Satisfiability Modulo Finite Fields
with applications to Zero-Knowledge Proof Compilers

Alex Ozdemir, Gereon Kremer, Riad S. Wahby, Cesare Tinelli, Fraser Brown, Clark Barrett
“Bounded Verification for Finite-Field Blasting” (CAV’23)
“Satisfiability Modulo Finite Fields” (CAV’23)
Private cryptocurrencies

integrity:
• authorization
• conservation of $ 

privacy:
• transactions are hidden

Possible with: Zero-Knowledge Proofs (ZKPs)
How a private cryptocurrency is built

Idea

High-level code

Compile

Application

Program

\( f_n() \rightarrow \text{bool} \{
\ldots
\}
\)

\( Z_p : +, x, = (mod p) \)

\( F \)-equations

\( \omega, (\omega, \cdot) = 0 \)

\( Z_p \)-blaster

\( F \)-blaster

We verify this using SMT+\( F \)

covertly insolvent!

Zero-Knowledge Proof System

DAG

(Bool, \( Z_{2^n} \), \( F \))

pass_1

\( \cdots \)

pass_n

\( \cdots \)
Today’s Talk

I. Correctness for a ZKP compiler

II. A framework for verifiable $\mathbb{F}$-blasting
   • With automatic (bounded) verification

III. Satisfiability modulo finite fields

IV. Case study: bugs in CirC’s $\mathbb{F}$-blaster
Part I: Correct ZKP compilers
What is a ZKP System?

\[ \text{prove: } \phi(x, w) \in C \text{ secret witness} \]

\[ \exists w, \phi(x, w)? \]

\[ \text{verify: } \{ \bot, T \} \text{ for payments} \]

Properties:

- **complete + sound***: valid \( w \leftrightarrow V \text{ accepts} \)
- **zero-knowledge***: hides \( w \)

* usually under computational assumptions
What is a correct ZKP compiler from $\mathcal{C} \rightarrow \mathcal{C'}$?

Compile($\phi(x, w) \in \mathcal{C}$) outputs:
- $\phi'(x', w') \in \mathcal{C'}$
- $E_x(x) \rightarrow x'$
- $E_w(x, w) \rightarrow w'$

Our definition: correctness as **equisatisfiability**:

**demonstrable completeness**
\[
\forall x, \forall w, \quad \phi(x, w) \Rightarrow \phi'(E_x(x), E_w(x, w))
\]

**demonstrable soundness**
\[
\exists \text{eff. } I(x, w') \rightarrow w, \ \forall x, \ \forall w', \quad \phi'(E_x(x), w') \Rightarrow \phi(x, I(x, w'))
\]
Properties of our definition

Theorem 1 (ZKPS Generalization):

Correct ZKP compiler from $C \rightarrow C'$  

Secure ZKPS for $C'$  

$\equiv$  

Secure ZKPS for $C$

Theorem 2 (Compiler Composition):

Correct ZKP compiler from $C \rightarrow C'$  

Correct ZKP compiler from $C' \rightarrow C''$  

$\equiv$  

Correct ZKP compiler from $C \rightarrow C''$
Part II: Verifiable $F$-blasting
The architecture of a Finite-Field blaster

High-Level Code

DAG (Bool, \(\mathbb{Z}_{2^n}\), \(\mathbb{F}\))

\[ (b \; ? \; x \; : \; y) \leq (x + y) \]

\( b \in \text{Bool} \quad x, y \in \mathbb{Z}_{2^n} \)

\(= 1\)

\[ \leq_u \]

encoding rule for \(\leq_u\) in \(\mathbb{Z}_{2^n}\)

encoding rule for \(+\) in \(\mathbb{Z}_{2^n}\)

encodings:

\[ b' \in \{0, 1\}, c \in \mathbb{F} \quad x', y' \in \mathbb{Z}_{2^n} \]
Example rule: \( n \)-ary Boolean AND (complex)

- Encode a Boolean function:
  - \( y = x_1 \land \cdots \land x_n \)
- Assume:
  - \( p \gg n \)
  - Field variables \( x_i' \):
    - \( x_i' = \text{IfThenElse}(x_i, 1, 0) \)
- Task:
  - emit \( \mathbb{F} \)-equations, new variables
  - Ensure \( y' = \text{IfThenElse}(y, 1, 0) \)

Naively, binary \( \land \):
- \( t_0' = 1 \)
- \( t_{i+1}' = x_i' t_i' \)
- \( y' = t_n' \)

Fewer non-linear \( \times \)s:
- Idea: \( y' = \text{AreEqual}(n, \sum_i x_i') \)
- Implemented as:
  1) \( (n - \sum_i x_i')z' = 1 - y' \)
  2) \( (n - \sum_i x_i')y' = 0 \)

What does this mean? {complete? sound?}
A framework for a verifiable $\mathbb{F}$-blaster

- **DSL for**
  - encoding schemes
  - encoding rules

- **Operational semantics**
  - $\mathbb{F}$-blaster

- **VCs for each rule**:
  - “assumes validly encoded inputs”
  - “must validly encode outputs”
  - Idea: verify with SMT+$\mathbb{F}$

<table>
<thead>
<tr>
<th>Original Term</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean $x$</td>
<td>$x' \in {0,1} \in \mathbb{F}$</td>
</tr>
<tr>
<td>bit-vector $x \in \mathbb{Z}_{2^b}$</td>
<td>$x' \in {0, \ldots, 2^b - 1} \in \mathbb{F}$</td>
</tr>
<tr>
<td>bit-vector $x \in \mathbb{Z}_{2^b}$</td>
<td>$x_1', \ldots, x_b' \in {0,1} \in \mathbb{F}$</td>
</tr>
<tr>
<td>field elem. $x \in \mathbb{F}$</td>
<td>$x' \in \mathbb{F}$</td>
</tr>
</tbody>
</table>

- **Encodings**
  - $\text{bv.add}$
  - $\text{bv.sub}$
  - $\text{bv.lshl}$
  - $\text{bv.shr}$
  - $\text{bv.ashl}$
  - $\text{bool.and}$
  - $\text{bool.or}$
  - $\text{ff.add}$
  - $\text{ff.div}$
  - conversions
  - new vars
  - priv. vars

Theorem 3: VCs $\Rightarrow$ correct $\mathbb{F}$-blaster
Part III: Satisfiability Modulo Finite Fields
Satisfiability Modulo Theories

\[ \neg (A[x]=y \land x \neq y) \Rightarrow \neg A[x]=x \land y > x \]

\[ A[x]=y \land x \neq y \]

SMT Core

CDCL!

SMT Formula

conjunctons

Integer Solver

Bit-Vector Solver

Field Solver

Array Solver

Datatype solver

UNSAT (+core) or SAT (+solution)
Prime-Order Finite Fields

• Written $\mathbb{Z}_p$, $\mathbb{F}_p$, or $\mathbb{F}$
• A set of integers: \{0, ..., $p - 1$\}
• Operations:
  • + (mod $p$)
  • $\times$ (mod $p$)
  • $=$
• Our theory: fixed $p$

\[
X = X + 1
\]
\[
\forall
\]
\[
Y^2 = Y
\]

SAT : $x = 0, y = 0$

\[
X = X + 1
\]

UNSAT
theory solver, preprocessing:

Step 1: homogenize (convert disequalities to equalities)

\[ \bigwedge a_i = b_i \land \bigwedge c_i \neq d_i \]

\[ \bigwedge (c_i - d_i)w - 1 = 0 \]
If theory solver, main sketch:

Problem: SAT in extension field \( x=F_i \) → Unsound!

Solution: field polynomials \( x^p-x \)

Pro: roots are (just) \( \mathbb{Z}_p \)

Con: degree is \( p \) (~2^{255})

\( \mathbb{Z}_p \)-equations

\( \bigwedge_i f_i(\bar{X}) = 0 \)

Gröbner Basis: \( \exists \in \langle f_i \rangle \)?
The Backtracking Search

At each node:

- Compute a GB $B$

- Branch in one of three ways:
  - If $\exists$ univariate $p(X) \in B$:
    - factor and branch on $X$ values
  - If $\dim \mathcal{V}(\langle B \rangle) = 0$:
    - compute a minimal $p(X) \in \langle B \rangle$, factor, and branch on $X$ values
  - Otherwise, exhaustion:
    - $X = 1, Y = 1, X = 2, Y = 2, \ldots$

Worst-case $O_{n,d}(p)$
SMT + $\mathbb{Z}_p$ is the best choice for ZK verification

cvc5 + $\mathbb{Z}$  Z3 + $\mathbb{Z}$  bitwuzla + $\mathbb{Z}_{2^n}$
SMT + $\mathbb{Z}_p$ is the best choice for ZK verification

solvers: bitwuzla-1.0pre (QF_BV), z3-4.11.2 (QF_NIA), cvc5 nightly 17 Dec ’22 (QF_NIA,QF_FF), CoCoALib 0.998
SMT + $\mathbb{Z}_p$ is the best choice for ZK verification.

Solvers: bitwuzla-1.0pre (QF_BV), z3-4.11.2 (QF_NIA), cvc5 nightly 17 Dec '22 (QF_NIA,QF_FF), CoCoALib 0.998
SMT + $\mathbb{Z}_p$ is the best choice for ZK verification

cvc5 + $\mathbb{Z}$  Z3 + $\mathbb{Z}$  bitwuzla + $\mathbb{Z}_{2^n}$  Comp. Alg.  cvc5 + $\mathbb{Z}_p$ (two versions)

solvers: bitwuzla-1.0pre (QF_BV), z3-4.11.2 (QF_NIA), cvc5 nightly 17 Dec ’22 (QF_NIA,QF_FF), CoCoALib 0.998
Part IV: Case Study
Verified $\mathbb{F}$-blasting in CirC

• CirC [IEEE S&P’22]
  • compiler infrastructure
    • to ZKPSs, MPCs, SMT, ...
  • state-of-the-art for ZKPS
• $\mathbb{F}$-blaster:
  • 4 encoding schemes
  • 80+ encoding rules
• Implementation:
  • $\mathbb{F}$-blaster DSL
  • operational semantics
  • VC generation
  • rules from original $\mathbb{F}$-blaster

4 bugs

cvc5 + $\mathbb{F}$
(Simplified) Soundness bug

• Rule for: \( z = x \geq_s 0 \)
  • \( x \in \mathbb{Z}_{2^b} \), signed comparison
  • \( \geq_s : \text{signed} \geq \)

• Assume: signed representation
  • \( x' \in \{-2^{b-1}, \ldots, 2^{b-1} - 1\} \subset \mathbb{F} \)

• Goal: \( z' = \text{IfThenElse}(z, 1, 0) \)
  • \( z' = 1 \) iff
  • \( x' \in \{0, \ldots, 2^{b-1} - 1\} \)

• Broken approach:
  • “Attempt” an unsigned decomposition
  • Introduce \( y' \in \{0, \ldots, 2^{b-1} - 1\} \)
    • \( y' = \sum_{i=0}^{b-2} 2^i b'_i \)
    • \( b'_i(b'_i - 1) = 0, i \in \{0, \ldots, b - 2\} \)
  • \( z' = \text{AreEqual}(x', y') \)
  • If \( x <_s 0 \): \( y' \) cannot equal \( x' \)
  • If \( x \geq_s 0 \): \( y' \) can still differ from \( x' \)!

• Potential impact: insolvency
• Fix: split \( x' \) into signed bits
  • flip sign bit
Satisfiability Modulo $\mathbb{F}$ & Bounded Verification for $\mathbb{F}$-Blasting

**Alex Ozdemir, Gereon Kremer, Riad S. Wahby, Cesare Tinelli, Fraser Brown, Clark Barrett**

**Talk Summary**

I. ZKP compiler correctness

II. Verified $\mathbb{F}$-blasting

III. Satisfiability modulo $\mathbb{F}$
   - Avoids field polynomials

IV. Case study: CirC (4 bugs)

**Thesis:** Verification for ZK is **crucial**. With SMT+$\mathbb{F}$ it is **feasible**.

What should we verify next?
Where else do fields matter?
Appendices
Effect of Field Polynomials

Field polys: terrible for even tiny fields

Field polys scale poorly with field size

(p: 4-12 bits, 8GB, 5min) (commonly solved SAT)
Comparison with Bit-Vectors

(p: 5-60 bits, 1CPU, 8GB, 5min)  

BV is worse, even for small fields

(commonly solved)

BV scales poorly with field size