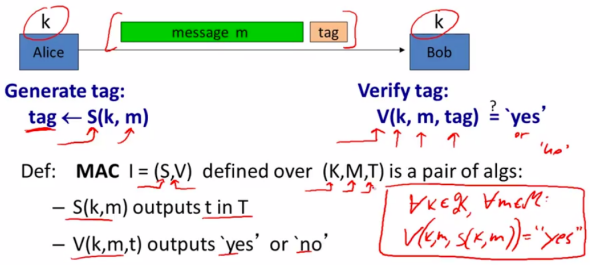
- **Message Integrity 1: definitions**
- **Message Authentication Codes**
- Message Integrity
  - Goal: Integrity, no confidentiality
  - Examples:
    - Protecting public binaries on disk
    - Protecting banner ads on web pages
- Message Integrity: MAC

Message integrity: MACs



  - tag 90 bits or 100 bits
- Integrity requires a secret key

Integrity requires a secret key

- Attacker can easily modify message m and re-compute CRC.
- CRC designed to detect **random**, not malicious errors.

  - CRC - cyclic redundancy check
- Secure MACs

Secure MACs

Attacker's power: **chosen message attack**

- for $m_1, m_2, \ldots, m_q$ attacker is given $t_i \leftarrow S(k, m_i)$

Attacker's goal: **existential forgery**

- produce some **new** valid message/tag pair (m,t).

$(m,t) \notin \{ (m_1,t_1), \ldots, (m_q,t_q) \}$

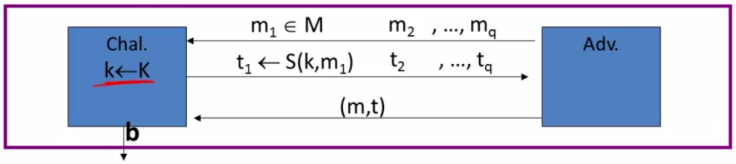
⇒ attacker cannot produce a valid tag for a new message

⇒ given (m,t), attacker cannot even produce (m,t') for $t' \neq t$

- Secure MACs

Secure MACs

- For a MAC I=(S,V) and adv. A define a MAC game as:



b=1 if V(k,m,t) = 'yes' and $(m,t) \notin \{(m_1,t_1), \ldots, (m_q,t_q)\}$

b=0 otherwise

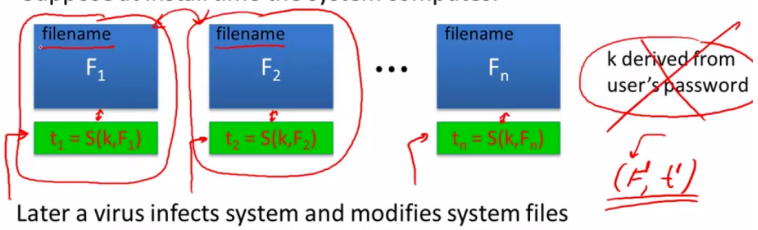
Def: I=(S,V) is a **secure MAC** if for all "efficient" A:

$Adv_{MAC}[A,I]$ = Pr[Chal. outputs 1] is "negligible."

- Example: protecting system files

Example: protecting system files

Suppose at install time the system computes:



Later a virus infects system and modifies system files

User reboots into clean OS and supplies his password

– Then: secure MAC ⇒ all modified files will be detected

- **MACs Based On PRFs**
- Review: Secure MACs

Review: Secure MACs

MAC: signing alg. **S(k,m)→t** and verification alg. **V(k,m,t) →0,1**

Attacker's power: **chosen message attack**

- for $m_1,m_2,\ldots,m_q$ attacker is given $t_i \leftarrow S(k,m_i)$

Attacker's goal: **existential forgery**

- produce some **new** valid message/tag pair (m,t).

$(m,t) \notin \{(m_1,t_1), \ldots, (m_q,t_q)\}$

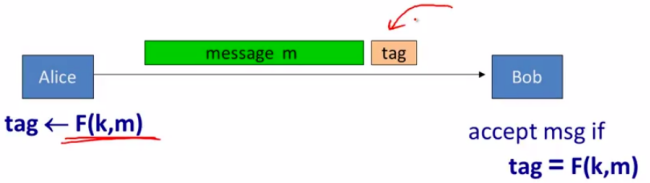
⇒ attacker cannot produce a valid tag for a new message

- Secure PRF => Secure MAC

Secure PRF ⇒ Secure MAC

For a PRF $F: K \times X \rightarrow Y$ define a MAC $I_F = (S,V)$ as:

- $S(k,m) := F(k,m)$
- $V(k,m,t)$: output 'yes' if $t = F(k,m)$ and 'no' otherwise.



- Security

Security

Thm: If $F: K \times X \rightarrow Y$ is a secure PRF and $1/|Y|$ is negligible (i.e. $|Y|$ is large) then I_F is a secure MAC.

In particular, for every eff. MAC adversary A attacking I_F there exists an eff. PRF adversary B attacking F s.t.:

$$Adv_{MAC}[A, I_F] \leq Adv_{PRF}[B, F] + 1/|Y|$$

neg. neg. neg.

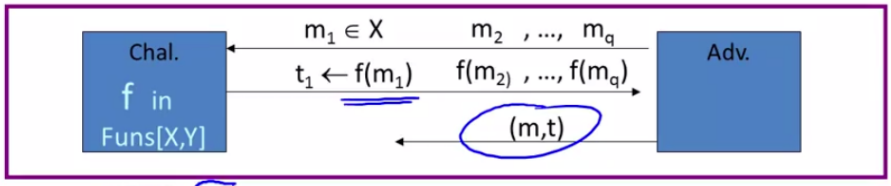
⇒ I_F is secure as long as $|Y|$ is large, say $|Y| = 2^{80}$.

- Proof Sketch

Proof Sketch

Suppose $f: X \rightarrow Y$ is a truly random function

Then MAC adversary A must win the following game:



A wins if $t = f(m)$ and $m \notin \{m_1, \ldots, m_q\}$

⇒ Pr[A wins] = 1/|Y| same must hold for F(k,x)

Dan Bo

- Examples

Examples

- AES: a MAC for 16-byte messages.
- Main question: how to convert Small-MAC into a Big-MAC ?
- Two main constructions used in practice:
  - **CBC-MAC** (banking – ANSI X9.9, X9.19, FIPS 186-3)
  - **HMAC** (Internet protocols: SSL, IPsec, SSH, …)
- Both convert a small-PRF into a big-PRF.

- Truncating MACs based on PRFs

Truncating MACs based on PRFs

Easy lemma: suppose $F: K \times X \rightarrow \{0,1\}^n$ is a secure PRF.

Then so is $F_t(k,m) = F(k,m)[1\ldots t]$ for all $1 \le t \le n$

output the first t bits

$\Rightarrow$ if (S,V) is a MAC is based on a secure PRF outputting n-bit tags

the truncated MAC outputting w bits is secure

… as long as $1/2^w$ is still negligible (say $w \ge 64$)

- MACs and PRFs

MACs and PRFs

Recall: secure PRF **F** $\Rightarrow$ secure MAC, as long as |Y| is large

S(k, m) = F(k, m)

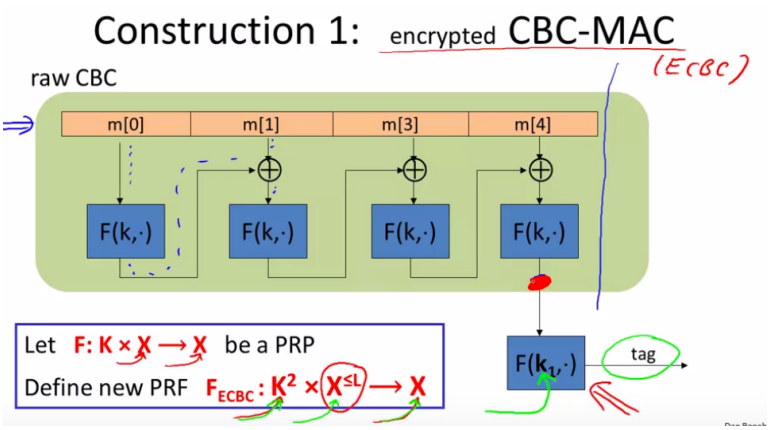
Our goal:

given a PRF for short messages (AES)

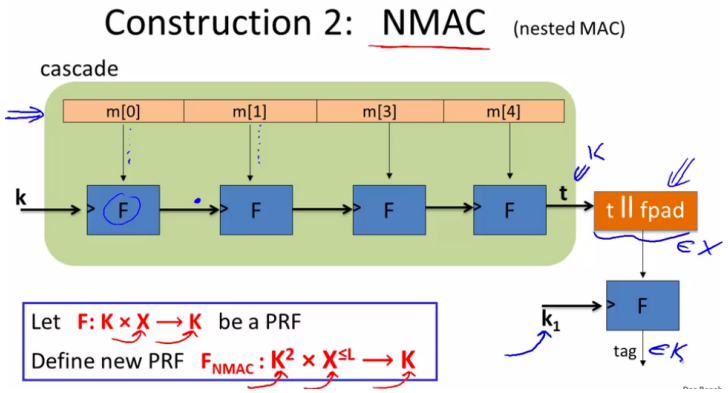
construct a PRF for long messages

From here on let $X = \{0,1\}^n$ (e.g. n=128)

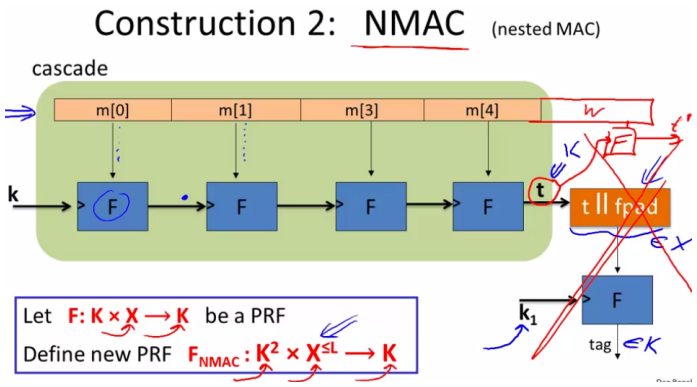
- Construction 1: ECBC-MAC



- Construction 2: NMAC



  - Not secure if only cascade



  - extension attack are the attacks on cascade
- Why the last encryption step in ECBC-MAC?

Suppose we define a MAC $I_{RAW} = (S,V)$ where

$S(k,m) = \text{rawCBC}(k,m)$

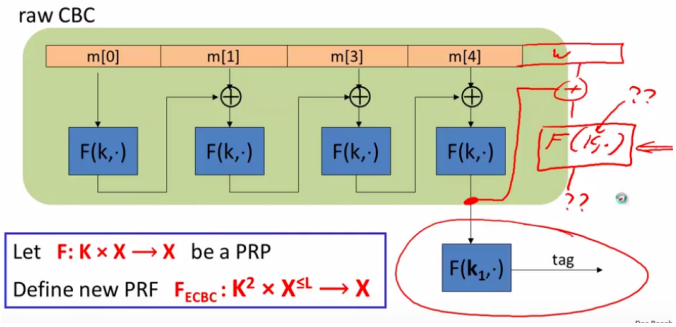
Then I_{RAW} is easily broken using a 1-chosen msg attack.

Adversary works as follows:

- Choose an arbitrary one-block message $m \in X$
- Request tag for m. Get $t = F(k,m)$
- Output t as MAC forgery for the 2-block message $(m,\ t \oplus m)$

Indeed: $\text{rawCBC}(k, (m,\ t \oplus m)) = F(k, F(k,m) \oplus (t \oplus m)) = F(k, t \oplus (t \oplus m)) = t$

- Construction 1: ECBC-MAC



- ECBE-MAC and NMAC analysis

Theorem: For any L>0,

For every eff. q-query PRF adv. A attacking F_{ECBC} or F_{NMAC} there exists an eff. adversary B s.t.:

$$\text{Adv}_{PRF}[A, F_{ECBC}] \le \text{Adv}_{PRP}[B, F] + 2\,q^2 / |X|$$

$$\text{Adv}_{PRF}[A, F_{NMAC}] \le q \cdot L \cdot \text{Adv}_{PRF}[B, F] + q^2 / 2|K|$$

CBC-MAC is secure as long as $q << |X|^{1/2}$

NMAC is secure as long as $q << |K|^{1/2}$ (2^{64} for AES-128)

- An example

An example

$$\text{Adv}_{PRF}[A, F_{ECBC}] \le \text{Adv}_{PRP}[B, F] + q^2 / |X|$$

q = # messages MAC-ed with k

Suppose we want $\text{Adv}_{PRF}[A, F_{ECBC}] \le 1/2^{32}$ $\Leftarrow$ $q^2 / |X| < 1/2^{32}$

- AES: $|X| = 2^{128}$ $\Rightarrow$ $q < 2^{48}$

  So, after 2^{48} messages must, must change key

- 3DES: $|X| = 2^{64}$ $\Rightarrow$ $q < 2^{16}$

- The security bounds are tight: an attack

After signing $|X|^{1/2}$ messages with ECBC-MAC or
$|K|^{1/2}$ messages with NMAC
the MACs become insecure

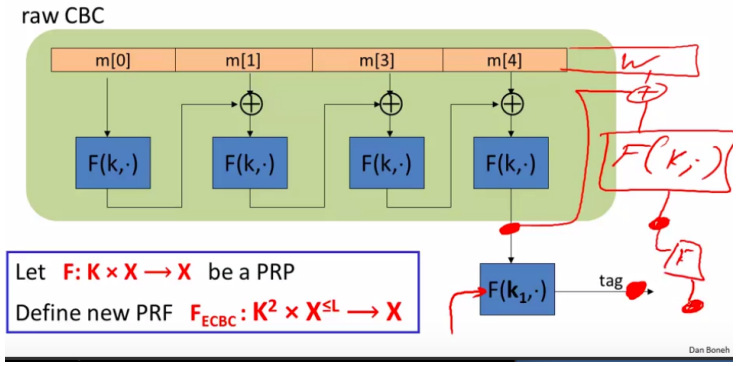
Suppose the underlying PRF F is a PRP (e.g. AES)

- Then both PRFs (ECBC and NMAC) have the following extension property:

$\forall x,y,w$: $F_{BIG}(k, x) = F_{BIG}(k, y) \Rightarrow F_{BIG}(k, \mathbf{xllw}) = F_{BIG}(k, \mathbf{yllw})$

- if two messages are the same after the output of the raw CBC function then adding another block will be computing the same value and when encrypting will get the same output and final output. If two values are the same for two distinct messages if appending w will get the same output for both of the messages.

- Construction 1: ECBC-MAC



- The security bounds are tight: an attack

Let F_{BIG}: $K \times X \rightarrow Y$ be a PRF that has the extension property

$F_{BIG}(k, x) = F_{BIG}(k, y) \Rightarrow F_{BIG}(k, \mathbf{xllw}) = F_{BIG}(k, \mathbf{yllw})$

Generic attack on the derived MAC:

step 1: issue $|Y|^{1/2}$ message queries for rand. messages in X.
obtain (m_i, t_i) for $i = 1, \ldots, |Y|^{1/2}$

step 2: find a collision $t_u = t_v$ for $u \neq v$ (one exists w.h.p by b-day paradox)

step 3: choose some w and query for $t := F_{BIG}(k, \mathbf{m_u llw})$

step 4: output forgery $(\mathbf{m_v llw}, \mathbf{t})$. Indeed $t := F_{BIG}(k, \mathbf{m_v llw})$

Dan Boneh

- Comparison

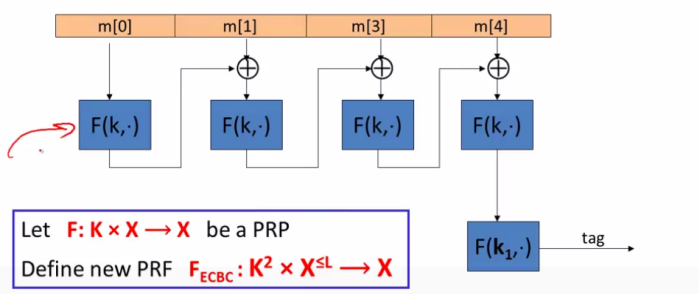
ECBC-MAC is commonly used as an AES-based MAC

- CCM encryption mode (used in 802.11i)
- NIST standard called CMAC

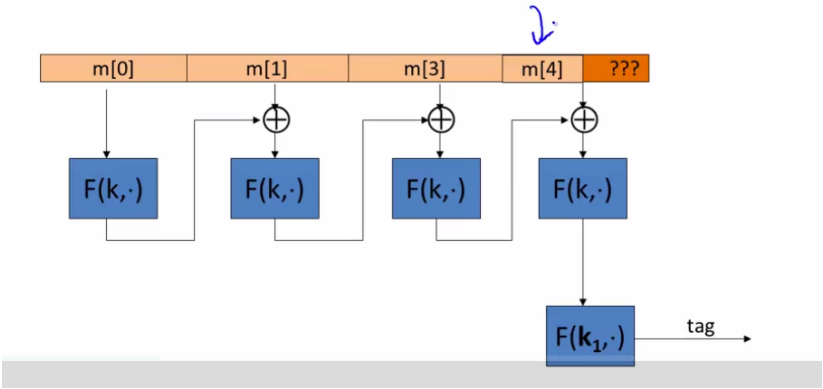
NMAC not usually used with AES or 3DES

- Main reason: need to change AES key on every block
  requires re-computing AES key expansion
- But NMAC is the basis for a popular MAC called HMAC (next)

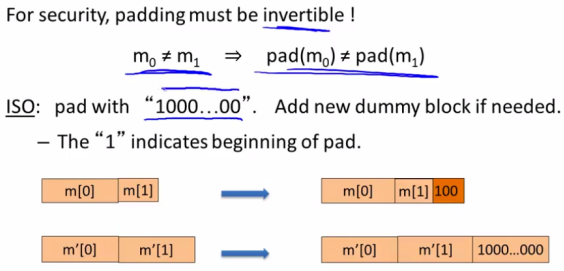
- **MAC Padding**
- Recall: ECBC-MAC



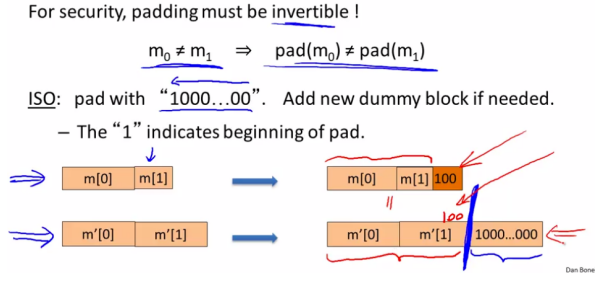
- What is msg. len. is not multiple of block size?



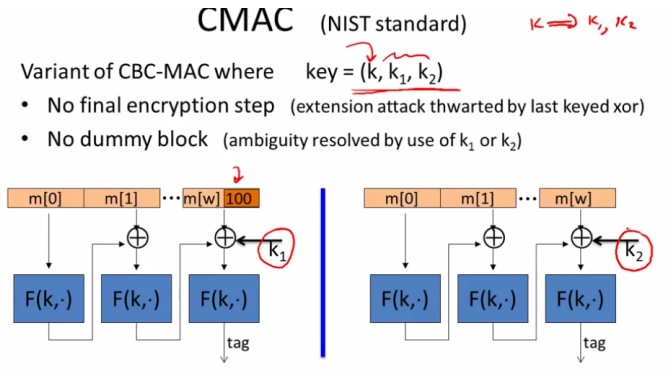
- CBC MAC Padding
  - padding function must be invertible that guarantees that the padding function is one to one



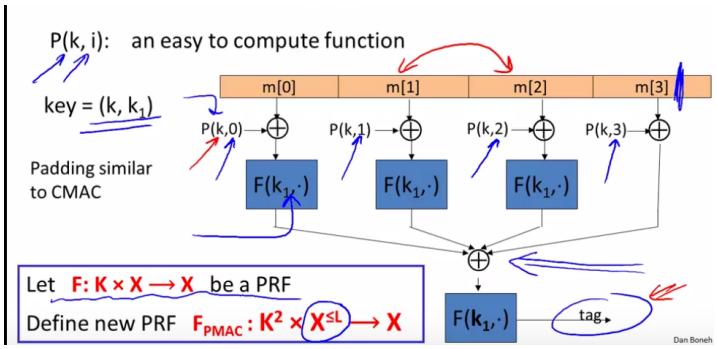
  - scan the message from right to left when reach first one remove the padding
  - have to add a dummy block if the message is a multiple of the block length to begin with



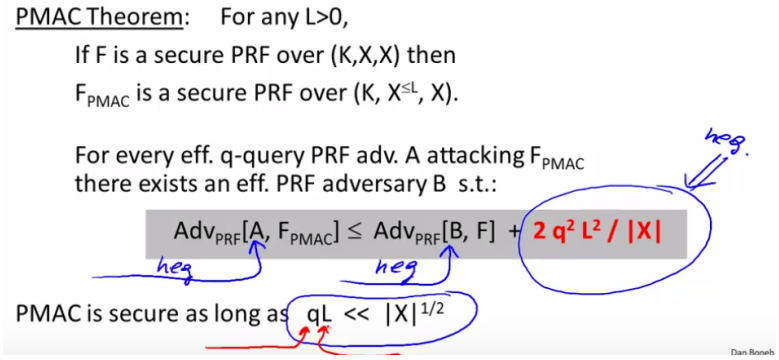
- CMAC (NIST standard)
  - impossible to do extension attack because the advisory does not know the last block that went into the function



- **Message Integrity 3: more constructions**
- **PMAC and the Carter-Wegman MAC**
- Construction 3: PMAC - parallel MAC
  - P is blocking the block swapping attack



- PMAC: Analysis

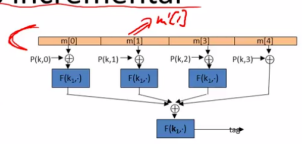


- PMAC is incremental

PMAC is incremental

Suppose F is a PRP.

When m[1] ⟶ m'[1]
can we quickly update tag?



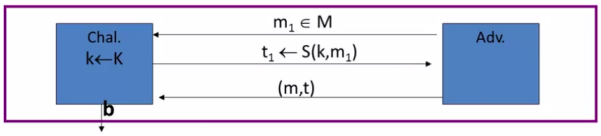
- ○ no, it can't be done
- ○ do $F^{-1}(k_1,tag) \oplus F(k_1, m'[1] \oplus P(k,1))$
- ○ do $F^{-1}(k_1,tag) \oplus F(k_1, m[1] \oplus P(k,1)) \oplus F(k_1, m'[1] \oplus P(k,1))$
- ○ do $tag \oplus F(k_1, m[1] \oplus P(k,1)) \oplus F(k_1, m'[1] \oplus P(k,1))$

Then apply $F(k_1, \cdot)$

- One time MAC (analog of one time pad)

One time MAC (analog of one time pad)

- For a MAC I=(S,V) and adv. A define a MAC game as:



b=1 if V(k,m,t) = 'yes' and $(m,t) \neq (m_1,t_1)$
b=0 otherwise

Def: I=(S,V) is a **secure MAC** if for all "efficient" A:

$Adv_{MAC}[A,I]$ = Pr[Chal. outputs 1] is "negligible."

- One-time MAC: an example
  - pick a prime number that is larger than the block size that is being used.

Can be secure against **all** adversaries and faster than PRF-based MACs

Let q be a large prime (e.g. $q = 2^{128}+51$)

key = (k, a) ∈ $\{1,\ldots,q\}^2$ (two random ints. in [1,q])

msg = (m[1], …, m[L]) where each block is 128 bit int.

$S(key, msg) = P_{msg}(k) + a \pmod q$

where $P_{msg}(x) = m[L]\cdot x^L + \ldots + m[1]\cdot x$ is a poly. of deg L.

Fact: given $S(key, msg_1)$ adv. has no info about $S(key, msg_2)$

- One-time MAC -> Many-time MAC

Let (S,V) be a secure one-time MAC over $(K_I, M, \{0,1\}^n)$.

Let $F: K_F \times \{0,1\}^n \longrightarrow \{0,1\}^n$ be a secure PRF.

Carter-Wegman MAC: $CW((k_1,k_2), m) = (r, F(k_1,r) \oplus S(k_2,m))$

(slow but short inp; fast long inp)

for random $r \leftarrow \{0,1\}^n$.

Thm: If (S,V) is a secure **one-time** MAC and F a secure PRF
then CW is a secure MAC outputting tags in $\{0,1\}^{2n}$.

- Construction 4: HMAC (Hash-MAC)
- **Collision Resistance 1: what is a collision resistant function?**
- **Introduction**
- Recap: message integrity
  - secure if an attacker is given a tag an arbitrary message of choice cannot construct a tag for a new message

So far, four MAC constructions:

PRFs
- **ECBC-MAC, CMAC** : commonly used with AES (e.g. 802.11i)
- **NMAC** : basis of HMAC (this segment)
- **PMAC**: a parallel MAC

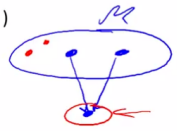
randomized MAC
- **Carter-Wegman MAC**: built from a fast one-time MAC

- Collision Resistance

Let $H: M \to T$ be a hash function ($|M| >> |T|$)

A **collision** for H is a pair $m_0, m_1 \in M$ such that:

$H(m_0) = H(m_1)$ and $m_0 \neq m_1$



A function H is **collision resistant** if for all (explicit) "eff" algs. A:

$\mathbf{Adv_{CR}[A,H] = Pr[\ A\ outputs\ collision\ for\ H]}$

is "neg".

Example: SHA-256 (outputs 256 bits)

Dan Boneh

- MACs from Collision Resistance

Let I = (S,V) be a MAC for short messages over (K,M,T) (e.g. AES)

Let $H: M^{big} \to M$

Def: $I^{big} = (S^{big}, V^{big})$ over (K, M^{big}, T) as:

$\mathbf{S^{big}(k,m) = S(k,H(m))}$; $\mathbf{V^{big}(k,m,t) = V(k,H(m),t)}$

Thm: If I is a secure MAC and H is collision resistant

then I^{big} is a secure MAC.

Example: $S(k,m) = AES_{2\text{-block-cbc}}(k, SHA\text{-}256(m))$ is a secure MAC.

Dan Boneh

$\mathbf{S^{big}(k, m) = S(k, H(m))}$; $\mathbf{V^{big}(k, m, t) = V(k, H(m), t)}$

Collision resistance is necessary for security:

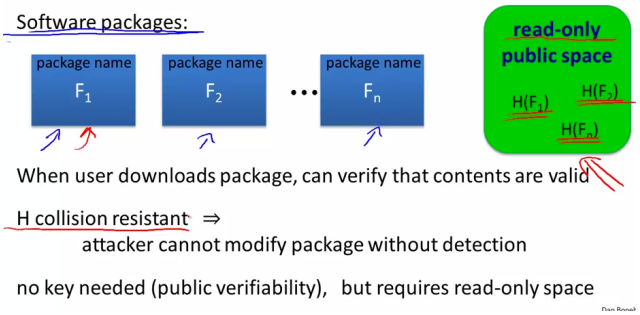
Suppose adversary can find $m_0 \neq m_1$ s.t. $H(m_0) = H(m_1)$.

Then: S^{big} is insecure under a 1-chosen msg attack

step 1: adversary asks for $t \leftarrow S(k, m_0)$

step 2: output (m_1, t) as forgery

- Protecting file integrity using C.R. hash



  - The value of his hash will not map to the value in the public space
- **Generic Birthday Attack**
- Generic attack on C.R. functions

Let $H: M \to \{0,1\}^n$ be a hash function ($|M| >> 2^n$)

Generic alg. to find a collision **in time** $O(2^{n/2})$ hashes

Algorithm:
1. Choose $2^{n/2}$ random messages in M: $m_1, \ldots, m_{2^{n/2}}$ (distinct w.h.p)
2. For i = 1, …, $2^{n/2}$ compute $t_i = H(m_i) \in \{0,1\}^n$
3. Look for a collision $(t_i = t_j)$. If not found, got back to step 1.

How well will this work?

- The birthday paradox

Let $r_1, \ldots, r_n \in \{1,\ldots,B\}$ be indep. identically distributed integers.

Thm: when $n = 1.2 \times B^{1/2}$ then $\Pr[\exists i \neq j: r_i = r_j] \geq \frac{1}{2}$

Proof: (for uniform indep. $r_1, \ldots, r_n$)

$$\Pr[\exists i \neq j: r_i = r_j] = 1 - \Pr[\forall i \neq j: r_i \neq r_j] = 1 - \left(\frac{B-1}{B}\right)\left(\frac{B-2}{B}\right)\cdots\left(\frac{B-n+1}{B}\right) =$$

$$= 1 - \prod_{i=1}^{n-1}\left(1 - \frac{i}{B}\right) \geq 1 - \prod_{i=1}^{n-1} e^{-i/B} = 1 - e^{-\frac{1}{B}\sum_{i=1}^{n-1} i} \geq 1 - e^{-n^2/2B}$$

$1 - x \leq e^{-x} = 1 - x + \frac{x^2}{2} - \cdots$

$$= 1 - e^{-0.72} = 0.53 > \frac{1}{2}$$

- Generic attack

$H: M \to \{0,1\}^n$. Collision finding algorithm:
1. Choose $2^{n/2}$ random elements in M: $m_1, \ldots, m_{2^{n/2}}$
2. For i = 1, …, $2^{n/2}$ compute $t_i = H(m_i) \in \{0,1\}^n$
3. Look for a collision $(t_i = t_j)$. If not found, got back to step 1.

Expected number of iteration ≈ 2

Running time: $O(2^{n/2})$ (space $O(2^{n/2})$)

- Sample C.R. hash functions

AMD Opteron, 2.2 GHz (Linux)

	function	digest size (bits)	Speed (MB/sec)	generic attack time
NIST standards	SHA-1	160	153	2^{80}
	SHA-256	256	111	2^{128}
	SHA-512	512	99	2^{256}
	Whirlpool	512	57	2^{256}

- **Collision Resistance 2: Constructions**
- **Collision resistance: review**

Let $H: M \rightarrow T$ be a hash function ($|M| >> |T|$)

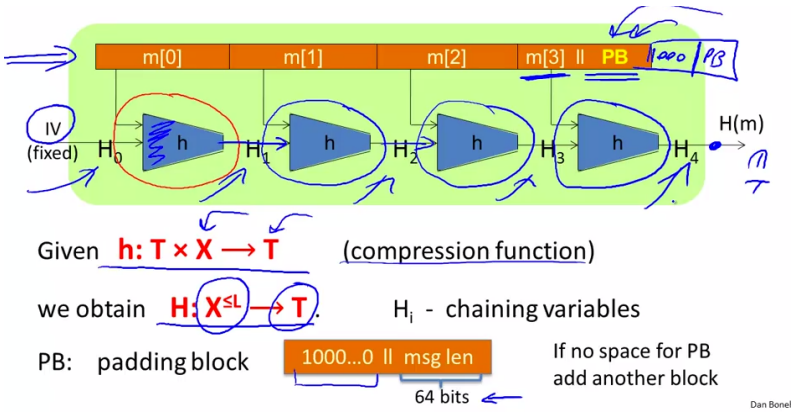
A **collision** for H is a pair $m_0, m_1 \in M$ such that:

$H(m_0) = H(m_1)$ and $m_0 \neq m_1$

Goal: collision resistant (C.R.) hash functions

Step 1: given C.R. function for **short** messages,
construct C.R. function for **long** messages

- The Merkle-Damgard iterated construction



- MD collision resistance
  - candidate collision for the compression function
  - arguments have to be distinct if the arguments are the same then don't have a collision for the compression function

Thm: if h is collision resistant then so is H.

Proof: collision on H ⇒ collision on h

Suppose $H(M) = H(M')$. We build collision for h.

$IV = H_0 \;,\; H_1 \;, \ldots, \; H_t \;,\; H_{t+1} = H(M)$

$IV = H_0' \;,\; H_1' \;, \ldots, \; H_r' \;,\; H_{r+1}' = H(M')$

$h(H_t, M_t \| PB) = H_{t+1} = H'_{r+1} = h(H'_r, M'_r \| PB')$

$H_t \neq H'_r$ or $M_t \neq M'_r$ or $PB \neq PB'$ ⇒ coll. for h

Dan Bonel

Suppose $H_t = H'_r$ and $M_t = M'_r$ and $PB = PB'$ → $t = r$

Then: $h(H_{t-1}, M_{t-1}) = H_t = H'_t = h(H'_{t-1}, M'_{t-1})$

$H_{t-1} \neq H'_{t-1}$ or $M_{t-1} \neq M'_{t-1}$ ⇒ coll. for h

Suppose $H_{t-1} = H'_{t-1}$ and $M_t = M'_t$ and $M_{t-1} = M'_{t-1}$

Iterate to begin of msg:

(1) find coll. for h, or

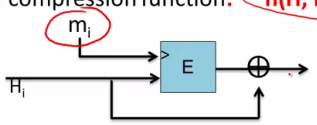
(2) $\forall i: M_i = M'_i \Rightarrow M = M'$

- If little function is collision resistance then the big function is collision resistance

- To construct collision resistance function, suffices to construct compression function
- **Constructing Compression Functions**
- Compression function from a block cipher

$E: K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ a block cipher.

The **Davies-Meyer** compression function: $h(H, m) = E(m, H) \oplus H$



Thm: Suppose E is an ideal cipher (collection of |K| random perms.).
Finding a collision $h(H,m)=h(H',m')$ takes $O(2^{n/2})$ evaluations of (E,D).

Best possible !!

Dan Bone

Suppose we define $h(H, m) = E(m, H)$

Then the resulting h(.,.) is not collision resistant:

to build a collision (H,m) and (H',m')

choose random (H,m,m') and construct H' as follows:

- ⇒ ○ H'=D(m', E(m,H)) ⇒ $E(m', H') = E(m, H)$
- ○ H'=E(m', D(m,H))
- ○ H'=E(m', E(m,H))
- ○ H'=D(m', D(m,H))

- Other block cipher constructions

Let $E: \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}^n$ for simplicity

Miyaguchi-Preneel: $h(H, m) = E(m, H) \oplus H \oplus m$ **(Whirlpool)**

$h(H, m) = E(H \oplus m, m) \oplus m$

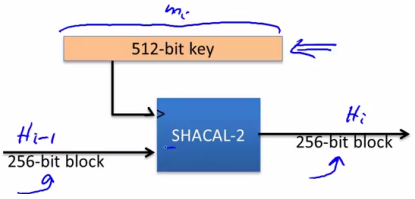
total of 12 variants like this

Other natural variants are insecure:

$h(H, m) = E(m, H) \oplus m$ (HW)

- Case study: SHA-256

- Merkle-Damgard function
- Davies-Meyer compression function
- Block cipher: SHACAL-2



- Provable compression functions

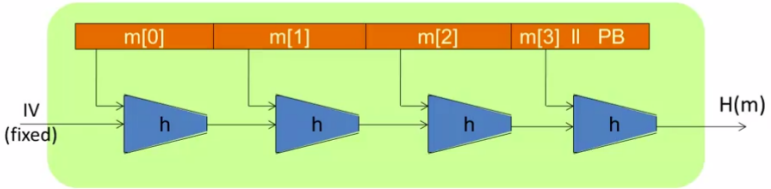
Choose a random 2000-bit prime p and random $1 \le u, v \le p$.

For $m,h \in \{0,\ldots,p-1\}$ define $h(H,m) = u^H \cdot v^m \pmod p$

Fact: finding collision for h(.,.) is as hard as solving "discrete-log" modulo p.

Problem: slow.

- **HMAC: a MAC from a hash function**
- **HMAC**
- The Merkle-Damgard iterated construction



Thm: h collision resistant ⇒ H collision resistant

Can we use H(.) to directly build a MAC?

- MAC from a Merkle-Damgard Hash Function

$H: X^{\leq L} \rightarrow T$ a C.R. Merkle-Damgard Hash Function

Attempt #1: $S(k, m) = H(k \| m)$

This MAC is insecure because:

- ○ Given $H(k \| m)$ can compute $H(w \| k \| m \| PB)$ for any w.
- ○ Given $H(k \| m)$ can compute $H(k \| m \| w)$ for any w.
- ○ Given $H(k \| m)$ can compute $H(k \| m \| PB \| w)$ for any w.
- ○ Anyone can compute $H(k \| m)$ for any m.

- Standardized method: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

H: hash function.
example: SHA-256 ; output is 256 bits

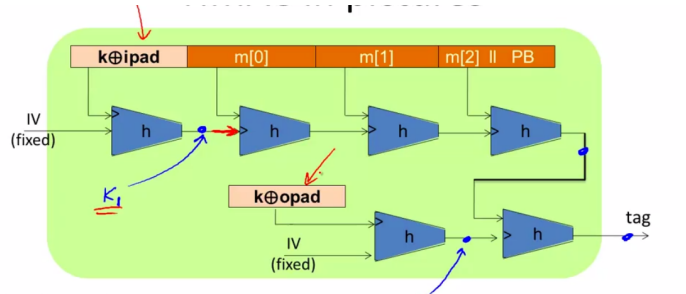
Building a MAC out of a hash function:

HMAC: $S(k, m) = H(k \oplus opad, H(k \oplus ipad \| m))$

512-bits 512-bits

Dan Boneh

- HMAC in pictures



Similar to the NMAC PRF.

main difference: the two keys k_1, k_2 are dependent

- HMAC properties

HMAC is assumed to be a secure PRF

- Can be proven under certain PRF assumptions about h(.,.)
- Security bounds similar to NMAC
  - Need $q^2/|T|$ to be negligible ($q << |T|^{1/2}$)

In TLS: must support HMAC-SHA1-96

- **Timing attacks on MAC verification**
- Warning: verification timing attacks

Example: Keyczar crypto library (Python) [simplified]

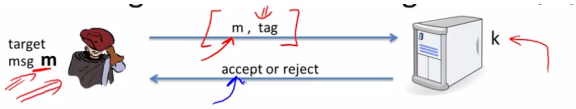
```
def Verify(key, msg, sig_bytes):
    return HMAC(key, msg) == sig_bytes
```

16-bytes

The problem: '==' implemented as a byte-by-byte comparison

- Comparator returns false when first inequality found

- Warning: verification timing attacks

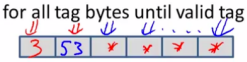


Timing attack: to compute tag for target message m do:

Step 1: Query server with random tag

Step 2: Loop over all possible first bytes and query server.
stop when verification takes a little longer than in step 1

Step 3: repeat for all tag bytes until valid tag found



- Defense #1

Make string comparator always take same time (Python) :

```
return false if sig_bytes has wrong length
result = 0
for x, y in zip( HMAC(key,msg) , sig_bytes):
    result |= ord(x) ^ ord(y)
return result == 0
```

16 16

Can be difficult to ensure due to optimizing compiler.

- Defense #2
  - doesn't know what strings are being compared

Make string comparator always take same time (Python) :

```
def Verify(key, msg, sig_bytes):
    mac = HMAC(key, msg)
    return HMAC(key, mac) == HMAC(key, sig_bytes)
```

Attacker doesn't know values being compared