internal/smt

Formal Verification for gnark Circuits with SMT Solvers

gnark Team

Consensys

The Problem

Circuit Bugs Are Expensive

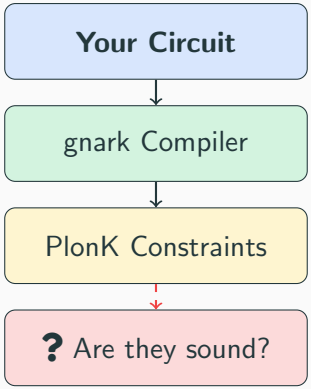
Common ZK Circuit Vulnerabilities:

- ⚠ Under-constrained variables
- ⚠ Missing range checks
- ⚠ Unconstrained hint outputs
- ⚠ Redundant constraints (inefficiency)

The Challenge:

Manual review doesn't scale

Testing can miss edge cases



The Solution

Formal Verification of Actual Compiled Constraints

What it does:

- Extracts real PlonK constraints from gnark
- Exports to SMT solver formats
- Detects under-constrained circuits
- Generates beautiful reports

Key Insight:

Don't verify hand-written approximations.

Verify the *actual constraints* that the gnark compiler produces!

```
qL*xa + qR*xb + qO*xc + qM*(xa*xb) + qC = 0
```

Simple API

```
// Compile your circuit and analyze it in 5 lines
circuit := &MyCircuit{}
opts := smt.DefaultCompileOptions()
opts.TestName = "MyCircuit"

result, err := smt.CompileCircuit(circuit, opts)
if err != nil {
    log.Fatal(err)
}

// Print analysis results
result.PrintSummary()

// Generate HTML report
result.WriteReportToFile("report.html", "MyCircuit")

// Export for cvc5 verification
result.WriteToFile("verify_mycircuit.cpp")
```

Features

Feature 1: Static Analysis

Instant Soundness Checks (No SMT Solver Required)

Detects:

- 🐞 Unused variables
- 🐞 Trivial/impossible constraints
- 🐞 Missing range checks on limbs
- 🐞 Unbounded secret variables
- 🐞 Under-constrained hint outputs
- 🐞 Unconstrained public outputs

Pattern Recognition:

- Decomposition patterns
- Inverse check patterns
- Range constraint detection
- Log-derivative argument detection

🚀 Runs in milliseconds

Feature 2: SMT Solver Export

Two Export Formats for cvc5

C++ API Export

- Full cvc5 C++ integration
- Ready-to-compile verification code
- Helper functions included
- Test scaffolding generated

```
g++ -std=c++17 verify.cpp -lcvc5
```

SMT-LIB2 Export

- Standard SMT format
- QF_FF logic (finite fields)
- Works with any compatible solver
- Human-readable formulas

```
cvc5 verify.smt2
```

Prove properties like:

"Variable X is uniquely determined by inputs"

"Removing constraint N allows invalid witnesses"

Feature 3: Source Location Tracking

Know Exactly Where Issues Originate

- Integrates with gnark's profiling system
- Maps constraints back to your Go source code
- Full stack traces for complex circuits
- Filters out internal gnark code automatically

```
[CRITICAL] missing-range-check
Variable:  limb_0
Location:  mycircuit.go:47
Stack trace:
→ MyCircuit.Define at mycircuit.go:47
api.ToBinary at api.go:123
```

Feature 4: Beautiful Reports

Multiple Output Formats

Terminal

- ANSI colors
- Progress indicators
- Quick feedback

HTML

- Dark theme UI
- Interactive elements
- Shareable reports

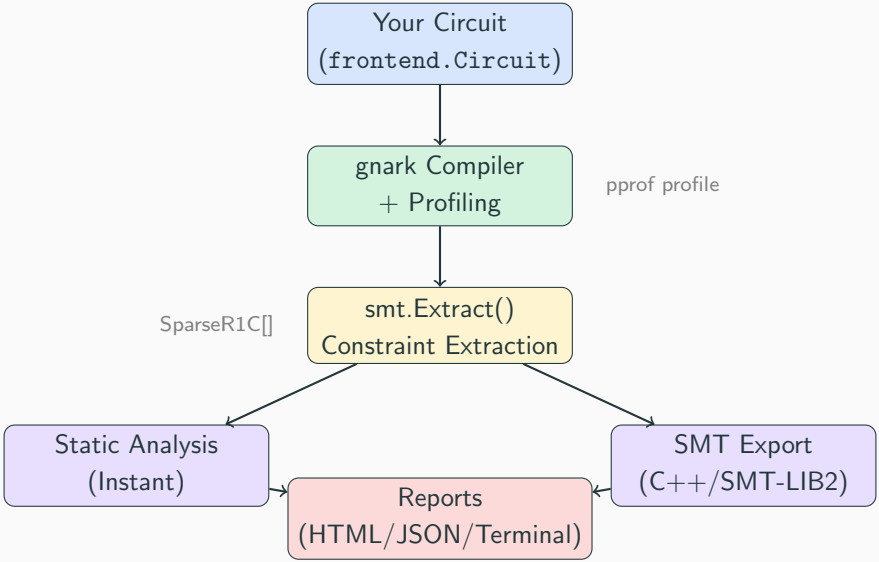
JSON

- Machine-readable
- CI/CD integration
- Structured data

```
[PASS] All internal variables are uniquely determined
[PASS] Constraint 42:  multiplication with non-zero constant
[WARN] Secret variable 'k' has no apparent range constraint
[FAIL] Hint output 'quotient' appears in only one constraint
```

How It Works

Architecture



Understanding What Gets Verified

Every gnark constraint compiles to:

$$q_L \cdot x_a + q_R \cdot x_b + q_O \cdot x_c + q_M \cdot (x_a \cdot x_b) + q_C = 0$$

Variables:

- x_a, x_b, x_c – Wire indices
- Public, Secret, or Internal

Coefficients:

- q_L, q_R, q_O – Linear terms
- q_M – Multiplication term
- q_C – Constant term

✅ The smt package extracts *exactly* these values from compiled circuits

Use Cases

Use Case 1: CI/CD Integration

Automated Soundness Checks in Your Pipeline

```
func TestCircuitSoundness(t *testing.T) {
    circuit := &MyCircuit{}
    result, _ := smt.CompileCircuit(circuit,
                    smt.DefaultCompileOptions())

    // Run static analysis
    analysis := result.Analyze("MyCircuit")

    // Fail on critical issues
    if analysis.HasCritical() {
        analysis.Print(os.Stdout)
        t.Fatal("Circuit has critical soundness issues")
    }
}
```

Use Case 2: Security Audits

Generate Artifacts for Auditors

What auditors get:

- Complete constraint listing
- Automatic issue detection
- Source location mapping
- Exportable verification code

Audit workflow:

1. Run static analysis
2. Review flagged issues
3. Export to cvc5 for deep checks
4. Generate report for docs

```
go run audit_tool.go -circuit=MyCircuit
-html=audit_report.html
-cpp=verify_constraints.cpp
```

Use Case 3: Development Debugging

Find Issues Before They Become Vulnerabilities

During development:

- Quick feedback loop
- Catch under-constraints early
- Understand constraint patterns
- Verify hint outputs

Example output:

```
Linear constraints:  42
Multiplication constraints:  18
Decomposition (a+k*b=c):  8
Inverse checks (a*b=1):  4

[CRITICAL] Hint output 'inv'
never constrained!
```

Catch the bug in development, not in production

Getting Started

Quick Start

```go
package main

import (
    "log"
    "github.com/consensys/gnark/internal/smt"
)

func main() {
    circuit := &MyCircuit{} // Your circuit

    // Compile and analyze
    opts := smt.DefaultCompileOptions()
    opts.TestName = "MyCircuit"
    opts.WithProfiling = true // Enable source tracking

    result, err := smt.CompileCircuit(circuit, opts)
    if err != nil { log.Fatal(err) }

    // Generate report
    result.WriteReportToFile("analysis.html", "MyCircuit")
```

Running SMT Verification

1. **Install cvc5:**

`brew install cvc5` or `apt install cvc5`

2. **Compile the generated C++:**

```
g++ -std=c++17 verify.cpp -lcvc5 -o verify
```

3. **Run verification:**

```
./verify
```

Summary

Why Use `internal/smt`?

Benefits:

- **Real constraints** – Not approximations
- **Instant feedback** – Static analysis in ms
- **Formal proofs** – SMT solver integration
- **Source tracking** – Know where bugs are
- **Beautiful reports** – HTML, JSON, Terminal
- **CI/CD ready** – Automated testing

Who should use it:

- Circuit developers
- Security auditors
- QA/Testing teams
- Anyone building with gnark

Find bugs before attackers do

Questions?

`github.com/consensys/gnark`

`internal/smt`