Satisfiability Modulo Finite Fields

(with applications to compilers for zero-knowledge proofs)

Alex Ozdemir, Gereon Kremer, Cesare Tinelli, Clark Barrett [CAV’23]
How Private Cryptocurrencies are Built

Idea

Programming/Compilation

bugs!

Equations ($\mathbb{Z}_p$)

$\chi_i + \chi_e = \omega_i$
$\omega, (\omega - i) = 0$

ZKP

(Zero-Knowledge Proof)

Private Cryptocurrency

insolvent!

Assembly

Programming/Compilation

bugs!

%a = sub %0, %1
%b = add %a, %0

CPU

Application

crash!
The Verification Problem

Idea

Equations ($\mathbb{Z}_p$)

Programming/Compilation

Can we verify this?

Private Cryptocurrency

Specification

Boolean logic ($\mathbb{Z}_2$)

Bit-vectors ($\mathbb{Z}_{2^b}$)

Arrays ...

Implementation

equations in $\mathbb{Z}_p$

Not SMT (previously)

SMT Solver: a general-purpose theorem prover/constraint solver
Our Contribution: $\mathbb{Z}_p$ reasoning in SMT

• A theory of finite fields ($\mathbb{Z}_p$)
• Verification problems
  • ZKP translation validation (Boolean functions)
• A decision procedure
  • based on computer algebra
  • constructs UNSAT cores
  • avoids field polynomials
• An implementation in cvc5
  • Empirically fast

Bonus: An application to partially verified finite-field blasting
Part I: Prime-Order Finite Fields

• Written $\mathbb{Z}_p$
• A set of integers: $\{0, \ldots, p - 1\}$
• Operations:
  • $+ \pmod{p}$
  • $\times \pmod{p}$
  • $=$

Theory of prime-order finite fields

• For each prime $p$
  • A sort $\mathbb{F}_p$
  • Function symbols $+, \times, =$
  • Constants $0, 1, \ldots, p - 1$

• Notes:
  • fixed $p$

$X = X + 1$

$X = X + 1$ unsat

$Y^2 = Y$

SAT : $X = 0, Y = 0$
Part II: Verifying encodings of Boolean function evaluation
Encoding a Boolean Function as Equations

• Consider a Boolean fn. \( f(z_i) \rightarrow z \)  
  • Ex: \( z = f(z_0, z_1) = z_0 \land z_1 \)  
  \[ z' = z'_0z'_1 \]

• We want a system \( \phi' \):
  • In variables:
    • \( z'_i \): represents \( z_i \)
    • \( z' \): represents \( z \)
    • \( w'_i \): additional variables
  • Such that
    • \( [\phi' \land \bigwedge_i z'_i = \text{ite}(z_i, 1,0)] \rightarrow z' = \text{ite}(f(z_i), 1,0) \)
How We Generate Verification Problems

1. Sample a Boolean function \( f(z_i) \rightarrow z \)
2. Compile it to a system of equations: \( \phi \leftarrow \text{ZKPCompiler}(f) \)
3. (Optional) remove some equations
4. Write the soundness and determinism conditions
   • Removals influence condition validity:
     • No removals: VALID (or the compiler has a bug!)
     • Removals: likely INVALID
   • Benchmarks are in an SMT-LIB2 extension to \( \mathbb{Z}_p \)
     • QF_FF: mix of Boolean and \( \mathbb{Z}_p \) \((p \approx 2^{256})\)
     • Alternatives: non-linear integer arithmetic, bit-vector arithmetic, pure \( \mathbb{Z}_p \)
Different ways to verify

• Option 1: verify these conditions as **QF_FF**:
  • Booleans
  • \( \mathbb{Z}_p \) equations like \( x \times y = z \) with \( x, y, z \in \mathbb{Z}_p \)

• Options 2, 3: explicit modulus
  • \((x \times y) \% p = z \) with
    • \( x, y, z \in \{0, \ldots, p - 1\} \subset \mathbb{Z} \) (**QF_NIA**)
    • \( x, y, z \in \{0, \ldots, p - 1\} \subset \mathbb{Z}_{2^b} \) (**QF_BV**)
      • Requires \( b > \lceil 2 \log p \rceil \)

• Option 4: encode the Booleans as \( \mathbb{Z}_p \) (**Pure FF**)
Part III: Decision Procedure

Building an SMT theory solver for $\mathbb{Z}_p$
SMT + Theory Solvers

Input Formula: conjunction of (dis)equations

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

CDCL!

SMT Core

UNSAT (+core) or SAT (+solution)

Integer Solver

Bit-Vector Solver

Field Solver
\( \mathbb{Z}_p \) Solver Pseudocode

1. Given: equalities & disequalities
2. Convert each to form: \( t_i = 0 \)
   - \( t_i \): a polynomial in the input variables
3. Exist polys \( p_i \) such that \( 1 = \sum_i p_i t_i \)?
   - Return UNSAT (with core)
4. Backtracking search for a solution

Rules:
- \( a_i = b_i \Rightarrow a_i - b_i = 0 \)
- \( a_i \neq b_i \Rightarrow (a_i - b_i)w_i - 1 = 0 \)

Example:
- \( X = 0 \)

Incomplete, fast UNSAT

Slow, complete.
Finds solution.
Exist polys $p_i$ such that $1 = \sum_i p_i t_i \Rightarrow \text{UNSAT}$

- Example:
  - $t_1 = XY - 1, t_2 = X$
  - $(-1)t_1 + (Y)t_2$
    - $= -XY + 1 + XY = 1$

- Gröbner Basis engine:
  - Determines whether the $p_i$ exist
  - Worst-case time: $O(d^{2n})$
  - Empirical time: fast (see eval)

- $\exists p_i \Rightarrow \text{UNSAT}$ (why?)
  - $1 = \sum_i p_i \cdot t_i = \sum_i p_i \cdot 0 = 0$

- Problem:
  - $\neg \exists p_i \Rightarrow \text{SAT}$
    - Example: $X^2 + 1 = 0$ is UNSAT in $\mathbb{Z}_3$
    - Key issue: $X = \sqrt{-1}$ is a solution

- Classic solution:
  - Add polynomials $X^p - X$
    - roots($X^p - X$) = $\{0, ..., p - 1\}$
  - Problem: high-degree $\Rightarrow$ slow GB

Live with the incompleteness...
Solver Pseudocode

1. Given: equalities & disequalities
2. Convert each to form: $t_i = 0$
   - $t_i$: a polynomial in the input variables
3. Exist polys $p_i$ such that $1 = \sum_i p_i t_i$?
   - Return UNSAT (with core)
4. Backtracking search for a solution
   - If $\exists$ univariate $t_i(X_j)$: factor & branch
   - If $\text{dim}(t_i) = 0$: compute a univariate, factor, & branch
   - Else: exhaustive search
Part IV: Experiments
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
Effect of Field Polynomials

(p: 4-12 bits, 8GB, 5min)

Field polys: terrible for even tiny fields

(commonly solved SAT)

Field polys scale poorly with field size
Main Experiment

Full-size field: $p \approx 2^{255}$

What we benchmarked:

- Our solver (FF)
- Our solver, no cores (FF-)
- Embed $\mathbb{Z}_p$ in bit-vectors (BV)
- Embed $\mathbb{Z}_p$ in integers (NIA)
- Embed Booleans in $\mathbb{Z}_p$ (CAS)

solvers: bitwuzla-1.0pre, z3-4.11.2, cvc5 nightly 17 Dec ’22, CoCoALib 0.998
More in the paper

• Translation validation for *determinism*
• Theory & decision procedure for non-prime finite fields.
• UNSAT cores
  • Calculus for proving ideal membership

**Future Work:**
• Worst-case runtime $O_{d,n}(\log p)$, not $O_{d,n}(p)$?
• Avoid full GB?
• Incremental computation *within* field solver.
Satisfiability Modulo Finite Fields

Alex Ozdemir, Gereon Kremer, Cesare Tinelli, Clark Barrett

ePrint: https://ia.cr/2023/091

- First SMT $\mathbb{Z}_p$ solver
  - UNSAT cores
  - No field polynomials
- First QF_FF benchmarks
  - Application area: ZKPs
- A foundation for $\mathbb{Z}_p$ verification
- Try it! (cvc5 main, all APIs)
Bonus: Bounded Verification for Finite-Field Blasting
The Verification Problem

Idea

Programming/Compilation

Equations \((\mathbb{Z}_p)\)

\[ \chi_i \cdot \chi_i = \omega_i \]

\[ \omega_i (\omega_i - i) = 0 \]

ZKP
(Zero-Knowledge Proof)

Private Cryptocurrency

Can we verify this?

Previous benchmarks:
translation validation

What about (automatic) verification?
The CirC Compiler Infrastructure

Program $\rightarrow$ IR[...] $\rightarrow$ IR[Bool,BV,\(\mathbb{Z}_p\)] $\rightarrow$ \(\mathbb{Z}_p\)-Equations

Front-End Optimization, Simplification

“Finite-Field Basting”
Correctness for a ZKP compiler

Syntax:

\[
\text{Compile}(\phi(x, w) \in \Phi) \rightarrow \phi'(x', w') \in \Phi'
\]
\[
\text{Ext}_x(x) \rightarrow x'
\]
\[
\text{Ext}_w(x, w) \rightarrow w'
\]

Correctness:

• *Demonstrable* Soundness
• *Demonstrable* Completeness

Key Properties:

• Closed under sequential composition
• ZKP is secure
The CirC Compiler Infrastructure

Program $\rightarrow$ IR[...] $\rightarrow$ IR[Bool,BV,\(\mathbb{Z}_p\)] $\rightarrow$ \(\mathbb{Z}_p\)-Equations

Front-End Optimization, Simplification “Finite-Field Blasting”

\textit{correct} \quad \textit{correct} \quad \textit{correct}
The anatomy of a field blaster

Post-order traversal

IR

\[
\begin{array}{c}
\text{v} \\
\leq \\
+ \\
\downarrow \\
x \quad y
\end{array}
\]

\[\mathbb{Z}_p\text{-Equations}\]

Encodings

\[
\text{Bool } x \to x' \in \{0,1\} \in \mathbb{Z}_p
\]

\[
\text{BV } x \to x' \in \{0, \ldots, 2^b - 1\} \in \mathbb{Z}_p
\]

\[
\text{BV } x \to x'_1, \ldots, x'_b \in \{0,1\} \in \mathbb{Z}_p
\]

\[
\text{Field Element } x \to x' \in \mathbb{Z}_p
\]

Encoding Rules

\[
\begin{align*}
\text{bv.add} & \quad \text{bv.mul} & \quad \text{bv.sub} & \quad \text{bv.udiv} \\
\text{bv.lshr} & \quad \text{bv.shr} & \quad \text{bv.ashr} & \\
\text{bv.urem} & \quad \text{bv.uor} & \quad \text{bool.xor} & \\
\text{bv.uge} & \quad \text{bv.xor} & \quad \text{ff.mul} & \\
\text{bv.or} & \quad \text{bv.and} & \quad \text{ff.add} & \\
\end{align*}
\]

Automation Needed

..................
Verification Recipe

• Post-order traversal → inductive proof
• Encoding ‘validity’ → inductive invariants
• Each encoding rule → verification conditions
  • Soundness + completeness
  • Automatically generated
  • Automatically verified (SMT)
    • Up to tiny bounds on:
      • bit-vector length (4)
      • operator arity (4)

4 Bugs
• Incompleteness in bv.*sh*
• Incompleteness in ff.div
• Unsoundness in bv.udiv
• Unsoundness in bv comparisons
  • Severe
**Bounded Verification for Finite-Field Blasting**

**Alex Ozdemir, Riad S. Wahby, Fraser Brown, Clark Barrett**

**Post-order traversal**

\[
\begin{align*}
\text{IR} & \quad \text{\(\mathbb{Z}_p\)-Equations} \\
\end{align*}
\]

- IR
- \(v\)
- \(\leq\)
- \(+\)

\[x\quad y\quad \cdots\]

**Encodings**

- \(\text{Bool} \; x \to x' \in \{0,1\} \in \mathbb{Z}_p\)
- \(\text{BV} \; x \to x' \in \{0,\ldots,2^b - 1\} \in \mathbb{Z}_p\)
- \(\text{BV} \; x \to x'_1,\ldots,x'_b \in \{0,1\} \in \mathbb{Z}_p\)
- Field Element \(x \to x' \in \mathbb{Z}_p\)

**Encoding Rules**

- \(\text{bv.add} \quad \text{bv.mul} \quad \text{bv.sub} \quad \text{bv.udiv} \quad \text{bv.urem} \quad \text{bv.lshr} \quad \text{bv.ashr} \quad \text{bv.ushr} \quad \text{bv.uge} \quad \text{bv.xor} \quad \text{bv.or} \quad \text{bv.and} \quad \text{ff.mul} \quad \text{ff.add} \quad \text{bool.xor} \quad \text{bool.or} \quad \text{bool.and} \quad \text{bool.not} \)

**Automation Needed**

- \(\text{variable introductions}\)
- \(\text{private variable introductions}\)
Two Properties

Soundness

Any solution with valid inputs has the correct output.

\[ \phi' \land \bigwedge_i z'_i = \text{ite}(z_i, 1, 0) \]
\[ \rightarrow \]
\[ z' = \text{ite}(f(z_i), 1, 0) \]

Determinism

Any solution, for fixed inputs, has a unique output.

\[ \phi' \land \phi'' \land \bigwedge_i z'_i = z'' \]
\[ \rightarrow \]
\[ z' = z'' \]