Cryptol version 2

The Language of Cryptography

What is Cryptol?

Cryptol is a domain-specific language for specifying cryptographic algorithms. A Cryptol implementation of an algorithm resembles its mathematical specification more closely than an implementation in a general purpose language.

Here is a comparison of a portion of the SHA-1 hash function specification and its representation in Cryptol.

Cryptol SHA-1 implementation

```
f : ([8], [32], [32], [32]) -> [32]
f (t, x, y, z) =
  if (0 <= t) && (t <= 19) then (x && y) ^ (~x && z)
   | (20 <= t) && (t <= 39) then x ^ y ^ z
   | (40 <= t) && (t <= 59) then (x && y) ^ (x && z) ^ (y && z)
   | (60 <= t) && (t <= 79) then x ^ y ^ z
   else error "f: t out of range"
```

The Cryptol implementation unambiguously captures both the English description and the mathematical specification below:

SHA-1 specification

SHA-1 uses a sequence of logical functions, $f_0, f_1, \ldots, f_{79}$. Each function f_t, where $0 \le t < 79$, operates on three 32-bit words, x, y, and z, and produces a 32-bit word as output. The function $f_t(x, y, z)$ is defined as follows:

$$f_t(x,y,z) = \begin{cases} Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x,y,z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x,y,z) = x \oplus y \oplus z & 60 \le t \le 79. \end{cases} \quad (4.1)$$

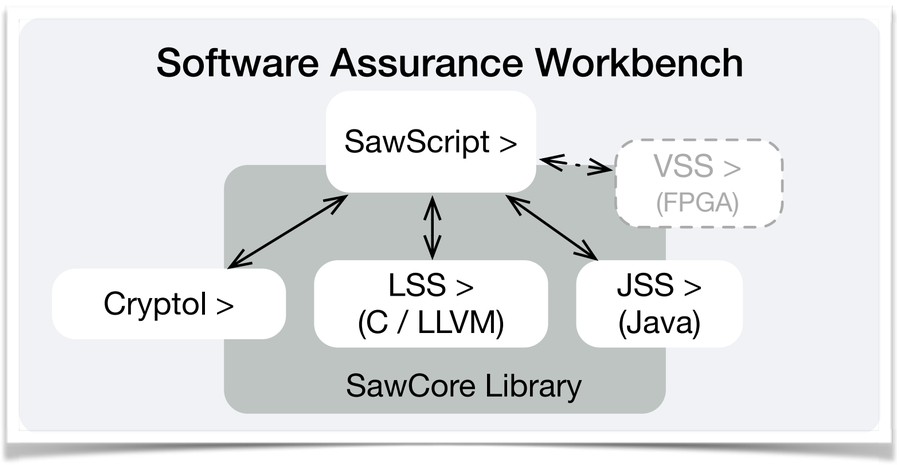
From page 10 of NIST 180-4 (http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf).

Write, Execute, Validate, and Verify Cryptographic Specifications

Recent news has highlighted the importance of correct cryptographic implementations for everyone. Previously, the challenge has been that cryptographic algorithms were written in academic papers in a non-executable mathematical notation. Someone often writes a reference implementation which does not usually look very much like the math. People then use or optimize that reference implementation in their applications, but there has not been an easy way to check those implementations against the mathematical specification.

Using a specification in Cryptol, programmers can generate their own test vectors, generate counter-examples, prove theorems, and (using other tools, like Galois' *SAW*, illustrated below) verify equivalence to their own programs, or even generate code or hardware from the specification.

Cryptol version 2 is now released as open source under a 3-clause BSD license on GitHub (`http://cryptol.net/`). Our goal is that it becomes a widely adopted standard for expressing, validating, and verifying cryptographic algorithms.



Cryptol's relationship with the Software Assurance Workbench (SAW)

Changes from version 1

Cryptol version 1 has been in use for a number of years. Cryptol version 2 makes some changes based on suggestions from the user community and lessons learned by the Cryptol design team. These include syntax changes and some extensions to the type system. Perhaps the the most disruptive change for current Cryptol programmers is that Cryptol version 2 interprets sequences in "big endian" mode, rather than "little endian."

Summary of Syntax Changes

Here is a short summary of the syntax changes made in Cryptol version 2 based upon user community input:

Cryptol version 2	Cryptol version 1	Summary
`[ False, True, True ] (==3)`	`[ False True True ] (== 6)`	Big-endian word representation
`[ 1, 1, 2, 3, 5 ]`	`[ 1 1 2 3 5 ]`	Commas separate sequence entries
`x = 1`	`x = 1;`	Uses *layout* instead of ;'s and {'s
`[ x | x <- [1 .. 10] ]`	`[| x || x <- [ 1 .. 10] |]`	Cleaner sequence constructor syntax
`f : {a,b} a -> b`	`f : {a b} a -> b`	Commas separate type variables
`take‘{1} xs`	`take(1, xs)`	First-class type parameters
`x ^^ 2`	`x ** 2`	`^^` for exponentiation
`<| x^^2 + 1 |>`	`<| x^2 + 1 |>`	Polynomial exponentiation now uniform
`[0 ..]:[_][8]`	`take(255, [0 ..]:[inf][8])`	Both produce `[0 .. 255]`
`[0 ...]:[inf][8]`	`[0 ..]:[inf][8]`	Both produce `[0 .. 255]`(repeated)
`[9, 8 .. 0]`	`[9 -- 0]`	Step defines decreasing sequences
`&&, ||, ^`	`&, |, ^`	Boolean operator syntax
`property foo xs=...`	`theorem foo: {xs}. xs==...`	Properties replace theorems (see below)

New Features in Cryptol version 2

- **Layout**: Version 1 of Cryptol used curly braces to delimit blocks and semicolons to separate expressions. In version 2, Cryptol has layout-based syntax which uses indentation to delimit blocks and \ to indicate line continuation.
  Here is the caesar cipher specified in the old and the new layouts:

```
caesar : {n} ([8], String n) -> String n;
caesar (s, msg) = [| shift x || x <- msg |] where {
    map     = ['A' .. 'Z'] <<< s;
    shift c = map @ (c - 'A');
};
```

```
caesar : {n} ([8], String n) -> String n
caesar (s, msg) = [ shift x | x <- msg ]
    where map     = ['A' .. 'Z'] <<< s
          shift c = map @ (c - 'A')
```

- **Multi-way If-Then-Else**: Cryptol version 2 supports a "case-statement"-like multi-way branch:

```
x = if y % 2 == 0 then 1
  | y % 3 == 0 then 2
  | y % 5 == 0 then 3
  else 7
```

- **First-class Type Variables**: Function definitions can now be parameterized over type-level constants, thus Cryptol is proper size-polymorphic dependently typed language. Type-level constants can be instantiated either positionally or by name. For example, the built-in `take` function in Cryptol has the following signature and definition:

```
take: {front, back, elem} (fin front) =>
      [front + back] elem -> [front] elem
take (x # _) = x
```

  This lets us call it like `take`{3}xs`, which means the same thing as `take`{front=3}xs`, which means the same thing as the Cryptol version 1 call `take(3,xs)`.

Cryptol is Open Source

By open sourcing Cryptol we hope that educators, applied cryptographers, and crypto system implementers will adopt Cryptol as a de facto standard for the specification and verification of cryptographic algorithms.

With the community's help, we plan to extend Cryptol to specify and reason about a broader scope of cryptographic systems, including protocols. Your ideas and contributions are welcome!

Sponsored by the ***Trusted Systems Research Group***
For more information, visit http://cryptol.net/