Privacy-Preserving Reasoning

Ruzica Piskac

Yale University

Fifteenth Summer School on Formal Techniques
May 26, 2026

Overview

- Privacy-preserving ZK-SMT proof verification
  - ZK proofs
  - Resolution proofs
  - Verifying ZK-UNSAT in Boolean logic
  - Verifying ZK SMT proofs
- Privacy-preserving SAT solver
  - DPLL algorithm
  - Secure multi-party computation
  - ppSAT
- Ou: efficient programming framework for ZK protocols
  - Language design
  - Compiler for cloud-ready ZK proofs
- Conclusions

Overview

- Privacy-preserving ZK-SMT proof verification
  - ZK proofs
  - Resolution proofs
  - Verifying ZK-UNSAT in Boolean logic
  - Verifying ZK SMT proofs
- Privacy-preserving SAT solver
  - DPLL algorithm
  - Secure multi-party computation
  - ppSAT
- Ou: efficient programming framework for ZK protocols
  - Language design
  - Compiler for cloud-ready ZK proofs
- Applications

What is Verification?

- Will the program crash?
- Does it compute the correct result?
- Does it leak private information?
- How long does it take to run?
- How much power does it consume?
- Can we guarantee that an RL agent always avoids unsafe states?
- Can we formally verify safety properties (e.g., "the model never classifies a stop sign as a speed limit sign") of these models?

Verification – a super simple example

```
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
  ??
  return y
}
```

Verification – a super simple example

```
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
  val y = x - 2
  return y
}
```

Formula corresponding to the program:

$$\forall x.\ \forall y.\ x > 0 \ \wedge y = x - 2 \rightarrow y > 0$$

Verification – a super simple example

```
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
  val y = x - 2
  return y
}
```

Formula corresponding to the program:

$$\forall x.\ \forall y.\ x > 0 \ \wedge y = x - 2 \rightarrow y > 0$$

Preconditions

Verification – a super simple example

```
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
  val y = x - 2
  return y
}
```

Formula corresponding to the program:

$$\forall x.\ \forall y.\ x > 0 \ \wedge y = x - 2 \rightarrow y > 0$$

Program

Verification – a super simple example

```
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
  val y = 1
  return y
}
```

Formula corresponding to the program:

$$\forall x.\ \forall y.\ x > 0 \ \wedge y = 1 \rightarrow y > 0$$

Postconditions

Verification – a super simple example

More at https://ssft-sri.github.io/old/2022/

```
//: assume (x > 0)
def simple (Int x)
//: ensures y > 0
{
  val y = x - 2
  return y
}
```

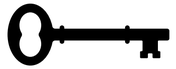
Formula corresponding to the program:

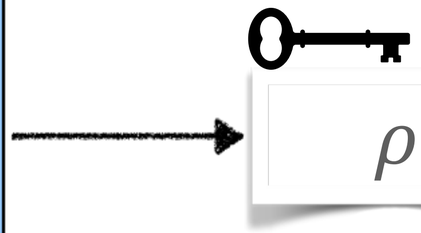
$$\forall x.\ \forall y.\ x > 0 \ \wedge y = x - 2 \rightarrow y > 0$$

Formula does not hold for input x = 1

A motivating example

```
1 int factorial(int x) {
2    int y = 1;
3    int z = 0;
4    while (z != x) {
5       z = z + 1;
6       y = y * z;
7    }
8    return y;
9 }
```





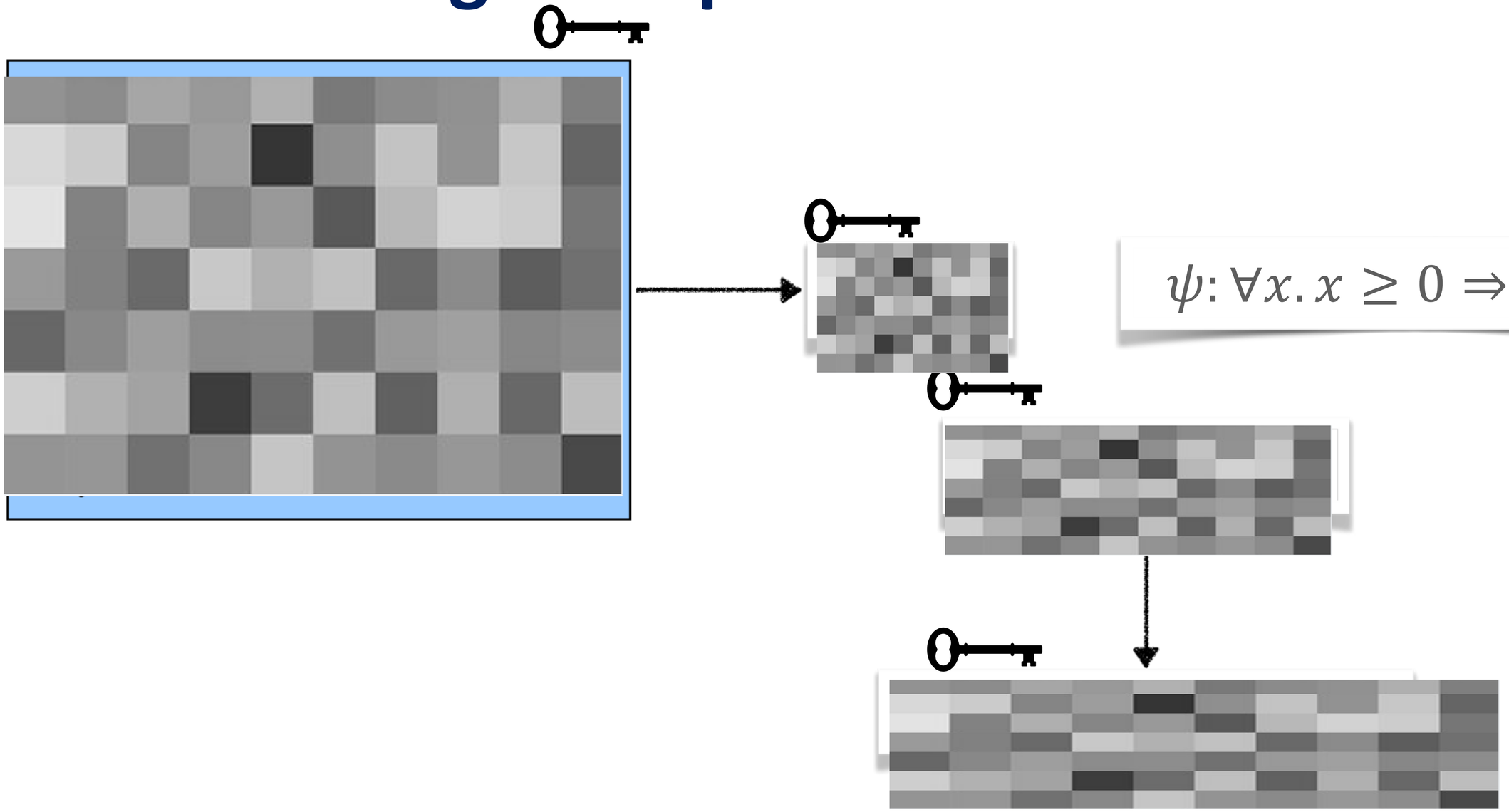
ρ

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

$\rho \wedge \neg\psi$

Resolution proof

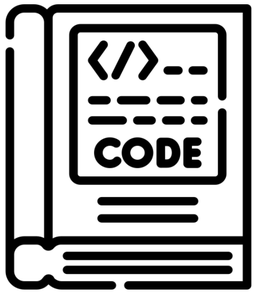
A motivating example



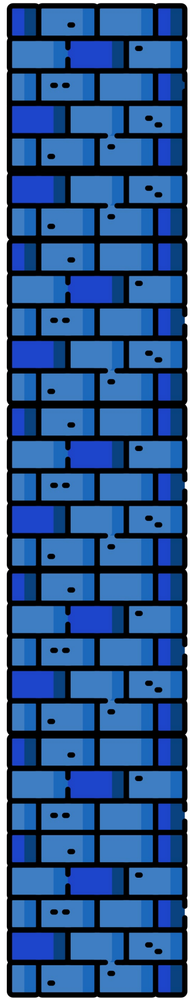
$\psi: \forall x. x \geq 0 \Rightarrow y = x!$

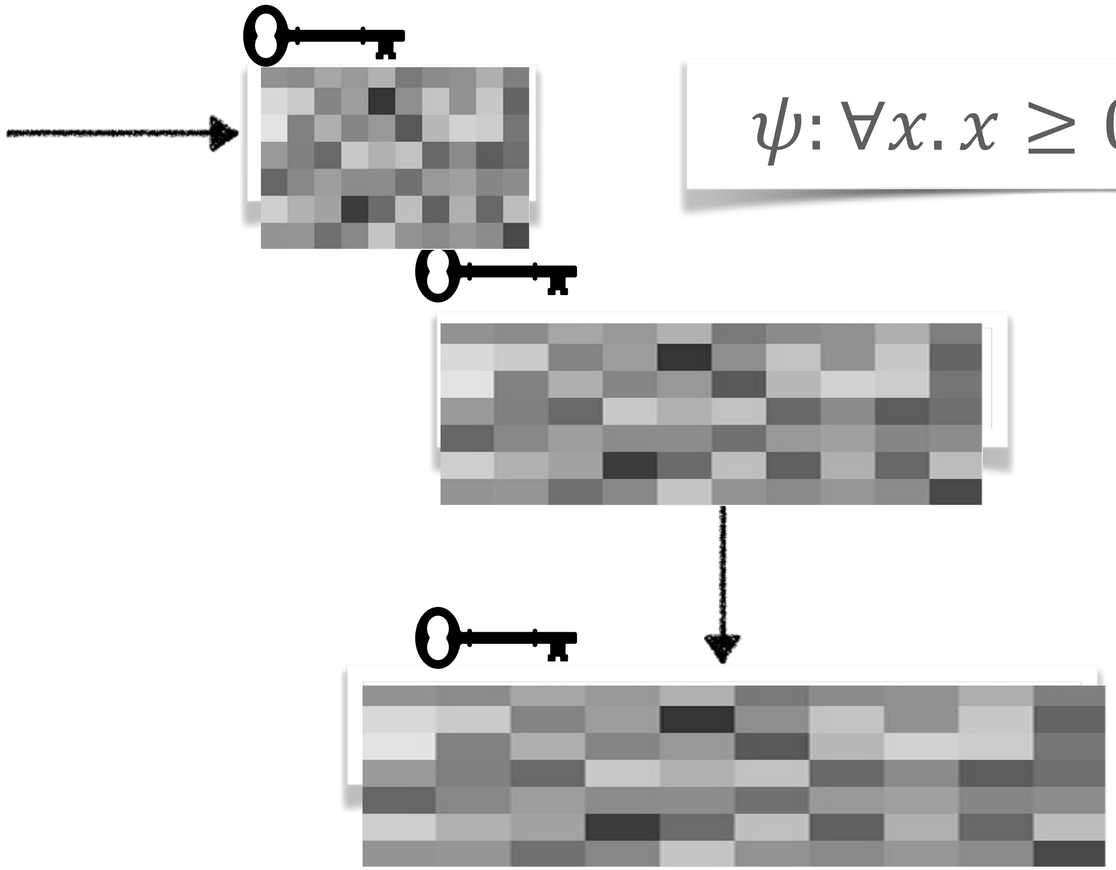
A motivating example

Proprietary software continues to **dominate** the market.



Source Code





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

Another motivating example

Policies Can Contain Sensitive Details

Financial Policy

…

[PERMIT Transaction < $10,000 OR Manually Reviewed]

…

Transfers of $10,000 or more must be manually reviewed by a bank employee for approval

Policy

Financial Policy

…

[PERMIT Transaction < $10,000 OR Manually Reviewed]

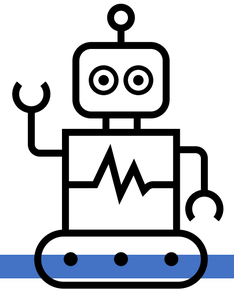
…

Compliance Check

Decision:

DENY

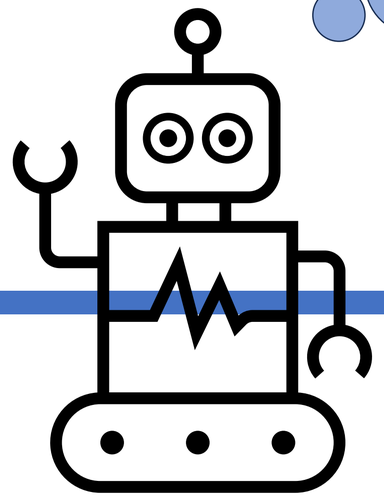
Manual Review Required



Agent

Agent: Transfer $12,000 please!

- My request was denied, let me check the policy…
- The policy clearly states transactions of at least $10,000 require manual review, so I cannot make progress…
- Wait, if I make 2 smaller $6,000 transactions, I can avoid this!



Policy

Financial Policy

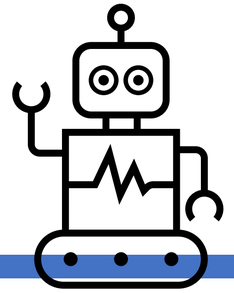
…

[PERMIT Transaction < $10,000 OR Manually Reviewed]

…

Compliance Check

Access Decision:
APPROVE



Agent

Agent: Transfer $6,000 please!

…

Agent: Transfer $6,000 please!

What Can We Do About This?

Financial Policy

…

[PERMIT Transaction < $10,000 OR Manually Reviewed]

…

The agent circumvented the policy because it could see inside!

Solution: if policies were private, we could prevent this.

Question: how to check compliance?

What Can We Do About This?

Financial Policy

…

[PERMIT Transaction < $10,000 OR Manually Reviewed]

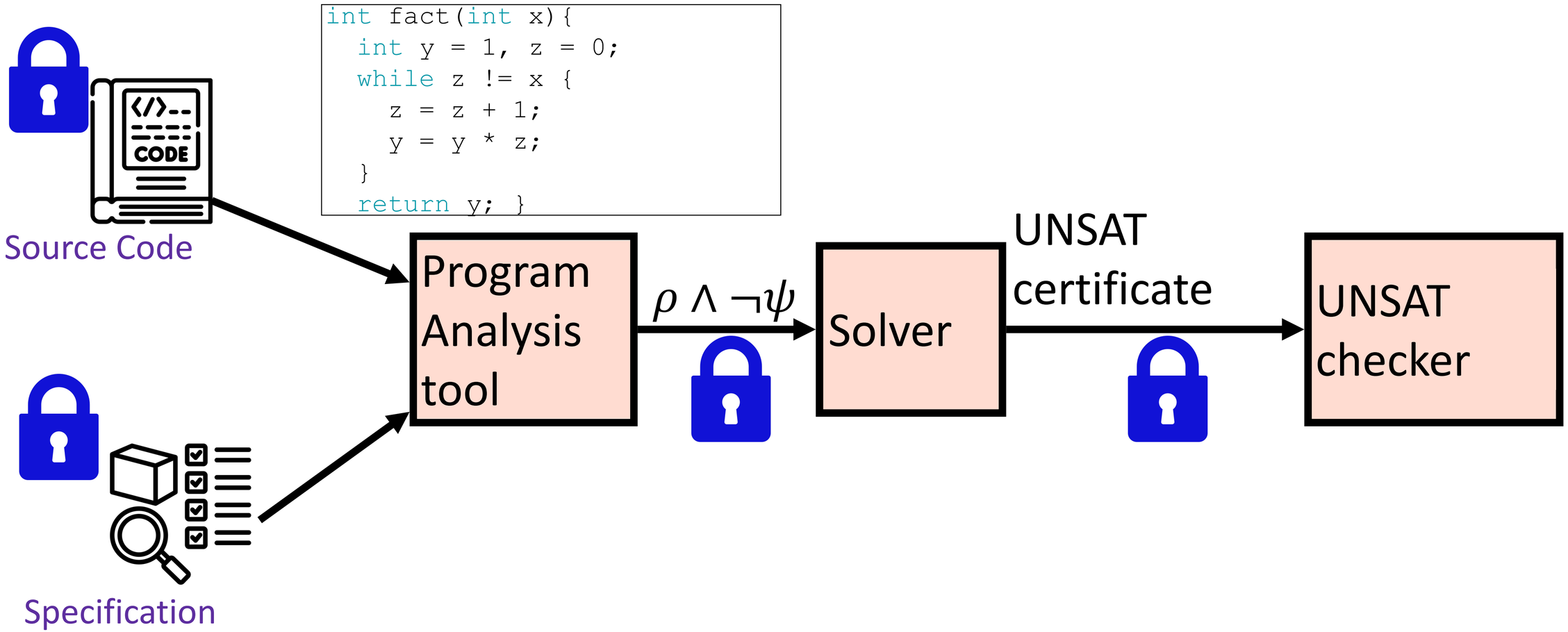
…

Solution: if policies were private, we could prevent this.

Question: how to check compliance?

Privacy-preserving reasoning

Privacy-Preserving Formal Verification



Zero-Knowledge Proofs

[Goldwasser, Micali, Rackoff, 1985]

Goldwasser & Micali - Turing award 2012

Zero-Knowledge Proofs

[Goldwasser, Micali, Rackoff, 1985]

Goldwasser & Micali - Turing award 2012

Main Idea:
Prove that
I **can** prove it

https://rdi.berkeley.edu/zkp-course/assets/Lecture1-2023-slides.pdf

Zero Knowledge Proof

Simple example

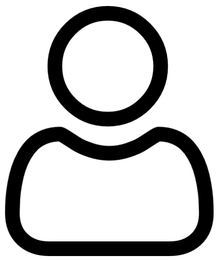
I know a method to tell coke or pepsi!

show me how otherwise I won't believe

I will prove that I know without showing how

Zero Knowledge Proof

Simple example

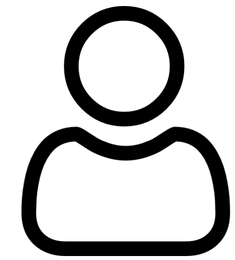


Prover

I know a method to tell coke or pepsi!

show me how otherwise I won't believe

I will prove that I know without showing how



Verifier

Zero-Knowledge Proofs

[Goldwasser, Micali, Rackoff, 1985]

Goldwasser & Micali - Turing award 2012

Correct and **private** computations are needed **everywhere**

(

Authentication Systems

Smart contracts

(e.g. ZK-rollups for computation efficiency)

zkApps

Privacy in digital currencies

(e.g. Zcash)

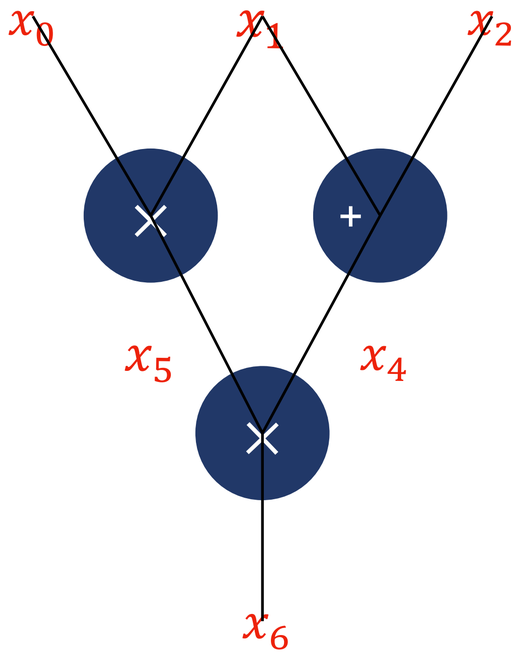
)

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.

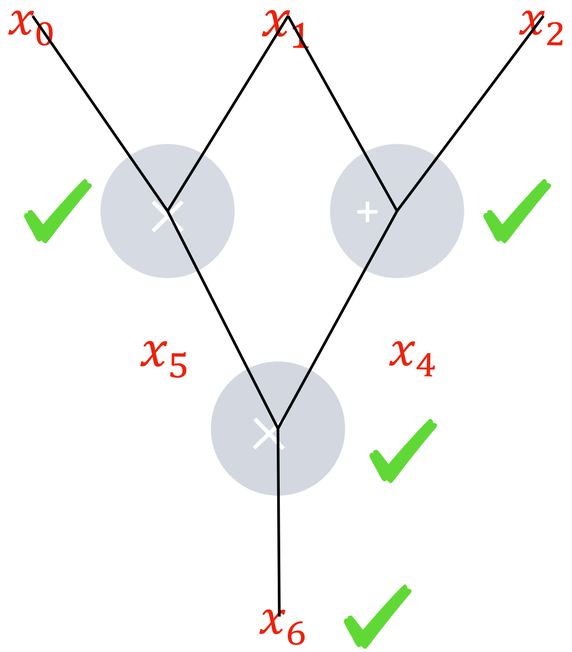
- Prover might cheat
- Verifier tries to learn w



A public circuit C

Zero Knowledge Proof

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



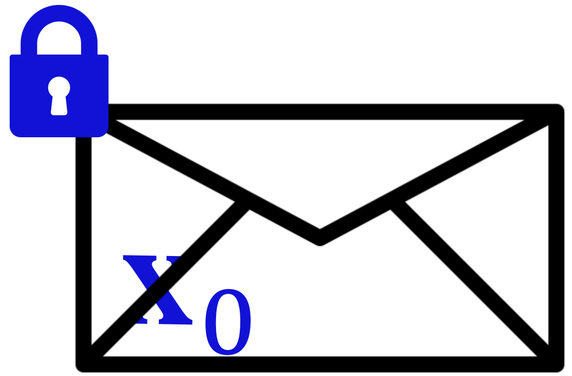
Gate-by-gate paradigm [Beaver et al. 90]:

- Prover authenticates the value over all wires
- Verifier checks if input and output of each gate is consistent over the ***ciphertext***
- Prover reveal the value of output of the circuit

Proving UNSAT of PRIVATE Formulae

Background: Commitment

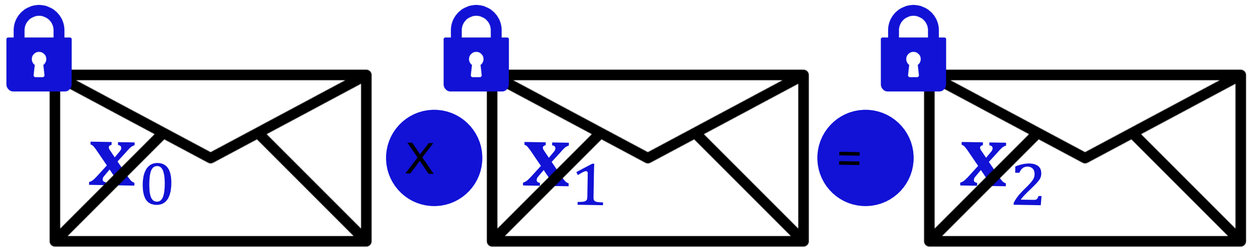
- **Commitment: ciphertext that**
  - **Hide** underlying value



Proving UNSAT of PRIVATE Formulae

Background: Commitment

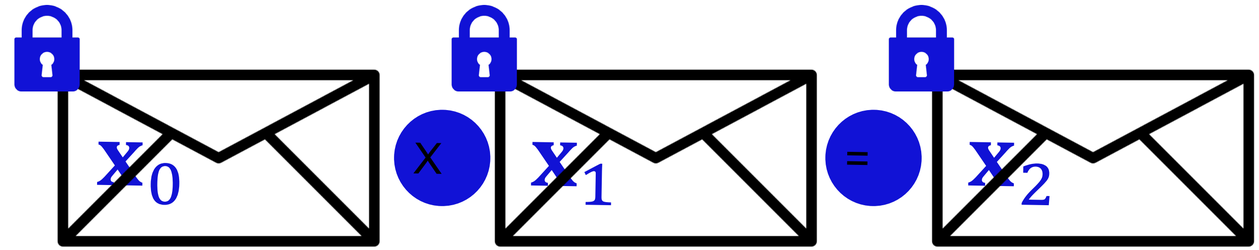
- **Commitment: ciphertext that**
  - **Hide** underlying value
  - Enable verifier to **check the relations**

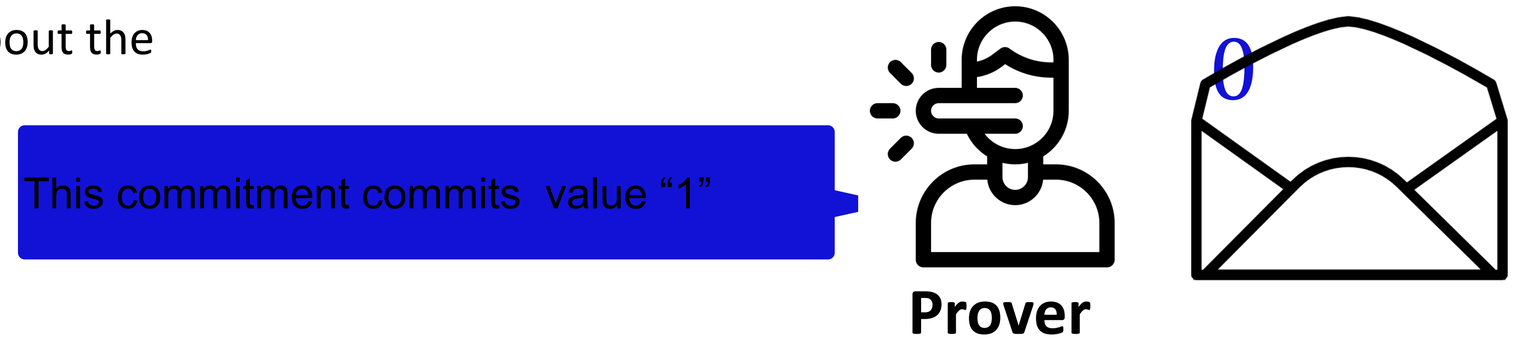


Proving UNSAT of PRIVATE Formulae

Background: Commitment

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





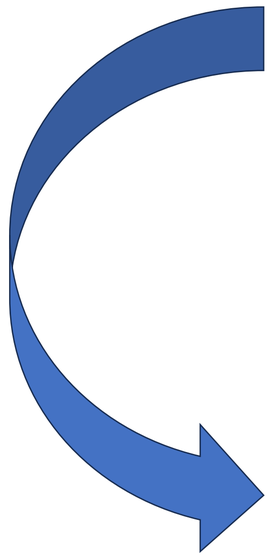
“Commitment” for Formal Methods People

- **Commitment: ciphertext that** ϵ
  - **Hide** underlying value
  - Because of polynomial encoding, $\epsilon(x+y) = \epsilon(x) + \epsilon(y)$ and $\epsilon(x*y) = \epsilon(x) * \epsilon(y)$

$(x - 2)(x-3)$

$\{\epsilon(2), \epsilon(3)\}$

$\{010010101, 00011001\}$



- **Prover** knows a resolution proof for a formula
  - Needs to convince the verifier that the formula is unsatisfiable
  - Keeps information about the formula and proof private
- **Verifier:**
  - Validates prover’s claim about the proof

Refutation Proof

Resolution Proof in Propositional Logic

$$\frac{C_1 = A \vee B, \qquad C_2 = G \vee \neg B}{C_r = A \vee G}$$

- Resolution: a rule of inference for formulas in conjunctive normal form
- New clause (resolvent) can be inferred from two other clauses.
- Construct a resolution refutation proof for a given unsatisfiable formula
    - The resolution rule alone is sufficient to derive false from the given clauses when their conjunction is unsatisfiable.
    - A resolution proof consists of a list of resolvents and how these resolvents could be obtained using the input formula and the prior intermediate resolvents.

Refutation Proof

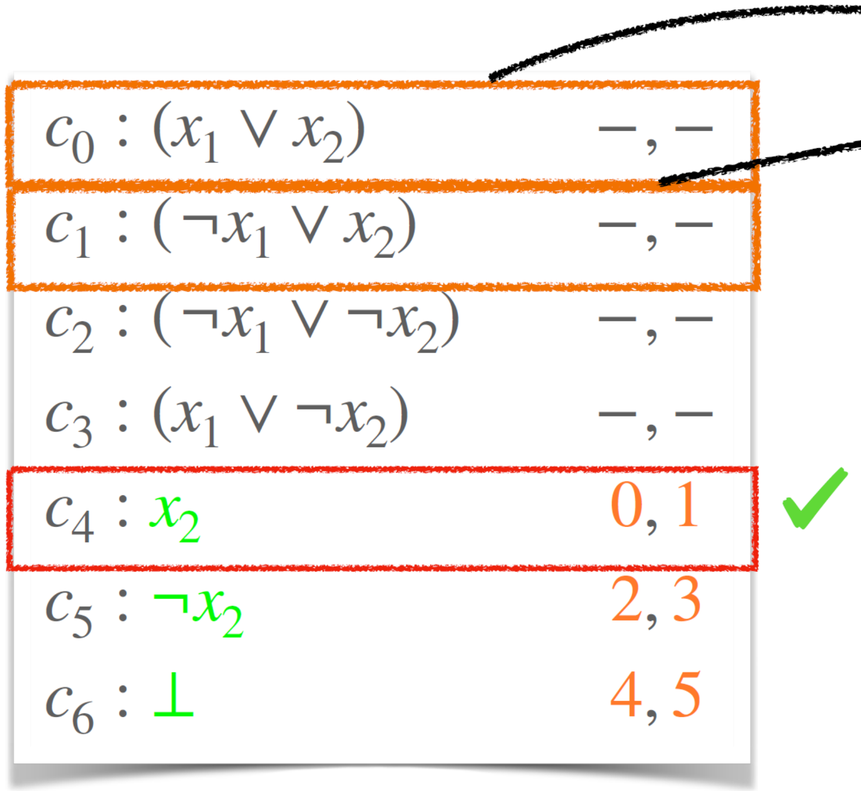
Resolution Proof in Propositional Logic

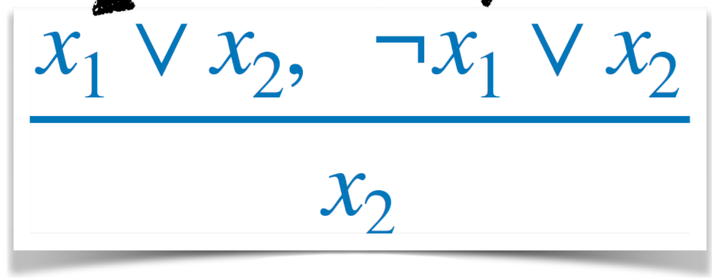
$$\frac{C_1 = D_1 \vee B, \qquad C_2 = D_2 \vee \neg B}{C_r = D_1 \vee D_2}$$

- Resolution: a rule of inference for formulas in conjunctive normal form
- New clause (resolvent) can be inferred from two other clauses.
- Construct a resolution refutation proof for a given unsatisfiable formula
  - The resolution rule alone is sufficient to derive false from the given clauses when their conjunction is unsatisfiable.
  - A resolution proof consists of a list of resolvents and how these resolvents could be obtained using the input formula and the prior intermediate resolvents.

Refutation Proof

Resolution Proof in Propositional Logic

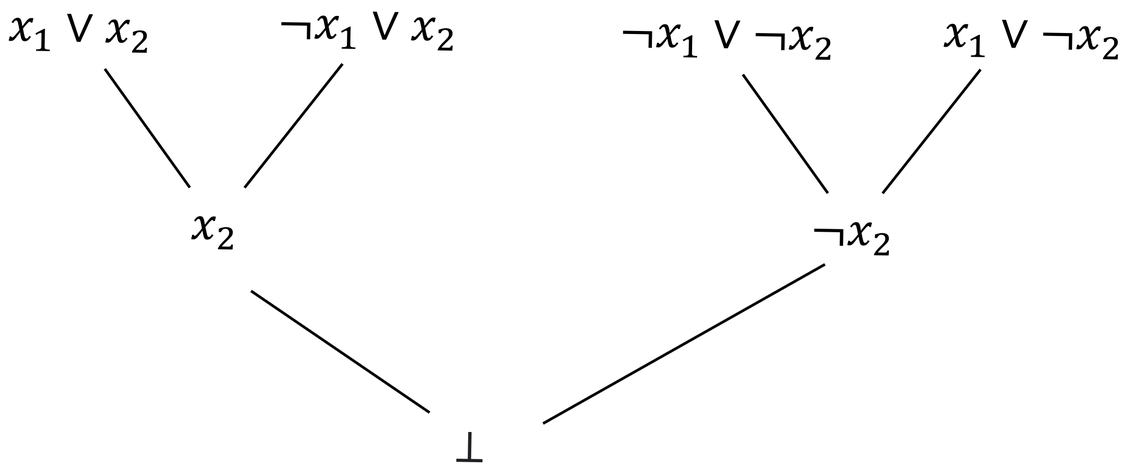




- Fetch two clauses used to derived the resolvent
- Check the application of resolution rule is correctly executed

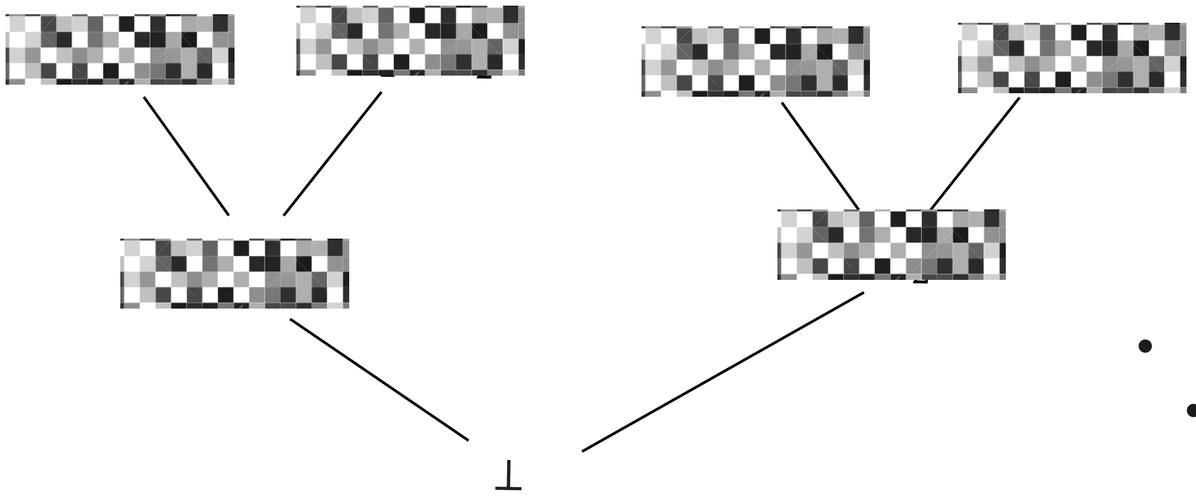
Resolution Proof

An example



Resolution Proof

An example

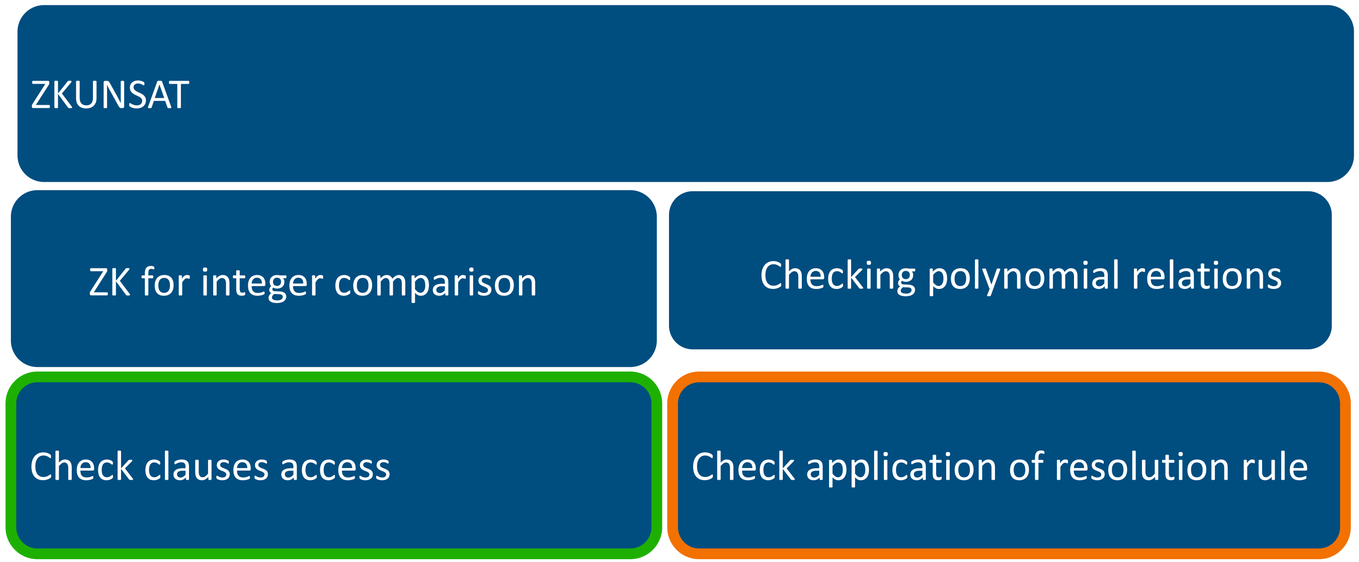


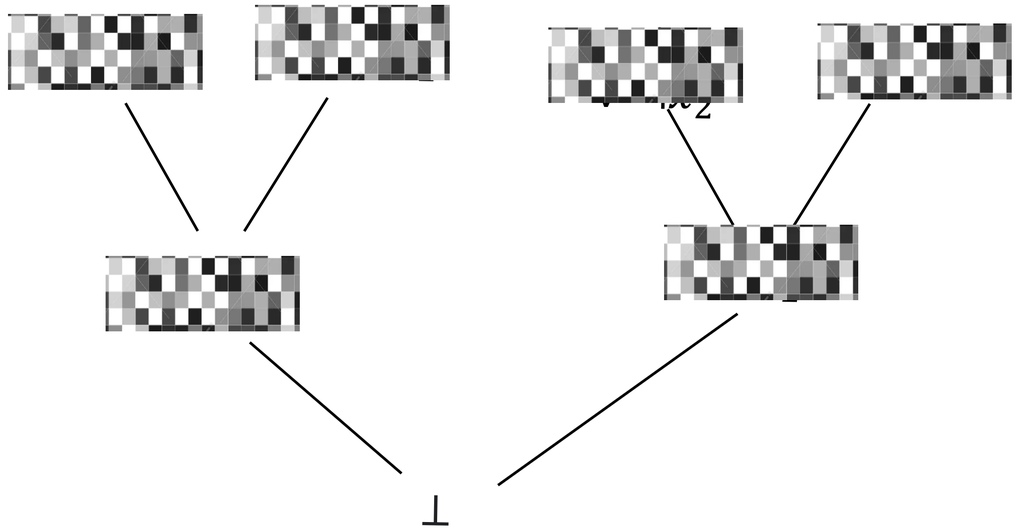
- **Prover** knows a resolution proof for a formula
  - Needs to convince the verifier that the formula is unsatisfiable
  - Keeps information about the formula and proof private

- **Verifier:**
  - Validates prover's claim about the proof

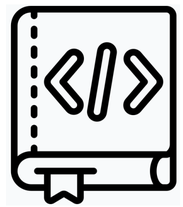
Unsatisfiability in Zero Knowledge Proof

- Technique challenges and design overview



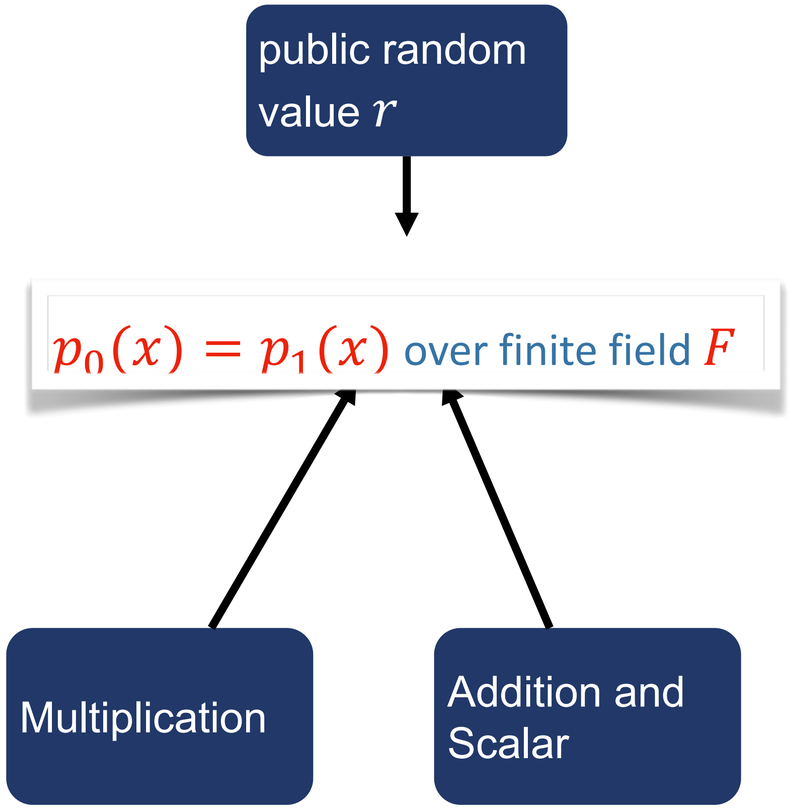


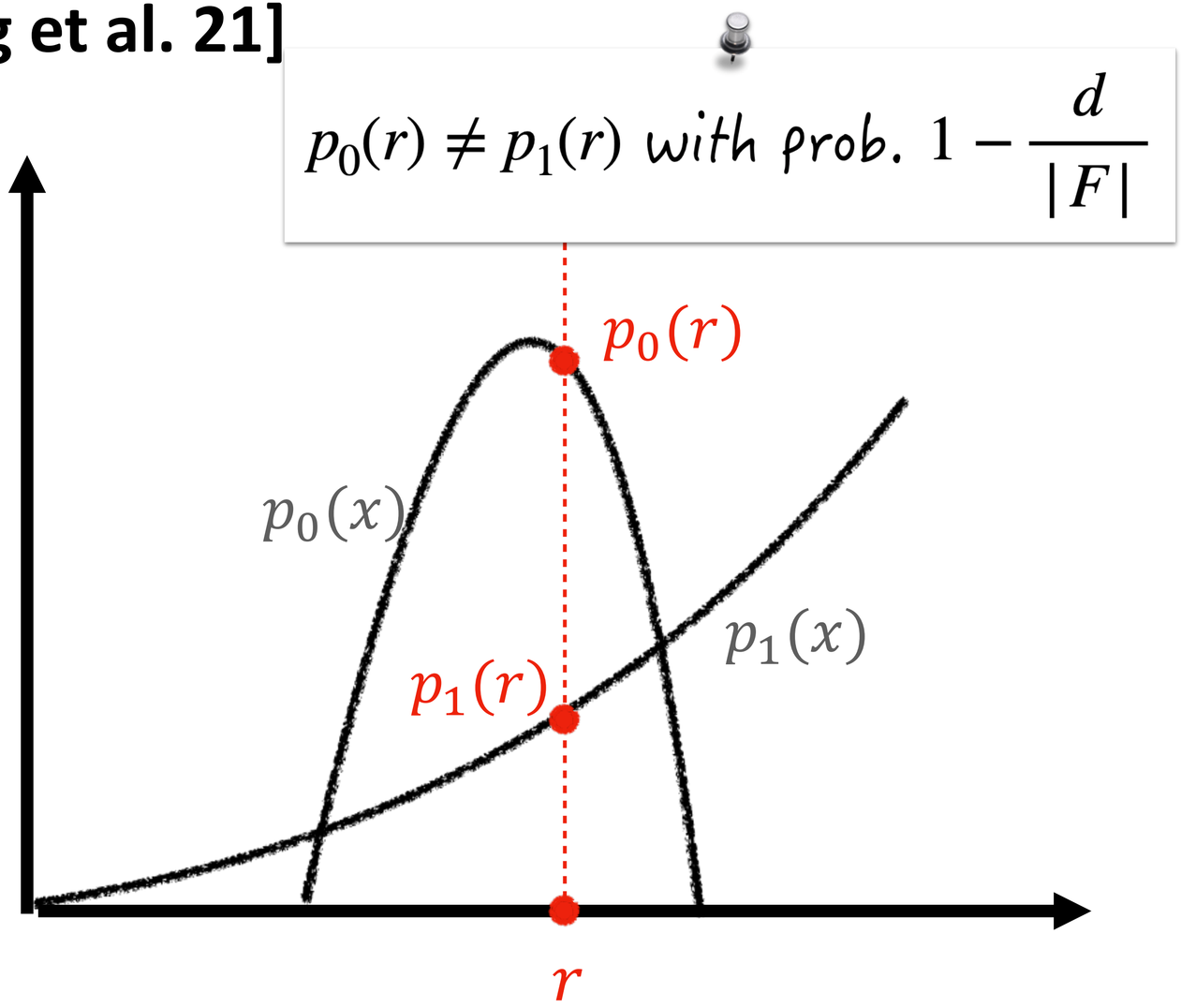
Efficient library for checking relations between polynomials in ZK



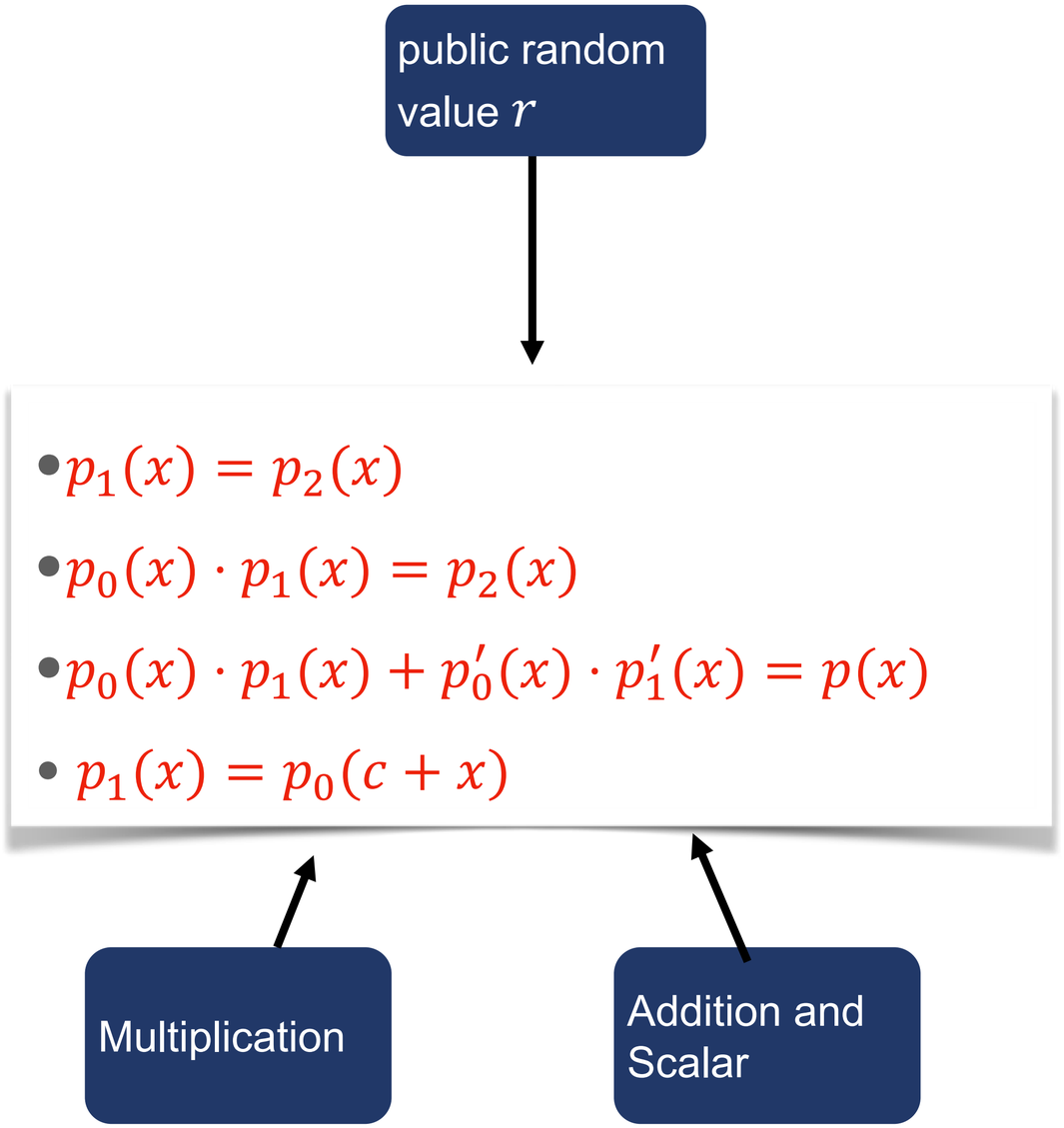
Zero Knowledge Proof

Knowledge for polynomials[Yang et al. 21]





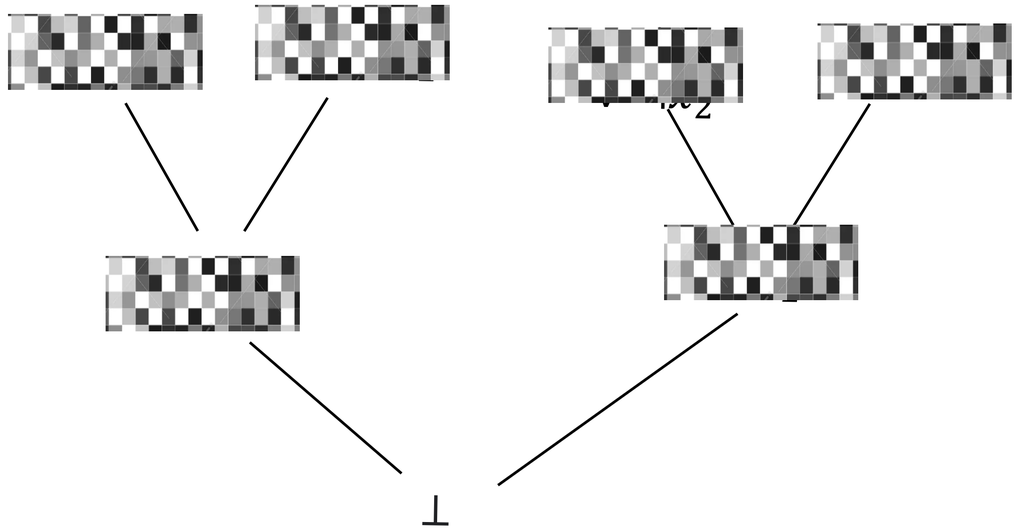
Zero Knowledge Proof



Unsatisfiability in Zero Knowledge Proof

- Technique challenges and design overview

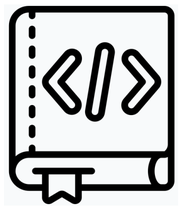
Check application of resolution rule



- **Prover** knows a resolution proof for a formula
  - Encodes clauses as polynomials
  - Every resolution step is checking a relationship between polynomials

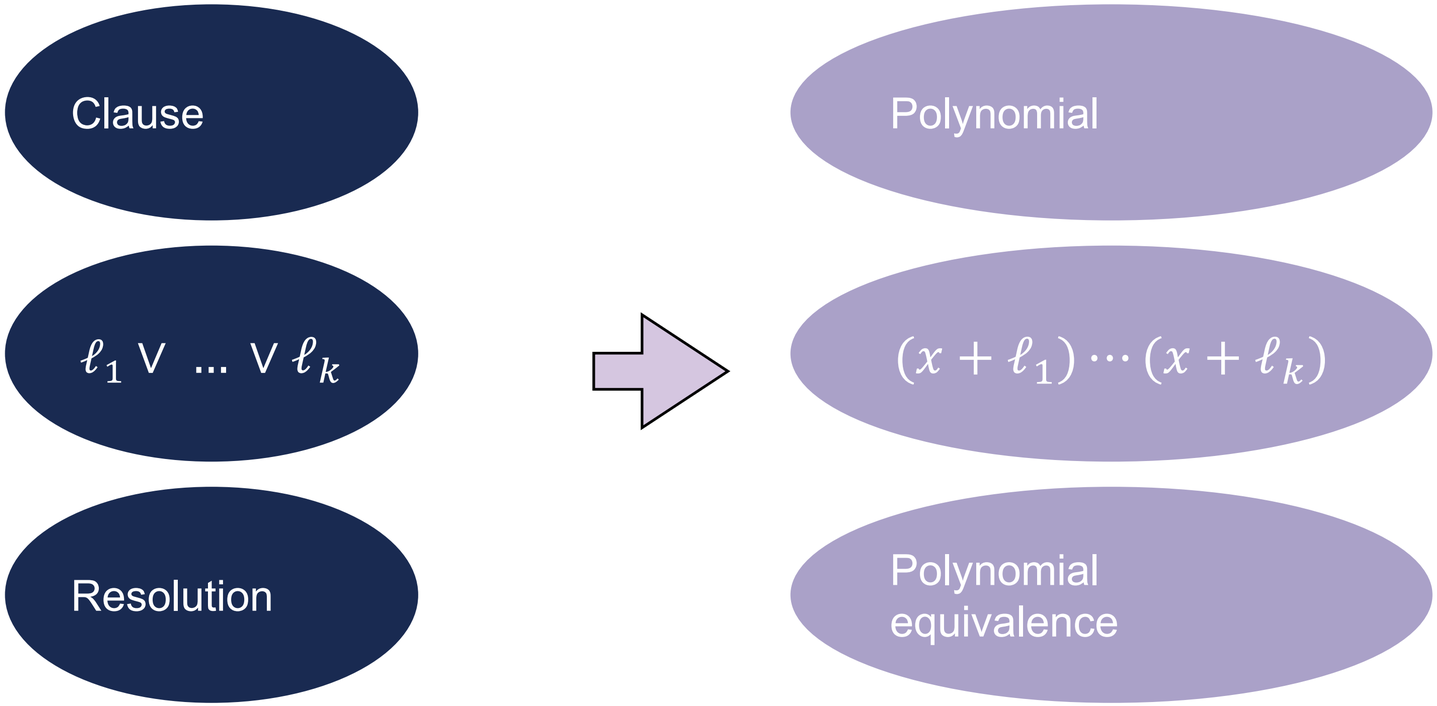
- **Verifier:**
  - Validates prover's claim about the proof

Efficient library for checking relations between polynomials in ZK



Encoding Scheme

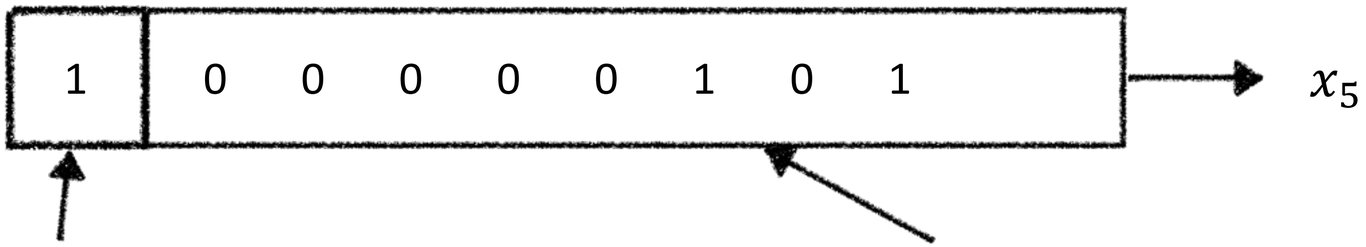
- Polynomial-based approach



Encoding Scheme

Polynomial-based approach

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



- Any Encoding scheme that satisfies
  - Injective
  - $\epsilon(\ell) + \epsilon(\neg\ell) = c$

Encoding Scheme

Polynomial-based approach

- Clauses then are encoded as polynomial over $\mathbf{F_{2^k}}$
  - $c = (\ell_0 \vee \ell_1 \vee \cdots \vee \ell_d) \rightarrow p_c(x) = (x + \epsilon(\ell_0)) \cdots (x + \epsilon(\ell_d))$
  - Ex. $c_A = (v_1 \vee v_2) \rightarrow p_A = (x + 2^{k-1} + 1)(x + 2^{k-1} + 2)$

Encoding Scheme

- Polynomial-based approach

$$\frac{c_A = v_1 \vee v_2 \vee v_3, \qquad c_B = v_4 \vee \neg v_3}{c_R = v_1 \vee v_2 \vee v_4}$$

$$\frac{p_A = (x + \epsilon(v_1))(x + \epsilon(v_2))(x + \epsilon(v_3)) \qquad p_B = (x + \epsilon(v_4))\,(x + \epsilon(\neg v_3))}{p_R = (x + \epsilon(v_1))(x + \epsilon(v_2))(x + \epsilon(v_4))}$$

  - Checking that the resolution step is correct reduces to checking that:

    $p_A | p_R \cdot (x + \epsilon(v_3))$ and $p_B | p_R \cdot (x + \epsilon(\neg v_3))$

Encoding Scheme

Polynomial-based approach

The prover needs to convince the verifier

$p_{c_a} | p_{c_r} \cdot (x + \epsilon(\ell_p))$ and $p_{c_b} | p_{c_r} \cdot (x + \epsilon(\neg\ell_p))$

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

$$\frac{c_a = v_1 \vee v_2 \qquad c_b = v_1 \vee \neg v_2}{c_r = v_1}$$

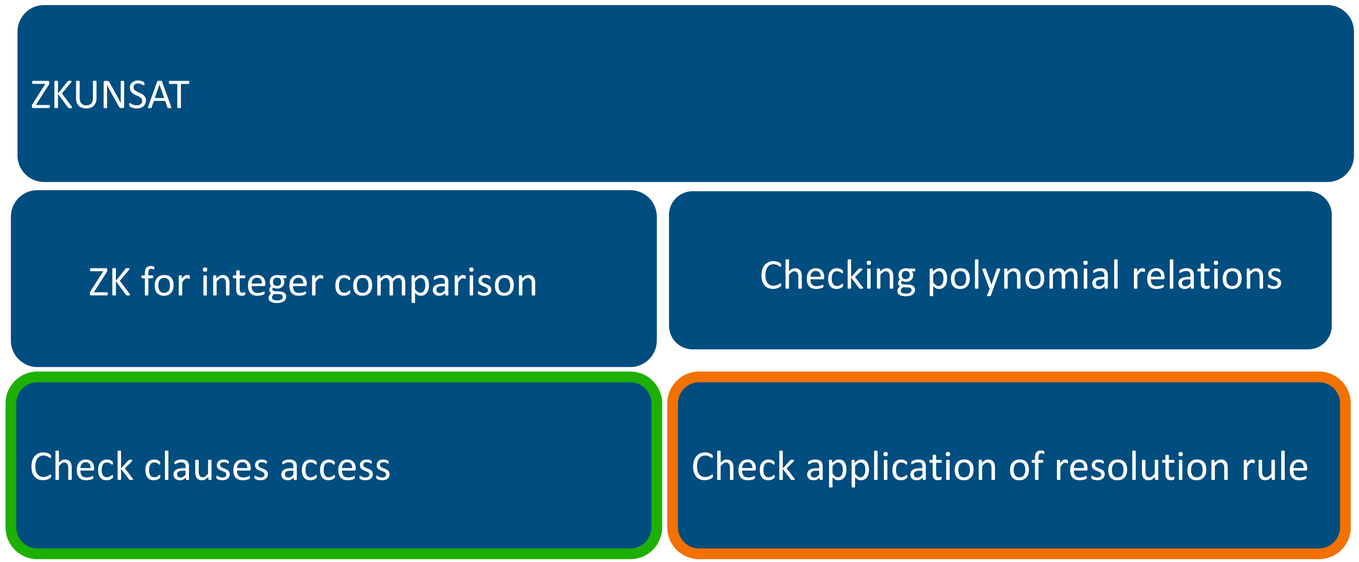
Encoding Scheme

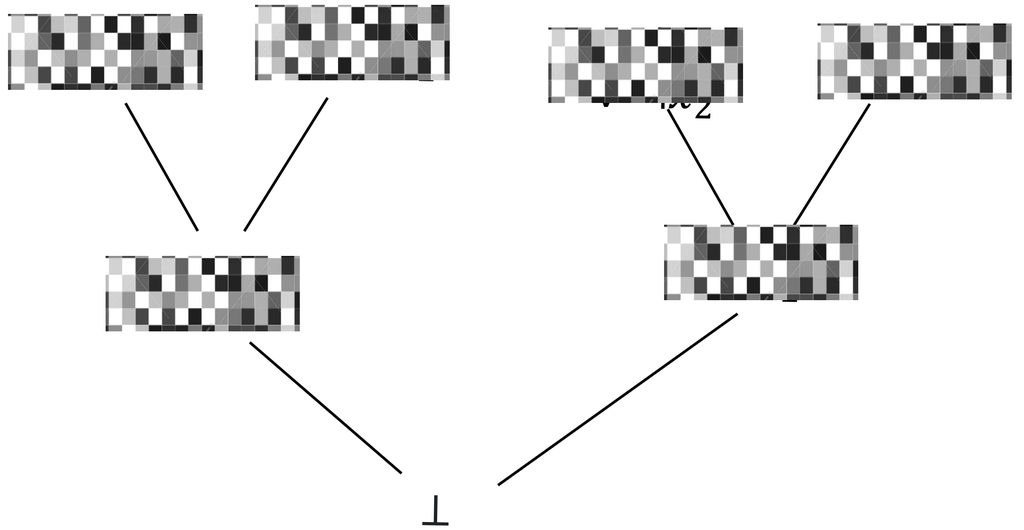
Polynomial-based approach

- $C_a \subseteq C_r \cup \{\ell_p\}, C_b \subseteq C_r \cup \{\neg\ell_p\}$
  - 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
    - $p_{c_a} \cdot w_a = p_{c_r} \cdot (x + \epsilon(\ell_p))$
    - $p_{c_b} \cdot w_b = p_{c_r} \cdot (x + \epsilon(\neg\ell_p))$

Unsatisfiability in Zero Knowledge Proof

- Technique challenges and design overview





Efficient library for checking relations between polynomials in ZK