Suppose H is a Merkle-Damgård hash function built from a secure compression function

Several ways to build a keyed function:

1. Prepend key: $F(k, m) := H(k \| m)$
   ↳ Insecure due to structure of Merkle-Damgård: can mount an "extension attack:" given $H(k \| m)$, can compute $H(k \| m \| m')$ by extending Merkle-Damgård chain
2. Append key: $F(k,m) := H(m \| k)$
   ↳ Similar to hash-then-MAC construction and vulnerable to same offline attack: adversary finds a collision in the Merkle-Damgård prefix and uses that to construct a forgery
   ↳ Structure exploited in SHA-1 collision demonstration (can generate arbitrary collisions once prefix matches) → for SHA-1, they used PDF files
3. Envelope method: $F(k,m) := H(k \| m \| k)$
4. Two-key nest: $F((k_1, k_2), m) := H(k_2 \| H(k_1 \| m))$

} for reasonable pseudorandomness assumptions on h (e.g., both $F_1(k,m) := h(k,m)$ and $F_2(k,m) := h(m,k)$ is a PRF), both of these constructions are secure PRFs on a variable-size domain

hash-based MAC

HMAC is a PRF/MAC based on the two-key nest (though with correlated keys):

$HMAC(k, m) := H(k_1 \| H(k_2, m))$

where $k_1 \leftarrow k \oplus ipad$ and $k_2 \leftarrow k \oplus opad$

and ipad and opad are fixed strings (specified in the HMAC standard)

0x36 repeated 0x5C repeated

Security: Since k_1 and k_2 are correlated, need to make stronger assumption on security (e.g., h remains pseudorandom under a related-key attack)

Instantiations: Typically, denoted HMAC-H where H is the hash function

e.g., HMAC-SHA1

HMAC-SHA256 — one of the most widely-used MAC on the web (used in SSL/TLS, IPsec, SSH, and more)

HMAC for key-derivation: Recall that under reasonable assumptions, HMAC is a secure PRF

In many protocols, we need to derive multiple keys from a single master key (e.g., derived from a password)

↳ To derive multiple independent cryptographic keys, a PRF is a natural primitive:

$k_{enc} \leftarrow HMAC(k_{master}, \text{"enc"})$

$k_{mac} \leftarrow HMAC(k_{master}, \text{"mac"})$

} PRF security says derived keys are computationally indistinguishable from uniform

derived keys master key tag (just has to be unique)

This approach is used in TLS and IPsec to derive session keys during session setup

↳ General paradigm is the "expand" step in hash-based key-derivation (HKDF — RFC 5869)

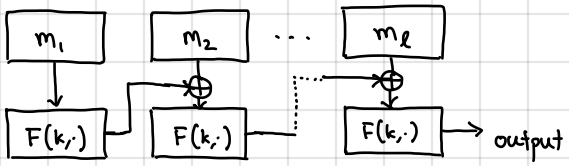
↳ Consists of two procedures:

- Extract: derive a master key from entropy source (e.g., a user password)
- Expand: derive sub-keys from the master key

Both steps rely on HMAC

Another approach to construct MACs: domain extension for PRFs [small-domain PRF ⇒ large-domain PRF]

Approach 1: use CBC (without IV)

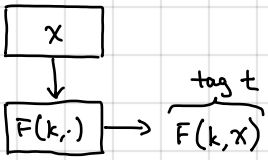


Not encrypting messages so no need for IV (or intermediate blocks)
↳ Mode often called "raw-CBC"

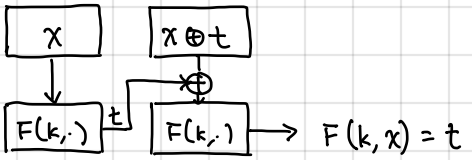
Raw-CBC is a way to build a large-domain PRF from a small-domain one
↳ Can show security for "prefix-free" messages [more precisely, raw-CBC is a prefix-free PRF: pseudorandom as long as PRF never evaluated on two values where one is a prefix of other]
↳ includes fixed-length messages as a special case

But not secure for variable-length messages: "Extension attack"

1. Query for MAC on arbitrary block x:



2. Output forgery on message $(x, x \oplus t)$ and tag t



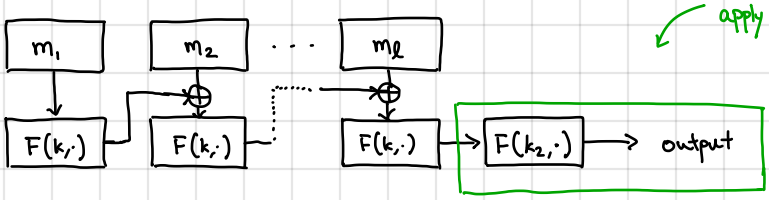
⇒ t is a valid tag on extended message $(x, t \oplus x)$
↳ Adversary succeed with advantage 1

raw CBC can be used to build a MAC on fixed-length messages, but not variable-length messages (more generally, prefix-free)

For variable-length messages, we use "encrypted CBC" (ECBC):
↳ variant used in ANSI X9.9, ANSI X19.9 standards (standards for banking / financial services)

critical for security (using the same key not secure)

apply another PRF with a different key to the output of rawCBC



To use encrypted CBC-MAC, we need to assume message length is even multiple of block size (similar to CBC encryption)
↳ to sign messages that are not a multiple of the block size, we need to first pad the message
↳ as was the case with encryption, padding must be injective
↳ in the case of encryption, injectivity needed for correctness
↳ in the case of integrity, injectivity needed for security [if $pad(m_0) = pad(m_1)$, m_0 and m_1 will have the same tag]

Standard approach to pad: append $1000\cdots0$ to fill up block [ANSI X9.9 and ANSI X9.19 standards]

- Note: if message is an even multiple of the block length, need to introduce a dummy block
  - ↳ Necessary for any injective function: $|\{0,1\}^{\leq n}| > |\{0,1\}^n|$
- This is a bit-padding scheme [PKCS #7 that we discuss previously in the context of CBC encryption is a byte-padding scheme]

Encrypted CBC-MAC drawbacks: always need at least 2 PRF evaluations (using different keys)
messages must be padded to block size
} especially bad for authenticating short (e.g., single-byte) messages

Better approach: raw CBC-MAC secure for prefix-free messages

↳ Can we apply a "prefix-free" encoding to the message?

- Option 1: Prepend the message length to the message ← equal-length messages cannot have one be prefix of other; different-length messages differ in first block
  - Problematic if we do not know message length at the beginning (e.g., in a streaming setting)
  - Still requires padding message to multiple of block size)
- Option 2: Apply a random secret shift to the last block of the message
  - $(x_1, x_2, \ldots, x_\ell) \mapsto (x_1, x_2, \ldots, x_\ell \oplus k)$ where $k \xleftarrow{R} \mathcal{X}$
  - Adversary that does not know k cannot construct two messages that are prefixes except with probability $1/|\mathcal{X}|$ (by guessing k)
  - ↳ basis for CMAC (standardized by NIST in 2005)