Verified Correctness and Security of OpenSSL HMAC

Lennart Beringer, Katherine Ye, Andrew W. Appel

Adam Petcher

Princeton

Harvard

MIT

Research supported by DARPA HACMS & Google ATAP

24th Usenix Security Symposium, Washington DC, August 12-14, 2015
Recently in the news
Wired.com, July 21st, 2015

Hackers Remotely Kill a Jeep on the Highway—With Me in It
Also

Wired.com, July 21st, 2015

skateboards...

gas pumps ...

sniper rifles...

medical devices...

cargo tracking systems...
Recently in the news

Wired.com, July 21\textsuperscript{st}, 2015

Hackers Remotely Kill a Jeep on the Highway—With Me in It

One lesson: don’t accept (control) messages unless they are properly authenticated!
Symmetric-key authentication

This talk: hash-based message authentication (HMAC)

“Keying Hash Functions for Message Authentication”, Mihir Bellare, Ran Canetti, Hugo Krawczyk, Crypto 96
3. **Explanation.** This Standard specifies an algorithm for applications requiring message authentication. Message authentication is achieved via the construction of a message authentication code (MAC). MACs based on cryptographic hash functions are known as HMACs.

The purpose of a MAC is to authenticate both the source of a message and its integrity without the use of any additional mechanisms. HMACs have two functionally distinct parameters, a message input and a secret key known only to the message originator and intended receiver(s). Additional applications of keyed-hash functions include their use in challenge-response identification protocols for computing responses, which are a function of both a secret key and a challenge message.
FIPS-198: functional specification

Inputs/outputs given as sequences of **bytes**
- length of **Data**: unconstrained
- length of **Key**: unconstrained, but normalization/padding to suit hash function (**SHA-256**)
- length of **output**: same as output of hash function
- **ipad, opad**: fixed-size constants
What could go wrong?

Is HMAC a PRF (assuming SHA256 is a PRF)? Cryptographers may have made a mistake in the proof....
What could go wrong?

Does FIPS-198-HMAC enjoy the same crypto-properties? Authors of FIPS may have accidentally introduced a mistake, or made additional/different assumptions…
What could go wrong?

Does a C implementation of HMAC compute the correct function? Implementors may have missed some subtle aspect of FIPS-198...or FIPS-180 for SHA...
What could go wrong?

Does a C implementation compute the correct function?

Does the C implementation perform additional memory access operations?

Implementors may have made programming mistakes....
What could go wrong?

Did the implementor and the compiler writer agree on what C is?
What could go wrong?

Is HMAC a PRF (assuming SHA256 is a PRF)? Cryptographers may have made a mistake in the proof.…

Does FIPS-198-HMAC enjoy the same crypto properties? Authors of FIPS may have accidentally introduced a mistake, or made additional/different assumptions.…

Does a C implementation of HMAC compute the correct function? Implementors may have missed some subtle aspect of FIPS-198.…

Does the C implementation perform additional memory access operations? Implementors may have made programming mistakes.…

Did the implementor and the compiler writer agree on what C is? Is the compiler correct?
What could go wrong?

What’s the compiler’s view of the processor? Side channels: timing, caches, memory model, ...

Did the implementor and the compiler writer agree on what C is?
Machine-checked functional and cryptographic correctness

Coq

FCF
HMAC (Coq)
HMAC equiv.
HMAC (Bellare)

SHA (FIPS-180)
SHA (Coq)
SHA (C)
HMAC (Coq)
HMAC (C)

VST

CompCert

Correctness of C code w.r.t. functional specification
Cryptographic verification
Memory safety, nonleakage, and integrity (VST)
Compiler correctness w.r.t. a formal processor model
FIPS-198: transcription into Coq

Definition NormAndPad k := zeropad (if |k| > 64 then SHA-256 k else k).

Definition HASH l m := SHA-256 (l++m).

Definition HmacCore m k := HASH (opad ⊕ k) (HASH (ipad ⊕ k) m)

Definition HMAC256 (m k : list Z) : list Z :=
let key = map Byte.repr (NormAndPad k)
in HmacCore m key

Inputs/outputs given as (Coq) lists of (mathematical) values

Specification function HMAC256 is ...

• ... amenable to mathematical reasoning inside Coq
• ...executable inside Coq, and extractable to Ocaml
Implementation: OpenSSL*

Client-visible data structure

```
struct HMAC_CTX {
    SHA_CTX md_ctx;
    SHA_CTX iSha;
    SHA_CTX oSha;
    unsigned int key_len;
    unsigned char key[64];
} // assumed stack allocated
// hence no alloc function
```

"Incremental" API

```
void HMAC_Init (HMAC_CTX c, unsigned char *k, int len);
   //initialize iSha/oSha with PaddedKey XOR ipad/opad if
   //needed, copy the former into md_ctx

void HMAC_Update (HMAC_CTX c, const void *data, int len);
   //hash data onto md_ctx, ie to inner sha

void HMAC_Final (HMAC_CTX c, unsigned char *md);
   //finish md_ctx (inner sha), memcpy result to temp, copy
   //oSha to md_ctx, hash temp onto it & finish it,
   //return result in md

void HMAC_Cleanup (HMAC_CTX c);
   //memset entire struct to 0

unsigned char *HMAC (...k, klen, data, dlen, md...);
   //create local HMC_CTX c, call above functions in order
```

* Statically linked version -- modern versions use envelopes ("home-brew" object system), engines...
Linking the C program to the functional specification in Coq (I)

Step 1: Model the incremental API in Coq

Coq data type matching HMAC_CTX:

Inductive HMAC-ABS := hmacabs: s256ABS → s256ABS → s256ABS → HMAC-ABS.
Linking the C program to the functional specification in Coq (II)

Step 1: Model the incremental API in Coq

Coq data type matching **HMAC_CTX**:

\[
\text{Inductive } \text{HMAC-ABS} := \text{hmacabs: s256ABS} \rightarrow \text{s256ABS} \rightarrow \text{s256ABS} \rightarrow \text{HMAC-ABS}.\]

Relations characterizing incremental functions:

**Definition** init (key:list Z) (h:HMAC-ABS):Prop := ... \text{sha_init} ...

**Definition** update (data:list Z) (h1 h2:HMAC-ABS):Prop := ... \text{sha_update} ...

**Definition** final (h:HMAC-ABS) (mac: list Z):Prop := ... \text{sha_finish} ...
Linking the C program to the functional specification in Coq (III)

Step 1: Model the incremental API in Coq

Coq data type matching `HMAC_CTX`:

```
Inductive HMAC-ABS := hmacabs: s256ABS --> s256ABS --> s256ABS --> HMAC-ABS.
```

Relations characterizing incremental functions:

**Definition** init (key:list Z) (h:HAMC-ABS):Prop := ... sha_init...

**Definition** update (data:list Z) (h1 h2:HAMC-ABS):Prop :=... sha_update ...

**Definition** finish (h:HAMC-ABS) (mac: list Z):Prop:= ... sha_finish...

**Lemma** : init key h \(\langle\) update data h h1 \(\rangle\) \(\langle\) final h1 mac \(\rangle\) \(\rightarrow\) mac = HMAC256 data key.

**Proof** ... Qed.

**Definition** HMAC256 m k = ... (see above)...

```
struct HMAC_CTX {
  SHA_CTX md_ctx;
  SHA_CTX iSha;
  SHA_CTX oSha
}
```
Linking the C program to the functional specification in Coq (IV)

Step 2: VST specifications for all functions, referencing the abstract model

- Representation predicates in Separation logic

```coq
Definition hmacState := shaState * shaState * shaState

Definition hmacRelate (h:HMAC-ABS) (r:hmacState) :=
  (* ...shaRelate on mdCtxt, iSha, Osha components ...*)

Definition hmacRep (h:HMAC-ABS) (c:val) :=
  EX r:hmacState, !!hmacRelate h r && data_at HMAC_CTX r c.
```

Repr of CTX as C values
Link with repr in Coq
SL predicate
Linking the C program to the functional specification in Coq (V)

Step 2: VST specifications for all functions, referencing the abstract model

- Representation predicates in Separation logic

\[
\text{Definition } \text{hmacState} := \text{shaState} * \text{shaState} * \text{shaState}
\]

\[
\text{Definition } \text{hmacRelate} (h: \text{HMAC-ABS}) (r: \text{hmacState}) :=
\]
\[
(* \text{... shaRelate on mdCtx, iSha, Osha components ...]*)
\]

\[
\text{Definition } \text{hmacRep} (h: \text{HMAC-ABS}) (c: \text{val}) :=
\]
\[
\text{EX } r: \text{hmacState}, \text{!!hmacRelate } h \ r \ \&\& \ \text{data_at HMAC_CTX } r \ c.
\]

- Hoare-style specification with pre- and post-conditions

\[
\text{Definition } \text{Update-spec} := \text{DECLARE } _\text{UPDATE}
\]
\[
\text{WITH } h \ c \ \text{data } d
\]
\[
\text{PRE } [\ ... ]
\]
\[
\text{PROP(} ... \text{) LOCAL(} ... \text{) SEP(} ... \text{hmacRep } h \ c, \ \text{data_block } \text{data } d )
\]
\[
\text{POST } [\ \text{returnType }]
\]
\[
\text{EX} \ h1, \ \text{PROP(} \text{update } \text{data } h \ h1 \text{) LOCAL()}
\]
\[
\text{SEP(} ... \text{hmacRep } h1 \ c, \ \text{data_block } \text{data } d ).
\]
Definition HMAC256-spec :=
DECLARE _HMAC
WITH kp: val, key:DATA, KV:val; mp: val, msg:DATA, shmd: share, md: val
PRE [ _key OF tptr tuchar, _keylen OF tint, _d OF tptr tuchar,
     _n OF tint, _md OF tptr tuchar ]
PROP(writable share shmd; has lengthK (LEN key) (CONT key);
    has lengthD 512 (LEN msg) (CONT msg))
LOCAL(temp _md md; temp _key kp; temp _d mp; temp _n (Vint (Int.repr (LEN msg)));
    temp _keylen (Vint (Int.repr (LEN key))); gvar _K256 KV)
SEP(`(data-block Tsh (CONT key) kp); `(data-block Tsh (CONT msg) mp);
    (K-vector KV); `(memory-block shmd (Int.repr 32) md))
POST [ tvoid ]
PROP()
LOCAL()
SEP(`(K-vector KV); `(data-block shmd (HMAC256 (CONT msg) (CONT key)) md);
    `(data-block Tsh (CONT key) kp); `(data-block Tsh (CONT msg) mp)).
Code verification

Interactively apply the rules of VST logic, using forward symbolic execution

Directly carried out in Coq, operating on CompCert AST
Cryptographic soundness

An abridged version of this paper appears in Advances in Cryptology – Crypto 96 Proceedings, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

Keying Hash Functions for Message Authentication

Mihir Bellare* Ran Canetti† Hugo Krawczyk‡

Abstract

The use of cryptographic hash functions like MD5 or SHA for message authentication has become a standard approach in many Internet applications and protocols. Though very easy to implement, such schemes are generally based on ad hoc techniques that lack a sound security basis.

We show that the standard message authentication schemes based on a cryptographic hash function, such as CBC-MAC and HMAC, are proven to be secure as long as the underlying hash function is (weakly) collision resistant.

In addition our schemes are efficient and use a black box, so that widely available hash functions can be used.

Theorem 4.1 If the keyed compression function \( f \) is an \((\epsilon_f, q, t, b)\)-secure MAC on messages of length \( b \) bits, and the keyed iterated hash \( F \) is \((\epsilon_F, q, t, L)\)-weakly collision-resistant then the NMAC function is an \((\epsilon_f + \epsilon_F, q, t, L)\)-secure MAC.
Foundational Cryptography Framework (FCF)

- Probabilistic programming language implemented in Coq, with a means for sampling uniformly random bit vectors
- Library of distributions, lemmas for bounding differences between events etc
- Game-based crypto proofs: adversary tries to tell oracles apart by interacting

\[
\begin{align*}
\text{Oracle 1 (pseudo-random function)?} \\
\text{Oracle 2 (HMAC)?}
\end{align*}
\]

Reduction of HMAC to hash function (SHA): probability of adversary A to separate HMAC from PRF is determined by probability of adversary B1 to separate HASHFUNCTION from PRF (assumed hard).

**Theorem** HMAC-PRF:

\[
\begin{align*}
\text{PRF-Advantage } &\quad \{0, 1\}^b \{0, 1\}^c \text{ HMAC A } \leq \\
\text{PRF-Advantage } &\quad \{0, 1\}^c \{0, 1\}^c \text{ HASHFUNCTION B1 } \\
&\quad + \ldots \text{ (other terms) } + \ldots
\end{align*}
\]

**Proof.** ... Qed.
Bridging the crypto gap

Coq program (FCF version)

- Specialization to SHA256
- Bits versus Bytes
- Exposure of Merkle-Damgard: iterative use of keyed hash function
- Instantiation of parameters (hash function, ipad/opad, ...)
- Padding (use of SHA-blocks versus byte-list)
- Vector versus lists (Coq)

Coq program HMAC256
Interactive proof assistants sufficiently mature

Domain-specific reasoning support helps automation

Verification of other crypto-primitives in progress

“No specification gaps” proofs possible across multiple abstraction layers: intermediate specs are NOT in TCB

Not covered:
- timing & other side channel attacks
- security-preserving compilation
- appropriateness of cryptographic bounds
- Protocols, OS correctness, ...
Discussion

Interactive proof assistants sufficiently mature

Domain-specific reasoning support helps automation

Verification of other cryptographic primitives in progress

"No specification gaps" proofs possible across multiple abstraction layers:
- Intermediate specs are NOT in TCB
- Interactive proof assistants sufficiently mature
- Domain-specific reasoning support helps automation

use of C -> other tools still apply verification of existing, open-source code: existing analyses still valid

Charlie Miller @0xcharlie • Jul 24
I wonder what is cheaper, designing secure cars or doing recalls?