Verified Correctness and Security of OpenSSL HMAC

Lennart Beringer, Katherine Ye, Andrew W. Appel

Princeton

Adam Petcher

Harvard MIT

Research supported by
DARPA HACMS & Google ATAP

24th Usenix Security Symposium,
Washington DC, August 12-14, 2015

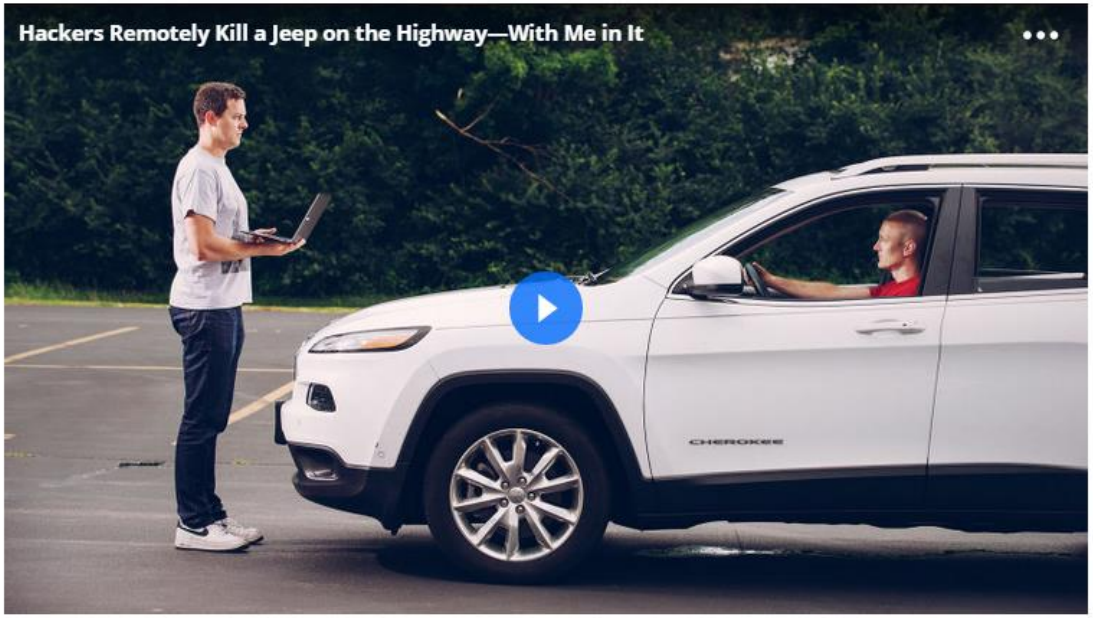
Recently in the news

Wired.com, July 21st, 2015

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

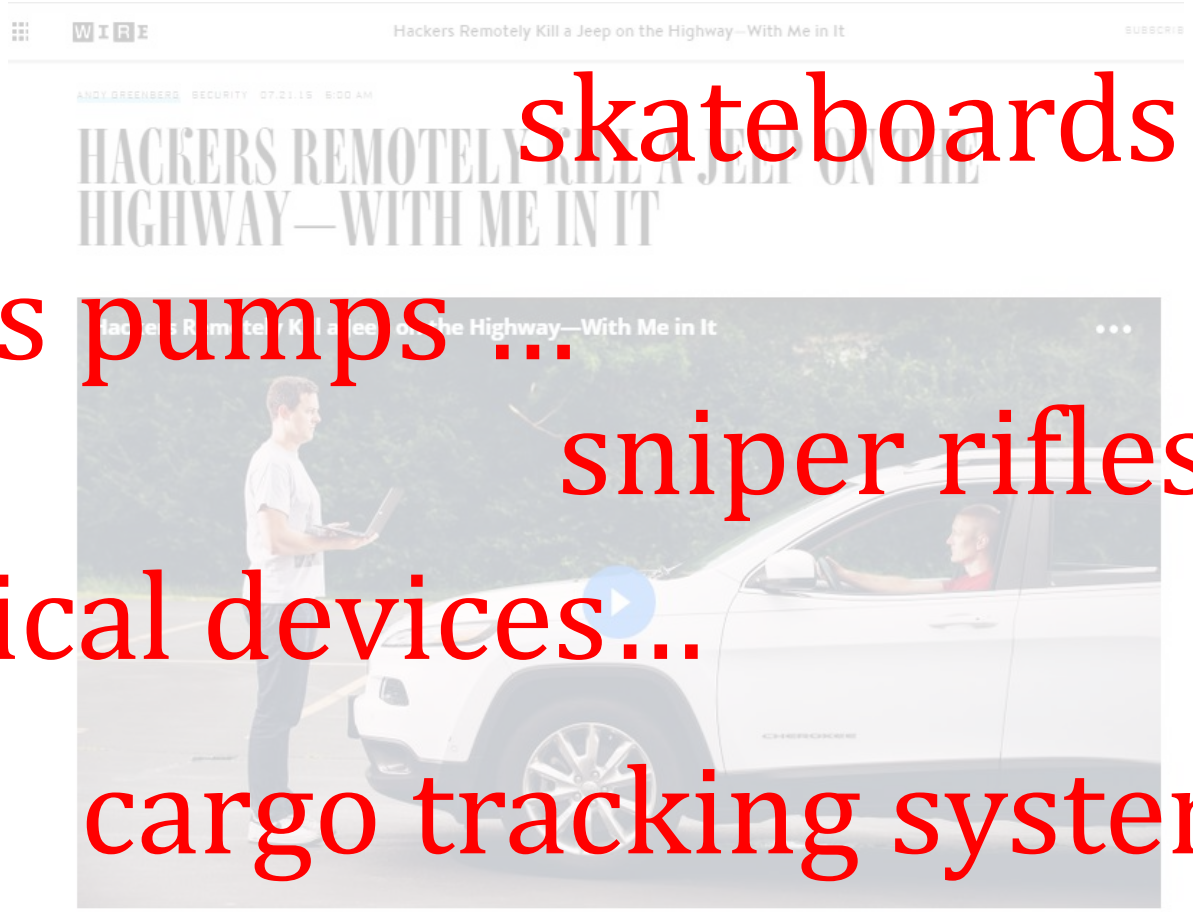
ANDY GREENBERG SECURITY 07.21.15 6:00 AM

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…

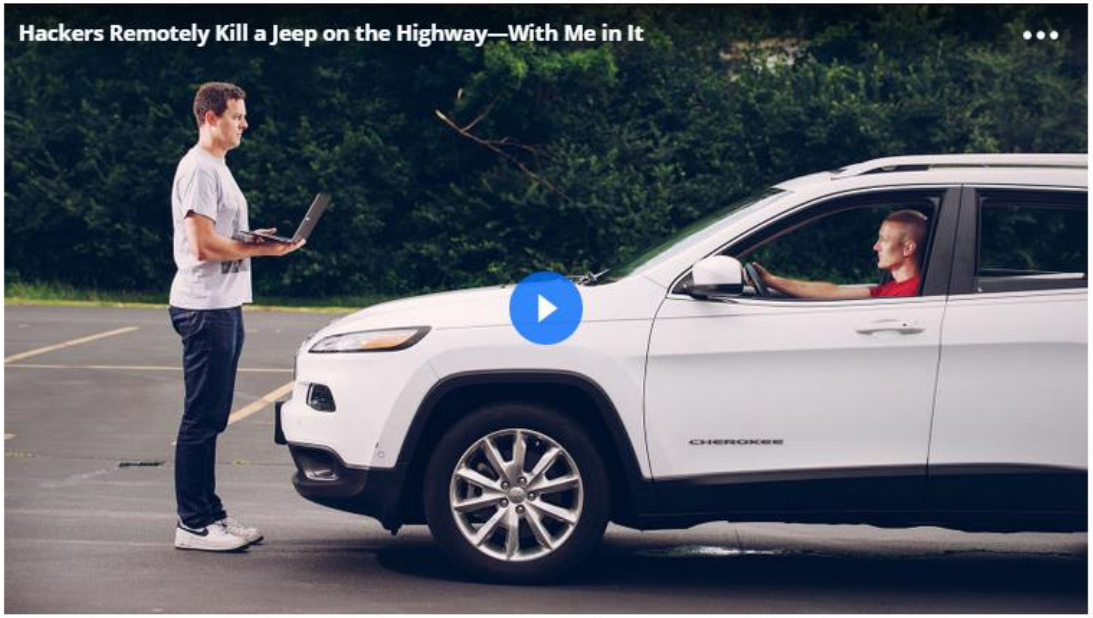
Recently in the news

Wired.com, July 21st, 2015

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

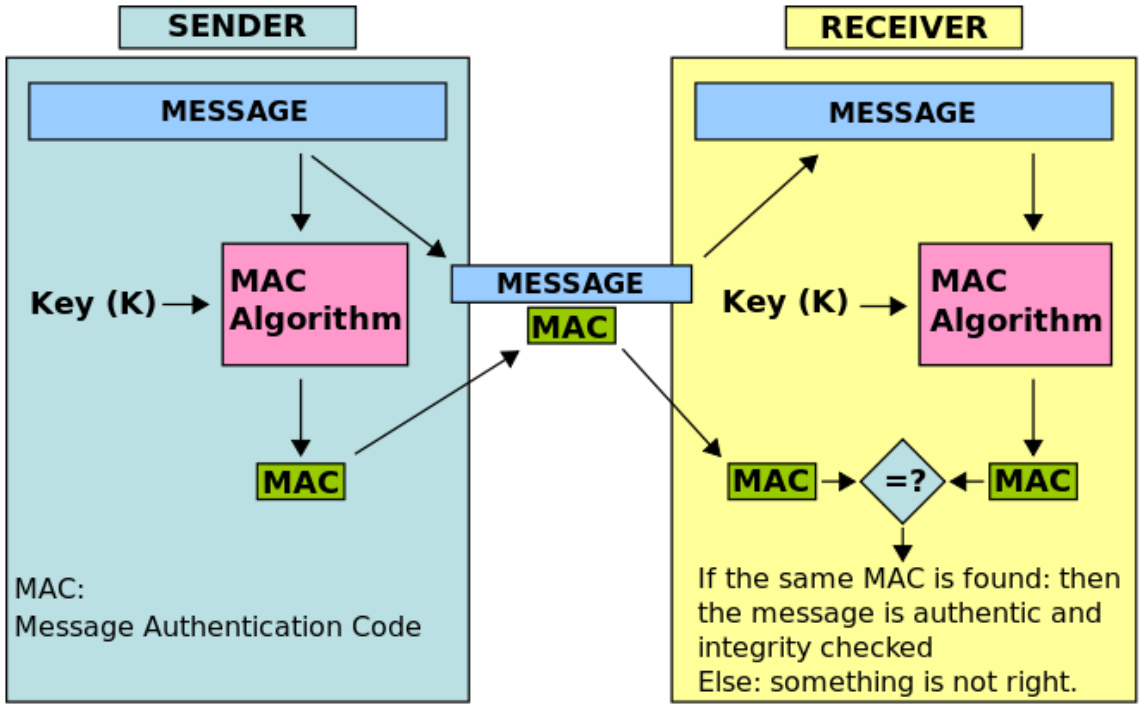
ANDY GREENBERG SECURITY 07.21.15 6:00 AM

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

FIPS 198

FIPS PUB 198-1

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

The Keyed-Hash Message Authentication Code (HMAC)

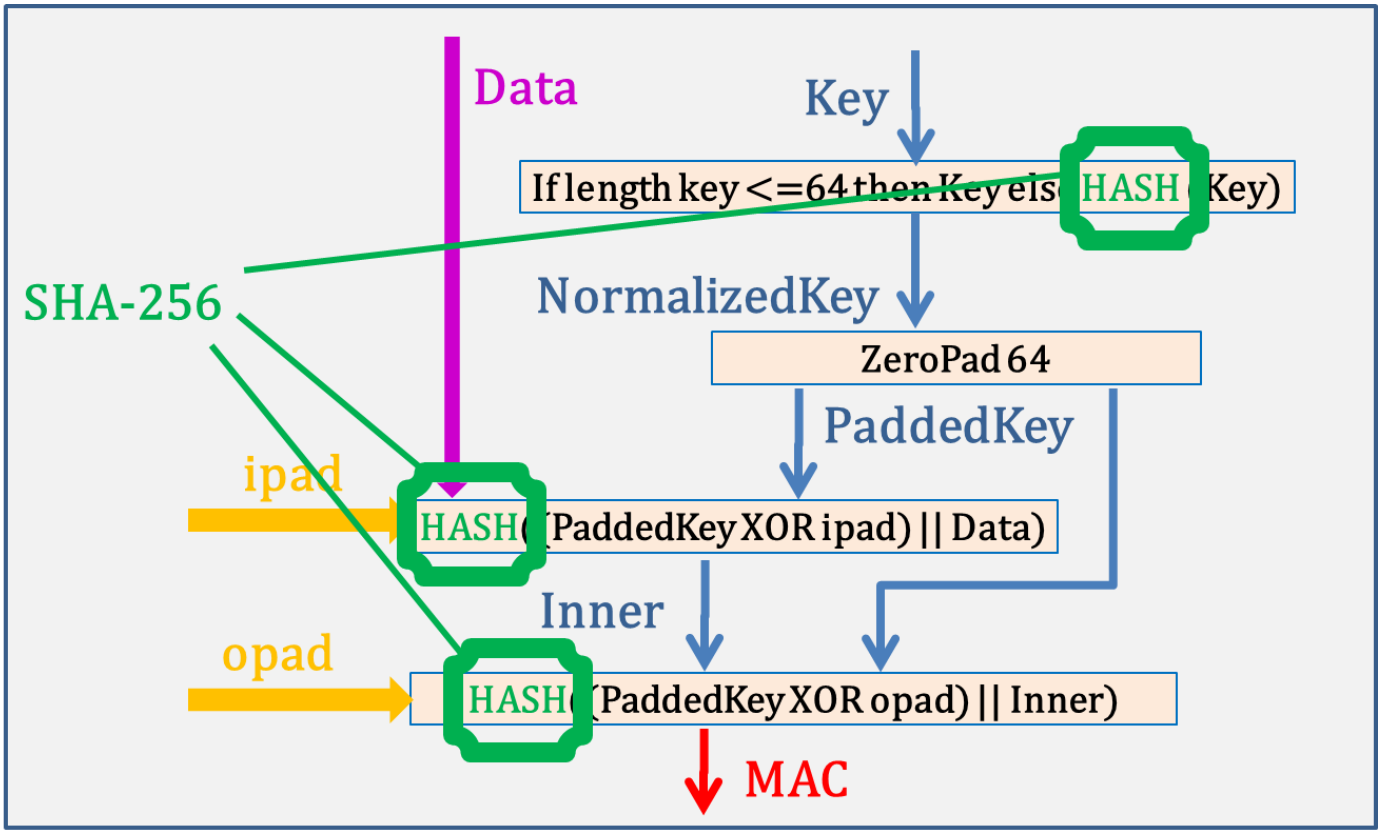
CATEGORY: COMPUTER SECURITY **SUBCATEGORY: CRYPTOGRAPHY**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

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.

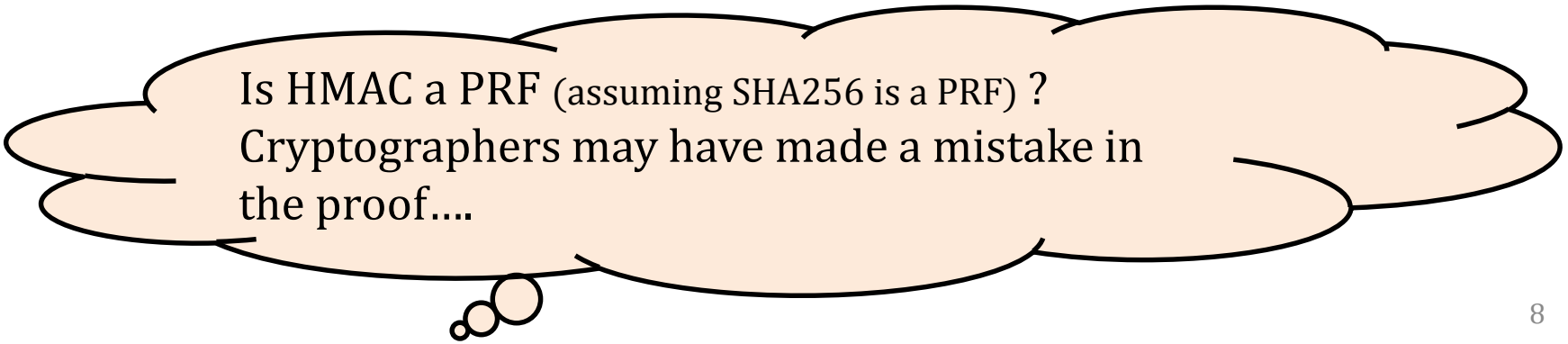
FIPS-198: functional specification



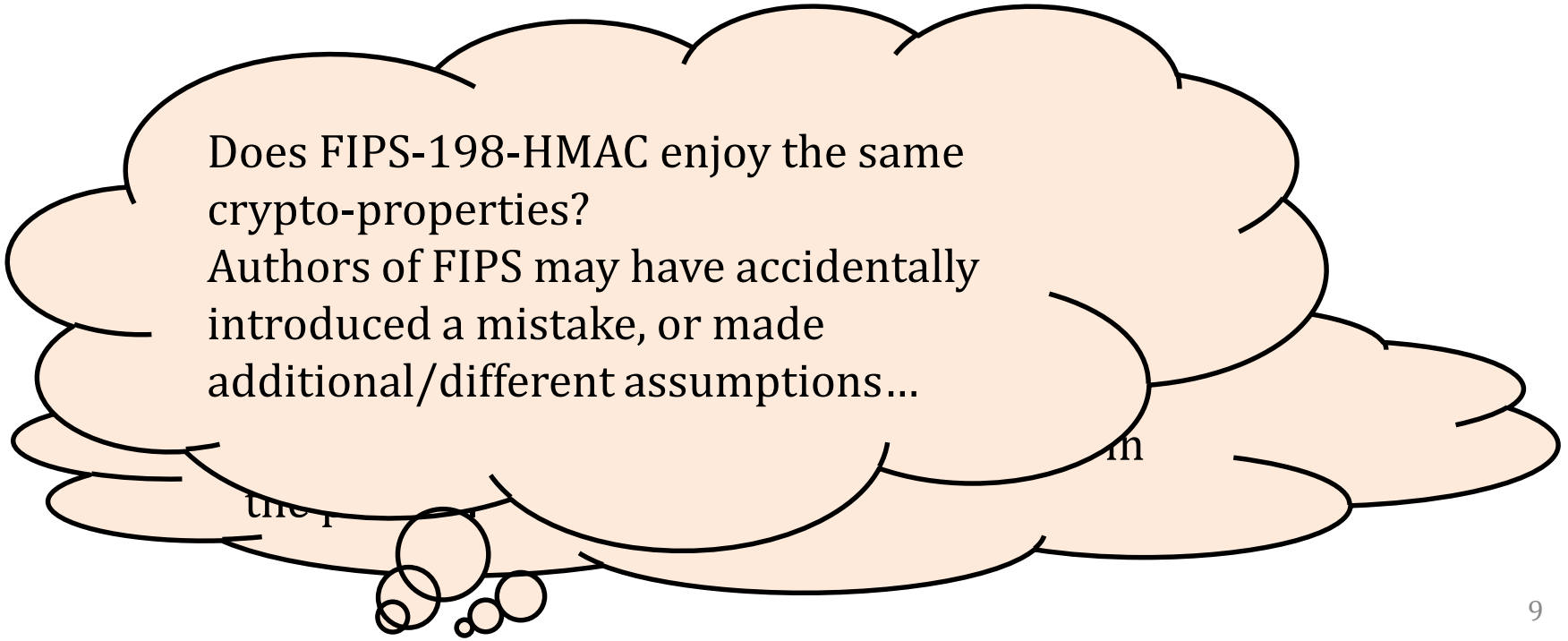
Inputs/outputs given as sequences of **bytes**

- 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

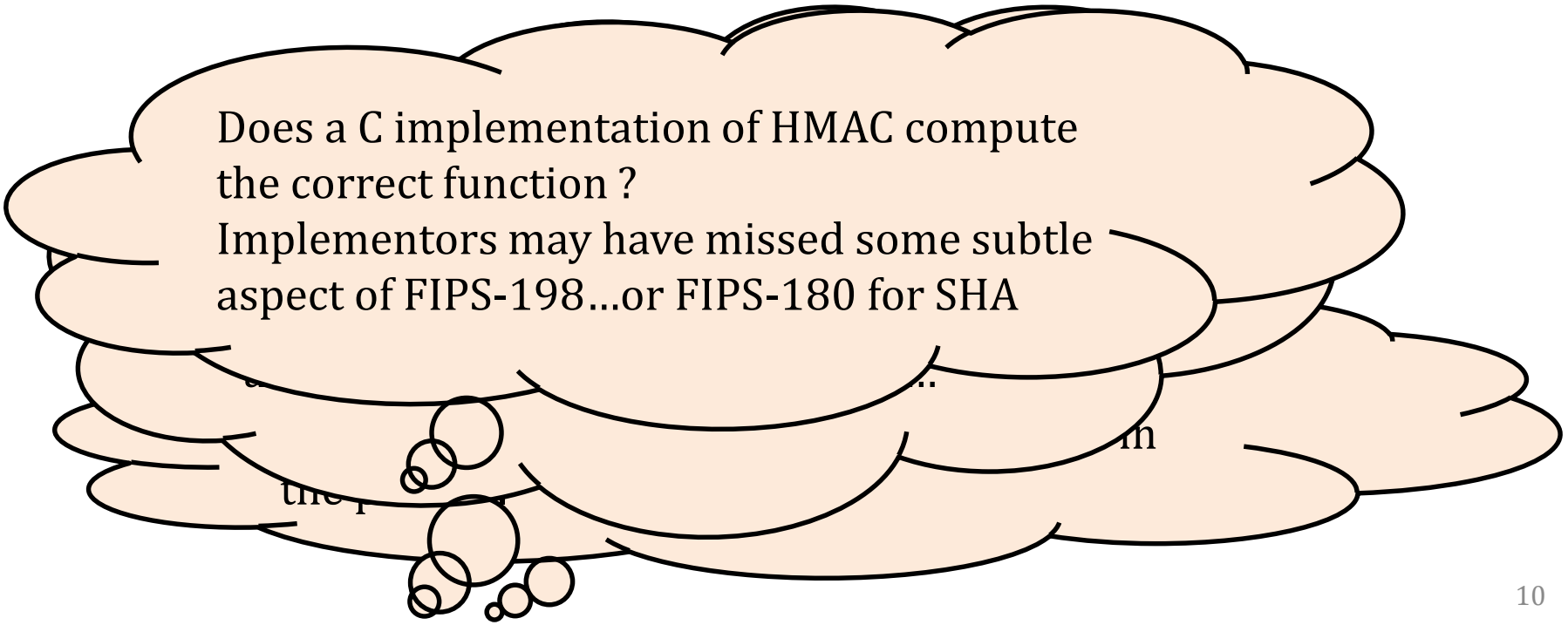
What could go wrong?



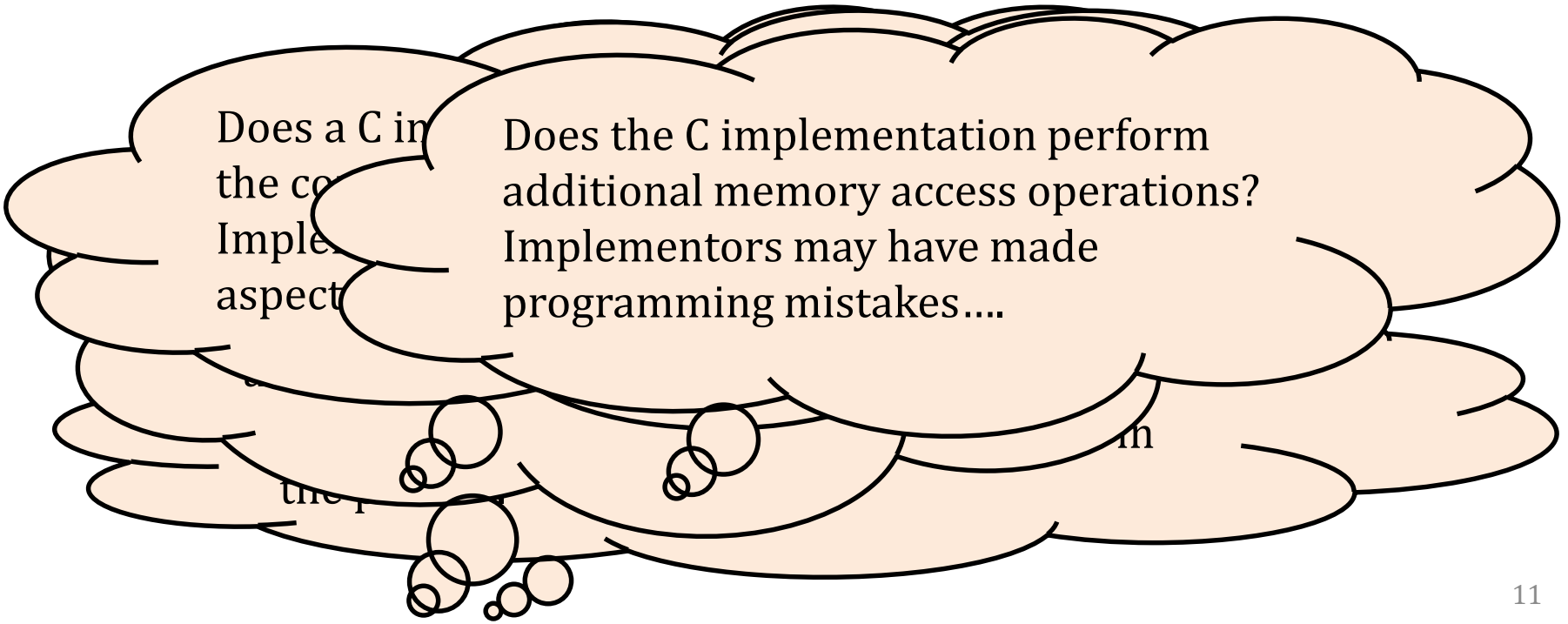
What could go wrong?



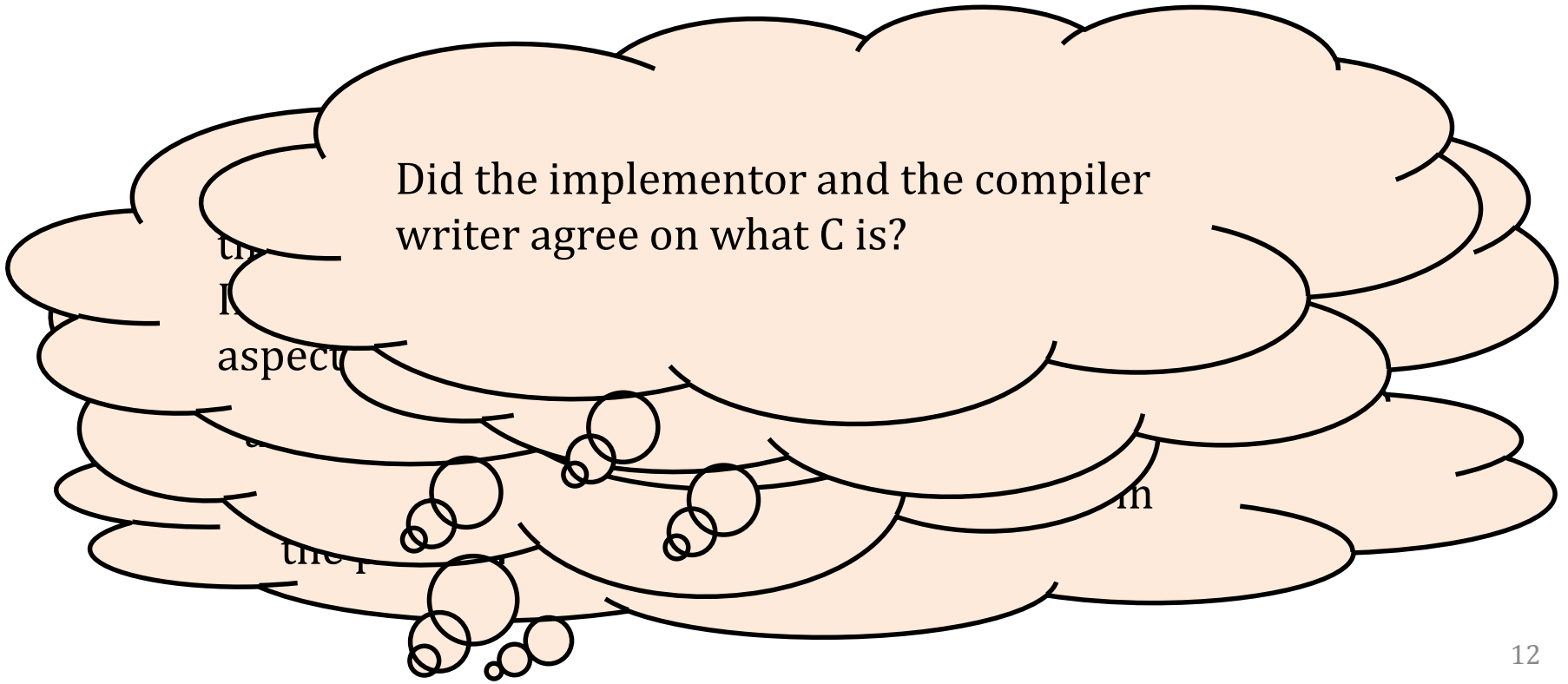
What could go wrong?



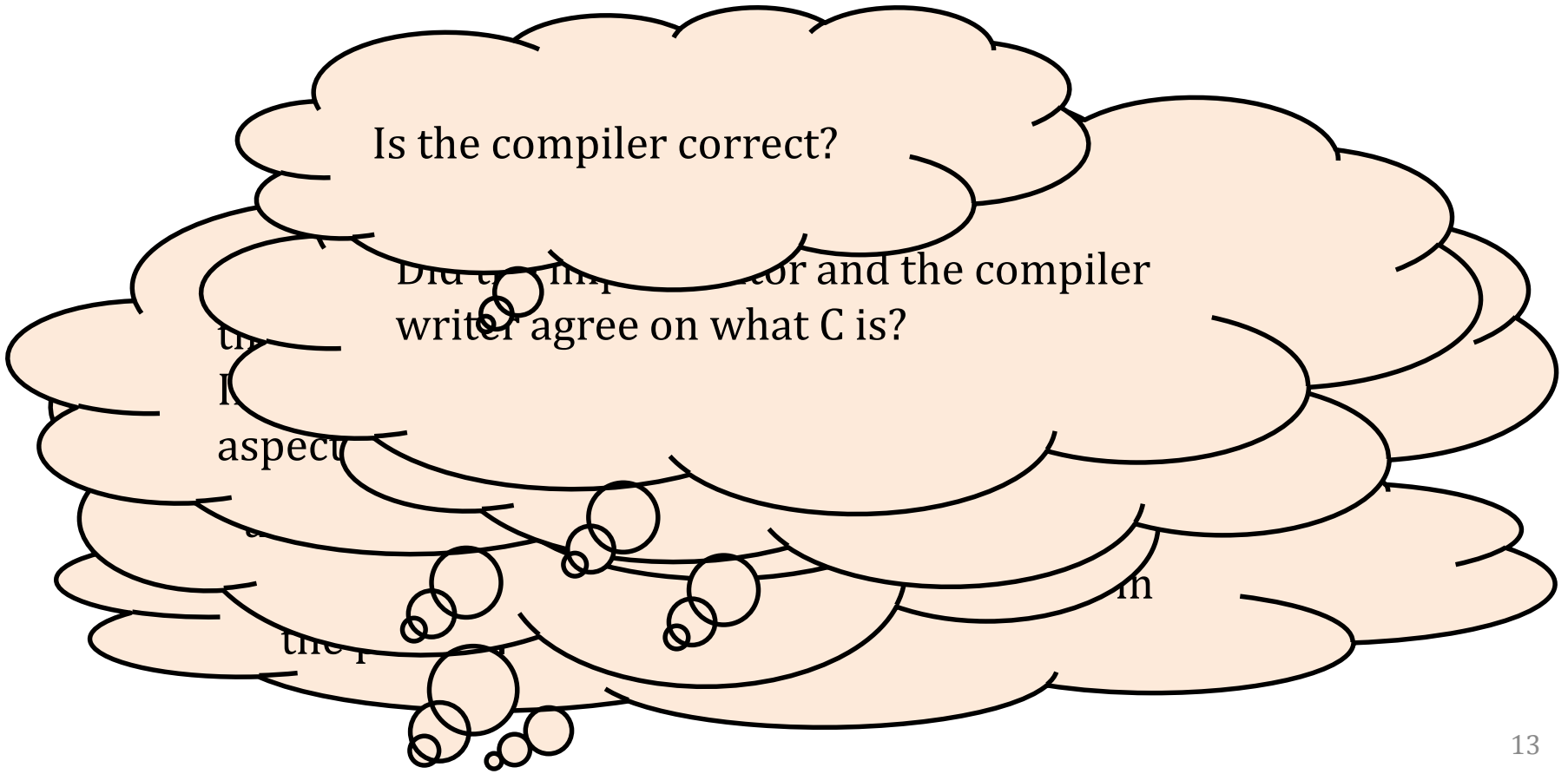
What could go wrong?



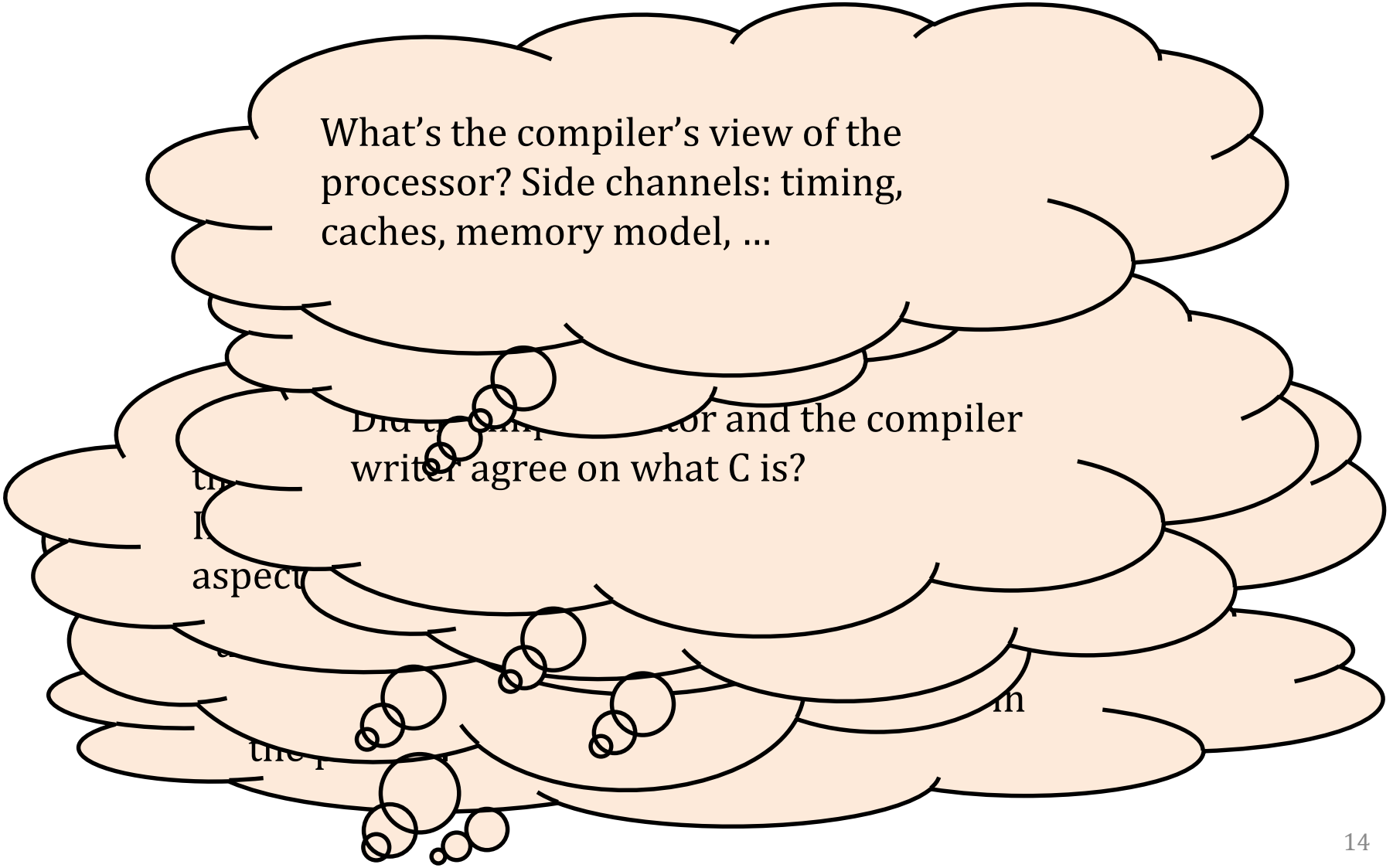
What could go wrong?



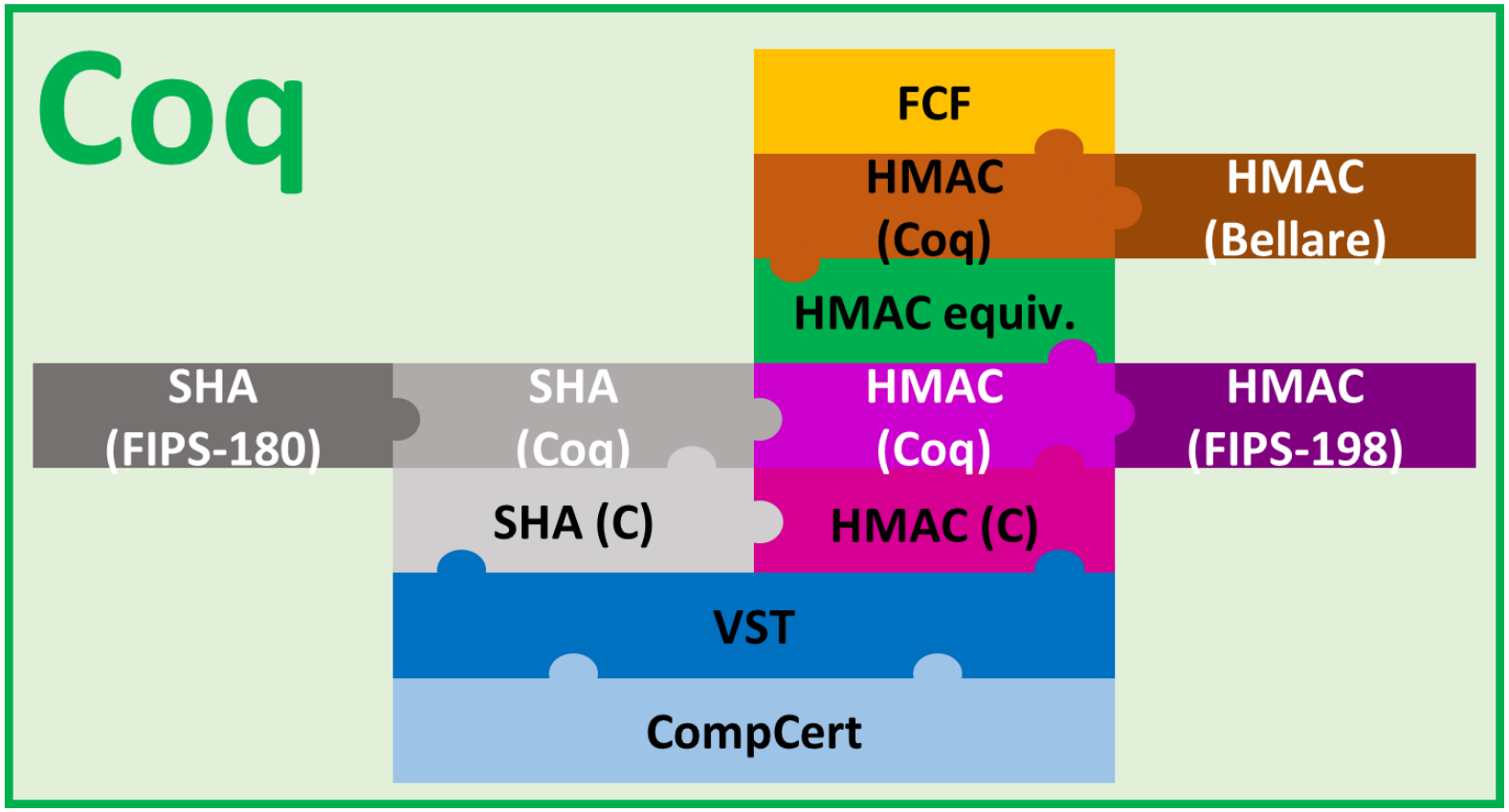
What could go wrong?



What could go wrong?



Machine-checked functional and cryptographic correctness



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

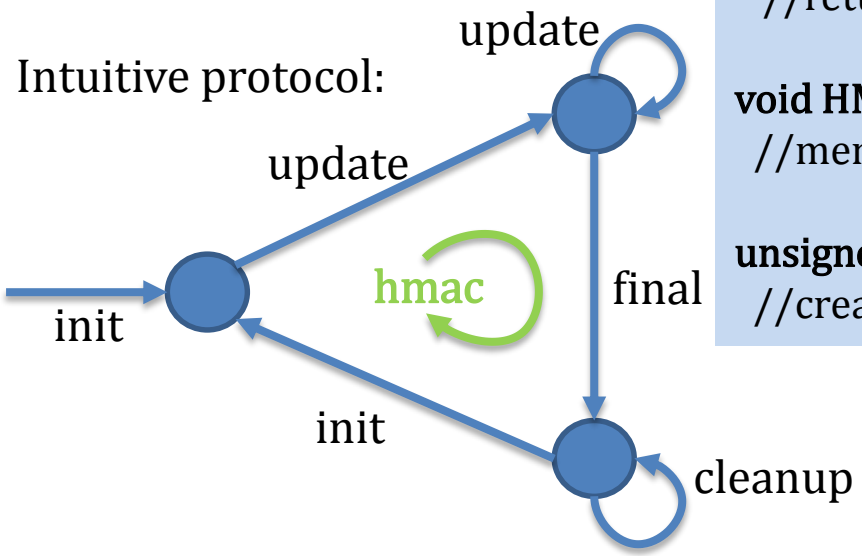
```
struct HMAC_CTX {
  SHA_CTX md_ctx;
  SHA_CTX iSha;
  SHA_CTX oSha;
  unsigned int key_len;
  unsigned char key[64];
}
// assumed stack allocated
// hence no alloc function
```

redundant

"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
```

Intuitive protocol:



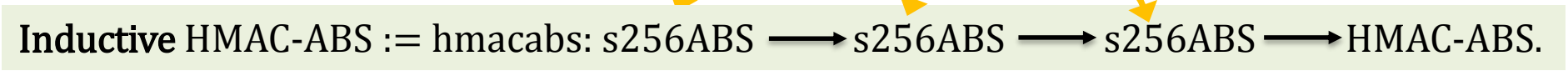
* Statically linked version -- modern versions use envelopes ("home-brew" object system), engines…

Linking the C program to the functional specification in Coq (I)

Step 1: Model the incremental API in Coq

```
struct HMAC_CTX {
  SHA_CTX md_ctx;
  SHA_CTX iSha;
  SHA_CTX oSha}
```

Coq data type matching **HMAC_CTX**:



Linking the C program to the functional specification in Coq (II)

Step 1: Model the incremental API in Coq

```
struct HMAC_CTX {
  SHA_CTX md_ctx;
  SHA_CTX iSha;
  SHA_CTX oSha}
```

Coq data type matching **HMAC_CTX**:

Inductive HMAC-ABS := hmacabs: s256ABS ⟶ s256ABS ⟶ s256ABS ⟶ HMAC-ABS.

Relations characterizing incremental functions:

relations for incremental sha

Definition init (key:list Z) (h:HMAC-ABS):Prop := … sha_init…

Definition update (data:list Z) (h1 h2:HMAC-ABS):Prop :=… sha_update …

Definition final (h:HMAC-ABS) (mac: list Z):Prop:= … sha_finish…

Linking the C program to the functional specification in Coq (III)

Step 1: Model the incremental API in Coq

```
struct HMAC_CTX {
  SHA_CTX md_ctx;
  SHA_CTX iSha;
  SHA_CTX oSha}
```

Coq data type matching **HMAC_CTX**:

Inductive HMAC-ABS := hmacabs: s256ABS ⟶ s256ABS ⟶ s256ABS ⟶ HMAC-ABS.

Relations characterizing incremental functions:

relations for incremental sha

Definition init (key:list Z) (h:HMAC-ABS):Prop := … sha_init…

Definition update (data:list Z) (h1 h2:HMAC-ABS):Prop :=… sha_update …

Definition finish (h:HMAC-ABS) (mac: list Z):Prop:= … sha_finish…

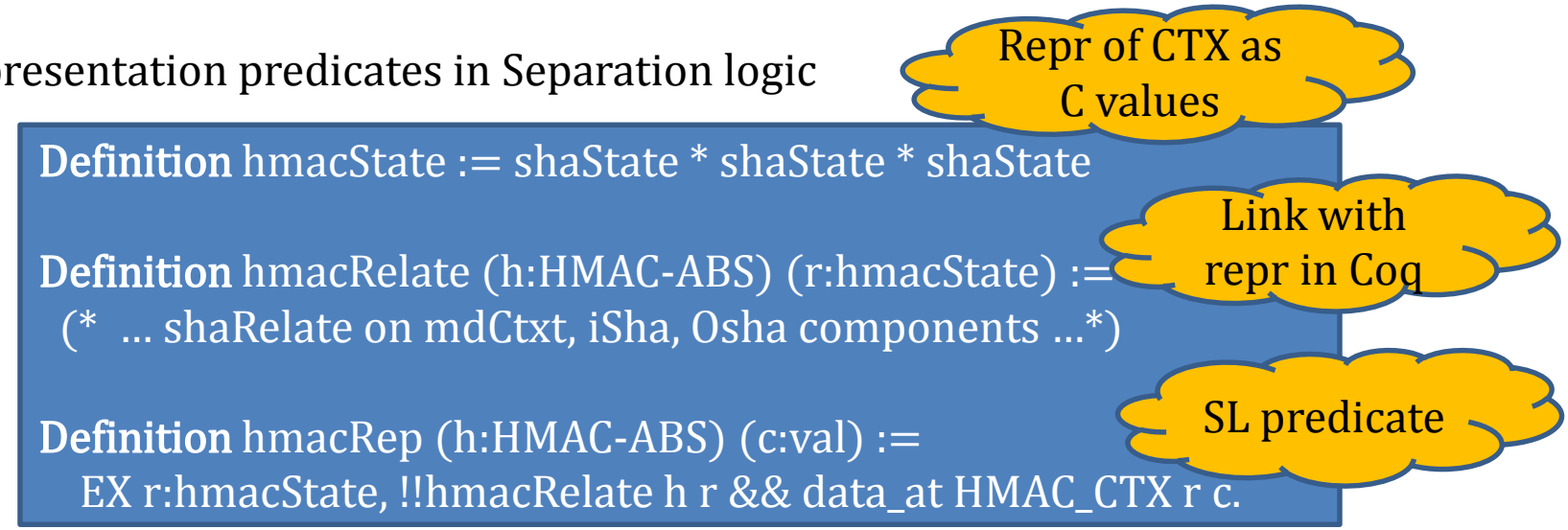
Lemma : init key h /\ update data h h1 /\ final h1 mac ⟶ mac = HMAC256 data key.
Proof. … Qed.

Definition HMAC256 m k = … (see above)…

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



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

Repr of CTX as C values

Link with repr in Coq

SL predicate

```
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.
```

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

```
Definition Update-spec :=DECLARE _UPDATE
WITH h c data d
PRE [ …]
   PROP(…) LOCAL(…)  SEP(… hmacRep h c, data_block data d)
POST [ returnType ]
   EX h1, PROP(update data h h1) LOCAL()
            SEP(… hmacRep  h1 c , data_block data d).
```

API-specification of HMAC function

```
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)).
```

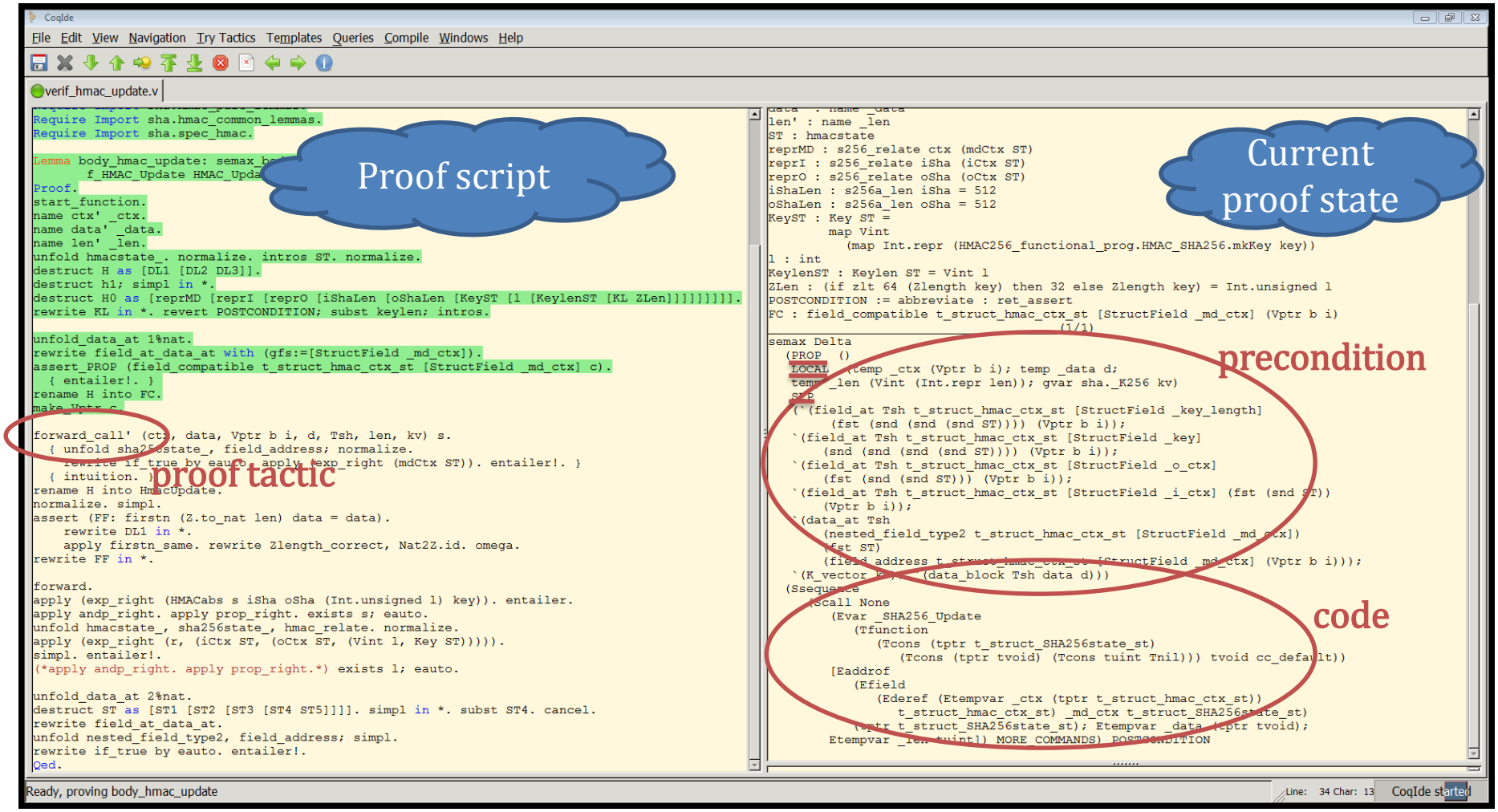
Result here

Key preserved

Message preserved

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

ually based on ad hoc techniques that lack a sound security

message authentication schemes based on a cryptographic

C and HMAC, are proven to be secure as long as the un-

In addition our schemes are efficient an

underlying hash function. Moreover th

a black box, so that widely available li

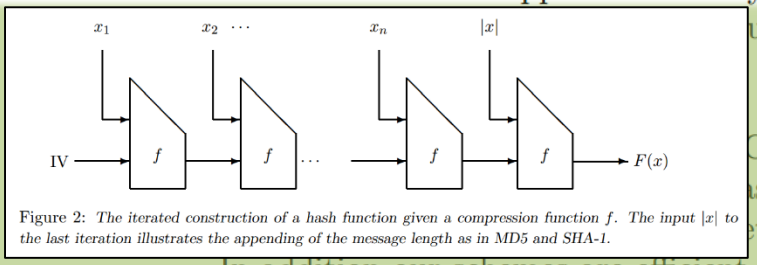


Figure 2: *The iterated construction of a hash function given a compression function f. The input $|x|$ to the last iteration illustrates the appending of the message length as in MD5 and SHA-1.*

4 The Nested Construction NMAC

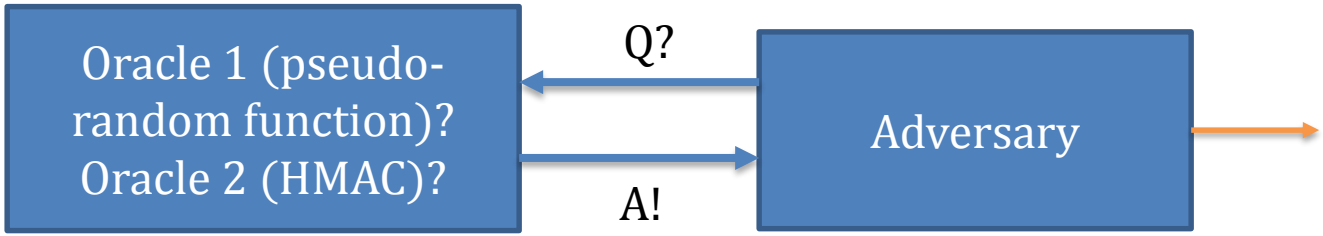
We present our basic construction NMAC (for "Nested MAC") and its analysis. In the next section we describe a variant, HMAC, that is further geared towards practical applications. Denote by f_k and F_k the keyed versions of a given compression function and its iterated function, as described in Section 2.

ndom strings of length ℓ each).
rbitrary length as

Theorem 4.1 If the keyed compression function f is an (ϵ_f, q, t, b)-secure MAC on messages of length b bits, and the keyed iterated hash F is (ϵ_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



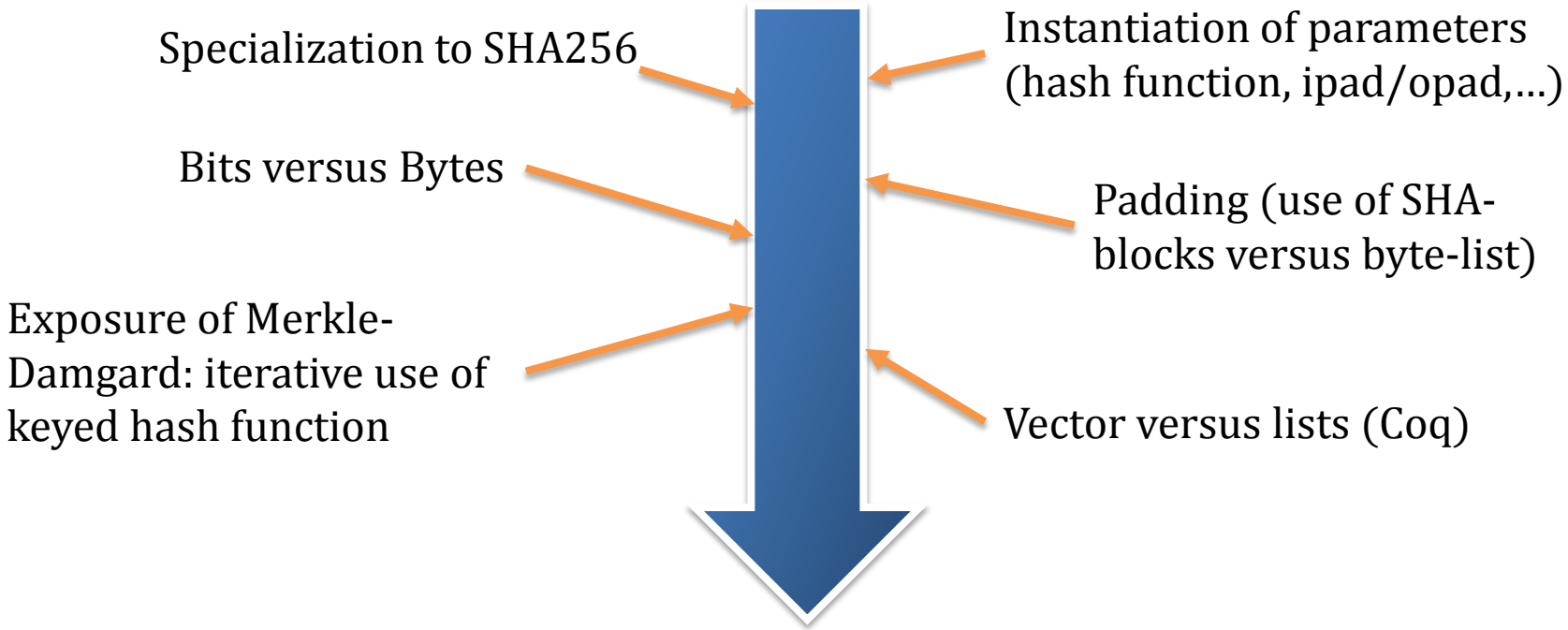
Reduction of HMAC to hash function (SHA): **probability of adversary A to separate HMAC from PRF** is determined by **probability of adversary B1 to separate HASHFUNCTION from PRF** (assumed hard).

Bellare-style: bit-oriented, NMAC-based, iterative,...

Theorem HMAC-PRF:
PRF-Advantage ({0, 1}^b) ({0, 1}^c) HMAC A <=
PRF-Advantage ({0, 1}^c) ({0, 1}^c) HASHFUNCTION B1
+ … (other terms) + ….
Proof. … **Qed**.

Bridging the crypto gap

Coq program (FCF version)



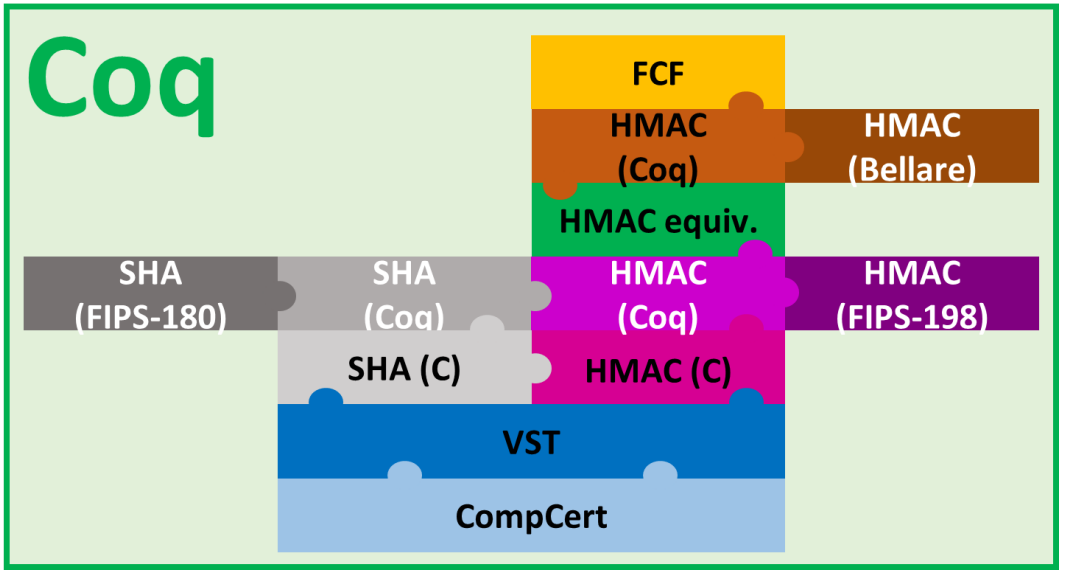
Coq program HMAC256

Discussion

Interactive proof assistants sufficiently mature

Domain-specific reasoning support helps automation

Verification of other crypto-primitives in progress



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

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

Not covered:

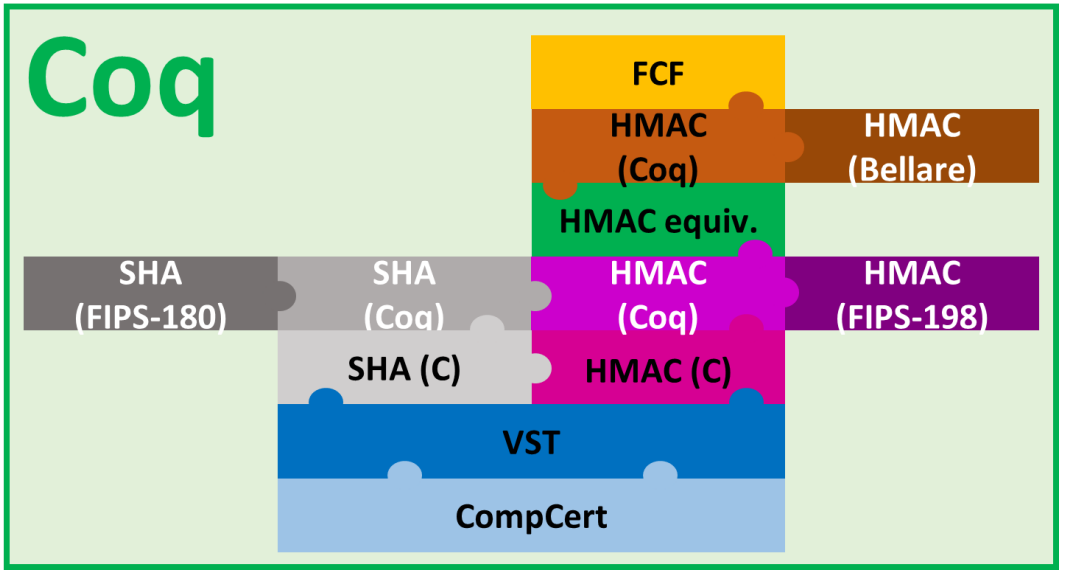
- timing & other side channel attacks
- security-preserving compilation
- appropriateness of cryptographic bounds
- Protocols, OS correctness, …

Discussion

Interactive proof assistants sufficiently mature

Domain-specific reasoning support helps automation

Verification of other crypto-primitives in progress



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

"No spe
across n
interme

Charlie Miller @0xcharlie · Jul 24

I wonder what is cheaper, designing secure cars or doing recalls?

165 128

ttacks
ion
phic bounds