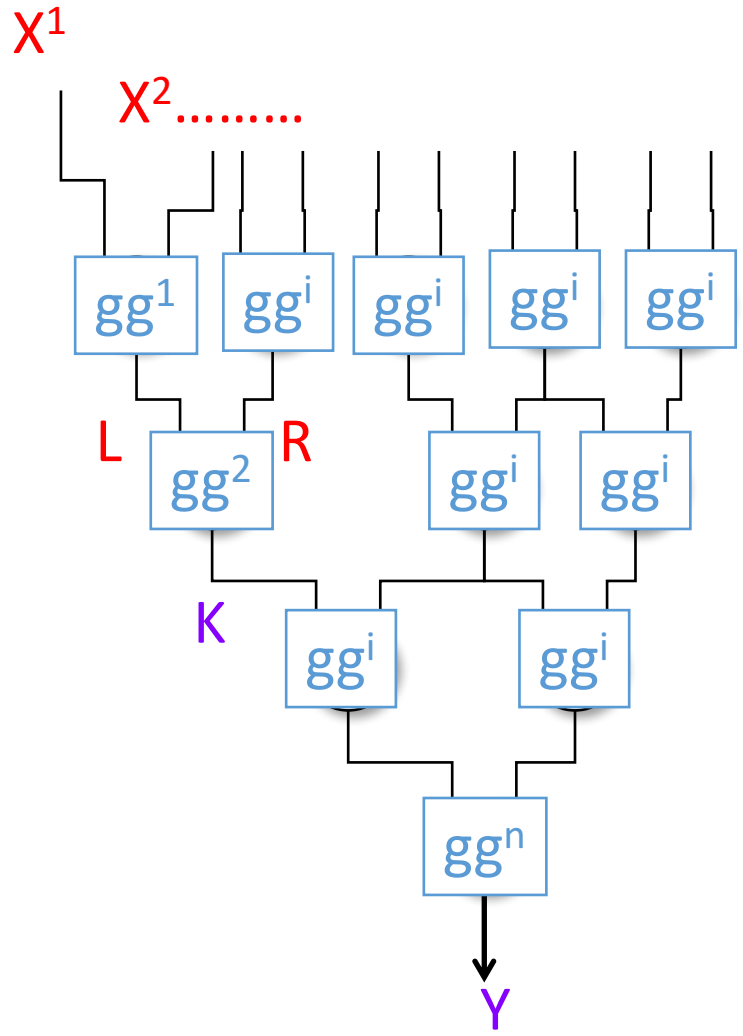
$[X] = En(e,x)$

- $e=\{ X^i_0, X^i_1\}$
- $x= \{ x_1,\ldots,x_n \}$
- $[X]=\{X^1_{x1},\ldots,X^n_{xn}\}$

$y=De(d,[Y])$

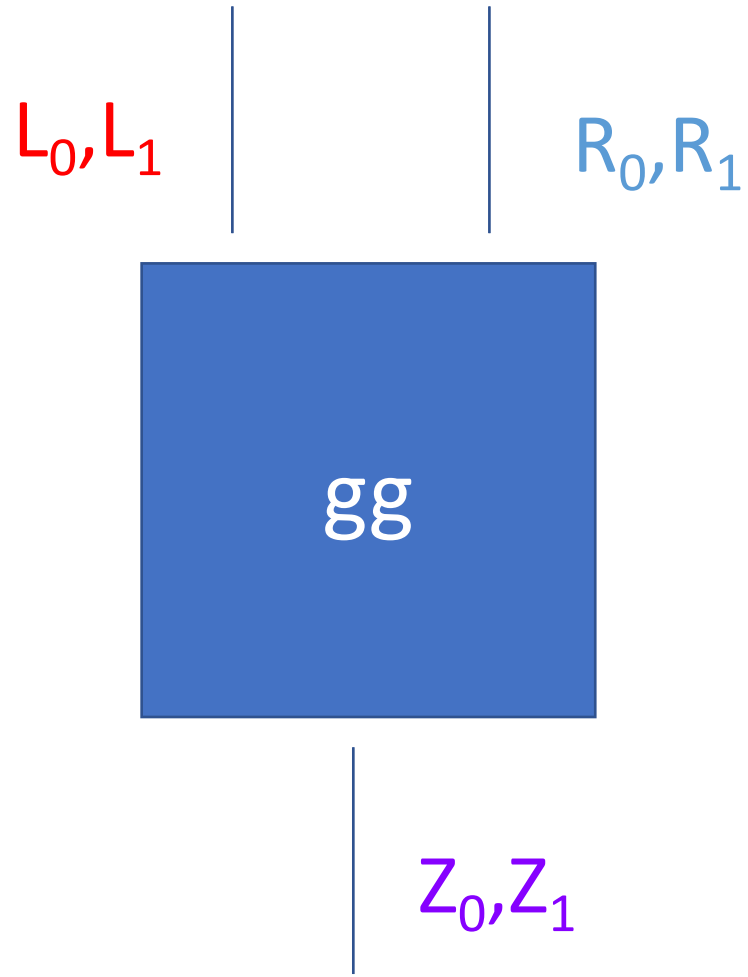
- $d = \{ Y_0,Y_1 \}$
- $[Y] = \{ K \}$
- $y=$
  - 0 if $K=Y_0$,
  - 1 if $K=Y_1$,
  - "abort" else

Evaluating a GC : [Y]← Ev([F],[X])



- Parse [X]={X^1,…,X^n} // x is known
- Parse [F]={gg^i}
- For each gate i compute
  - $K_{g(a,b)}$ ← Ev(gg^i,L,a,R,b) //a,b known!
- Output
  - Y //y is known!

Notation



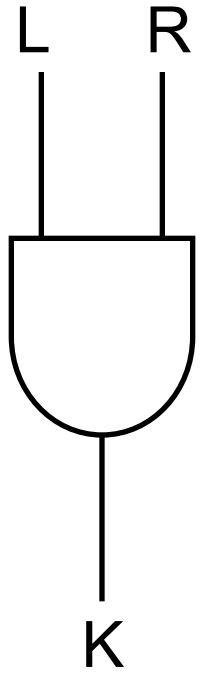
- A *(privacy-free)* garbled gate is a gadget that given two inputs keys gives you the right output key *(and nothing else)*

- $(gg, Z_0, Z_1) \leftarrow Gb(g, L_0, L_1, R_0, R_1)$
- $Z_{g(a,b)} \leftarrow Ev(gg, L, a, R, b)$

- //and not $Z_{1-g(a,b)}$

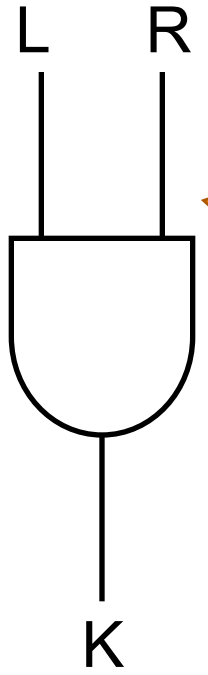
Yao Garbling

C
$C_1 = H(L_0,R_0) \oplus K_0$
$C_2 = H(L_0,R_1) \oplus K_0$
$C_3 = H(L_1,R_0) \oplus K_0$
$C_4 = H(L_1,R_1) \oplus K_1$



Yao Garbling

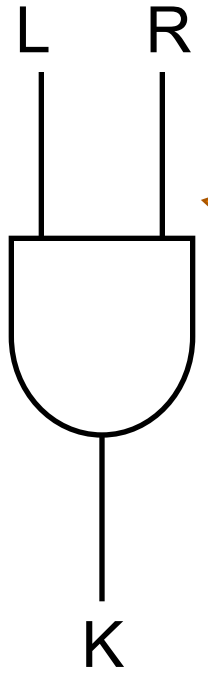
C
$C_1 = H(L_0,R_0) \oplus K_0$
$C_2 = H(L_0,R_1) \oplus K_0$
$C_3 = H(L_1,R_0) \oplus K_0$
$C_4 = H(L_1,R_1) \oplus K_1$



If output is 0 the evaluator should not know why!!!

Privacy-Free Garbling

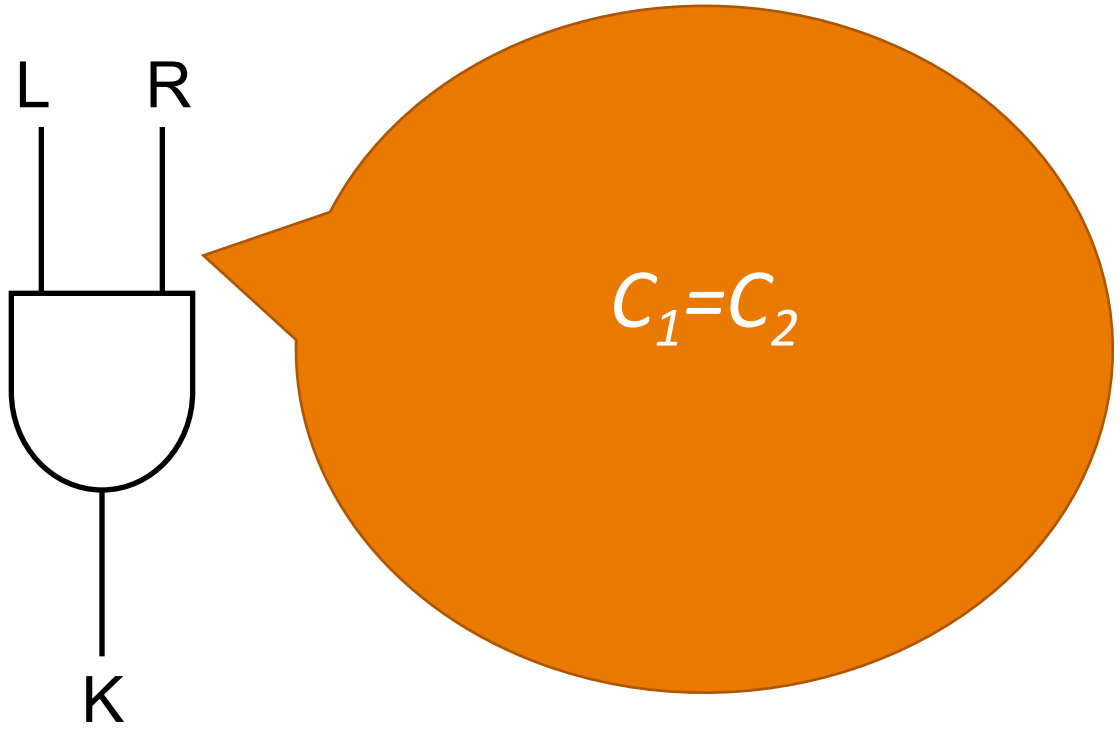
C
$C_1 = H(L_0,R_0) \oplus K_0$
$C_2 = H(L_0,R_1) \oplus K_0$
$C_3 = H(L_1,R_0) \oplus K_0$
$C_4 = H(L_1,R_1) \oplus K_1$



Evaluator knows plain inputs/outputs

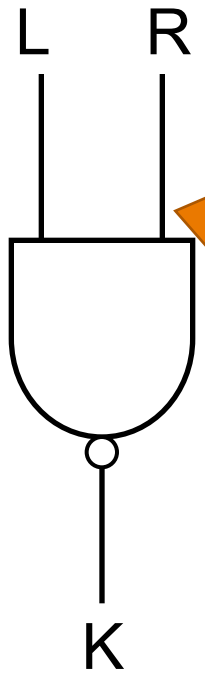
Privacy-Free Garbling

C
$C_1 = H(L_0) \oplus K_0$
$C_2 = H(L_0) \oplus K_0$
$C_3 = H(L_1,R_0) \oplus K_0$
$C_4 = H(L_1,R_1) \oplus K_1$



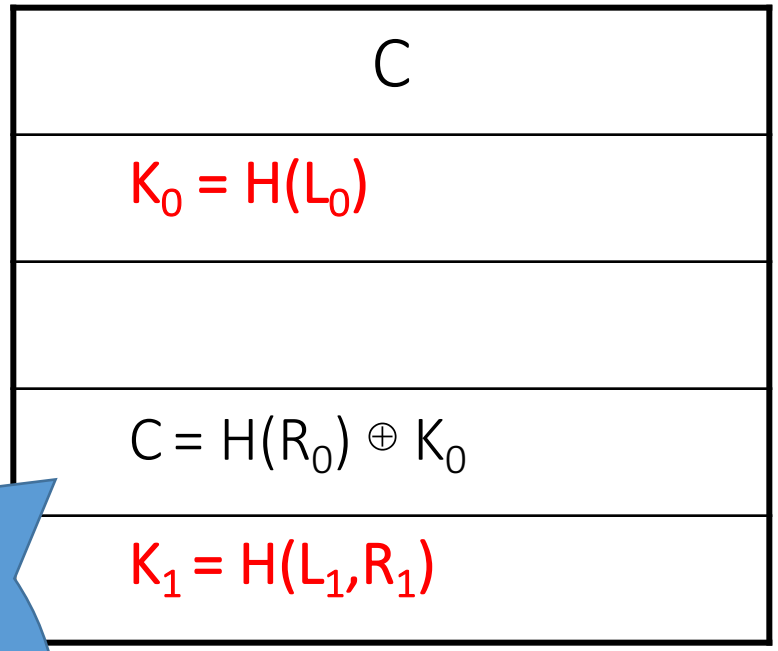
Privacy-Free Garbling

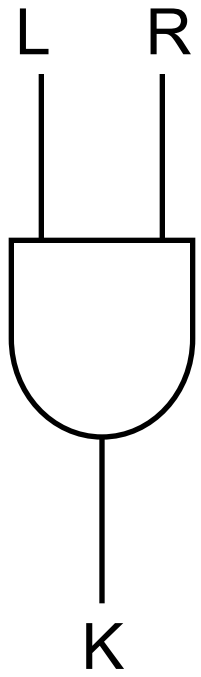
C
$C_1 = H(L_0) \oplus K_0$
$C_3 = H(R_0) \oplus K_0$
$C_4 = H(L_1,R_1) \oplus K_1$



Output is 0
If either input is 0

Privacy-Free Garbling





Standard "row-reduction" technique

Only 1 ciphertext!

Privacy-Free Evaluation

Eval(gg, L,a,R,b)

- If a=0
  - Output $K_0 = H(L_0)$
- If b=0
  - Output $K_0 = C \oplus H(R_0)$
- else
  - Output $K_1 = H(L_1, R_1)$

gg
$C = H(R_0) \oplus K_0$

Runtime (rough estimates)

- Proof of “c=AES(k,m)” for secret k and public (c,m)
- AES: 35k gates (7k ANDs/28k XORs)
- **Communication**: **204kB** (98% GC)
- **Runtime**:
  - **OT**: 29.4ms (Using Chou-Orlandi OT) (|w|=128)
  - **Garbling**: 721µs (Using JustGarble GaXR)
  - **Eval**: 273 µs
  - **Total** (Garble+OT+Eval+Garble) **~ 31.2ms** (+network)

Applications

Hu, Mohassel, Rosulek

- ***Sublinear ZK (via ORAM)***, Crypto 2015

Chase, Ganesh, Mohassel,

- **Privacy-Preserving Credentials**, Crypto 2016

Kolesnikov, Krawczyk, Lindell, Malozemoff, Rabin,

- **Attribute-Based KE with General Policies**, CCS 2016

Baum; Katz, Malozemoff, Wang,

- **Input validity in 2PC**, SCN 2016; eprint

…

ZKBoo: Faster Zero-Knowledge for Boolean Circuits

Giacomelli, Madsen, Orlandi

Usenix 2016

From ZKGC to ZKBoo

- ZKGC is inherently **interactive** (**private coin,** cannot use Fiat-Shamir)

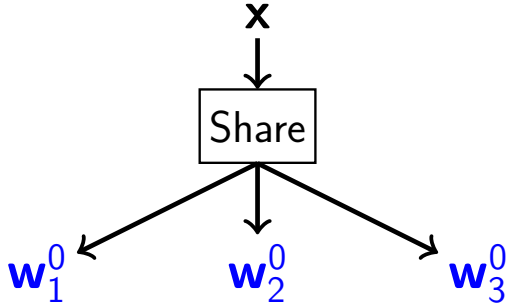
- **IKOS** (*Ishai, Kushilevitz, Ostrovsky, Sahai*) proposed in 2007 a method to get ZK from MPC. Plugging the right MPC protocol one can get ZK with very good ***asymptotic complexity.***

- ***ZKBoo*** can be seen as a generalization, simplification and implementation of IKOS with the sole goal of ***practical efficiency.***

To build ZKBoo, we need to find a suitable

$(2,3)$-**decomposition** for C:

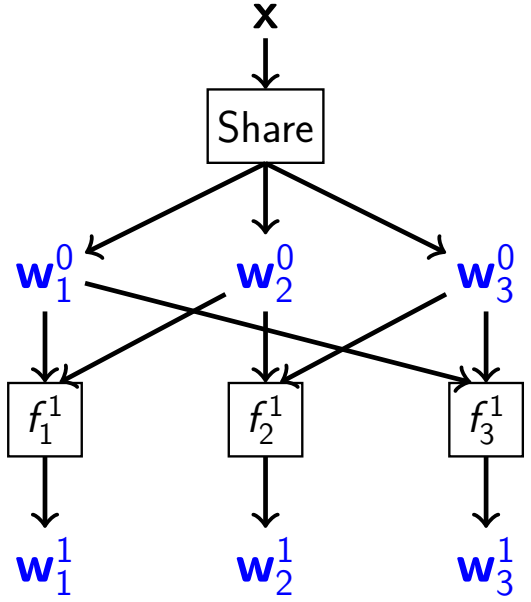
$$\{\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}\}$$
$$\cup$$
$$\{f_1^{(j)}, f_2^{(j)}, f_3^{(j)}\}_{j=1,\ldots,N}$$



To build ZKBoo, we need to find a suitable

$(2,3)$-**decomposition** for C:

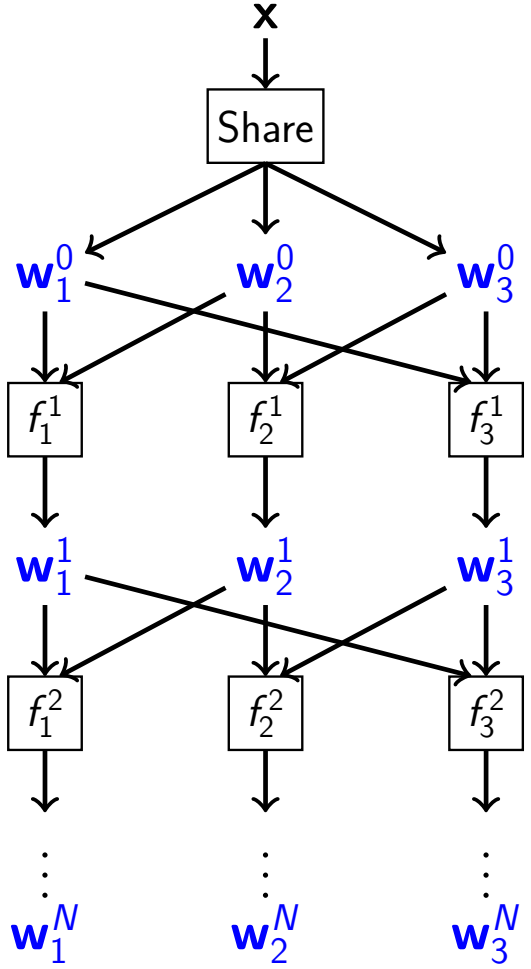
$$\{\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}\}$$
$$\cup$$
$$\{f_1^{(j)}, f_2^{(j)}, f_3^{(j)}\}_{j=1,\ldots,N}$$



To build ZKBoo, we need to find a suitable

$(2,3)$-**decomposition** for C:

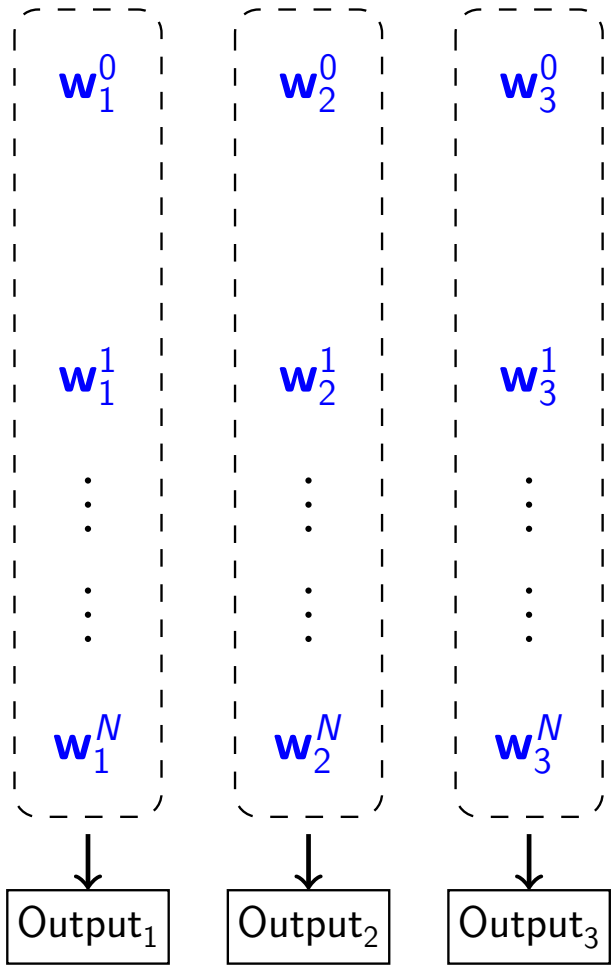
$$\{\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}\}$$
$$\cup$$
$$\{f_1^{(j)}, f_2^{(j)}, f_3^{(j)}\}_{j=1,\ldots,N}$$



To build ZKBoo, we need to find a suitable

$(2,3)$-**decomposition** for C:

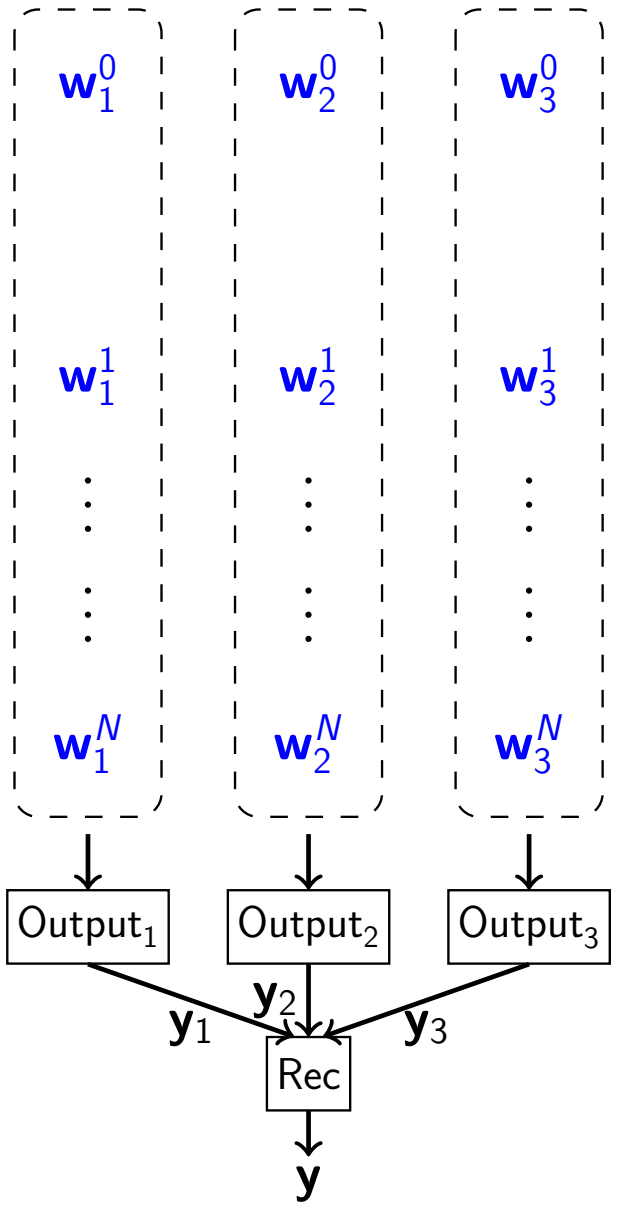
$$\{\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}\}$$
$$\cup$$
$$\{f_1^{(j)}, f_2^{(j)}, f_3^{(j)}\}_{j=1,\ldots,N}$$



To build ZKBoo, we need to find a suitable

$(2,3)$-**decomposition** for C:

$$\{\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}\}$$
$$\cup$$
$$\{f_1^{(j)}, f_2^{(j)}, f_3^{(j)}\}_{j=1,\ldots,N}$$



To build ZKBoo, we need to find a suitable

$(2,3)$-**decomposition** for C:

$$\{\text{Share}, \text{Output}_1, \text{Output}_2, \text{Output}_3, \text{Rec}\}$$
$$\cup$$
$$\{f_1^{(j)}, f_2^{(j)}, f_3^{(j)}\}_{j=1,\ldots,N}$$

- correct: $\mathbf{y} = C(\mathbf{x})$
- 2-private: $\forall\, e \in [3]$ $\exists$ a PPT simulator S_e that perfectly simulate the distribution of $(\{\mathbf{w}_i\}_{i\in\{e,e+1\}}, \mathbf{y}_{e+2})$

Example: the linear decomposition

- *Computation in a ring $(R,+,\cdot)$*
- *Share(x)*
  - *Get random $x_1, x_2 \leftarrow R$*
  - *Let $x_3 = x - x_1 - x_2$*
- *Rec(y_1,y_2,y_3)*
  - $y = y_1 + y_2 + y_3$
- *Add(x_1,x_2,x_3,y_1,y_2,y_3)*
  - $z_1 = x_1 + y_1$
  - $z_2 = x_2 + y_2$
  - $z_3 = z_3 + y_3$
- *Mul(x_1,x_2,x_3,y_1,y_2,y_3)*
  - $z_1 = x_1y_1 + x_1y_2 + x_2y_1 + r_1 - r_2$
  - $z_2 = x_2y_2 + x_2y_3 + x_3y_2 + r_2 - r_3$
  - $z_3 = x_3y_3 + x_3y_1 + x_1y_3 + r_3 - r_1$

...composition

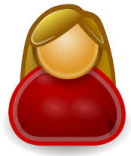
Correctness:
$z_1+z_2+z_3 =$
$(x_1+x_2+x_3)\ (y_1+y_2+y_3)$

- *Sha...*

2-privacy:
Any pair (z_i,z_{i+1}) is uniform random (thanks to r_1,r_2,r_3)

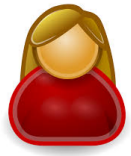
- *Add*$(x_1,x_2,x_3,y_1,y_2,y_3)$
  - $z_1 = x_1 + y_1$
  - $z_2 = x_2 + y_2$
  - $z_3 = z_3 + y_3$
- *Mul*$(x_1,x_2,x_3,y_1,y_2,y_3)$
  - $z_1 = x_1y_1 + x_1y_2 + x_2y_1 + r_1 - r_2$
  - $z_2 = x_2y_2 + x_2y_3 + x_3y_2 + r_2 - r_3$
  - $z_3 = x_3y_3 + x_3y_1 + x_1y_3 + r_3 - r_1$

Public data: $C : \{0,1\}^n \to \{0,1\}^m$ (boolean circuit) and $\mathbf{y} \in \{0,1\}^m$

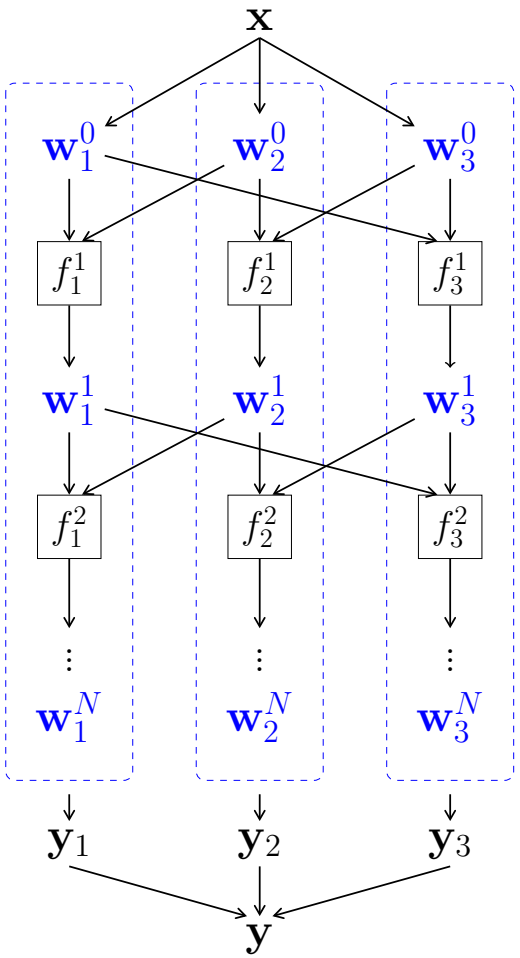


Input: $\mathbf{x}$ s.t. $C(\mathbf{x}) = \mathbf{y}$

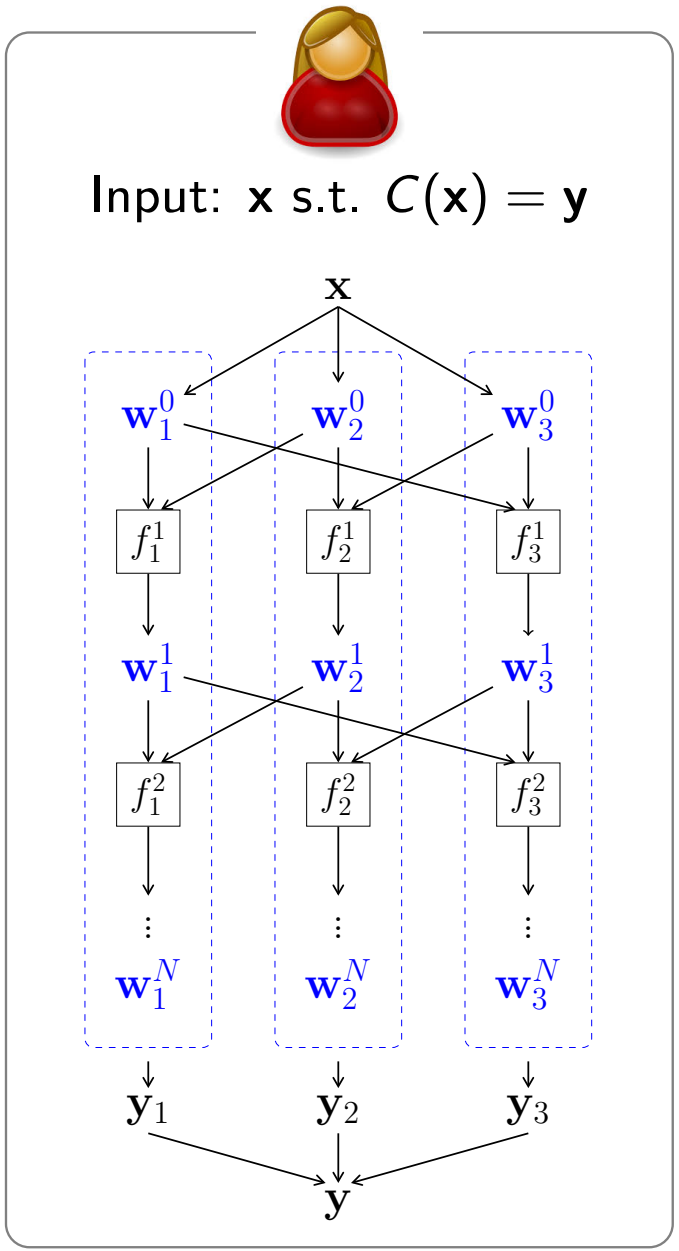
Public data: $C : \{0,1\}^n \to \{0,1\}^m$ (boolean circuit) and $\mathbf{y} \in \{0,1\}^m$



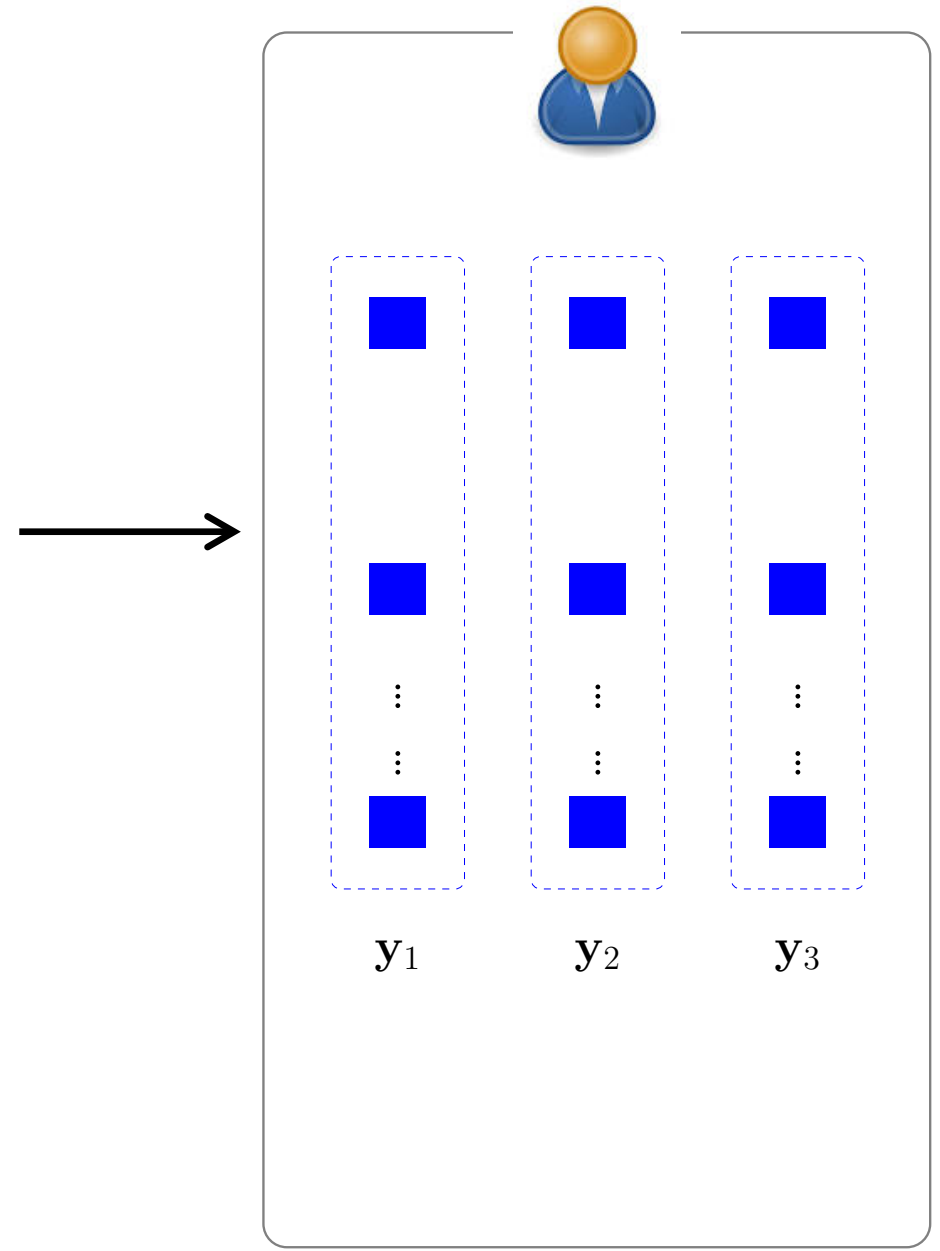
Input: $\mathbf{x}$ s.t. $C(\mathbf{x}) = \mathbf{y}$



Public data: $C : \{0,1\}^n \to \{0,1\}^m$ (boolean circuit) and $\mathbf{y} \in \{0,1\}^m$

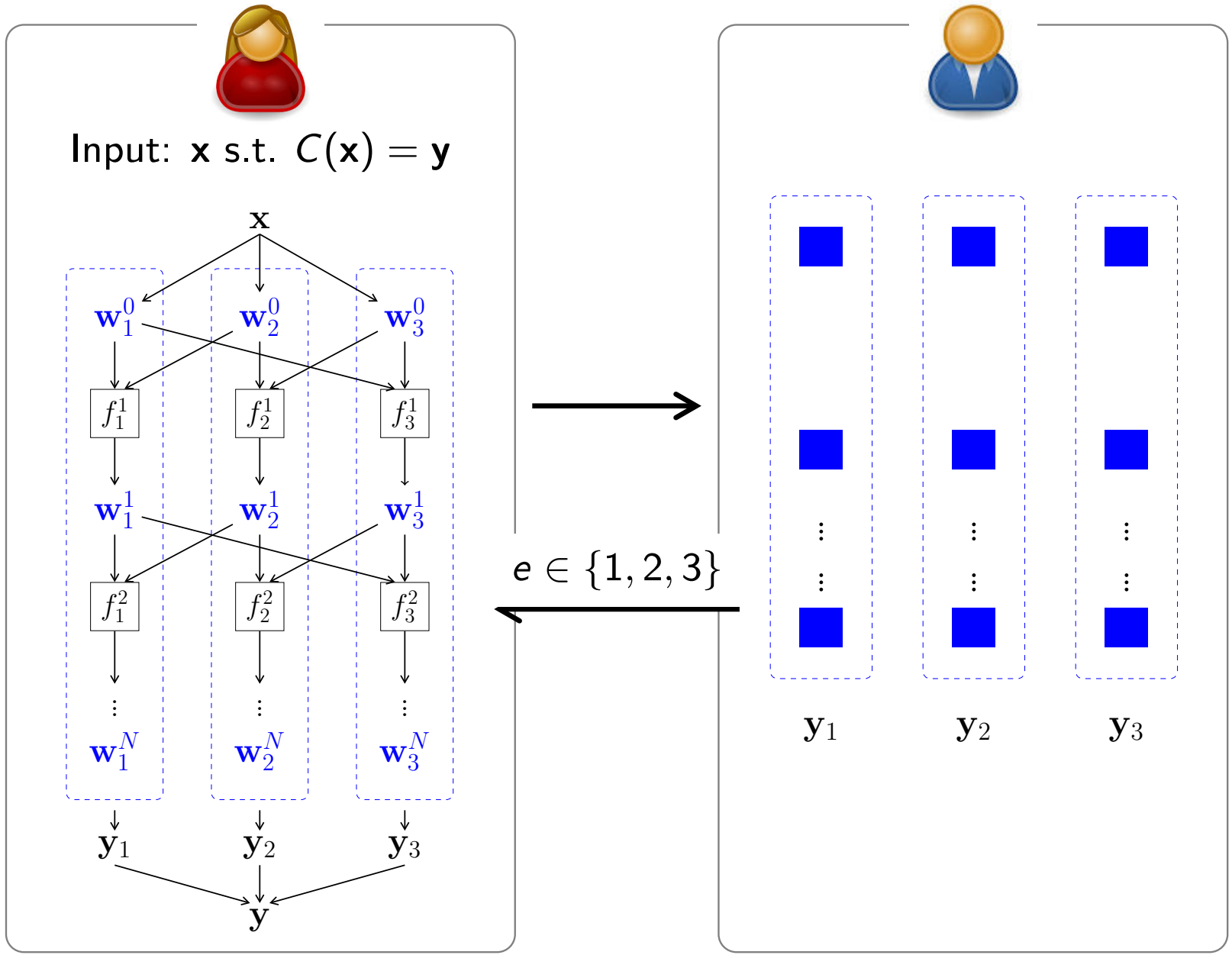


Input: $\mathbf{x}$ s.t. $C(\mathbf{x}) = \mathbf{y}$

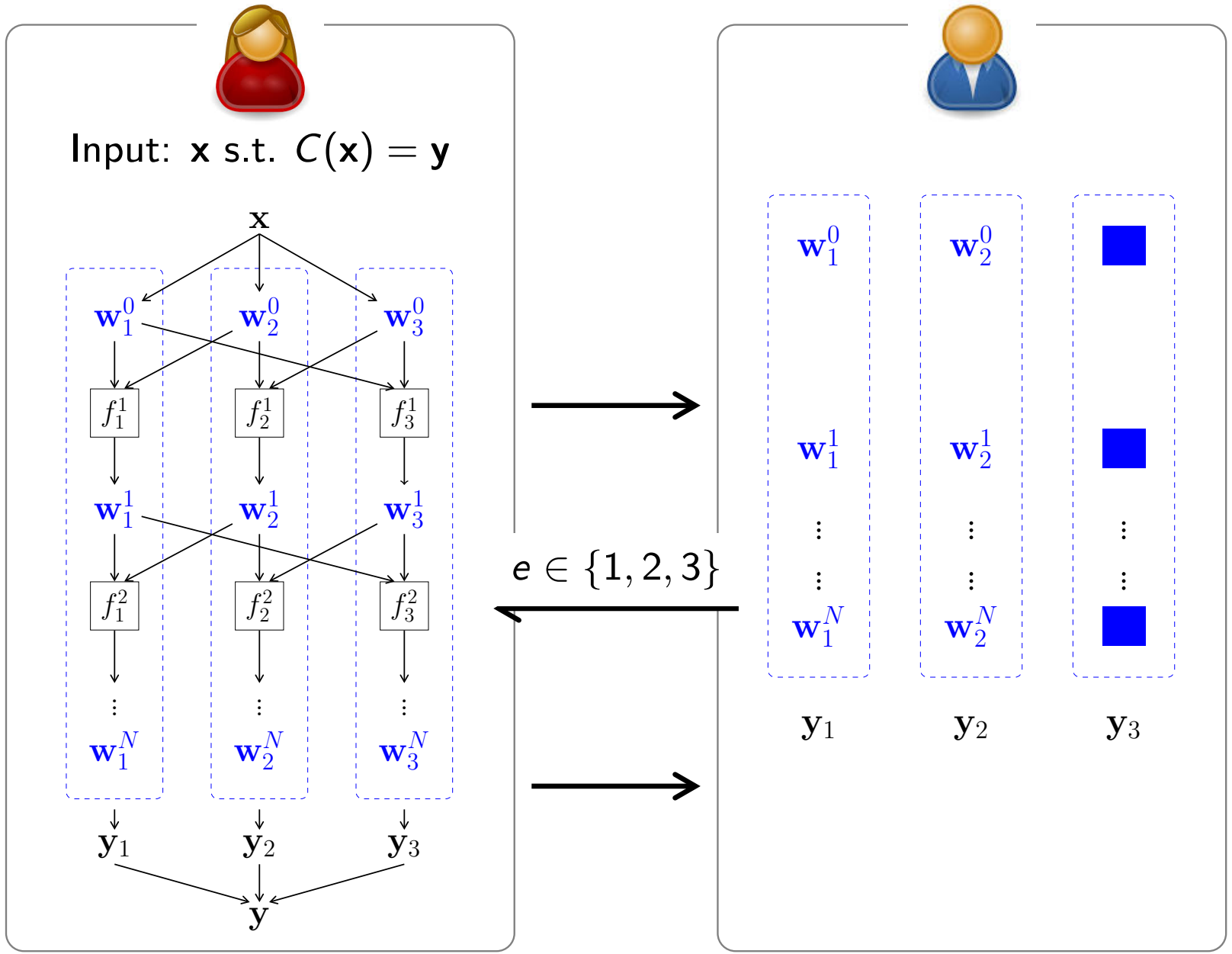


$\mathbf{y}_1$ $\mathbf{y}_2$ $\mathbf{y}_3$

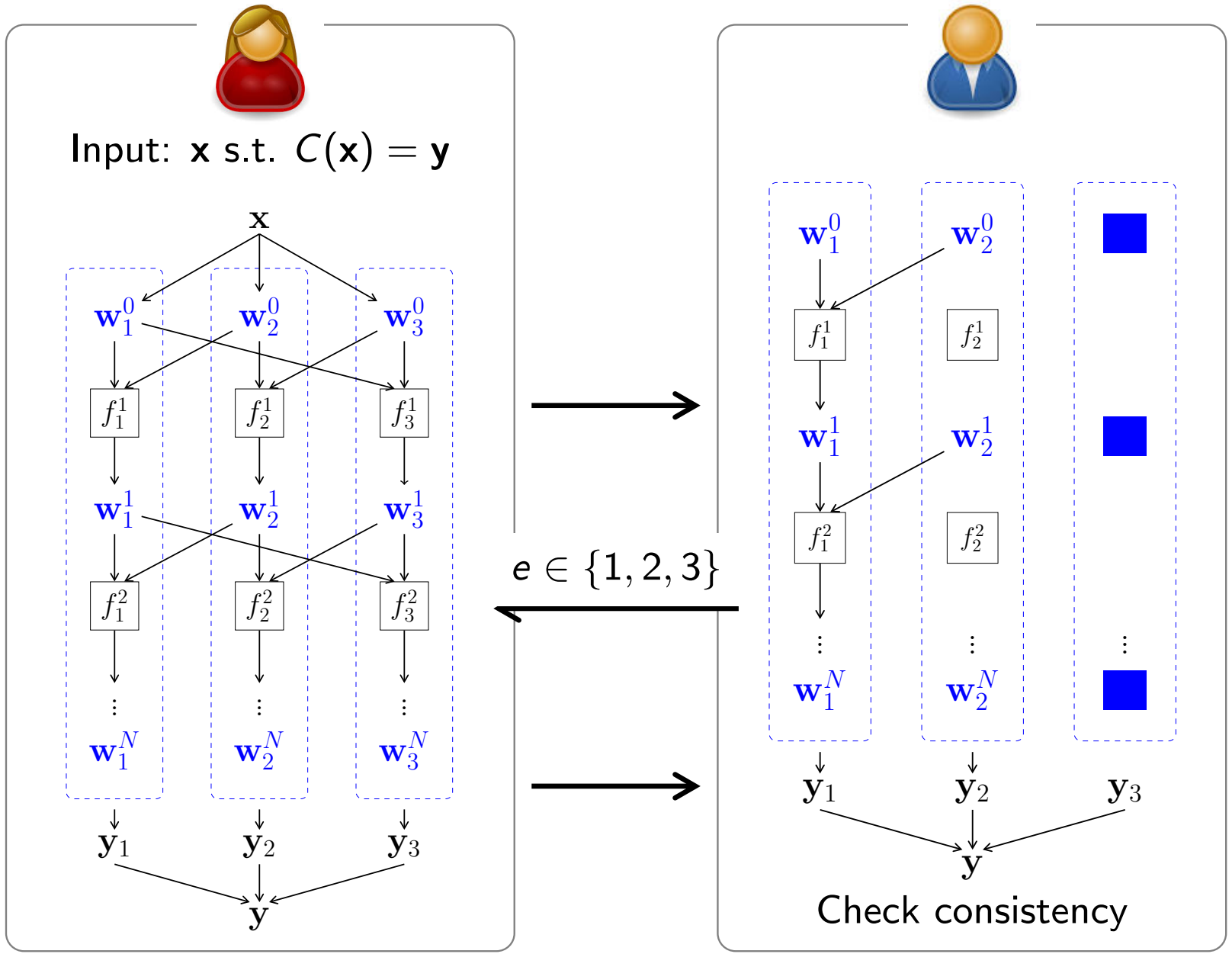
Public data: $C : \{0,1\}^n \to \{0,1\}^m$ (boolean circuit) and $\mathbf{y} \in \{0,1\}^m$



Public data: $C : \{0,1\}^n \to \{0,1\}^m$ (boolean circuit) and $\mathbf{y} \in \{0,1\}^m$



Public data: $C : \{0,1\}^n \to \{0,1\}^m$ (boolean circuit) and $\mathbf{y} \in \{0,1\}^m$



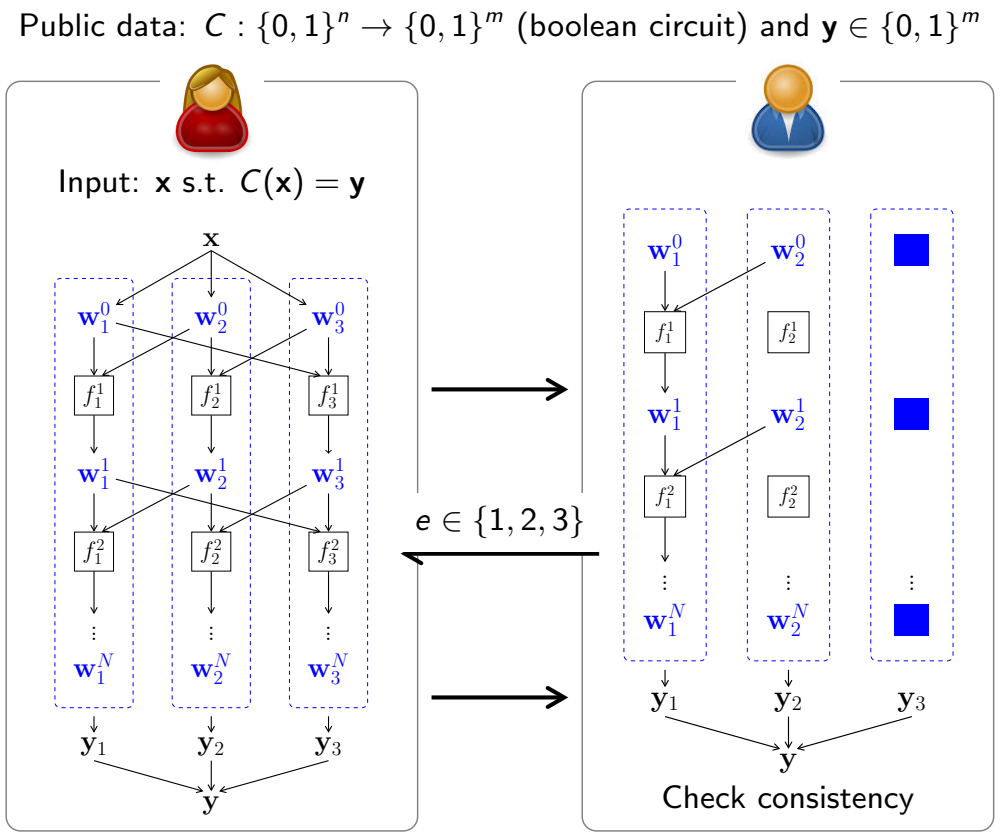
Linear Decomposition: Consistency Check

- Mul(x_1,x_2,x_3,y_1,y_2,y_3,r_1,r_2,r_3)
  - $z_1 = x_1y_1 + x_1y_2 + x_2y_1 + r_1 - r_2$
  - $z_2 = x_2y_2 + x_2y_3 + x_3y_2 + r_2 - r_3$
  - $z_3 = x_3y_3 + x_3y_1 + x_1y_3 + r_3 - r_1$

- Verify(. ,x_2,x_3,. ,y_2,y_3,. ,r_2,r_3)
  - ?
  - $z_2 = x_2y_2 + x_2y_3 + x_3y_2 + r_2 - r_3$
  - ?

ZKBoo

- **Soundness/PoK**
  - Correctness of decomposition
  - Commitments are binding
- **Zero-Knowledge**
  - 2-privacy of decomposition
  - Commitments are hiding
- **Efficiency**
  - Comm. and comp. complexity ~ # mul
  - Only very efficient crypto involved (secret sharing, commitments)



	SHA-1		SHA-256	
	Serial	Paral.	Serial	Paral.
Prover (ms)	31.73	12.73	54.63	15.95
Verifier (ms)	22.85	4.39	67.74	13.20
Proof size (KB)	444.18		835.91	

Soundness error: 2^{-80}

	SHA-1		SHA-256	
	Serial	Paral.	Serial	Paral.
Prover (ms)	18.98	8.12	30.81	12.45
Verifier (ms)	11.68	2.35	34.16	6.77
Proof size (KB)	223.71		421.01	

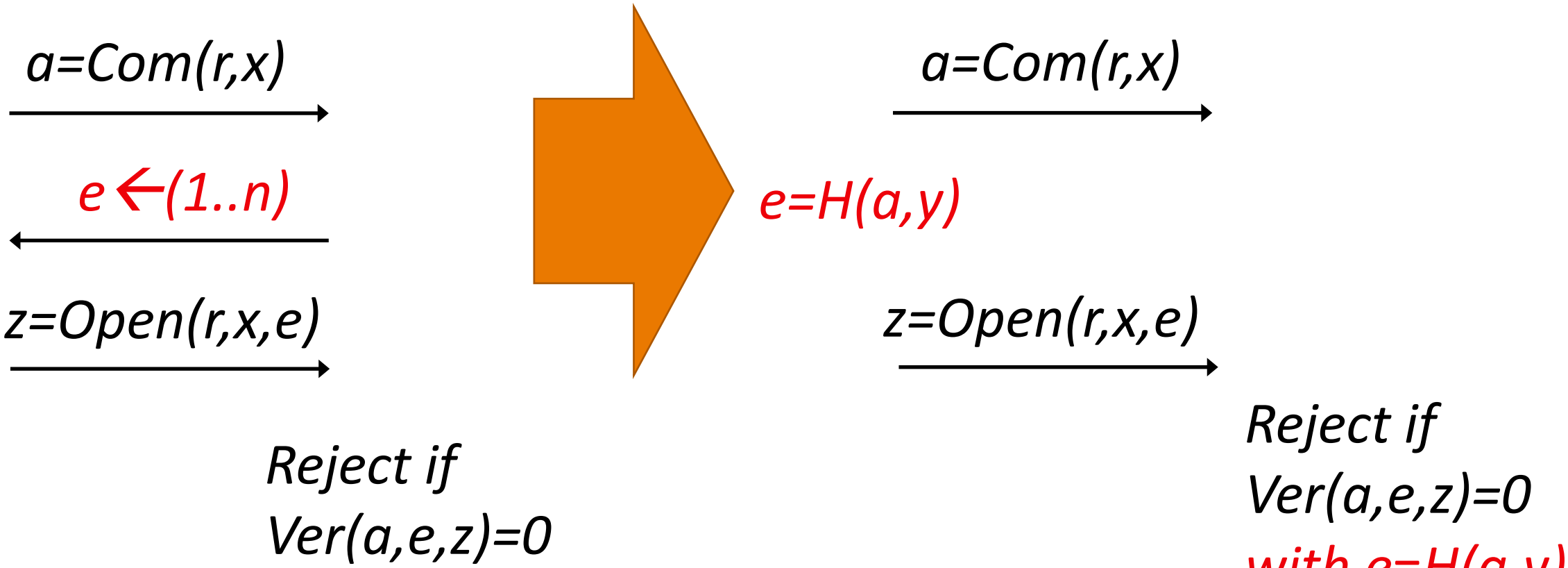
Soundness error: 2^{-40}

Digital Signatures from Symmetric-Key Primitives

Derler, Orlandi, Ramacher, Rechberger, Slamanig
ePrint 2016/1085

Fiat-Shamir Heuristic

P(x) "I know x s.t. f(x)=y" V(y)



Signatures from Fiat-Shamir

Gen

- *sk* : x
- *vk* : y = OWF(x)

Sig(sk,m)

- a = *Com*(r,x)
- z = *Open*(r,x,H(m,a,y))
- output (a,z)

Ver(vk,m,(a,z))

- reject if:
  Ver(a,H(m,a,y),z)=0

Signatures from ZKBoo + LowMC

LowMC

Block cipher with low AND Complexity

Different instances give different tradeoffs between comp./comm. overheadh

Scheme	Gen [ms]	Sign [ms]	Verify [ms]	|sk| [bytes]	|pk| [bytes]	|σ| [bytes]	Tight	M	PQ
MQ 5pass	0.96	7.21	5.17	32	74	40952	×	ROM	✓
Fish-128-128-12-26	0.01	9.46	6.25	16	16	86747	×	ROM	×
Fish-256-256-20-31	0.01	21.55	13.98	32	32	151368	×	ROM	✓
Begol-256-256-20-31	0.01	43.69	28.85	64	32	302697	✓	ROM	✓
SPHINCS-256	0.82	13.44	0.58	1088	1056	41000	✓	SM	✓
BLISS-I	44.16	0.12	0.02	2048	7168	5732	✓	ROM	✓
Ring-TESLA	16528.50	0.06	0.03	12288	8192	1568	✓	ROM	✓
TESLA-768	48570.01	0.65	0.36	3293216	4227072	2336	✓	(Q)ROM	✓

Table 1. Timings and sizes of private keys (sk), public keys (pk) and signatures (σ).

Candidate for PQ signature from symmetric primitive only!

Conclusions and directions

Thanks!

After >30 years of ZK ***we have the first truly efficient protocols*** for generic statements.

Many applications ***are enabled*** by efficient ZK for arbitrary circuits.

And I expect many more to come!

ZKGC vs ZKBoo?

- ZKBoo allows Fiat-Shamir ☺
- ZKBoo does not need OT ☺

The end of ZKGC?

- Are there better privacy-free GCs?

Improving ZKBoo?

- Large proof size: Can you find better decompositions?