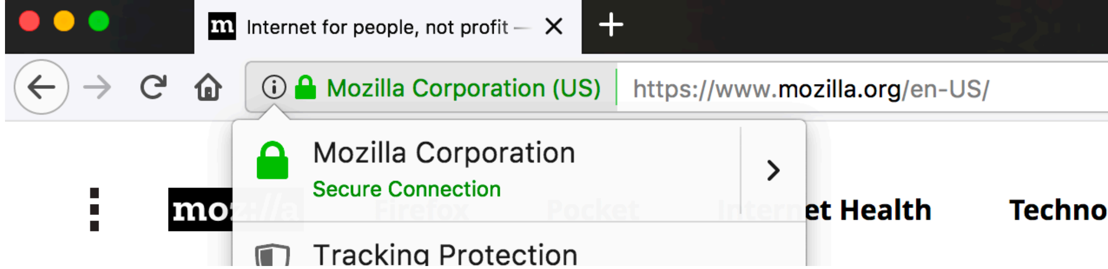



# HACL* in Mozilla Firefox

*Formal methods and high assurance applications for the web*


B. Beurdouche
K. Bhargavan
J. Protzenko
J-K. Zinzindohoué
**(Project Everest)**

F. Kiefer
E. Rescorla
T. Taubert
M. Thomson
**(Mozilla)**


Real World Crypto 2018

# Let’s focus on Crypto[graphy] !

# Implementing cryptography is difficult

Memory Safety

(think Heartbleed)

Side channels

(think Lucky 13)

Functional correctness

# Functional correctness is difficult

[2016] Integer overflow in OpenSSL's Poly1305

```
/* last reduction step: */
/* a) h2:h0 = h2<<128 + d1<<64 + d0 */
h0 = (u64)d0;
h1 = (u64)(d1 += d0 >> 64);
h2 += (u64)(d1 >> 64);
/* b) (h2:h0 += (h2:h0>>130) * 5) %= 2^130 */
c = (h2 >> 2) + (h2 & ~3UL);
h2 &= 3;
h0 += c;
h1 += (c = CONSTANT_TIME_CARRY(h0,c));    /* doesn't overflow */
```

It does !

# Implementing is hard for everyone

agl / **curve25519-donna**

Watch 20

Code | Issues 2 | Pull requests 7 | Projects 0 | Wiki | Insights

[2014] Curve25519-Donna

### Correct bounds in 32-bit code.

```
The 32-bit code was illustrative of the tricks used in the original
curve25519 paper rather than rigorous. However, it has proven quite
popular.

This change fixes an issue that Robert Ransom found where outputs between
2^255-19 and 2^255-1 weren't correctly reduced in fcontract. This
appears to leak a small fraction of a bit of security of private keys.

Additionally, the code has been cleaned up to reflect the real-world
needs. The ref10 code also exists for 32-bit, generic C but is somewhat
slower and objections around the lack of qhasm availibility have been
raised.
```

master 1.3

agl committed on Jun 9, 2014 1 parent c22bb55 commit 2647eeba59fb628

[2014] TweetNaCl

```c
sv pack25519(u8 *o, const gf n)
{
  int i,j,b;
  gf m,t;
  FOR(i,16) t[i]=n[i];
  car25519(t);
  car25519(t);
  car25519(t);
  FOR(j,2) {
    m[0]=t[0]-0xffed;
    for(i=1;i<15;i++) {
      m[i]=t[i]-0xffff-((m[i-1]>>16)&1);
      m[i-1]&=0xffff;
    }
    m[15]=t[15]-0x7fff-((m[14]>>16)&1);
    b=(m[15]>>16)&1;
    m[15]&=0xffff;
    sel25519(t,m,1-b);
  }
  FOR(i,16) {
    o[2*i]=t[i]&0xff;
    o[2*i+1]=t[i]>>8;
  }
}
```

This bug is triggered when the last limb `n[15]` of the input argument `n` of this function is greater or equal than `0xffff`. In these cases the result of the scalar multiplication is not reduced as expected resulting in a wrong packed value. This code can be fixed simply by replacing `m[15]&=0xffff;` by `m[14]&=0xffff;`.

Even for very skilled programmers or cryptographers !

# Network Security Services (NSS) library

## Multi product security library

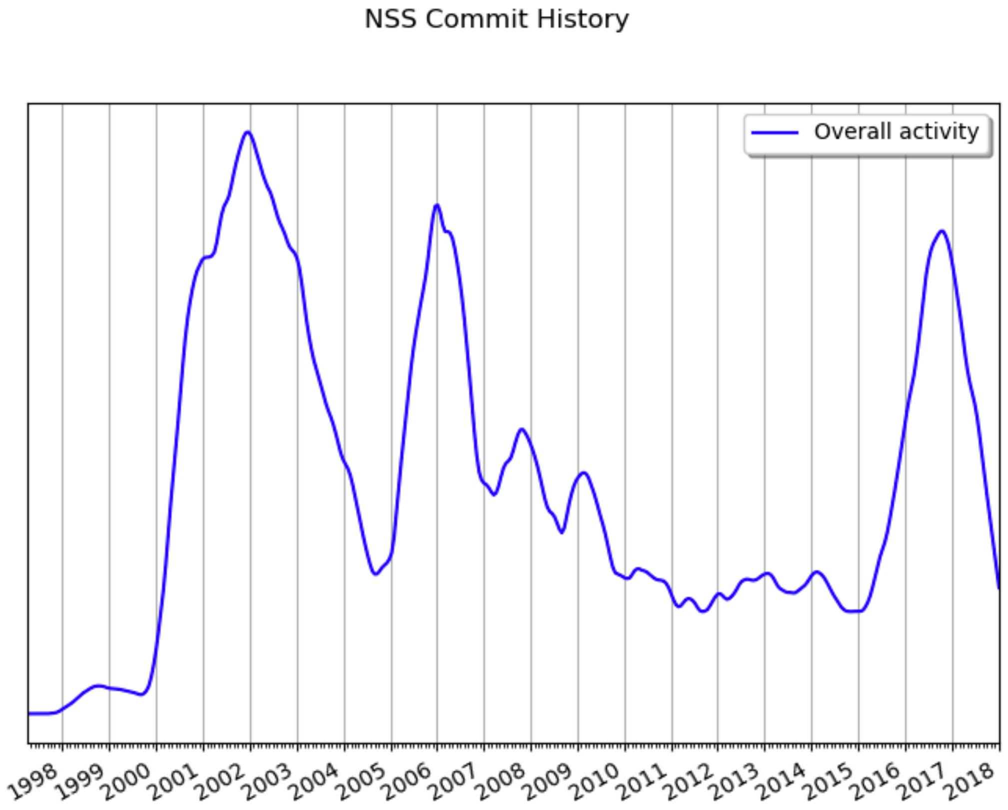
- Joint effort from Mozilla, RedHat…
- Security Library for Firefox in C/C++
- Used in RHEL, Fedora, BSDs…

## Large number of primitives

- Both recent and legacy primitives for interoperability

## Higher level components

- Protocols (TLS…)
- Cryptographic APIs (WebCrypto, PKCS...)



# Redesigning NSS

"NSS is old, there is a lot of legacy code"

"How can we make NSS more modern and get higher confidence in its correctness ?"

moz://a

There was no clear way on how to get there...

- Clean room redesign "à la BoringSSL"
- More money ?! More hiring ?!

Decision

- Improve step-by-step the confidence in code correctness using formal verification

# Research challenge from the OpenSSL team

## How can the community help?

- Formal verification of crypto code
  - Hitting < 2^{-64} corner cases with unit testing is difficult.
  - New-ish elliptic curve implementations: P-224, P-256, P-521 - fast and constant-time. But are they correct?
  - Regression testing (again!) for bug attacks and oracle attacks.

Emilia Kasper, Real World Crypto (2015)

# Formal methods inbound

## Recent academic developments for Cryptography

**Verification of a Cryptographic Primitive: SHA-256**

ANDREW W. APPEL, Princeton University

**Verifying Curve25519 Software**

g Chen[1], Chang-Hong Hsu[2], Hsin-Hung Lin[3], Peter Schwabe[4], Mi
Bow-Yaw Wang[1], Bo-Yin Yang[1], and Shang-Yi Yang[1] *

[1] Institute of Information Science
Academia Sinica
128 Section 2 Academia Road, Taipei 115-29, Taiwan

**Verified correctness and security of OpenSSL HMAC**

To appear in 24th Usenix Security Symposium, August 12, 2015

Lennart Beringer, *Princeton Univ.*
Adam Petcher, *Harvard Univ. and*
Katherine Q. Ye, *Princeton Univ.*
Andrew W. Appel, *Princeton Univ.*

**Verifiable side-channel security of cryptographic implementations: constant-time MEE-CBC**

José Bacelar Almeida[12], Manuel Barbosa[13], Gilles Barthe[4], and François Dupressoir[4]

[1] HASLab – INESC TEC
[2] University of Minho
[3] DCC-FC, University of Porto
[4] IMDEA Software Institute

**Verifying Constant-Time Implementations**

José Bacelar Almeida, *HASLab - INESC TEC & Univ. Minho*
Manuel Barbosa, *HASLab - INESC TEC & DCC FCUP*
Gilles Barthe, *IMDEA Software Institute*
François Dupressoir, *IMDEA Software Institute*
Michael Emmi, *Bell Labs, Nokia*

**"Automated Verification of Real-World Cryptographic Implementations"**, Aaron Tomb, *IEEE Security & Privacy*, vol. 14, no. , pp. 26-33, Nov.-Dec. 2016

# What kind of verification and how ?

PORTABILITY

PROOF EFFORT

READABILITY

VERIFICATION TIME

Assembly, C or High-Level Languages ?

Code generation or Verification of existing code ?

COMPILER TRUST

PERFORMANCE

SIDE-CHANNELS

CCS 2017 - https://eprint.iacr.org/2017/536

# HACL*: A Verified Modern Cryptographic Library


Jean Karim Zinzindohoué
INRIA

Karthikeyan Bhargavan
INRIA

Jonathan Protzenko
Microsoft Research

Benjamin Beurdouche
INRIA



## ABSTRACT

HACL* is a verified portable C cryptographic library that implements modern cryptographic primitives such as the ChaCha20 and Salsa20 encryption algorithms, Poly1305 and HMAC message authentication, SHA-256 and SHA-512 hash functions, the Curve25519 elliptic curve, and Ed25519 signatures.

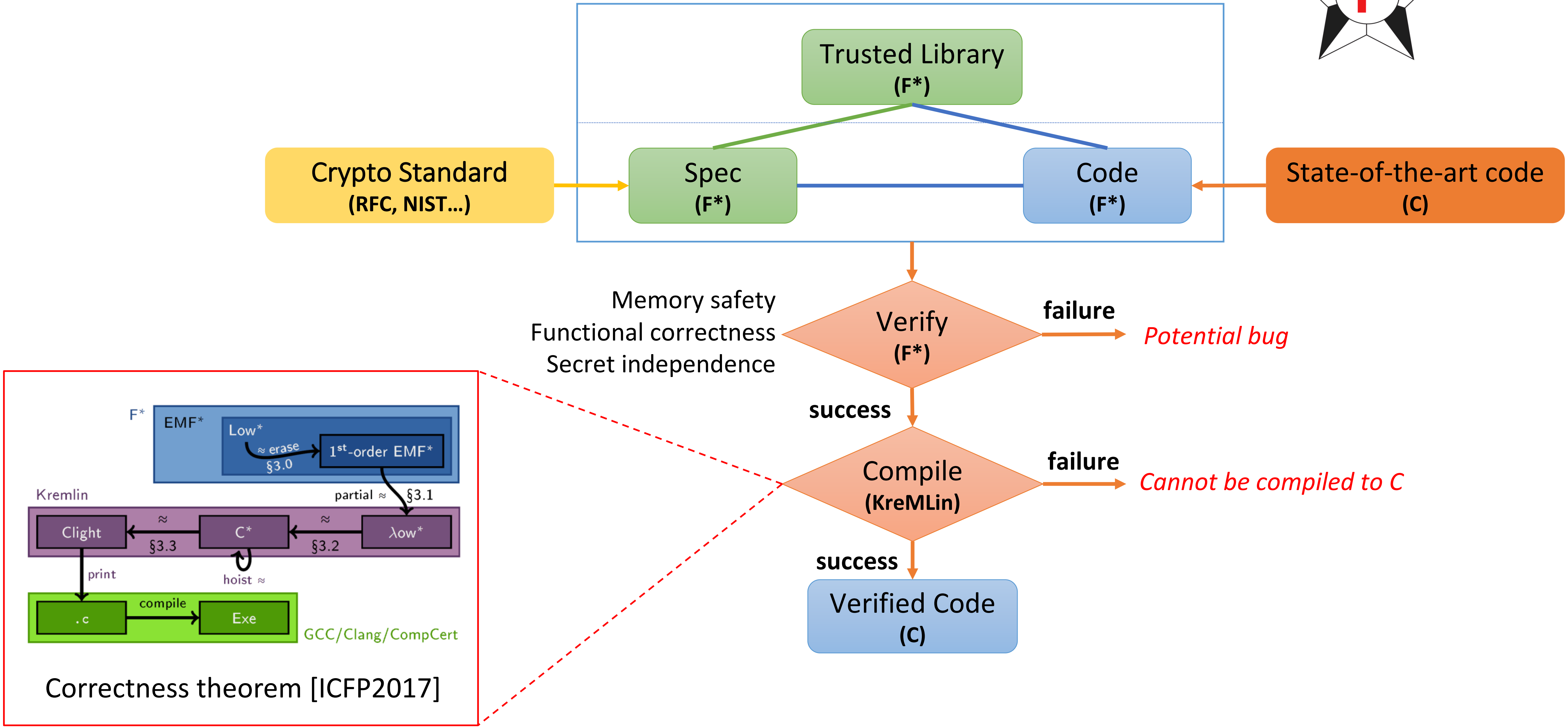
HACL* is written in the F* programming language and then compiled to readable C code. The F* source code for each cryptographic primitive is verified for memory safety, mitigations against timing side-channels, and functional correctness with respect to a succinct high-level specification of the primitive derived from its published standard. The translation from F* to C preserves these properties and the generated C code can itself be compiled via the CompCert verified C compiler or mainstream compilers like GCC or CLANG. When compiled with GCC on 64-bit platforms, our primitives are as fast as the fastest pure C implementations in OpenSSL and libsodium, significantly faster than the reference C code in TweetNaCl, and between 1.1x-5.7x slower than the fastest hand-optimized vectorized assembly code in SUPERCOP.

HACL* implements the NaCl cryptographic API and can be used as a drop-in replacement for NaCl libraries like libsodium and


the absence of entire classes of potential bugs. In this paper, we will show how to implement a cryptographic library and prove that it is memory safe and functionally correct with respect to its published standard specification. Our goal is to write verified code that is as fast as state-of-the-art C implementations, while implementing standard countermeasures to timing side-channel attacks.

**A Library of Modern Cryptographic Primitives.** To design a high-assurance cryptographic library, we must first choose which primitives to include. The more we include, the more we have to verify, and their proofs can take considerable time and effort. Mixing verified and unverified primitives in a single library would be dangerous, since trivial memory-safety bugs in unverified code often completely break the correctness guarantees of verified code. General-purpose libraries like OpenSSL implement a notoriously large number of primitives, totaling hundreds of thousands of lines of code, making it infeasible to verify the full library. In contrast, minimalist easy-to-use libraries such as NaCl [17] support a few carefully chosen primitives and hence are better verification targets. For example, TweetNaCl [19], a portable C implementation of NaCl is fully implemented in 700 lines of code.

# F* verification workflow



# HACL* - High Assurance Crypto Library

CCS 2017 - https://eprint.iacr.org/2017/536

Formal verification can scale up !

Functionalities

- Hash function (SHA-2)
- Message authentication (HMAC, Poly1305)
- Symmetric ciphers (Chacha20, Salsa20)
- Key Exchange algorithm (Curve25519)
- Signature scheme (Ed25519)
- AEAD (Chacha20Poly1305)

| **Algorithm** | Spec (F* loc) | Code+Proofs (Low* loc) | C Code (C loc) | Verification (s) |
|---|---|---|---|---|
| Salsa20 | 70 | 651 | 372 | 280 |
| Chacha20 | 70 | 691 | 243 | 336 |
| Chacha20-Vec | 100 | 1656 | 355 | 614 |
| SHA-256 | 96 | 622 | 313 | 798 |
| SHA-512 | 120 | 737 | 357 | 1565 |
| HMAC | 38 | 215 | 28 | 512 |
| Bignum-lib | - | 1508 | - | 264 |
| Poly1305 | 45 | 3208 | 451 | 915 |
| X25519-lib | - | 3849 | - | 768 |
| Curve25519 | 73 | 1901 | 798 | 246 |
| Ed25519 | 148 | 7219 | 2479 | 2118 |
| AEAD | 41 | 309 | 100 | 606 |
| SecretBox | - | 171 | 132 | 62 |
| Box | - | 188 | 270 | 43 |
| **Total** | 801 | 22,926 | 7,225 | 9127 |

**Table 1: HACL* code size and verification times**

# Specification for Poly1305

```
module Spec.Poly1305

(* Field types and parameters *)
let prime = pow2 130 - 5
type elem = e:ℤ{e ≥ 0 ∧ e < prime}
let fadd (e1:elem) (e2:elem) = (e1 + e2) % prime
let fmul (e1:elem) (e2:elem) = (e1 × e2) % prime
let zero : elem = 0
let one  : elem = 1

(* Specification code *)
let encode (w:word) =
  (pow2 (8 × length w)) `fadd` (little_endian w)

let rec poly (txt:text) (r:e:elem) : Tot elem (decreases (length txt)) =
  if length txt = 0 then zero
  else
    let a = poly (Seq.tail txt) r in
    let n = encode (Seq.head txt) in
    (n `fadd` a) `fmul` r

let encode_r (rb:word_16) =
  (little_endian rb) &| 0x0ffffffc0ffffffc0ffffffc0fffffff

let finish (a:elem) (s:word_16) : Tot tag =
  let n = (a + little_endian s) % pow2 128 in
  little_bytes 16ul n

let rec encode_bytes (txt:bytes) : Tot text (decreases (length txt)) =
  if length txt = 0 then createEmpty
  else
    let w, txt = split txt (min (length txt) 16) in
    append_last (encode_bytes txt) w

let poly1305 (msg:bytes) (k:key) : Tot tag =
  let text = encode_bytes msg in
  let r = encode_r (slice k 0 16) in
  let s = slice k 16 32 in
  finish (poly text r) s
```

# How does the stateful code and proofs look like ?

Hacl.Impl.Poly1305_64.fst

File Edit Options Buffers Tools F★ Help

## Low* code

memory safety

math spec

code

proof

```
[@"substitute"]
val poly1305_last_pass_:
  acc:felem →
  Stack unit
    (requires (λ h → live h acc ∧ bounds (as_seq h acc) p44 p44 p42))
    (ensures (λ h0 _ h1 → live h0 acc ∧ bounds (as_seq h0 acc) p44 p44 p42
      ∧ live h1 acc ∧ bounds (as_seq h1 acc) p44 p44 p42
      ∧ modifies_1 acc h0 h1
      ∧ as_seq h1 acc == Hacl.Spec.Poly1305_64.poly1305_last_pass_spec_ (as_seq h0 acc)))

[@"substitute"]
let poly1305_last_pass_ acc =
  let a0 = acc.(0ul) in
  let a1 = acc.(1ul) in
  let a2 = acc.(2ul) in
  let open Hacl.Bignum.Limb in
  let mask0 = gte_mask a0 Hacl.Spec.Poly1305_64.p44m5 in
  let mask1 = eq_mask a1 Hacl.Spec.Poly1305_64.p44m1 in
  let mask2 = eq_mask a2 Hacl.Spec.Poly1305_64.p42m1 in
  let mask  = mask0 &^ mask1 &^ mask2 in
  UInt.logand_lemma_1 (v mask0); UInt.logand_lemma_1 (v mask1); UInt.logand_lemma_1 (v mask2);
  UInt.logand_lemma_2 (v mask0); UInt.logand_lemma_2 (v mask1); UInt.logand_lemma_2 (v mask2);
  UInt.logand_associative (v mask0) (v mask1) (v mask2);
  cut (v mask = UInt.ones 64 ⟹ (v a0 ≥ pow2 44 - 5 ∧ v a1 = pow2 44 - 1 ∧ v a2 = pow2 42 - 1));
  UInt.logand_lemma_1 (v Hacl.Spec.Poly1305_64.p44m5); UInt.logand_lemma_1 (v Hacl.Spec.Poly1305_64.p44m1);
  UInt.logand_lemma_1 (v Hacl.Spec.Poly1305_64.p42m1); UInt.logand_lemma_2 (v Hacl.Spec.Poly1305_64.p44m5);
  UInt.logand_lemma_2 (v Hacl.Spec.Poly1305_64.p44m1); UInt.logand_lemma_2 (v Hacl.Spec.Poly1305_64.p42m1);
  let a0' = a0 -^ (Hacl.Spec.Poly1305_64.p44m5 &^ mask) in
  let a1' = a1 -^ (Hacl.Spec.Poly1305_64.p44m1 &^ mask) in
  let a2' = a2 -^ (Hacl.Spec.Poly1305_64.p42m1 &^ mask) in
  upd_3 acc a0' a1' a2'
```

-:**- **Hacl.Impl.Poly1305_64.fst** 55% L394 Git-master (F★ FlyC- company ElDoc Wrap)

Poly1305_64.c

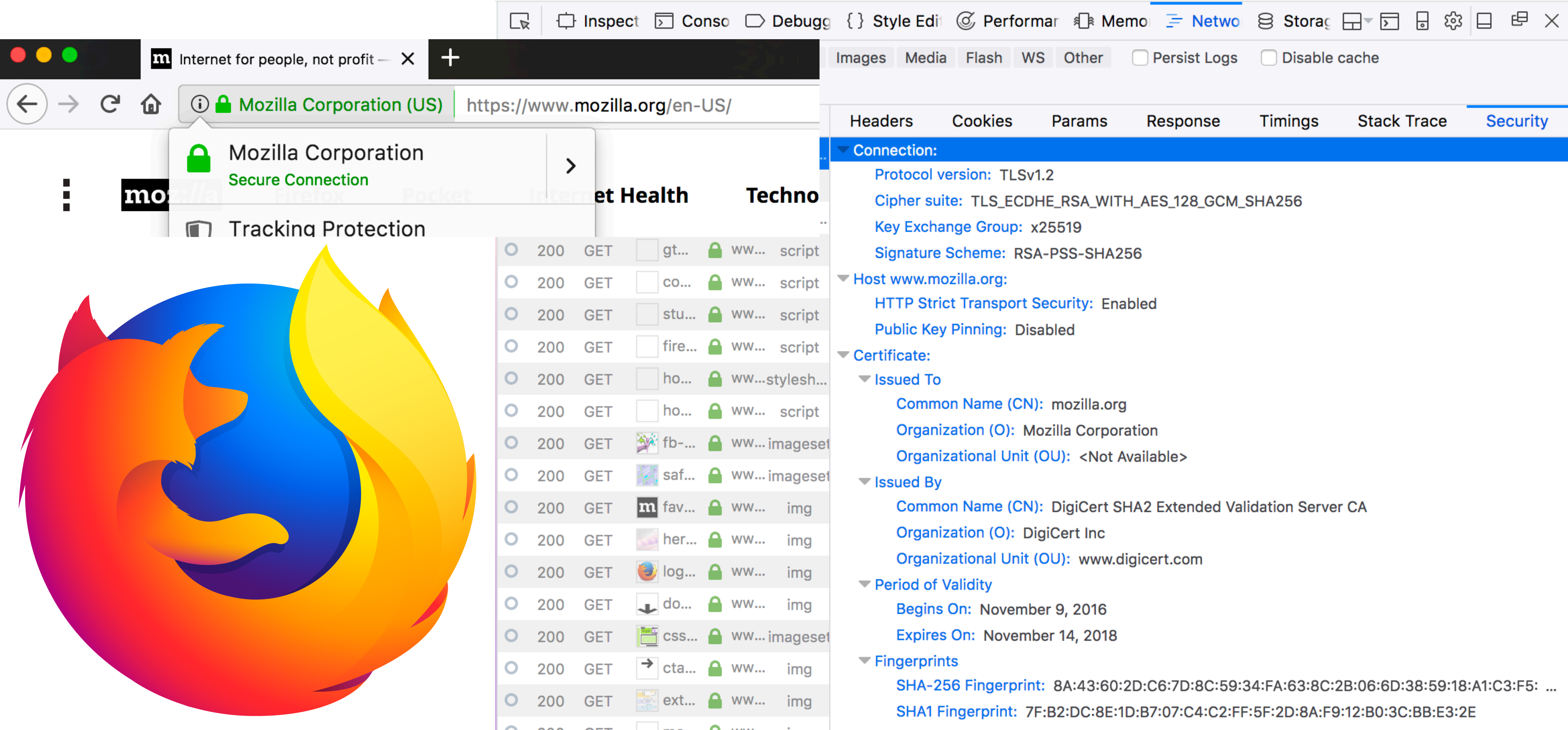
File Edit Options Buffers Tools C Help

## C code

```
static void Hacl_Impl_Poly1305_64_poly1305_last_pass(uint64_t *acc)
{
  Hacl_Bignum_Fproduct_carry_limb_(acc);
  Hacl_Bignum_Modulo_carry_top(acc);
  uint64_t a0 = acc[0];
  uint64_t a10 = acc[1];
  uint64_t a20 = acc[2];
  uint64_t a0_ = a0 & (uint64_t )0xfffffffffff;
  uint64_t r0 = a0 >> (uint32_t )44;
  uint64_t a1_ = (a10 + r0) & (uint64_t )0xfffffffffff;
  uint64_t r1 = (a10 + r0) >> (uint32_t )44;
  uint64_t a2_ = a20 + r1;
  acc[0] = a0_;
  acc[1] = a1_;
  acc[2] = a2_;
  Hacl_Bignum_Modulo_carry_top(acc);
  uint64_t i0 = acc[0];
  uint64_t i1 = acc[1];
  uint64_t i0_ = i0 & (((uint64_t )1 << (uint32_t )44) - (uint64_t )1);
  uint64_t i1_ = i1 + (i0 >> (uint32_t )44);
  acc[0] = i0_;
  acc[1] = i1_;
  uint64_t a00 = acc[0];
  uint64_t a1 = acc[1];
  uint64_t a2 = acc[2];
  uint64_t mask0 = FStar_UInt64_gte_mask(a00, (uint64_t )0xffffffffffb);
  uint64_t mask1 = FStar_UInt64_eq_mask(a1, (uint64_t )0xfffffffffff);
  uint64_t mask2 = FStar_UInt64_eq_mask(a2, (uint64_t )0x3ffffffffff);
  uint64_t mask = mask0 & mask1 & mask2;
  uint64_t a0_0 = a00 - ((uint64_t )0xffffffffffb & mask);
  uint64_t a1_0 = a1 - ((uint64_t )0xfffffffffff & mask);
  uint64_t a2_0 = a2 - ((uint64_t )0x3ffffffffff & mask);
  acc[0] = a0_0;
  acc[1] = a1_0;
  acc[2] = a2_0;
}
```

-:**- **Poly1305_64.c** 49% L272 Git-master (C/l company A

# HACL* in Mozilla Firefox



# HACL* in Mozilla Firefox

## Firefox 57 "Quantum" was a major release for Mozilla

- Includes verified cryptography from HACL* (Curve25519)

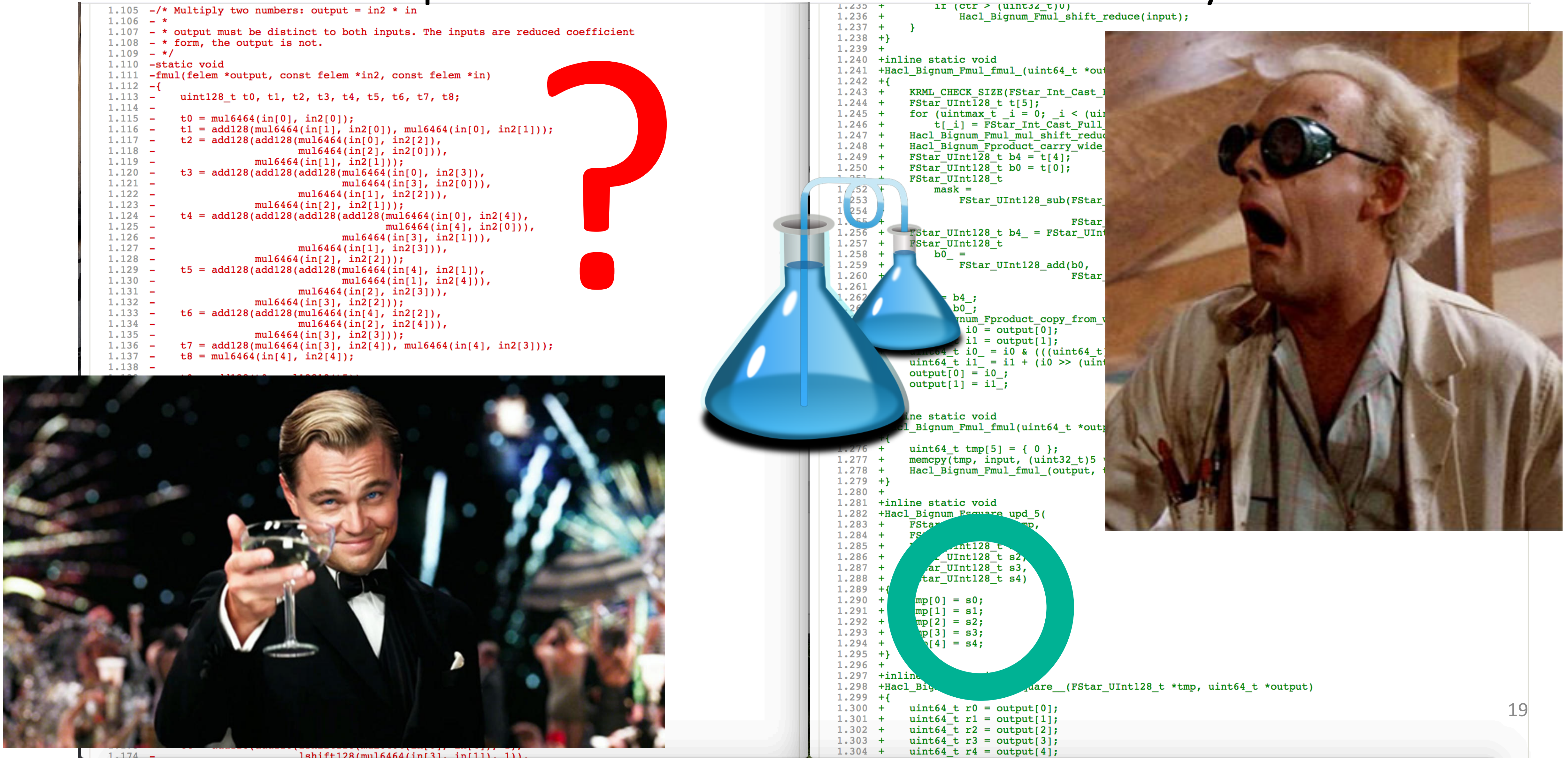
## Firefox Nightly already has more

- Chacha20 and Poly1305

## Next batch of primitives on its way

- Vectorized Chacha20Poly1305 + Ed25519
- SHA2 + HMAC + HKDF
- RSA_PSS + P256 …

# How does one go from an academic project to production code in the industry?



# Integration process constraints

## Performance

- Reducing performance is not acceptable (in general)

## Code integration

- Readable, reviewable code

## Toolchain integration

- Insert verification into the current dev. workflow

## Deployment and support

- NSS runs on almost everything
- API and ABI stability

# HACL* Performance (C code)

**CPU cycles/byte**

Lower is better

Encrypt, Hash, or MAC 16KB

1 Diffie-Hellman

Sign, verify 16KB

| Algorithm | HACL* | OpenSSL | libsodium | TweetNaCl | OpenSSL (asm) |
|---|---|---|---|---|---|
| SHA-256 | 13.43 | 16.11 | 12.00 | - | 7.77 |
| SHA-512 | 8.09 | 10.34 | 8.06 | 12.46 | 5.28 |
| Salsa20 | 6.26 | - | 8.41 | 15.28 | - |
| ChaCha20 | 6.37 (ref)<br>2.87 (vec) | 7.84 | 6.96 | - | 1.24 |
| Poly1305 | 2.19 | 2.16 | 2.48 | 32.65 | 0.67 |
| Curve25519 | 154,580 | 358,764 | 162,184 | 2,108,716 | - |
| Ed25519 sign | 63.80 | - | 24.88 | 286.25 | - |
| Ed25519 verify | 57.42 | - | 32.27 | 536.27 | - |
| AEAD | 8.56 (ref)<br>5.05 (vec) | 8.55 | 9.60 | - | 2.00 |
| SecretBox | 8.23 | - | 11.03 | 47.75 | - |
| Box | 21.24 | - | 21.04 | 148.79 | - |

+20 % faster than previous NSS code

# Code review (Phabricator)

```
    if (i == (uint32_t)0) {

    } else {
```

**ekr** Not Done

WAT?

```
      uint32_t i_ = i - (uint32_t)1;
      Hacl_EC_Ladder_SmallLoop_cmult_small_loop_double_step(nq, nqpq, nq2, nqpq2, q, byt);
      uint8_t byt_ = byt << (uint32_t)2;
      Hacl_EC_Ladder_SmallLoop_cmult_small_loop(nq, nqpq, nq2, nqpq2, q, byt_, i_);
    }
}
```

Removing empty branches, unreachable code…

# Improving code quality

```diff
 inline static void
 Hacl_Bignum_AddAndMultiply_add_and_multiply(uint64_t *acc,
 {
   for (uint32_t i = (uint32_t )0; i < (uint32_t )3; i = i
   {
-    uint64_t uu____871 = acc[i];
-    uint64_t uu____874 = block[i];
-    uint64_t uu____870 = uu____871 + uu____874;
-    acc[i] = uu____870;
   }
   Hacl_Bignum_Fmul_fmul(acc, acc, r);
 }
```

```diff
 inline static void
 Hacl_Bignum_AddAndMultiply_add_and_multiply(uint64_t *acc,
 {
   for (uint32_t i = (uint32_t )0; i < (uint32_t )3; i = i
   {
+    uint64_t xi = acc[i];
+    uint64_t yi = block[i];
+    acc[i] = xi + yi;
   }
   Hacl_Bignum_Fmul_fmul(acc, acc, r);
 }
```

Better variable naming

Removing intermediate variables

# HACL* verification toolchain in NSS CI (treeherder)

**Bug 1395549 (hacl-ci)**

**CI integration for HACL* code**

**RESOLVED FIXED** in 3.33

Fri Dec 1, 16:24:18 - franziskuskiefer@gmail.com

| | | |
|---|---|---|
| ecab583d0466 BB *Bug 1399763 - formally verified code from HACL* for Poly1305 (64bits, non-vectoriz* | nss-decision opt | D |
| ce395e6b2908 TT *Bug 1422326 - Use fewer layers in HACL* docker image r=franziskus* | | |

Fri Dec 8, 02:11:27 - franziskuskiefer@gmail.com

| | | |
|---|---|---|
| b70446c6adc0 FK *try: -p none -t hacl* | nss-tools opt | hacl |
| | nss-decision opt | D I |

Fri Dec 8, 01:02:43 - franziskuskiefer@gmail.com

| | | |
|---|---|---|
| d6c3ae0cbe92 FK *try: -p none -t hacl* | nss-tools opt | hacl |
| | nss-decision opt | D I |

Thu Dec 7, 23:24:25 - franziskuskiefer@gmail.com

| | | |
|---|---|---|
| d5433562dee4 BB *Bug 1399763 - formally verified code from HACL* for Poly1305 64bits, r=franziskus,t* | nss-tools opt | hacl |
| | nss-decision opt | D I |

```
All verification conditions discharged successfully
Verified module: Hacl.Impl.Poly1305_64 (83902 milliseconds)
All verification conditions discharged successfully
Verified module: AEAD.Poly1305_64 (12536 milliseconds)
All verification conditions discharged successfully
[31mFATA[0m[3165] unexpected EOF
[taskcluster 2017-12-01 16:29:30.734Z] === Task Finished ===
[taskcluster 2017-12-01 16:29:30.856Z] Artifact "public/image.tar" not found at "/artifacts/image.tar"
[taskcluster 2017-12-01 16:29:31.198Z] Unsuccessful task run with exit code: 1 completed in 3204.402 seconds
```

| | |
|---|---|
| Linux opt | B Bogo CRMF Certs DB EC FIPS Gtest Gtest Interop Lo (+4) SSL(+4) Test(+2) |
| Linux debug | B Bogo CRMF Certs DB EC FIPS Gtest Gtest Interop Lo (+4) SSL(+4) Test(+2) |
| Linux x64 opt | B Bogo CRMF Certs DB EC FIPS Gtest Gtest Interop Lo (+4) SSL(+4) Test(+2) |
| Linux x64 asan | B Bogo CRMF Certs DB EC FIPS Gtest Gtest Interop Lo +4) Test(+2) |
| Linux x64 debug | B Bogo CRMF Certs DB EC FIPS Gtest Gtest Interop Lo (+4) SSL(+4) Test(+2) |
| Windows 2012 opt | B CRMF Certs DB EC FIPS Gtest Lowhash Merge SDR |
| Windows 2012 debug | B CRMF Certs DB EC FIPS Gtest Lowhash Merge SDR |
| Windows 2012 x64 opt | B CRMF Certs DB EC FIPS Gtest Lowhash Merge SDR |
| Windows 2012 x64 debug | B CRMF Certs DB EC FIPS Gtest Lowhash Merge SDR |
| aarch64 debug | B CRMF Certs DB EC FIPS Gtest Gtest Lowhash Merge +2) |
| aarch64 opt | B CRMF Certs DB EC FIPS Gtest Gtest Lowhash Merge +2) |
| nss-tools opt | clang-format-3.9 scan-build-4.0 |
| Linux fuzz | B CertDN Gtest QuickDER MPI(+11) TLS(+9) |
| Linux x64 fuzz | B CertDN Gtest QuickDER MPI(+11) TLS(+9) |
| Windows 2012 x64 make | B CRMF Certs Chains DB EC FIPS Gtest Lowhash Merg |
| Linux make | B Bogo CRMF Certs Chains DB EC FIPS Gtest Interop Cipher(+4) SSL(+4) |
| Windows 2012 make | B CRMF Certs Chains DB EC FIPS Gtest Lowhash Merg |
| Linux x64 make | B Bogo CRMF Certs Chains DB EC FIPS Gtest Interop Cipher(+4) SSL(+4) |
| nss-decision opt | D |

# Supporting multiple platforms

## Large number of supported platforms

- CI does not support all platforms

Bug 1396301

**verified/kremlib.h:204:23: error: implicit declaration of function 'le64t declaration]**

**RESOLVED FIXED**

- Trusted code base is a problem

Bug 1419009

**Sigsegv at Hacl_EC_crypto_scalarmult on Solaris**

**RESOLVED INVALID**

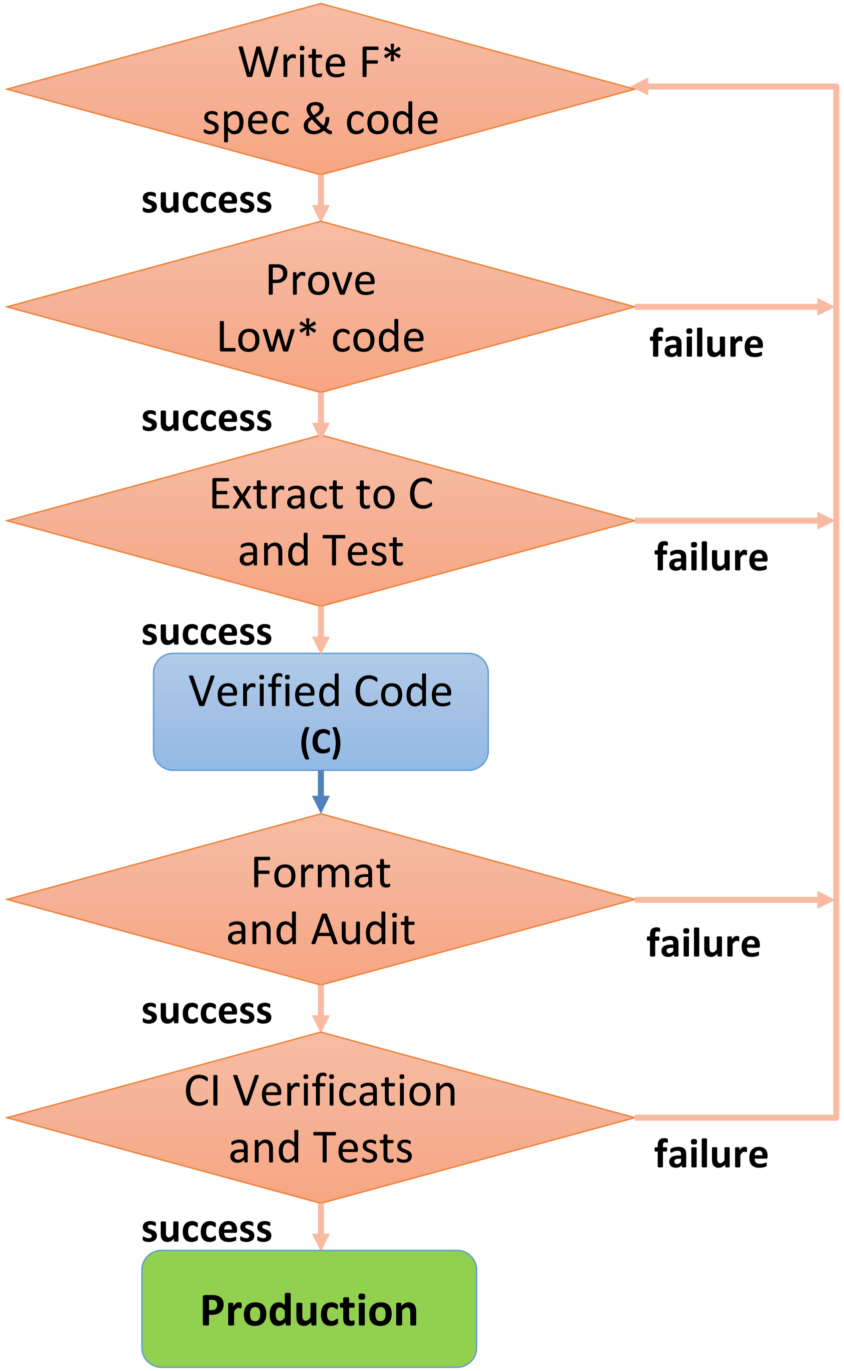
- Some bugs can be introduced by contributors

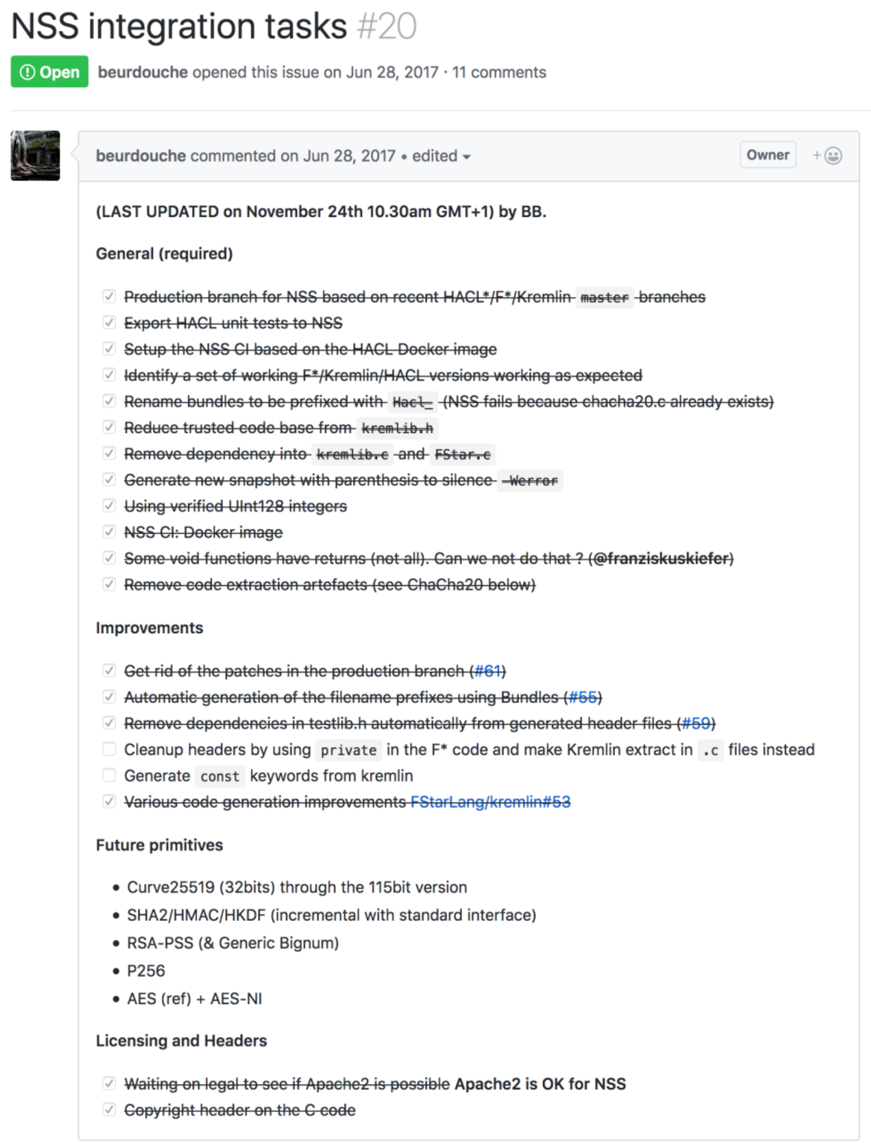
Bug 1405268

**shlibsign fails on Solaris due missing htole64 symbol**

**RESOLVED DUPLICATE** of bug 1420060

# A common workflow



NSS integration tasks #20

Open beurdouche opened this issue on Jun 28, 2017 · 11 comments

beurdouche commented on Jun 28, 2017 • edited

Owner

(LAST UPDATED on November 24th 10.30am GMT+1) by BB.

**General (required)**

- [x] ~~Production branch for NSS based on recent HACL*/F*/Kremlin `master` branches~~
- [x] ~~Export HACL unit tests to NSS~~
- [x] ~~Setup the NSS CI based on the HACL Docker image~~
- [x] ~~Identify a set of working F*/Kremlin/HACL versions working as expected~~
- [x] ~~Rename bundles to be prefixed with `Hacl_` (NSS fails because chacha20.c already exists)~~
- [x] ~~Reduce trusted code base from `kremlib.h`~~
- [x] ~~Remove dependency into `kremlib.c` and `FStar.c`~~
- [x] ~~Generate new snapshot with parenthesis to silence `-Werror`~~
- [x] ~~Using verified UInt128 integers~~
- [x] ~~NSS CI: Docker image~~
- [x] ~~Some void functions have returns (not all). Can we not do that ? (@franziskuskiefer)~~
- [x] ~~Remove code extraction artefacts (see ChaCha20 below)~~

**Improvements**

- [x] ~~Get rid of the patches in the production branch (#61)~~
- [x] ~~Automatic generation of the filename prefixes using Bundles (#55)~~
- [x] ~~Remove dependencies in testlib.h automatically from generated header files (#59)~~
- [ ] Cleanup headers by using `private` in the F* code and make Kremlin extract in `.c` files instead
- [ ] Generate `const` keywords from kremlin
- [x] ~~Various code generation improvements FStarLang/kremlin#53~~

**Future primitives**

- Curve25519 (32bits) through the 115bit version
- SHA2/HMAC/HKDF (incremental with standard interface)
- RSA-PSS (& Generic Bignum)
- P256
- AES (ref) + AES-NI

**Licensing and Headers**

- [x] ~~Waiting on legal to see if Apache2 is possible~~ Apache2 is OK for NSS
- [x] ~~Copyright header on the C code~~

# What’s next ?

## The future of NSS

- Removing more obsolete code
- Mixing-in other formal methods
- Integrate formally verified assembly
- Verifying parsers and protocols

moz://a

## The future of HACL*

- Implement new primitives
- Reduce proof effort and verification time
- Reduce trust in our tools (verify KreMLin, F*…)
- Support more platforms (WASM, RIOT…)

# Use it ! Test it ! Break it !

(NSS crypto is eligible to Mozilla's bug bounty program)

# Get in touch !

@beurdouche
benjamin.beurdouche@inria.fr