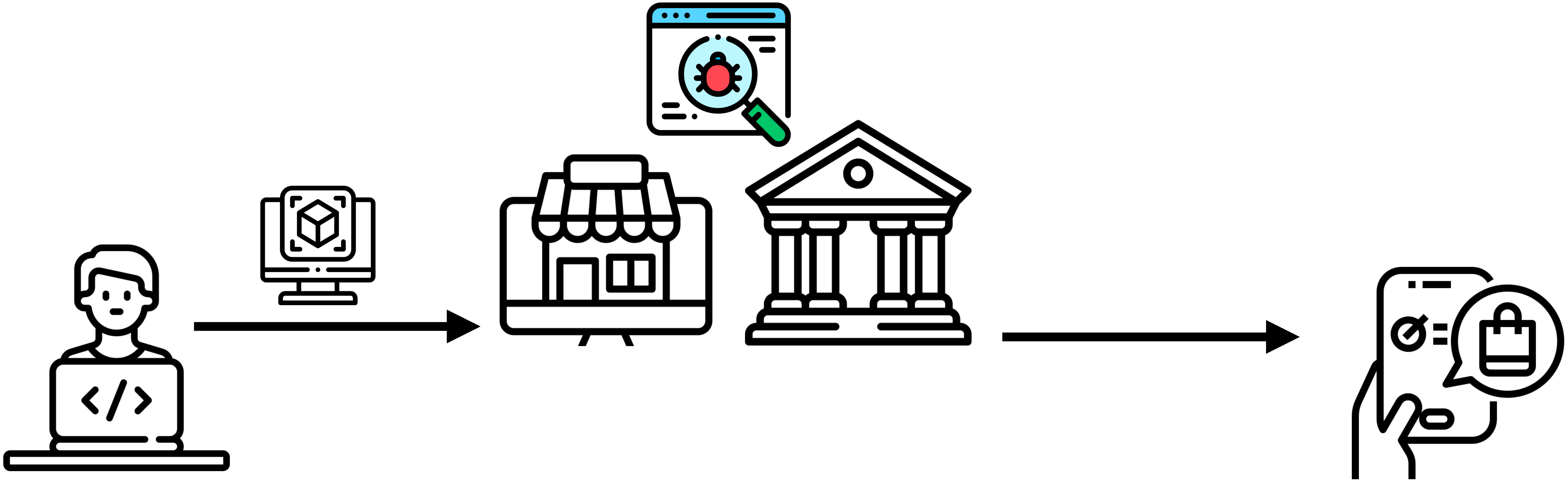
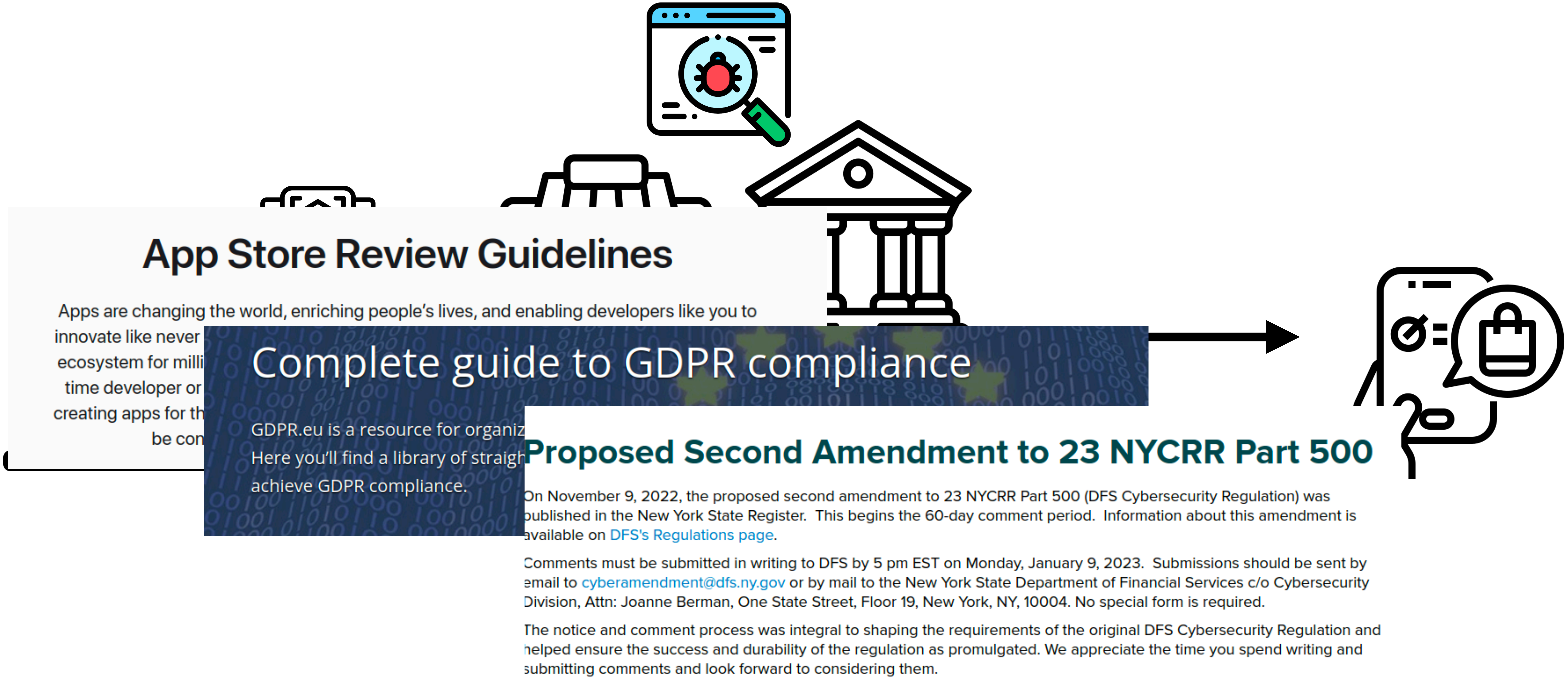
ZKUNSAT
Proving UNSAT in Zero Knowledge

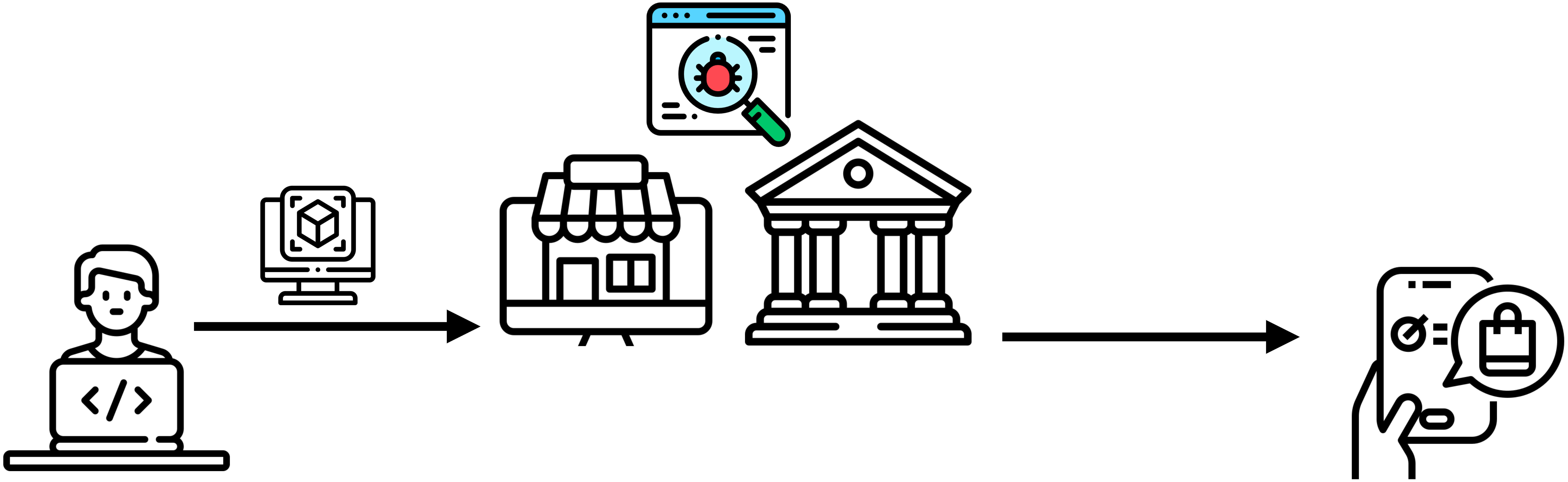
SAT Reunion Workshop, Simons Institute, UC Berkeley

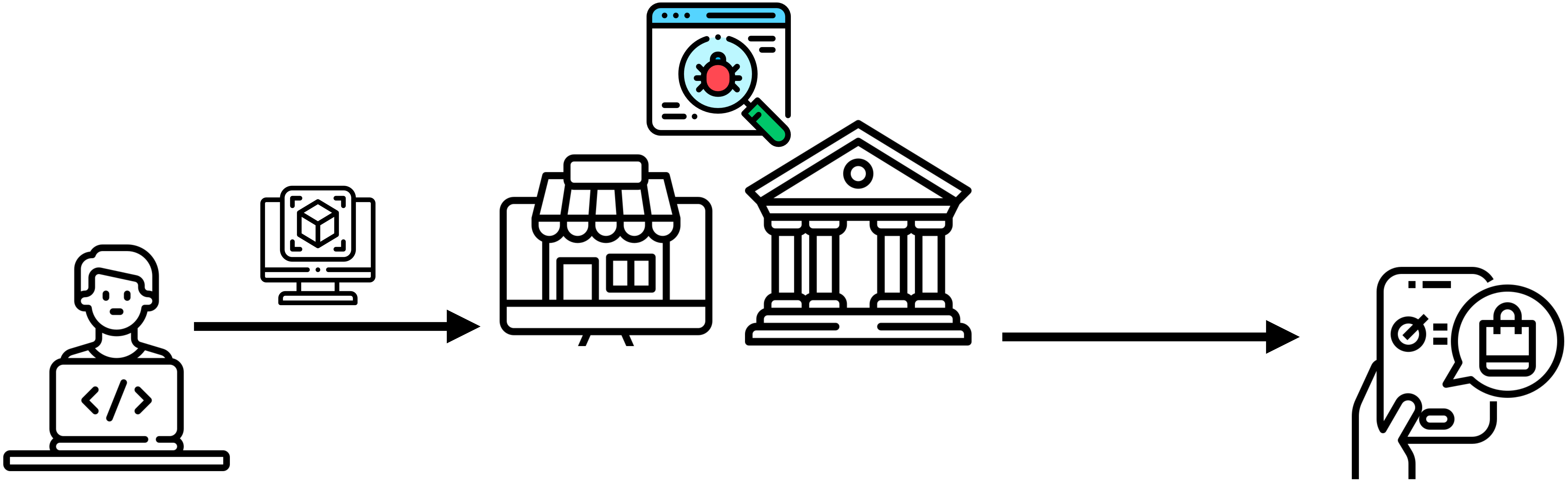
Ning Luo, Timos Antonopoulos, Bill Harris, Ruzica Piskac, Eran Tromer, Xiao Wang

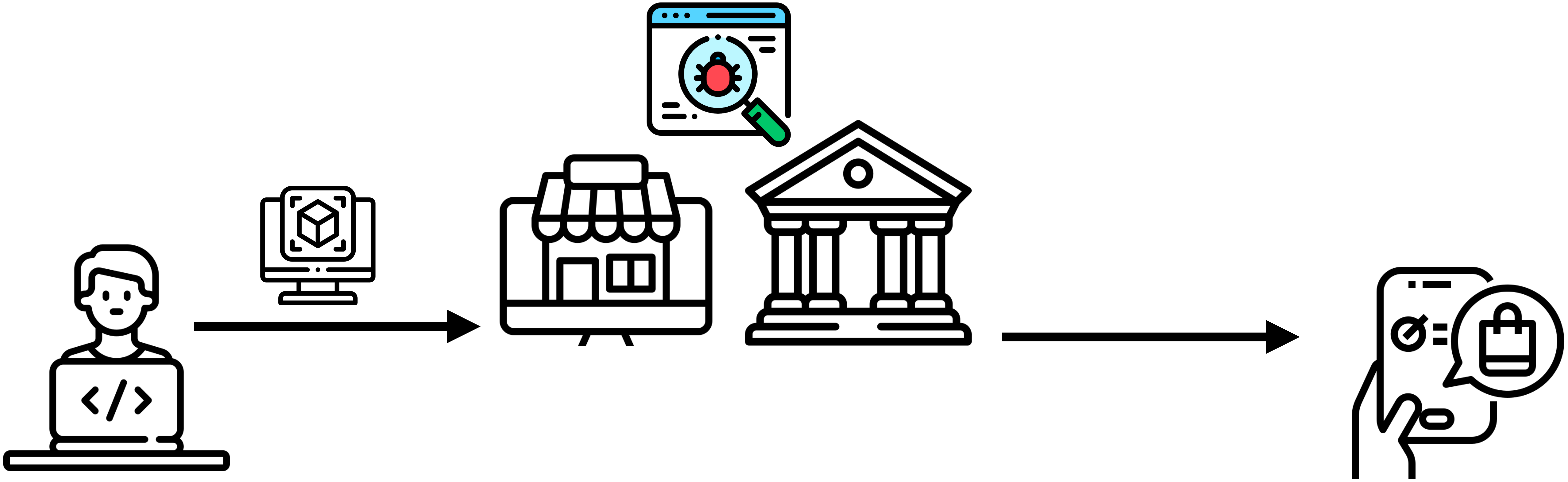




Verifications can be required by outside entities

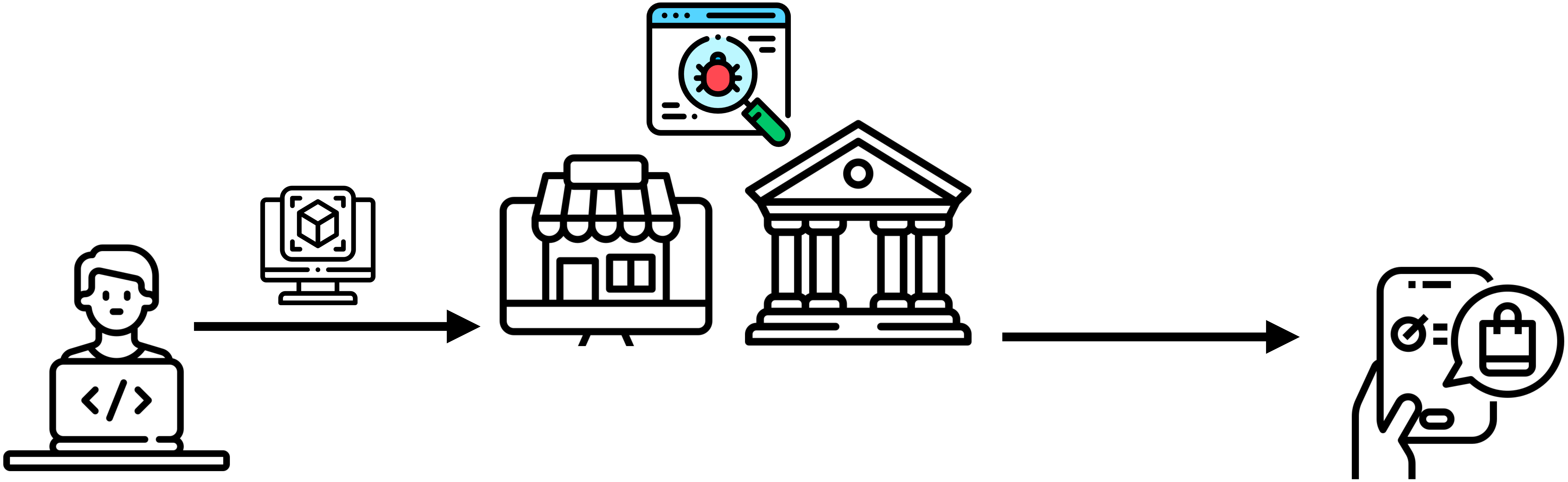


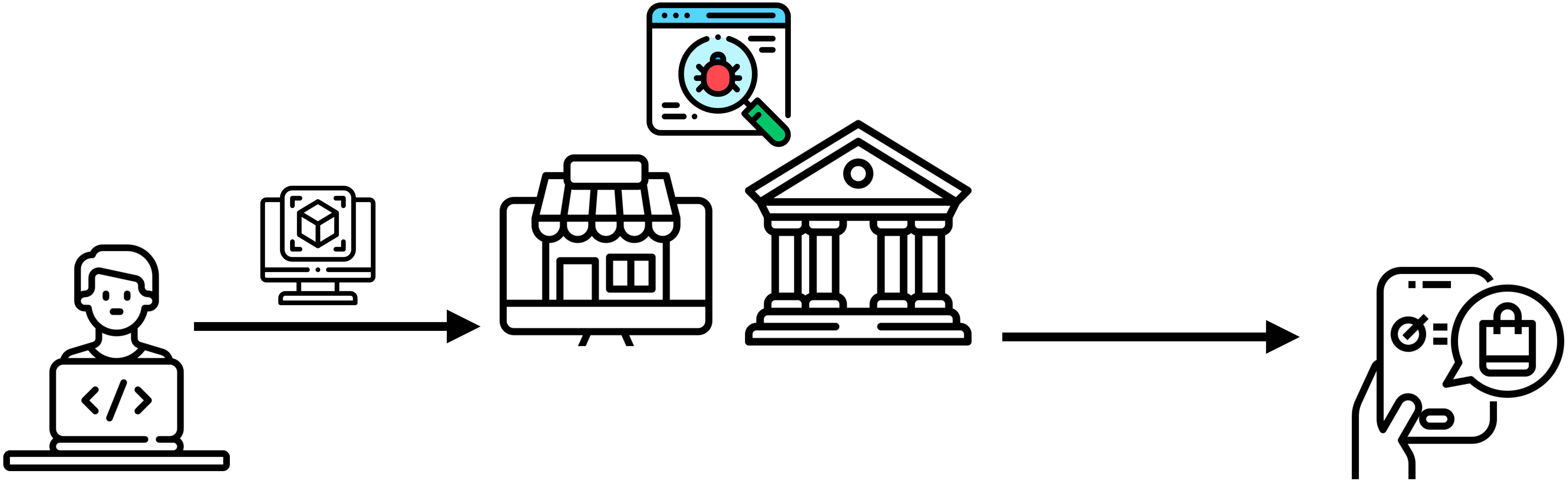




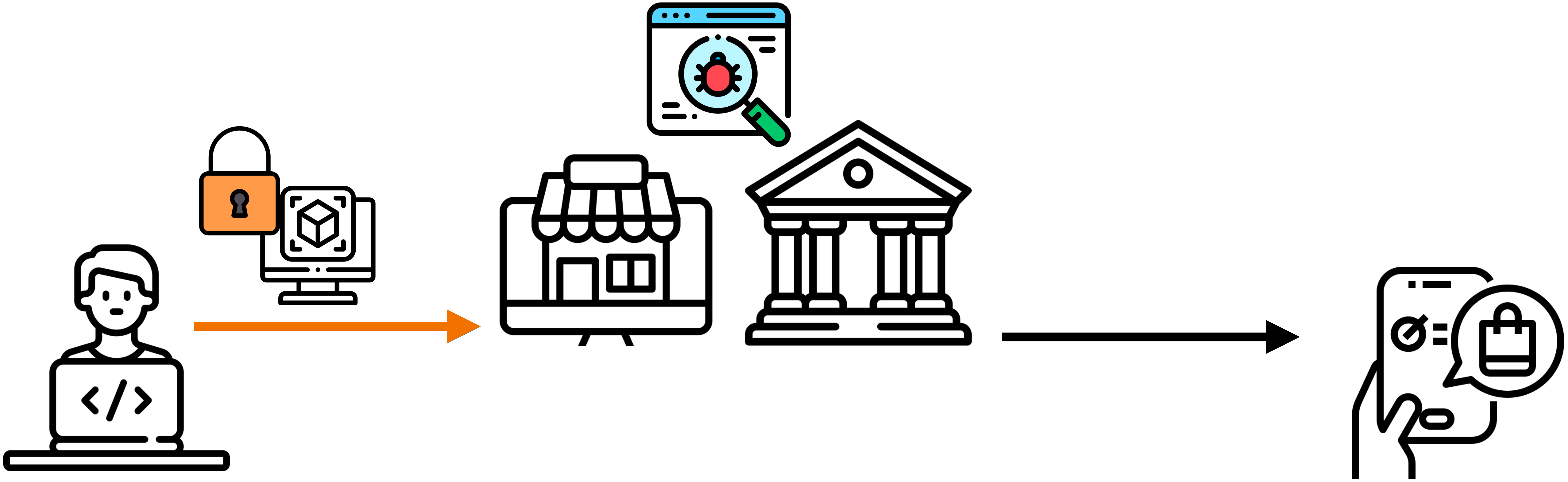
Centralized softwares verification and distribution：

Solution to a lot of problems, eg.supply-chain attacks

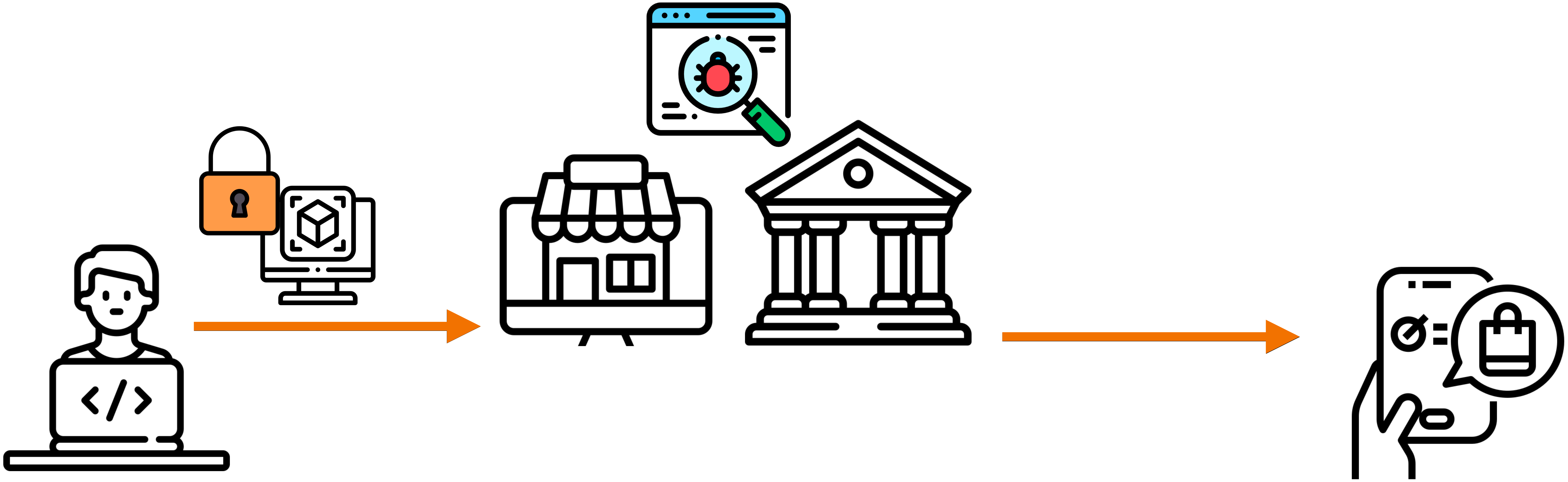




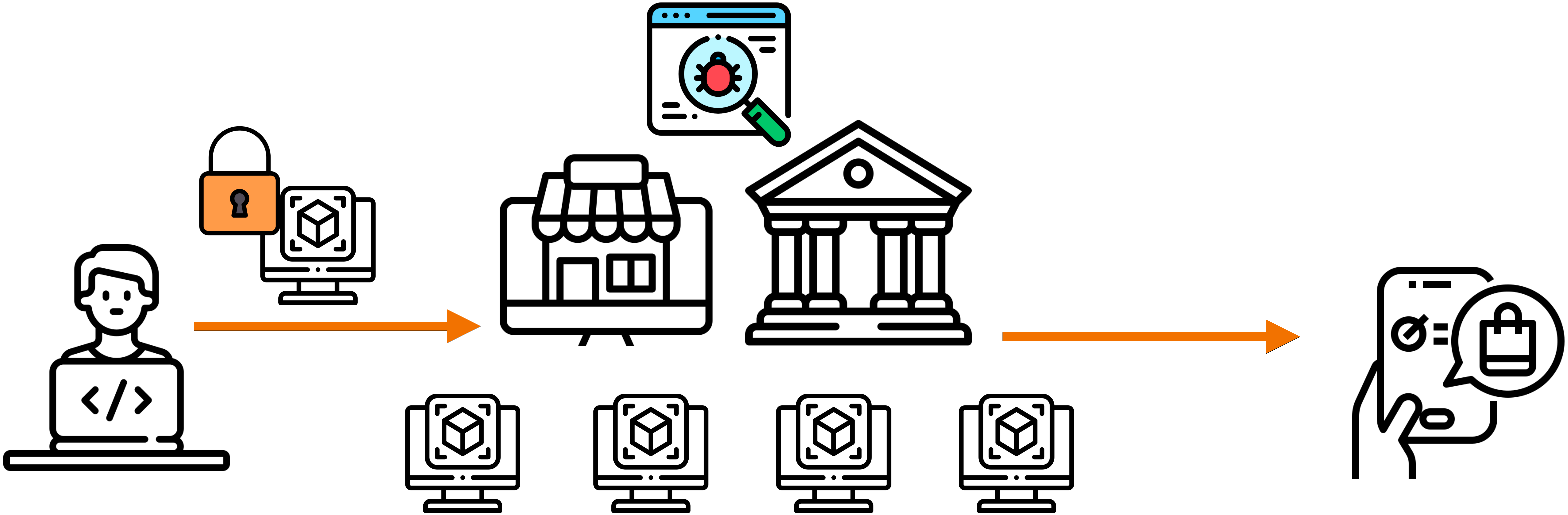
Both the developers and users should trust the centralized marketplace

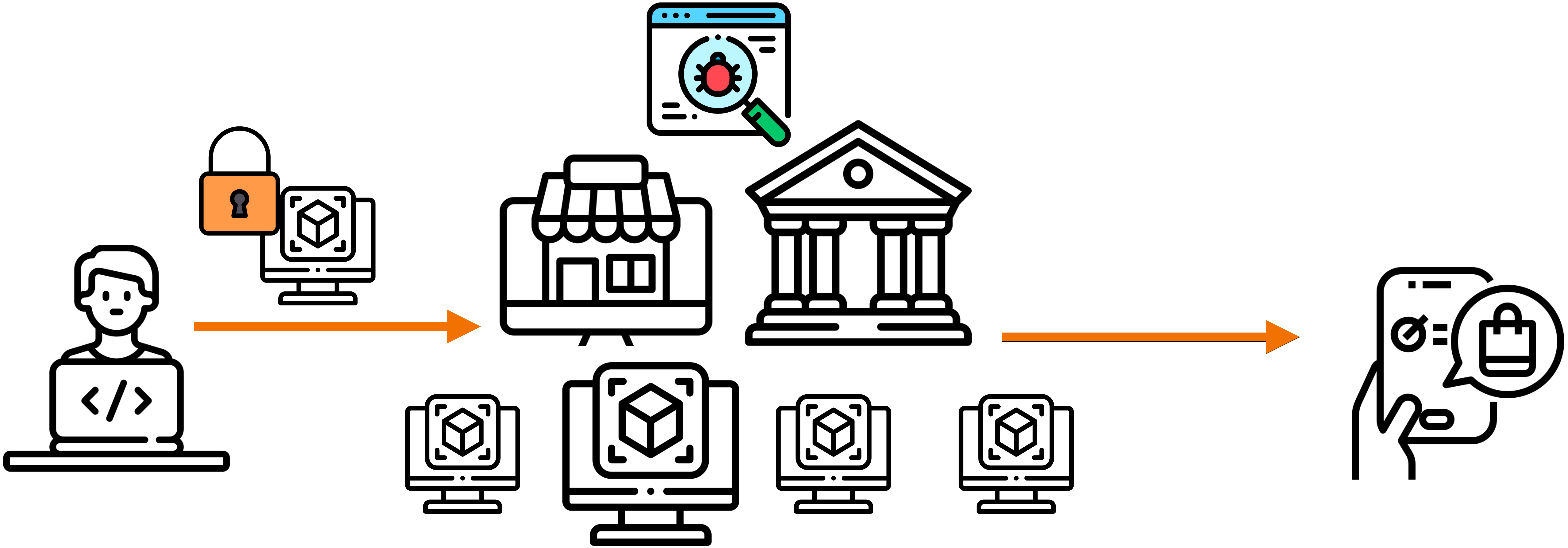


Both the developers and users should trust the centralized marketplace



Both the developers and users should trust the centralized marketplace





08.11.2021

Blumenthal, Blackburn & Klobuchar Introduce Bipartisan Antitrust Legislation to Promote App Store Competition

Apple Gives Ground in a Strategic Retreat From Strict App Store Rules

The company, under pressure from app developers and regulators, is making concessions while protecting lucrative parts of its App Store.

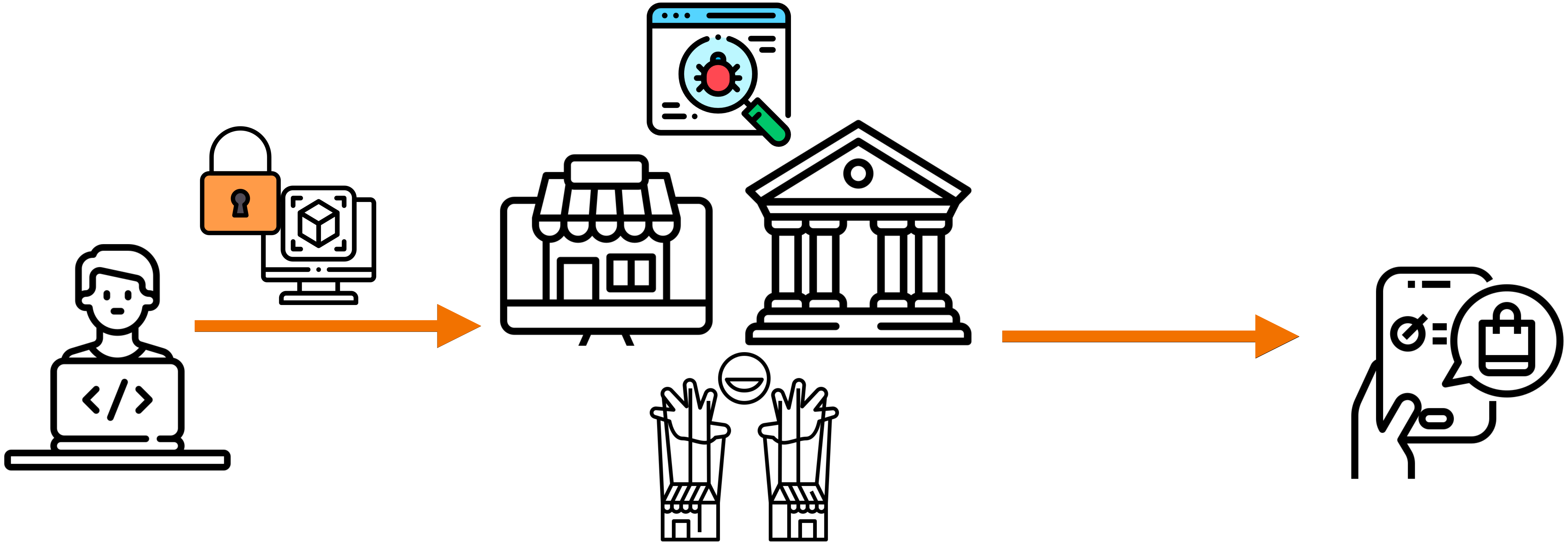
08.11.2021

Blumenthal, Blackburn & Klobuchar Introduce Bipartisan Antitrust Legislation to Promote App Store Competition

Apple Gives Ground in a Strategic Retreat From Strict App Store Rules

The company, under pressure from app developers and regulators, is making concessions while protecting lucrative parts of its App Store.

Centralized curation naturally produces a monopoly



THE SEPARATION OF PLATFORMS AND COMMERCE

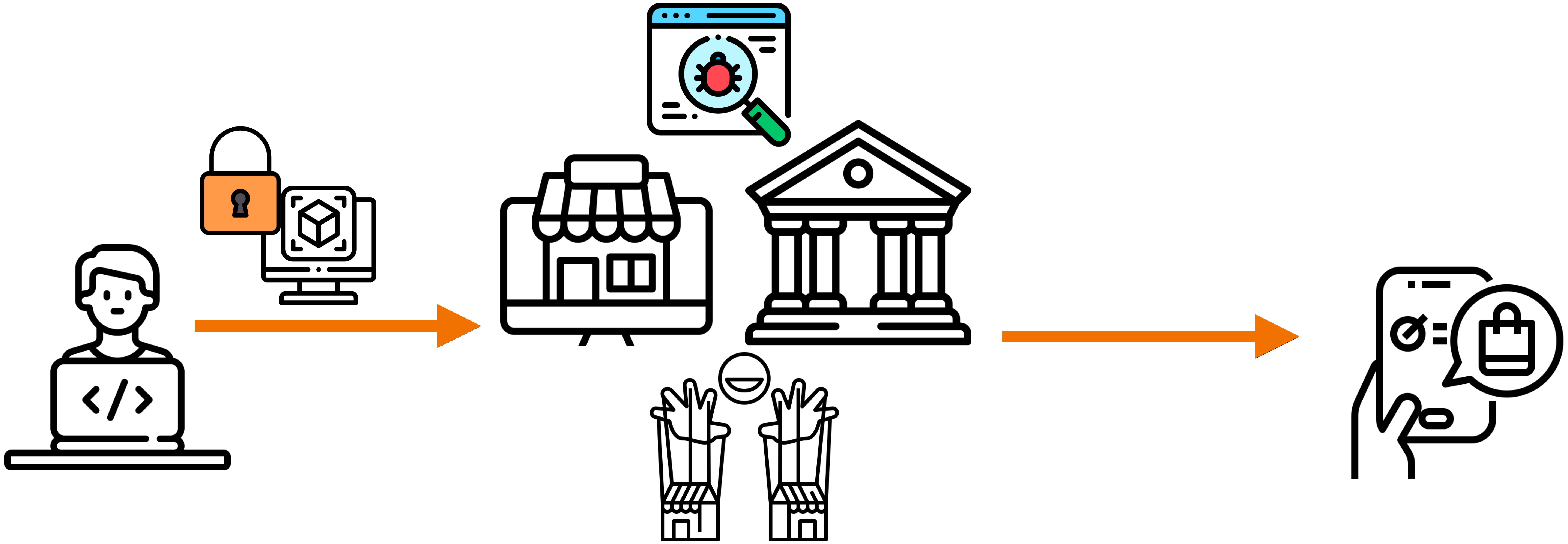
*Lina M. Khan**

A handful of digital platforms mediate a growing share of online commerce and communications. By structuring access to markets, these firms function as gatekeepers for billions of dollars in economic activity. One feature dominant digital platforms share is that they have integrated across business lines such that they both operate a platform and market their own goods and services on it. This structure places dominant platforms in direct competition with some of the businesses that depend on them, creating a conflict of interest that platforms can exploit to further entrench their dominance, thwart competition, and stifle innovation.

Read the Antitrust Lawsuit Against Google

Dozens of States Sue Google Over App Store Fees

Software developers have accused the company of harsh policies and taking a large cut of financial transactions in their apps.



THE SEPARATION OF PLATFORMS AND COMMERCE

*Lina M. Khan**

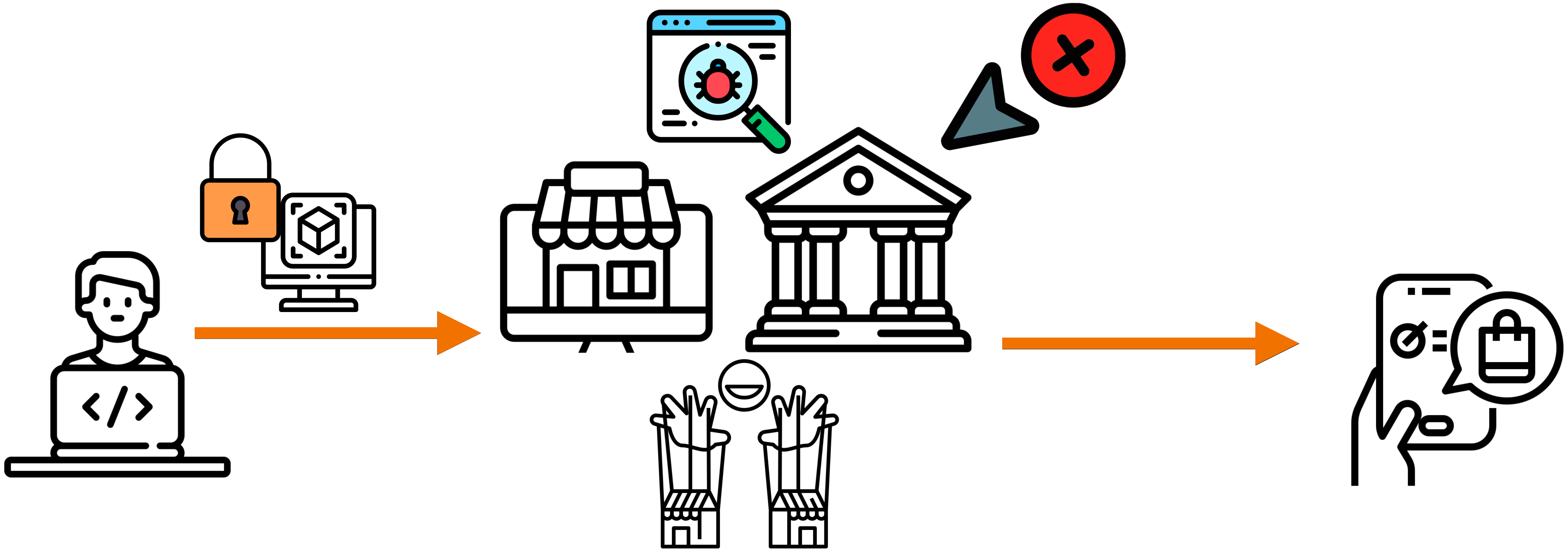
A handful of digital platforms mediate a growing share of online commerce and communications. By structuring access to markets, these firms function as gatekeepers for billions of dollars in economic activity. One feature dominant digital platforms share is that they have integrated across business lines such that they both operate a platform and market their own goods and services on it. This structure places dominant platforms in direct competition with some of the businesses that depend on them, creating a conflict of interest that platforms can exploit to further entrench their dominance, thwart competition, and stifle innovation.

Read the Antitrust Lawsuit Against Google

Dozens of States Sue Google Over App Store Fees

Software developers have accused the company of harsh policies and taking a large cut of financial transactions in their apps.

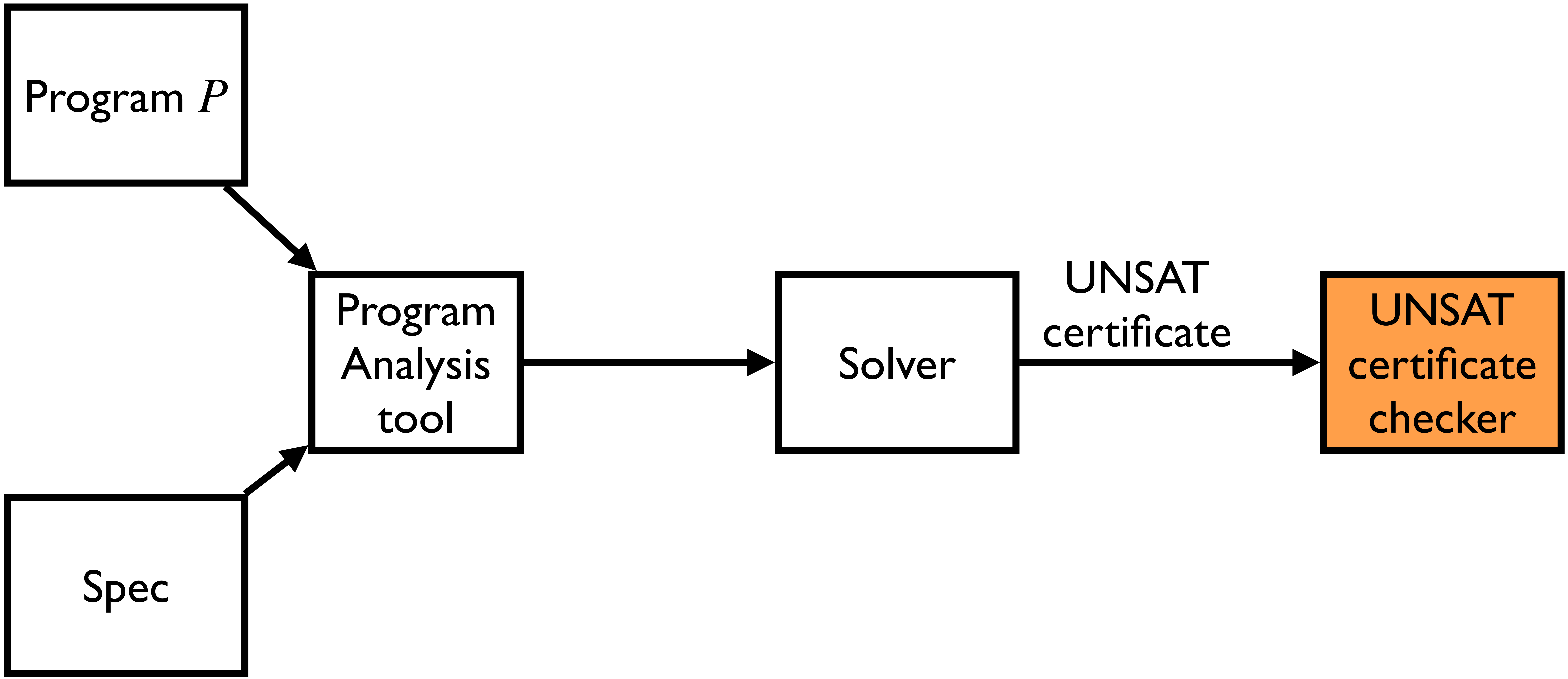
Remove centralized verification and distribution



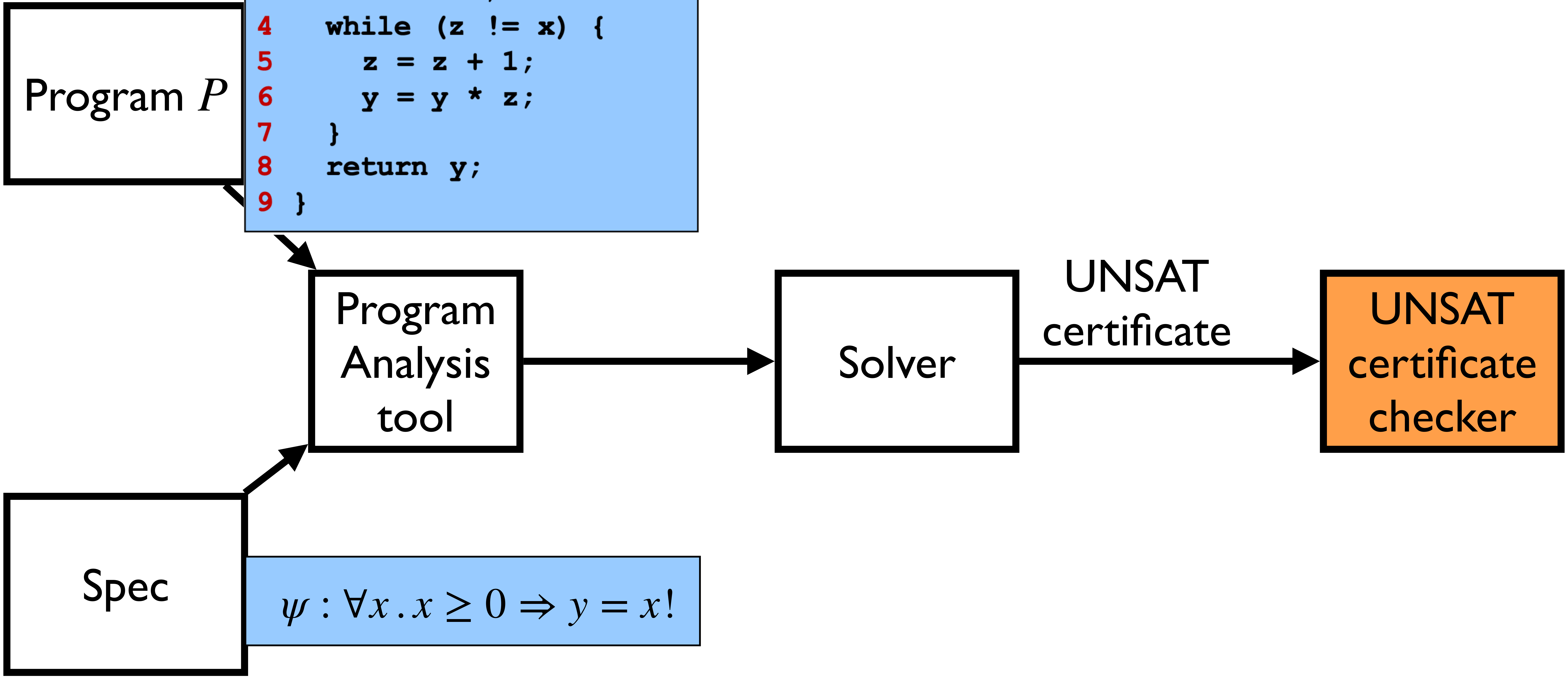

PPFM
Privacy-Preserving
Formal Methods

ZKUNSAT: The First Step towards PPFM

ZKUNSAT: The First Step towards PPFM



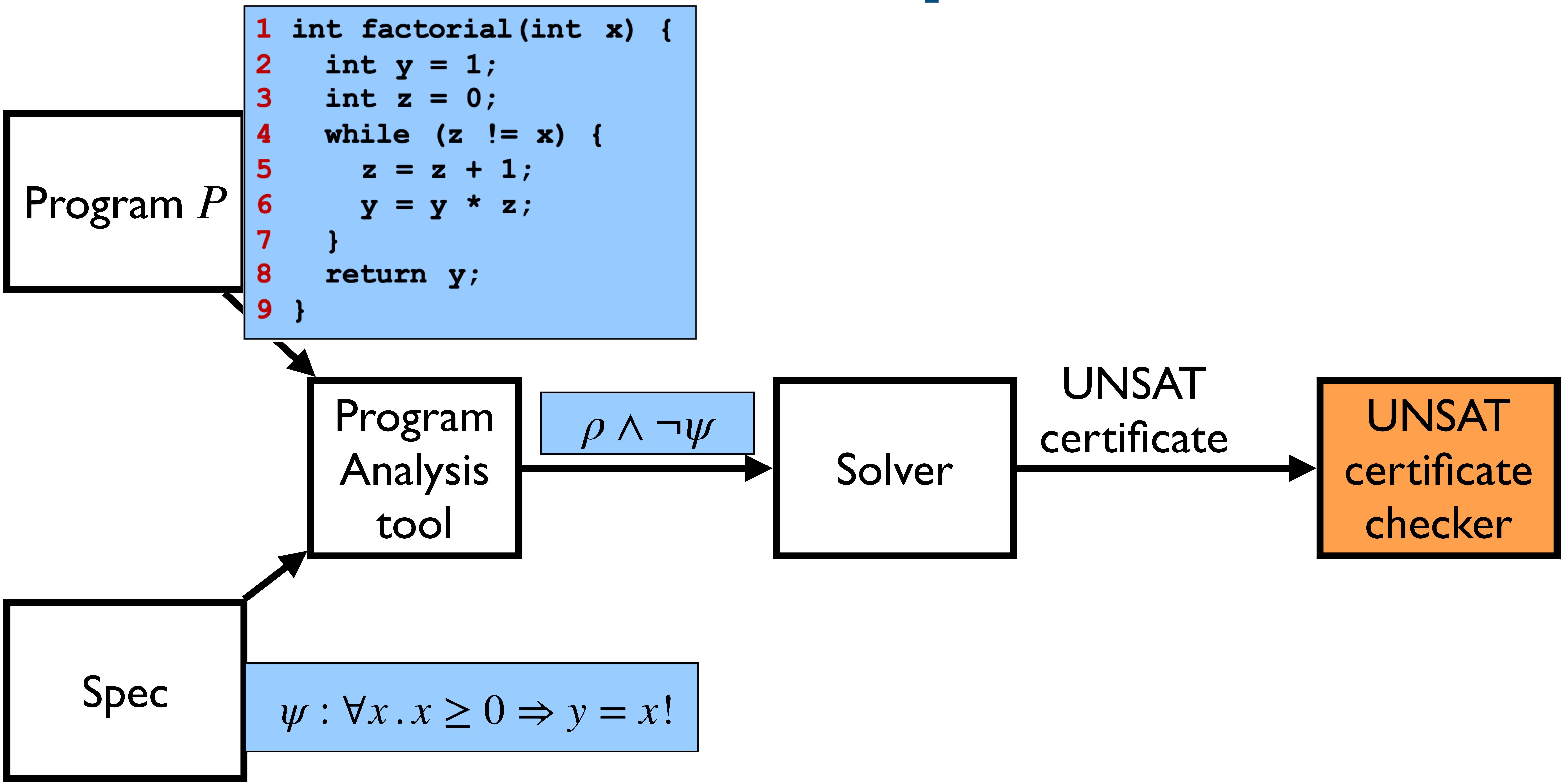
ZKUNSAT: The First Step towards PPFM



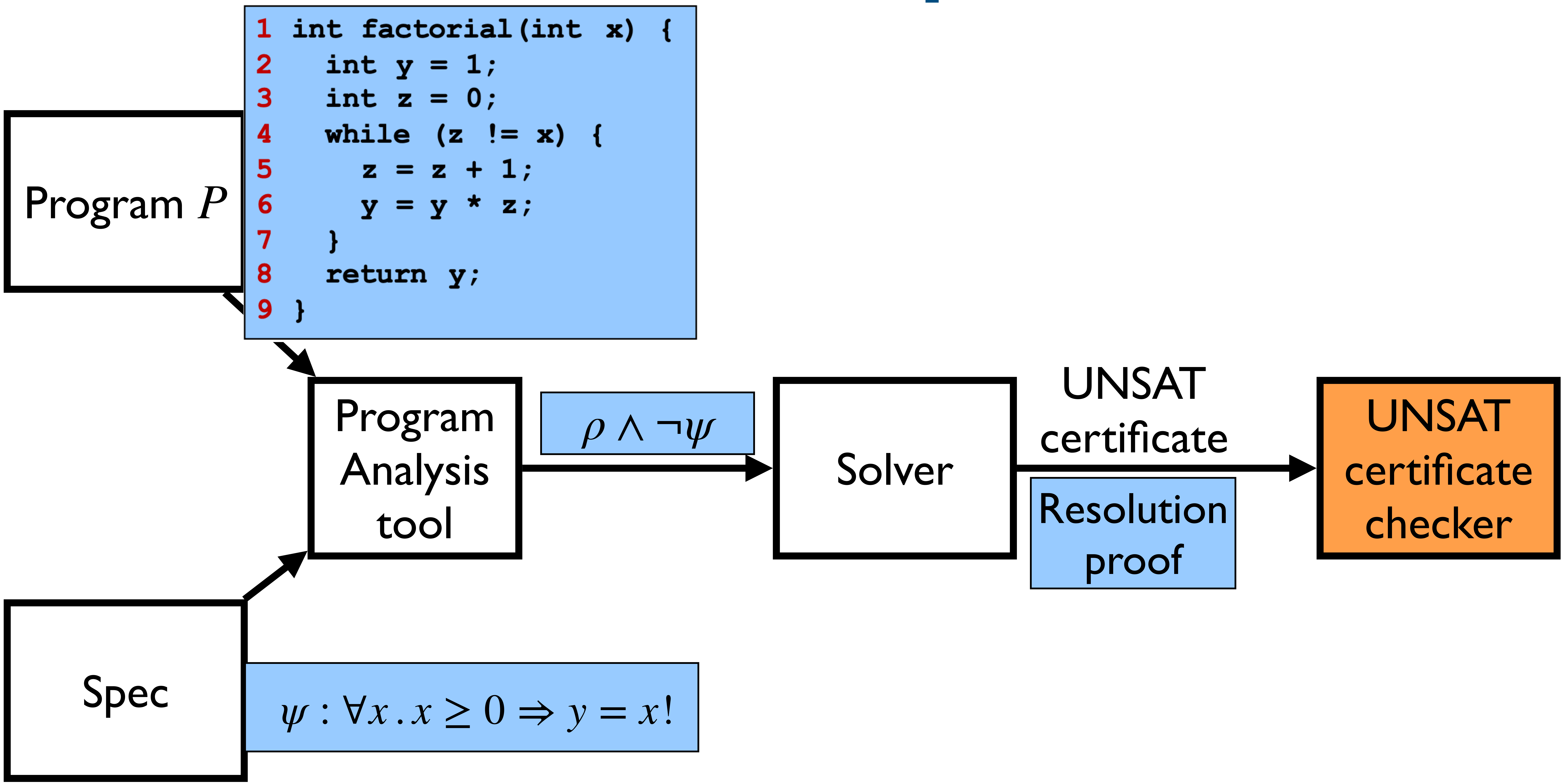

```
int factorial(int x) {
  int y = 1;
  int z = 0;
  while (z != x) {
    z = z + 1;
    y = y * z;
  }
  return y;
}
```

$$\psi : \forall x \,.\, x \geq 0 \Rightarrow y = x!$$

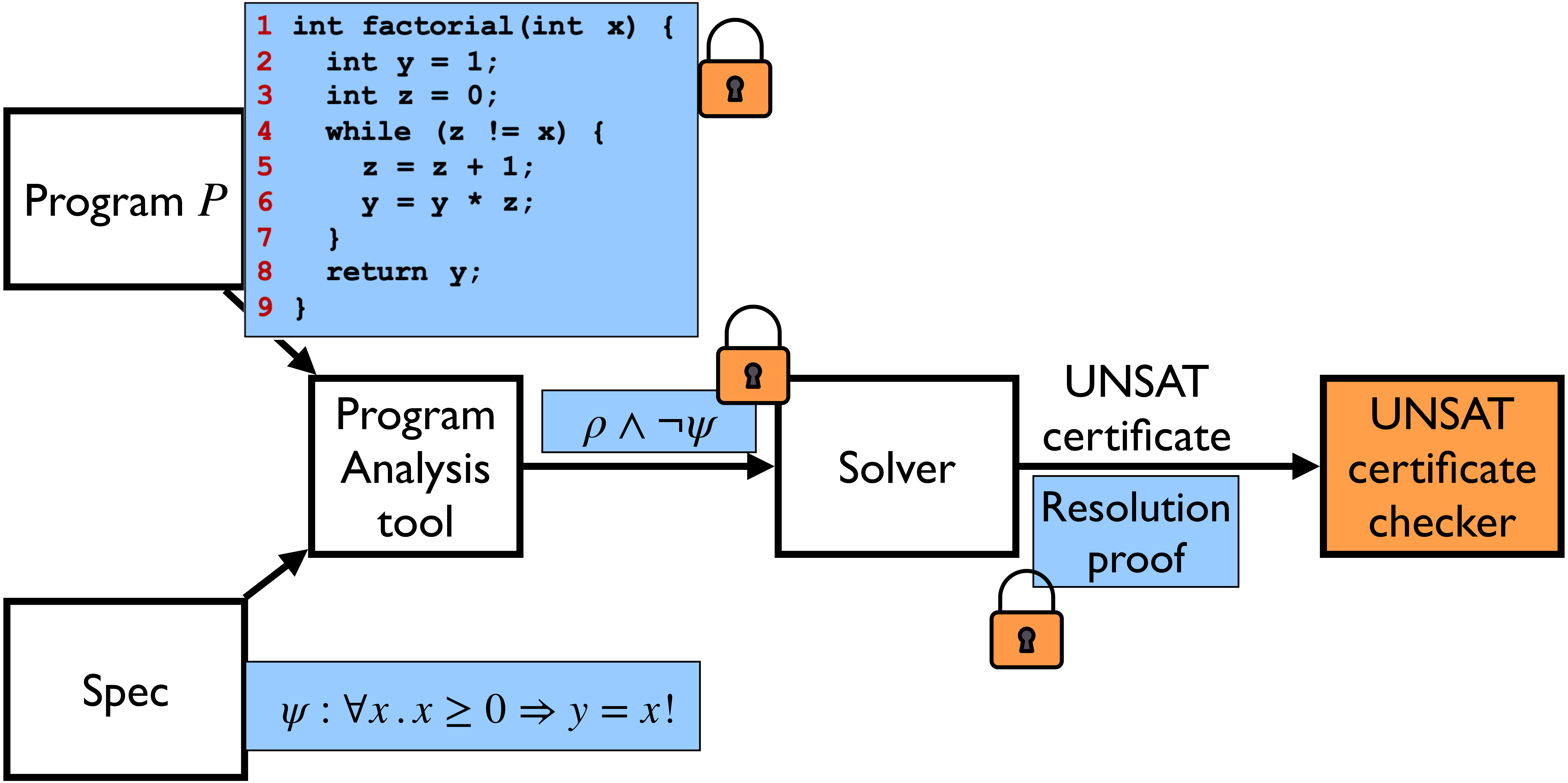
ZKUNSAT: The First Step towards PPFM



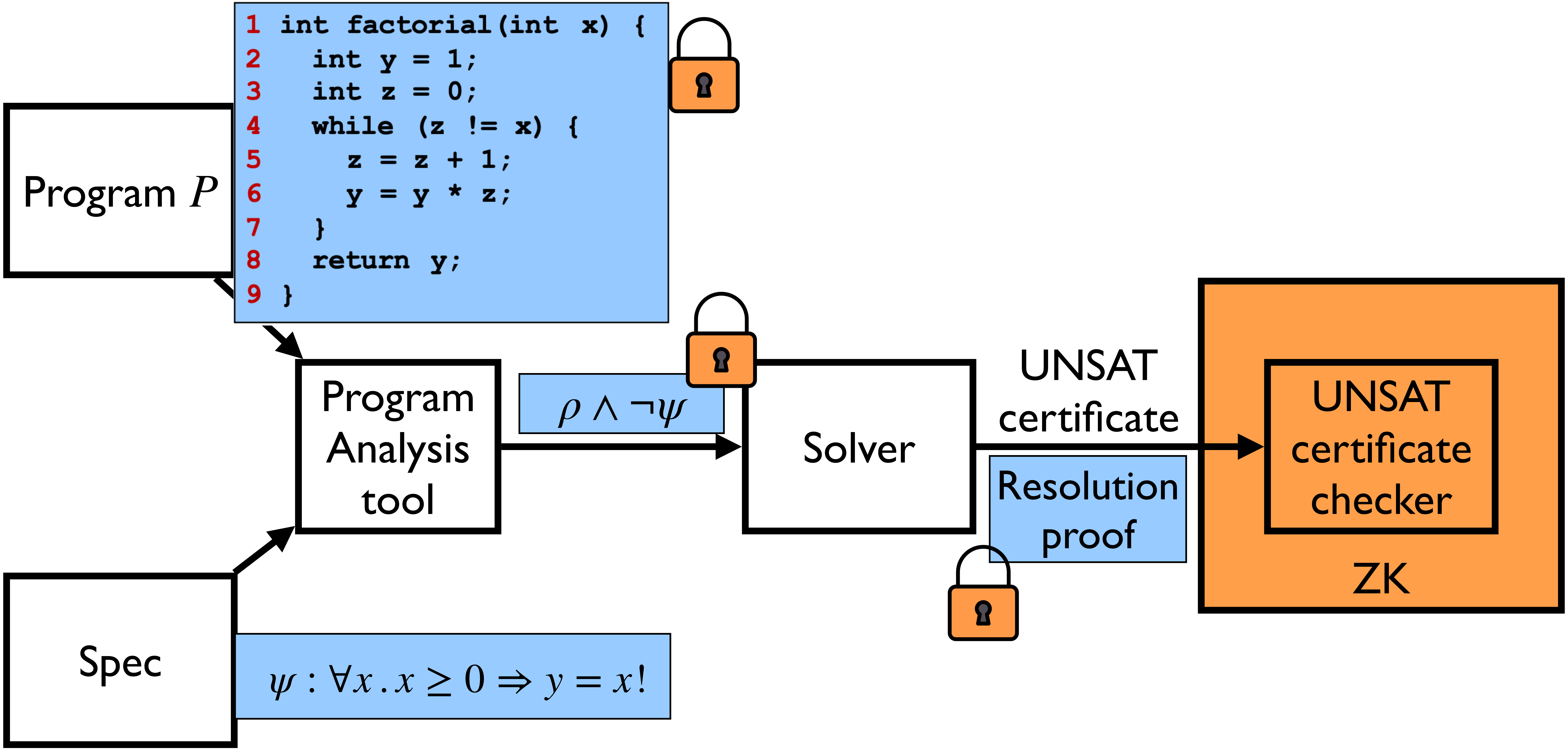
ZKUNSAT: The First Step towards PPFM



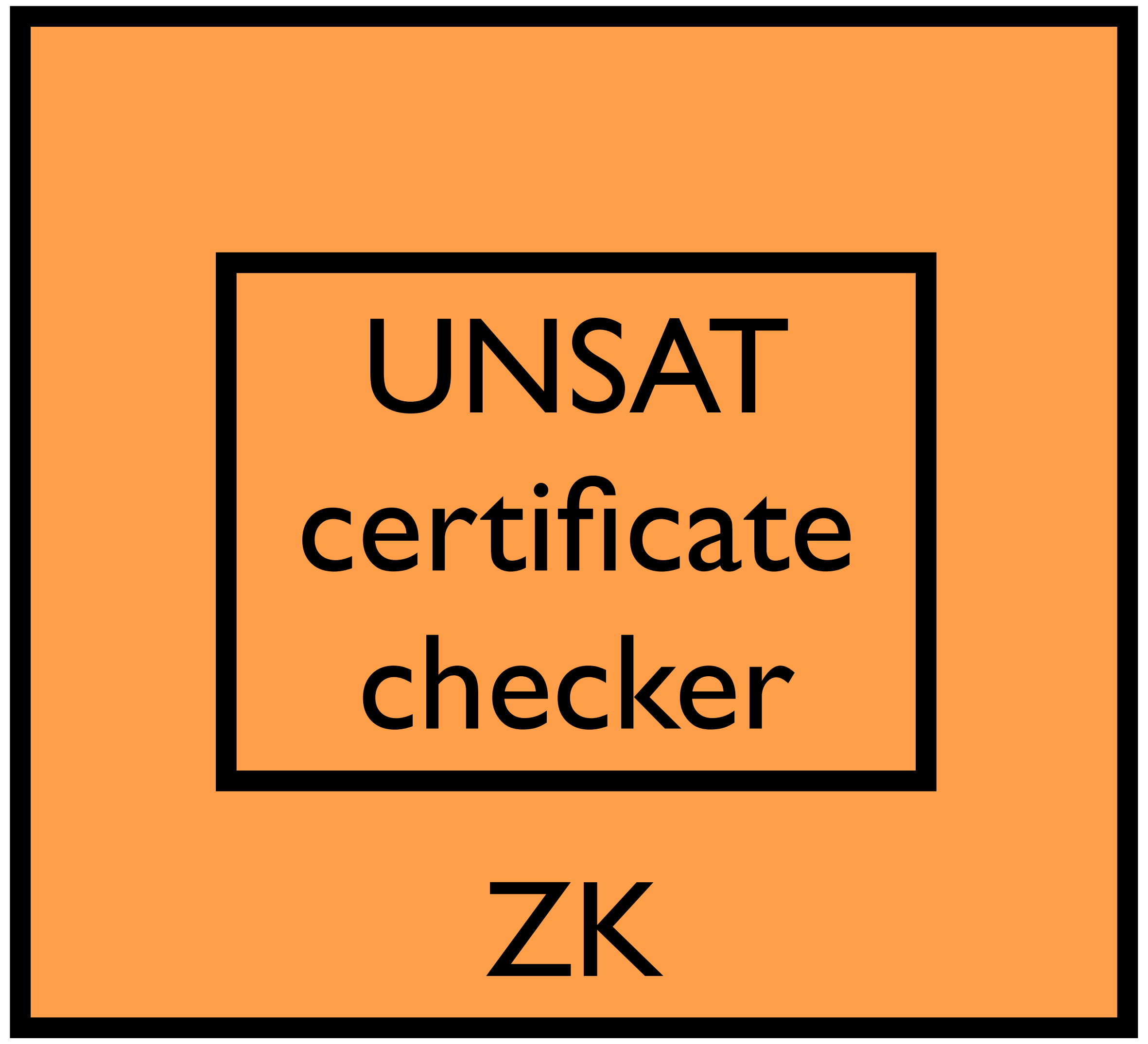
ZKUNSAT: The First Step towards PPFM



ZKUNSAT: The First Step towards PPFM

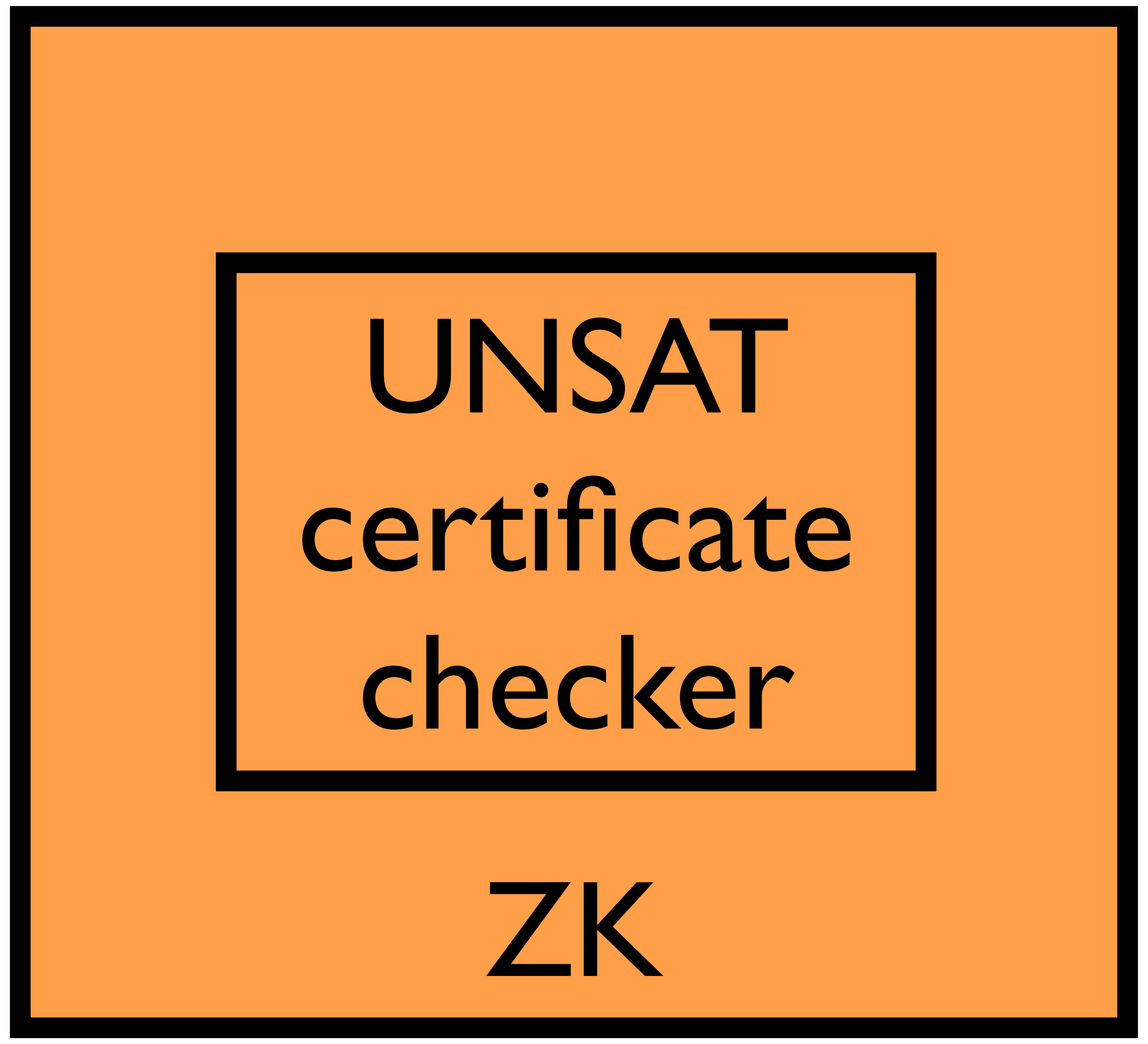


The First Step towards Decentralized Verification



~~**generic implementation of the certificate checker in ZK**~~

The First Step towards Decentralized Verification



The First Step towards Decentralized Verification

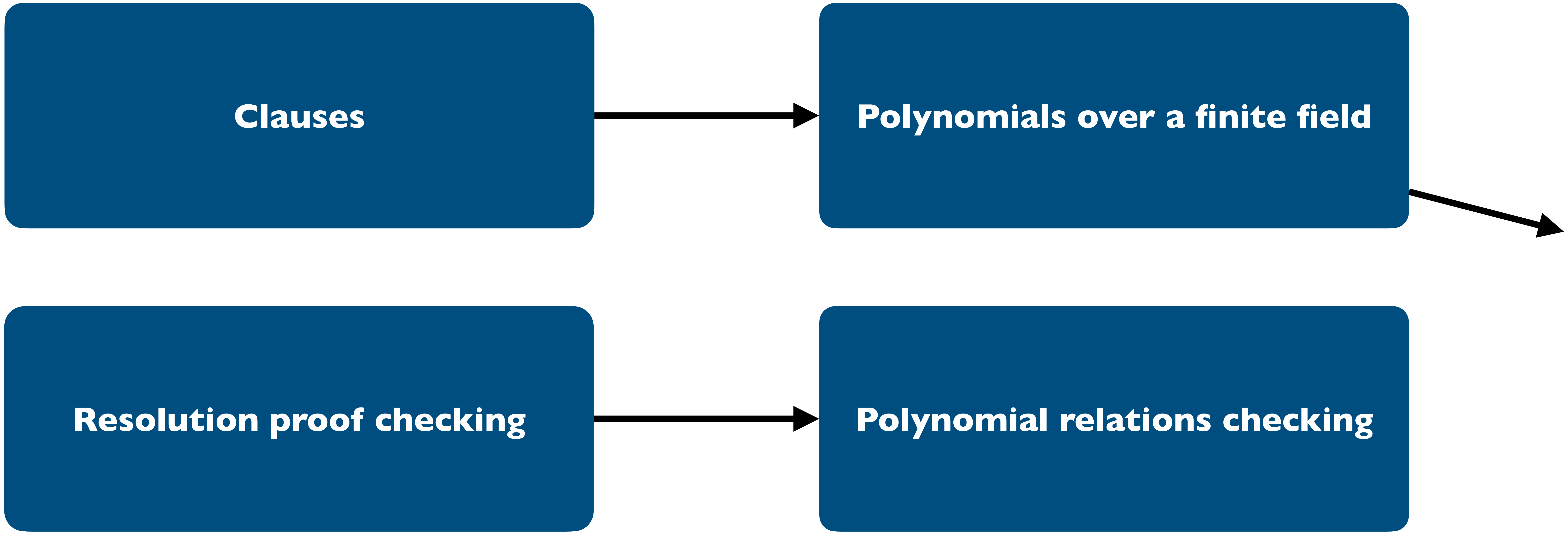
Scales to real-world verification tasks including these for Linux+Windows drivers

The First Step towards Decentralized Verification

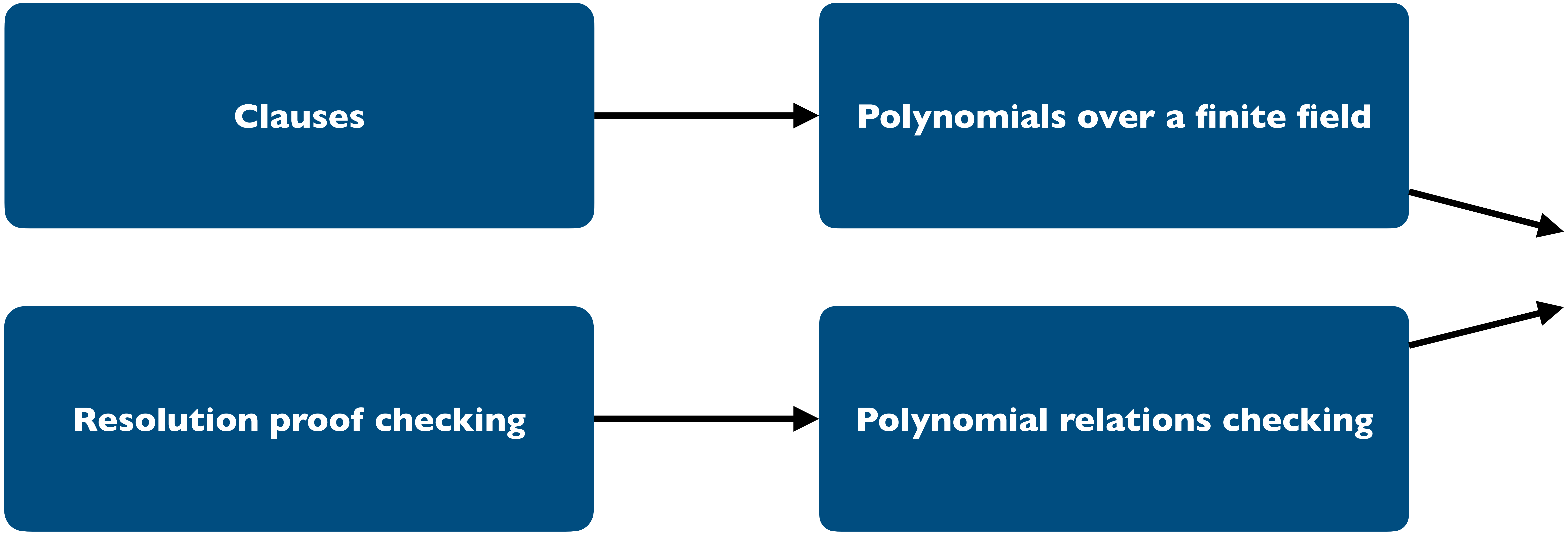
Clauses → Polynomials over a finite field

Resolution proof checking → Polynomial relations checking

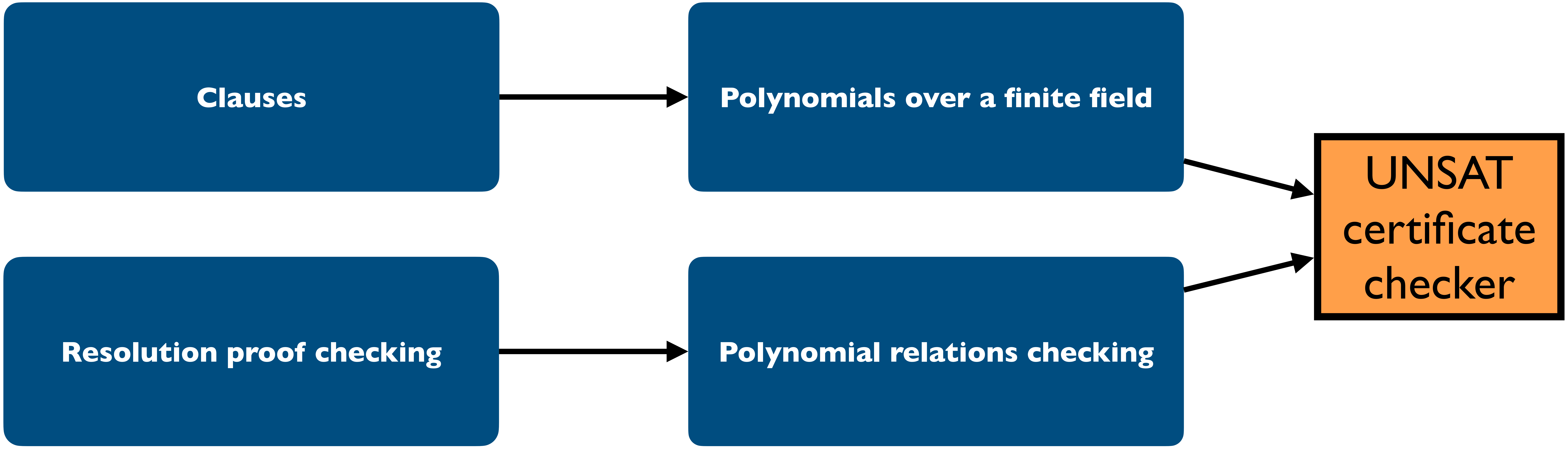
The First Step towards Decentralized Verification



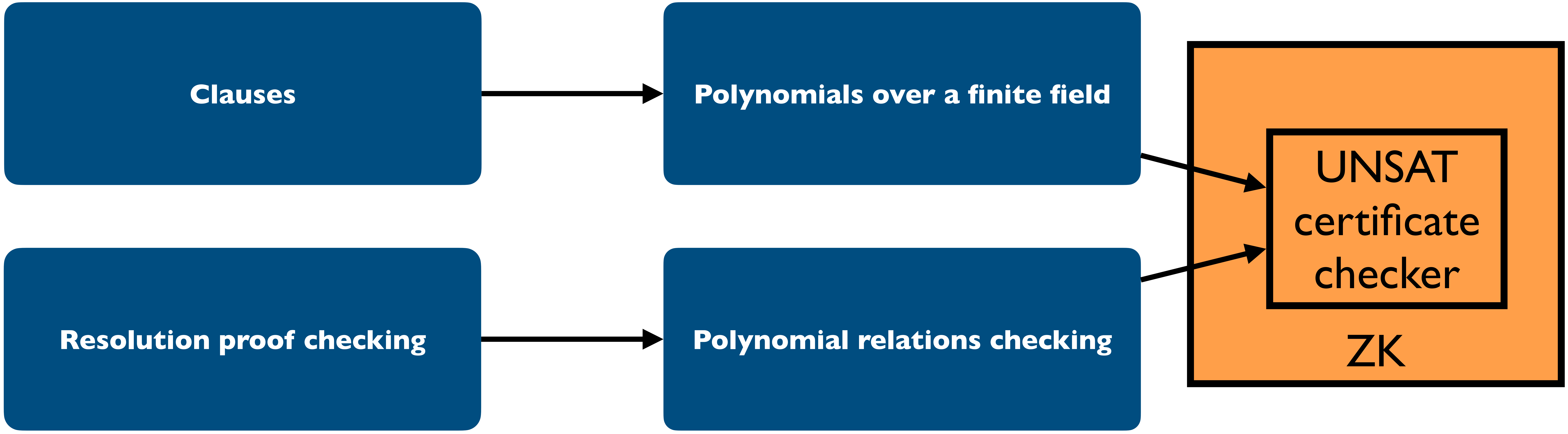
The First Step towards Decentralized Verification



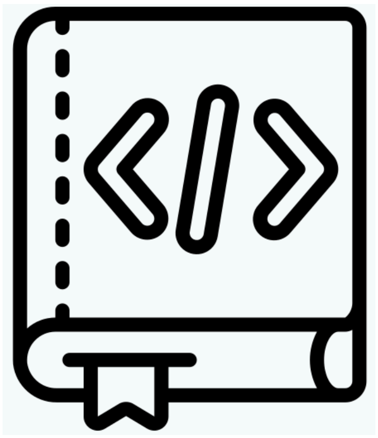
The First Step towards Decentralized Verification



The First Step towards Decentralized Verification



Efficient library for checking relations between polynomials in zero knowledge



Outline

- **Background**
  - Resolution Proof
  - Zero knowledge proof
- **ZKUNSAT**
  - Clause encoding and validating resolvents
  - Clause access and checking consistency
  - Evaluation
- **Future Work**

Refutation Proof

Resolution Rule [Robinson, 65]

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_3 \vee \neg x_2}{c_r = x_1 \vee x_3}$$

Refutation Proof

Resolution Rule [Robinson, 65]

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_3 \vee \neg x_2}{c_r = x_1 \vee x_3}$$

Refutation Proof

Resolution Rule [Robinson, 65]

$$\frac{c_a = x_1 \vee \cancel{x_2}, \quad c_b = x_3 \vee \cancel{\neg x_2}}{c_r = x_1 \vee x_3}$$

Refutation Proof

Resolution Proof in Propositional Logic

$$f = c_0 \wedge c_1 \wedge c_2 \wedge c_3$$

$c_0 : (x_1 \vee x_2)$		$-,-$
$c_1 : (\neg x_1 \vee x_2)$		$-,-$
$c_2 : (\neg x_1 \vee \neg x_2)$		$-,-$
$c_3 : (x_1 \vee \neg x_2)$		$-,-$

Refutation Proof

Resolution Proof in Propositional Logic

$$f = c_0 \land c_1 \land c_2 \land c_3$$

$c_0 : (x_1 \lor x_2)$	$-,-$
$c_1 : (\neg x_1 \lor x_2)$	$-,-$
$c_2 : (\neg x_1 \lor \neg x_2)$	$-,-$
$c_3 : (x_1 \lor \neg x_2)$	$-,-$

Theorem: first-order logic is refutationally complete.

- The proof of the theorem relies on the resolution proof of unsatisfiability.
- If a formula is UNSAT, we can always derive $\perp$ using resolution

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-,-$
$c_1 : (\neg x_1 \vee x_2)$	$-,-$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-,-$
$c_3 : (x_1 \vee \neg x_2)$	$-,-$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Theorem: first-order logic is refutationally complete.

- The proof of the theorem relies on the resolution proof of unsatisfiability.
- If a formula is UNSAT, we can always derive $\perp$ using resolution

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Theorem: first-order logic is refutationally complete.

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-,-$
$c_1 : (\neg x_1 \vee x_2)$	$-,-$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-,-$
$c_3 : (x_1 \vee \neg x_2)$	$-,-$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Theorem: first-order logic is refutationally complete.

- A resolution proof consists of a list of

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Theorem: first-order logic is refutationally complete.

- A resolution proof consists of a list of
  - Resolvents

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Theorem: first-order logic is refutationally complete.

- A resolution proof consists of a list of
  - Resolvents
  - How these resolvents could be obtained.

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

$$\frac{x_1 \vee x_2, \quad \neg x_1 \vee x_2}{x_2}$$

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-,-$
$c_1 : (\neg x_1 \vee x_2)$	$-,-$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-,-$
$c_3 : (x_1 \vee \neg x_2)$	$-,-$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

$$\frac{x_1 \vee x_2, \quad \neg x_1 \vee x_2}{x_2}$$

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \lor x_2)$	$-, -$
$c_1 : (\neg x_1 \lor x_2)$	$-, -$
$c_2 : (\neg x_1 \lor \neg x_2)$	$-, -$
$c_3 : (x_1 \lor \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \bot$	$4, 5$

$$\frac{x_1 \lor x_2, \quad \neg x_1 \lor x_2}{x_2}$$

Refutation Proof

Resolution Proof in Propositional Logic

$\boxed{c_0 : (x_1 \vee x_2)}$	$\boxed{-,-}$
$\boxed{c_1 : (\neg x_1 \vee x_2)}$	$\boxed{-,-}$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-,-$
$c_3 : (x_1 \vee \neg x_2)$	$-,-$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \bot$	$4, 5$

$$\frac{x_1 \vee x_2, \quad \neg x_1 \vee x_2}{x_2}$$

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

$$\frac{x_1 \vee x_2, \quad \neg x_1 \vee x_2}{x_2}$$

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \lor x_2)$	$-, -$
$c_1 : (\neg x_1 \lor x_2)$	$-, -$
$c_2 : (\neg x_1 \lor \neg x_2)$	$-, -$
$c_3 : (x_1 \lor \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

$$\frac{x_1 \lor x_2, \quad \neg x_1 \lor x_2}{x_2}$$

- Fetch two clauses used to derived the resolvent

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \bot$	$4, 5$

$$\frac{x_1 \vee x_2, \quad \neg x_1 \vee x_2}{x_2}$$

- Fetch two clauses used to derived the resolvent

Refutation Proof

Resolution Proof in Propositional Logic

c_0 :	$(x_1 \lor x_2)$	$-,-$
c_1 :	$(\neg x_1 \lor x_2)$	$-,-$
c_2 :	$(\neg x_1 \lor \neg x_2)$	$-,-$
c_3 :	$(x_1 \lor \neg x_2)$	$-,-$
c_4 :	x_2	$0, 1$ ✓
c_5 :	$\neg x_2$	$2, 3$
c_6 :	$\perp$	$4, 5$

$$\frac{x_1 \lor x_2, \quad \neg x_1 \lor x_2}{x_2}$$

- Fetch two clauses used to derived the resolvent

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \bot$	$4, 5$

✓

$$\frac{x_1 \vee x_2, \quad \neg x_1 \vee x_2}{x_2}$$

- Fetch two clauses used to derived the resolvent
- Check the application of resolution rule is correctly executed

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-,-$
$c_1 : (\neg x_1 \vee x_2)$	$-,-$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-,-$
$c_3 : (x_1 \vee \neg x_2)$	$-,-$
$c_4 : x_2$	$0, 1$ ✓
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Repeat for each resolvent until meet a contradiction

Refutation Proof

Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Repeat for each resolvent until meet a contradiction

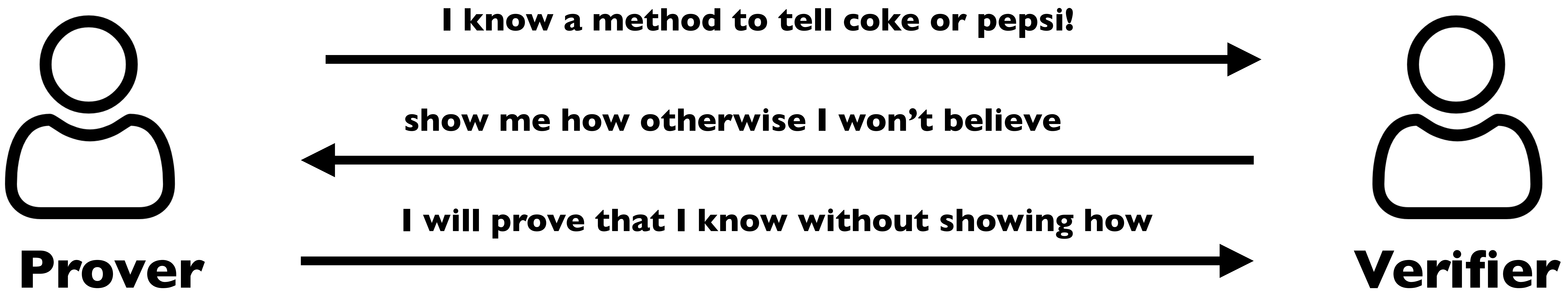
Refutation Proof

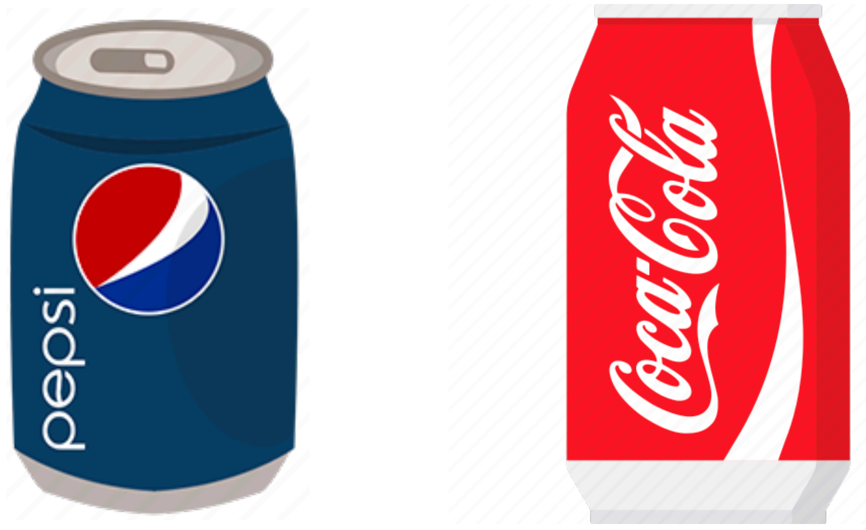
Resolution Proof in Propositional Logic

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$ ✓
$c_5 : \neg x_2$	$2, 3$ ✓
$c_6 : \perp$	$4, 5$ ✓

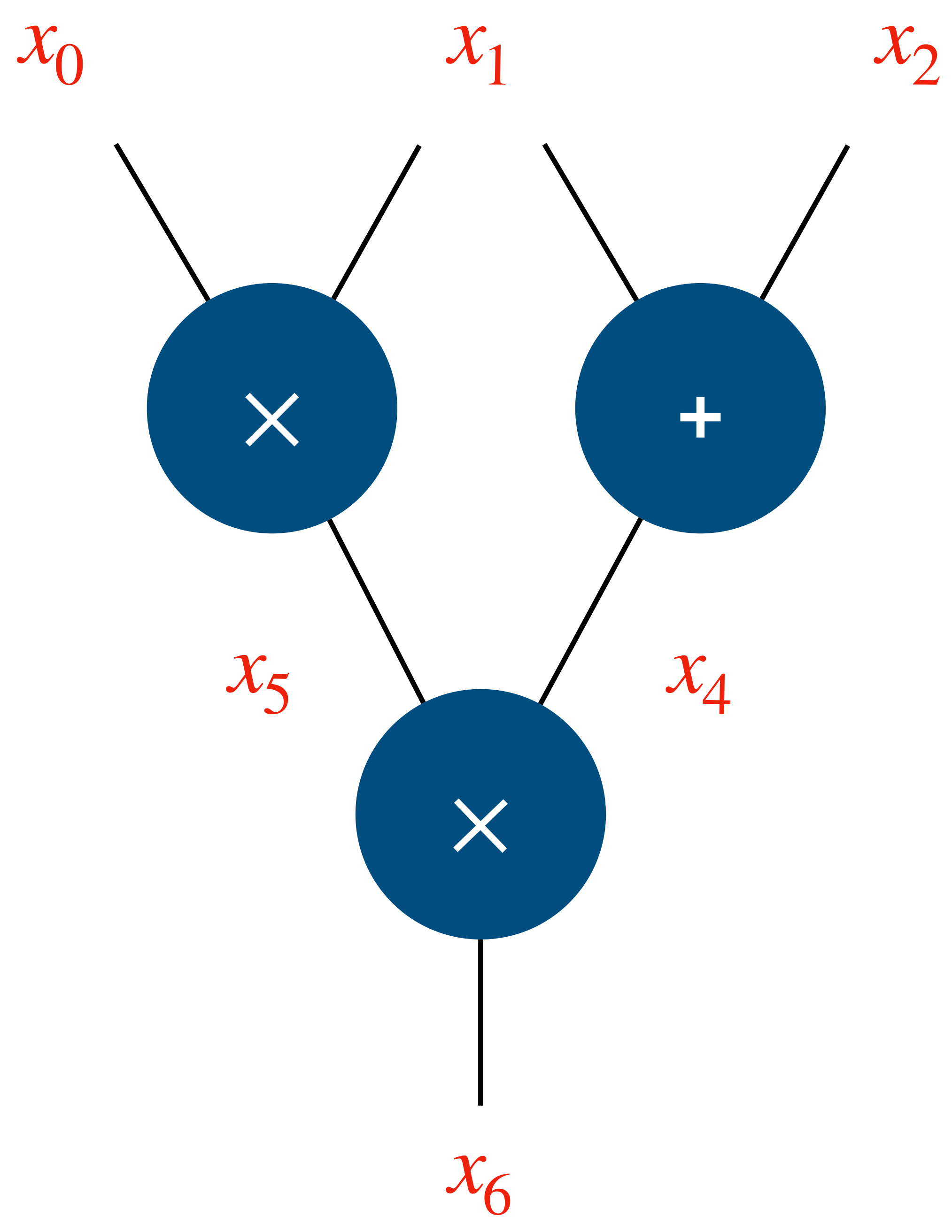
Repeat for each resolvent until meet a contradiction

Zero Knowledge Proof : Coke or Pepsi





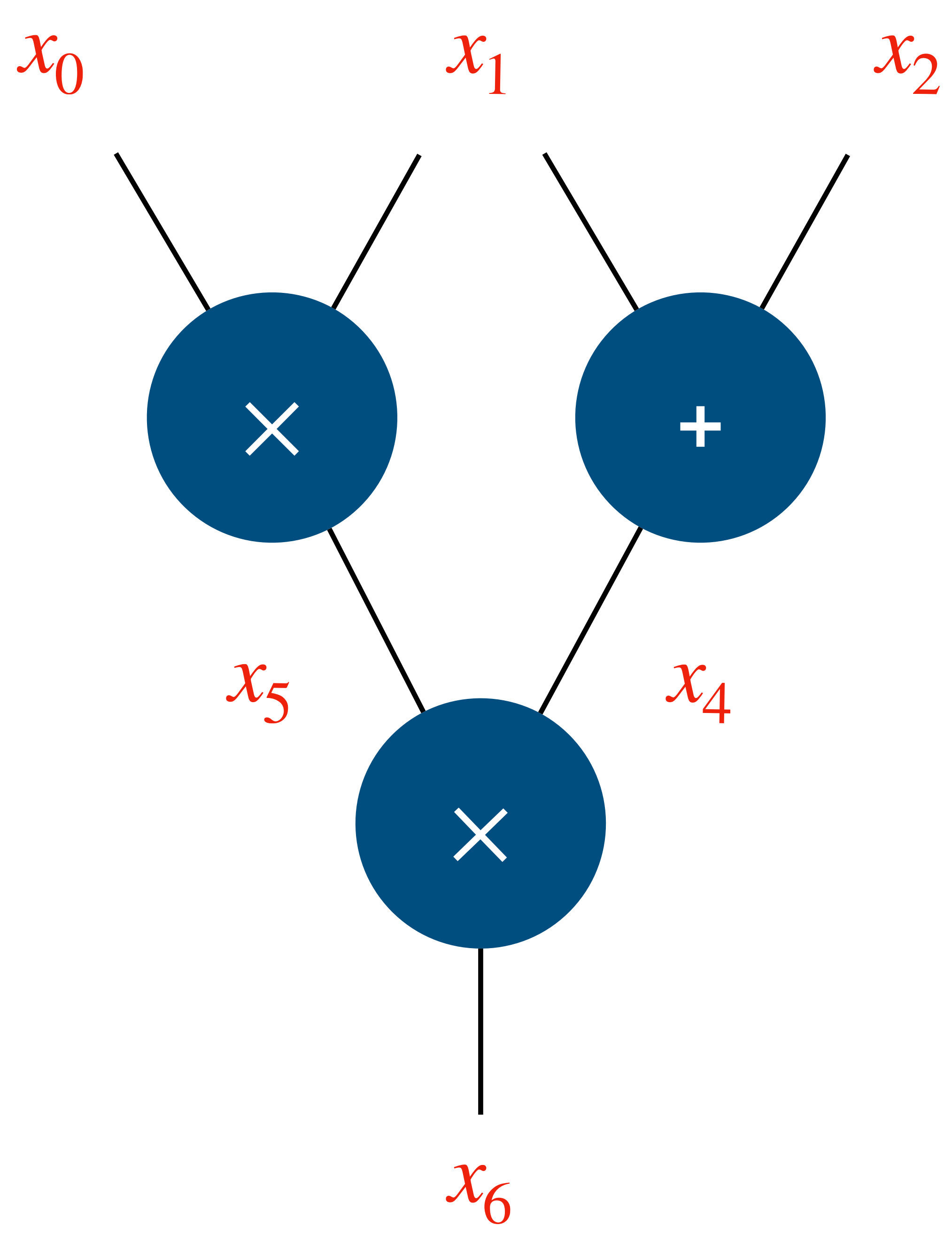
Zero Knowledge Proof



A public circuit C

Zero Knowledge Proof

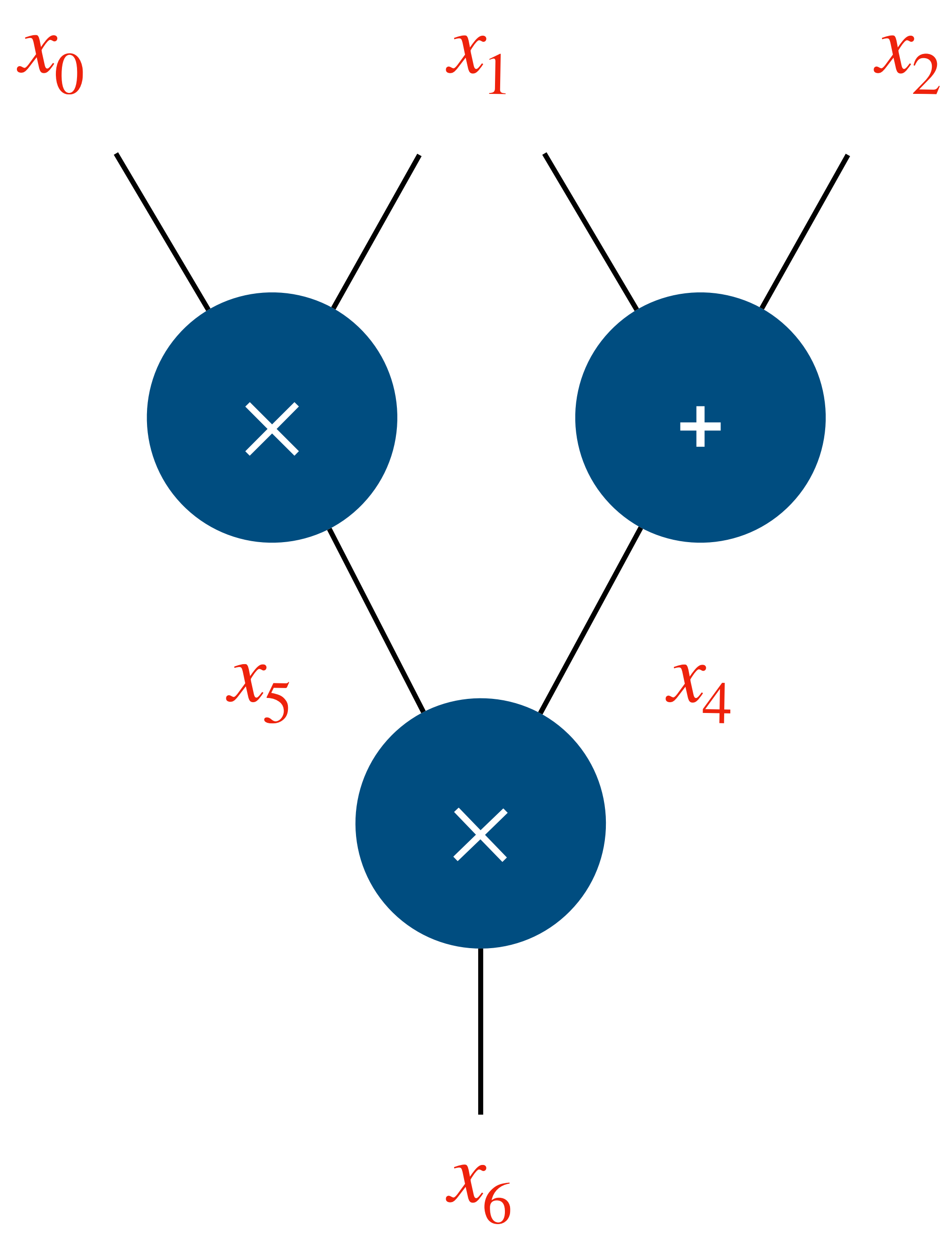
- **Prover** knows an input w such that $C(w) = 1$, and tries to
  - Convince the verifier that $C(w) = 1$
  - Keep information of w private



A public circuit C

Zero Knowledge Proof

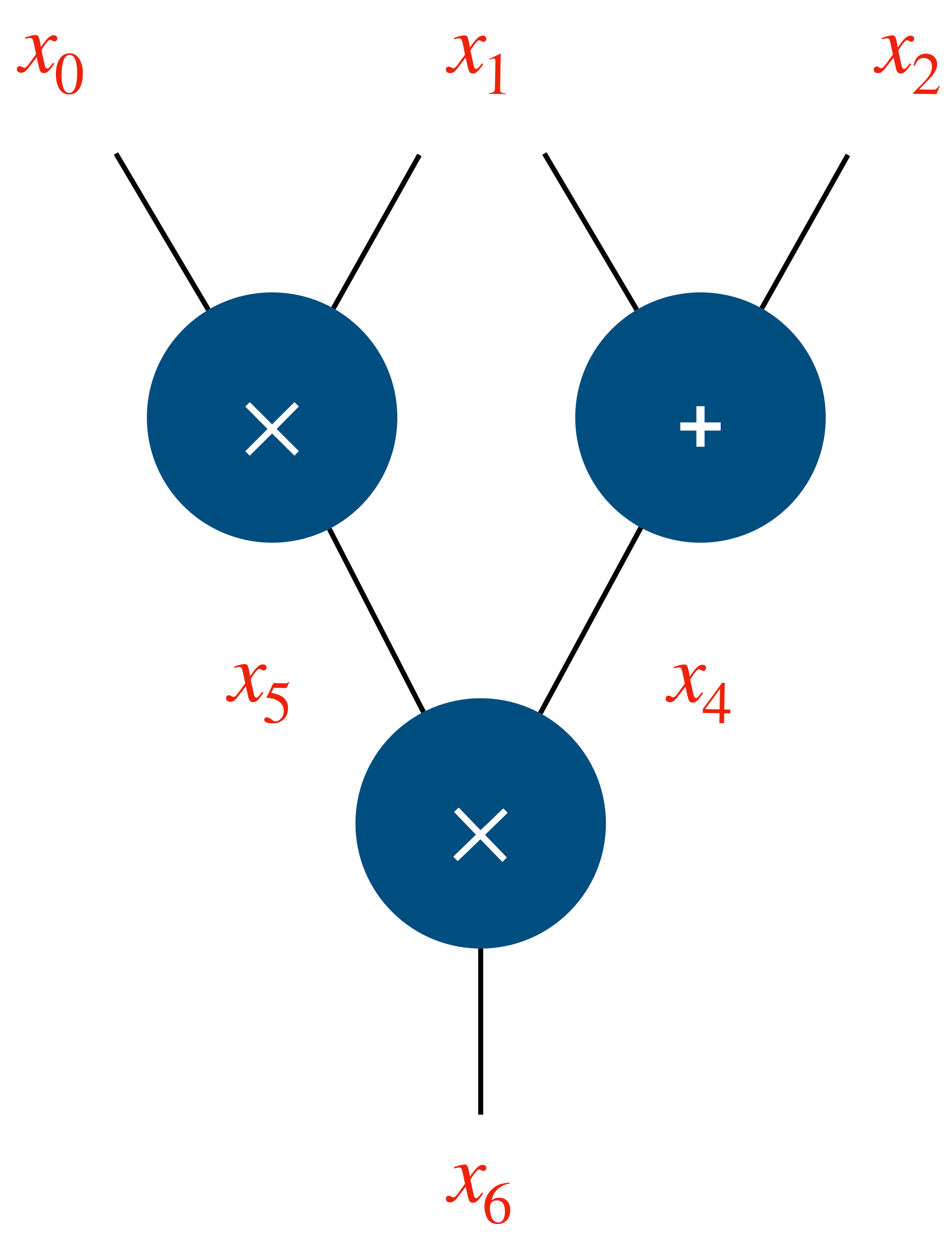
- **Prover** knows an input w such that $C(w) = 1$, and tries to
  - Convince the verifier that $C(w) = 1$
  - Keep information of w private
- **Verifier**:
  - Validate prover's claim about the circuit C



A public circuit C

Zero Knowledge Proof

- **Prover** knows an input w such that $C(w) = 1$, and tries to
  - Convince the verifier that $C(w) = 1$
  - Keep information of w private
- **Verifier**:
  - Validate prover's claim about the circuit C



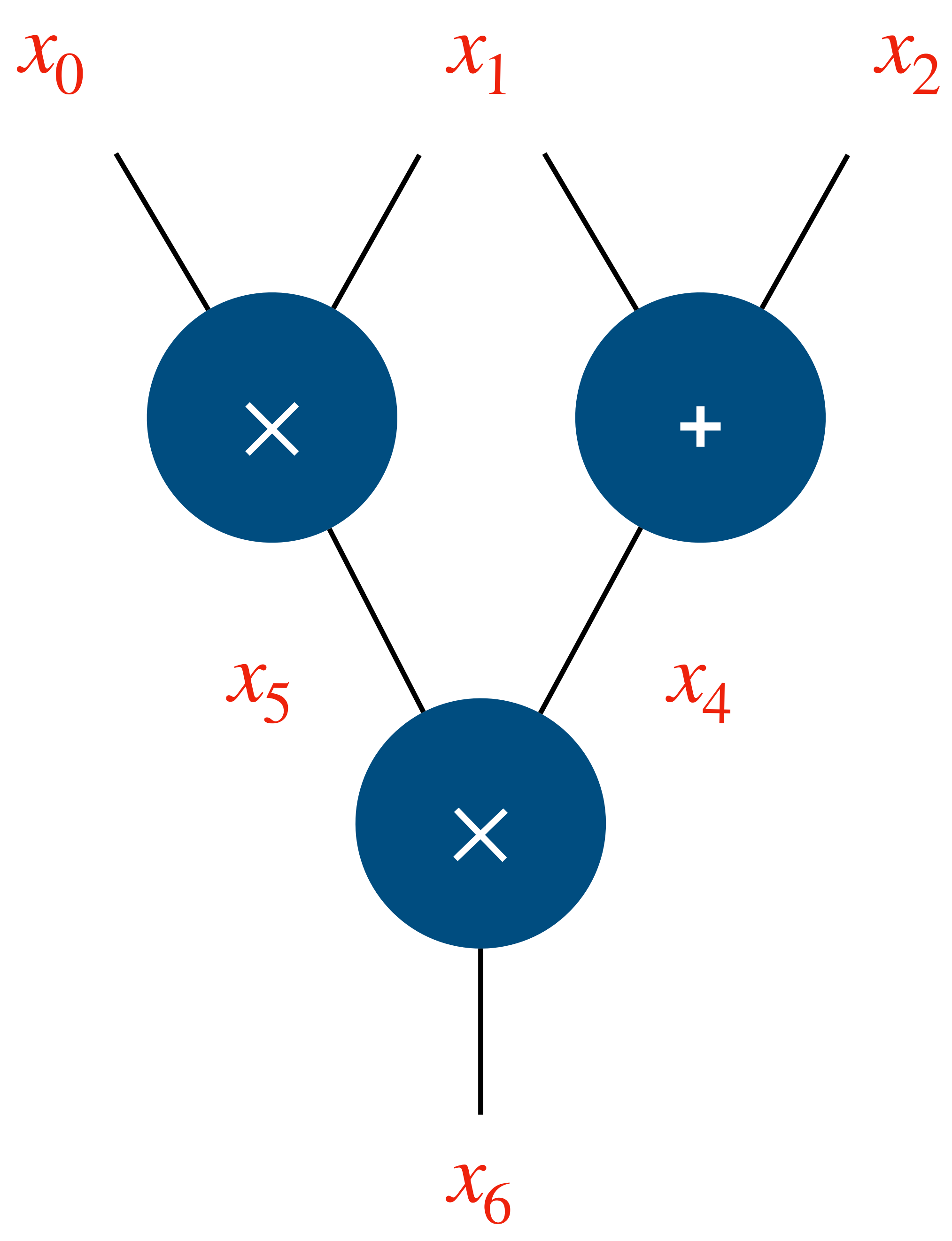
A public circuit C

Zero Knowledge Proof

- **Prover** knows an input w such that $C(w) = 1$, and tries to
  - Convince the verifier that $C(w) = 1$
  - Keep information of w private
- **Verifier**:
  - Validate prover's claim about the circuit C

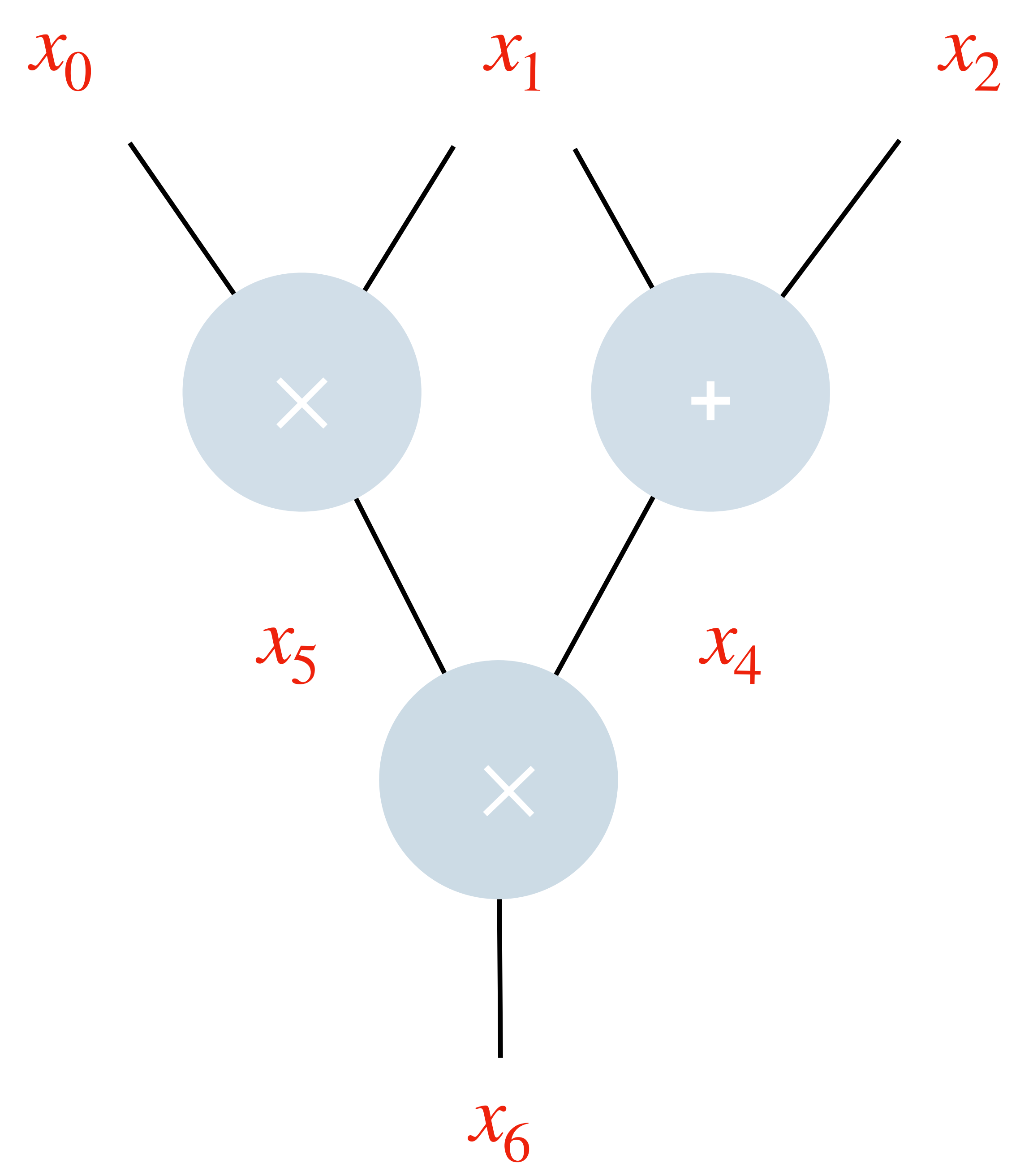
Both of prover and verifier can be malicious.

- Prover might cheat
- Verifier tries to learn w



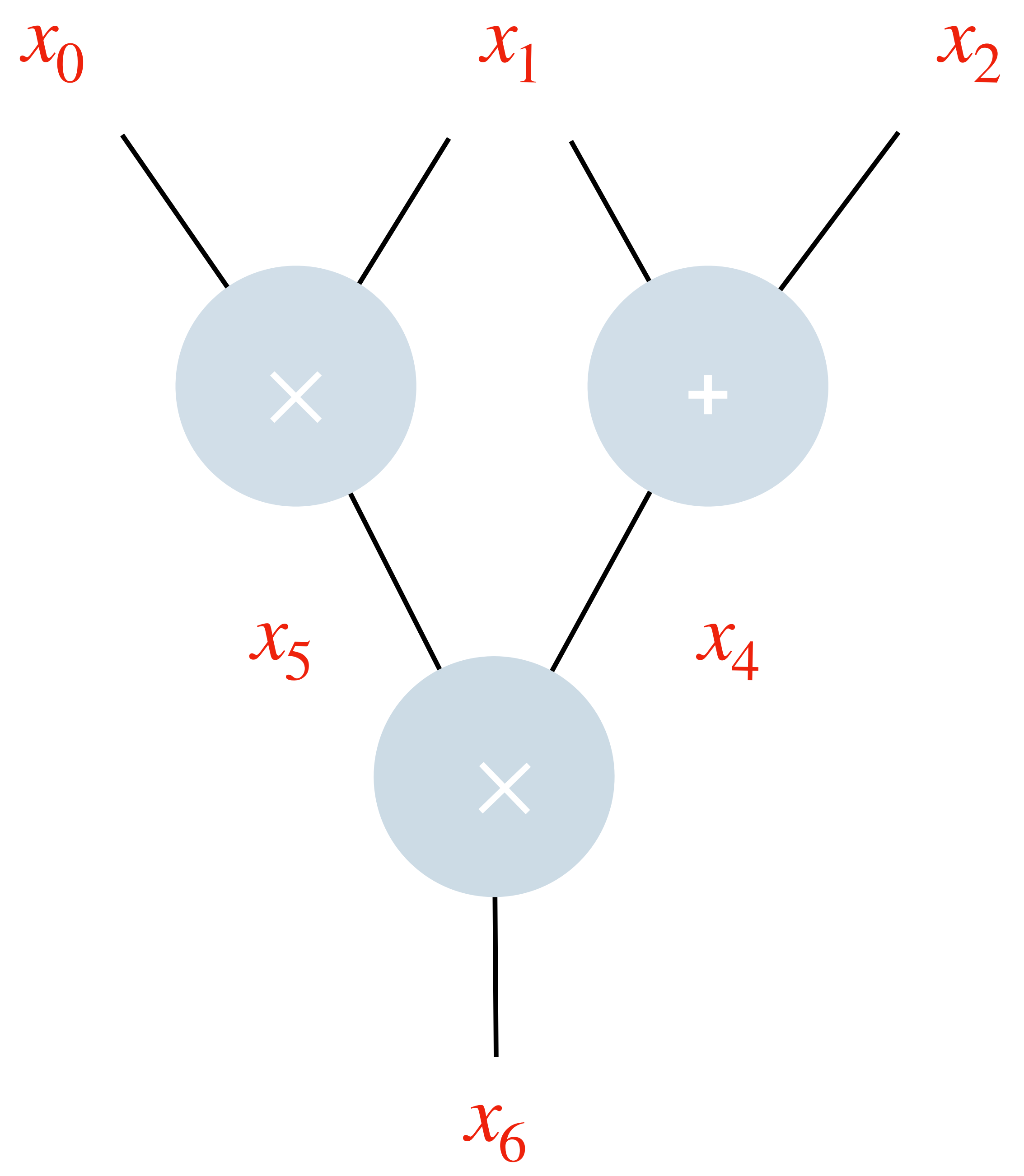
A public circuit C

Zero Knowledge Proof



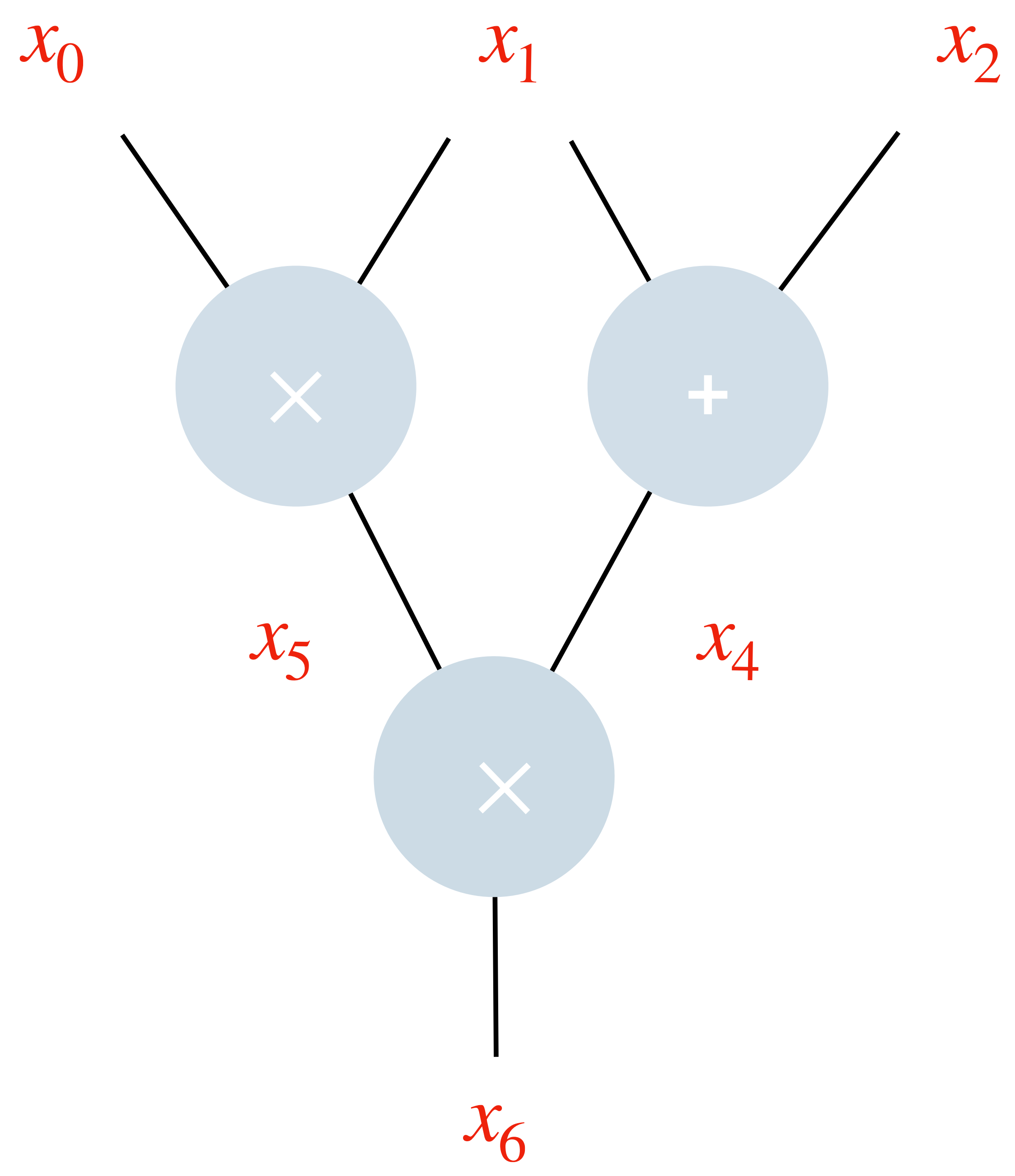
Zero Knowledge Proof

- Authenticated value: ciphertext that



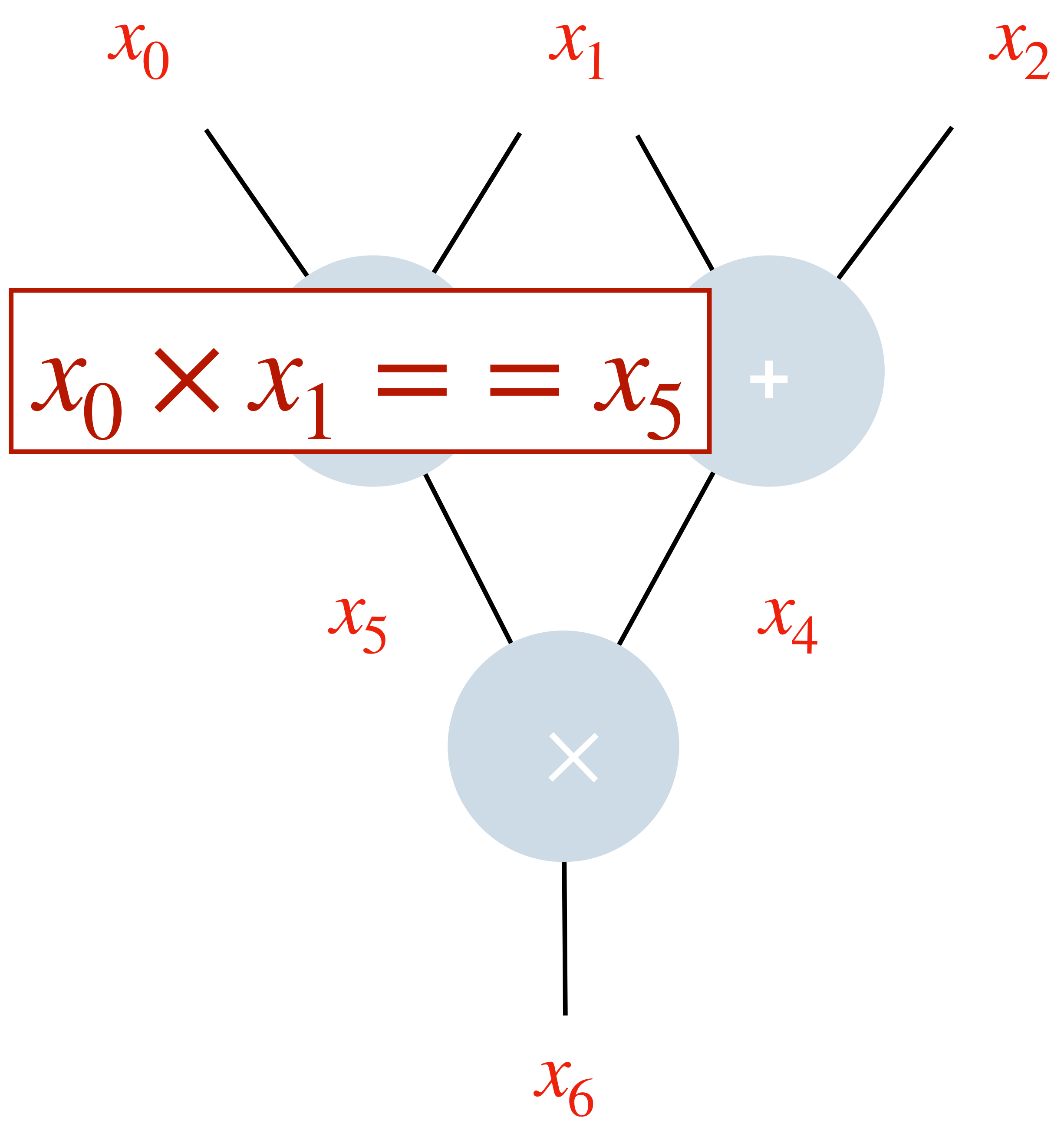
Zero Knowledge Proof

- Authenticated value: ciphertext that
    - Hide underlying value



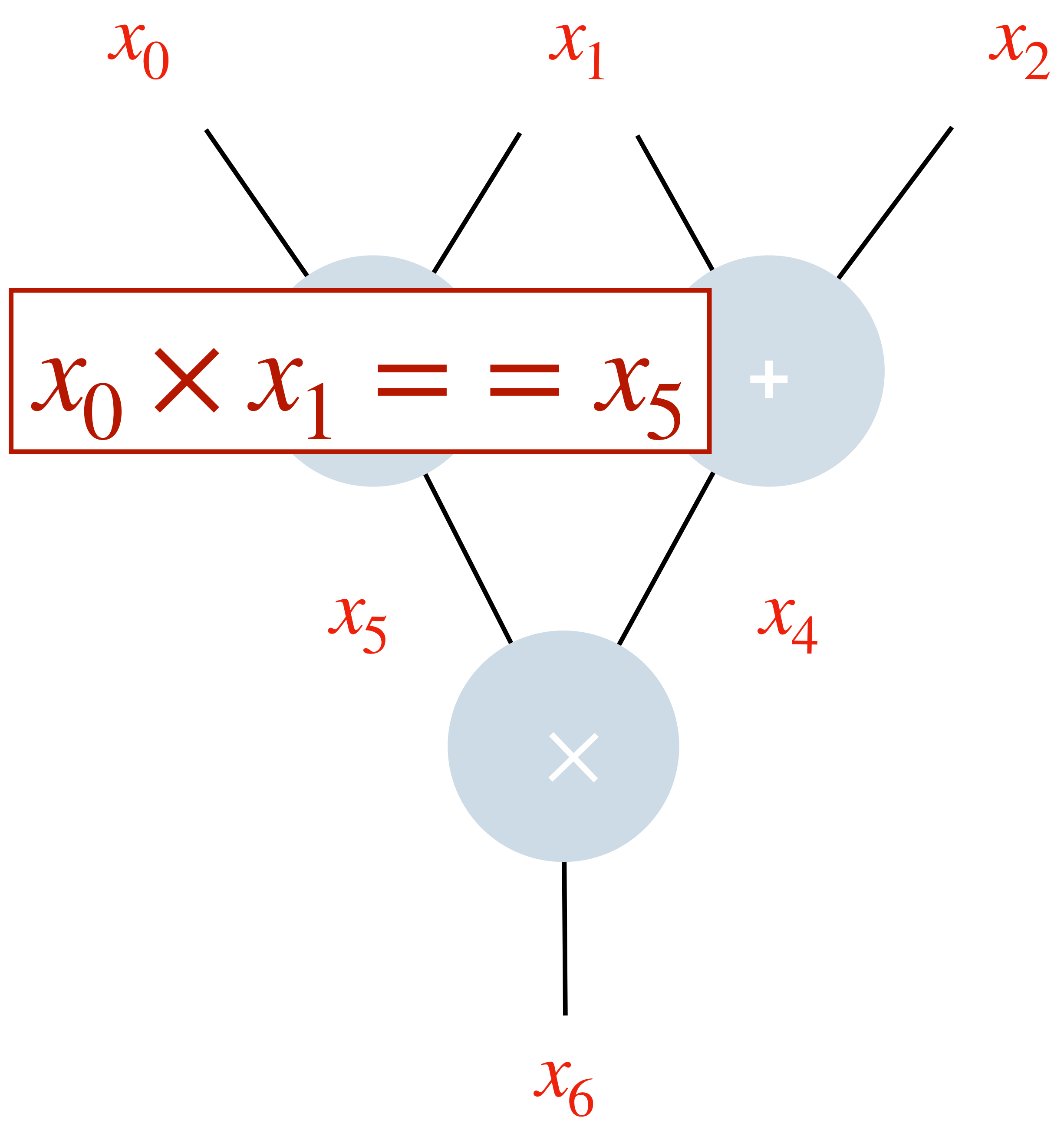
Zero Knowledge Proof

- Authenticated value: ciphertext that
  - Hide underlying value



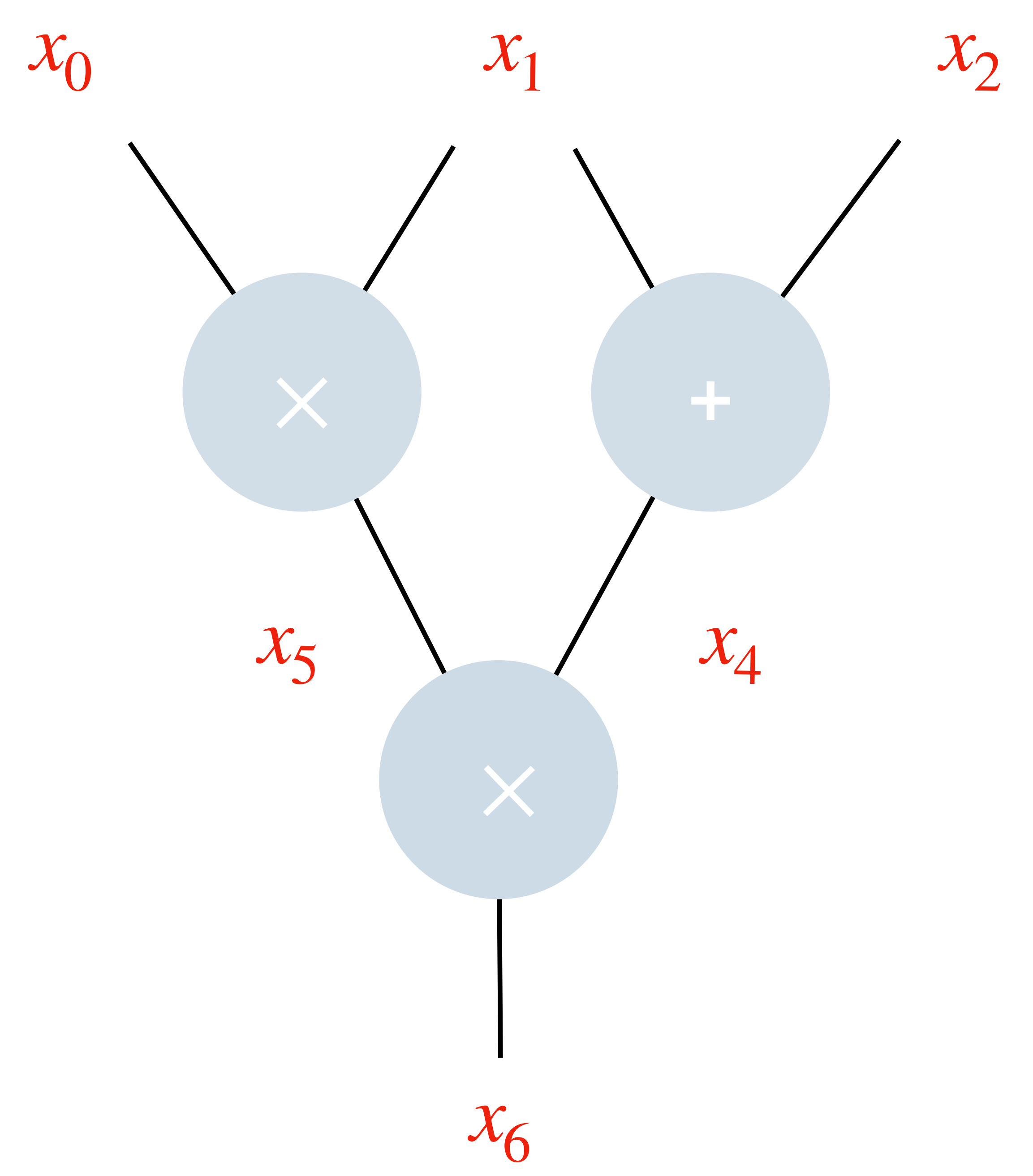
Zero Knowledge Proof

- Authenticated value: ciphertext that
  - Hide underlying value
  - Enable verifier to check the relations



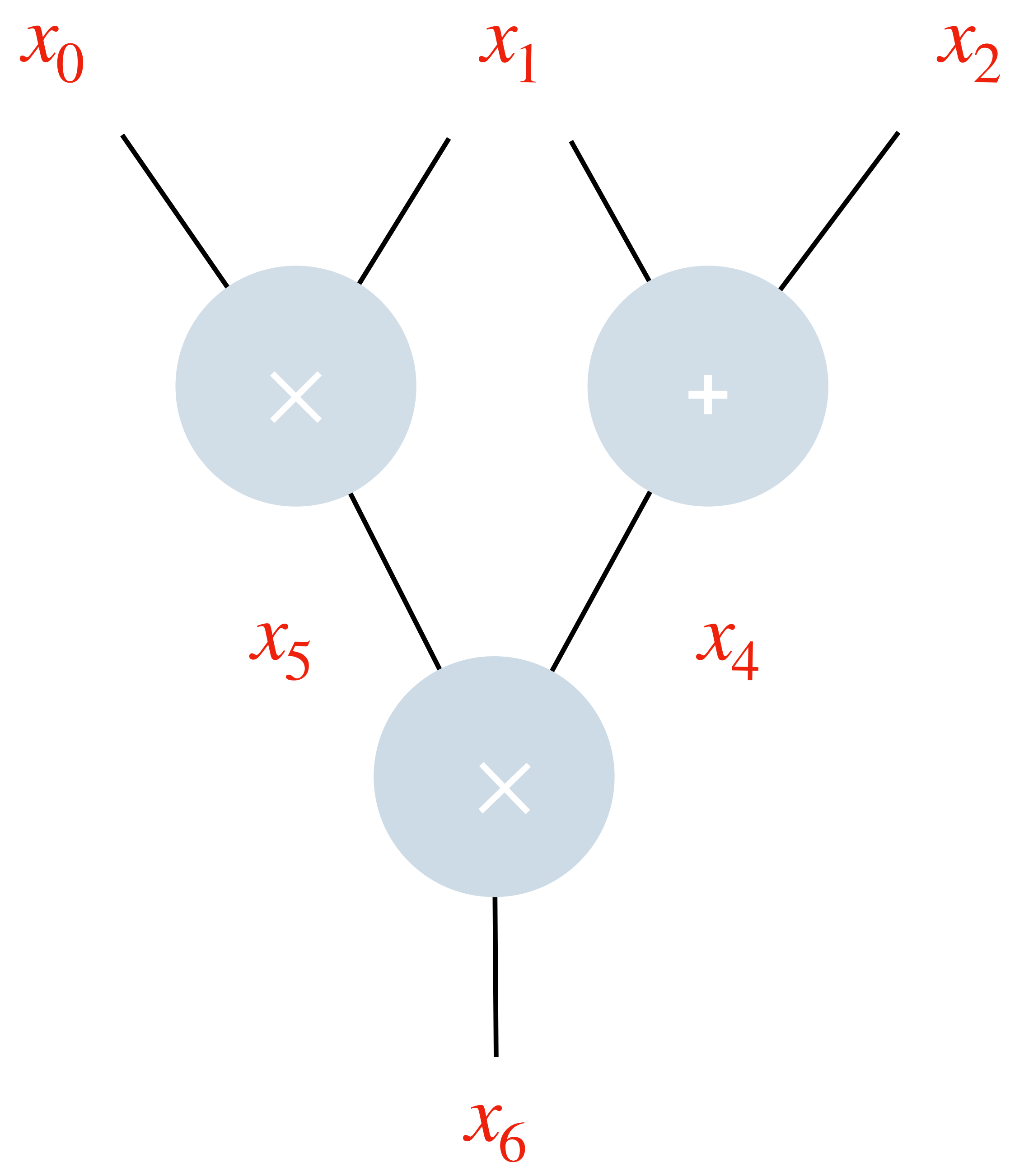
Zero Knowledge Proof

- Authenticated value: ciphertext that
  - Hide underlying value
  - Enable verifier to check the relations



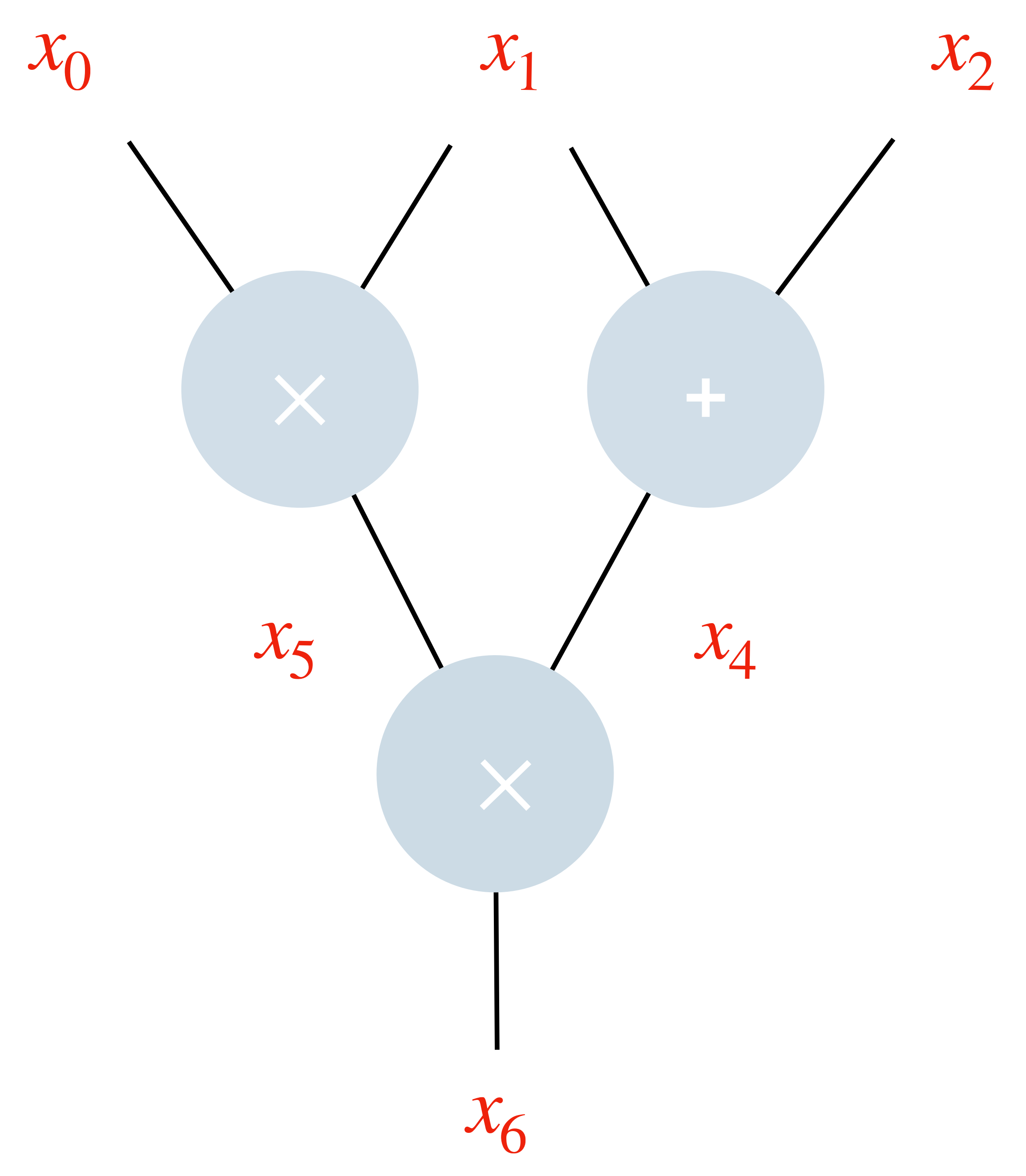
Zero Knowledge Proof

- Authenticated value: ciphertext that
  - Hide underlying value
  - Enable verifier to check the relations
  - Prevent prover from cheating about the underlying value



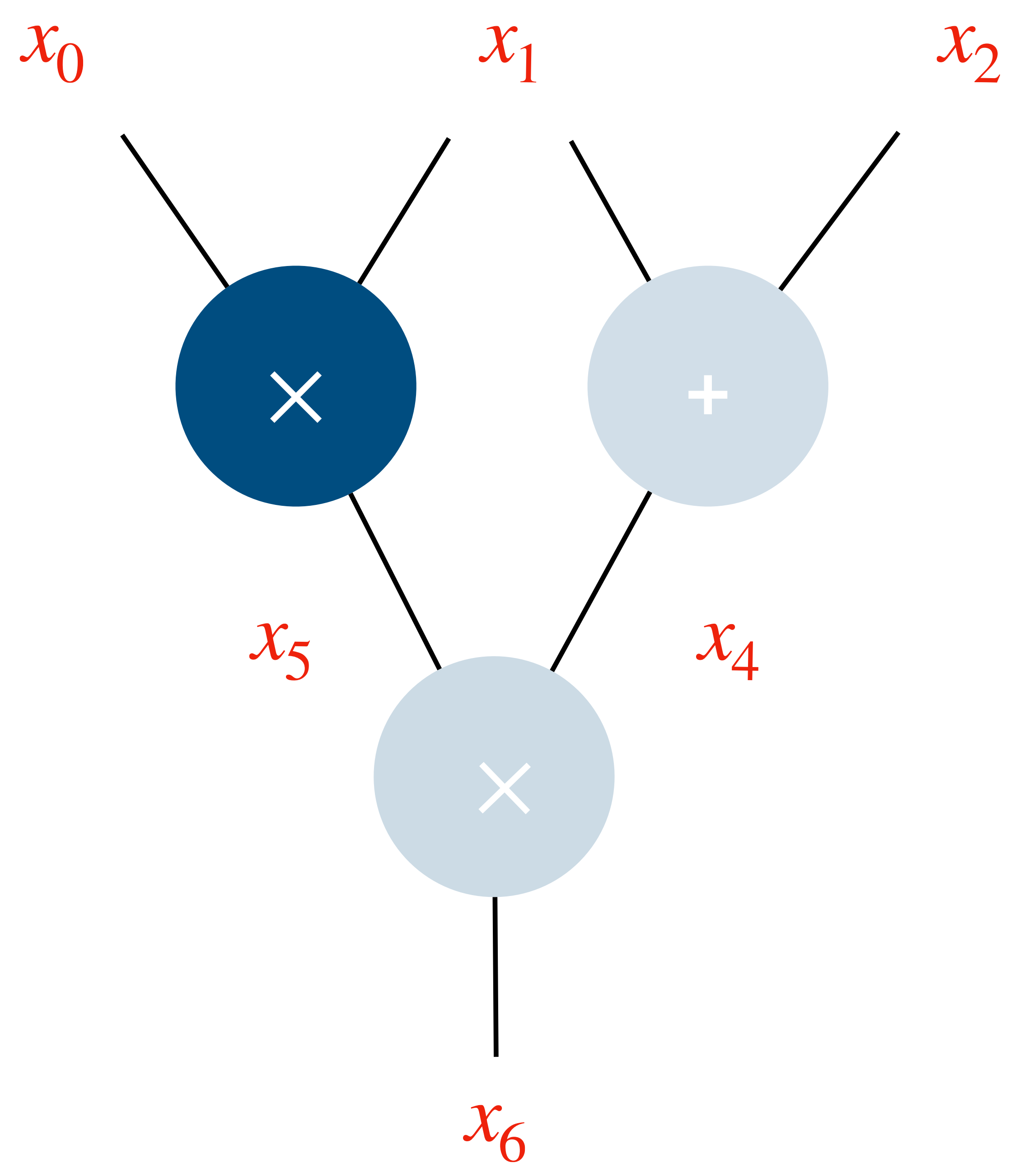
Zero Knowledge Proof

- Authenticated value: ciphertext that
    - Hide underlying value
    - Enable verifier to check the relations
    - Prevent prover from cheating about the underlying value
- Gate-by-gate paradigm [Beaver et al. 90]:
    - Prover authenticates the value over all wires
    - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
    - Prover reveals the value of output of the circuit



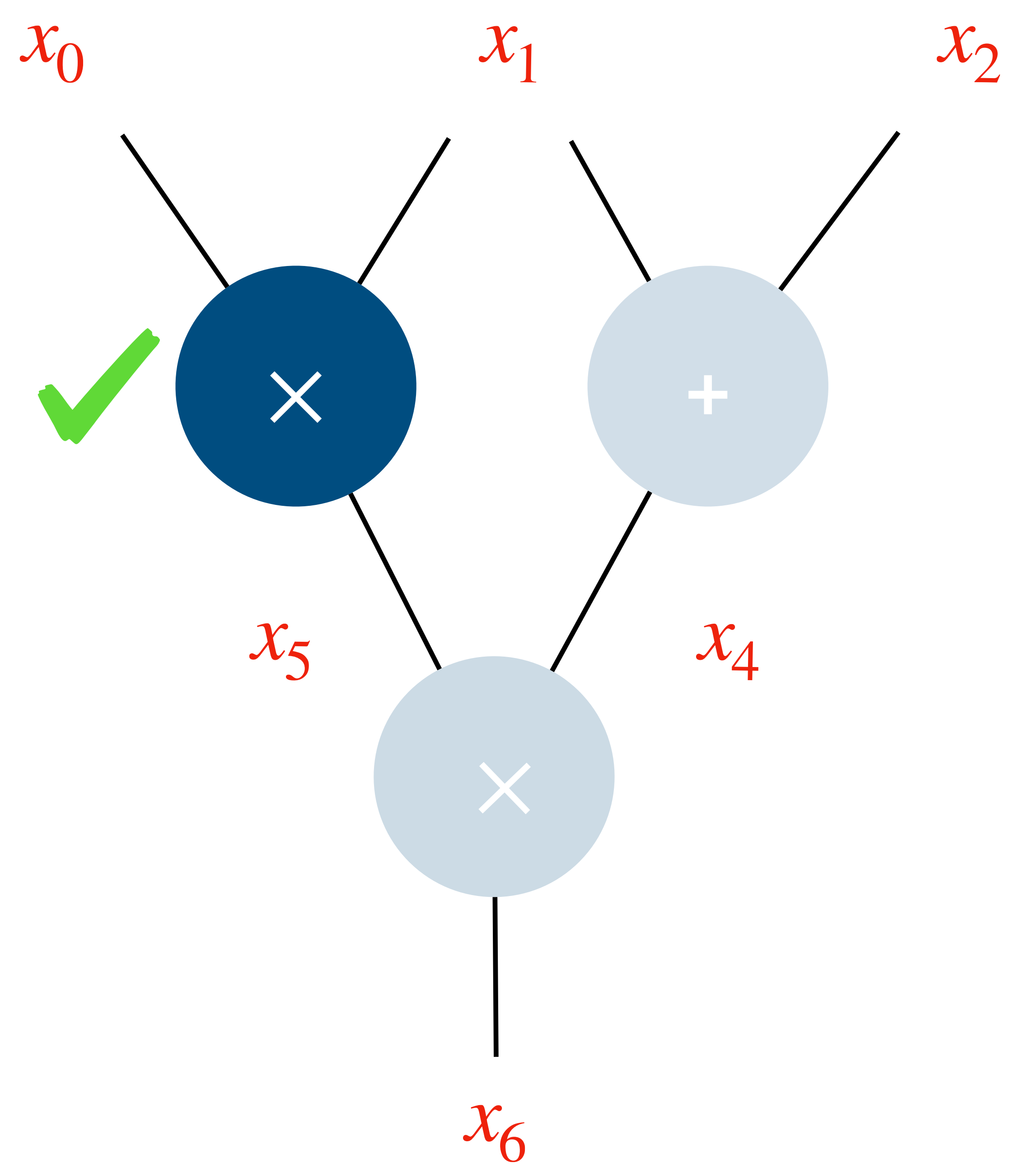
Zero Knowledge Proof

- Authenticated value: ciphertext that
  - Hide underlying value
  - Enable verifier to check the relations
  - Prevent prover from cheating about the underlying value
- Gate-by-gate paradigm [Beaver et al. 90]:
  - Prover authenticates the value over all wires
  - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
  - Prover reveals the value of output of the circuit



Zero Knowledge Proof

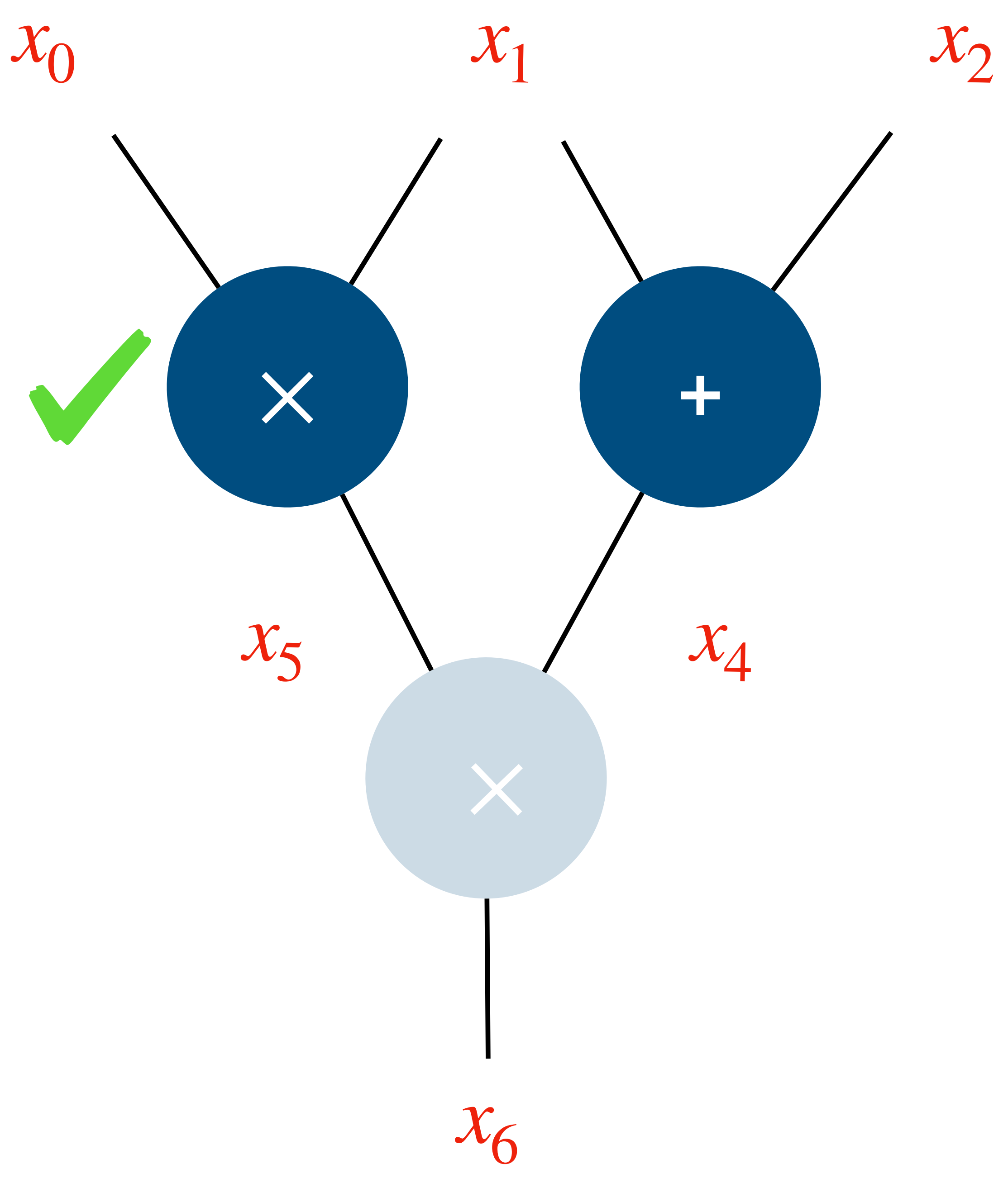
- Authenticated value: ciphertext that
    - Hide underlying value
    - Enable verifier to check the relations
    - Prevent prover from cheating about the underlying value



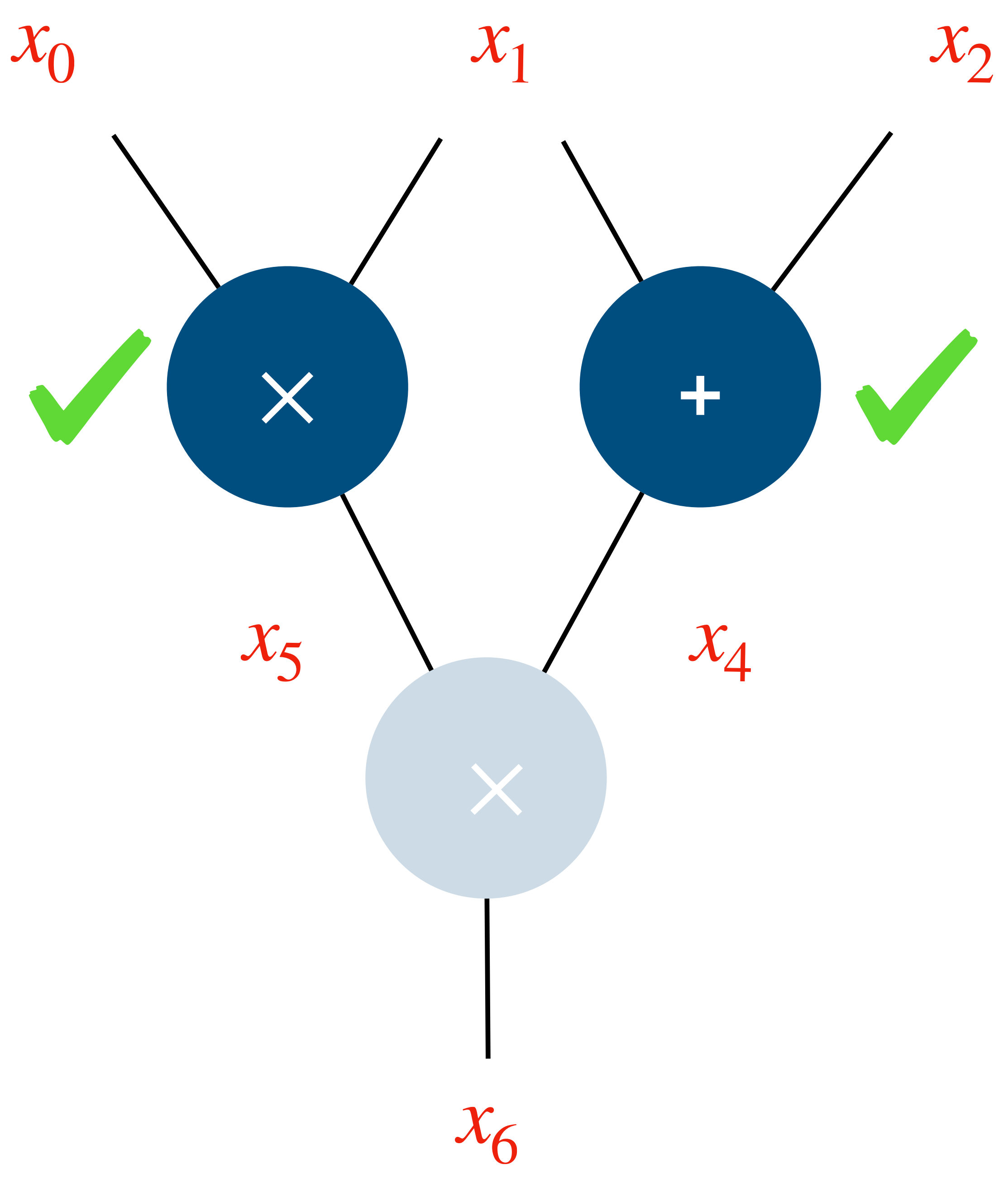
- Gate-by-gate paradigm [Beaver et al. 90]:
    - Prover authenticates the value over all wires
    - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
    - Prover reveals the value of output of the circuit

Zero Knowledge Proof

- Authenticated value: ciphertext that
  - Hide underlying value
  - Enable verifier to check the relations
  - Prevent prover from cheating about the underlying value
- Gate-by-gate paradigm [Beaver et al. 90]:
  - Prover authenticates the value over all wires
  - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
  - Prover reveals the value of output of the circuit



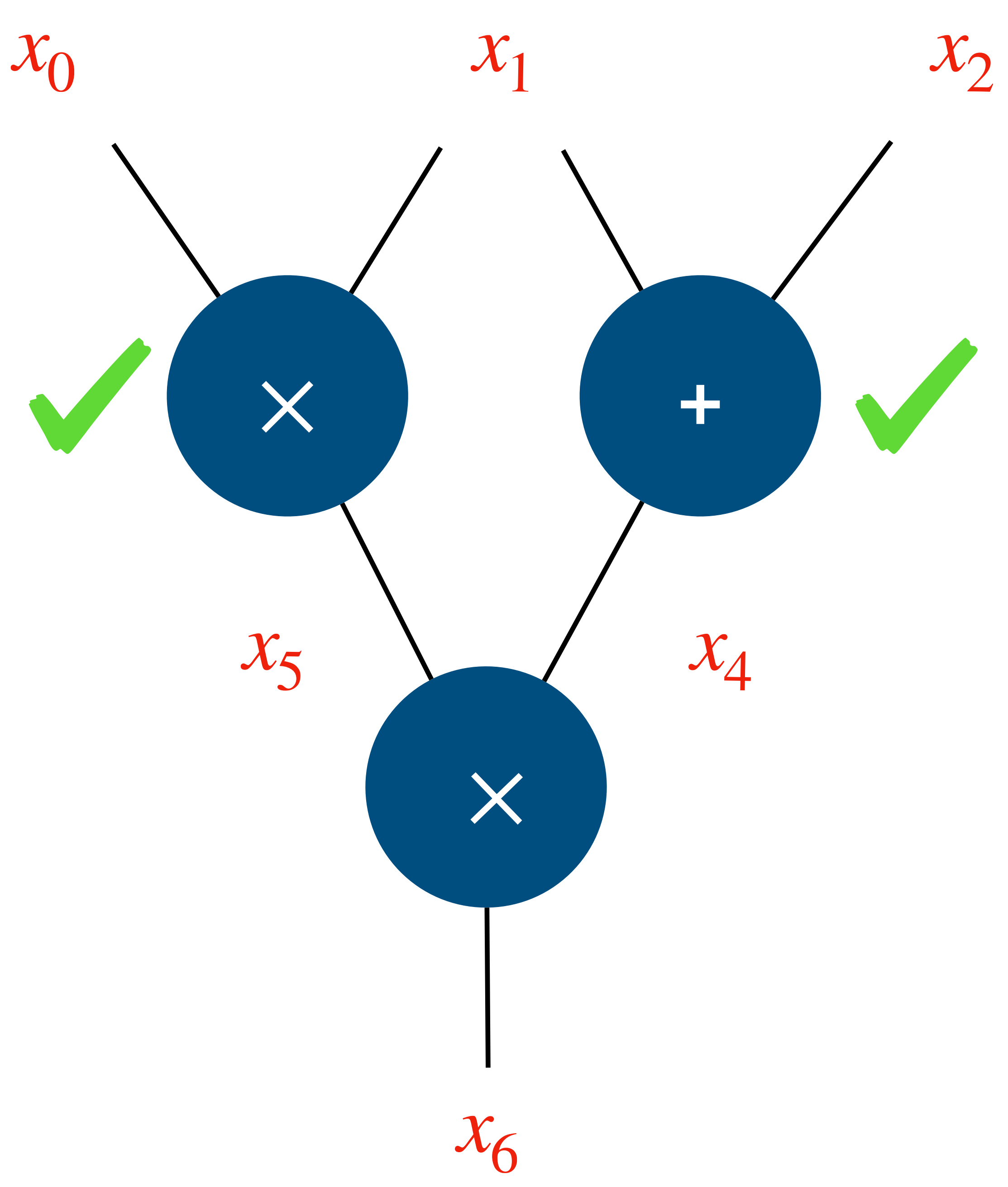
Zero Knowledge Proof



- Authenticated value: ciphertext that
  - Hide underlying value
  - Enable verifier to check the relations
  - Prevent prover from cheating about the underlying value
- Gate-by-gate paradigm [Beaver et al. 90]:
  - Prover authenticates the value over all wires
  - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
  - Prover reveals the value of output of the circuit

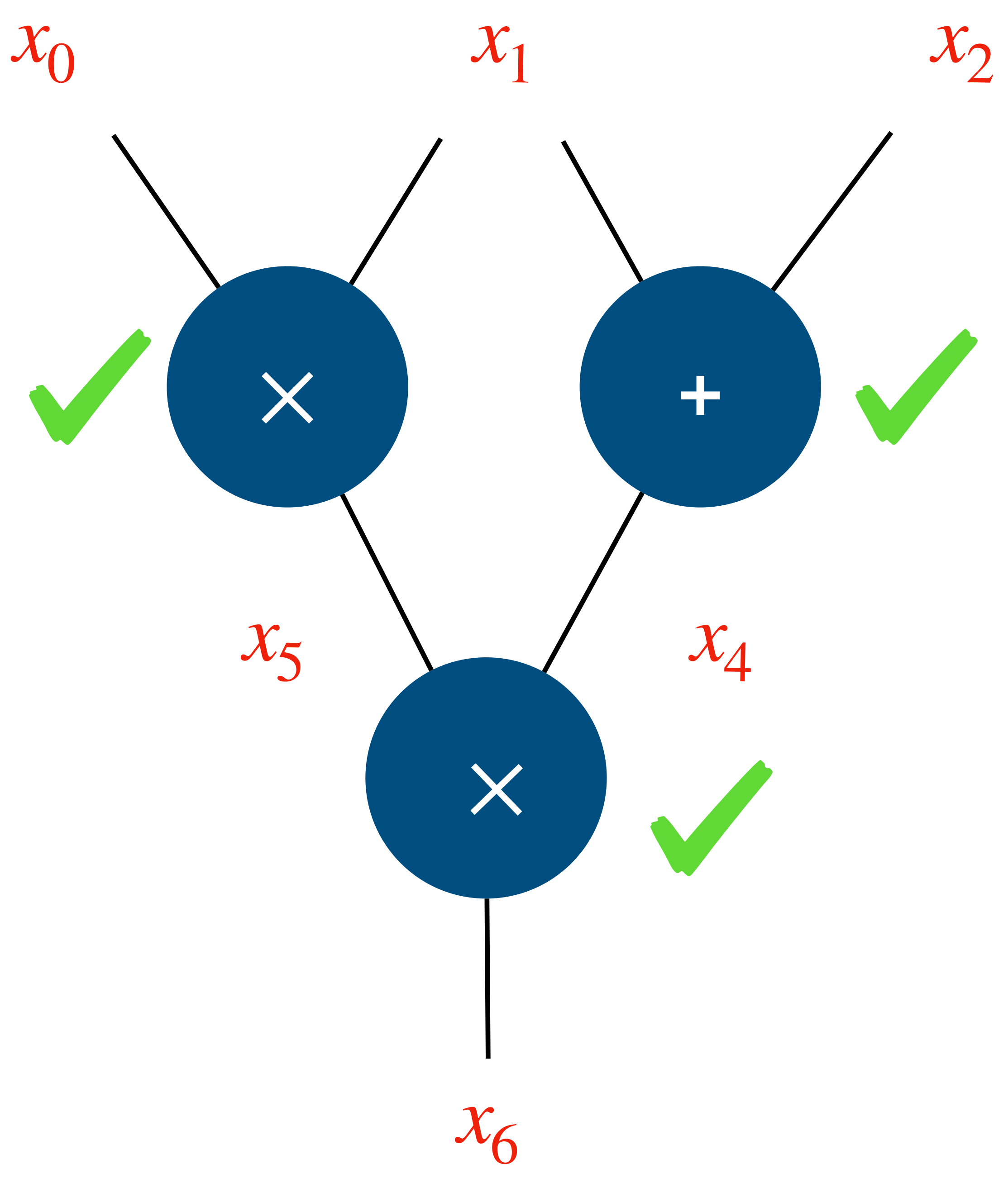
Zero Knowledge Proof

- Authenticated value: ciphertext that
    - Hide underlying value
    - Enable verifier to check the relations
    - Prevent prover from cheating about the underlying value
- Gate-by-gate paradigm [Beaver et al. 90]:
    - Prover authenticates the value over all wires
    - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
    - Prover reveals the value of output of the circuit



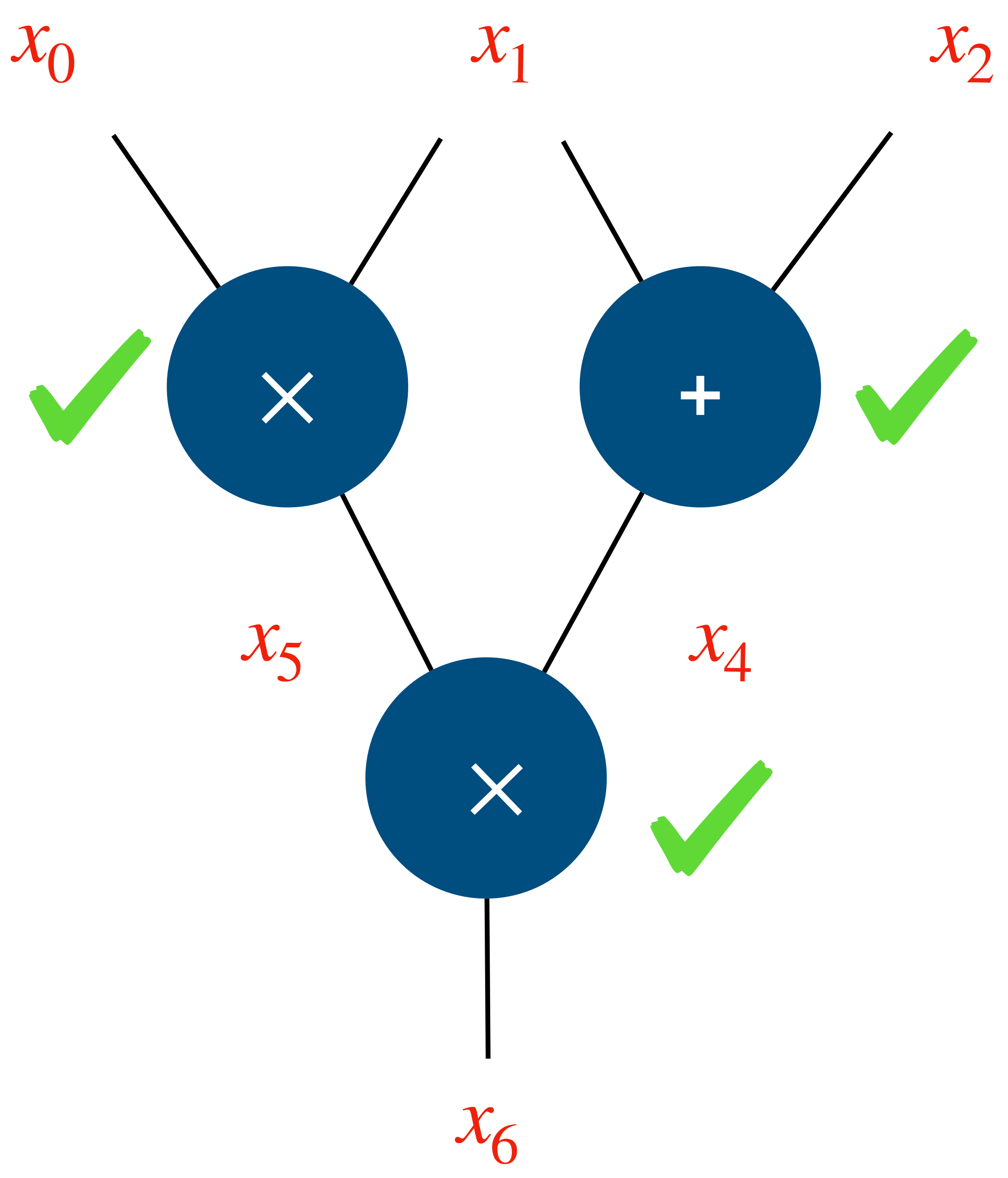
Zero Knowledge Proof

- Authenticated value: ciphertext that
  - Hide underlying value
  - Enable verifier to check the relations
  - Prevent prover from cheating about the underlying value
- Gate-by-gate paradigm [Beaver et al. 90]:
  - Prover authenticates the value over all wires
  - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
  - Prover reveals the value of output of the circuit



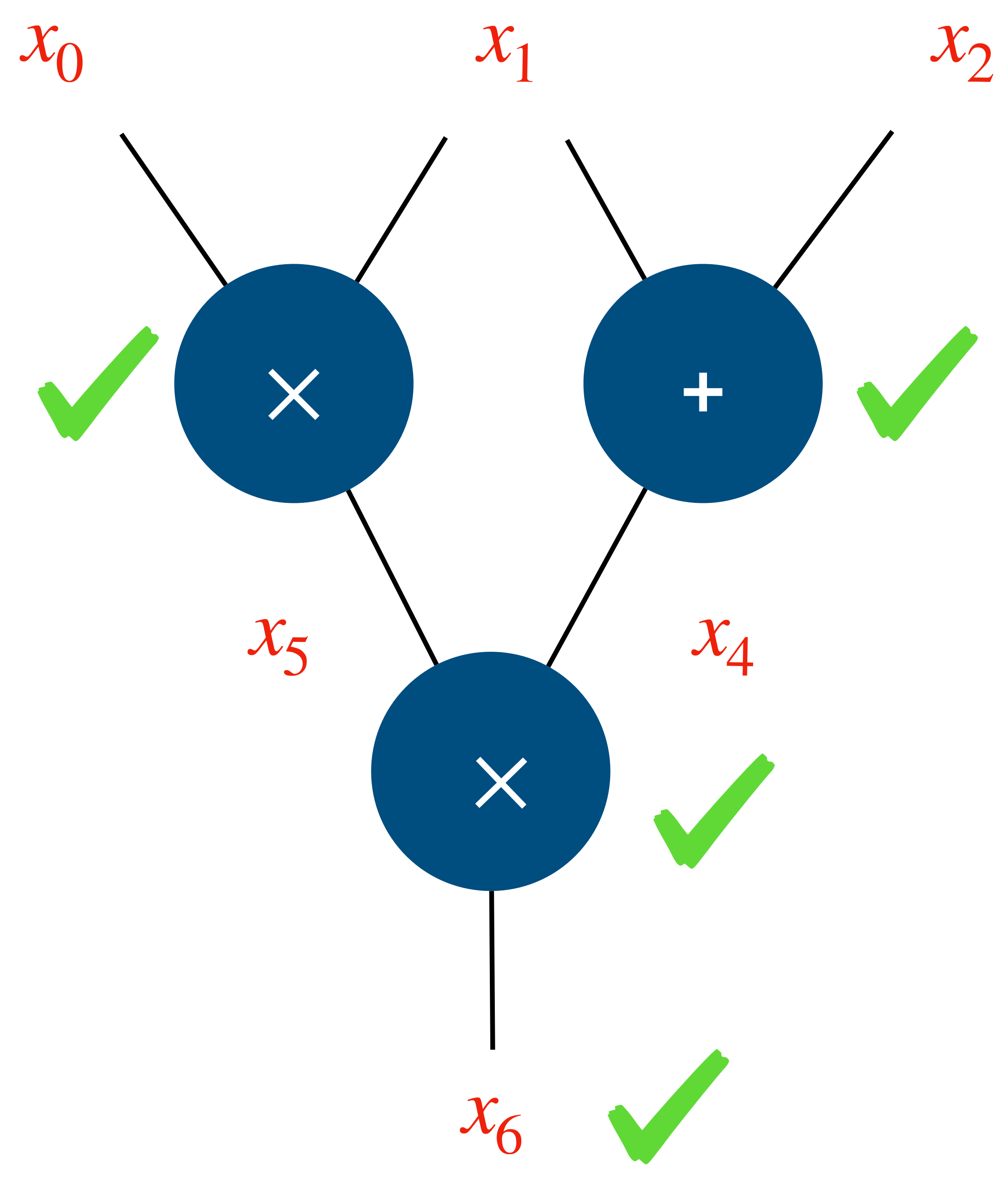
Zero Knowledge Proof

- Authenticated value: ciphertext that
  - Hide underlying value
  - Enable verifier to check the relations
  - Prevent prover from cheating about the underlying value
- Gate-by-gate paradigm [Beaver et al. 90]:
  - Prover authenticates the value over all wires
  - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
  - Prover reveals the value of output of the circuit



Zero Knowledge Proof

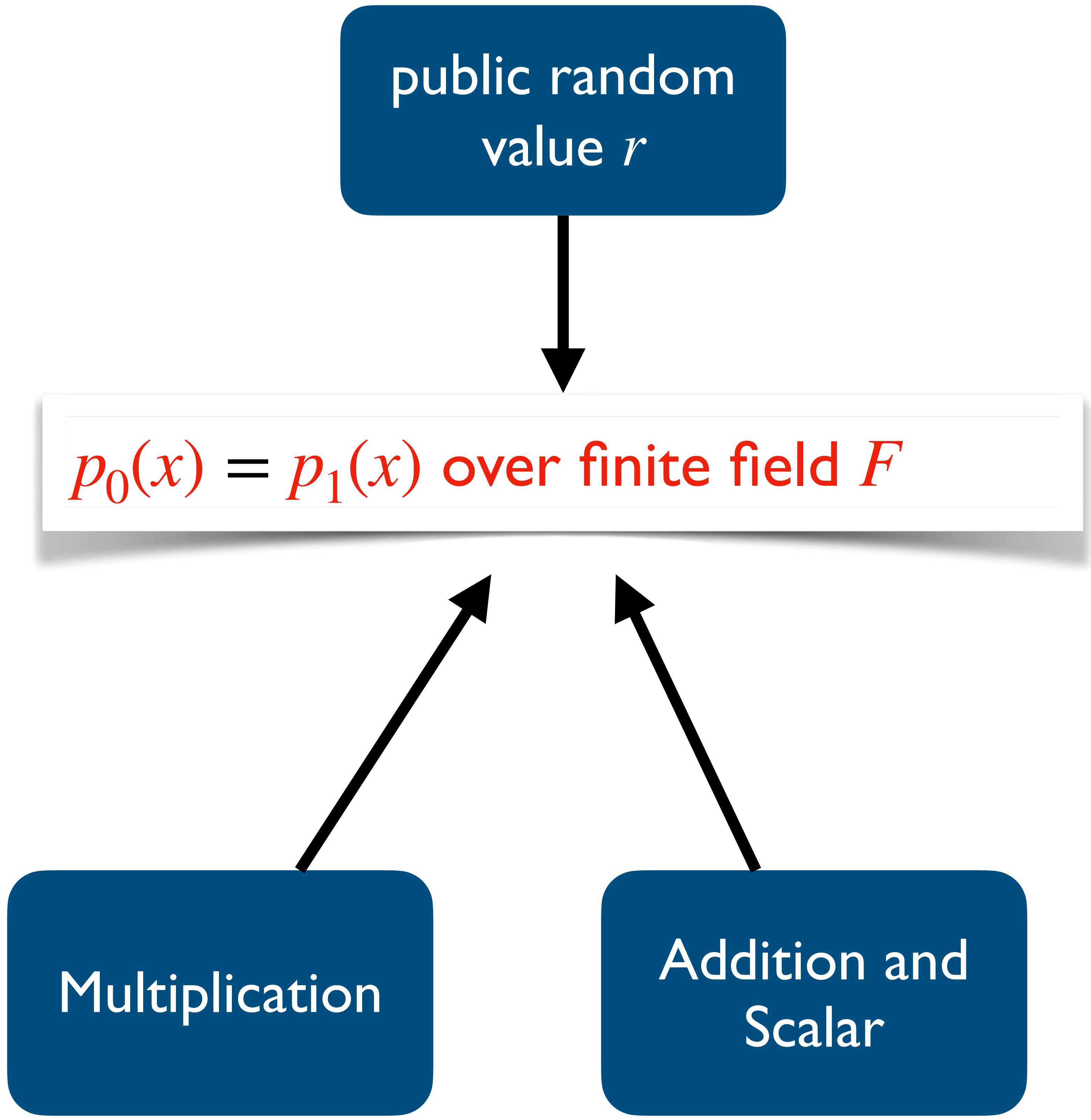
- Authenticated value: ciphertext that
    - Hide underlying value
    - Enable verifier to check the relations
    - Prevent prover from cheating about the underlying value



- Gate-by-gate paradigm [Beaver et al. 90]:
    - Prover authenticates the value over all wires
    - Verifier checks if input and output of each gate is consistent over the ***ciphertext***
    - Prover reveals the value of output of the circuit

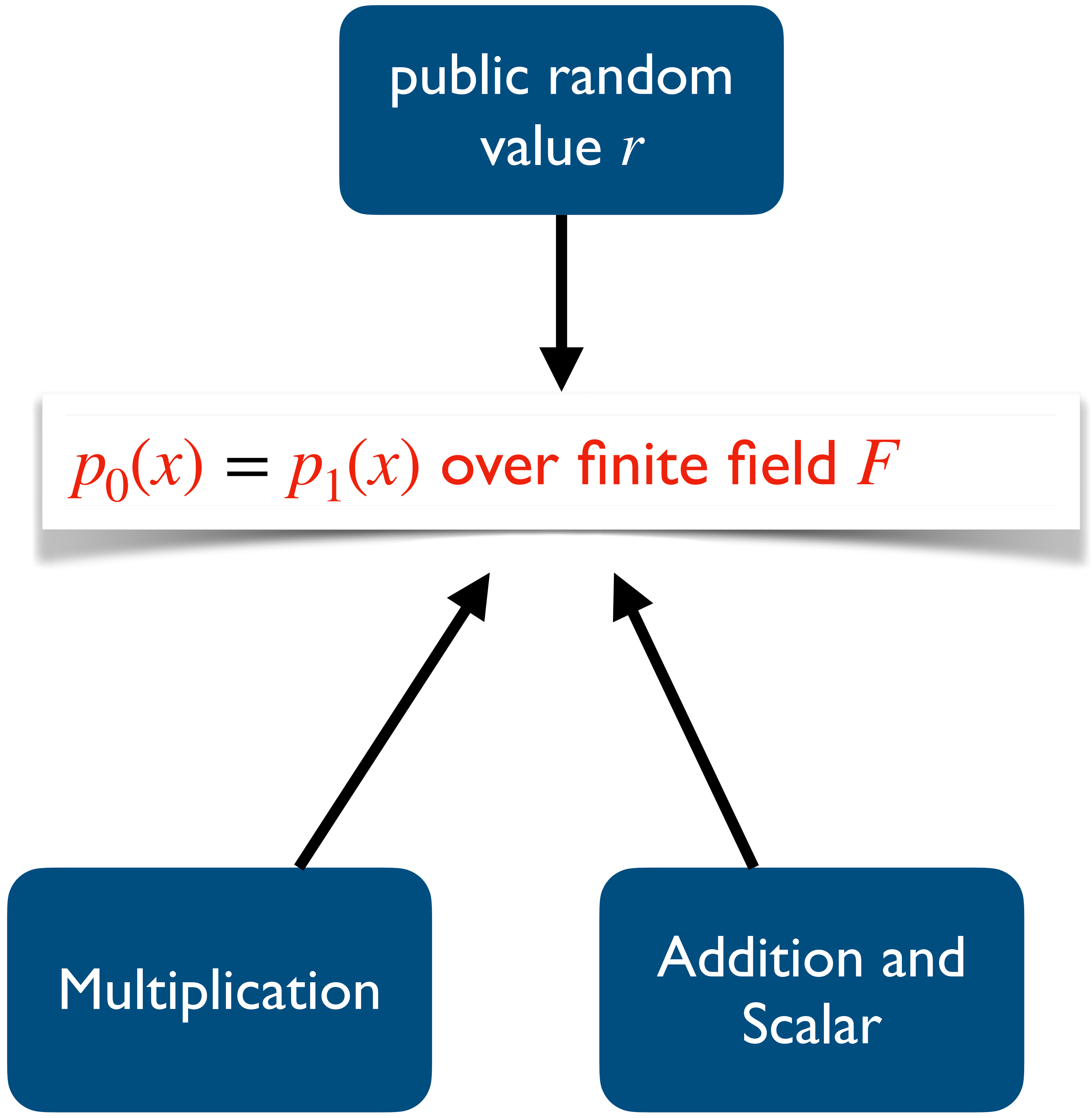
Zero Knowledge Proof

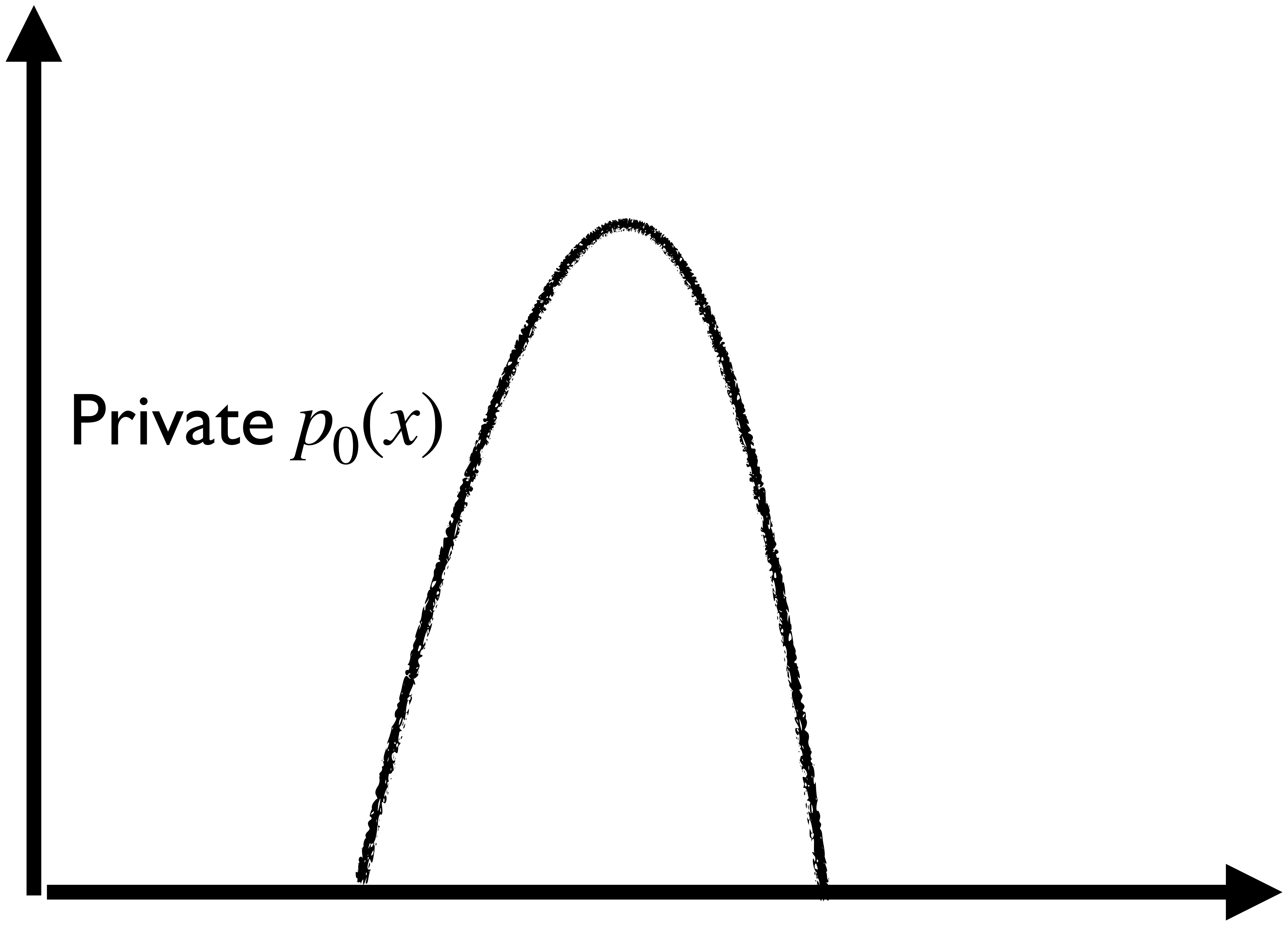
Knowledge for Polynomials[Yang et al. 21]



Zero Knowledge Proof

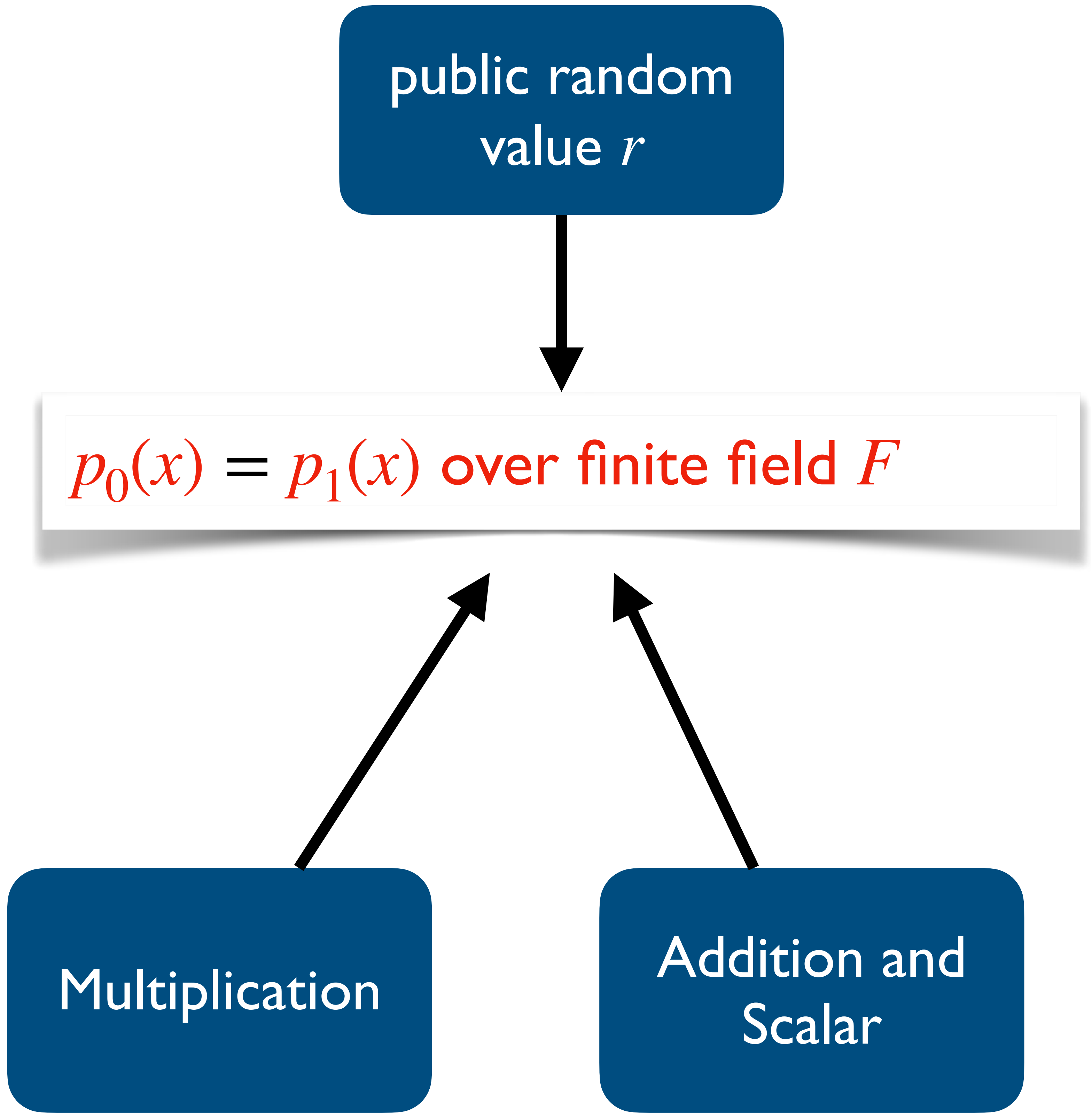
Knowledge for Polynomials[Yang et al. 21]

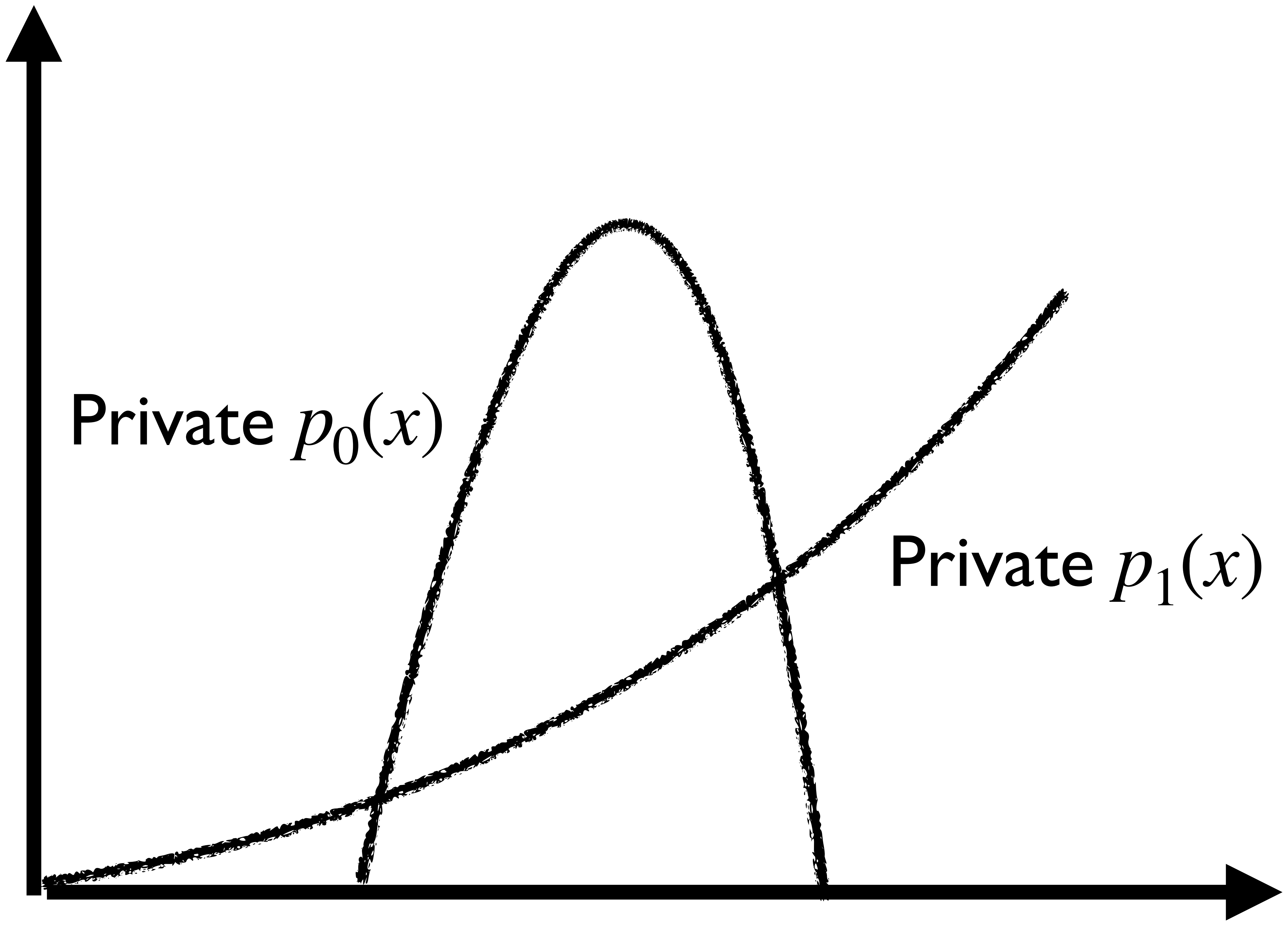




Zero Knowledge Proof

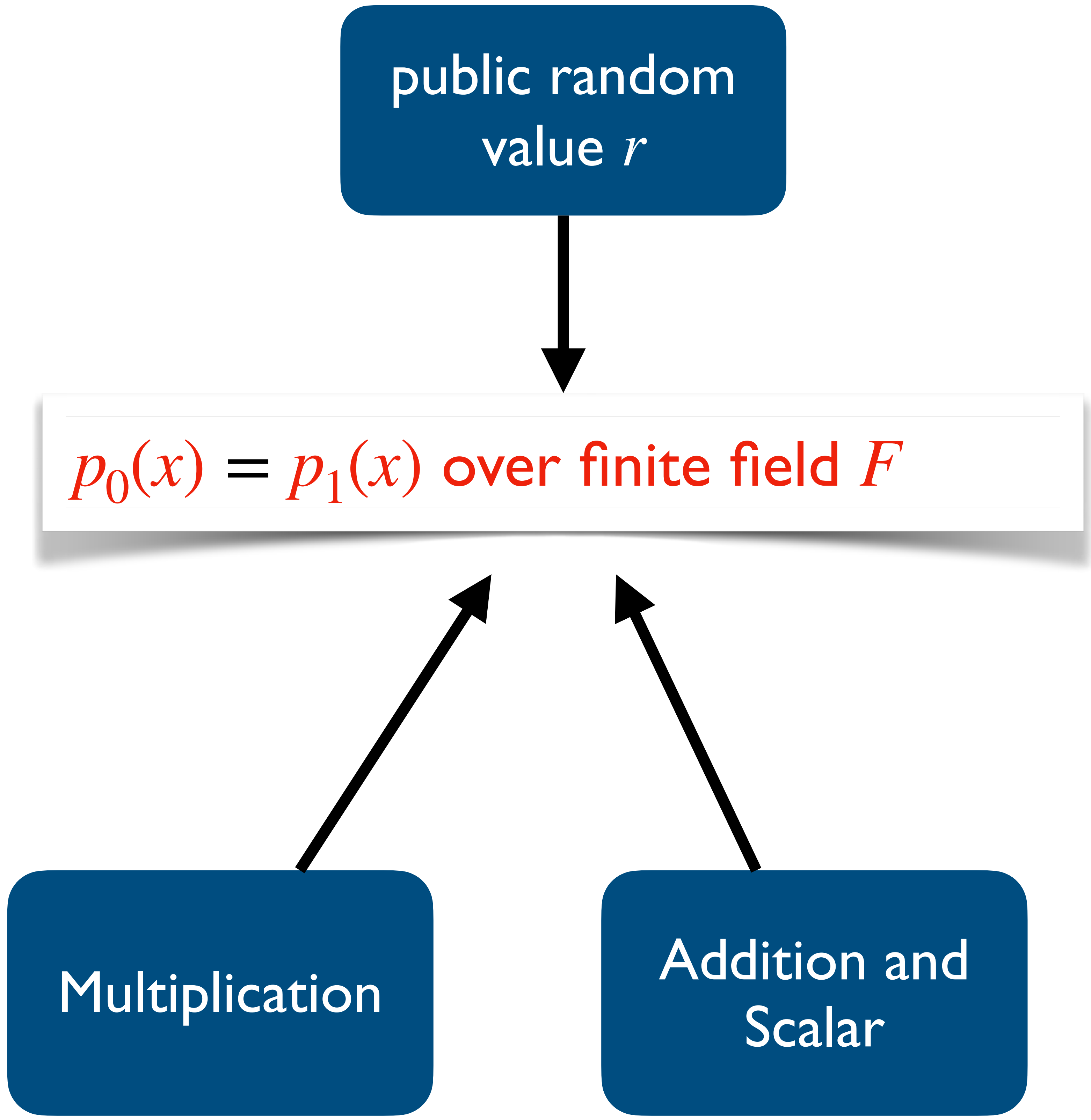
Knowledge for Polynomials[Yang et al. 21]

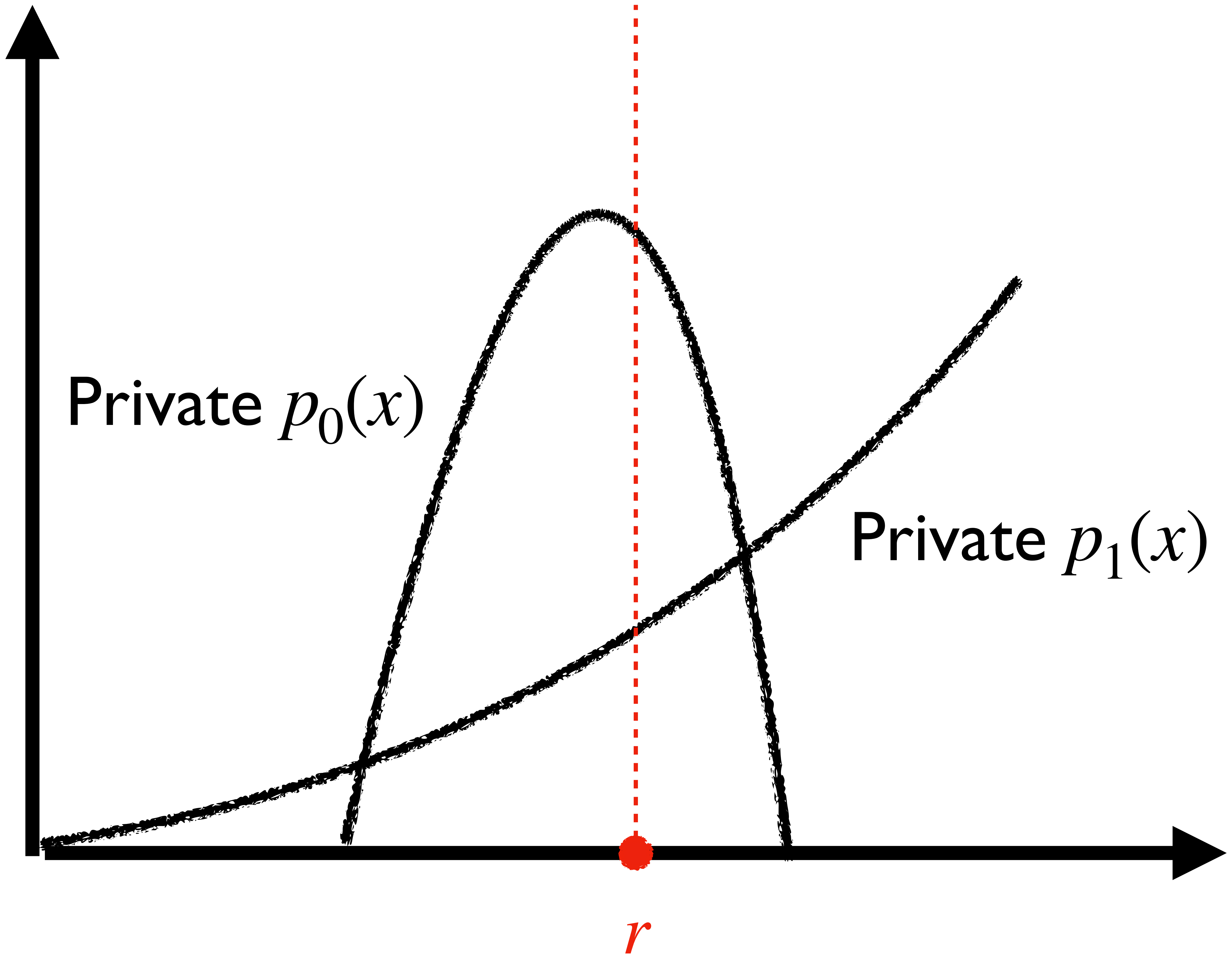




Zero Knowledge Proof

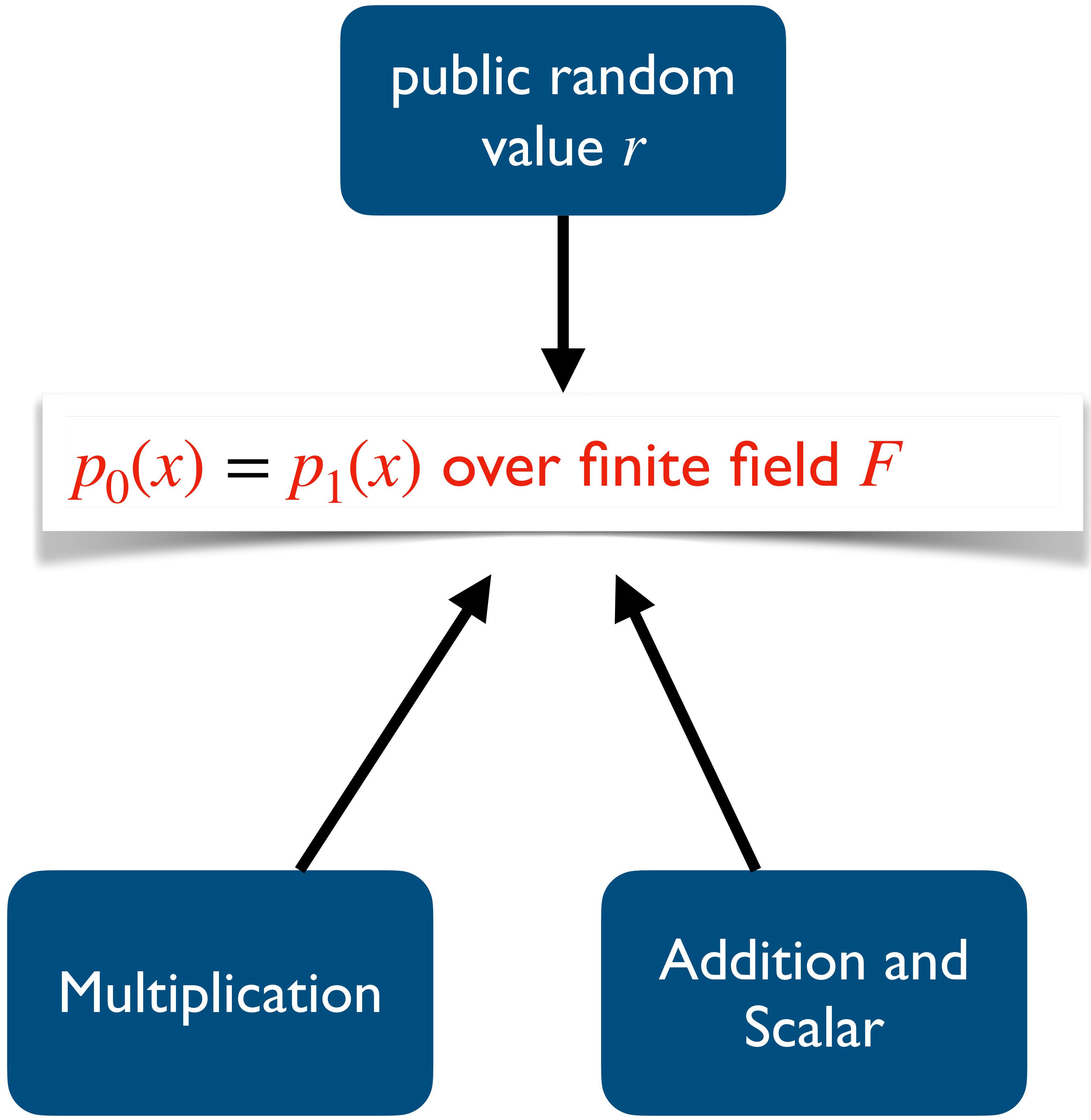
Knowledge for Polynomials[Yang et al. 21]

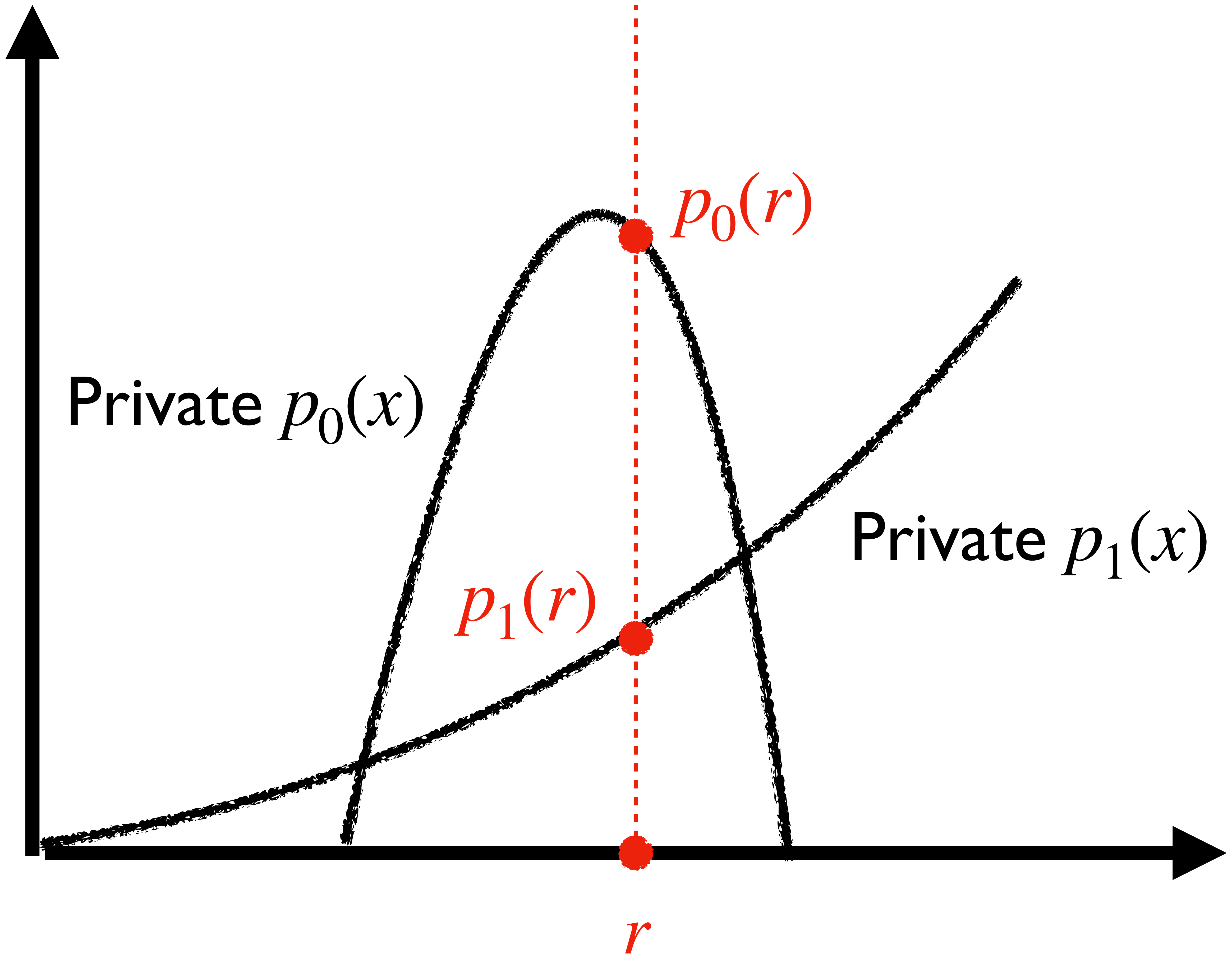




Zero Knowledge Proof

Knowledge for Polynomials[Yang et al. 21]





Zero Knowledge Proof

Knowledge for Polynomials[Yang et al. 21]

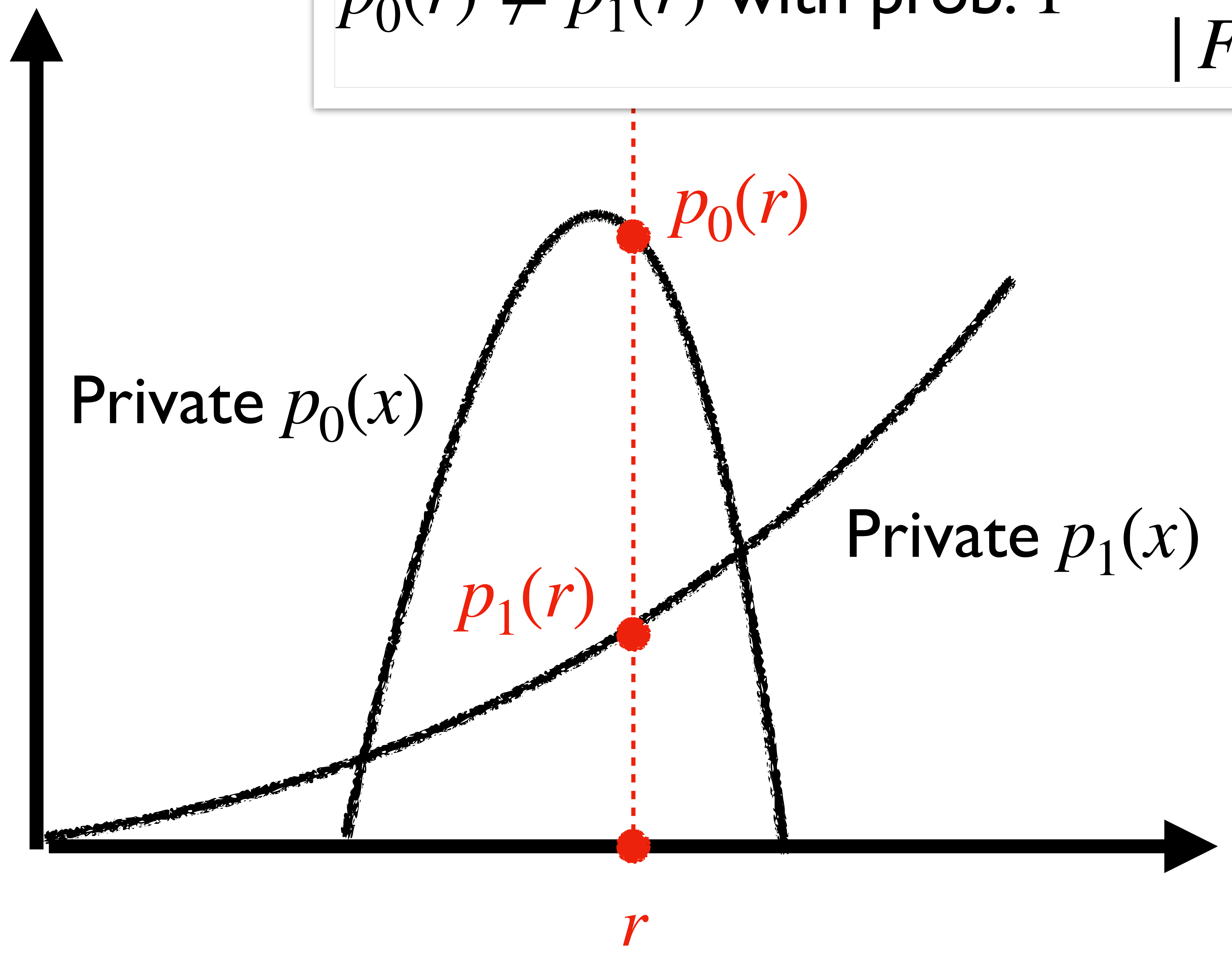
$$p_0(r) \neq p_1(r) \text{ with prob. } 1 - \frac{d}{|F|}$$

public random value r

$p_0(x) = p_1(x)$ over finite field F

Multiplication

Addition and Scalar



Zero Knowledge Proof

Knowledge for Polynomials[Yang et al. 21]

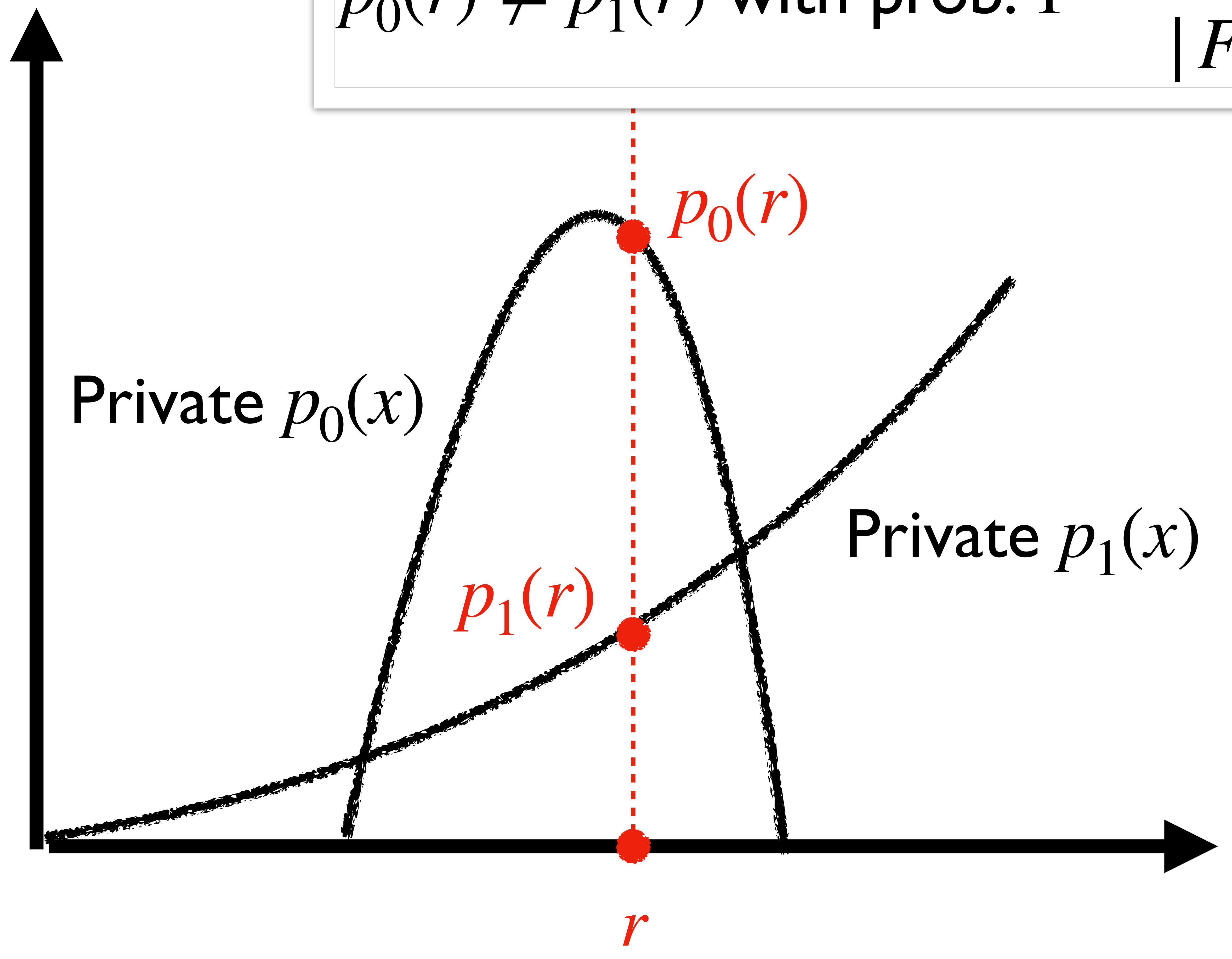
$$p_0(r) \neq p_1(r) \text{ with prob. } 1 - \frac{d}{|F|}$$

public random value r

- $p_1(x) = p_2(x)$

Multiplication

Addition and Scalar



Zero Knowledge Proof

Knowledge for Polynomials[Yang et al. 21]

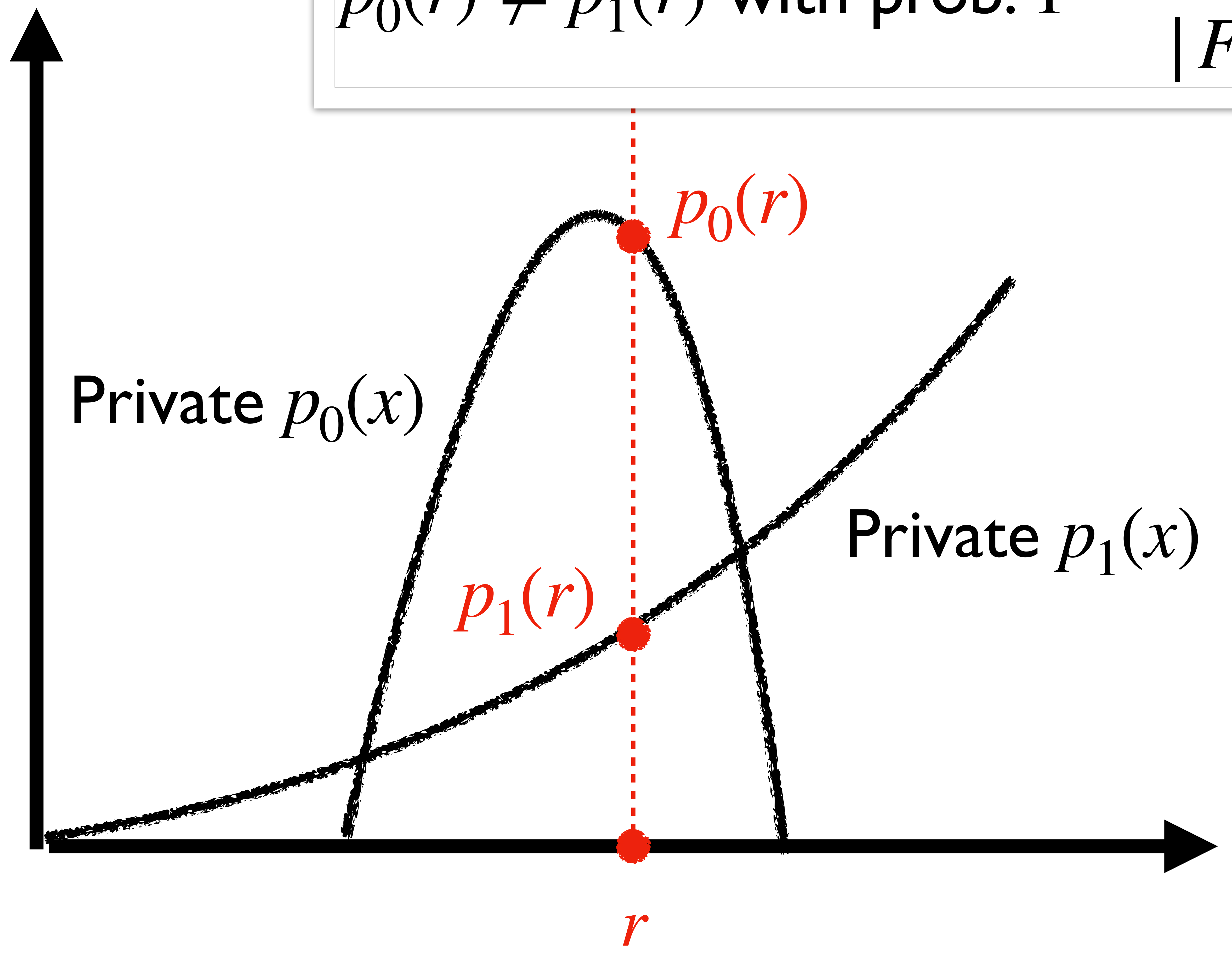
$p_0(r) \neq p_1(r)$ with prob. $1 - \frac{d}{|F|}$

public random value r

- $p_1(x) = p_2(x)$
- $p_0(x) \cdot p_1(x) = p_2(x)$

Multiplication

Addition and Scalar



Zero Knowledge Proof

Knowledge for Polynomials[Yang et al. 21]

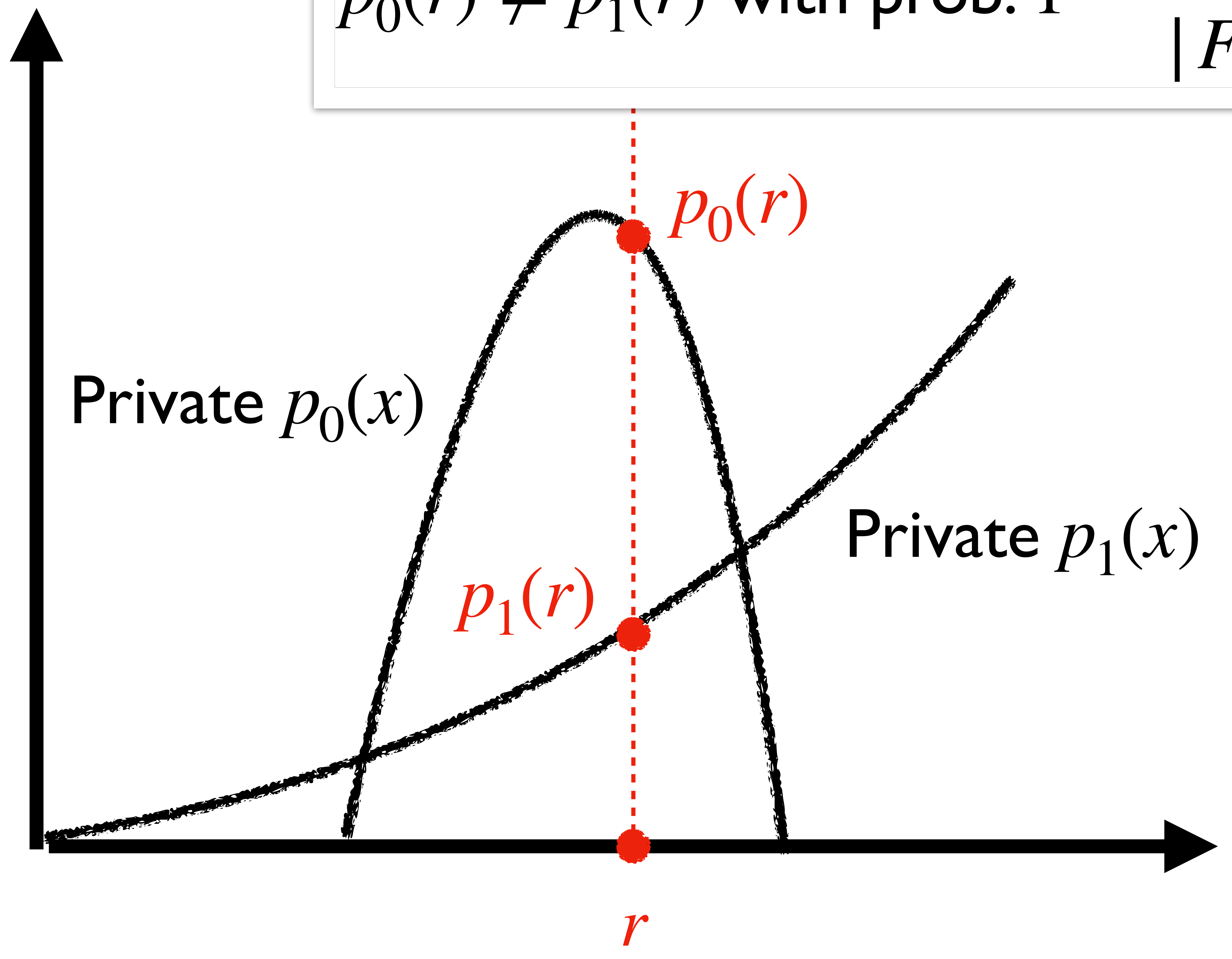
$$p_0(r) \neq p_1(r) \text{ with prob. } 1 - \frac{d}{|F|}$$

public random value r

- $p_1(x) = p_2(x)$
- $p_0(x) \cdot p_1(x) = p_2(x)$
- $p_0(x) \cdot p_1(x) + p_0'(x) \cdot p_1'(x) = p(x)$

Multiplication

Addition and Scalar



Zero Knowledge Proof

Knowledge for Polynomials[Yang et al. 21]

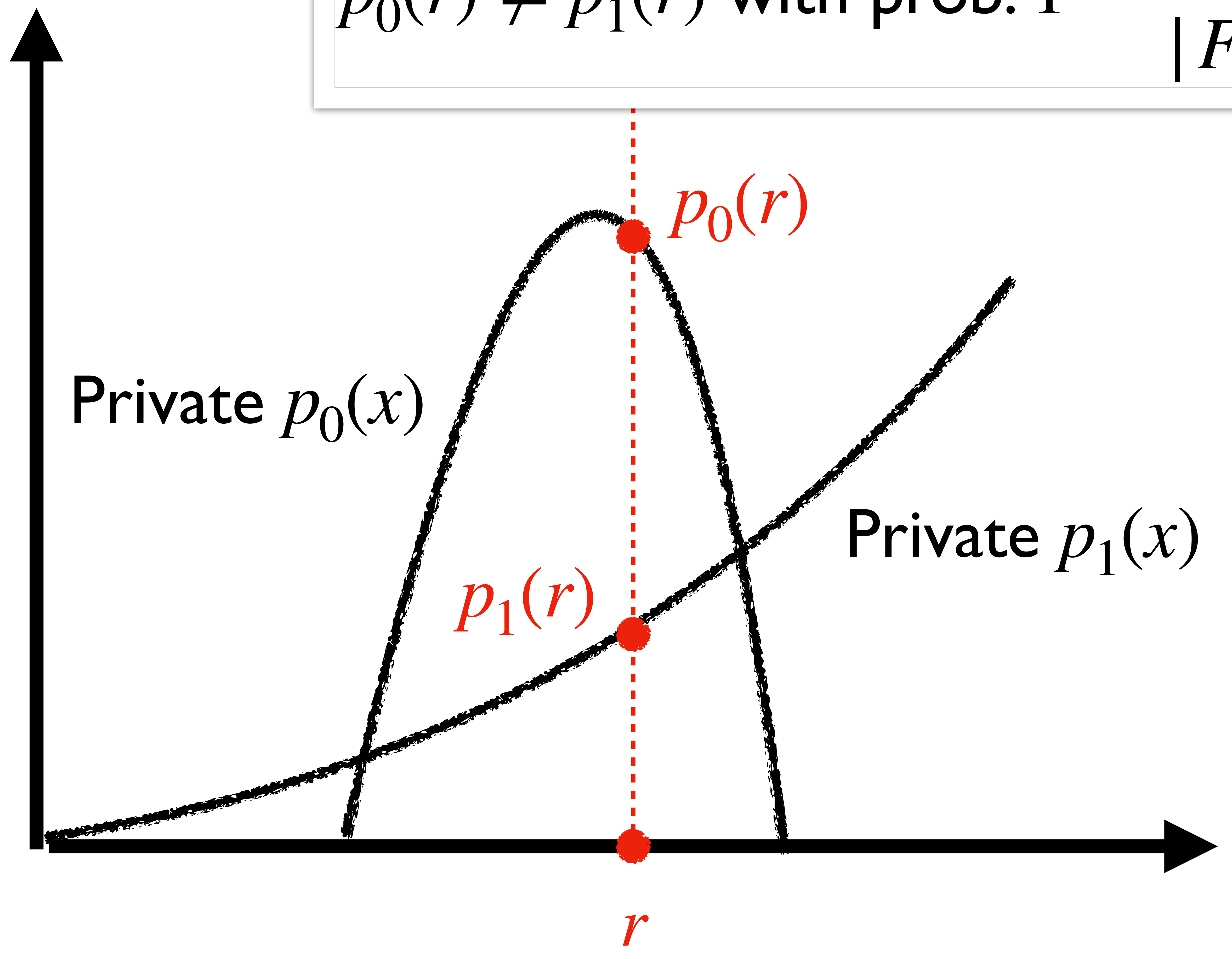
$$p_0(r) \neq p_1(r) \text{ with prob. } 1 - \frac{d}{|F|}$$

public random value r

- $p_1(x) = p_2(x)$
- $p_0(x) \cdot p_1(x) = p_2(x)$
- $p_0(x) \cdot p_1(x) + p_0'(x) \cdot p_1'(x) = p(x)$
- $p_1(x) = p_0(c + x)$

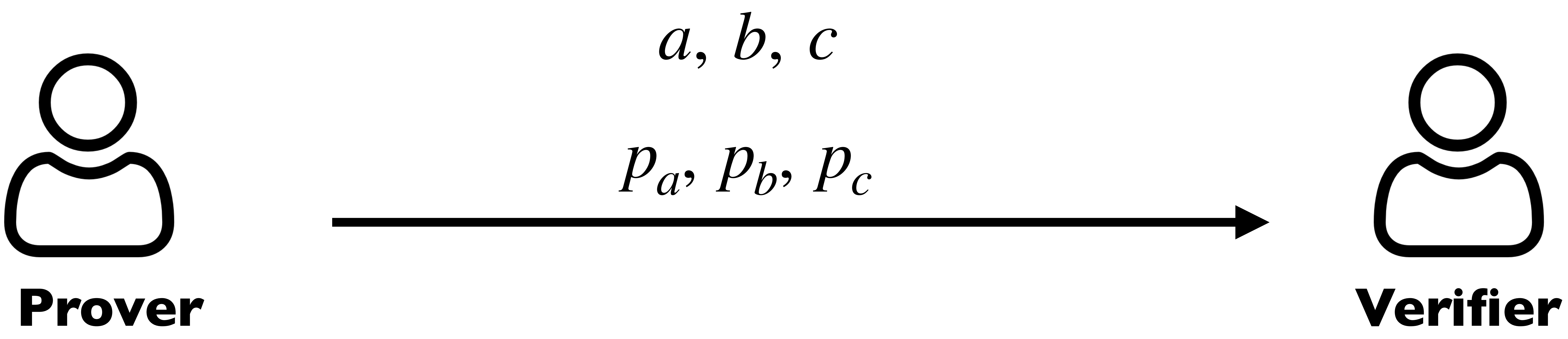
Multiplication

Addition and Scalar



Zero Knowledge Proof: Take Away

Verifier can **verify**
the **relations** between
private values or polynomials
without learning the values themselves



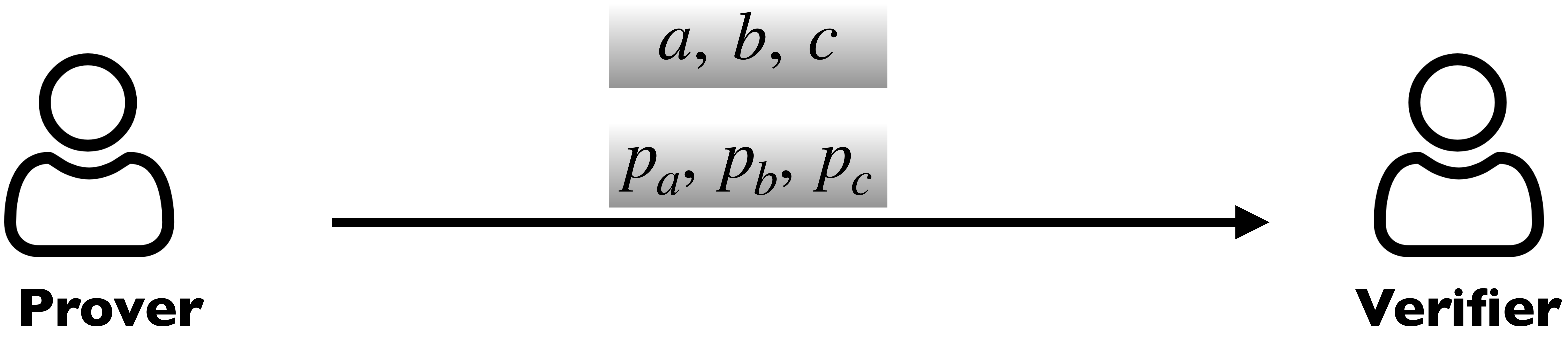
Zero Knowledge Proof: Take Away

Verifier can **verify**

the **relations** between

private values or polynomials

without learning the values themselves



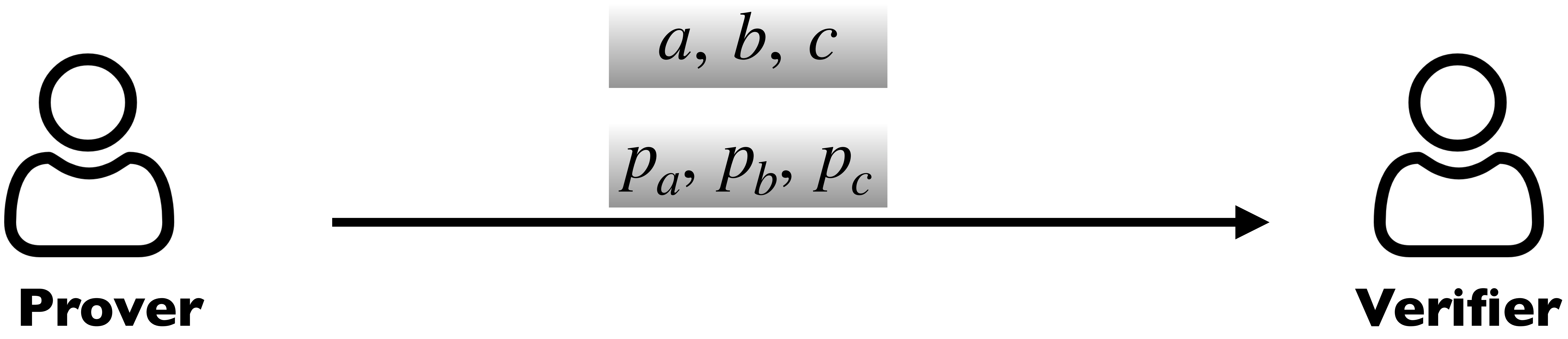
Zero Knowledge Proof：Take Away

Verifier can **verify**

the **relations** between

private values on $\boxed{\text{polynomials}}$

without learning the values themselves



Unsatisfiability in Zero Knowledge Proof

Unsatisfiability in Zero Knowledge Proof

- Prover knows a resolution proof Prf for a formula ϕ

Unsatisfiability in Zero Knowledge Proof

- Prover knows a resolution proof Prf for a formula ϕ

$\phi = c_0 \land c_1 \land c_2 \land c_3$

Resolution Proof

Unsatisfiability in Zero Knowledge Proof

- Prover knows a resolution proof Prf for a formula ϕ
  - Convince the verifier that ϕ is **unsatisfiable**

Unsatisfiability in Zero Knowledge Proof

- Prover knows a resolution proof Prf for a formula ϕ
  - Convince the verifier that ϕ is **unsatisfiable**
  - Keep information about Prf and ϕ **private**

Unsatisfiability in Zero Knowledge Proof

- Prover knows a resolution proof Prf for a formula ϕ
    - Convince the verifier that ϕ is **unsatisfiable**
    - Keep information about Prf and ϕ **private**

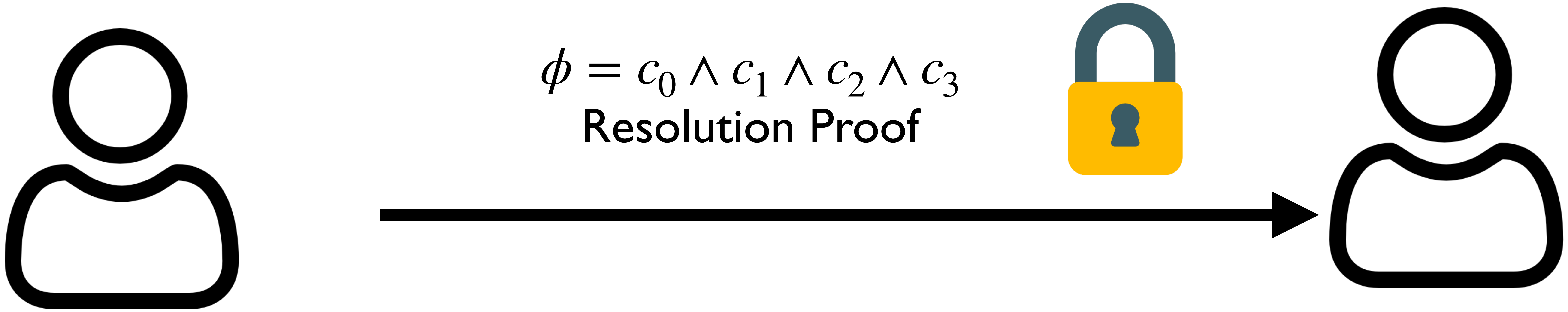
Unsatisfiability in Zero Knowledge Proof

- Prover knows a resolution proof Prf for a formula ϕ
    - Convince the verifier that ϕ is **unsatisfiable**
    - Keep information about Prf and ϕ **private**
- Verifier:

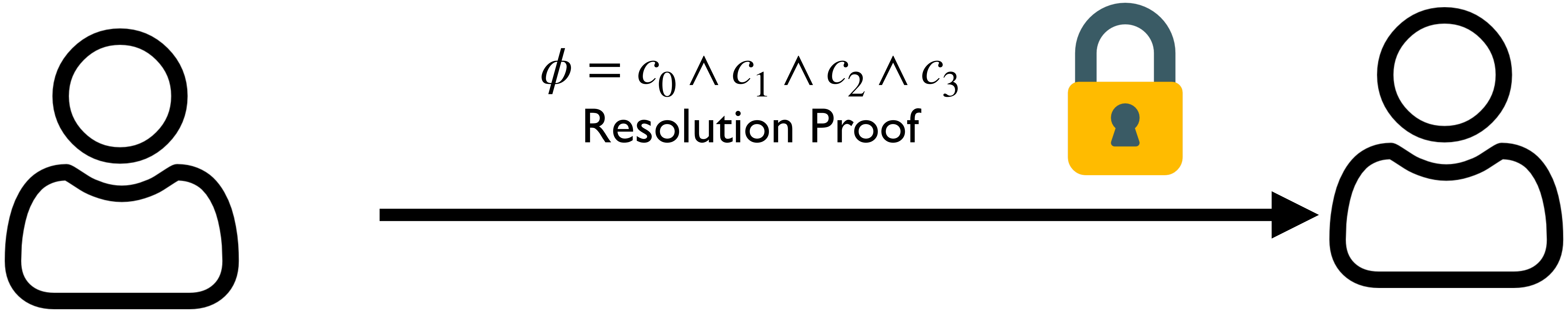
Unsatisfiability in Zero Knowledge Proof

- Prover knows a resolution proof Prf for a formula ϕ
    - Convince the verifier that ϕ is **unsatisfiable**
    - Keep information about Prf and ϕ **private**
- Verifier:
    - Validate prover's claim about ϕ

Unsatisfiability in Zero Knowledge Proof

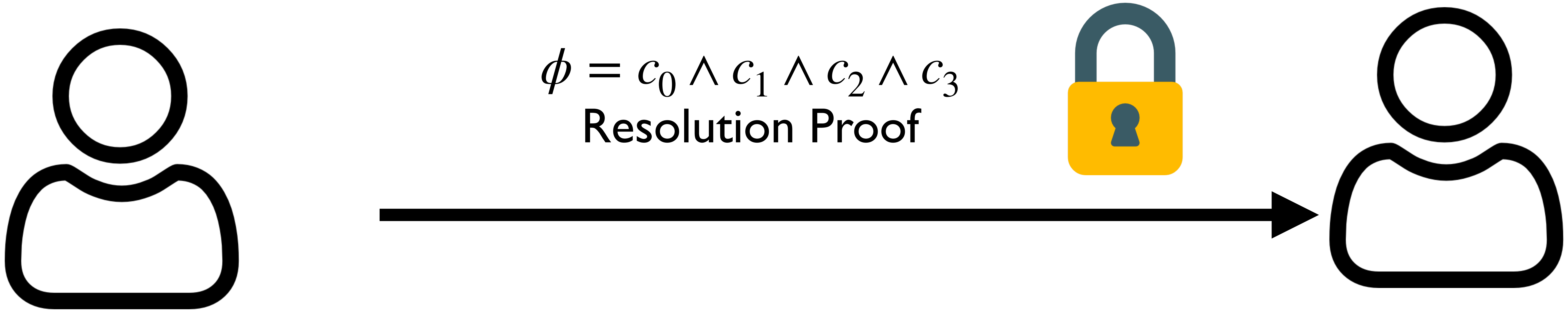


Unsatisfiability in Zero Knowledge Proof



Both Prover and Verifier can be **malicious**.

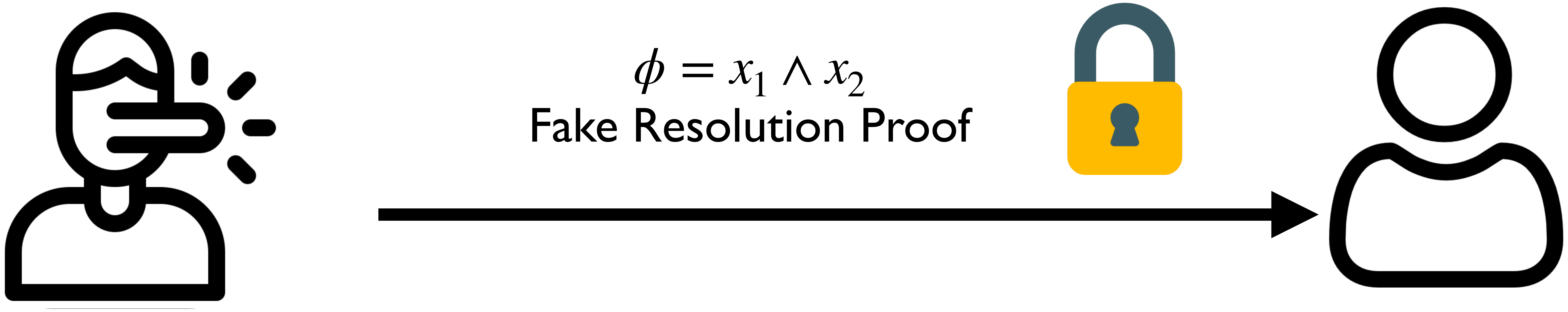
Unsatisfiability in Zero Knowledge Proof



Both Prover and Verifier can be **malicious**.

- Prover might cheat about unsatisfiability of ϕ

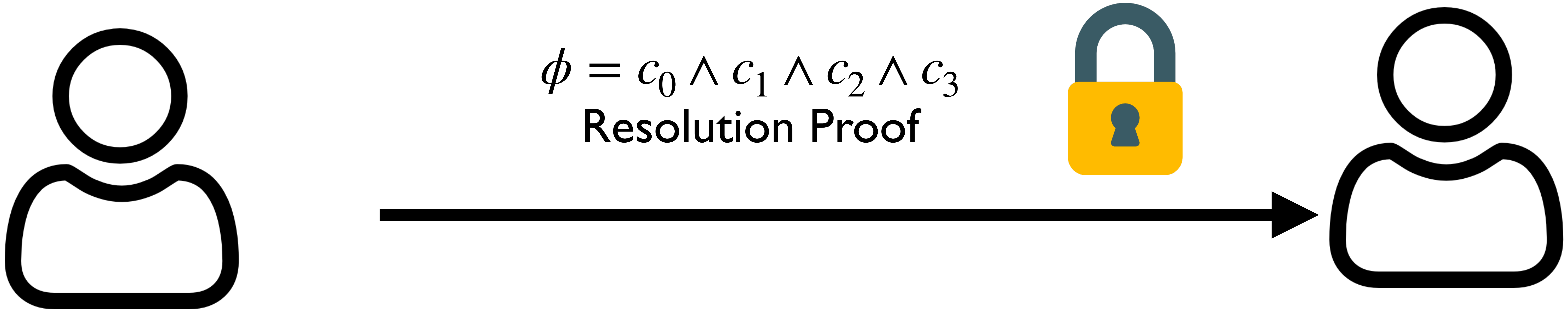
Unsatisfiability in Zero Knowledge Proof



Both Prover and Verifier can be **malicious**.

- Prover might cheat about unsatisfiability of ϕ

Unsatisfiability in Zero Knowledge Proof



Both Prover and Verifier can be **malicious**.

- Prover might cheat about unsatisfiability of ϕ

Unsatisfiability in Zero Knowledge Proof

Both Prover and Verifier can be **malicious**.

- Prover might cheat about unsatisfiability of ϕ

Unsatisfiability in Zero Knowledge Proof

Both Prover and Verifier can be **malicious**.

- Prover might cheat about unsatisfiability of ϕ
- Verifier tries to learn information about Prf and ϕ

Unsatisfiability in Zero Knowledge Proof

Technique challenges and design overview

ZKUNSAT

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Unsatisfiability in Zero Knowledge Proof

Technique challenges and design overview

ZKUNSAT

Fetch clauses

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Unsatisfiability in Zero Knowledge Proof

Technique challenges and design overview

ZKUNSAT

Fetch clauses

Check application of resolution rule

$c_0 : (x_1 \vee x_2)$	$-,-$
$c_1 : (\neg x_1 \vee x_2)$	$-,-$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-,-$
$c_3 : (x_1 \vee \neg x_2)$	$-,-$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Unsatisfiability in Zero Knowledge Proof

Technique challenges and design overview

ZKUNSAT

Fetch clauses

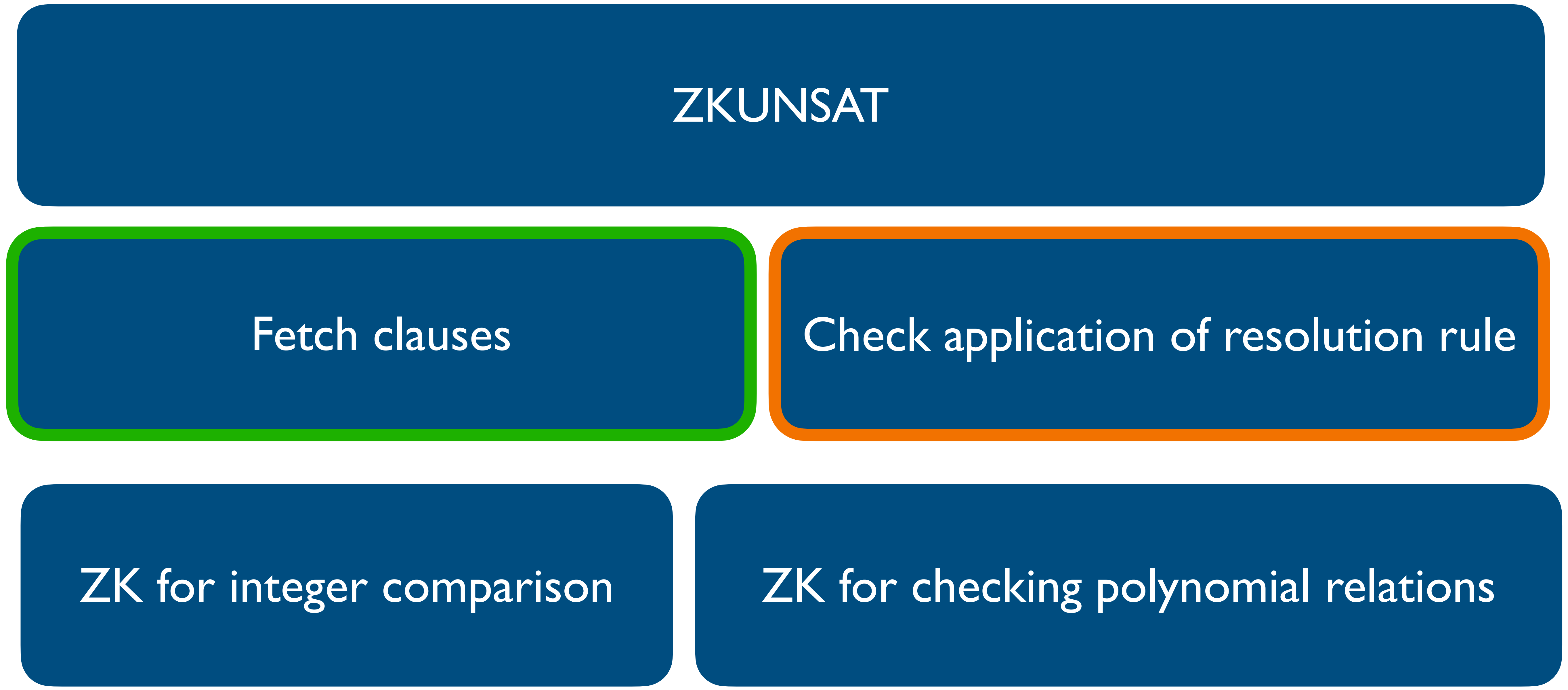
Check application of resolution rule

ZK for integer comparison

$c_0 : (x_1 \vee x_2)$	$-,-$
$c_1 : (\neg x_1 \vee x_2)$	$-,-$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-,-$
$c_3 : (x_1 \vee \neg x_2)$	$-,-$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \bot$	$4, 5$

Unsatisfiability in Zero Knowledge Proof

Technique challenges and design overview



$c_0 : (x_1 \lor x_2)$	$-, -$
$c_1 : (\neg x_1 \lor x_2)$	$-, -$
$c_2 : (\neg x_1 \lor \neg x_2)$	$-, -$
$c_3 : (x_1 \lor \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Check application of resolution rule

Polynomial-based approach

Clause

$\ell_1 \vee \cdots \vee \ell_k$

Resolution

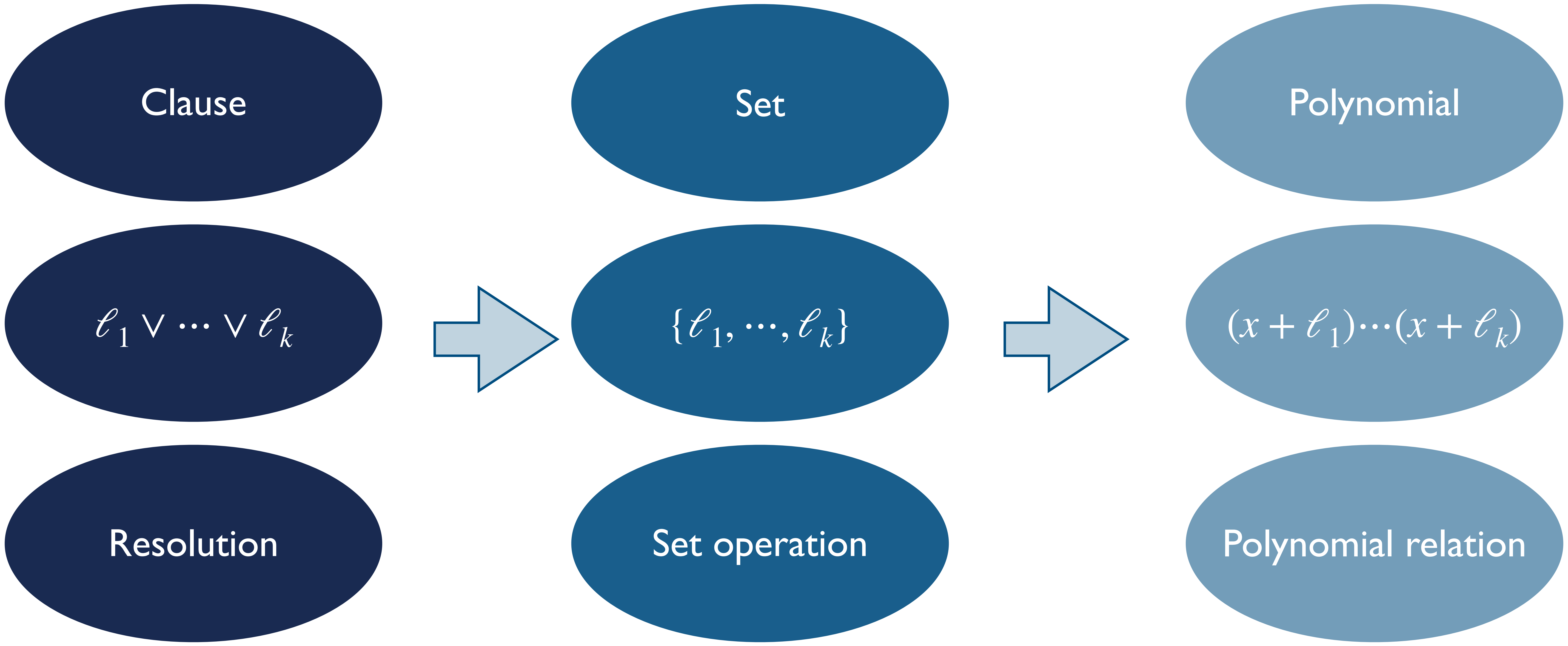
Check application of resolution rule

Polynomial-based approach

Clause		Set
$\ell_1 \vee \cdots \vee \ell_k$	$\Rightarrow$	$\{\ell_1, \cdots, \ell_k\}$
Resolution		Set operation

Check application of resolution rule

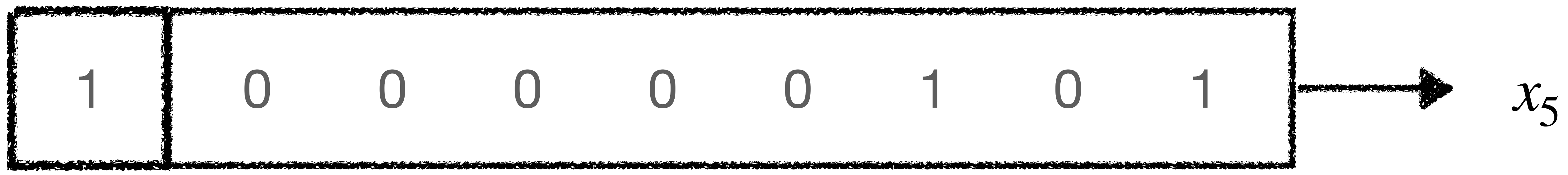
Polynomial-based approach



Check application of resolution rule

Polynomial-based approach

- An Encoding scheme ϵ from literals $\mathscr{L}$ to finite field $\mathbf{F}_{2^k} \backslash \mathbf{0}$

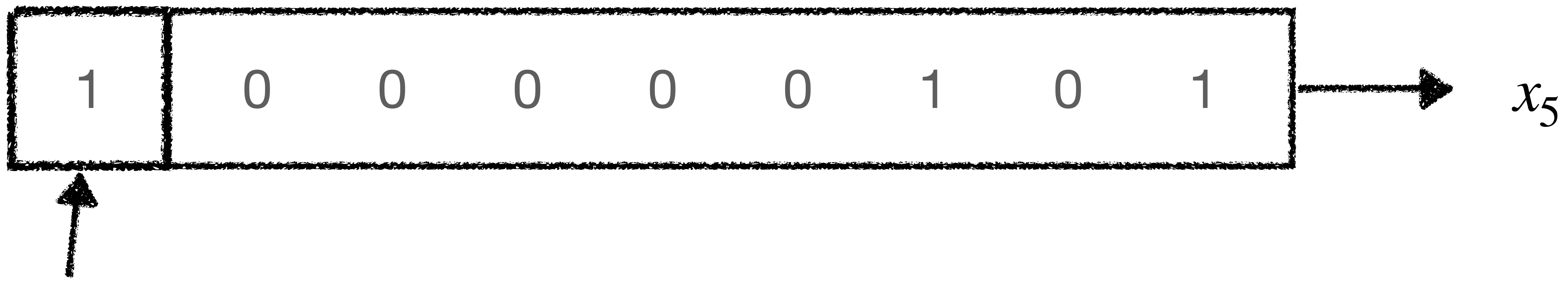


- Any Encoding scheme that satisfies
  - Injective
  - $\epsilon(\ell) + \epsilon(\neg \ell) = c, \ c$ is a public constant

Check application of resolution rule

Polynomial-based approach

- An Encoding scheme ϵ from literals $\mathscr{L}$ to finite field $\mathbf{F}_{2^k}\backslash\mathbf{0}$



- Any Encoding scheme that satisfies
  - Injective
  - $\epsilon(\ell) + \epsilon(\neg \ell) = c, \ c$ is a public constant

Check application of resolution rule

Polynomial-based approach

- An Encoding scheme ϵ from literals $\mathscr{L}$ to finite field $\mathbf{F}_{2^k} \backslash 0$

- Any Encoding scheme that satisfies
  - Injective
  - $\epsilon(\ell) + \epsilon(\neg \ell) = c, \;\; c$ is a public constant

Check application of resolution rule

Polynomial-based approach

- An Encoding scheme ϵ from literals $\mathscr{L}$ to finite field $\mathbf{F}_{2^k}\backslash\mathbf{0}$

- Any Encoding scheme that satisfies
    - Injective
    - $\epsilon(\ell) + \epsilon(\neg\ell) = c,\ c$ is a public constant

Check application of resolution rule

Polynomial-based approach

- An Encoding scheme ϵ from literals $\mathscr{L}$ to finite field $\mathbf{F}_{2^k}\backslash\mathbf{0}$

- Any Encoding scheme that satisfies
  - Injective
  - $\epsilon(\ell) + \epsilon(\neg\ell) = c,\ c$ is a public constant

Check application of resolution rule

Polynomial-based approach

- Clauses are encoded as polynomial over $\mathbf{F}_{2^k}$
  - $c = (\ell_0 \vee \ell_1 \vee \cdots \vee \ell_d) : p_c(x) = (x + \epsilon(\ell_0))\cdots(x + \epsilon(\ell_d))$
- Example

$$c_a = (x_1 \vee x_2) : p_{c_a}(x) = (x+2^{k-1}+1)(x+2^{k-1}+2)$$

$$c_b = (x_1 \vee \neg x_2) : p_{c_b}(x) = (x+2^{k-1}+1)(x+2)$$

Check application of resolution rule

Polynomial-Based Approach

Pivot literal

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

Check application of resolution rule

Polynomial-Based Approach

- Checking one resolution proof step:

Pivot literal

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

Check application of resolution rule

Polynomial-Based Approach

- Checking one resolution proof step:
  - Prover prepares and inputs a pair of pivot polynomials: $(x + \epsilon(\ell_p)), (x + \epsilon(\neg \ell_p))$

Pivot literal

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

Check application of resolution rule

Polynomial-Based Approach

- Checking one resolution proof step:
  - Prover prepares and inputs a pair of pivot polynomials: $(x + \epsilon(\ell_p)), (x + \epsilon(\neg \ell_p))$
    - Pivot polynomials: $(x + 2^{k-1} + 2)$ and $(x + 2)$

Pivot literal

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

Check application of resolution rule

Polynomial-Based Approach

- Checking one resolution proof step:
  - Prover prepares and inputs a pair of pivot polynomials: $(x + \epsilon(\ell_p)), (x + \epsilon(\neg \ell_p))$
    - Pivot polynomials: $(x + 2^{k-1} + 2)$ and $(x + 2)$
  - Prover prepares and inputs polynomial of the resolvent

Pivot literal

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

Check application of resolution rule

Polynomial-Based Approach

- Checking one resolution proof step:
  - Prover prepares and inputs a pair of pivot polynomials: $(x + \epsilon(\ell_p)), (x + \epsilon(\neg \ell_p))$
    - Pivot polynomials: $(x + 2^{k-1} + 2)$ and $(x + 2)$
  - Prover prepares and inputs polynomial of the resolvent
    - $p_{c_r} = (x + 2^{k-1} + 1)$

Pivot literal

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

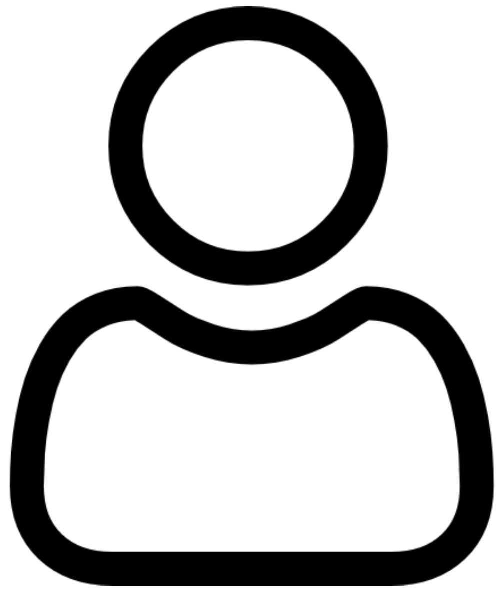
Check application of resolution rule

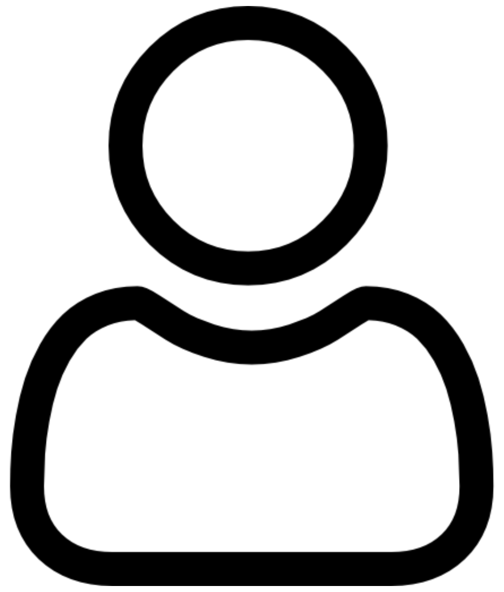
Polynomial-Based Approach

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

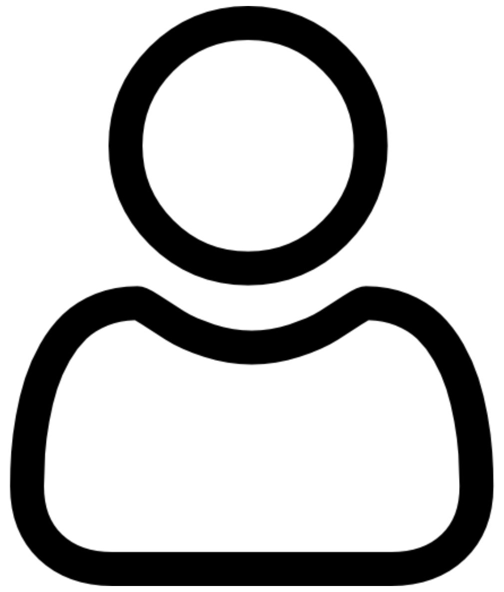
- $C_a \subseteq C_r \cup \{\ell_p\}$, $C_b \subseteq C_r \cup \{\neg \ell_p\}$
  - Proving $p_{c_a} | p_{c_r} \cdot (x + \epsilon(\ell_p))$ and $p_{c_b} | p_{c_r} \cdot (x + \epsilon(\neg \ell_p))$
  - Prover prepares and inputs w_a and w_b
  - Verifier checks
    - $p_{c_a} | p_{c_r} \cdot (x + \epsilon(\ell_p))$ via $p_{c_a} \cdot w_a = p_{c_r} \cdot (x + \epsilon(\ell_p))$
    - $p_{c_b} | p_{c_r} \cdot (x + \epsilon(\neg \ell_p))$ via $p_{c_b} \cdot w_b = p_{c_r} \cdot (x + \epsilon(\neg \ell_p))$

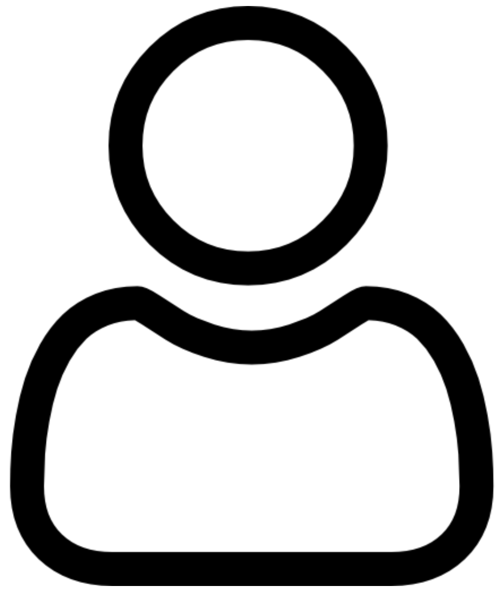
Check application of resolution rule





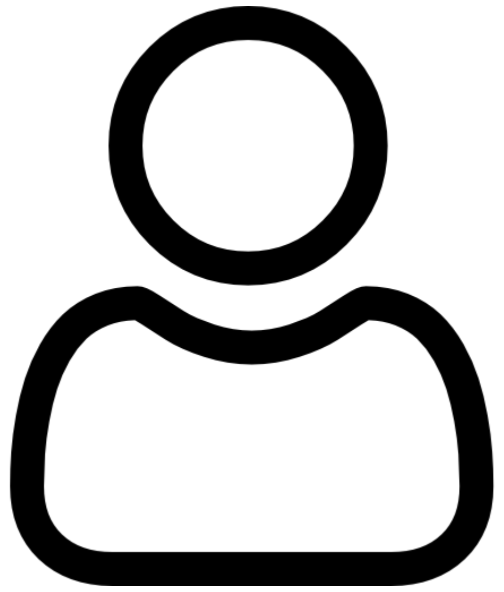
Check application of resolution rule

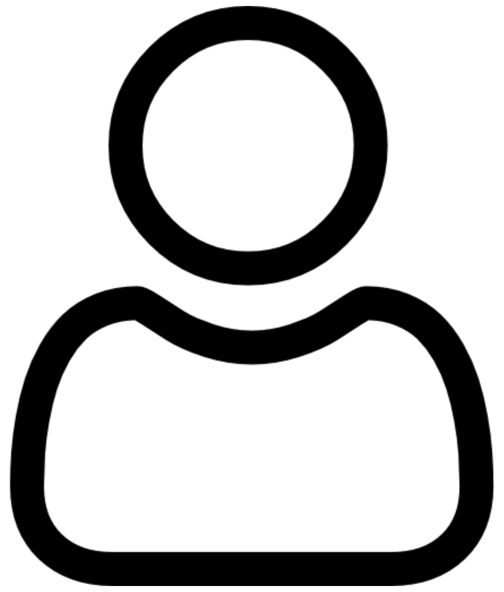




$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

Check application of resolution rule





$$p_{c_a} = (x + 2^{k-1} + 1)(x + 2^{k-1} + 2)$$

$$p_{c_b} = (x + 2^{k-1} + 1)(x + 2)$$

$$p_{c_r} = (x + 2^{k-1} + 1)$$

pivot literals: $p_{x_2} = (x + 2^{k-1} + 2),\ p_{\neg x_2} = (x + 2)$

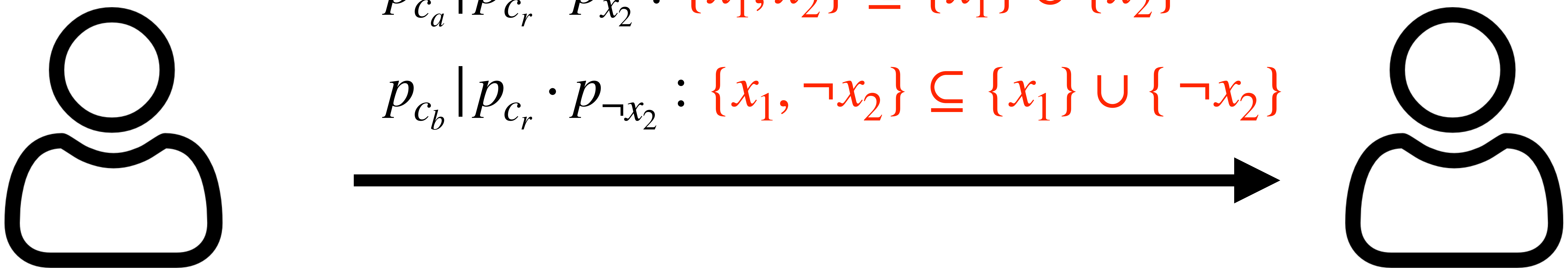
$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

Check application of resolution rule

c_a c_r pivot

$p_{c_a} | p_{c_r} \cdot p_{x_2} : \{x_1, x_2\} \subseteq \{x_1\} \cup \{x_2\}$

$p_{c_b} | p_{c_r} \cdot p_{\neg x_2} : \{x_1, \neg x_2\} \subseteq \{x_1\} \cup \{\neg x_2\}$



$$p_{c_a} = (x + 2^{k-1} + 1)(x + 2^{k-1} + 2)$$

$$p_{c_b} = (x + 2^{k-1} + 1)(x + 2)$$

$$p_{c_r} = (x + 2^{k-1} + 1)$$

pivot literals: $p_{x_2} = (x + 2^{k-1} + 2),\ p_{\neg x_2} = (x + 2)$

$$\frac{c_a = x_1 \vee x_2, \quad c_b = x_1 \vee \neg x_2}{c_r = x_1}$$

Unsatisfiability in Zero Knowledge Proof

ZKUNSAT

Fetch Clauses

Check application of resolution rule

ZK for integer comparison

ZK for polynomial relations

$$
\begin{array}{ll}
c_0 : (x_1 \vee x_2) & -, - \\
c_1 : (\neg x_1 \vee x_2) & -, - \\
c_2 : (\neg x_1 \vee \neg x_2) & -, - \\
c_3 : (x_1 \vee \neg x_2) & -, - \\
c_4 : x_2 & 0, 1 \\
c_5 : \neg x_2 & 2, 3 \\
c_6 : \perp & 4, 5
\end{array}
$$

Unsatisfiability in Zero Knowledge Proof

ZKUNSAT

Fetch Clauses | Check application of resolution rule

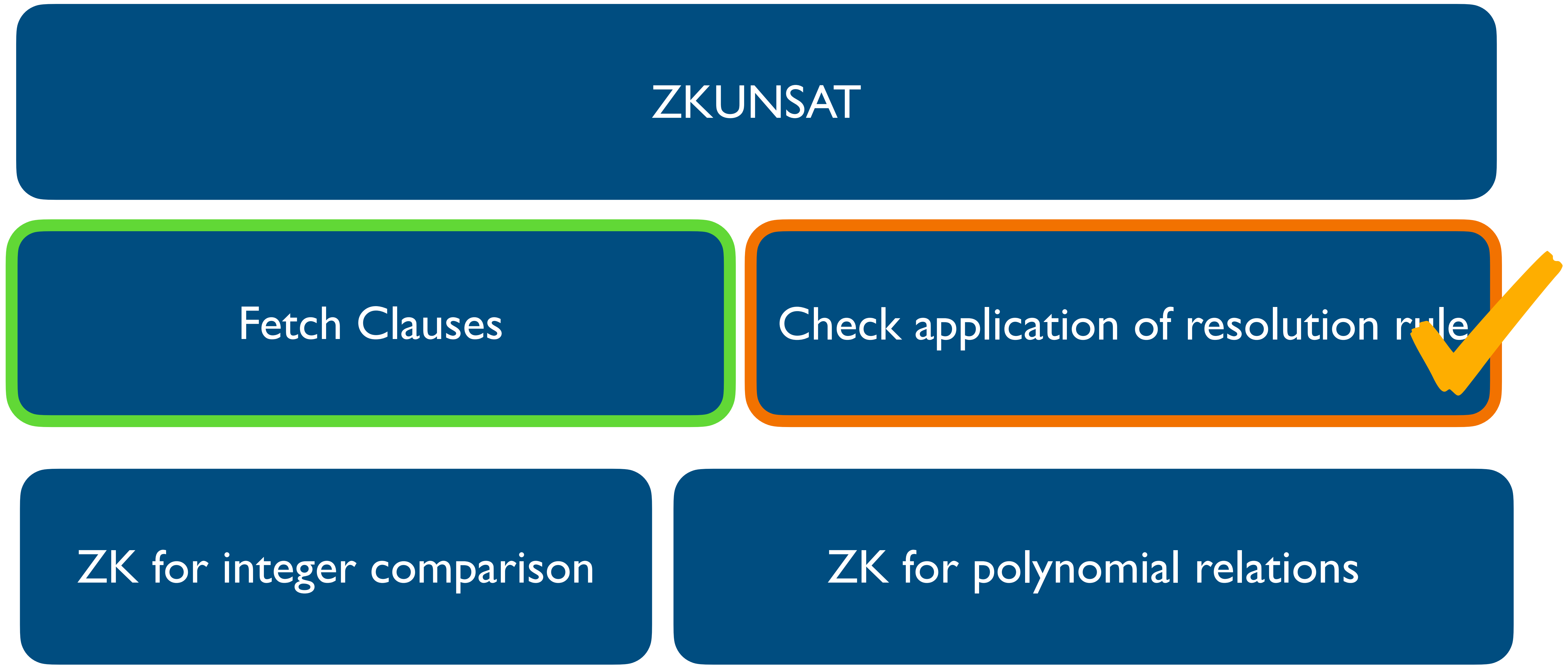
ZK for integer comparison | ZK for polynomial relations

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Unsatisfiability in Zero Knowledge Proof

Given clauses c_a, c_b, c_r , we now can check $c_a, c_b \vdash_{res} c_r$

But how we fetch c_a and c_b without revealing the access?



$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \bot$	$4, 5$

Unsatisfiability in Zero Knowledge Proof

Given clauses c_a, c_b, c_r, we now can check $c_a, c_b \vdash_{res} c_r$

But how we fetch c_a and c_b without revealing the access?

ZKUNSAT

Fetch Clauses

Check application of resolution rule ✓

ZK for integer comparison

ZK for polynomial relations

c_0	$: (x_1 \vee x_2)$	$-, -$
c_1	$: (\neg x_1 \vee x_2)$	$-, -$
c_2	$: (\neg x_1 \vee \neg x_2)$	$-, -$
c_3	$: (x_1 \vee \neg x_2)$	$-, -$
c_4	$: x_2$	$0, 1$
c_5	$: \neg x_2$	$2, 3$
c_6	$: \perp$	$4, 5$

Zero Knowledge Proof

Read-Only Array in ZK [Franzese, 21]

$c_0 : (x_1 \vee x_2)$	$-,-$
$c_1 : (\neg x_1 \vee x_2)$	$-,-$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-,-$
$c_3 : (x_1 \vee \neg x_2)$	$-,-$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_1$	$2, 4$
$c_6 : \neg x_2$	$5, 3$
$c_7 : \perp$	$4, 6$

Zero Knowledge Proof

Read-Only Array in ZK [Franzese, 21]

- Each time fetch the clause c_i for resolvent j

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_1$	$2, 4$
$c_6 : \neg x_2$	$5, 3$
$c_7 : \bot$	$4, 6$

Zero Knowledge Proof

Read-Only Array in ZK [Franzese, 21]

- Each time fetch the clause c_i for resolvent j
  - Prover sends i, c_i to the verifier when needed.

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_1$	$2, 4$
$c_6 : \neg x_2$	$5, 3$
$c_7 : \perp$	$4, 6$

Zero Knowledge Proof

Read-Only Array in ZK [Franzese, 21]

- Each time fetch the clause c_i for resolvent j
  - Prover sends i, c_i to the verifier when needed.
  - Verifier checks that $i < j$

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_1$	$2, 4$
$c_6 : \neg x_2$	$5, 3$
$c_7 : \perp$	$4, 6$

Zero Knowledge Proof

Read-Only Array in ZK [Franzese, 21]

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_1$	$2, 4$
$c_6 : \neg x_2$	$5, 3$
$c_7 : \bot$	$4, 6$

- Each time fetch the clause c_i for resolvent j
  - Prover sends i, c_i to the verifier when needed.
  - Verifier checks that $i < j$
  - Record $(0,c_0), (1,c_1), (2,c_2), (4,c_4), (5,c_5), (3,c_3), (4,c_4), (6,c_6)$

Zero Knowledge Proof

Read-Only Array in ZK (example)

Zero Knowledge Proof

Read-Only Array in ZK (example)

- Check if all records are consistent periodically

Zero Knowledge Proof

Read-Only Array in ZK (example)

- Check if all records are consistent periodically
  - $L = [(0,c_0), (1,c_1), (2,c_2), (4,c_4), (5,c_5), (3,c_3), (4,c_4), (6,c_6)]$

Zero Knowledge Proof

Read-Only Array in ZK (example)

- Check if all records are consistent periodically
  - $L = [(0, c_0), (1, c_1), (2, c_2), (4, c_4), (5, c_5), (3, c_3), (4, c_4), (6, c_6)]$
  - Prover sorts the list of records and gets
    $L' = [(0, c_0), (1, c_1), (2, c_2), (3, c_3), (4, c_4), (4, c_4), (5, c_5), (6, c_6)]$

Zero Knowledge Proof

Read-Only Array in ZK (example)

- Check if all records are consistent periodically
    - $L = [(0,c_0),(1,c_1),(2,c_2),(4,c_4),(5,c_5),(3,c_3),(4,c_4),(6,c_6)]$
    - Prover sorts the list of records and gets
      $L' = [(0,c_0),(1,c_1),(2,c_2),(3,c_3),(4,c_4),(4,c_4),(5,c_5),(6,c_6)]$
        - $L = L'$ in the sense of set (using polynomial equivalence)

Zero Knowledge Proof

Read-Only Array in ZK (example)

- Check if all records are consistent periodically
  - $L = [(0,c_0),(1,c_1),(2,c_2),(4,c_4),(5,c_5),(3,c_3),(4,c_4),(6,c_6)]$
  - Prover sorts the list of records and gets
    $L' = [(0,c_0),(1,c_1),(2,c_2),(3,c_3),(4,c_4),(4,c_4),(5,c_5),(6,c_6)]$
    - $L = L'$ in the sense of set (using polynomial equivalence)
    - In L', either $i'_{t+1} > i'_t$ or $c_{i'_t} = c_{i'_{t+1}}$ when $i'_t = i'_{t+1}$ holds

Zero Knowledge Proof

Read-Only Array in ZK (example)

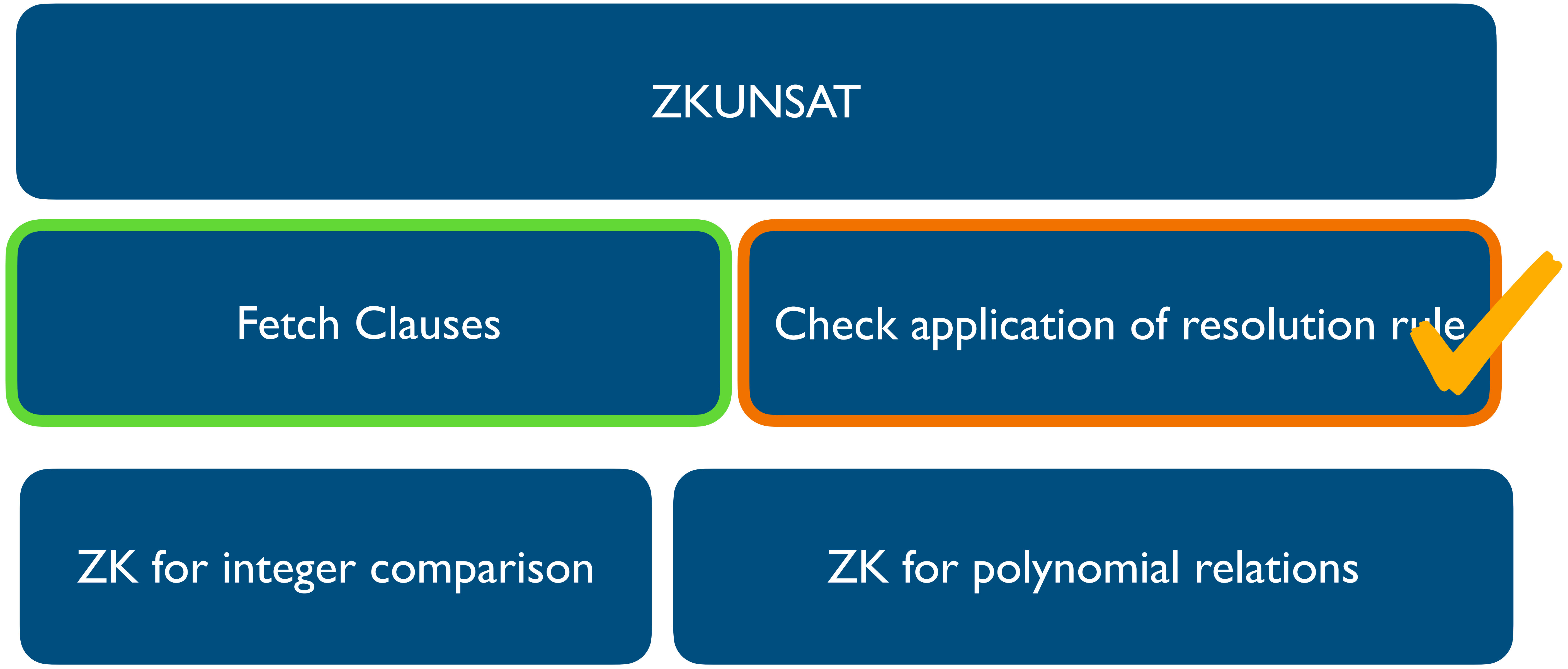
- Check if all records are consistent periodically
  - $L = [(0,c_0),(1,c_1),(2,c_2),(4,c_4),(5,c_5),(3,c_3),(4,c_4),(6,c_6)]$
  - Prover sorts the list of records and gets
    $L' = [(0,c_0),(1,c_1),(2,c_2),(3,c_3),(4,c_4),(4,c_4),(5,c_5),(6,c_6)]$
    - $L = L'$ in the sense of set (using polynomial equivalence)
    - In L', either $i'_{t+1} > i'_t$ or $c_{i'_t} = c_{i'_{t+1}}$ when $i'_t = i'_{t+1}$ holds

Zero Knowledge Proof

Read-Only Array in ZK (example)

- Check if all records are consistent periodically
  - $L = [(0,c_0), (1,c_1), (2,c_2), (4,c_4), (5,c_5), (3,c_3), (4,c_4), (6,c_6)]$
  - Prover sorts the list of records and gets
    $L' = [(0,c_0), (1,c_1), (2,c_2), (3,c_3), (4,c_4), (4,c_4), (5,c_5), (6,c_6)]$
    - $L = L'$ in the sense of set (using polynomial equivalence)
    - In L', either $i'_{t+1} > i'_t$ or $c_{i'_t} = c_{i'_{t+1}}$ when $i'_t = i'_{t+1}$ holds

Unsatisfiability in Zero Knowledge Proof

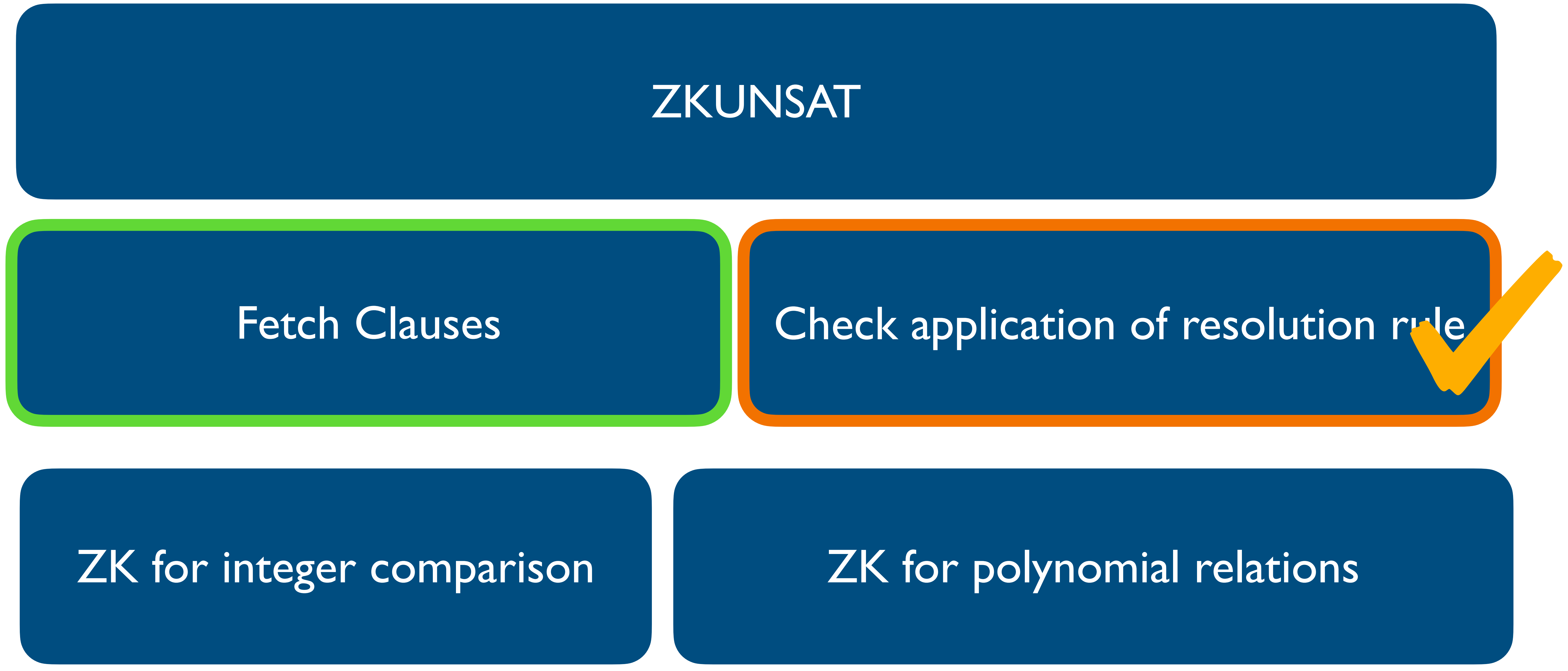


$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Unsatisfiability in Zero Knowledge Proof

Put them together:

- Fetch input clauses for each resolution using ROARRAY in ZK
- Check application of the resolution rule using polynomial relations in ZK

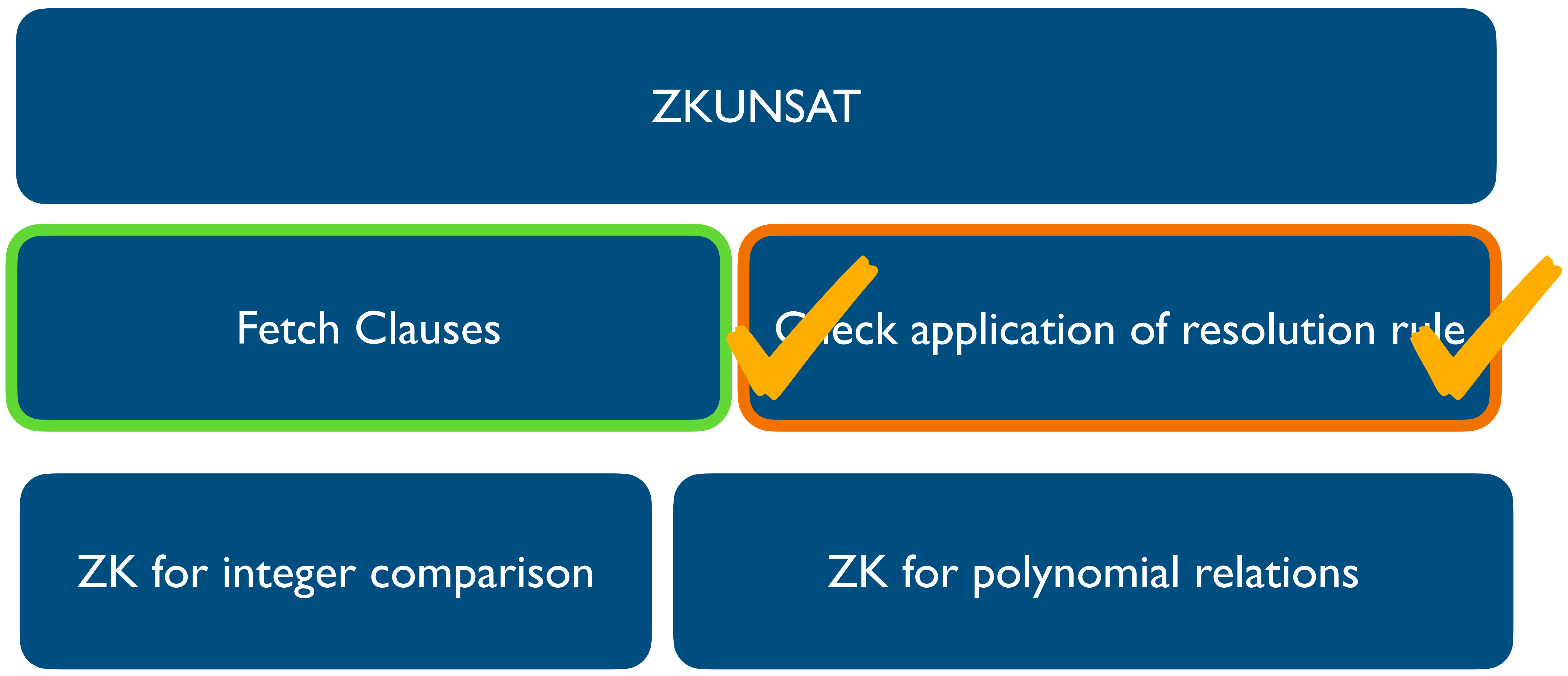


$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Unsatisfiability in Zero Knowledge Proof

Put them together:

- Fetch input clauses for each resolution using ROARRAY in ZK
- Check application of the resolution rule using polynomial relations in ZK



$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Evaluation

Benchmark setting

- AWS instances of type r5b.2xlarge
- 64 GB of memory, 16 vCPUs
- 10 Gbps network connection between the prover and the verifier

Evaluation

$c_0 : (x_1 \vee x_2)$	$-, -$
$c_1 : (\neg x_1 \vee x_2)$	$-, -$
$c_2 : (\neg x_1 \vee \neg x_2)$	$-, -$
$c_3 : (x_1 \vee \neg x_2)$	$-, -$
$c_4 : x_2$	$0, 1$
$c_5 : \neg x_2$	$2, 3$
$c_6 : \perp$	$4, 5$

Width: the maximum number of literals a clause in the proof can have.

Width = 2 in this example

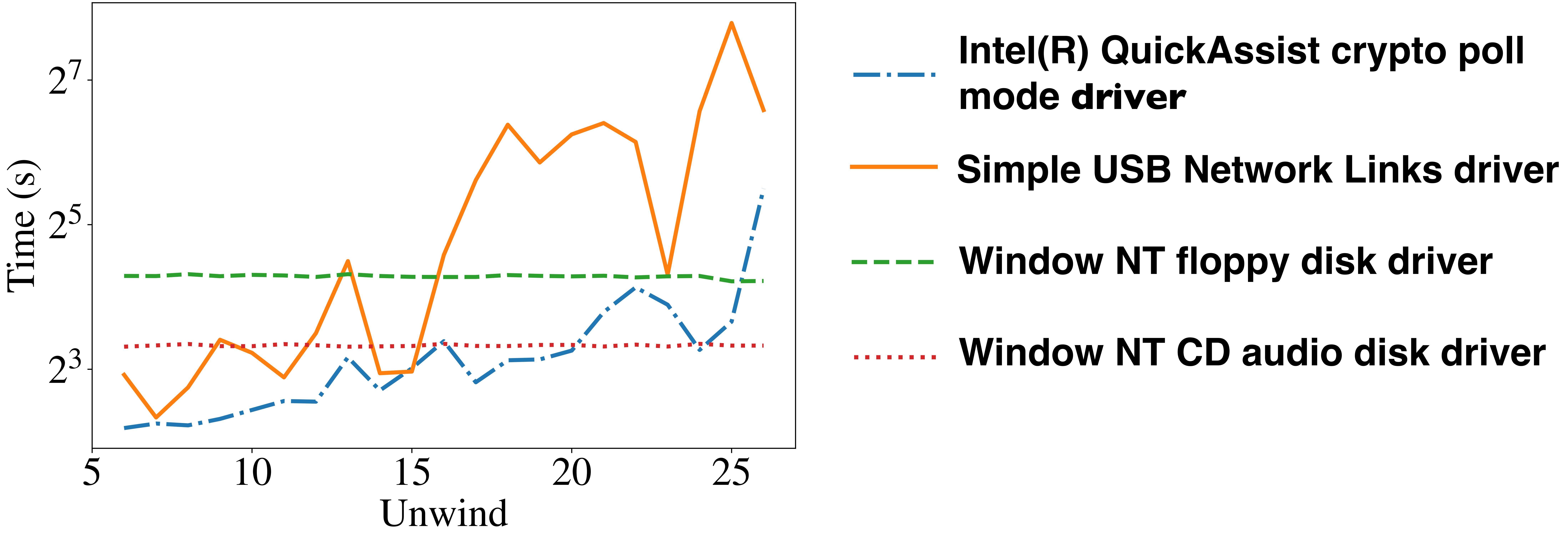
Length: the number of applications of resolution rule.

Length = 3 in this example

- Time and memory requirements **depend on the length and clausal width** of the proof
- ZKUNSAT takes **less than 1 min** to verify proofs of large width (400) and length (8000)

Evaluation

Verification tasks for system drivers



- Unwind is a parameter for translating the verification tasks to Boolean formulae
- Width ≤ 256 and Length $\leq 65K$
- ZKUNSAT can verify UNSAT of formulae from **system drivers verification tasks within 5min**

Evaluation

Other large instances

Program	Len. (K)	Width	Time (s)
inv-square-int	194	414	172.5
rlim-invariant	481	198	1943.3
sin-interpolated-smallrange	375	308	2571.8
interpolation	135	790	3771.6
inv-sqrt-quake	182	749	5764.1
zonotope-loose	35	2887	9996.9
zonotope-tight	64	2887	11143.3
interpolation2	600	1047	OOM

Evaluation

Other large instances

Program	Len. (K)	Width	Time (s)
inv-square-int	194	414	172.5
rlim-invariant	481	198	1943.3
sin-interpolated-smallrange	375	308	2571.8
interpolation	135	790	3771.6
inv-sqrt-quake	182	749	5764.1
zonotope-loose	35	2887	9996.9
zonotope-tight	64	2887	11143.3
interpolation2	600	1047	OOM

Evaluation

Other large instances

Program	Len. (K)	Width	Time (s)
inv-square-int	194	414	172.5
rlim-invariant	481	198	1943.3
sin-interpolated-smallrange	375	308	2571.8
interpolation	135	790	3771.6
inv-sqrt-quake	182	749	5764.1
zonotope-loose	35	2887	9996.9
zonotope-tight	64	2887	11143.3
interpolation2	600	1047	OOM

Further improvement via computing clusters: work to appear in CCS 2023

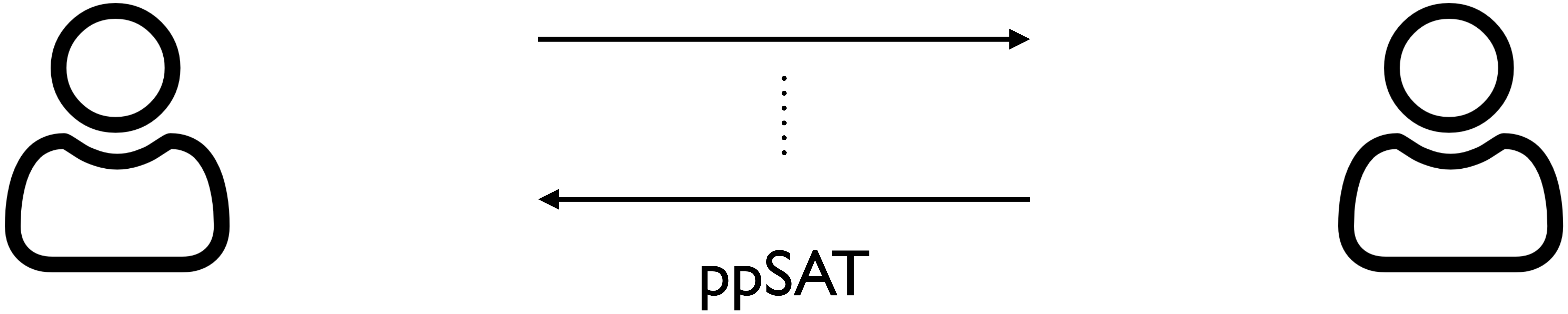
Contribution

- Privacy preserving program verification is in demand
- Encoding resolution proof by polynomials
- UNSAT in ZK is practical

Future Work

Future Work

- **SAT : Privacy-preserving SAT solving (ppSAT)**

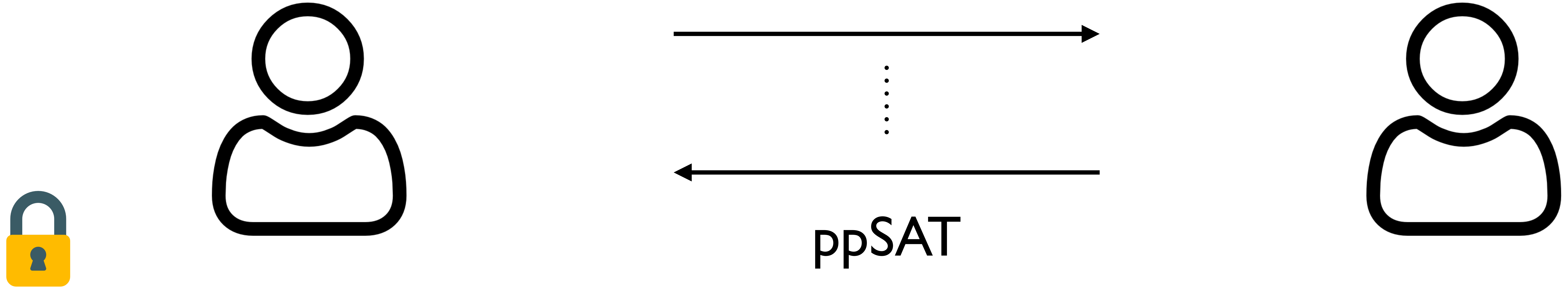


$\phi_0 = (x_0 \vee x_1) \wedge (x_0 \vee \neg x_1)$

$\phi_1 = (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$

Future Work

- **SAT : Privacy-preserving SAT solving (ppSAT)**

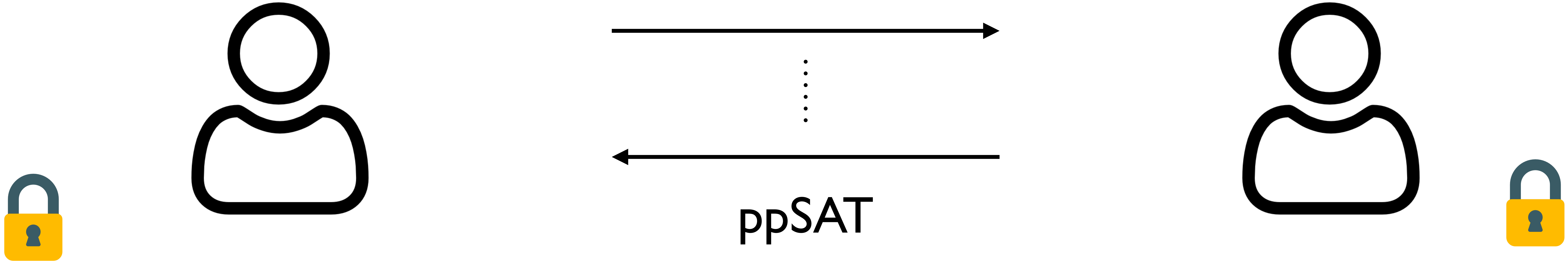


$\phi_0 = (x_0 \vee x_1) \wedge (x_0 \vee \neg x_1)$

$\phi_1 = (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$

Future Work

- **SAT : Privacy-preserving SAT solving (ppSAT)**

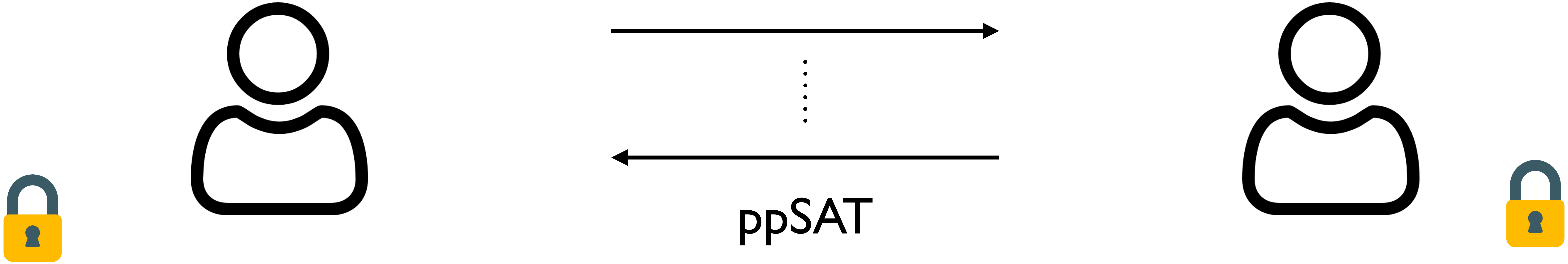


$\phi_0 = (x_0 \lor x_1) \land (x_0 \lor \neg x_1)$

$\phi_1 = (\neg x_0 \lor x_1) \land (\neg x_0 \lor \neg x_1)$

Future Work

- **SAT : Privacy-preserving SAT solving (ppSAT)**



$\phi_0 = (x_0 \vee x_1) \wedge (x_0 \vee \neg x_1)$

$\phi_1 = (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$

ppSAT: Towards Two-Party Private SAT Solving, USENIX Security 2022

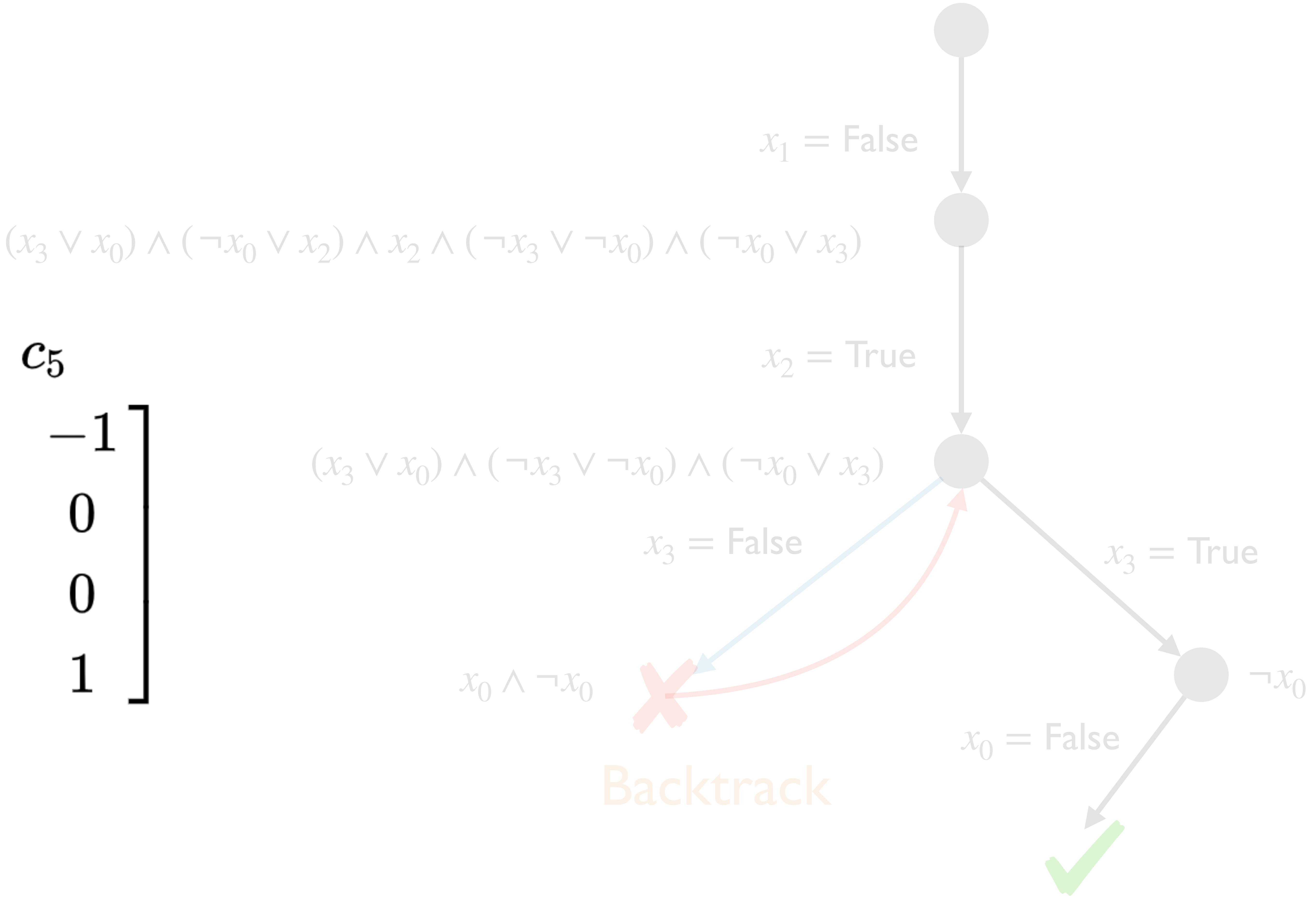
Boolean SAT Solving: A DPLL Example

$$\phi(x_0, x_1, x_2, x_3) = (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$

Boolean SAT Solving: A DPLL Example

$$\phi(x_0, x_1, x_2, x_3) = (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$

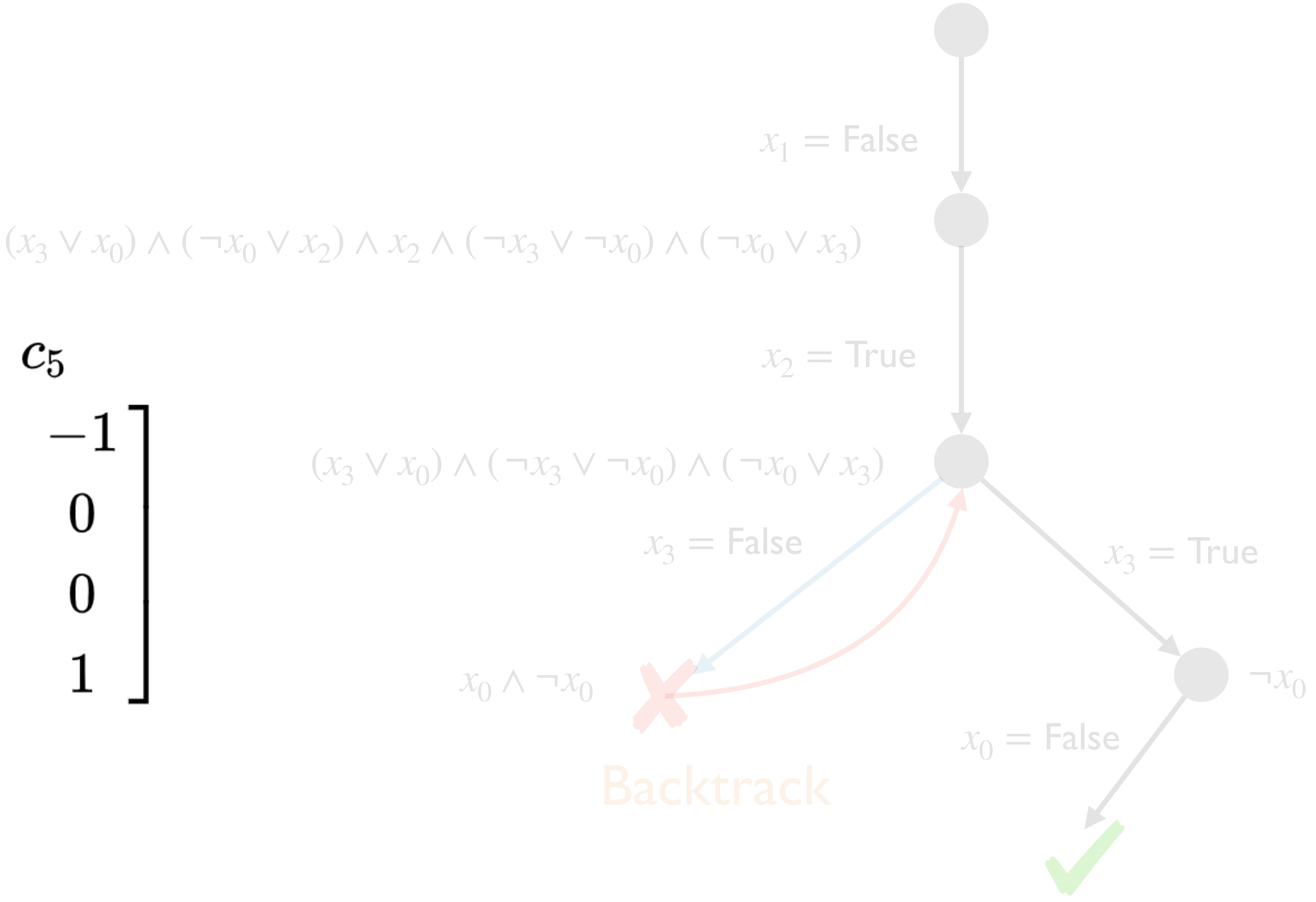
$$\begin{array}{cc} & \begin{array}{cccccc} c_0, & c_1, & c_2, & c_3, & c_4, & c_5 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \\ x_3 \end{array} & \begin{bmatrix} 1 & -1 & 0 & 0 & -1 & -1 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$



Boolean SAT Solving: A DPLL Example

$$\phi(x_0, x_1, x_2, x_3) = (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$

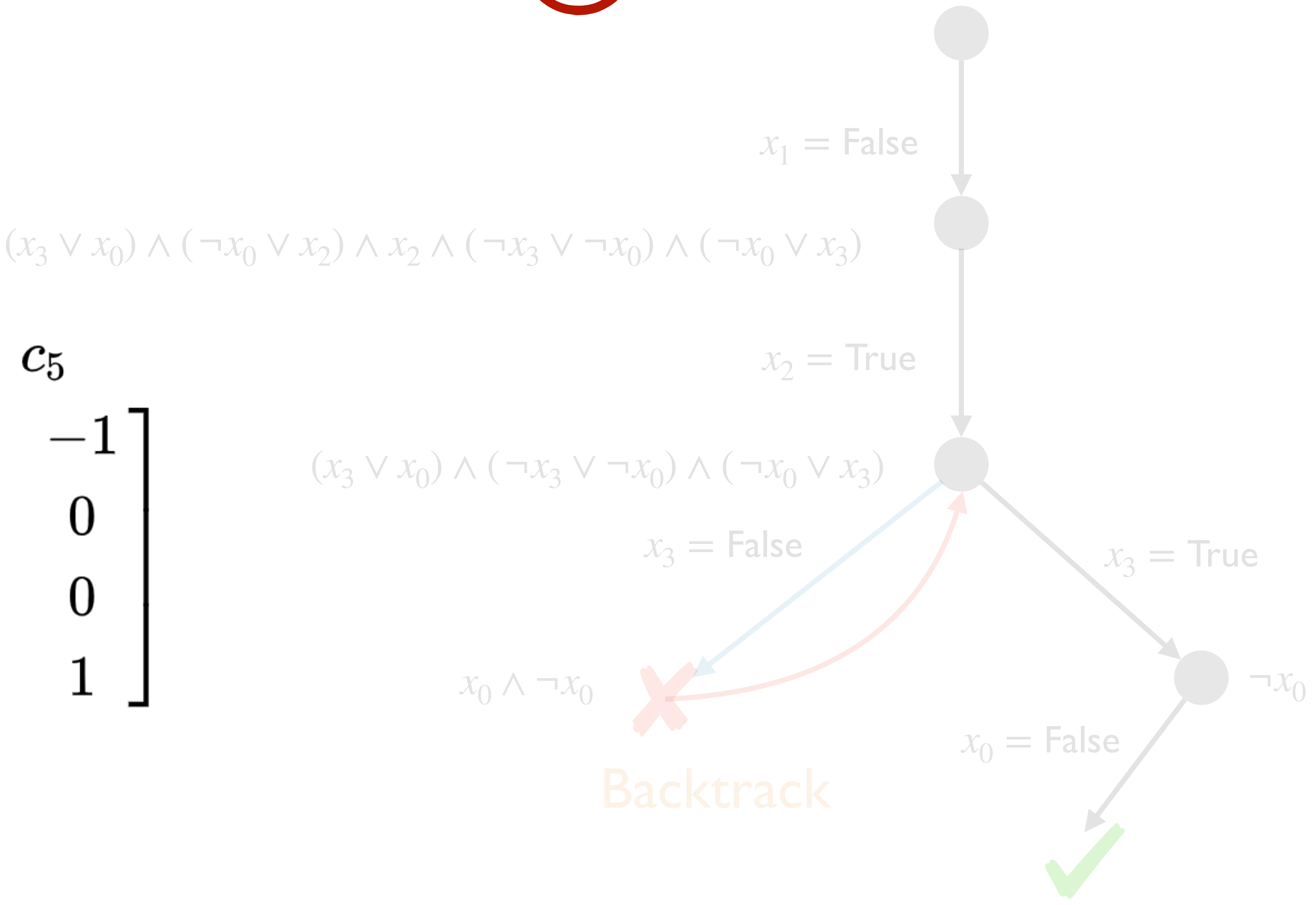
$$\begin{array}{cc} & \begin{array}{cccccc} c_0, & c_1, & c_2, & c_3, & c_4, & c_5 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \\ x_3 \end{array} & \begin{bmatrix} 1 & -1 & 0 & 0 & -1 & -1 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$



Boolean SAT Solving: A DPLL Example

$$\phi(x_0, x_1, x_2, x_3) = (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$

$$\begin{array}{c c} & \begin{array}{cccccc} c_0, & c_1, & c_2, & c_3, & c_4, & c_5 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \\ x_3 \end{array} & \begin{bmatrix} 1 & -1 & 0 & 0 & -1 & -1 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

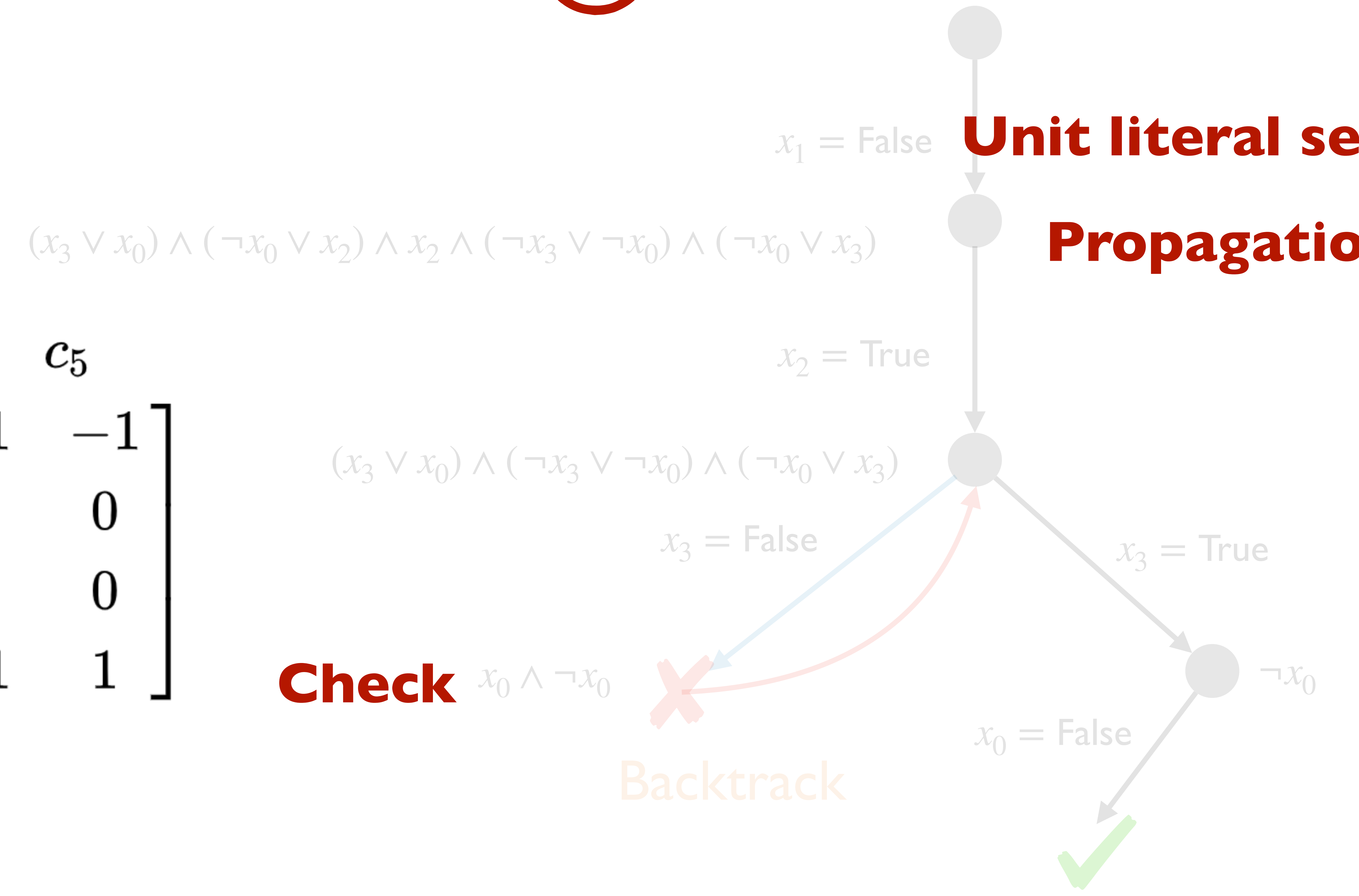


Boolean SAT Solving: A DPLL Example

$$\phi(x_0, x_1, x_2, x_3) = (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$

Unit literal search

Propagation

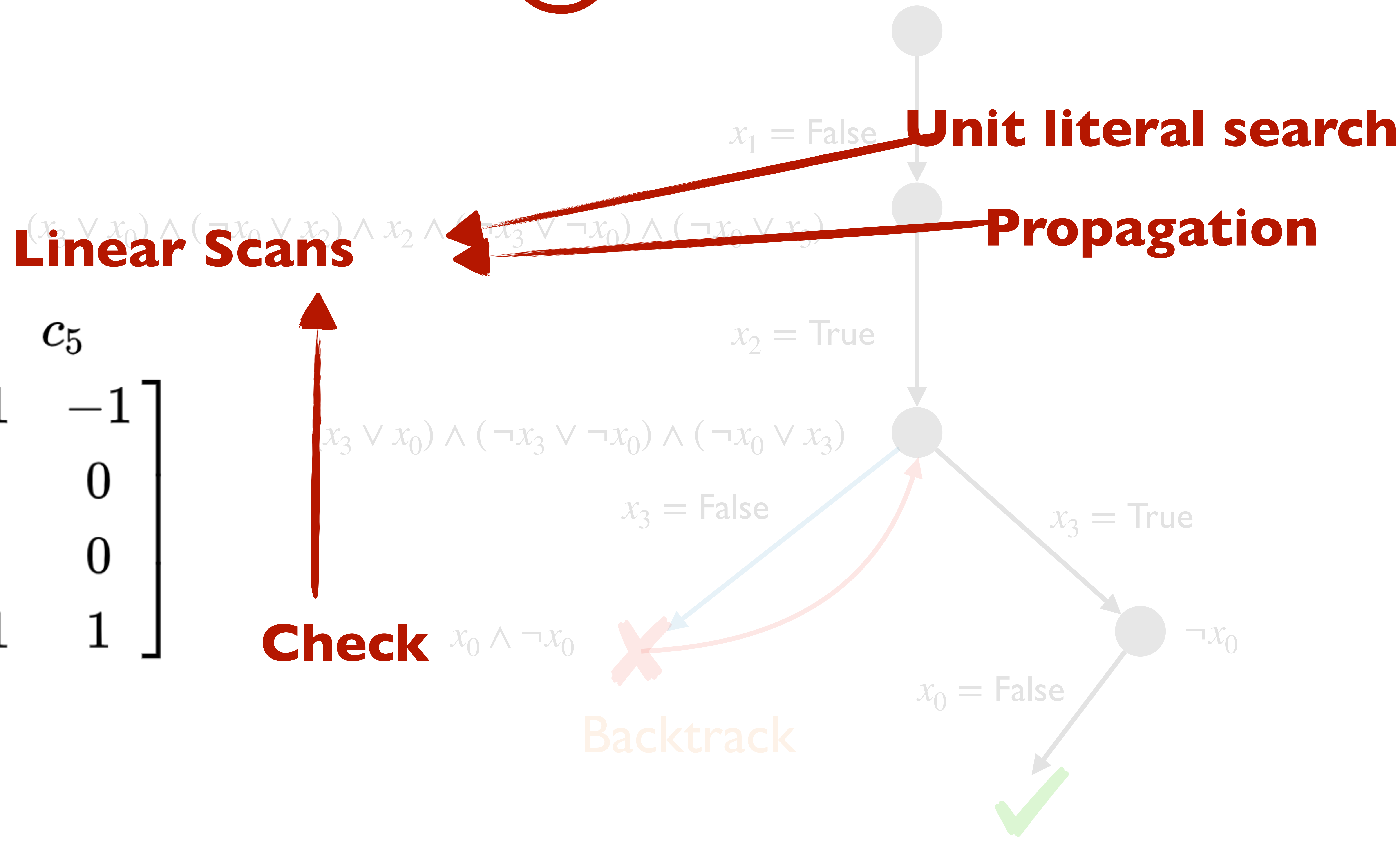


$$\begin{array}{c c} & \begin{matrix} c_0, & c_1, & c_2, & c_3, & c_4, & c_5 \end{matrix} \\ \begin{matrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 1 & -1 & 0 & 0 & -1 & -1 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

Check

Boolean SAT Solving: A DPLL Example

$$\phi(x_0, x_1, x_2, x_3) = (x_3 \vee x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge (x_1 \vee x_2) \wedge \neg x_1 \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$$



$$\begin{array}{c c}
 & \begin{array}{cccccc} c_0, & c_1, & c_2, & c_3, & c_4, & c_5 \end{array} \\
\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ x_3 \end{array} & \begin{bmatrix} 1 & -1 & 0 & 0 & -1 & -1 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}
\end{array}$$

Future Work

- **DLIS**
  - Select the most commonly appearing literal and the smallest index
  - Return the assignment that makes it true
  - Deterministic
  - Example: DLIS will guess $\neg x_0$ for $(x_3 \vee x_0) \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$

Heuristics

- **Rand**
  - Uniformly select a random undecided literal
  - Randomized
  - Example: $(x_3 \vee x_0) \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$

Randomly guess one of $\{x_0, \neg x_0, x_3, \neg x_3\}$ each with $\dfrac{1}{4}$ probability

Future Work

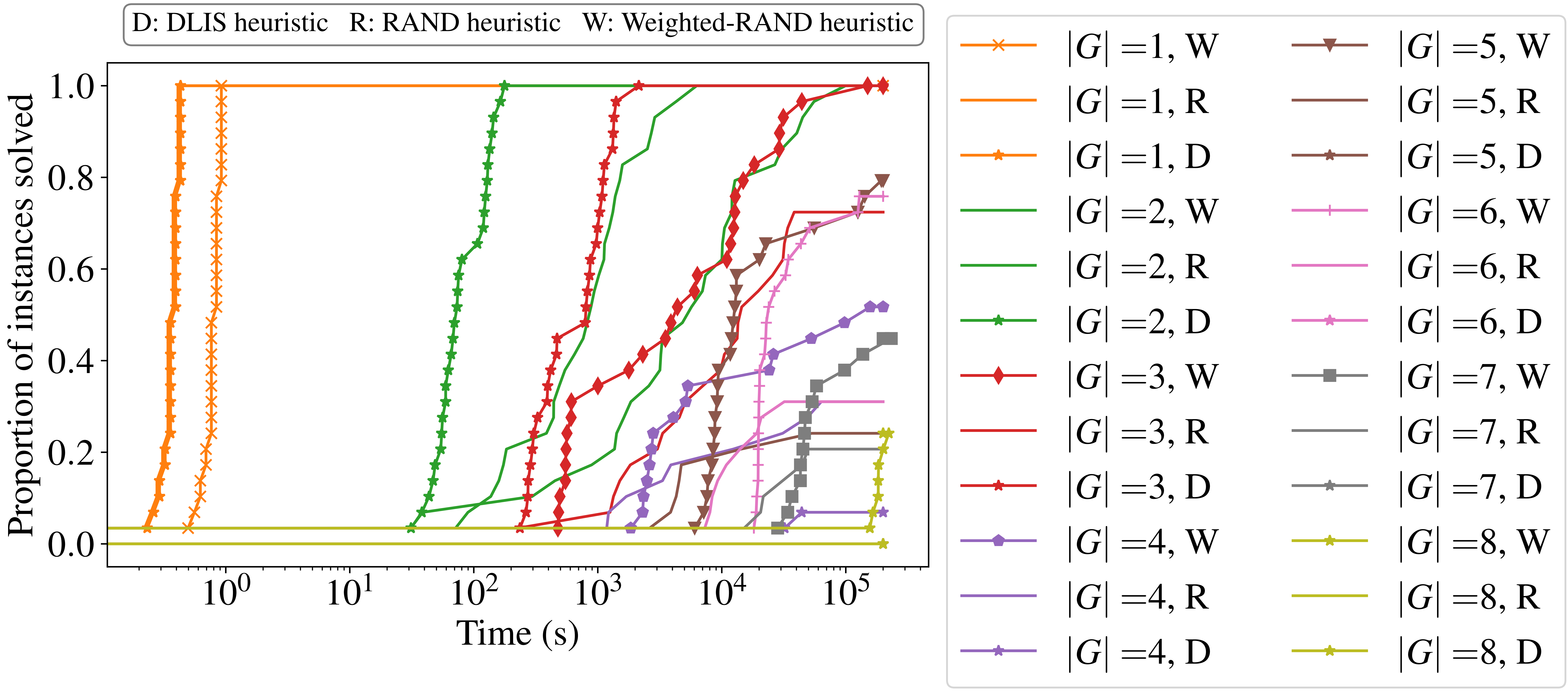
- **Weighted-Rand**
  - Select a random undecided literal according to its frequency
  - Randomized
  - Example: $(x_3 \vee x_0) \wedge (\neg x_3 \vee \neg x_0) \wedge (\neg x_0 \vee x_3)$

guess x_0 with $\frac{1}{6}$ chance, guess $\neg x_0$ with $\frac{2}{6}$ chance, etc.

Future Work

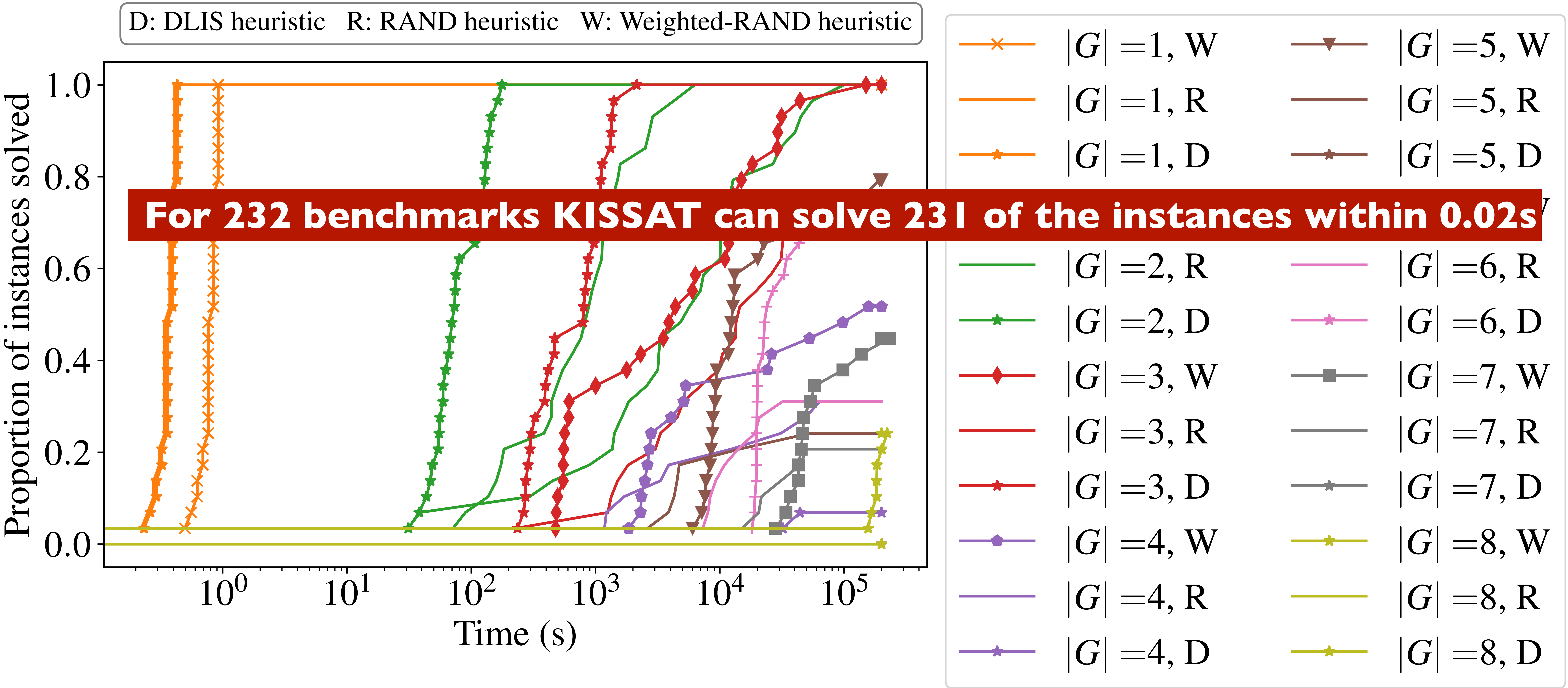
Future Work

ppSAT: Towards Two-Party Private SAT Solving, USENIX Security 2022



Future Work

ppSAT: Towards Two-Party Private SAT Solving, USENIX Security 2022



For 232 benchmarks KISSAT can solve 231 of the instances within 0.02s

Future Work

Future Work

Better heuristics when it comes to privacy preserving setting?

Thank you!

Ning Luo: ning.luo@northwestern.edu

https://github.com/PP-FM

I am on job market

Reference

Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In 22nd ACM STOC, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

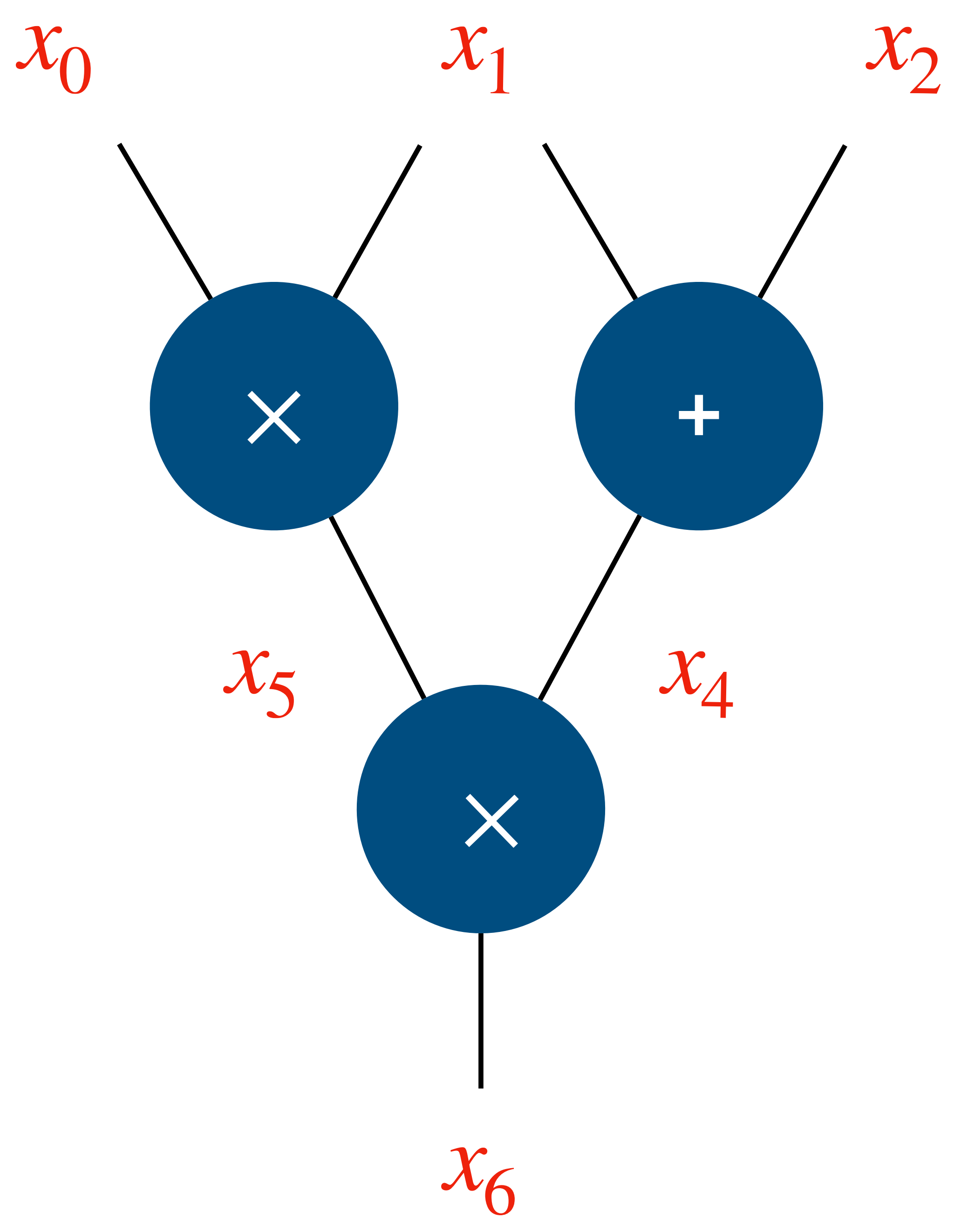
Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In ACM Conf. on Computer and Communications Security (CCS) 2021. ACM Press, 2021.

J. A. Robinson. 1965. A Machine-Oriented Logic Based on the Resolution Principle. J. ACM 12, 1 (Jan. 1965), 23–41. DOI:https://doi.org/10.1145/321250.321253

Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. 2021. Constant-Overhead Zero-Knowledge for RAM Programs. In <i>Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security</i> (<i>CCS '21</i>). Association for Computing Machinery, New York, NY, USA, 178–191. DOI:https://doi.org/10.1145/3460120.3484800

Zero Knowledge Proof

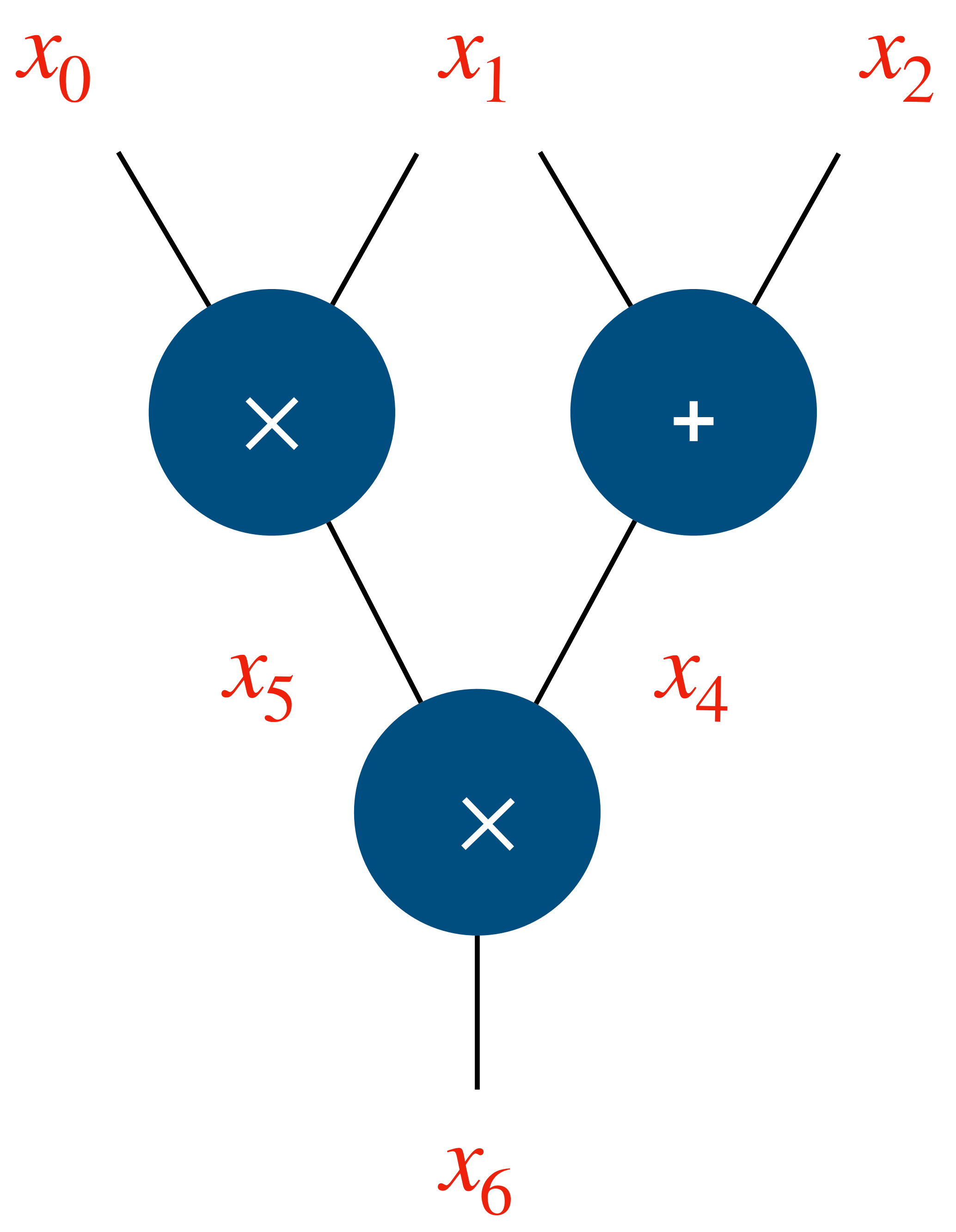
ZKP based on information theoretic MAC



Zero Knowledge Proof

ZKP based on information theoretic MAC

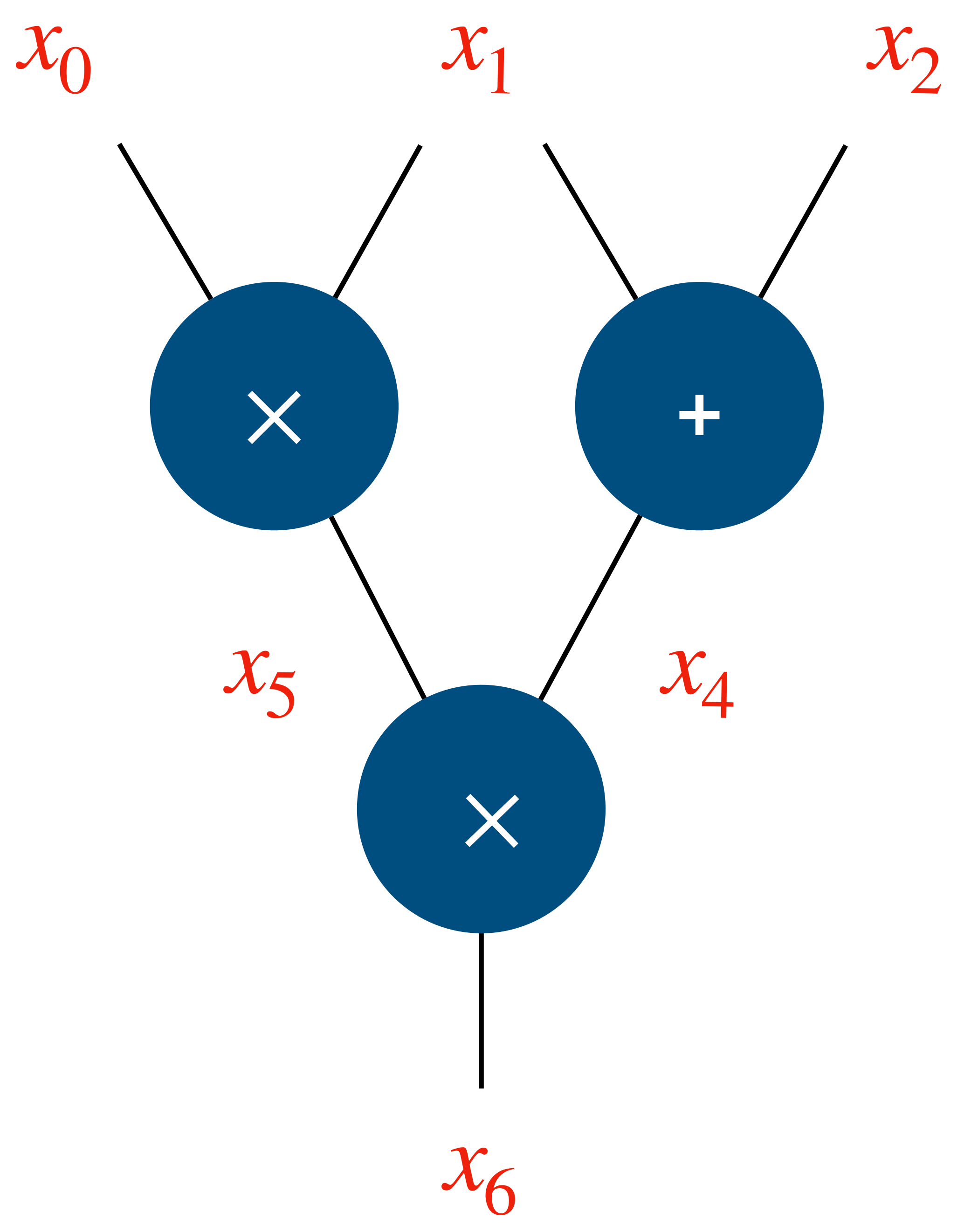
- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$



Zero Knowledge Proof

ZKP based on information theoretic MAC

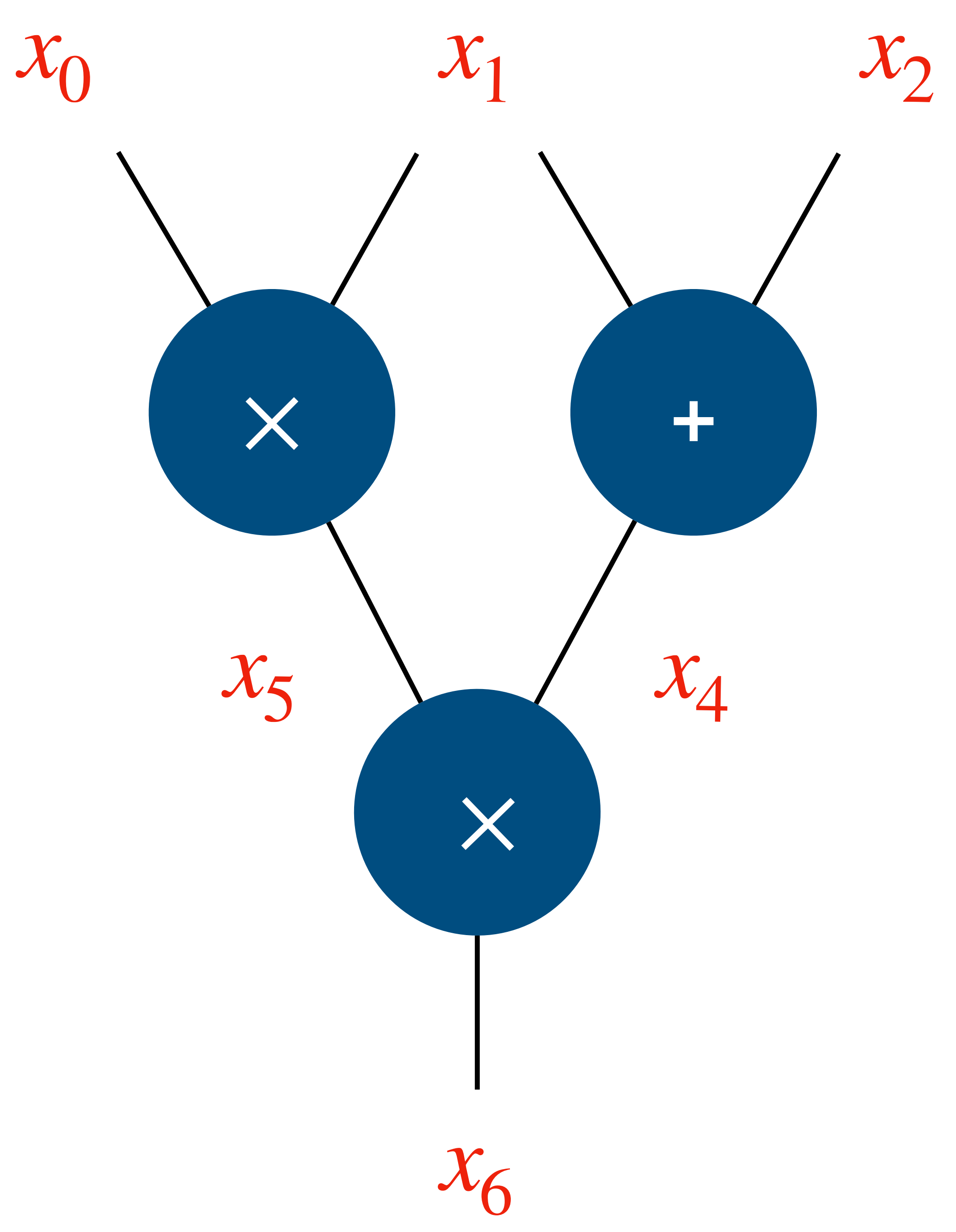
- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$



Zero Knowledge Proof

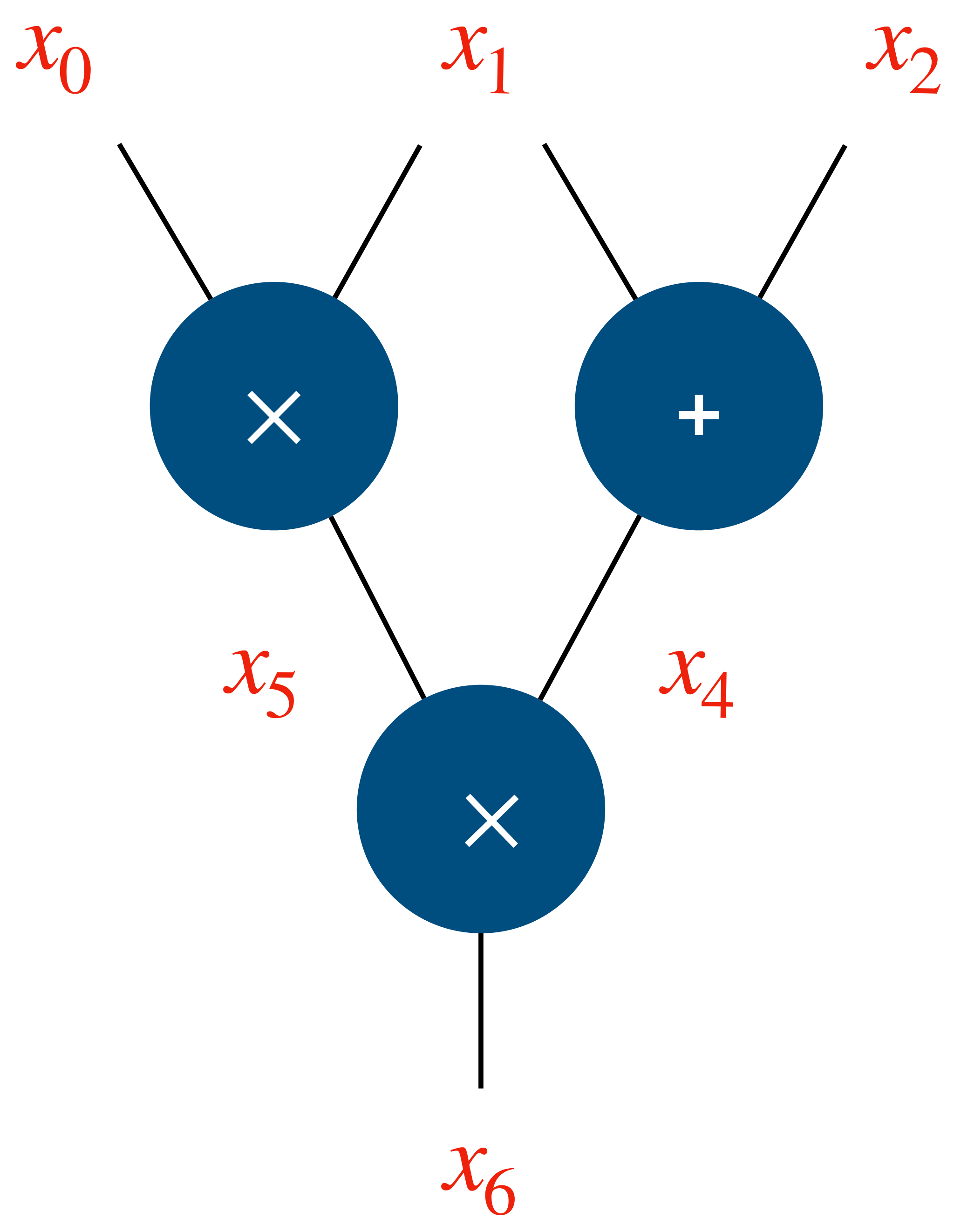
ZKP based on information theoretic MAC

- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$
    - x if x is shared in such a way



Zero Knowledge Proof
ZKP based on information theoretic MAC

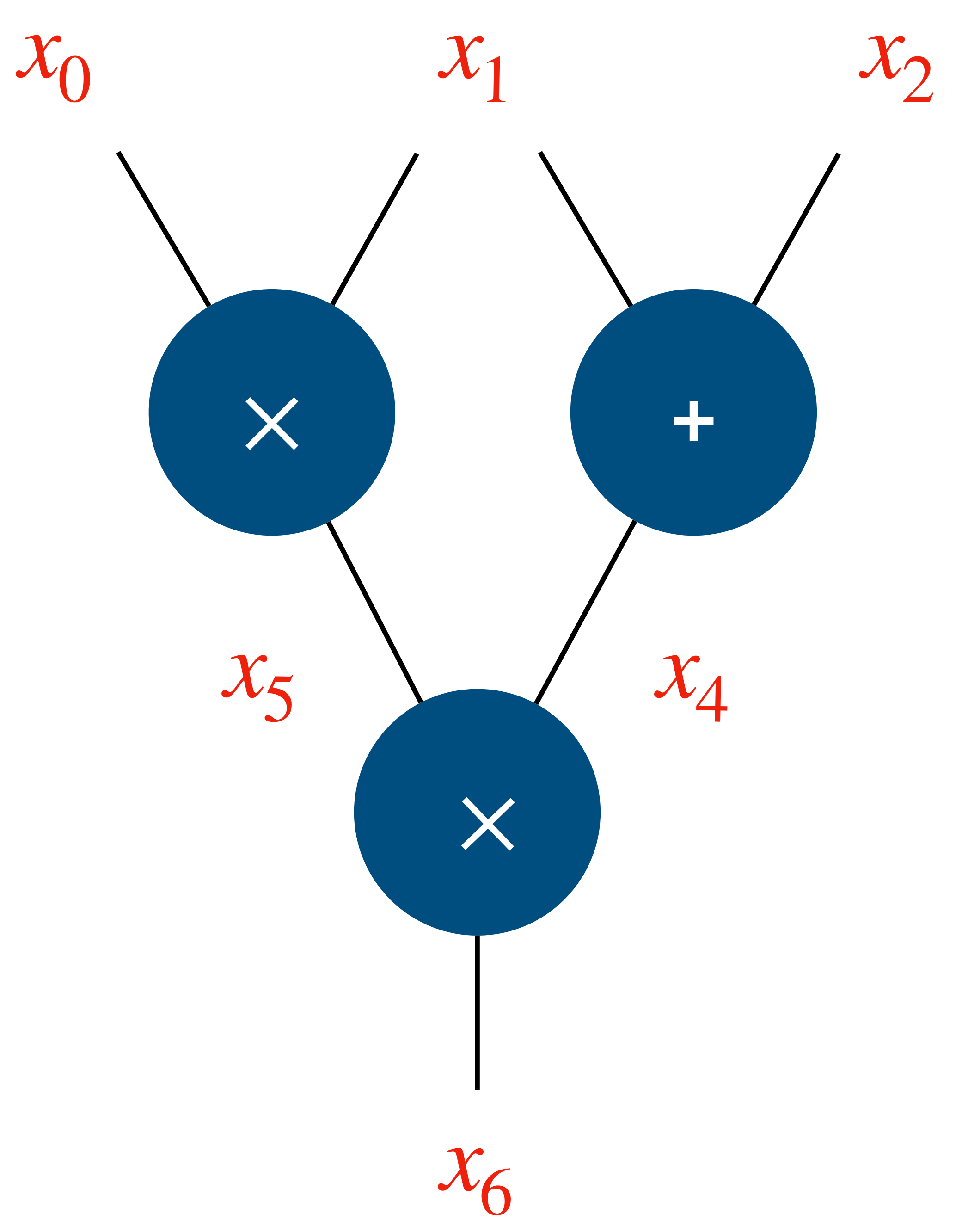
- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$
  - x if x is shared in such a way
- Addition gate: MAC for $p(x, y) = a \cdot x + b \cdot y$



Zero Knowledge Proof

ZKP based on information theoretic MAC

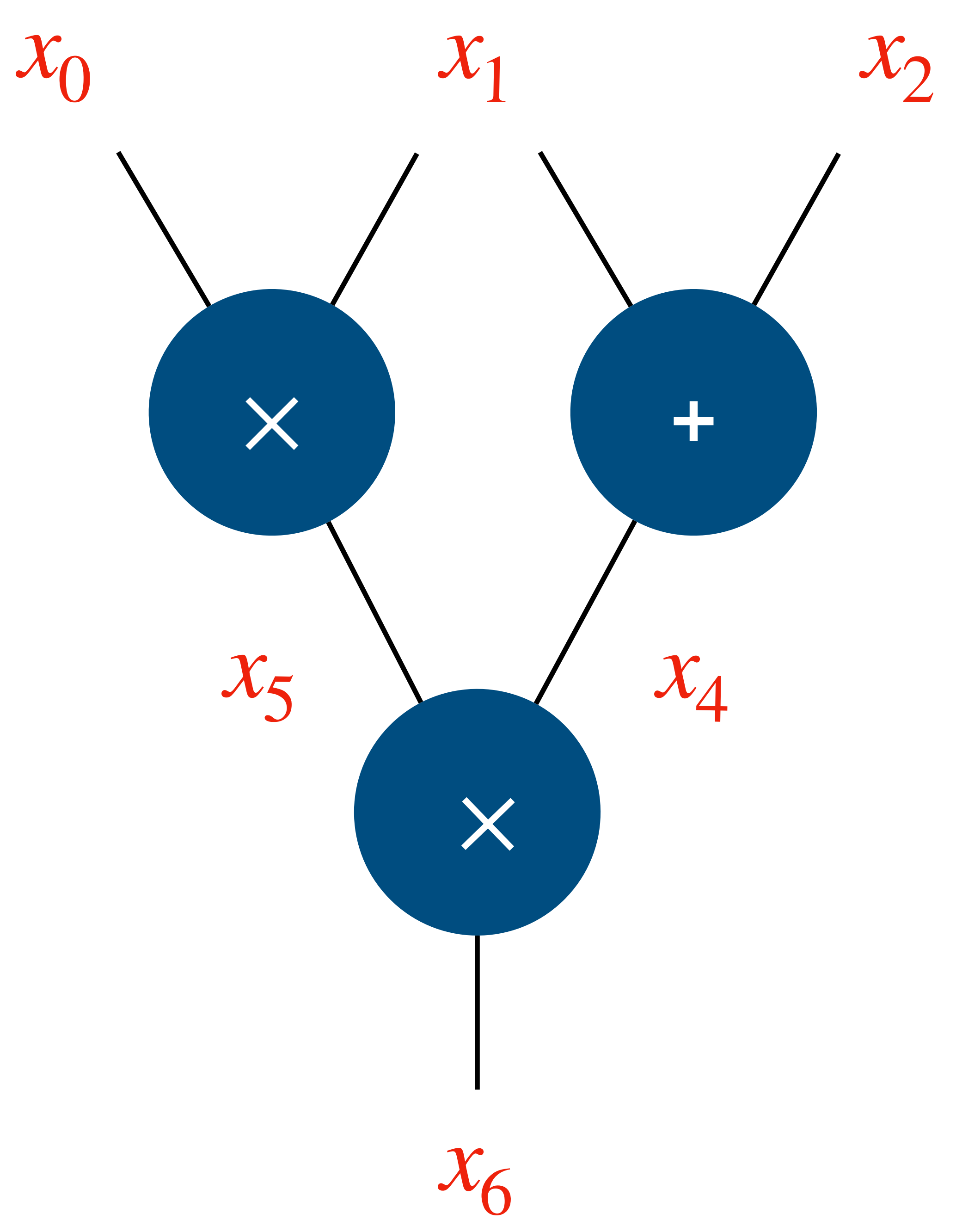
- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$
    - x if x is shared in such a way
- Addition gate: MAC for $p(x, y) = a \cdot x + b \cdot y$
    - Prover locally computes $M_{p(x,y)} = a \cdot M_x + b \cdot M_y$



Zero Knowledge Proof

ZKP based on information theoretic MAC

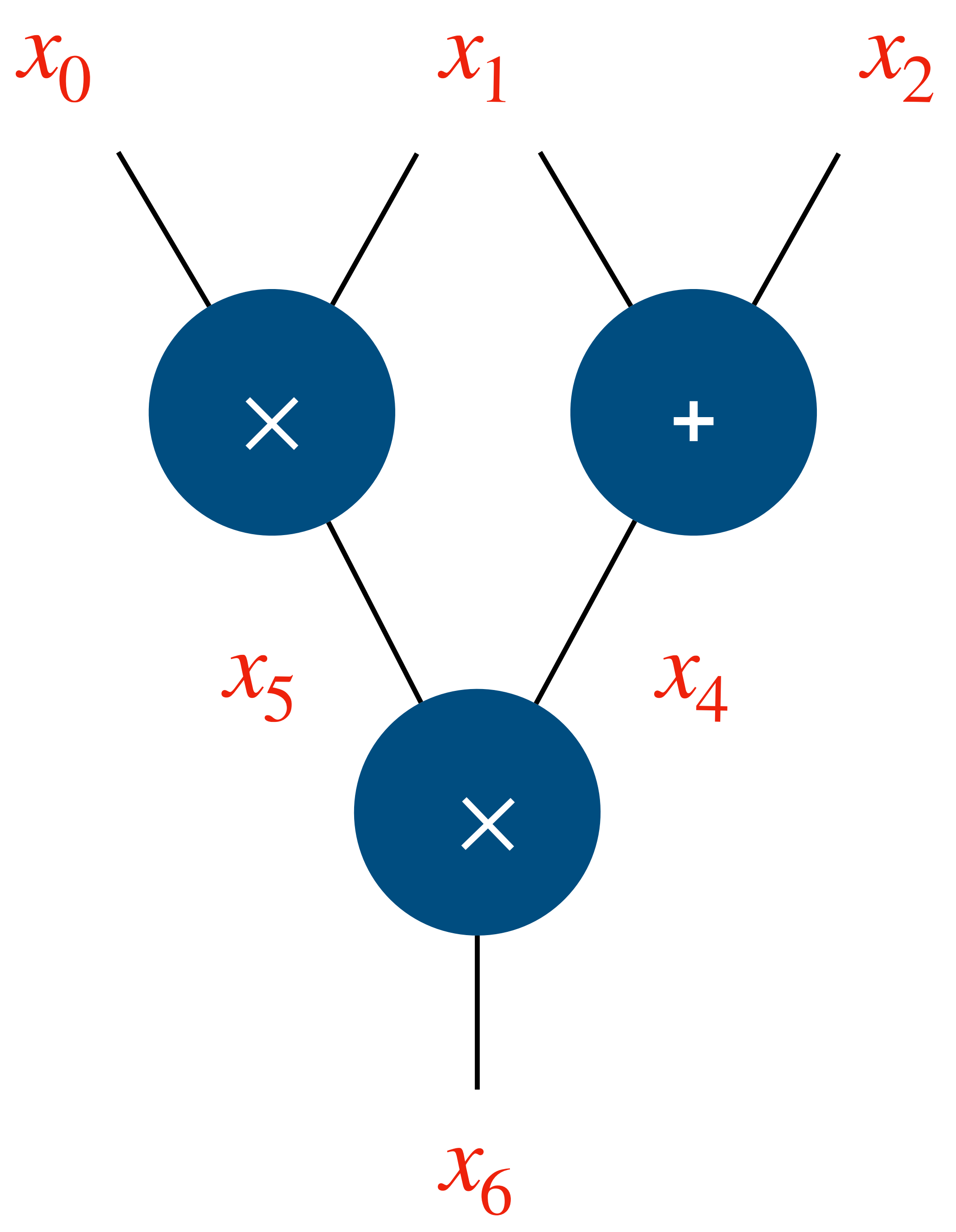
- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$
    - x if x is shared in such a way
- Addition gate: MAC for $p(x, y) = a \cdot x + b \cdot y$
    - Prover locally computes $M_{p(x,y)} = a \cdot M_x + b \cdot M_y$
    - Verifier locally computes $k_{p(x,y)} = a \cdot k_x + b \cdot k_y$



Zero Knowledge Proof

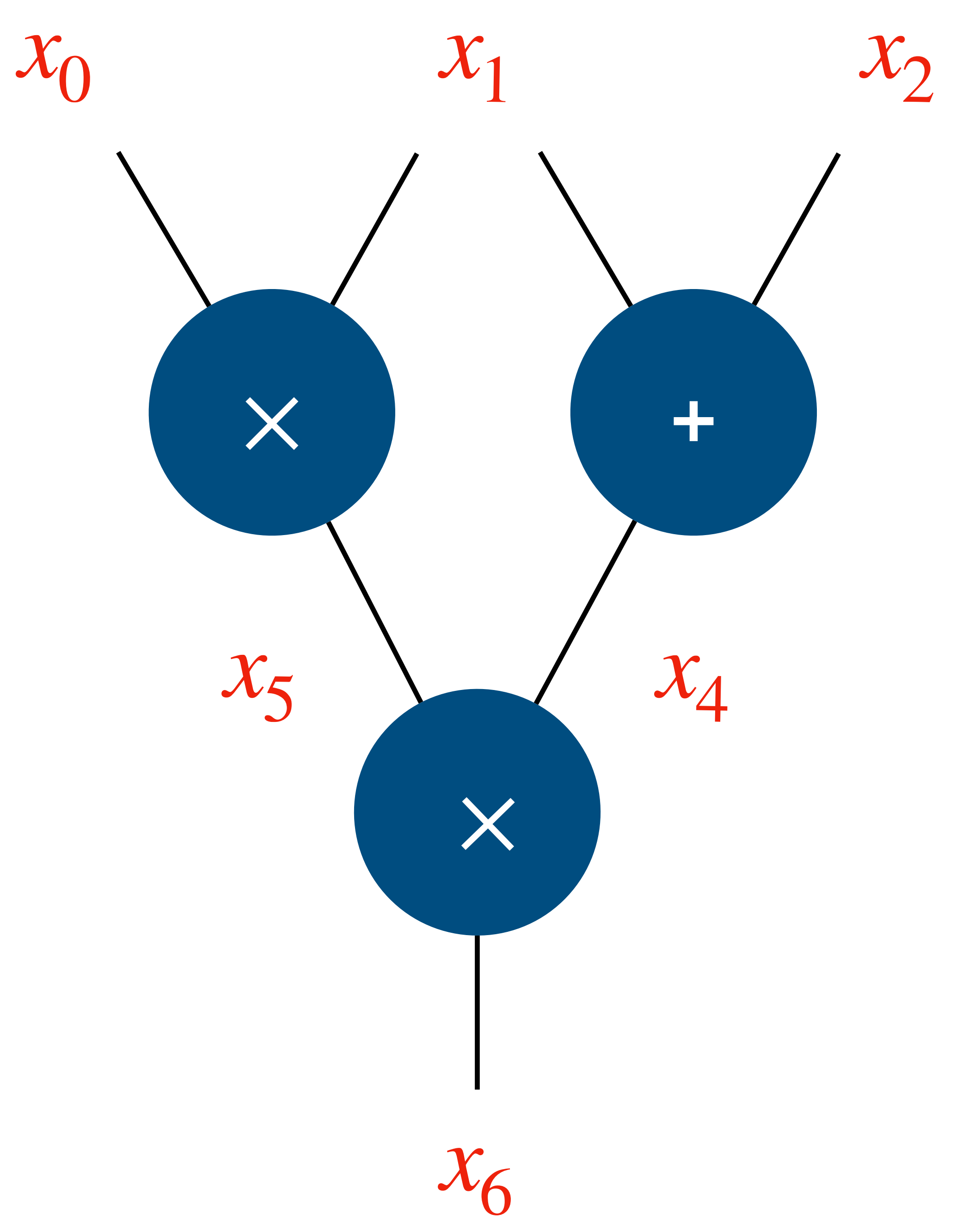
ZKP based on information theoretic MAC

- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$
  - x if x is shared in such a way
- Addition gate: MAC for $p(x, y) = a \cdot x + b \cdot y$
  - Prover locally computes $M_{p(x,y)} = a \cdot M_x + b \cdot M_y$
  - Verifier locally computes $k_{p(x,y)} = a \cdot k_x + b \cdot k_y$
- Multiplication gate: x, y, z verify if $xy = z$ holds



Zero Knowledge Proof

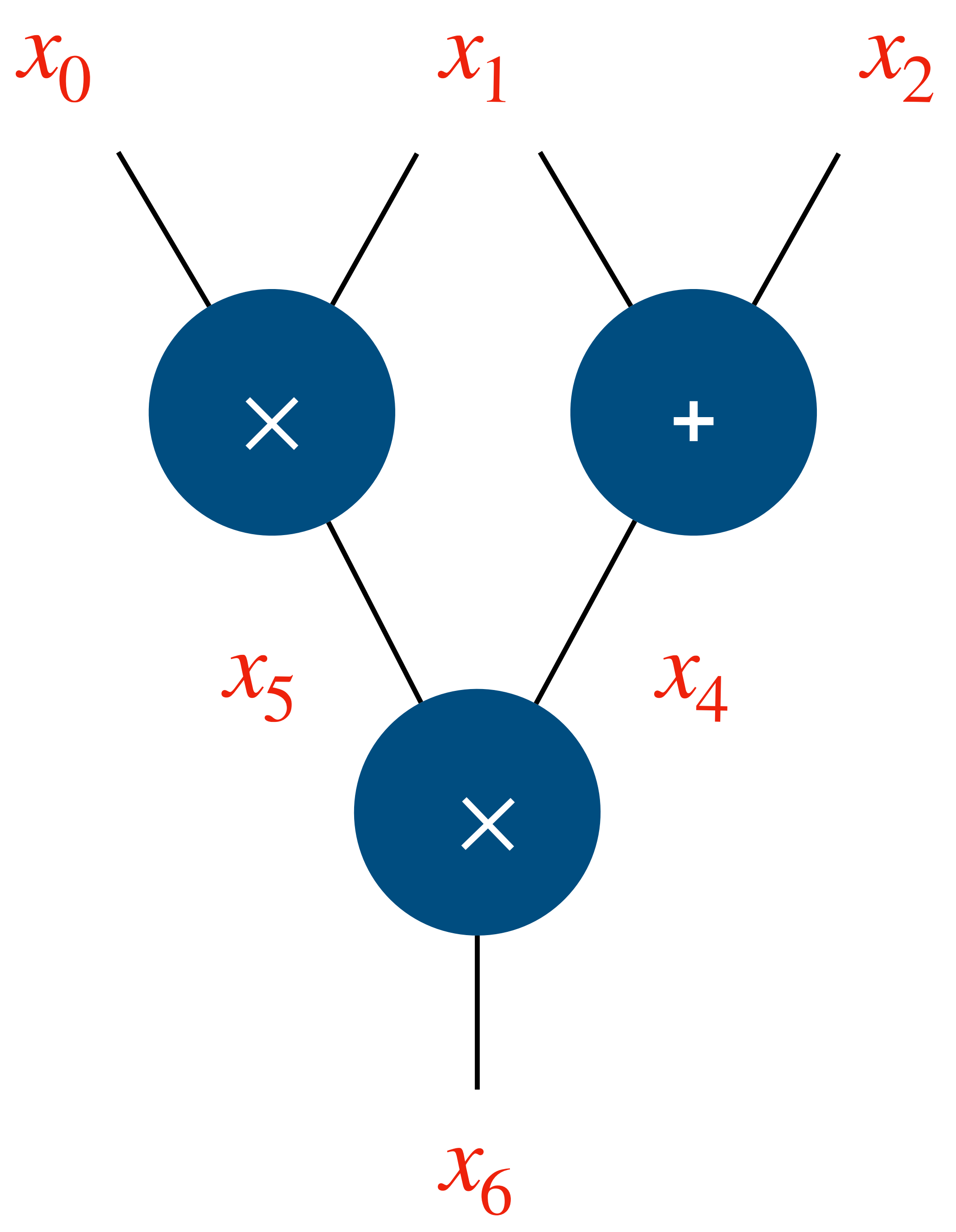
ZKP based on information theoretic MAC



- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$
    - x if x is shared in such a way
- Addition gate: MAC for $p(x, y) = a \cdot x + b \cdot y$
    - Prover locally computes $M_{p(x,y)} = a \cdot M_x + b \cdot M_y$
    - Verifier locally computes $k_{p(x,y)} = a \cdot k_x + b \cdot k_y$
- Multiplication gate: x, y, z verify if $xy = z$ holds
    - With small probability $k_x \cdot k_y - \Delta \cdot k_z = M_x \cdot M_y + (M_x \cdot y + M_y \cdot x - M_z) \cdot \Delta$ holds

Zero Knowledge Proof

ZKP based on information theoretic MAC



- Information-Theoretic MAC : $k_x = M_x + x \cdot \Delta$
  - x if x is shared in such a way
- Addition gate: MAC for $p(x, y) = a \cdot x + b \cdot y$
  - Prover locally computes $M_{p(x,y)} = a \cdot M_x + b \cdot M_y$
  - Verifier locally computes $k_{p(x,y)} = a \cdot k_x + b \cdot k_y$
- Multiplication gate: x, y, z verify if $xy = z$ holds
  - With small probability $k_x \cdot k_y - \Delta \cdot k_z = M_x \cdot M_y + (M_x \cdot y + M_y \cdot x - M_z) \cdot \Delta$ holds
  - Batching for multiple multiplication gates