Cryptographic Module Design with Domain Specific Languages

NIST Workshop on Cryptography for Emerging Technologies and Applications

John Launchbury, Nov 2011
Creating a crypto algorithm requires skills in math AND programming.

Variety of target architectures

Validation is complex and tedious

Variety of requirements
Requirements for a Crypto Domain-Specific Language

- High-level domain-specific language for design capture and exploration
- Specifications guide and document implementations
- Neutral to implementation platform
- Language should be high-level, yet detailed
  - Can talk about the bits, but in a platform-independent way
One Specification - Many Uses

Domain-specific design capture

Design

Build

Validate

Assured implementation

Formal Models and test cases

Verify crypto implementations

Hardware Implementation

Special purpose processor

Software Implementation

C, Haskell, ...

Cryptol

Cryptol Workbench

FPGA
Cryptol: Specifications and Formal Tools

- Domain-specific declarative specification language
  - Language tailored to the crypto domain
  - Designed with feedback from NSA
  - Non-proprietary language

- Execution and Validation Tools
  - Tool suite for different implementation and verification applications
  - In use by crypto-implementers
Key Ideas in Cryptol

- Domain-specific data and control abstractions
  - Sequences
  - Recurrence relations (not for-loops)
- Powerful data transformations
  - Data may be viewed in many ways
  - Machine independent
- Algorithms parameterized on size
  - Size constraints are explicit in many specs
  - Number of iterations may depend on size
  - A sized type system captures and maintains size constraints

Choosing what to leave out is critical
Cryptol Programs

- File of mathematical definitions
  - Two kinds of definitions: values and functions
  - Definitions may be accompanied by a type declarations (a signature)

- Definitions are computationally neutral
  - Cryptol tools provide the computational content (interpreters, compilers, code generators, verifiers)

```plaintext
x : [4][32];
x = [23 13 1 0];

F : ([16],[16]) -> [16];
F (x, x') = 2 * x + x';
```
Cryptol: Specify interfaces unambiguously

From the Advanced Encryption Standard definition†

### 3.1 Inputs and Outputs

The **input** and **output** for the AES algorithm each consist of sequences of **128 bits** (digits with values of 0 or 1). These sequences will sometimes be referred to as **blocks** and the number of bits they contain will be referred to as their length. The **Cipher Key** for the AES algorithm is a **sequence of 128, 192 or 256 bits**. Other input, output and Cipher Key lengths are not permitted by this standard.

**blockEncrypt** : \( \{ k \} (k \geq 2, 4 \geq k) \Rightarrow ([128], [64 \times k]) \rightarrow [128] \)

- For all \( k \)
- ...between 2 and 4
- First input is a sequence of 128 bits
- Second input is a sequence of 128, 192, or 256 bits
- Output is a sequence of 128 bits

Basic Cryptol Use Case

- Create a Cryptol reference specification
- Execute the specification, including assertion checks
- Generate test vectors with Quickcheck to bundle with the reference specification

Benefits:
- A clear and unambiguous model
  - E.g. bit-order and endian-ness
- Natural notation
  - Simplifies expression, inspection, and re-use
- Specification can be validated
  - Validate any part of algorithm
- Re-usable models
  - Validate, re-use many times
  - Specification for both hardware and software implementations
- Specifications can easily be re-factored
# Case Study: Cryptol in the development process

<table>
<thead>
<tr>
<th>Description/Purpose</th>
<th>Language</th>
<th>Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eg: NIST / NSA spec, technical paper</td>
<td>Pseudo-code/Mathematics</td>
<td>Conventional specification</td>
</tr>
<tr>
<td>Test understanding of specification</td>
<td>Cryptol</td>
<td>Reference model</td>
</tr>
<tr>
<td>Capture structure of implementation</td>
<td>Cryptol</td>
<td>Implementation model</td>
</tr>
<tr>
<td>Capture semantics of code fragments</td>
<td>Cryptol</td>
<td>Fragment models</td>
</tr>
<tr>
<td>Create code for proprietary platform</td>
<td>Microcode with Cryptol annotations</td>
<td>Implementation</td>
</tr>
</tbody>
</table>

![Diagram of Cryptol development process]

1. Cryptol Code
2. Published Test Vectors
3. Cryptol Interpreter
4. Reference Model
5. Current Model
6. Equivalence Checker
7. Assurance Evidence
8. Evaluation

© 2011 Galois, Inc. All rights reserved.
“The Cryptol specification removes ambiguities that are inevitable in the English-language descriptions and removes platform dependencies (like word-size) that creep into the C snippets.”

“...an experienced Cryptol programmer given a new crypto program specification and a soft copy of test vectors can be expected to learn the algorithm and have a fully functional and verified Cryptol model in a few days to a week.”

Alan Newman, General Dynamics C4 Systems
The SHA-3 Candidates in Cryptol

- Skein (Schneir et al.)
  - Galois verified two third-party VHDL implementations
- Blake (Aumasson et al., Switzerland)
  - Verification of third-party VHDL implementation in process
- CubeHash (Bernstein, USA)
- MD-6 (Rivest et al, since withdrawn from competition)
- SANDstorm (Sandia, since withdrawn from competition)
- Groestl (Knudsen et al, Denmark)
  - Students at U.Minho in Portugal generated a respectable FPGA implementation and verified it against the Cryptol specification
- Shabal (Misarsky, France)
  - Cryptol specification written at INRIA
Examples of Other Cryptol Tools

A Domain Specific Specification Language
- Precise, Declarative Semantics
- High level design exploration

Automated Synthesis down to FPGA
- Algebraic rewrite-based compilation
- Traceability back to specification

Automated Verification
- AIG-based Equivalence Checking
- SAT Solver technology
Cryptol in the VHDL Development Process

An FPGA engineer:
- Uses the reference specification to guide the VHDL implementation
- Produces intermediate specifications to reflect design decisions
- Generates test vectors to test portions of the VHDL
- Uses equivalence checkers to ensure that the implementation is correct
Cryptol in an FPGA Development Process: emphasis on high-level design

- It is easier to experiment with new microarchitectures in a specification language like Cryptol than in VHDL.
Cryptol in the evaluation process

A crypto-device evaluator:
- Creates a reference specification and associated formal model
- Checks the equivalence of the implementation models at several points in the tool

The process works for both hand-written and Cryptol-generated designs
Questions?

www.cryptol.net

- Language open
- Free download of interpreter
- Documentation