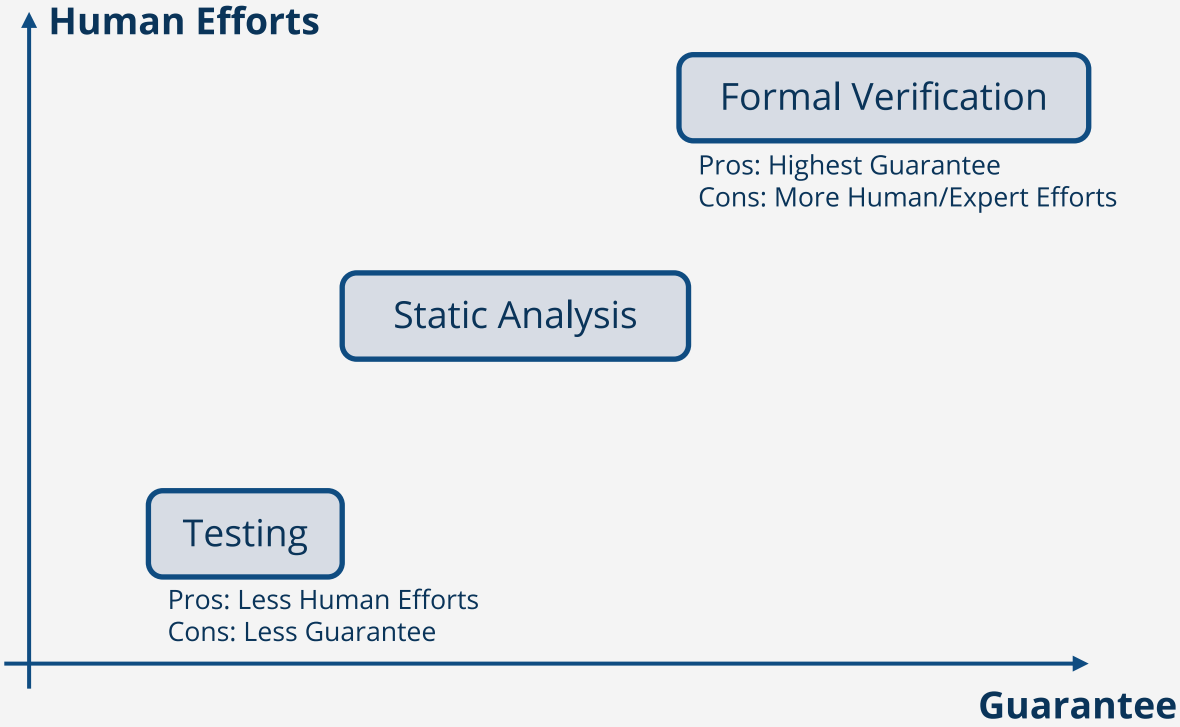
Formal Verification for ZK Circuits

Yanju Chen & Junrui Liu
University of California, Santa Barbara
& Veridise Inc.

Contents

Overview

Verification Techniques Spectrum

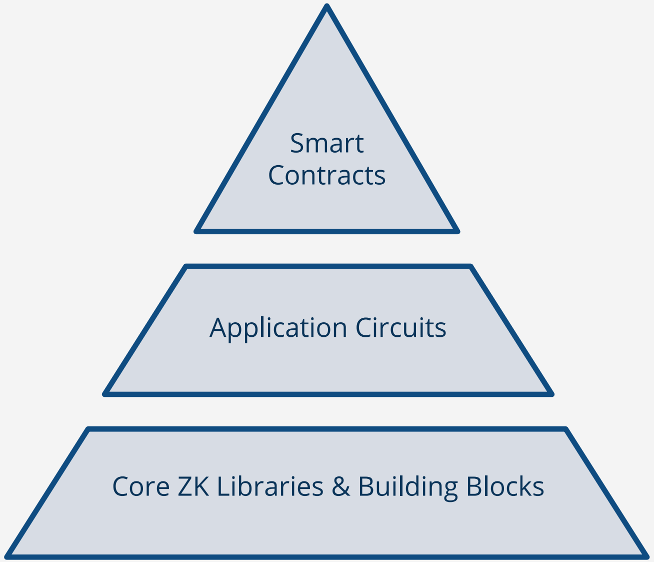


ZK Circuit Verification

Why?
- Application Security
- Library Reusability
- ...

Challenges?
- Scalability
- Coverage
- Extensibility
- ...

What?
- Functional Correctness
- Uniqueness Property
- ...



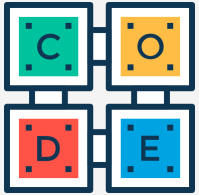
ZK Circuit Verification: Current Roadmap

Core Library Circuits (Manual)

- A tiny but critical and frequently used set of circuit building blocks (e.g., circomlib)
- Formal verification using interactive theorem proving
- This will provide the highest guarantee, but requires manual/expert efforts

Application Circuits (Automated / Semi-Automated)

- Majority of the application circuits belong to this category
- Automatically translating program into machine checkable formula
- Abstract level static analysis to over-approximate the range of each variable/wire



Functional Correctness

"Do the constraints correctly represent user intent?"

$\text{int } x[3]; y = x[j]$

"y should be sampled from array x"

```
pragma circom 2.0.0;

template test() {
    signal input x[3], j;
    signal output y;
    signal i0, i1, i2;
    signal y0, y1, y2;

    i0 <-- j==0? 1:0;
    i0 * (j-0) === 0;

    i1 <-- j==1? 1:0;
    i1 * (j-1) === 0;

    i2 <-- j==2? 1:0;
    i2 * (j-2) === 0;

    y0 <== i0*x[0];
    y1 <== i1*x[1];
    y2 <== i2*x[2];

    y <== y0 + y1 + y2;
}

component main = test();
```

Example Circom Snippet
Written by User

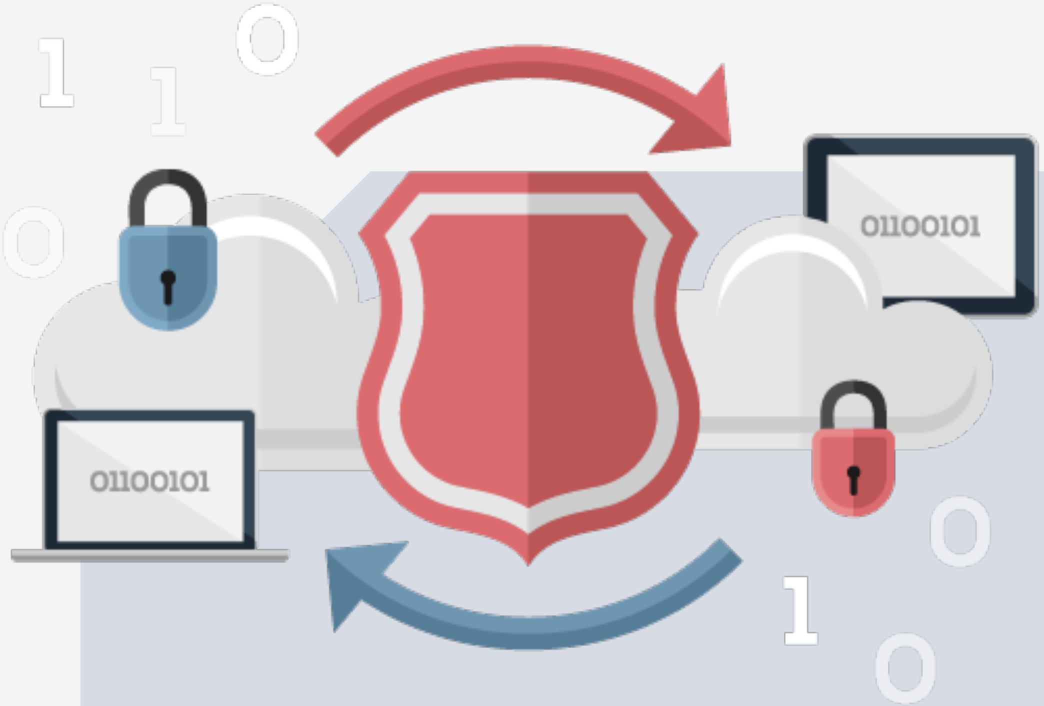
```
pragma circom 2.0.0;

template test() {
    signal input x[3], j;
    signal output y;
    signal i0, i1, i2;
    signal y0, y1, y2;

    i0 <-- j==0? 1:0;
    i0 * (j-0) === 0;
    i0 * (i0-1) === 0;

    i1 <-- j==1? 1:0;
    i1 * (j-1) === 0;
    i1 * (i1-1) === 0;

    i2 <-- j==2? 1:0;
    i2 * (j-2) === 0;
    i2 * (i2-1) === 0;

    i0+i1+i2 === 1;

    y0 <== i0*x[0];
    y1 <== i1*x[1];
    y2 <== i2*x[2];

    y <== y0 + y1 + y2;
}

component main = test();
```

$y = x[0] \,||\, y = x[1] \,||\, y = x[2]$

Query/Specification

Formal Verification for Core Library Circuits (Junrui)

Formal Verification for Application Circuits



Vulnerability/Bug Detection in Application Circuits

Application Circuits

- Large (>5000 LOC, millions of constraints)
- Contains non-library constraints

What to Verify / Sources of Bugs

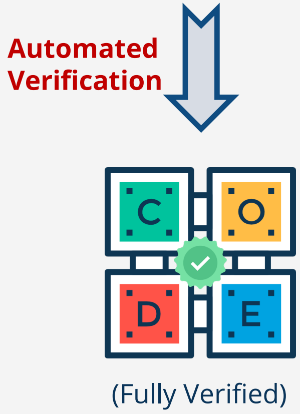
- Functional Correctness

  "I think what I wrote is all I want! ... no?"

- Uniqueness Property

  "I think I've already included all range checks! ...probably?"

Automated Verification of Functional Correctness



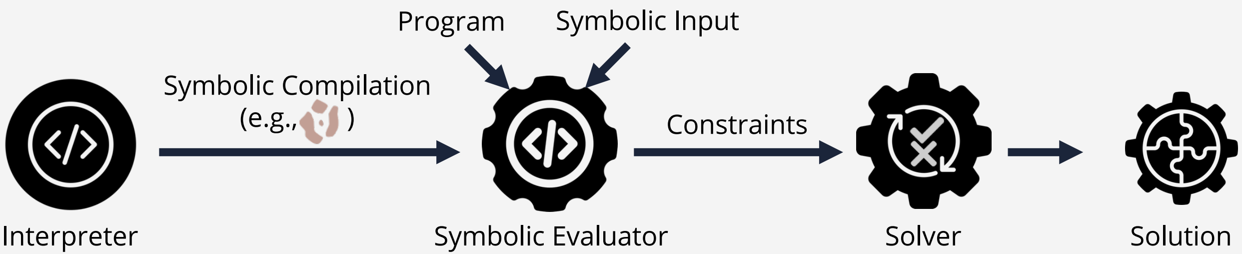
Automated Verification Techniques

- Symbolic Execution
- Abstract Interpretation
- ...

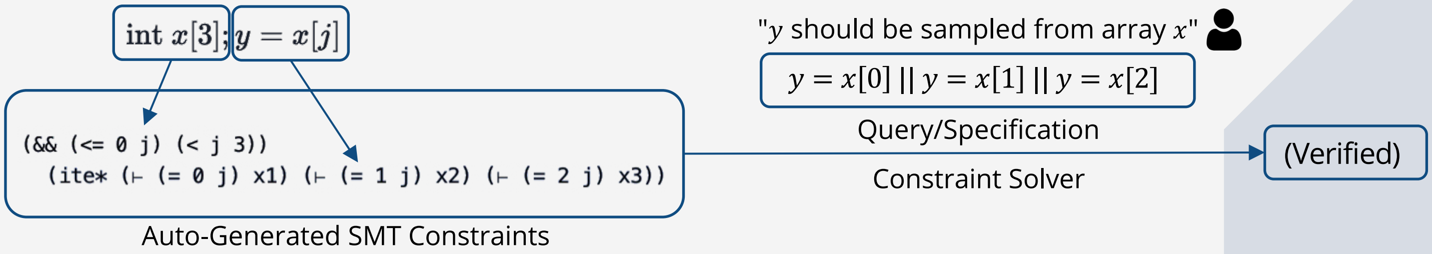
Picus is based on symbolic execution.
(https://github.com/chyanju/Picus)

Symbolic Execution

An interpreter follows the program, assuming symbolic values for inputs rather than obtaining actual inputs as normal execution of the program would.



Example:



Uniqueness Property

ref: https://0xparc.org/blog/ecne

Weak Verification (IO Uniqueness)

- This tests if, given the input variables in a QAP, the output variables have uniquely determined values.
- Example: $x[1]$, $x[2]$, $x[3]$ and j are fixed, y is queried.

Witness Uniqueness

- This tests if all the witness variables that appear in all equations, and not just input and output variables, collectively are uniquely determined.
- Example: $x[1]$, $x[2]$, $x[3]$ and j are fixed, $i_?$, $y_?$ and y are queried.

Strong Uniqueness

- This tests if the QAP is exactly equivalent to a formal mathematical specification.
- (Similar to function correctness)

$$
\text{int } x[3]; y = x[j] \xrightarrow{\text{R1CS}} \begin{cases} \text{input } x[1], x[2], x[3], j \\ \text{output } y \\ i_1 \cdot (j-0) = 0 \\ i_2 \cdot (j-1) = 0 \\ i_3 \cdot (j-2) = 0 \\ i_1 + i_2 + i_3 = 1 \\ y_1 = i_1 \cdot x[1] \\ y_2 = i_2 \cdot x[2] \\ y_3 = i_3 \cdot x[3] \\ y = y_1 + y_2 + y_3 \end{cases}
$$

Automated Verification of Uniqueness Property

Related Work: Ecne (https://github.com/franklynwang/EcneProject)

- Ecne is based on a worklist + fixed point algorithm
- Needs manually devised inference rule for deducing uniqueness
- Applies well to circuits within inference scope
- Specialized for weak (witness) verification

Picus (https://github.com/chyanju/Picus)

- Picus is based on symbolic execution
- Supports **customized** specifications/queries besides weak (witness) uniqueness property
- Automated verification, less manual efforts required, incorporates optimizations from existing solvers

Problems & Existing Challenges

- Scalability: Difficult Constraints
- Coverage: Unsupported Cases
- Extensibility: New Emerging Language Interfaces
- ...

Potential Approaches for ZK Circuit Verification

Abstract Interpretation with Interval Analysis

- Obtain constraint annotations from user or static analysis

Unified Intermediate Representation for ZK Constraint Verification (Domain-Specific IR)

- CirC
- Vamp IR
- ...

Prime Field Theory for Existing Solvers

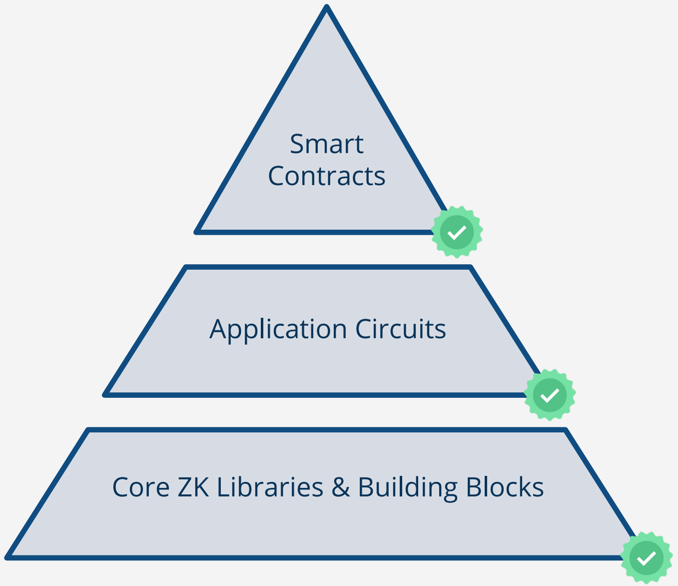
- Based on Gröbner bases solvers
- Based on Integer theory with annotated range intervals
- ...

Plans & Next Steps

Plans & Next Steps

Core Library Circuits
- Core circomlib
- BigInt Arithmetic
- Elliptic Curve Arithmetic
- circom-ecdsa / circom-pairing

Application Circuits
- Application Circuit Benchmarks
- Constraint Annotation (Manual / Automated Analysis)
- Incorporation of Verified Core Library into Picus
- Abstract Interpretation for Uniqueness Analysis



THANKS