Automated Detection of Under-Constrained Circuits in Zero-Knowledge Proofs

Shankara Pailoor*[18], Yanju Chen*[28], Franklyn Wang[37], Clara Rodríguez[4], Jacob Van Geffen[58], Jason Morton[6], Michael Chu[7], Brian Gu[7], Yu Feng[28], Isil Dillig[18]

* Equal Contribution

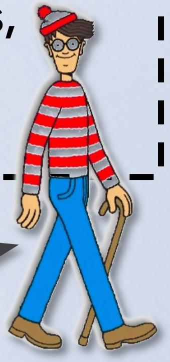
1. The University of Texas at Austin; 2. University of California, Santa Barbara; 3. Harvard University;
4. Complutense University of Madrid; 5. University of Washington; 6. ZKonduit; 7. 0xPARC; 8. Veridise.

Zero-Knowledge Proofs

A zero-knowledge proof system allows users to prove statements while using but not revealing some secret information.

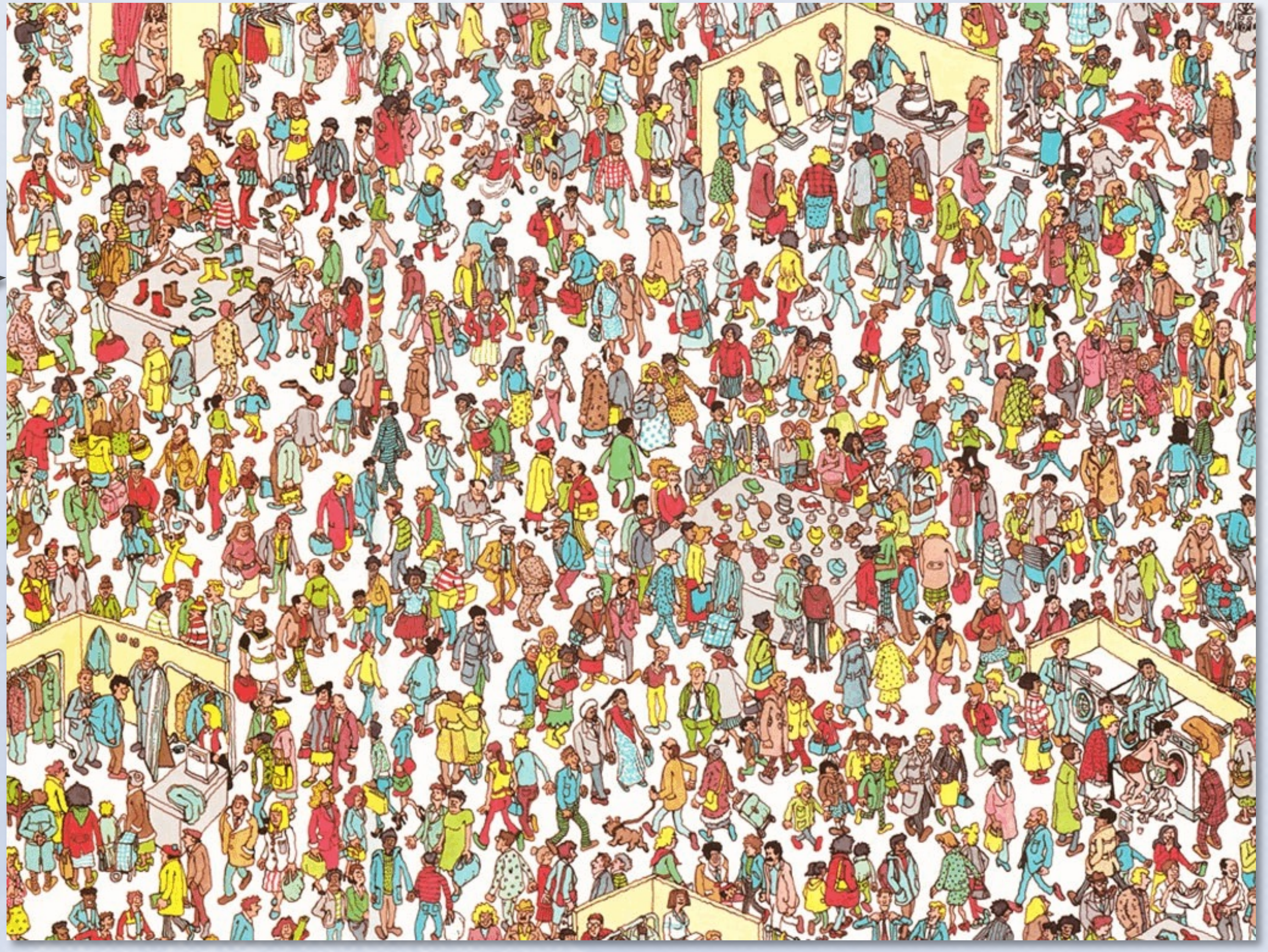
Example: Where's Wally?

How do you show your friend that you have knowledge of where Wally is, without giving away his location?



Wally

Show Wally through a cutout.

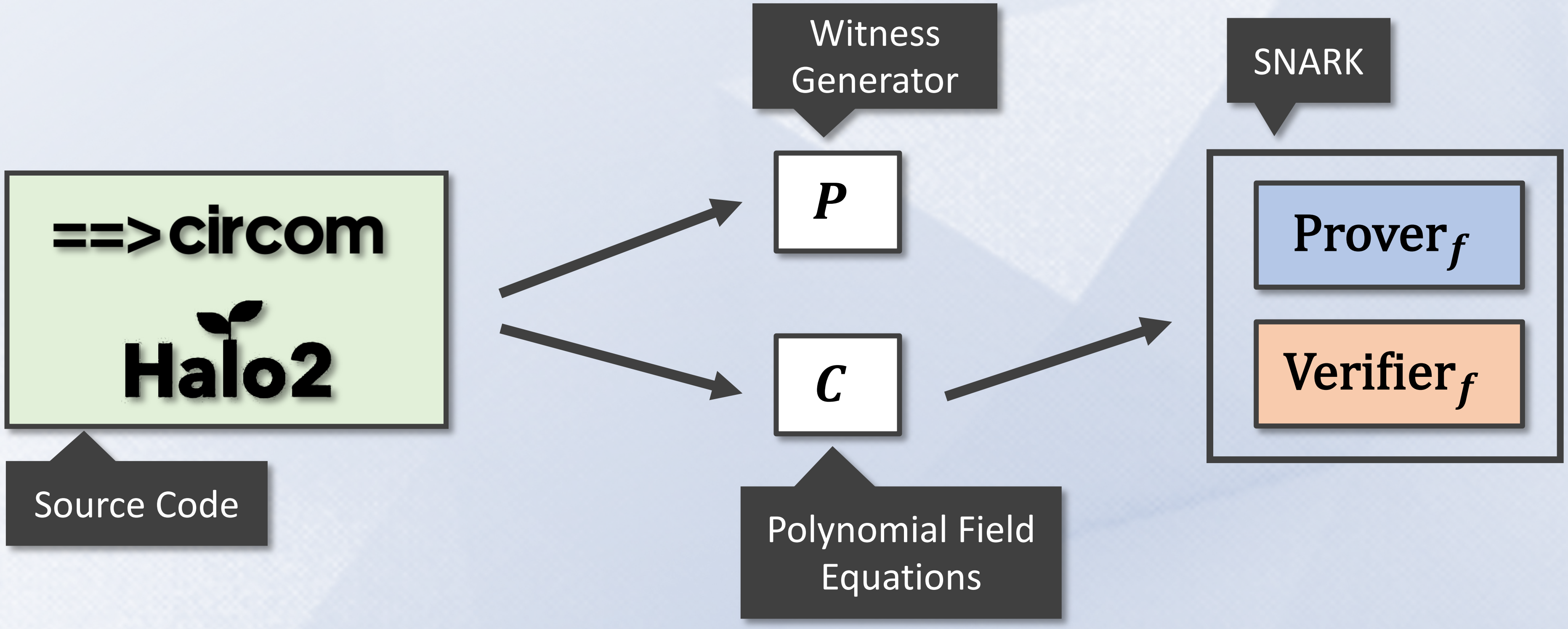


Credit: https://www.circularise.com/blogs/zero-knowledge-proofs-explained-in-3-examples

Real-World Zero-Knowledge Proofs

- Verifiable Computation
- Secured Messaging
- Safe Whistle-Blowing
- Voting Systems
- Supply Chains
- Identity Management
- Gaming
- Cryptocurrencies
- File System Control
- Privacy in Banking
- IoT Security
- Complex Documentation
- Privacy in Public Blockchains

ZK Circuit Workflow



Source Code: Witness Generation <u>and</u> Constraints

Witness generation and constraints should (generally) be <u>equivalent</u>!

What is equivalence?

Program: $\boldsymbol{P}$
Input: $\boldsymbol{x}$
Output: $\boldsymbol{y}$

Set of Constraints: $\boldsymbol{C}$
Inputs: $\boldsymbol{x}, \boldsymbol{y}$
Output: *true* or *false*

For every $\boldsymbol{x}$ and $\boldsymbol{y}$, $\boldsymbol{P}(\boldsymbol{x}) = \boldsymbol{y}$ if and only if $\boldsymbol{C}(\boldsymbol{x}, \boldsymbol{y})$ is *true*.

Every input-output of P must satisfy C

Every (x, y) which satisfies C must be an input-output pair of P

🤔 How can this be violated?

Equivalence Violations

Two Requirements:

(1) Every input-output pair of P satisfies C

(2) For any x and y which satisfy C, $P(x) = y$

Over-Constrained Bugs

Exists x and y where $P(x) = y$ but $C(x, y)$ is *false*.

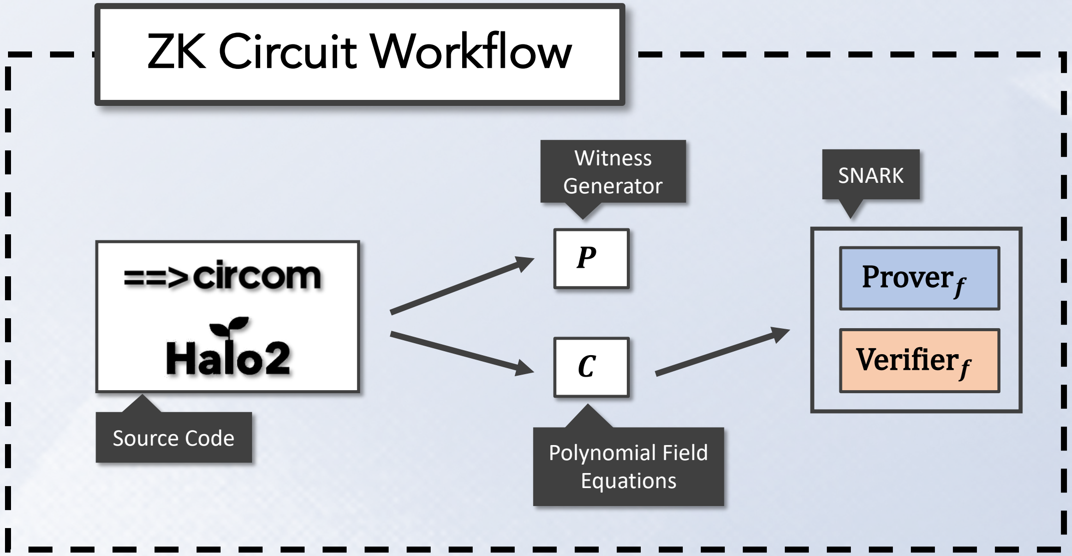
Most ZK languages (e.g., Circom, Halo2) add field equations as assertions to circuit!

Under-Constrained Bugs

Exists x and y where $C(x, y)$ is *true* but $P(x) \neq y$.

Focus of this talk!

Why do we care?



Under-constrained bugs: Verifier can accept bad inputs/outputs.

Tornado Cash

Oct 12, 2019 · 3 min read · Listen

Tornado.cash got hacked. By us.

Could be used to drain all tokens

BigMod incorrectly omits range checks on the remainder #10

Merged xu3kev merged 1 commit into `0xPARC:master` from `ecnerwala:rangecheckmod` on Apr 26

Disclosure of recent vulnerabilities

We have recently patched two severe bugs in Aztec 2.0. The first was found by an Aztec engineer and the second by community members.

1. Lack of range constraints for the `tree_index` variable

Double spend

Simple Example: Under-Constrained Bug

bitify.circom

```
template Num2Bits(n) {
    signal input in;
    signal output out[n];
    var lc1 = 0;
    var e2 = 1;
    for (var i=0; i<n-1; i++) {
        out[i] <-- (in >> i) & 1;
        out[i] * (out[i] - 1) === 0;
        lc1 += out[i] * e2;
        e2 = e2 + e2;
    }
    lc1 === in;
}
```

Developer-added constraints

Constraints for $\boldsymbol{n = 3}$

$$\begin{cases} \text{input } in \\ \text{output } out_0, out_1, out_2 \\ out_0 \cdot (out_0 - 1) = 0 \\ out_1 \cdot (out_1 - 1) = 0 \\ out_0 + 2 \cdot out_1 = in \end{cases}$$

out_2 is under-constrained

Attacker can pass in any value for $\boldsymbol{out_2}$

Source: https://github.com/iden3/circomlib/blob/master/circuits/bitify.circom

Strategies: SMT v.s. Static Analysis

SMT Strategy

Under-constrained property can be expressed as SMT query.

⬇

$$\exists y_1, y_2. P[y_1/y] \wedge P[y_2/y] \wedge y_1 \neq y_2$$

SAT means the circuit is under-constrained.

Otherwise, we say y is *properly constrained*, or *unique*.

Too slow; can't scale!

V.S.

Static Analysis Strategy

Apply pre-defined rules to quickly detect if a circuit is properly constrained.

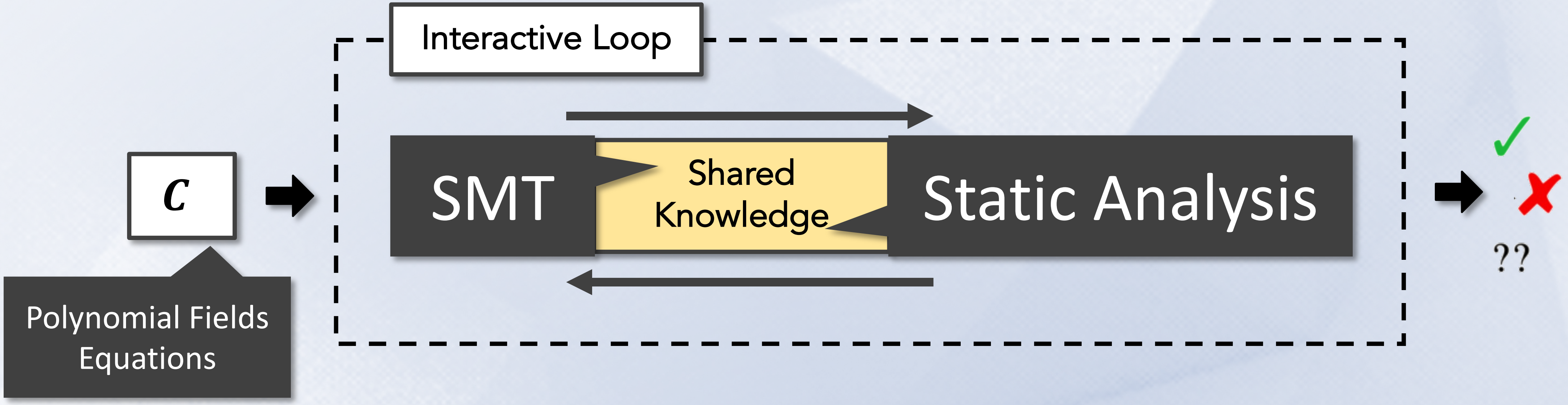
⬇

$$\begin{cases} \text{input } x \\ \text{output } y \\ z = 3x + 4 \\ y = z + 2x \end{cases}$$

Since y is linear in x and z, we immediately infer it is not under-constrained.

False positives; one-time deal

QED²: An Overview

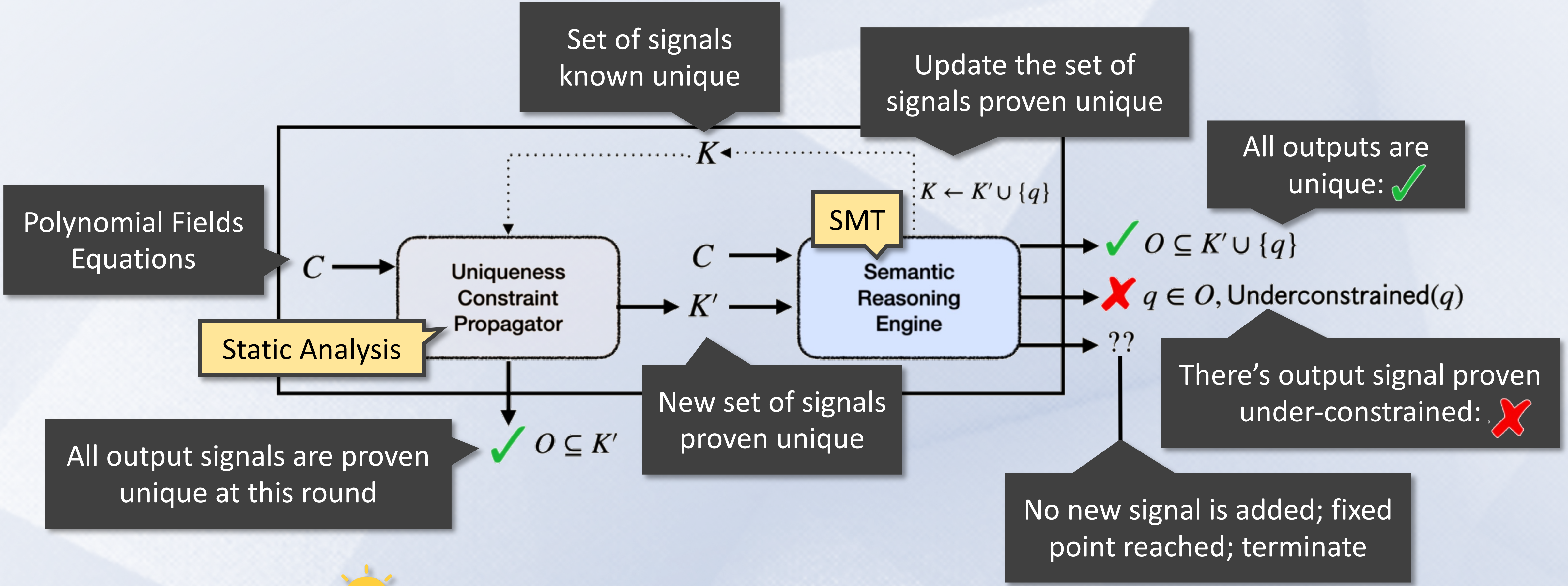


Combine the strengths of static analysis and SMT!

Static analysis and SMT phases interact in a loop.

Partial results from one phase can be useful for another.

QED^2: An Interactive Loop



Knowledge from both phases are shared.

Multiple turns scale reasoning up.

Example: Solving Num2Bits

We show how QED[2] detects the under-constrained bugs in Num2Bits, and construct a counter-example as attack vector.

bitify.circom

```
template Num2Bits(n) {
    signal input in;
    signal output out[n];
    var lc1 = 0;
    var e2 = 1;
    for (var i=0; i<n-1; i++) {
        out[i] <-- (in >> i) & 1;
        out[i] * (out[i] - 1) === 0;
        lc1 += out[i] * e2;
        e2 = e2 + e2;
    }
    lc1 === in;
}
```

```
...
# unique: #<set:>.
# refined known-set: #<set: 0 4>
# refined unknown-set: #<set: 1 2 3>
  # propagation (linear lemma): none.
  # propagation (binary01 lemma): none.
  # propagation (basis2 lemma): #<set: 1 2> added.
  # propagation (aboz lemma): none.
  # propagation (aboz lemma): none.
  # propagation (linear lemma): none.
  # propagation (binary01 lemma): none.
  # propagation (basis2 lemma): none.
  # propagation (aboz lemma): none.
  # propagation (aboz lemma): none.
  # checking: (x3 y3), sat.
# final unknown set #<set: 3>.
# weak uniqueness: unsafe.
# counter-example:
  #hash((m1.main.in . 2) (m1.main.out[0] . 0)
(m1.main.out[1] . 1) (m1.main.out[2] . 1)
(m2.main.out[0] . 0) (m2.main.out[1] . 1)
(m2.main.out[2] . 0)).
```

UCP

SMT

Attack Vector

Output of QED[2] showing the bug.

Bug: Any values of out_2 is accepted.

Benchmark Suite: ZkBench

We gathered an extensive benchmark suite from circomlib, the standard library for Circom.

Utility templates for fixed-width integer computation and commonly used blockchain primitives → circomlib-utils

circomlib-core ← In-depth coverage of 50 most security-critical templates

Benchmark Set	# circuits	Avg. # constraints	Avg. # output signals
circomlib-utils	59	352	10
circomlib-core	104	6,690	32
All	163	4,396	24

Key statistics of ZkBench.

Three categories:
- Small
- Medium
- large

Evaluation: Effectiveness

Solved Benchmarks

Solving Time

RQ1: Is QED² effective?

RQ2: Is QED² useful for detecting unknown bugs in real-world circuits?

Benchmark	circomlib-utils				circomlib-core				overall
Size	small	medium	large	overall	small	medium	large	overall	
Total (#)	47	7	5	59	61	23	20	104	163
Avg. Time (s)	9s	10s	9s	9s	8s	13s	18s	10s	9s
✓ (#)	36	4	3	43	44	10	4	58	101
✗ (#)	6	0	0	6	7	0	0	7	13
Solved (%)	89%	57%	60%	83%	84%	43%	20%	63%	**70%**

Key results for effectiveness evaluation.

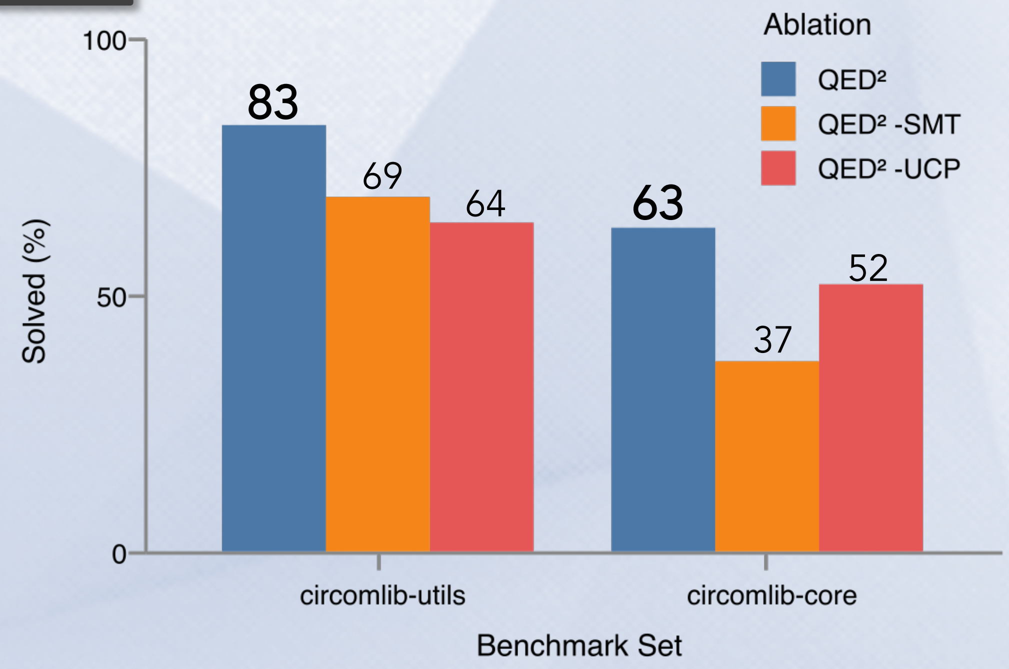
QED² solves 70% of the benchmarks, averaged 18s for each of them.

QED² finds 8 serious unknown vulnerabilities.

Evaluation: Ablation

RQ3: What's the relative importance of SMT and UCP?

- **QED²** — Full-fledged version
- **QED²-SMT** — Only performs symbolic reasoning (SMT)
- **QED²-UCP** — Only performs uniqueness constraint propagation (static analysis)



Comparison between QED² and its ablations.

UCP is crucial in quick verification for large circuits;
SMT is precise in solving difficult circuits.

The synergistic bond between SMT and UCP is effective.

Conclusions

New algorithm for automatic checking of under-constrained zero-knowledge circuits

Lightweight Inference + SMT-Based Reasoning

ZKBENCH, an open-source benchmark suite for systematic evaluation of ZK circuits

Our tool: solves 70% benchmarks and detects 8 unknown under-constrained vulnerabilities

Check out our tool on Github!

https://github.com/chyanju/Picus/tree/main

Artifacts Available acm V1.1

Artifacts Evaluated acm Reusable V1.1

Questions?