hacspec

towards verifiable cryptographic specifications

Karthikeyan Bhargavan
IETF 101

Implementing crypto correctly is hard

- Memory safety bugs
- Side channel leaks
- Functional correctness bugs

- Testing is inadequate for low-probability bugs
- Formal verification can provide high assurance

  … but it requires effort and expertise

High Assurance Crypto Software

- Verification results for C implementations
  - *Primitives:* SHA-2, Chacha20, Poly1305, AES-GCM, MEE-CBC, Curve25519, Ed25519, NIST P-256, RSA-OAEP
  - *Tools*: Cryptol/SAW, Coq (VST, Fiat-Crypto), F*, EasyCrypt
- Verification results for assembly implementations
  - *Primitives:* SHA-2, Poly1305, AES-GCM, Curve25519
  - *Tools*: Vale, Boolector, Cryptol/SAW, Jasmin
- Research now applied to mainstream libraries
  - Mozilla NSS, Google boringssl, Amazon s2n, Microsoft Everest

How do you verify crypto code?

- Write a formal specification that states desired goals
  - correctness, memory safety, side-channels, crypto security, …
- Prove that your implementation meets this spec

Verification Methods	Implementation	Specification
Cryptol/SAW	C, Java, assembly	Cryptol
HACL*, Vale	C, x86/arm assembly	F*
VST, Fiat-Crypto	C	Coq
EasyCrypt	C	EasyCrypt

Writing formal crypto specifications

- HACS Workshop 2016-2018
  - Co-located with Real World Crypto
  - Discussions between crypto developers and verification experts
- Difficult for developers to understand, compare, compose proofs based on “obscure” spec languages
- We need specs that crypto designers can read/write
  - A single target for verification, in a well-understood syntax

hacspec: a new specification language

Design Goals:

- Succinct and readable
  - Can be integrated into RFCs as pseudocode
- Executable
  - Can be treated as a reference implementation
- Compact formal semantics
  - Can be used as a formal spec for verification

hacspec: a new specification language

Version 1 (feedback needed):

- A subset of python 3.6 (with type annotations)
  - Native bignums and arrays, not much else
  - Types enable static checking and precise translations

- Compilers to various formal languages
  - Translations to F*, EasyCrypt, Cryptol, Coq
- Library of specifications and common constructions
  - AEAD-Chacha20-Poly1305, SHA-2, (kyber, xmss, blake2,...)

Example: poly1305

```
p130m5 = (2 ** 130) - 5
felem_t = refine(nat,lambda x: x < p130m5)
def felem(x:nat) -> felem_t:
    return (x % p130m5)
def fadd(x:felem_t,y:felem_t) -> felem_t:
    return felem(x + y)
def fmul(x:felem_t,y:felem_t) -> felem_t:
    return felem(x * y)
```

Example: chacha20

```
index_t  = range_t(0,16)
rotval_t = range_t(1,32)
state_t  = array_t(uint32_t,16)

def line(a: index_t, b: index_t, d: index_t, s: rotval_t, m: state_t) -> state_t:
    m    = array.copy(m)
    m[a] = m[a] + m[b]
    m[d] = m[d] ^ m[a]
    m[d] = uint32.rotate_left(m[d],s)
    return m

def quarter_round(a: index_t, b: index_t, c:index_t, d: index_t, m: state_t) -> state_t :
    m = line(a, b, d, 16, m)
    m = line(c, d, b, 12, m)
    m = line(a, b, d,  8, m)
    m = line(c, d, b,  7, m)
    return m
```

Example: chacha20 compiled to F*

```
let index_t = range_t 0x0 0x10
let rotval_t = range_t 0x1 0x20
let state_t = array_t uint32_t 0x10

let line (a:index_t) (b:index_t) (d:index_t) (s:rotval_t) (m:state_t) : state_t =
  let m = copy m in
  let m = m.[a] ← (m.[a] +. m.[b]) in
  let m = m.[d] ← (m.[d] ^. m.[a]) in
  let m = m.[d] ← rotate_left m.[d] (u32 s) in
  m
let quarter_round (a:index_t) (b:index_t) (c:index_t) (d:index_t) (m:state_t) : state_t =
  let m = line a b d 0x10 m in
  let m = line c d b 0xc m in
  let m = line a b d 0x8 m in
  let m = line c d b 0x7 m in
  m
```

Example: verified chacha20 in C

```
static void
QR(unsigned int* x,
   unsigned int a,
   unsigned int b,
   unsigned int c,
   unsigned int d)
{
  x[a] = x[a]+x[b]; x[d] = L32(x[d]^x[a],16);
  x[c] = x[c]+x[d]; x[b] = L32(x[b]^x[c],12);
  x[a] = x[a]+x[b]; x[d] = L32(x[d]^x[a], 8);
  x[c] = x[c]+x[d]; x[b] = L32(x[b]^x[c], 7);
}
```

Can also verify optimized vectorized code in C or assembly against same spec

We need you

- Interested in using hacspec in your next RFC?
  - As a formal specification and prototype implementation
  - Help promote high-assurance implementations
- Give us feedback on the hacspec language/specs
  - What features are we missing? What will make it more usable?
  - Ongoing compilers to EasyCrypt, Cryptol, Coq
  - Ongoing specs for SHA-3, PQ crypto, argon2, …

- Code: https://github.com/HACS-workshop/hacspec
- List: https://moderncrypto.org/mailman/listinfo/hacspec