High-assurance Cryptography

Real World Crypto (RWC 2016)

Dave Archer, Tom DuBuisson, Nathan Collins, Joey Dodds, Trevor Eliott, Iavor Diatchki, Rob Dockins, Adam Foltzer, Joe Hendrix, Brian Huffman, Joe Kiniry, John Launchbury, Dylan McNamee, Aaron Tomb, Daniel Wagner, Simon Winwood, Dan Zimmerman, and many other current and past Galois employees who worked on high-assurance cryptography

Galois, Inc.

January 2016
Founded in 1999

60+ full time employees


Areas of expertise:

Programming Languages, Formal Methods, Security.

Company Facts
Galois has developed tools for showing that different cryptographic implementations compute the same values for all possible keys and inputs.

Uses formal verification techniques including symbolic simulation, rewriting, and third-party SAT and SMT-solvers.

From a user’s perspective, our tools act like compilers and perform exhaustive test coverage.
Cryptol: The Language of Cryptography

- Declarative specification language, tailored to the crypto domain, designed with feedback from cryptographers, and is dependently typed, pure functional language

- Cryptol 2 is open source, BSD licensed, on GitHub, has seen several releases over the past two years, and runs on all major platforms (check out [http://cryptol.net/](http://cryptol.net/))

- Cryptol includes a REPL, support for literate programming, a parser, type checker, symbolic evaluator, quickcheck-style runtime validation, and SAT and SMT-based verification

- Cryptol specifications exist for nearly every standardized or proposed cryptographic algorithm, including many curves and some post-quantum algorithms
One specification - Many uses

**Domain-specific design capture**

- Design
- Validate
- Build

**Assured implementation**

- Formal models and test cases
- Verify crypto implementations
- Hardware implementation
- Software implementation

\[
w_0 = u^{-1}l \mod p + u^{-1}w_l \mod p
\]

\[s = f \times (w_0 + p_w^2) \mod q\]
SAW: The Software Assurance Workbench

- capable of reasoning about the equivalency of Cryptol, LLVM, and JVM specifications and implementations
- highly tuned toward bit-centric computing (e.g., crypto, compression, codecs, etc.)
- works in tandem with Cryptol
- is open source, non-commercial licensed, on GitHub, was released mid-last year, and runs on all major platforms (see http://saw.galois.com/)
- includes a REPL and a proof specification language that supports compositional proof techniques spanning platforms and solvers
- SAWcore is the IR for semantic representation (dependently-
Verification Ecosystem

VHDL

C

Language

LLVM

Java

Cryptol

SAW

SYSTEM VERILOG

Verification Ecosystem
Verification of
Suite B Algorithms and more
## Suite B Verification Efforts

<table>
<thead>
<tr>
<th>Role</th>
<th>Implementation</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128 Symmetric Key Cipher</td>
<td>BouncyCastle (Java)</td>
<td>817</td>
</tr>
<tr>
<td>SHA-384 Secure Hash Function</td>
<td>libgcrypt (C)</td>
<td>423</td>
</tr>
<tr>
<td>ECDSA (P-384) Digital Signature Scheme</td>
<td>galois (Java)</td>
<td>2348</td>
</tr>
</tbody>
</table>
Some Suite B Problem Sizes

<table>
<thead>
<tr>
<th></th>
<th>Lines of Code</th>
<th>AIG Size</th>
<th>Decomposition Steps Required</th>
<th>Verification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>817</td>
<td>1MB</td>
<td>None needed</td>
<td>40 min</td>
</tr>
<tr>
<td>BouncyCastle</td>
<td></td>
<td></td>
<td>Fully automatic</td>
<td></td>
</tr>
<tr>
<td>AESFastEngine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-384</td>
<td>423</td>
<td>3.2MB</td>
<td>10 steps All solved via SAT</td>
<td>160 min</td>
</tr>
<tr>
<td>libgcrypt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECDSA (P-384)</td>
<td>2348</td>
<td>More than 5GB</td>
<td>48 steps Multiple tactics required</td>
<td>10 min</td>
</tr>
<tr>
<td>(galois)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ECC Verification Target

- verify an efficient implementation of ECDSA over NIST P-384 curve in Java (to our knowledge, the fastest in existence)
  - use known optimizations such as twin multiplication, projective coordinates, optimized field arithmetic
  - specification can use the same high-performance published algorithms as the implementation
  - implementation uses many low-level tricks for improving efficiency, often verified via a refinement style of verification
Implementing ECC

Cryptographic Protocols

**ECDSA**
- Digital Signatures

**ECDH**
- Key Agreement

One Way Functions

**R = s ⋅ P**
- Scalar Multiplication

**R = s ⋅ P + t ⋅ Q**
- Twin Multiplication

Point Operations

**R = P + Q**
- Addition

**R = P - Q**
- Subtraction

**R = 2 ⋅ P**
- Doubling

Field Operations

**Multiplication**

**Addition**

**Squaring**

**Subtraction**

**Division**

**Doubling**
ECC Benchmarks
Sign & Verify

- BC (64bit): 70ms
- Galois (32bit): 30ms
- OpenSSL (32bit): 10ms
- Galois (64bit): 50ms
- OpenSSL (64bit): 60ms
Verification Statistics

- 48 method specifications in total
  - 2 protocol specifications (verify & sign)
  - 8 scalar multiplication specifications
  - 3 point specifications (add, subtract, double)
  - 20 field specifications
  - 15 bitvector specifications
- Total verification time is under 10 minutes
Found Three Bugs

- sign & verify failed to clear all intermediate results
- boundary condition due to use of less-than where less-than-or-equal was needed
- modular reduction failed to propagate one overflow
Automatic Synthesis of High-Performance Software and Hardware Implementations
Synthesis

- our flagship product in the synthesis space was Cryptol version 1
- it is capable of generating verifiable C, JVM, VHDL, and Verilog implementations of Cryptol specs
- implementations witness decent performance
- implementations can be verified with other toolchains
- synthesis goals previously focused exclusively on code as the target artifact, not validation or verification artifacts like test benches or proofs, resp.
Synthesis

- we are forward porting ideas and code from our synthesis tools into Cryptol version 2 and SAW

- we now have prototypes of fully automatic synthesis of rigorously engineered C, LLVM, and SystemVerilog[-CSP] implementations

- artifacts included in the development method for synthesis include a domain model, requirements, correctness and security policies, and an architecture specification

- artifacts synthesized in addition to implementations include source documentation, validation artifacts (unit and system runtime verification harness), and theorems for other verification systems
Galois High-Assurance Crypto Tool Suite

**SPECIFICATIONS**
- Cipher and Protocol Specification via Standards or Research Papers
- Research Papers Characterizing Cipher or Protocol Properties
- Formal Specification of Cipher or Protocol Theorems, including Test Vectors

**CRYPTOL**
- Type Checker
- Interpreter
- Test Generator
- SAWCore Translator
- Synthesis Tuner

**SAW**
- Type Checker
- Symbolic Interpreter
- SAWScript
- Structured VC Gen

**BACKENDS**
- VHDL
- System Verilog
- Other HDLs
- SAT/SMT
- C

**EXECUTABLES**
- CIPHERS IMPLS
  - C
  - VHDL
  - System Verilog
  - Other HDLs

**CIPHER TEST BENCHES**
- C
- VHDL
- System Verilog
- Other HDLs

**PERFORMANCE, ENERGY, SPACE EVALUATION**
- Runtime Verification on HDL Simulation
- Runtime Verification on FPGA Test Bench
- Runtime Verification on ASIC Test Bench

**CORRECTNESS EVIDENCE**
- Test Vector Runtime Verification
- Test Vector Formal Verification
- Runtime Verification of Theorems as Parameterized Tests
- Formal Verification of Theorems

**SAWSCRIPT**
- Symbolic Interpreter

**CLUSTER OF SUBSYSTEMS**
- A depends upon B
- dataflow from A to B

**Key for architecture specification**

---

© Galois, Inc. 2016