CSE 433S: Introduction to Computer Security

Message Integrity
- Message Authentication Code
- Hash Functions

Washington University in St. Louis

Slides contain content from Professor Dan Boneh at Stanford University

Review Questions

- How is block cipher different from stream cipher, how is it similar to stream cipher?
- What are PRP and PRF, what constructions will allow one to construction a PRP from PRF?
- What are the four key design principles of block cipher?
- What the root cause behind the vulnerability in ECB mode of AES?
- What are the two approaches we studied in class to address the problem of one-time-key?
- What are the requirements for IVs in block cipher modes of operation?
- T/F questions
  - DES is still secure
  - The key length of block cipher need to be the same as the length of the block
  - When the file is not a multiple of blocksize, we pad it with random bytes to secure it, since the goal is to have the output as random as possible
  - The entries in the S-box has to be non-linear, therefore we just randomly generate it

Common Security Goals

- Confidentiality
- Integrity
- Availability

How would you break AES CTR mode ?

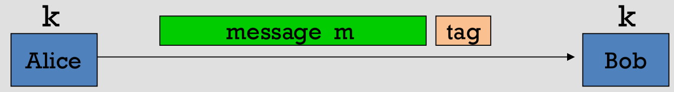
How would you break OTP?

Message Integrity

Goal: **Integrity**, no confidentiality.

Examples:
- Protecting public binaries on disk.
- Protecting banner ads on web pages.

Message integrity: MACs



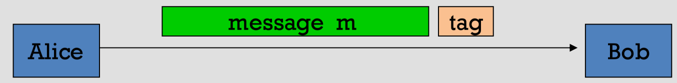
Generate tag:
tag ← S(k, m)

Verify tag:
V(k, m, tag) $\stackrel{?}{=}$ `yes'

Def: **MAC** I = (S,V) defined over (K,M,T) is a pair of algs:

- S(k,m) outputs t in T
- V(k,m,t) outputs `yes' or `no'

Integrity requires a secret key



Generate tag:
tag ← CRC(m)

Verify tag:
V(m, tag) $\stackrel{?}{=}$ `yes'

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

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).

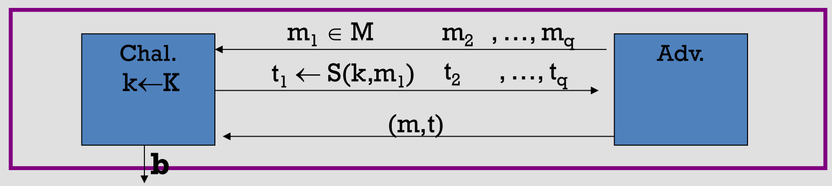
$(m,t) \notin \{ (m_1,t_1), \ldots, (m_q,t_q) \}$

⇒ 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

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



$$\begin{cases} \mathbf{b}=1 & \text{if } V(k,m,t) = \text{`yes'} \text{ and } (m,t) \notin \{ (m_1,t_1), \ldots, (m_q,t_q) \} \\ \mathbf{b}=0 & \text{otherwise} \end{cases}$$

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

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

Let $I = (S,V)$ be a MAC.

Suppose an attacker is able to find $m_0 \neq m_1$ such that

$S(k, m_0) = S(k, m_1)$ for ½ of the keys k in K

Can this MAC be secure?

Yes, the attacker cannot generate a valid tag for m_0 or m_1

No, this MAC can be broken using a chosen msg attack

It depends on the details of the MAC

Let $I = (S,V)$ be a MAC.

Suppose $S(k,m)$ is always 5 bits long

Can this MAC be secure?

No, an attacker can simply guess the tag for messages

It depends on the details of the MAC

Yes, the attacker cannot generate a valid tag for any message

Example: protecting system files

Suppose at install time the system computes:

filename F_1	filename F_2	…	filename F_n	k derived from user's password
$t_1 = S(k,F_1)$	$t_2 = S(k,F_2)$		$t_n = S(k,F_n)$	

Later a virus infects system and modifies system files

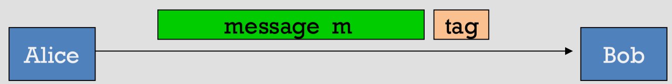
User reboots into clean OS (from external media) and supplies his password

- Then: secure MAC $\Rightarrow$ all modified files will be detected

Secure PRF $\Rightarrow$ Secure MAC

For a PRF $F: K \times X \longrightarrow Y$ 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.



tag ← F(k,m)

accept msg if **tag = F(k,m)**

A bad example

Suppose **$F: K \times X \longrightarrow Y$** is a secure PRF with **$Y = \{0,1\}^{10}$**

Is the derived MAC I_F a secure MAC system?

Yes, the MAC is secure because the PRF is secure

No tags are too short: anyone can guess the tag for any msg

It depends on the function F

Security

Thm: If $\mathbf{F: K \times X \longrightarrow Y}$ 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.:

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

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

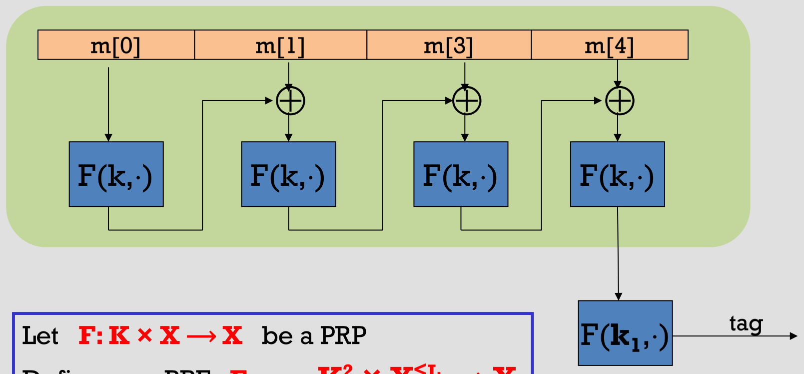
Examples

- AES: a MAC for 16-byte messages.
- Main question: how to convert Small-message MAC into a Big-message-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.

CBC-MAC

Construction 1: encrypted CBC-MAC

raw CBC

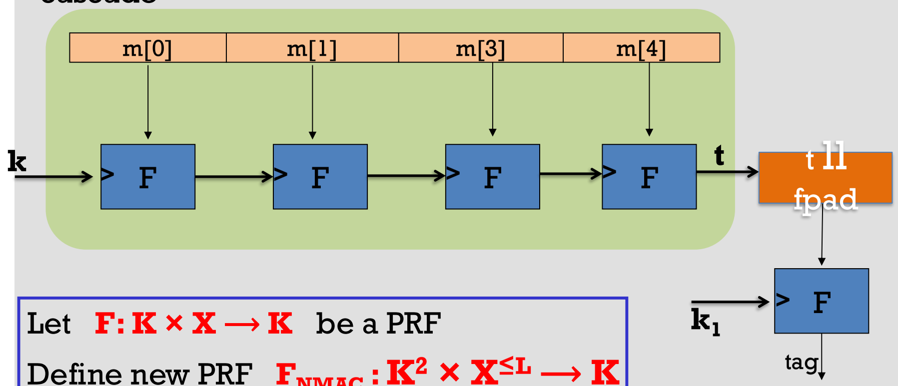


Let $\mathbf{F}: \mathbf{K} \times \mathbf{X} \longrightarrow \mathbf{X}$ be a PRP

Define new PRF $\mathbf{F}_{\mathbf{ECBC}}: \mathbf{K}^2 \times \mathbf{X}^{\leq L} \longrightarrow \mathbf{X}$

Construction 2: NMAC (nested MAC)

cascade



Let $\mathbf{F}: \mathbf{K} \times \mathbf{X} \longrightarrow \mathbf{K}$ be a PRF

Define new PRF $\mathbf{F}_{\mathbf{NMAC}}: \mathbf{K}^2 \times \mathbf{X}^{\leq L} \longrightarrow \mathbf{K}$

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)

Construction 3: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

… but, we first we need to discuss hash function.

Further reading

- J. Black, P. Rogaway: CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. J. Cryptology 18(2): 111-131 (2005)
- K. Pietrzak: A Tight Bound for EMAC. ICALP (2) 2006: 168-179
- J. Black, P. Rogaway: A Block-Cipher Mode of Operation for Parallelizable Message Authentication. EUROCRYPT 2002: 384-397
- M. Bellare: New Proofs for NMAC and HMAC: Security Without Collision-Resistance. CRYPTO 2006: 602-619
- Y. Dodis, K. Pietrzak, P. Puniya: A New Mode of Operation for Block Ciphers and Length-Preserving MACs. EUROCRYPT 2008: 198-219

Hash Functions

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:

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

is "neg".

Example: SHA-256 (outputs 256 bits)

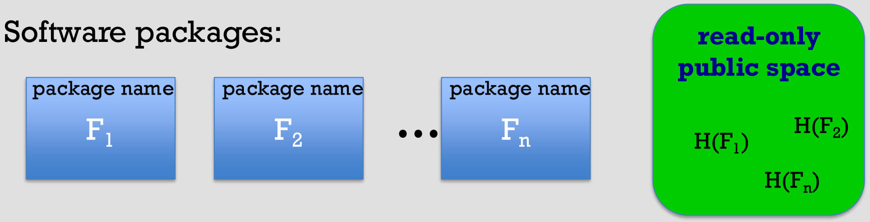
Security Requirements for *Cryptographic* Hash Functions

Given a function $h:X \to Y$, then we say that h is:

- Preimage resistant (one-way):
  if given $y \in Y$ it is computationally infeasible to find a value $x \in X$ s.t. $h(x) = y$
- 2-nd preimage resistant (weak collision resistant):
  if given $x \in X$ it is computationally infeasible to find a value $x' \in X$, s.t. $x' \neq x$ and $h(x') = h(x)$
- Collision resistant (strong collision resistant):
  if it is computationally infeasible to find two distinct values $x',x \in X$, s.t. $h(x') = h(x)$

Protecting file integrity using C.R. hash

Software packages:



When user downloads package, can verify that contents are valid

H collision resistant ⇒
attacker cannot modify package without detection

no key needed (public verifiability), but requires read-only space

Sample C.R. hash functions: Crypto++ 5.6.0 [Wei Dai]

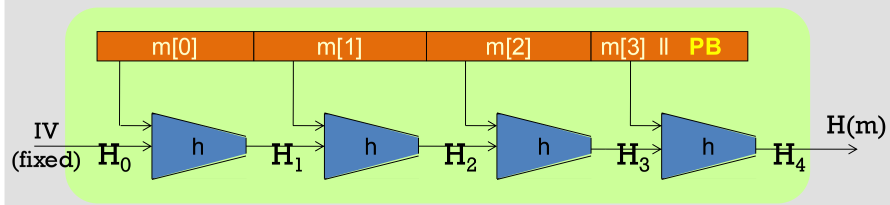
AMD Opteron, 2.2 GHz (Linux)

	function	digest size (bits)	generic Speed (MB/sec)	attack time
	MD5	128	*(Completely broken in 2004)*	
NIST standards	SHA-1	160	153	2^{80}
NIST standards	SHA-256	256	111	2^{128}
NIST standards	SHA-512	512	99	2^{256}
	Whirlpool	512	57	2^{256}

Google already found collision of SHA-1

Now we know the key properties as well as the application of hash function, what are the key internals?

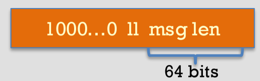
The Merkle-Damgard iterated construction



Given $h: T \times X \longrightarrow T$ (compression function)

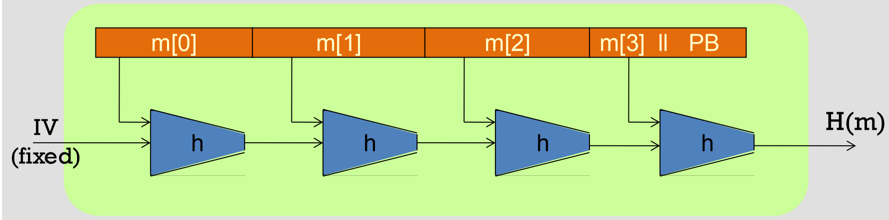
we obtain $H: X^{\leq L} \longrightarrow T$. H_i - chaining variables

PB: padding block



If no space for PB
add another block

The Merkle-Damgard iterated construction



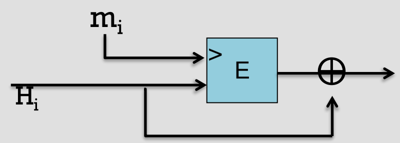
Thm: h collision resistant ⇒ H collision resistant

Goal: construct compression function **h: T × X ⟶ T**

Compr. func. from a block cipher

E: K× {0,1}ⁿ ⟶ {0,1}ⁿ a block cipher.

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

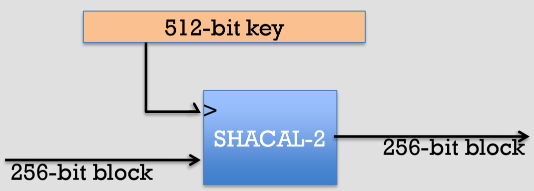


<u>Thm</u>: Suppose E is an ideal cipher (collection of |K| random perms.). Finding a collision **h(H,m)=h(H',m')** takes **O($2^{n/2}$)** evaluations of (E,D).

Best possible !!

Case study: SHA-256

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



Standardized method: HMAC (Hash-MAC)

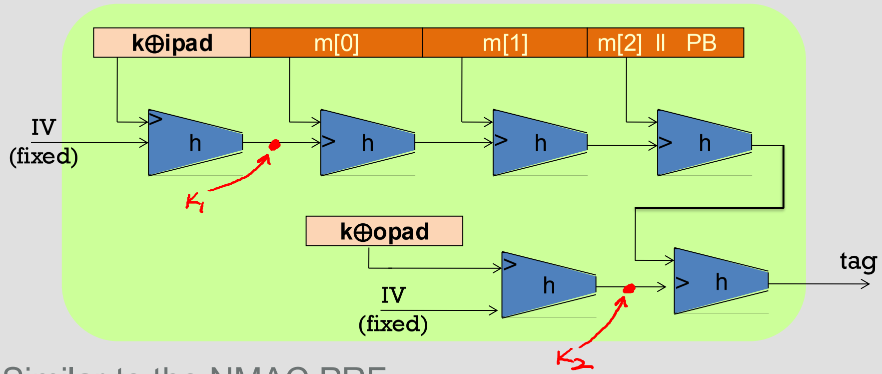
Most widely used MAC on the Internet.

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

Building a MAC out of a hash function:

HMAC: S(k, m) = H(k⊕opad ll **H(k⊕ipad ll m)**)

HMAC in pictures



Similar to the NMAC PRF.

main difference: the two keys k_1, k_2 are dependent

HMAC properties

Built from a black-box implementation of SHA-256.

HMAC is assumed to be secure
- Can be proven under certain PRF assumptions about h(.,.)

In TLS: must support HMAC-SHA1-96

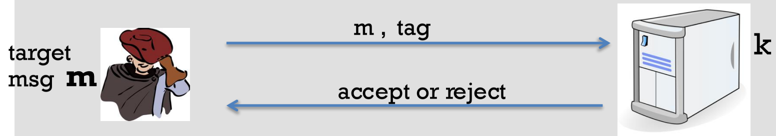
Warning: verification timing attacks [L'09]

Example: Keyczar crypto library (Python)
[simplified]

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

The problem: '==' implemented as a byte-by-byte comparison
- Comparator returns false when first inequality found

Warning: verification timing attacks [L'09]



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

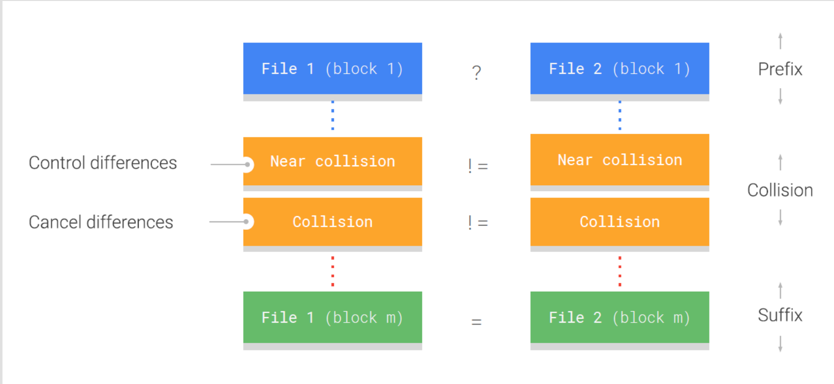
Numbers

MD5
1 smartphone
30 sec

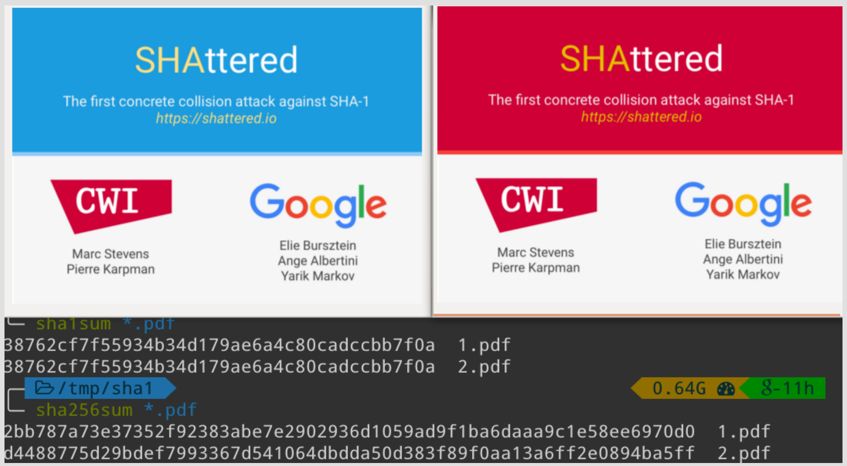
SHA-1 shattered
110 GPU
1 year

SHA-1 bruteforce
12,000,000 GPU
1 year

Main Idea



The end Result



Future

SHA-1 is dead
long live to
SHA-256 & SHA-3

End of an era

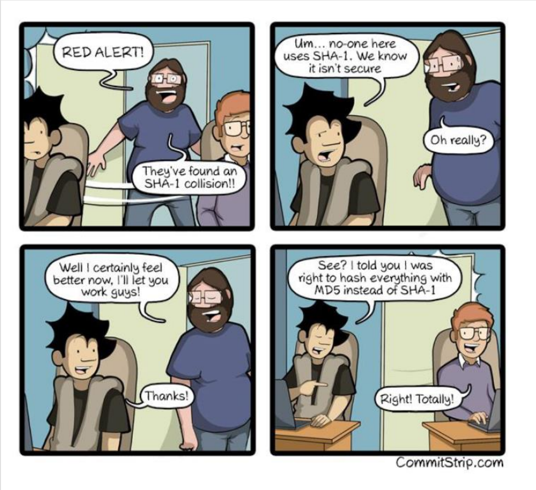
Counter-cryptanalysis
as a means of
detection

Hash cryptanalysis as a mean to detect unknown collisions

Hash diversity
as a safeguard for
the years to come

We now have a very diverse set of hash function constructions

Take away



Summary – Message Integrity

- Message Authentication Code (MAC) – Defend against existential forgery attack
  - ECBC-MAC, CMAC (NIST)
  - NMAC
- Hash Function
  - Collision Resistant
  - Merkle-Damgard iteration
  - Davies-Meyer Compression Function
  - Collision attack on SHA1