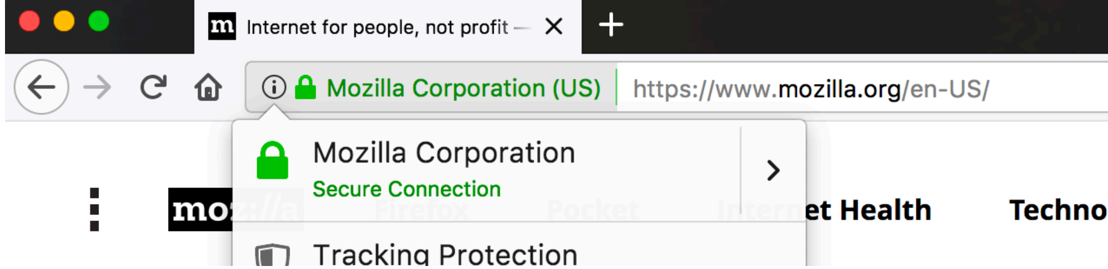
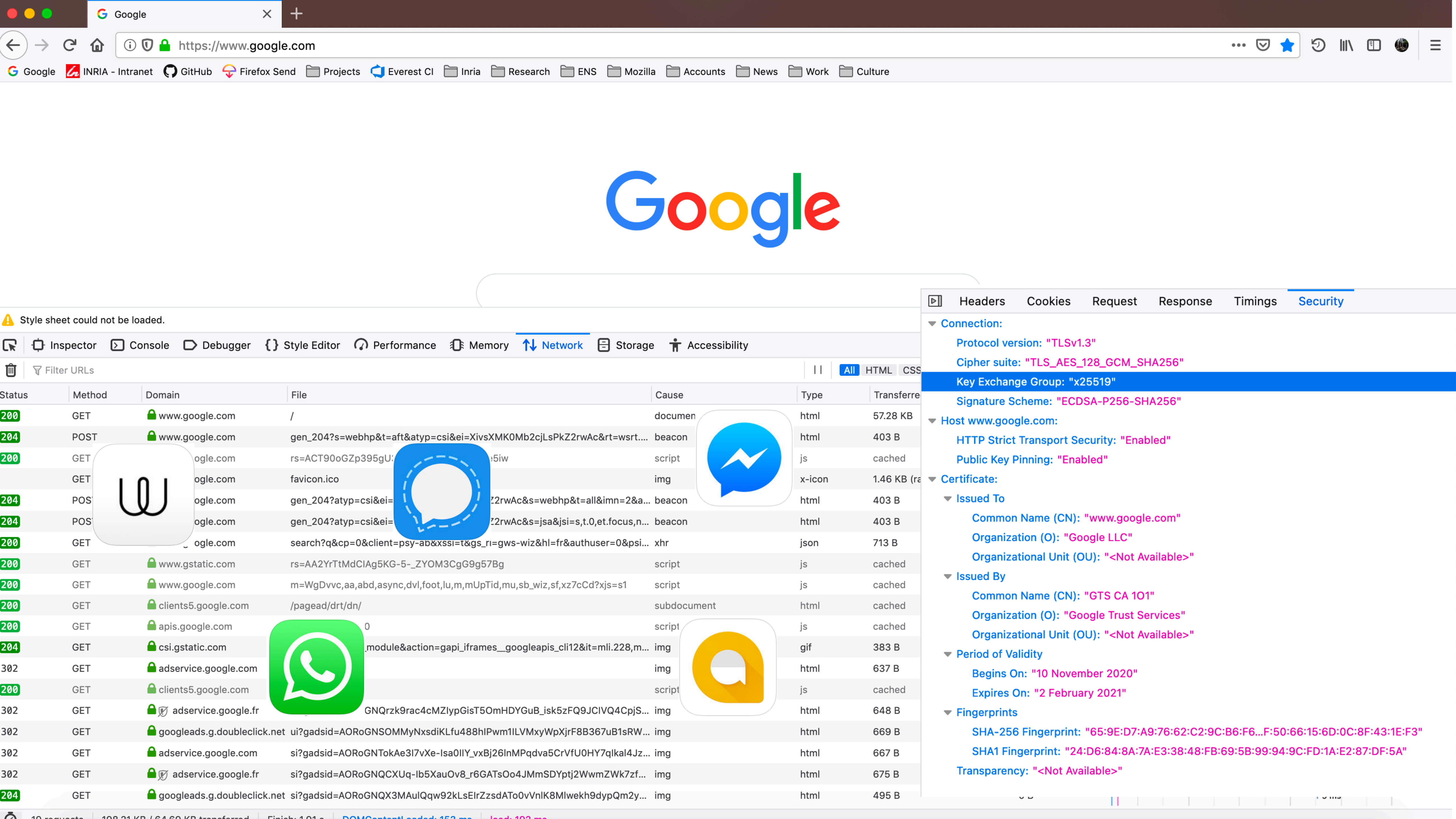


Formal verification for security software in F*

Application to cryptographic protocols and primitives.

Benjamin Beurdouche

PhD Defense @ Inria Paris - 2020/12/18


Google
https://www.google.com
Google
INRIA - Intranet
GitHub
Firefox Send
Projects
Everest CI
Inria
Research
ENS
Mozilla
Accounts
News
Work
Culture
Google
Style sheet could not be loaded.
Inspector
Console
Debugger
Style Editor
Performance
Memory
Network
Storage
Accessibility
Filter URLs
All
HTML
CSS
Status
Method
Domain
File
Cause
Type
Transferred
200 GET www.google.com / document html 57.28 KB
204 POST www.google.com gen_204?s=webhp&t=aft&atyp=csi&ei=XivsXMK0Mb2cjLsPkZ2rwAc&rt=wsrt.... beacon html 403 B
200 GET www.gstatic.com rs=AA2YrTtMdClAg5KG-5-_ZYOM3CgG9g57Bg script js cached
200 GET www.google.com m=WgDvvc,aa,abd,async,dvl,foot,lu,m,mUpTid,mu,sb_wiz,sf,xz7cCd?xjs=s1 script js cached
200 GET clients5.google.com /pagead/drt/dn/ subdocument html cached
302 GET adservice.google.com img html 637 B
302 GET googleads.g.doubleclick.net ui?gadsid=AORoGNSOMMyNxsdiKLfu488hIPwm1ILVMxyWpXjrF8B367uB1sRW... img html 669 B
302 GET adservice.google.com si?gadsid=AORoGNTokAe3l7vXe-Isa0IlY_vxBj26lnMPqdva5CrVfU0HY7qlkal4Jz... img html 667 B
302 GET adservice.google.fr si?gadsid=AORoGNQCXUq-Ib5XauOv8_r6GATsOo4JMmSDYptj2WwmZWk7zf... img html 675 B
204 GET googleads.g.doubleclick.net si?gadsid=AORoGNQX3MAulQqw92kLsElrZzsdATo0vVnlK8Mlwekh9dypQm2y... img html 495 B
Headers
Cookies
Request
Response
Timings
Security
Connection:
Protocol version: "TLSv1.3"
Cipher suite: "TLS_AES_128_GCM_SHA256"
Key Exchange Group: "x25519"
Signature Scheme: "ECDSA-P256-SHA256"
Host www.google.com:
HTTP Strict Transport Security: "Enabled"
Public Key Pinning: "Enabled"
Certificate:
Issued To
Common Name (CN): "www.google.com"
Organization (O): "Google LLC"
Organizational Unit (OU): "<Not Available>"
Issued By
Common Name (CN): "GTS CA 1O1"
Organization (O): "Google Trust Services"
Organizational Unit (OU): "<Not Available>"
Period of Validity
Begins On: "10 November 2020"
Expires On: "2 February 2021"
Fingerprints
SHA-256 Fingerprint: "65:9E:D7:A9:76:62:C2:9C:B6:F6...F:50:66:15:6D:0C:8F:43:1E:F3"
SHA1 Fingerprint: "24:D6:84:8A:7A:E3:38:48:FB:69:5B:99:94:9C:FD:1A:E2:87:DF:5A"
Transparency: "<Not Available>"

Let's have a look at the security of TLS implementations!

A messy state of the union: Taming the composite state machines of TLS
B Beurdouche, K Bhargavan, A Delignat-Lavaud, C Fournet, M Kohlweiss, ...
2015 IEEE Symposium on Security and Privacy, 535-552

FlexTLS: A Tool for Testing TLS Implementations
B Beurdouche, A Delignat-Lavaud, N Kobeissi, A Pironti, K Bhargavan
9th USENIX Workshop on Offensive Technologies (WOOT 15)

TLS Standards & Implementations

Internet Standard

1994	Netscape's Secure Sockets Layer
1995	SSL3
1999	TLS 1.0 (≈SSL3)
2006	TLS 1.1
2008	TLS 1.2
2018	TLS 1.3

Implementations

OpenSSL SChannel NSS SecureTransport mbedTLS JSSE GnuTLS miTLS

Large C/C++ codebase (400K LOC), many forks
Optimized cryptographic code for 50 platforms
Not the best API
Frequent security patches https://openssl.org/news/vulnerabilities.html

Breaking TLS implementations

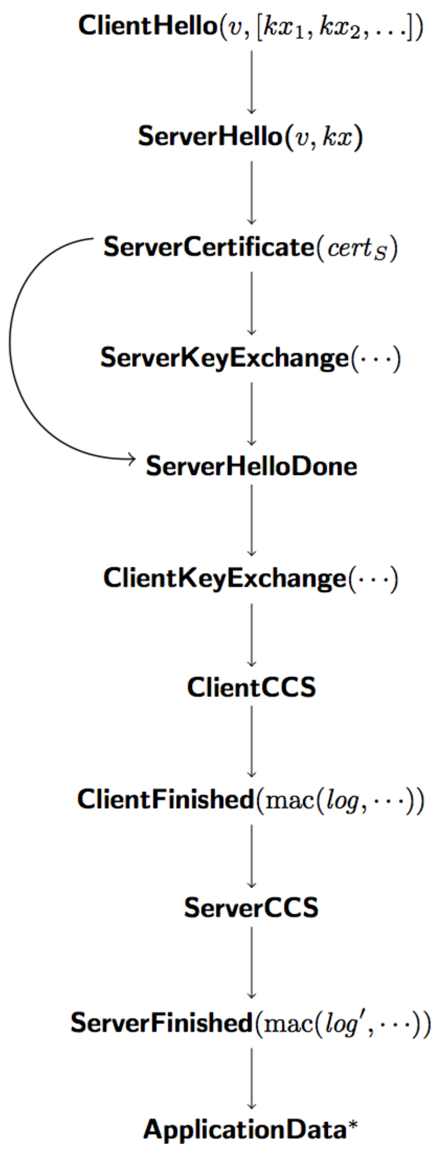
Exploiting incorrect state-machines in TLS 1.2 libraries

TLS offers many ciphersuites, optional messages, extensions… sharing the same state machine.

FlexTLS provides a way to test TLS state machine.

We systematically generated and tested deviant traces against TLS implementations (skipping, inserting, reordering valid messages)

We found many many exploitable bugs!
…including FREAK (weak crypto)…

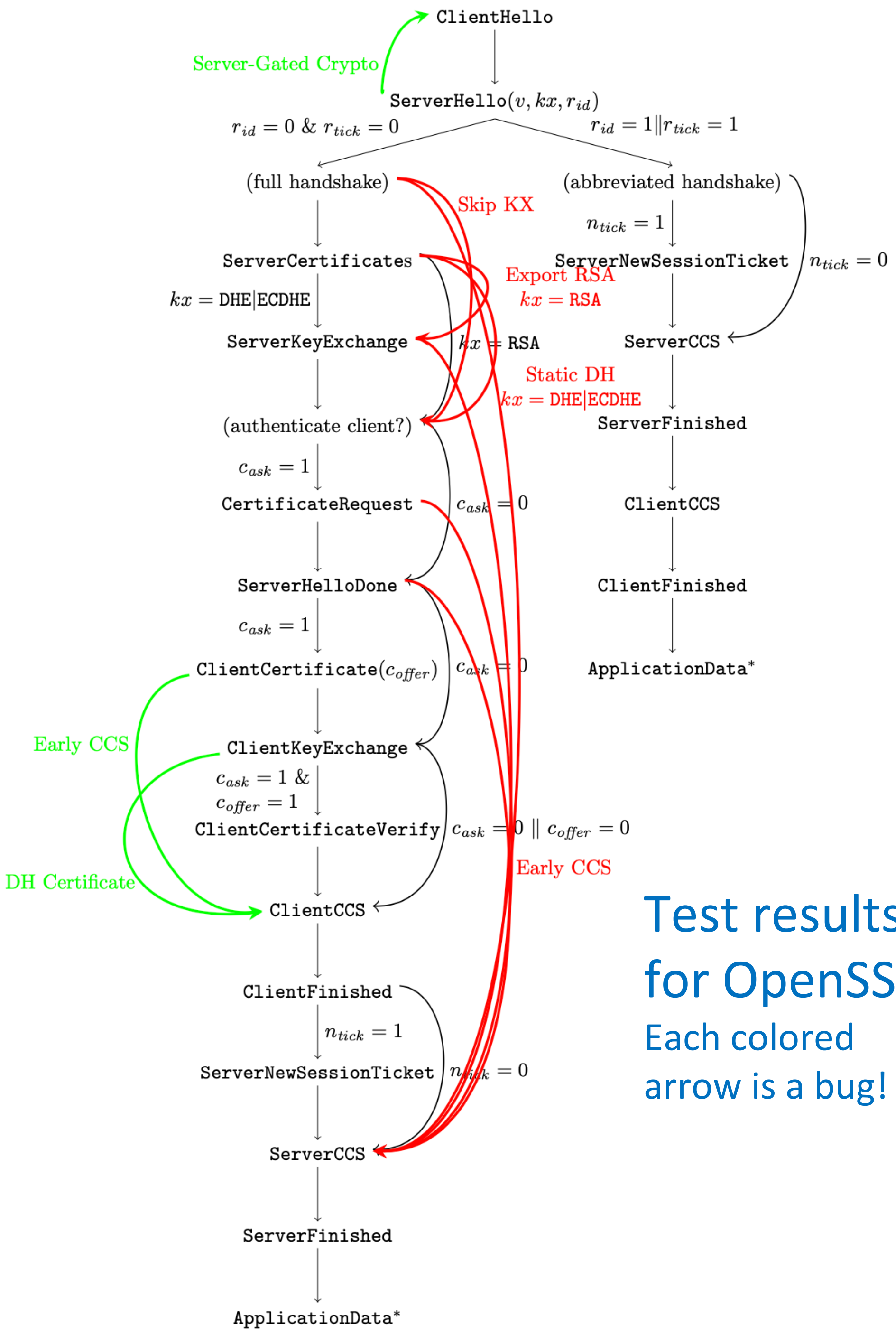


Breaking TLS implementations

Exploiting incorrect state-machines in TLS 1.2 libraries

```
static member server (listening_address:string, ?port:int) : unit =
  (* Genuine www.google.com certificate chain *)
  let g1 = new System.Security.Cryptography.X509Certificates("google.com-1.crt") in
  let g2 = new System.Security.Cryptography.X509Certificates("google.com-2.crt") in
  let g3 = new System.Security.Cryptography.X509Certificates("google.com-3.crt") in
  let chain = [g1.RawData; g2.RawData; g3.RawData] in

  let port = defaultArg port FlexConstants.defaultTCPPort in
  while true do
    (* Accept TCP connection from the client *)
    let st,cfg = FlexConnection.serverOpenTcpConnection(listening_address,
                                                        listening_address,
                                                        port) in
    (* Start RSA key exchange *)
    let st,nsc,fch = FlexClientHello.receive(st) in
    let fsh = { FlexConstants.nullFServerHello with ciphersuite =
                Some(TLSConstants.TLS_DHE_RSA_WITH_AES_128_CBC_SHA)} in
    let st,nsc,fsh = FlexServerHello.send(st,fch,nsc,fsh) in
    let st, nsc, fc = FlexCertificate.send(st, Server, chain, nsc) in
    let verify_data = FlexSecrets.makeVerifyData nsc.si (abytes [||]) Server st.hs_log in

    (* Skip key exchange messages and send Finished *)
    let st,fin = FlexFinished.send(st,verify_data=verify_data) in
    let st = FlexAppData.send(st,"HTTP/1.1 200 OK ...") in
    Tcp.close st.ns
  done
end
```

FlexTLS code for the Early Finished Attack



Test results for OpenSSL

Each colored arrow is a bug!

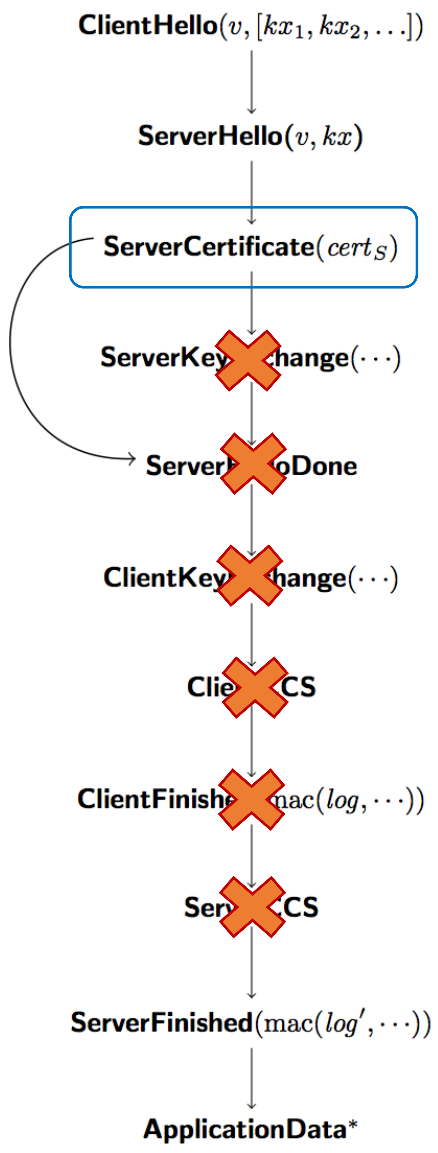
Breaking TLS implementations

Exploiting incorrect state-machines in TLS 1.2 libraries

An attack against TLS Java Library (open for 10 years)

A network attacker impersonates the Server and skips 6 messages (including the Server's signature).

JSSE's client assumes the key exchange is finished, uses uninitialized 0x000000… as session key!



Thesis goals

Secure implementations of **Cryptographic primitives**

Scale to **Cryptographic protocols**

Improve process for **Designing new Cryptographic protocols**

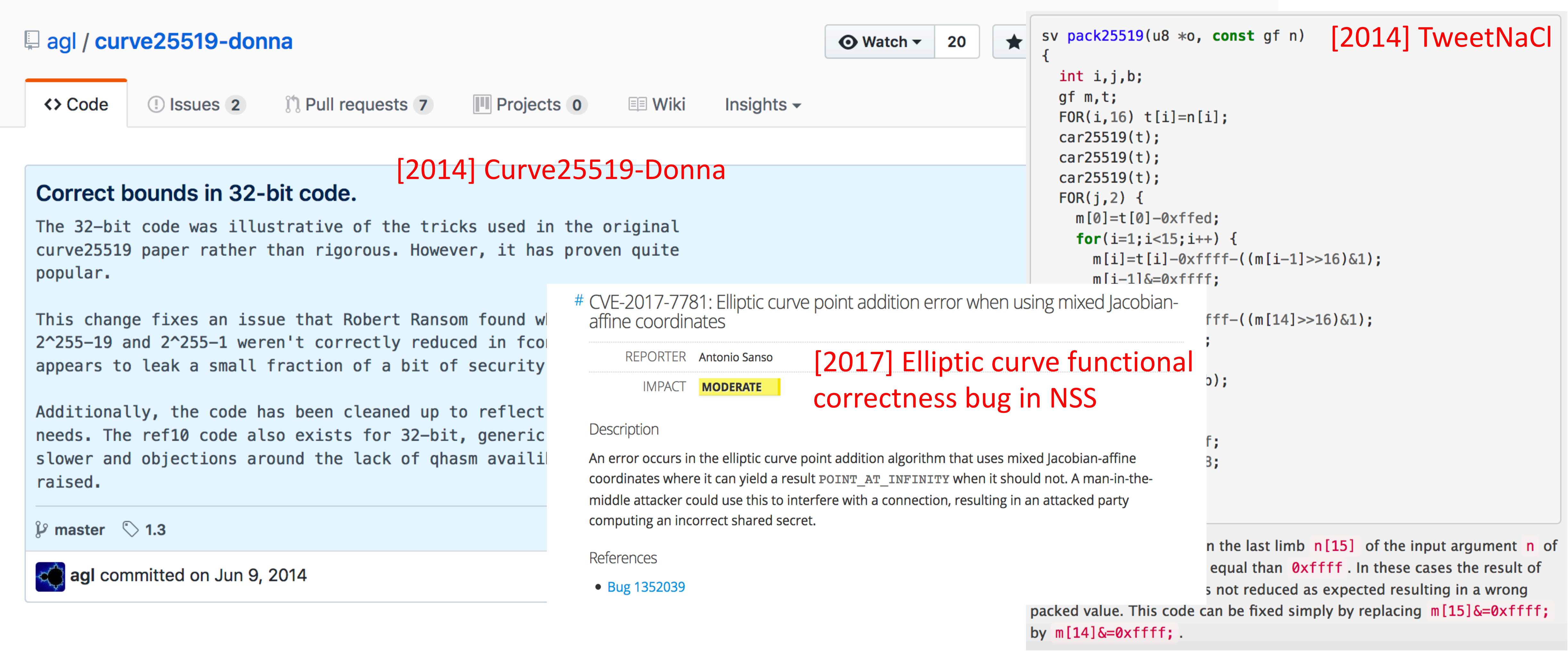
HACL*: a library of formally verified cryptographic primitives

HACL*: A Verified Modern Cryptographic Library
JK Zinzindohoué, K Bhargavan, J Protzenko, B Beurdouche
ACM CCS 2017

Evercrypt: A fast, verified, cross-platform cryptographic provider
J Protzenko, B Parno, A Fromherz, C Hawblitzel, M Polubelova, ...
2020 IEEE Symposium on Security and Privacy (SP), 634-653

HACLxN: Verified Generic SIMD Crypto
M Polubelova, K Bhargavan, J Protzenko, B Beurdouche, A Fromherz, ...

Implementing is hard for everyone



[2014] TweetNaCl

[2014] Curve25519-Donna

[2017] Elliptic curve functional correctness bug in NSS

Even for very skilled programmers or cryptographers !

What are the properties interesting for us?

Memory Safety

(buffer overflow, out of bounds r/w…)

Secret Independence

(execution time leaks secrets)

Functional correctness

(incorrect code logic)

Formal methods inbound!

Recent academic developments for Cryptography

Verification of a Cryptographic Primitive: SHA-256

ANDREW W. APPEL, Princeton University

Verifying Curve25519 Software

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

[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

Formal verification: what and how ?

Memory Safety

Portability

Side-channel resistance (timing)

Performance

Functional correctness

Compiler Trust

Proof Effort

Code generation or Verification of existing code?

Work on ASM, C or High-Level Languages?

Readability

Reproducibility

Maintenance

Verification time

CCS 2017 - https://eprint.iacr.org/2017/536 S&P 2020 - https://eprint.iacr.org/2019/757 CCS 2020 - https://eprint.iacr.org/2020/572

HACL*: A Verified Modern Cryptographic Library

Jean Karim Zinzindohoué
INRIA

Karthikeyan Bhargavan
INRIA

Jonathan Protzenko
Microsoft Research

Benjamin Beurdouche
INRIA

EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider

Jonathan Protzenko*, Bryan Parno‡, Aymeric Fromherz‡, Chris Hawblitzel*, Marina Polubelova†, Karthikeyan Bhargavan†
Benjamin Beurdouche†, Joonwon Choi*§, Antoine Delignat-Lavaud*, Cédric Fournet*, Natalia Kulatova†,
Tahina Ramananandro*, Aseem Rastogi*, Nikhil Swamy*, Christoph M. Wintersteiger*, Santiago Zanella-Beguelin*
*Microsoft Research ‡Carnegie Mellon University †Inria §MIT

HACLxN: Verified Generic SIMD Crypto (for all your favorite platforms)

Marina Polubelova
Inria Paris

Karthikeyan Bhargavan
Inria Paris

Jonathan Protzenko
Microsoft Research

Benjamin Beurdouche
Inria Paris and Mozilla

Aymeric Fromherz
Carnegie Mellon University

Natalia Kulatova
Inria Paris

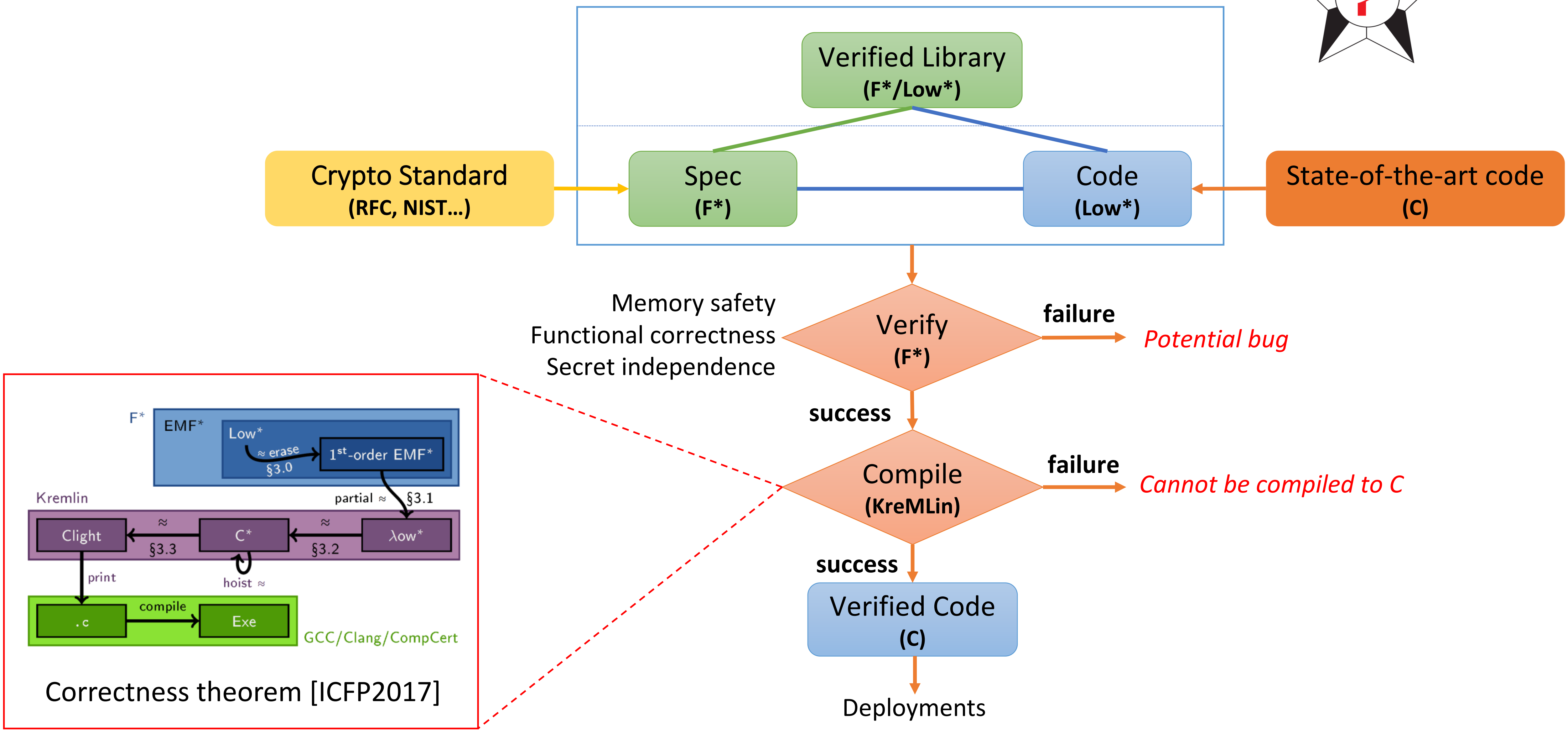
Santiago Zanella-Béguelin
Microsoft Research

ABSTRACT

Single Instruction Multiple Data (SIMD) vectorization. Since most

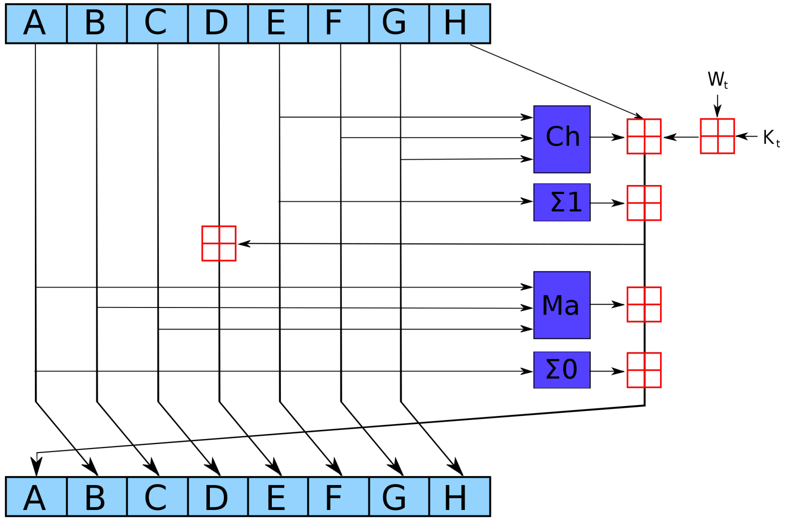
HACL* verification workflow

[POPL 2016]



Writing code for the SHA2 shuffle function

```
let shuffle_core a block hash ws t =
  let a0 = hash.(0ul) in
  let b0 = hash.(1ul) in
  let c0 = hash.(2ul) in
  let d0 = hash.(3ul) in
  let e0 = hash.(4ul) in
  let f0 = hash.(5ul) in
  let g0 = hash.(6ul) in
  let h0 = hash.(7ul) in

  let w = ws.(t) in
  let t1 = h0 +. (Spec._Sigma1 a e0) +. (Spec._Ch a e0 f0 g0) +. (k0 a).(t) +. w in
  let t2 = (Spec._Sigma0 a a0) +. (Spec._Maj a a0 b0 c0) in

  hash.(0ul) ← t1 +. t2;
  hash.(1ul) ← a0;
  hash.(2ul) ← b0;
  hash.(3ul) ← c0;
  hash.(4ul) ← d0 +. t1;
  hash.(5ul) ← e0;
  hash.(6ul) ← f0;
  hash.(7ul) ← g0;
```



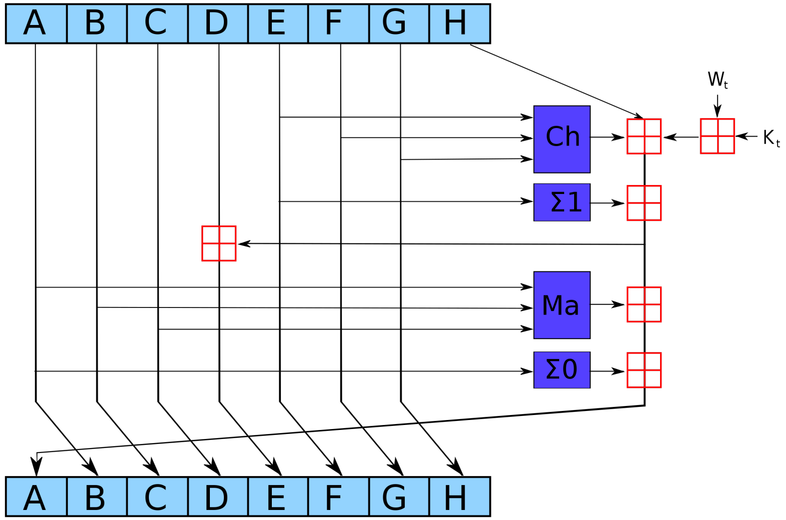
One round of SHA-2 compression
internal state of 8 (32/64-bit) words

This is a stateful function performing in-place modifications of the hash array.

[Picture from Wikipedia user:Kockmeyer]

Proving Memory safety in F*

```
val shuffle_core
  (a: sha2_alg)
  (block: block_b a)
  (hash: words_state a)
  (ws: ws_w a)
  (t: U32.t { U32.v t < Spec.size_k_w a }):
  Stack unit
    (requires (λ h →
      live h block ∧ live h hash ∧ live h ws ∧
      disjoint block hash ∧ disjoint block ws ∧ disjoint hash ws))
    (ensures (λ h0 _h1 →
      modifies hash h0 h1))
```



One round of SHA-2 compression
internal state of 8 (32/64-bit) words

Live and Disjoint predicates are required to hold on inputs

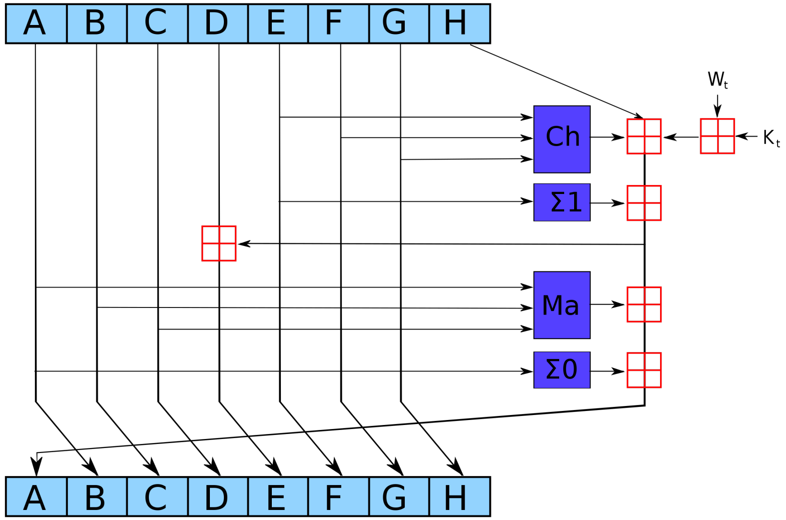
- Live: “the pointer is not null”
- Disjoint: “the objects don’t occupy the same space in memory”

Modifies ensures that hash is the only array modified by this function.

[Picture from Wikipedia user:Kockmeyer]

Proving Functional correctness in F*

```
val shuffle_core
  (a: sha2_alg)
  (block: block_b a)
  (hash: words_state a)
  (ws: ws_w a)
  (t: U32.t { U32.v t < Spec.size_k_w a }):
  Stack unit
    (requires (λ h →
      let b = block_words_be a h block in
      h.[ws] == S.init (Spec.size_k_w a) (Spec.ws a b)))
    (ensures (λ h0 _h1 →
      let b = block_words_be a h0 block in
      h1.[hash] == Spec.shuffle_core a b h0.[hash] (U32.v t)))
```



One round of SHA-2 compression
internal state of 8 (32/64-bit) words

Some preconditions are required for the values of the ws array.

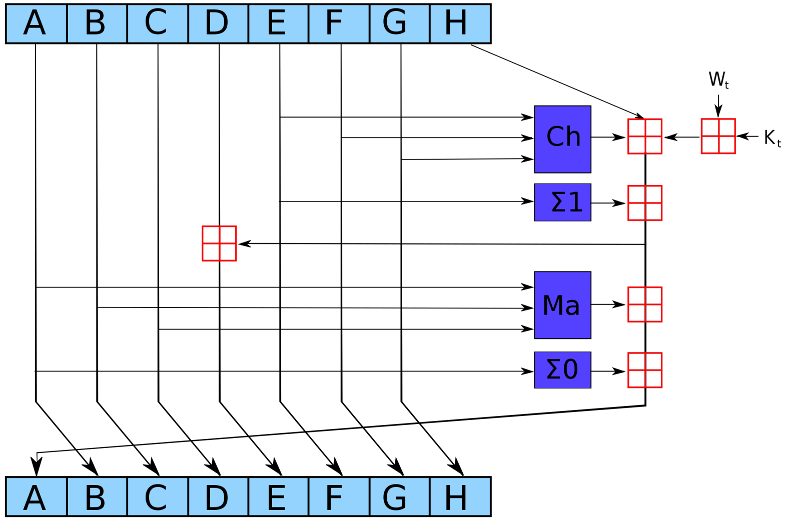
The postcondition ensures that the output of the efficient stateful function presented is equal to applying the Specification on the same inputs.

[Picture from Wikipedia user:Kockmeyer]

Overall function signature

```
val shuffle_core
  (a: sha2_alg)
  (block: block_b a)
  (hash: words_state a)
  (ws: ws_w a)
  (t: U32.t { U32.v t < Spec.size_k_w a }):
  Stack unit
    (requires (λ h →
      let b = block_words_be a h block in
      live h block ∧ live h hash ∧ live h ws ∧
      disjoint block hash ∧ disjoint block ws ∧ disjoint hash ws ∧
      h.[ws] == S.init (Spec.size_k_w a) (Spec.ws a b)))
    (ensures (λ h0 _h1 →
      let b = block_words_be a h0 block in
      modifies hash h0 h1 ∧
      h1.[hash] == Spec.shuffle_core a b h0.[hash] (U32.v t)))
```

Everything together…



One round of SHA-2 compression
internal state of 8 (32/64-bit) words

[Picture from Wikipedia user:Kockmeyer]

Proving Secret Independence for Machine Integers

```
inline_for_extraction
let int_t (t:inttype) (l:secrecy_level) =
  match l with
  | PUB → pub_int_t t
  | SEC → sec_int_t t

let uint_t (t:inttype{unsigned t}) (l:secrecy_level) = int_t t l

type uint1 = uint_t U1 SEC
type uint8 = uint_t U8 SEC
type uint16 = uint_t U16 SEC
type uint32 = uint_t U32 SEC
type uint64 = uint_t U64 SEC

val add: #t:inttype → #l:secrecy_level
  → a:int_t t l
  → b:int_t t l{range (v a + v b) t}
  → int_t t l

val div: #t:inttype{¬(U128? t) ∧ ¬(S128? t)}
  → a:int_t t PUB
  → b:int_t t PUB{v b ≠ 0 ∧ (unsigned t ∨ range FStar.Int.(v a / v b) t)}
  → int_t t PUB
```

Secret Integers cannot:

- be compared (using =)
- be used for array indexing
- perform non-CT operations (may depend on platform)

What are the properties interesting for us?

Memory Safety

(buffer overflow, out of bounds r/w…)

Secret Independence

(execution time leaks secrets)

Functional correctness

(incorrect code logic)

HACL* - High Assurance Crypto Library

Formal verification can scale up !

Functionalities

- Hash functions
- Message authentication codes
- Encryption schemes
- Elliptic curve algorithms
- Signature schemes

- High Level APIs

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

Is this ready for production?

Improving code quality for Production

```
inline static void
Hacl_Bignum_AddAndMultiply_add_and_multiply(uint64_t *acc, uint64_
{
  for (uint32_t i = (uint32_t )0; i < (uint32_t )3; i = i + (uint3
  {
-   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);
}
```

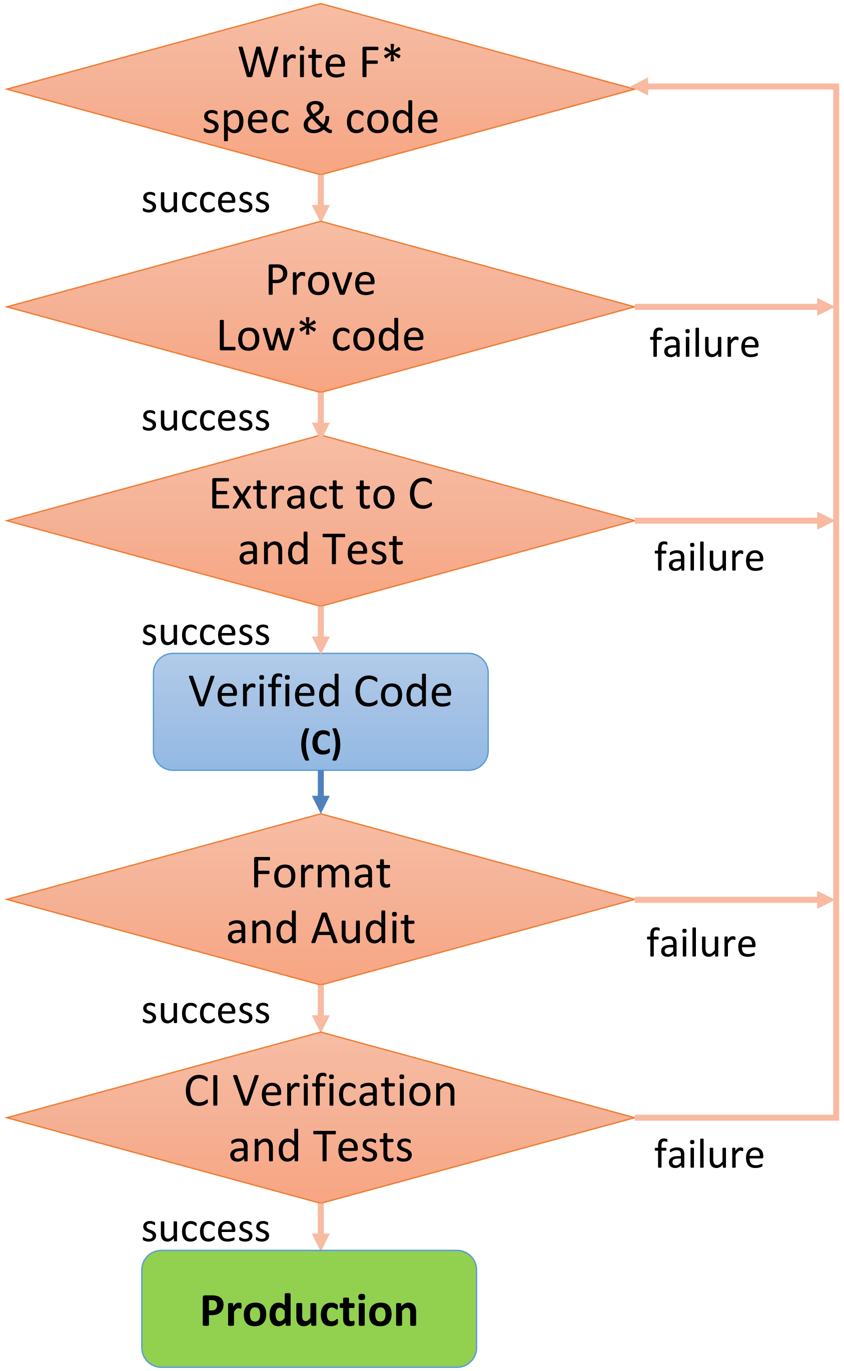
```
inline static void
Hacl_Bignum_AddAndMultiply_add_and_multiply(uint64_t *acc, uint64_t
{
  for (uint32_t i = (uint32_t )0; i < (uint32_t )3; i = i + (uint32
  {
+   uint64_t xi = acc[i];
+   uint64_t yi = block[i];
+   acc[i] = xi + yi;

  }
  Hacl_Bignum_Fmul_fmul(acc, acc, r);
}
```

Better variable naming

Removing intermediate variables

Production workflow

Firefox



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

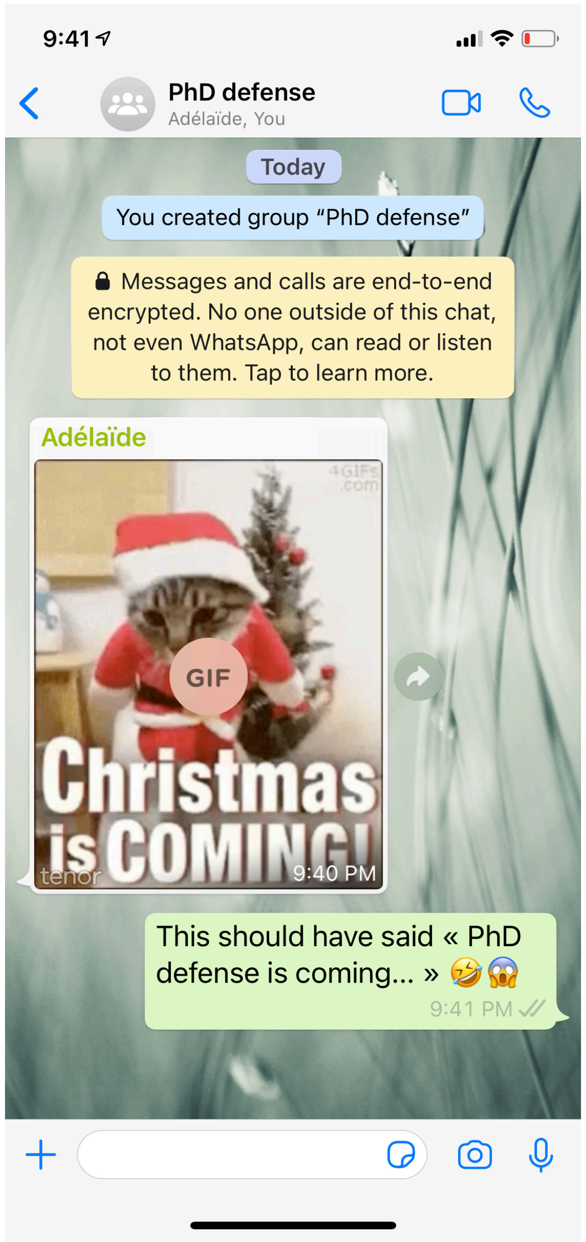
- [x] ~~Waiting on legal to see if Apache2 is possible~~ **Apache2 is OK for NSS**
- [x] ~~Copyright header on the C code~~

Formally verified protocol software

Formally Verified Cryptographic Web Applications in WebAssembly
J Protzenko, B Beurdouche, D Merigoux, K Bhargavan
2019 IEEE Symposium on Security and Privacy (SP), 1002-1020

Signal protocol

WhatsApp



Signal Protocol is a pairwise secure channel used in many messaging applications.

Provides strong security guarantees in the 2-party setting, including:

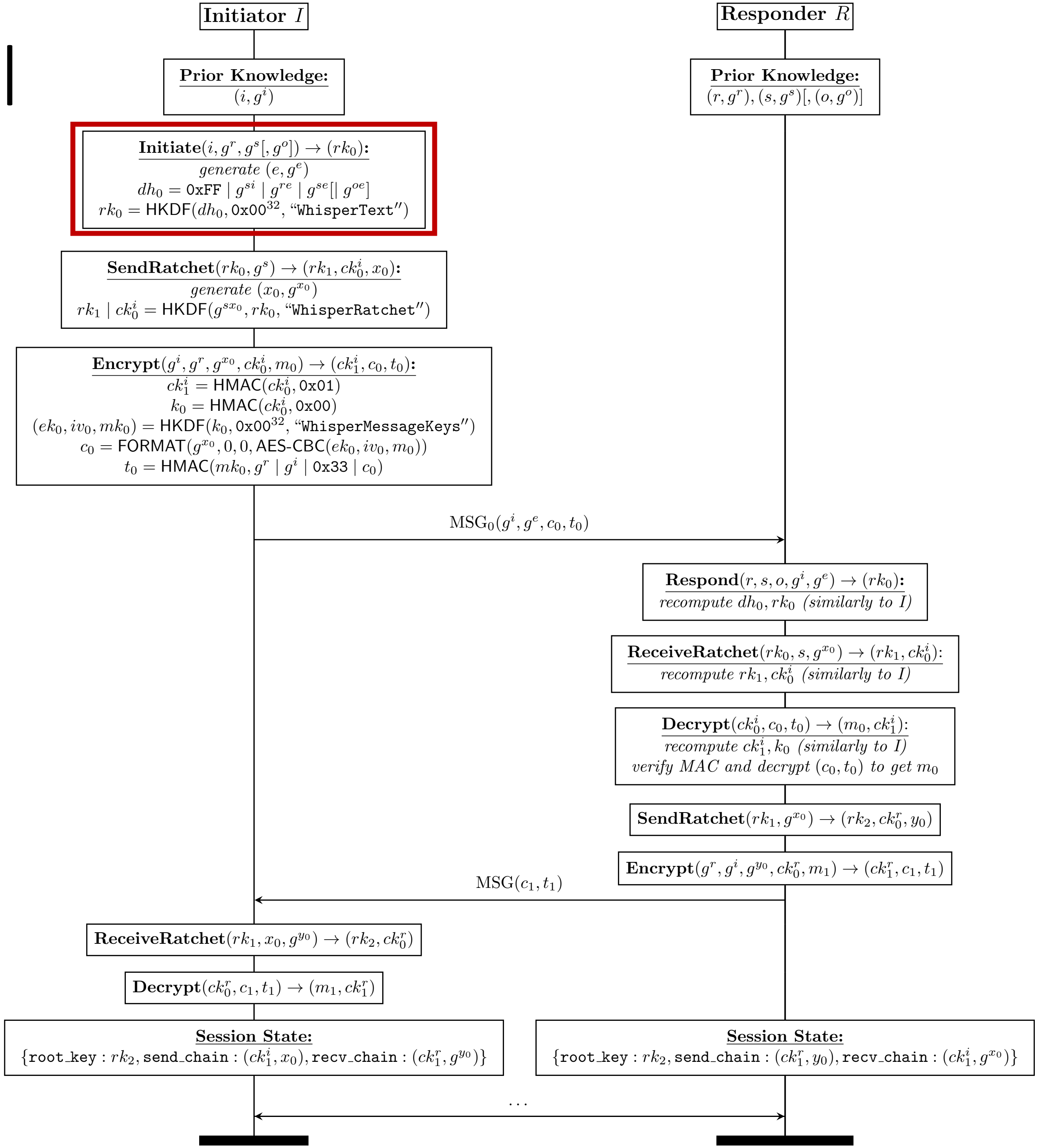
- Forward Secrecy (FS)
- Post Compromise Security (PCS)

Significant scientific literature and analysis

- Both symbolic and computational models
- Both mechanized and manual proofs

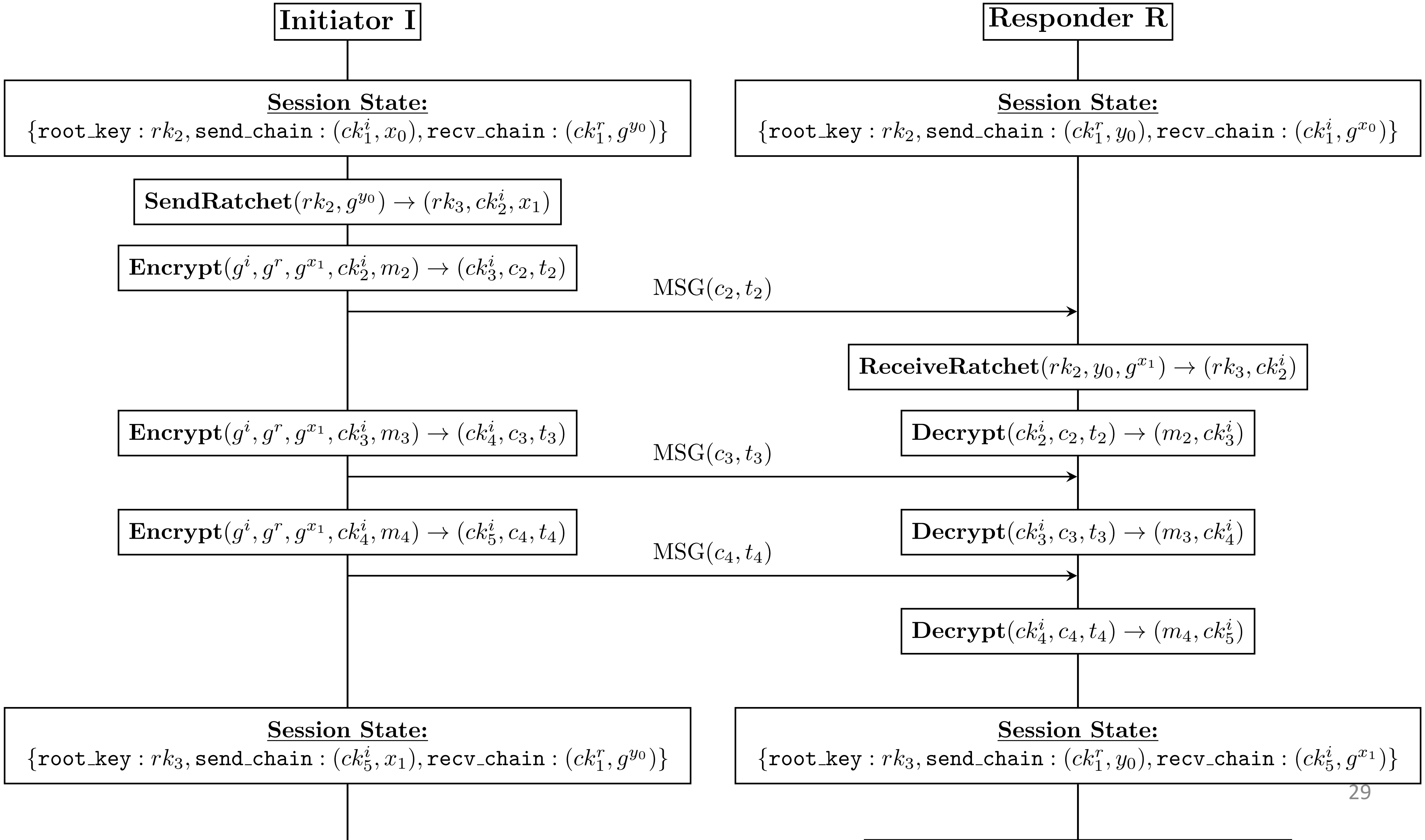
Signal Protocol

X3DH



Signal Protocol

Double Ratchet



Specifying Signal in F*

X3DH (initiate)

```
val initiate:
  our_identity_priv_key: privkey →(* i *)
  our_onetime_priv_key: privkey →(* e *)
  their_identity_pub_key: pubkey →(* g^r *)
  their_signed_pub_key: pubkey →(* g^s *)
  their_onetime_pub_key: option pubkey →(* g^o, optional *)
  Tot (lbytes 32) (* output: rk_0 *)

let initiate iidsk iesk ridpk rspk orepk =
  let dh1 = dh iidsk rspk in
  let dh2 = dh iesk ridpk in
  let dh3 = dh iesk rspk in
  let ss =
    match orepk with
    | None →ff @| dh1 @| dh2 @| dh3
    | Some repk →
          let dh4 = dh iesk repk in
          ff @| dh1 @| dh2 @| dh3 @| dh4 in
  let rk0 = hkdf1 ss zz label_WhisperText in
  rk0
```

$$\textbf{Initiate}(i, g^r, g^s[, g^o]) \rightarrow (rk_0):$$

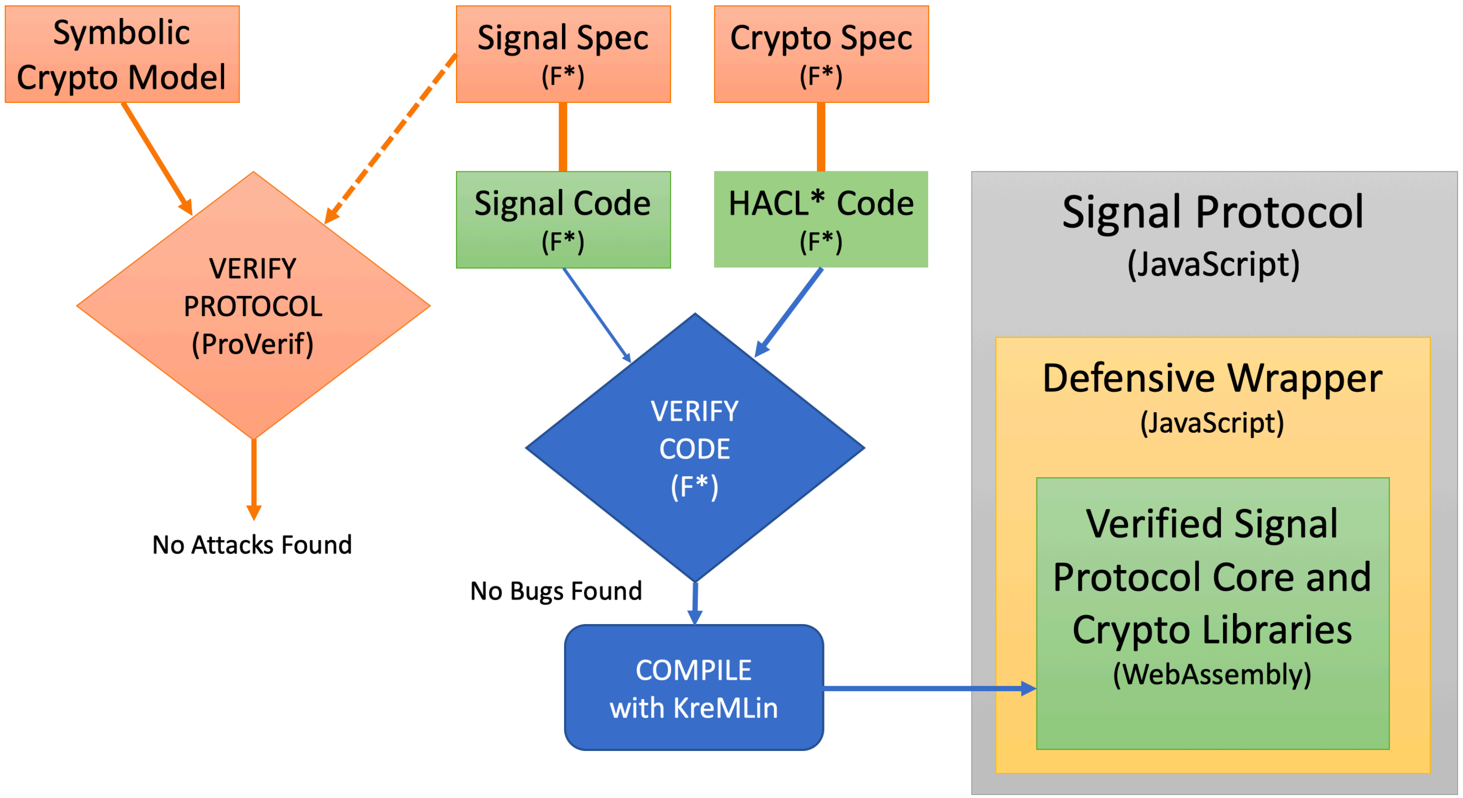
$$\textit{generate}\ (e, g^e)$$

$$dh_0 = \texttt{0xFF} \mid g^{si} \mid g^{re} \mid g^{se}[\| g^{oe}]$$

$$rk_0 = \text{HKDF}(dh_0, \texttt{0x00}^{32}, \text{``}\texttt{WhisperText}\text{''})$$

Figure 4.9 – Functional specification of Signal's initiate function

A verified interoperable implementation of Signal



Kind of message	F*-WebAssembly	Vanilla Signal
Initiate/Respond	61.6 ms	74.7 ms
Diffie-Hellman ratchet	21.7 ms	35.4 ms
Hash ratchet	2.19 ms	3.52 ms

Figure 4.19 – Performance evaluation of LibSignal, taken from the execution of the Signal test-suite. Numbers correspond to the mean execution time of the processing for messages involving the same number of key derivations.

Designing and verifying MLS

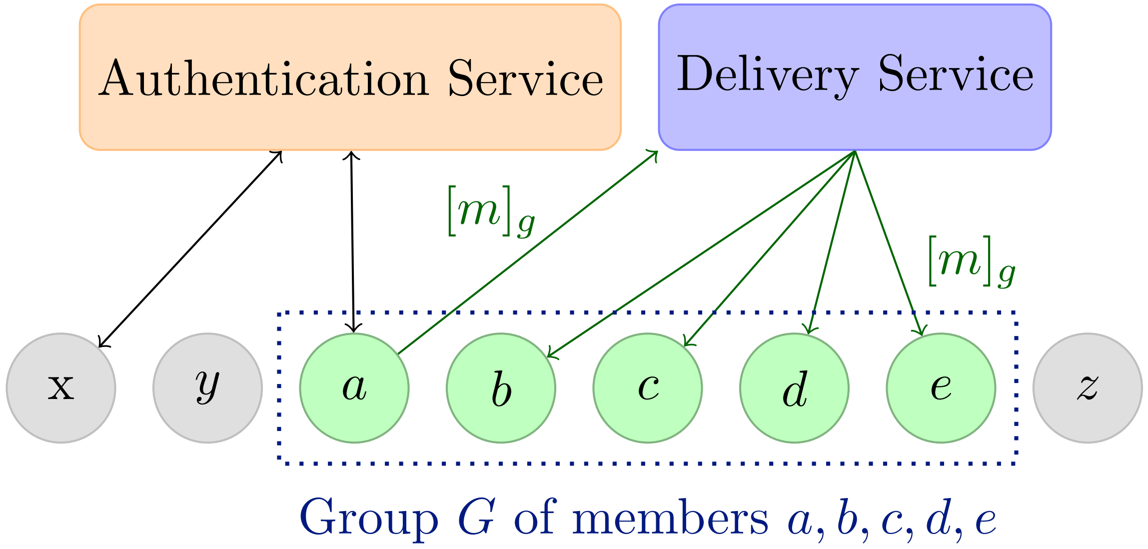
The Messaging Layer Security (MLS) Protocol
R Barnes, B Beurdouche, J Millican, E Omara, K Cohn-Gordon, R Robert
Internet Engineering Task Force

The Messaging Layer Security (MLS) Architecture
E Omara, B Beurdouche, E Rescorla, S Inguva, A Kwon, A Duric
Internet Engineering Task Force

Messaging Layer Security

A new secure group messaging protocol at the IETF

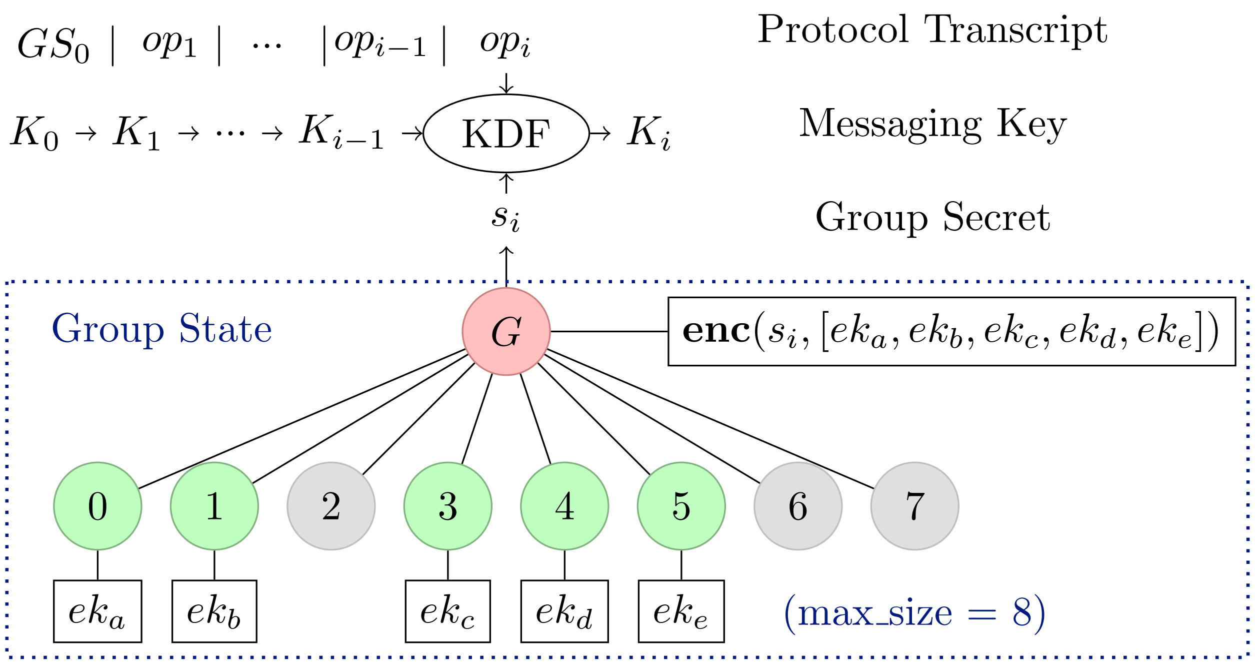
Architecture of a Secure Messaging System



The Authentication Service (AS) is often trusted (not necessarily).
The Delivery Service (DS) is untrusted.

Group Key Agreements

Chained mKEM



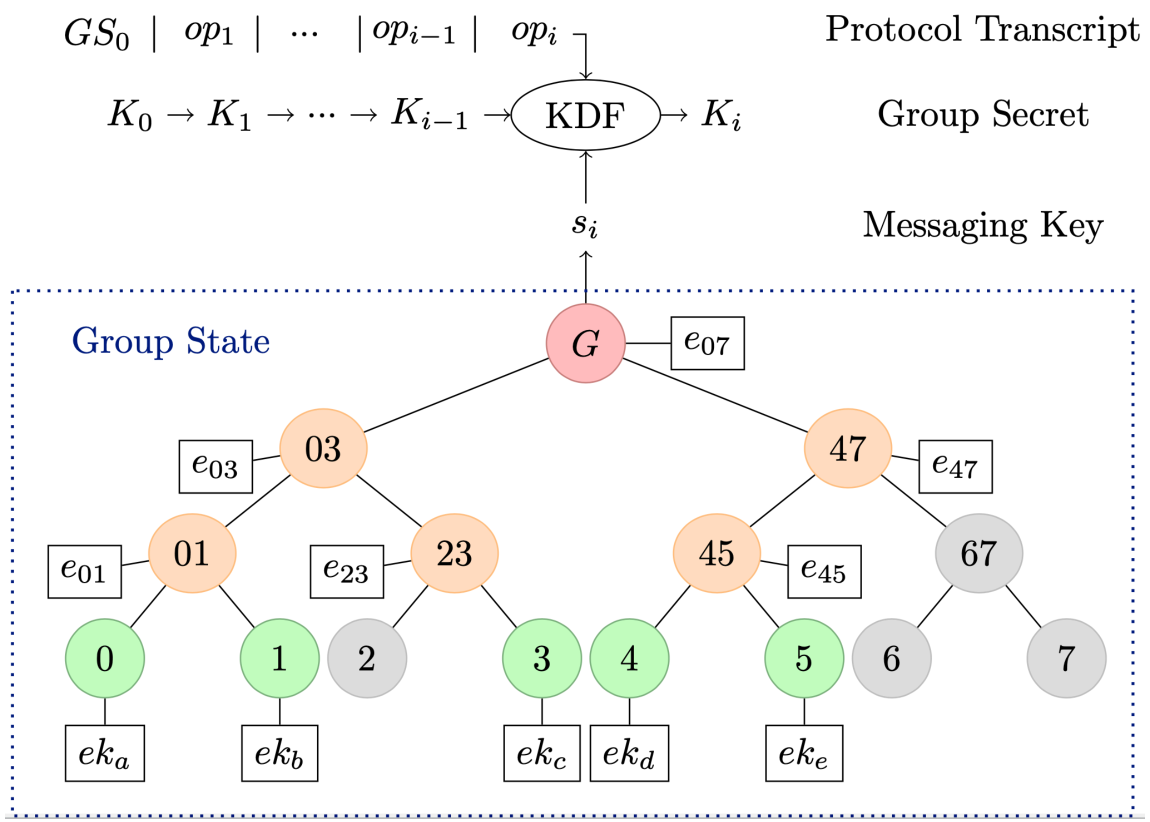
O(N) Public Key operations for the sender on Creation, Update and Remove
O(1) Public Key operations for the sender on Add
O(1) Public Key operations for the receiver

O(1) AEAD Encryption/Decryption for messages

O(N) Storage

TreeKEM: Tree-based Group Key Agreement for MLS

TreeKEM



Node Keypairs:

$dk_{01} \stackrel{\text{def}}{=} \textbf{kdf}(dk_b, \textbf{right}, ek_a)$

$ek_{01} \stackrel{\text{def}}{=} \textbf{pub}(dk_{01})$

$(dk_{23}, ek_{23}) \stackrel{\text{def}}{=} (dk_c, ek_c)$

$dk_{45} \stackrel{\text{def}}{=} \textbf{kdf}(dk_d, \textbf{left}, ek_e)$

$ek_{45} \stackrel{\text{def}}{=} \textbf{pub}(dk_{45})$

$dk_{03} \stackrel{\text{def}}{=} \textbf{kdf}(dk_{01}, \textbf{left}, ek_{23})$

$ek_{03} \stackrel{\text{def}}{=} \textbf{pub}(dk_{03})$

$(dk_{47}, ek_{47}) \stackrel{\text{def}}{=} (dk_{45}, ek_{45})$

$dk_{07} \stackrel{\text{def}}{=} \textbf{kdf}(dk_{47}, \textbf{right}, ek_{03})$

Group Secret: $s_{07} \stackrel{\text{def}}{=} dk_{07}$

Node Ciphertexts:

$e_{01} \stackrel{\text{def}}{=} \textbf{left}, ek_{01}, \textbf{enc}(dk_{01}, ek_a)$

$e_{23} \stackrel{\text{def}}{=} \textbf{right}, ek_{23}, \_$

$e_{45} \stackrel{\text{def}}{=} \textbf{left}, ek_{45}, \textbf{enc}(dk_{45}, ek_e)$

$e_{03} \stackrel{\text{def}}{=} \textbf{left}, ek_{03}, \textbf{enc}(dk_{03}, ek_{23})$

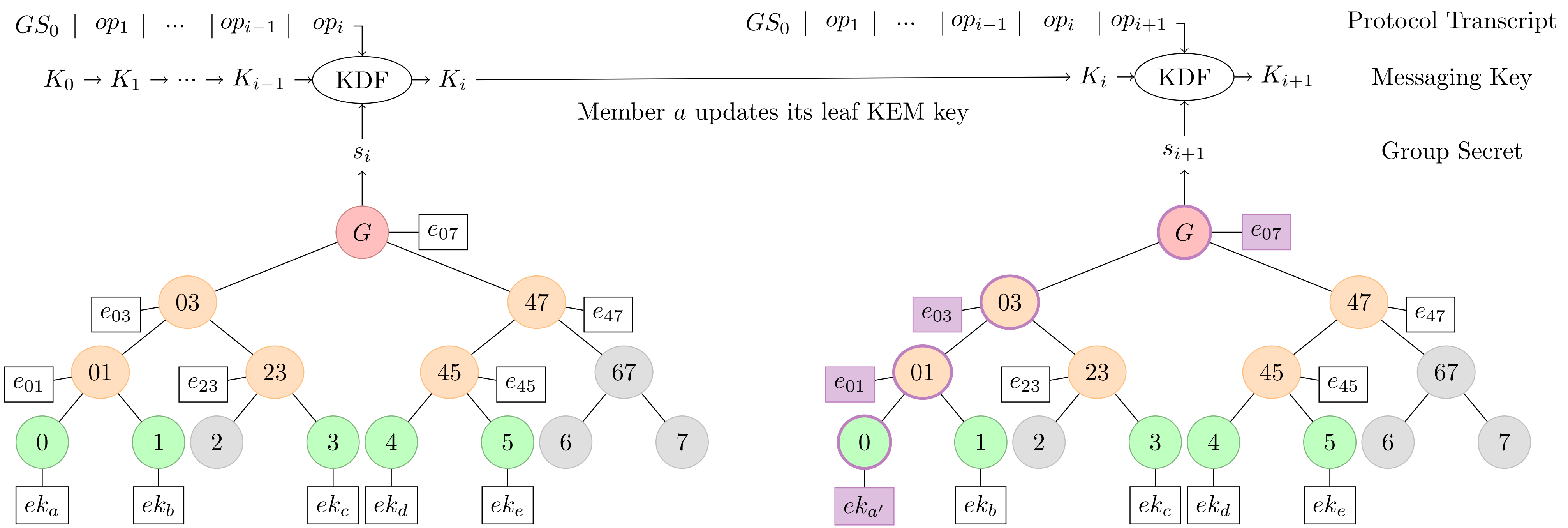
$e_{47} \stackrel{\text{def}}{=} \textbf{left}, ek_{45}, \_$

$e_{07} \stackrel{\text{def}}{=} \textbf{right}, ek_{07}, \textbf{enc}(dk_{07}, ek_{03})$

O(N) Public Key operations for the sender on Creation
O(log N) Public Key operations for the receiver of Create
O(log N) Public Key operations for the sender on Add, Update and Remove
O(1) Public Key operations for the receiver

TreeKEM: Tree-based Group Key Agreement for MLS

Update

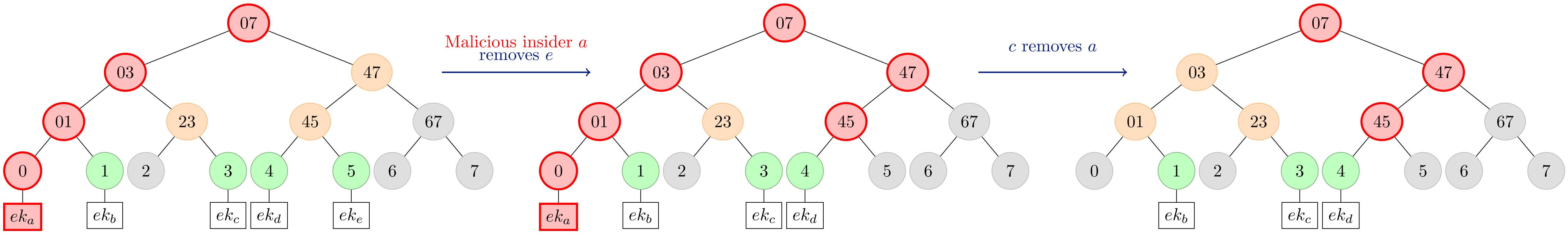


Member A updates its public key encryption keypair, derives new intermediate values and encrypts them for the sibling subgroup.

O(logN) public key encryptions for the sender. O(1) decryptions for the receiver.

The Double Join problem in TreeKEM

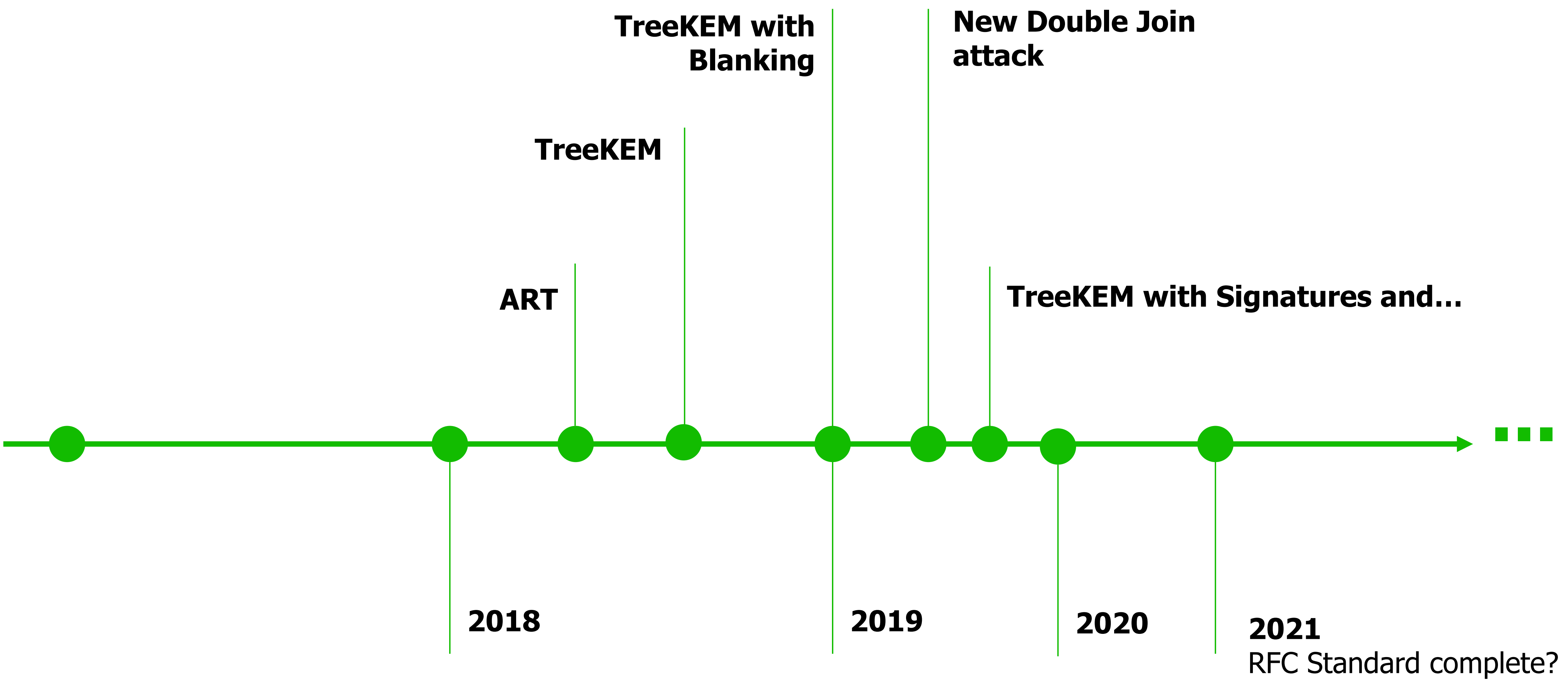
TreeKEM



The sender removes the leaf and provides new keys for all the subgroups the removed node belonged to.

Because A provides secrets for nodes 45 and 47 to their subgroups
it might still know these after being removed

MLS Evolution



Challenges of formalizing MLS

Succinct formal specification.
An IETF document is quite informal...

Build an executable model for group messaging.
Testing multi-party protocols like MLS is difficult!

Prove security for arbitrary size groups.
Analyzing recursive data-structures like trees require induction
Handling an unbounded number of group participants is hard for automated tools

F* is the only framework that can handle all these aspects!

Formalizing Group Messaging in F*

Types and functions for

1. group states: trees
2. group operations: add, remove, update
3. group secrets: TreeKEM computations

```
(* Public Information about a Group Member *)
type member_info = {
  cred: credential;
  version: nat;
  current_enc_key: enc_key }

(* Secrets belonging to a Group Member *)
val member_secrets: datatype

(* Group State Data Structure *)
val group_state: datatype
val group_id: group_state → nat
val max_size: group_state → nat
val epoch: group_state → nat
type index (g:group_state) = i:nat{i < max_size g}
type member_array (sz:nat) =
     a:array (option member_info){length a = sz}
val membership: g:group_state → member_array (max_size g)

(* Create a new Group State *)
val create: gid:nat → sz:pos → init:member_array sz
        → entropy:bytes → option group_state

(* Group Operation Data Structure *)
val operation: datatype

(* Apply an Operation to a Group *)
val apply: group_state → operation → option group_state

(* Create an Operation *)
val modify: g:group_state → actor:index g
        → i:index g → mi':option member_info
        → entropy:bytes → option operation

(* Group Secret shared by all Members *)
val group_secret: datatype

(* Calculate Group Secret *)
val calculate_group_secret: g:group_state → i:index g
        → ms:member_secrets → option group_secret
        → option group_secret
```

Figure 5.2 – An F* Interface for MLS Protocols. Each protocol must implement a Group Management and Key Exchange (GMKE) component that establishes a shared group secret.

Formalizing Group Messaging in F*

Types and functions for

1. messages
2. encryption
3. decryption

High-level spec. is 300 lines of F*

Symbolically executable

```
(* Protocol Messages *)
type msg =
      | AppMsg: ctr:nat → m:bytes → msg
      | Create: g:group_state → msg
      | Modify: operation → msg
      | Welcome: g:group_state → i:index g
              → secrets:bytes → msg
      | Goodbye: msg

(* Encrypt Protocol Message *)
val encrypt_msg: g:group_state → gs:group_secret
        → sender:index g → ms:member_secrets → m:msg
        → entropy:bytes → (bytes * group_secret)

(* Decrypt Initial Group State *)
val decrypt_initial: ms:member_secrets
        → c:bytes → option msg

(* Decrypt Protocol Message *)
val decrypt_msg: g:group_state → gs:group_secret
        → receiver:index g → c:bytes
        → option (msg * sender:index g * group_secret)
```

Figure 5.3 – An F* Interface for MLS Protocols. Each protocol must implement a Message Protection (MP) component that uses the group secret to protect messages.

Ongoing work for security analysis

Perspective as one of the designers of MLS

Ensure that the protocol can be studied and modelled using current formal analysis techniques.

Update the protocol to include feedback from research teams.

Security analysis and improvements for the IETF MLS standard for group messaging
J Alwen, S Coretti, Y Dodis, Y Tselekounis - Annual International …, 2020 - Springer

[PDF] Efficient Post-Compromise Security Beyond One Group
C Cremers, B Hale, K Kohbrok - 2019 - eprint.iacr.org

Keep the dirt: tainted treekem, adaptively and actively secure continuous group key agreement
J Alwen, M Capretto, M Cueto, C Kamath, K Klein… - 2019 - computer.org

[PDF] Key Agreement for Decentralized Secure Group Messaging with Strong Security Guarantees
M Weidner, M Kleppmann, D Hugenroth, AR Beresford - 2020 - eprint.iacr.org

[PDF] An Analysis of TLS 1.3 and its use in Composite Protocols
J Hoyland - 2018 - core.ac.uk

End-to-end secure mobile group messaging with conversation integrity and deniability
M Schliep, N Hopper - Proceedings of the 18th ACM Workshop on …, 2019 - dl.acm.org

[PDF] An Analysis of Hybrid Public Key Encryption.
B Lipp - IACR Cryptol. ePrint Arch., 2020 - eprint.iacr.org

…

Formal Models and Verified Protocols for Group Messaging: Attacks and Proofs for IETF MLS
K Bhargavan, B Beurdouche, P Naldurg

Ongoing work for security analysis

Perspective from the researcher side

We have written executable formal specification for an early draft (-06) and did a symbolic security analysis.

It is missing new elements and we are in the process of updating the specification and proofs.

Our goal is to have a full proof to publish alongside the RFC.

Finally we want to have a verified implementation.

Conclusions

Conclusions

Contributed to a real-world verified cryptographic library

created libraries, wrote verified primitives, developed a new workflow to include code in multiple products.

Analysed and implemented real world protocols

Found attacks on TLS 1.2 and helped build a verified interoperable implementation of Signal.

Designed a new group messaging protocol

Co-authored RFCs for MLS by using formal verification to guide a principled approach.

Looking forward

Towards more complex cryptographic primitives

PQ primitives, zero-knowledge proofs would certainly benefit from verified implementations

Bridging the gap between formally verified implementations and cryptographic proofs

Link proofs from tools like CryptoVerif with F* implementations

Improving the verification toolchain

reducing the trusted code base, reducing proof effort…

Thank you !