Cryptography
CS 555

Topic 13: HMACs and Generic Attacks

Recap

- Cryptographic Hash Functions
- Merkle-Damgård Transform

Today's Goals:

- HMACs (constructing MACs from collision-resistant hash functions)
- Generic Attacks on Hash functions

MACs for Arbitrary Length Messages

$\text{Mac}_K(m)$=

- Select random n/4 bit string r
- Let $t_i = \text{Mac}'_K(r \parallel \ell \parallel i \parallel m_i)$ for i=1,…,d
  - (Note: encode i and ℓ as n/4 bit strings)
- **Output** $\langle r, t_1, \ldots, t_d \rangle$

Theorem 4.8: If Π' is a secure MAC for messages of fixed length n, above construction $\Pi = (\text{Mac}, \text{Vrfy})$ is secure MAC for arbitrary length messages.

MACs for Arbitrary Lengt…

Disadvantage 1: Long output

Two Disadvantages:

1. Lose Strong-MAC Guarantee
2. Security game arguably should give attacker Vrfy(.) oracle
(CPA vs CCA security)

… i ‖

- (Note: … and ℓ as n/4 …
- **Output** $\langle r, t_1, \ldots, t_d \rangle$

Randomized Construction (no canonical verification). Disadvantage?

Theorem 4.8: If Π' i…
above constructio…
messages.

Hash and MAC Construction

Start with (Mac,Vrfy) a MAC for messages of fixed length and (Gen_H,H) a collision resistant hash function

$$Mac'_{\langle K_M, S\rangle}(m) = Mac_{K_M}\left(H^s(m)\right)$$

Theorem 5.6: Above construction is a secure MAC.

Note: If $\mathrm{Vrfy}_{K_M}(m, t)$ is canonical then $\mathrm{Vrfy}'_{\langle K_M, S\rangle}(m, t)$ can be canonical.

Hash and MAC Construction

Start with (Mac,Vrfy) a MAC for messages of fixed length and (Gen_H,H) a collision resistant hash function

$$Mac'_{\langle K_M, S\rangle}(m) = Mac_{K_M}\left(H^s(m)\right)$$

Theorem 5.6: Above construction is a secure MAC.

Proof Intuition: If attacker successfully forges a valid MAC tag t' for unseen message m' then either

- **Case 1:** $H^s(m') = H^s(m_i)$ for some previously requested message m_i
- **Case 2:** $H^s(m') \neq H^s(m_i)$ for every previously requested message m_i

Hash and MAC Construction

Theorem 5.6: Above construction is a secure MAC.

Proof Intuition: If attacker successfully forges a valid MAC tag t' for unseen message m' then either

- **Case 1:** $H^s(m') = H^s(m_i)$ for some previously requested message m_i
  - **Attacker can find hash collisions!**
- **Case 2:** $H^s(m') \neq H^s(m_i)$ for every previously requested message m_i
  - **Attacker forged a valid new tag on the "new message" $\boldsymbol{H^s(m')}$**
  - **Violates security of the original fixed length MAC**

MAC from Collision Resistant Hash

- Failed Attempt:

$$Mac_{\langle k,S\rangle}(m) = H^s(k \parallel m)$$

Broken if H^s uses Merkle-Damgård Transform

$$Mac_{\langle k,S\rangle}(m_1 \parallel m_2 \parallel m_3) = h^s(h^s(h^s(h^s(0^n \parallel k) \parallel m_1) \parallel m_2) \parallel m_3)$$
$$= h^s\left(Mac_{\langle k,S\rangle}(m_1 \parallel m_2) \parallel m_3\right)$$

Why does this mean $\boldsymbol{Mac_{\langle k,S\rangle}}$ is broken?

HMAC

$$Mac_{\langle k,S\rangle}(m) = H^s\Big((k \oplus \text{opad}) \parallel H^s\big((k \oplus \text{ipad}) \parallel m\big)\Big)$$

ipad?

HMAC

$$Mac_{\langle k,S\rangle}(m) = H^s\left((k \oplus \text{opad}) \parallel H^s\left((k \oplus \text{ipad}) \parallel m\right)\right)$$

$$\text{ipad} = \text{inner pad}$$
$$\text{opad} = \text{outer pad}$$

Both ipad and opad are fixed constants.

Why use key twice?

Allows us to prove security from *weak collision resistance* of H^s

HMAC Security

$$Mac_{\langle k,S\rangle}(m) = H^s\Big((k \oplus \mathrm{opad}) \parallel H^s\big((k \oplus \mathrm{ipad}) \parallel m\big)\Big)$$

Theorem (Informal): Assuming that H^s is weakly collision resistant and that (certain other plausible assumptions hold) this is a secure MAC.

Weak Collision Resistance: Give attacker oracle access to $f(m) = H^s(k \parallel m)$ (secret key k remains hidden).

Attacker Goal: Find distinct m,m’ such that $f(m) = f(m')$

HMAC in Practice

- MD5 can no longer be viewed as collision resistant

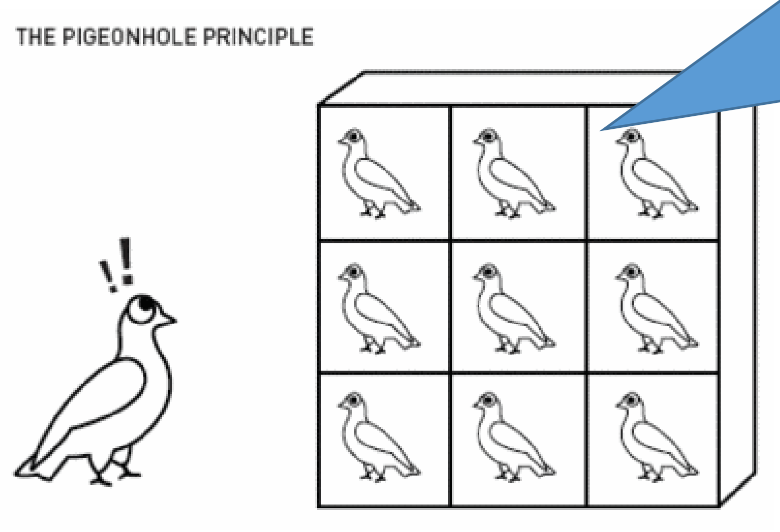
- However, HMAC-MD5 remained unbroken after MD5 was broken
  - Gave developers time to replace HMAC-MD5
  - Nevertheless, don’t use HMAC-MD5!

- HMAC is efficient and unbroken
  - CBC-MAC was not widely deployed because it as “too slow”
  - Instead practitioners often used heuristic constructions (which were breakable)

Finding Collisions

- Ideal Hashing Algorithm
  - Random function H from {0,1}* to $\{0,1\}^{\ell}$
  - Suppose attacker has oracle access to H(.)

- **Attack 1:** Evaluate H(.) on 2^{ℓ}+1 distinct inputs.



Can we do better?

Birthday Attack for Finding Collisions

- Ideal Hashing Algorithm
  - Random function H from $\{0,1\}^*$ to $\{0,1\}^{\ell}$
  - Suppose attacker has oracle access to H(.)

- **Attack 2:** Evaluate H(.) on $q = 2^{(\ell/2)+1} + 1$ distinct inputs $x_1,\ldots,x_q$.

$$\Pr[\forall i < j.\, H(x_i) \neq H(x_j)] = 1\left(1 - \frac{1}{2^{\ell}}\right)\left(1 - \frac{2}{2^{\ell}}\right)\left(1 - \frac{3}{2^{\ell}}\right)\ldots\left(1 - \frac{2^{(\ell/2)+1}}{2^{\ell}}\right) < \frac{1}{2}$$

Birthday Attack for Finding Collisions

- Ideal Hashing Algorithm
  - Random function H from $\{0,1\}^*$ to $\{0,1\}^\ell$
  - Suppose attacker has oracle access to H(.)

- **Attack 2:** Evaluate H(.) on $q = 2^{(\ell/2)+1} + 1$ distinct inputs $x_1,\ldots,x_q$.
- Store values $(x_i, H(x_i))$ in a hash table of size q
  - Requires time/space $O(q) = O\left(\sqrt{2^\ell}\right)$
  - Can we do better?

Small Space Birthday Attack

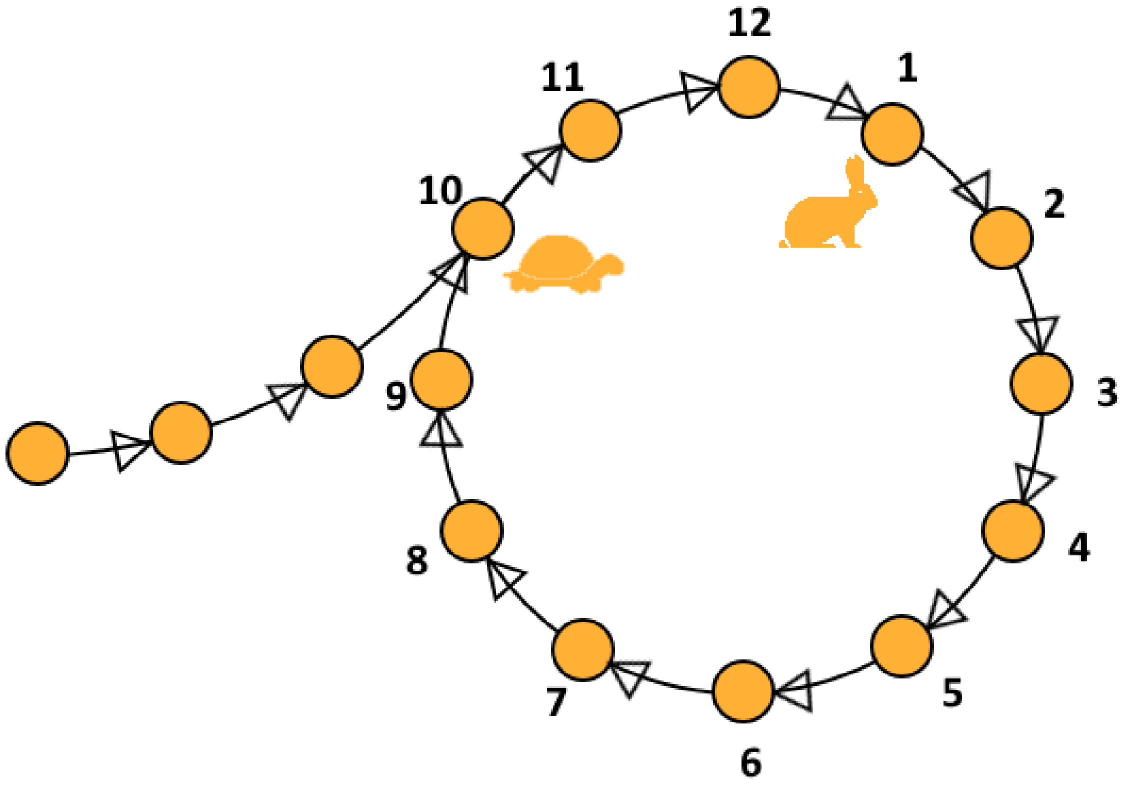
- **Attack 2:** Select random x_0, define $x_i = H(x_{i-1})$
  - Initialize: $x=x_0$ and $x'=x_0$
  - Repeat for i=1,2,…
    - x:=H(x) now $x = x_i$
    - x':=H(H(x')) now $x' = x_{2i}$
    - If x=x' then break
  - Reset $x=x_0$ and set $x'=x$
  - Repeat for j=1 to i
    - If H(x) = H(x') then output x,x'
    - Else x:= H(x), x' = H(x) Now $x=x_j$ AND $x' = x_{i+j}$

Small Space Birthday Attack

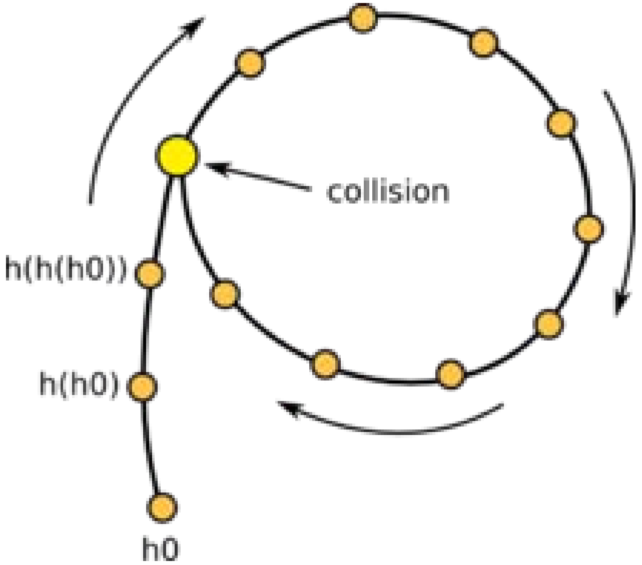
- **Attack 2:** Select random x_0, define $x_i = H(x_{i-1})$
  - Initialize: $x=x_0$ and $x'=x_0$
  - Repeat for i=1,2,…
    - $x:=H(x)$ now $x = x_i$
    - $x':=H(H(x'))$ now $x' = x_{2i}$
    - If $x=x'$ then break
  - Reset $x=x_0$ and set $x'=x$
  - Repeat for j=1 to i
    - If $H(x) = H(x')$ then output x,x'
    - Else $x:= H(x)$, $x' = H(x)$ Now $x=x_j$ AND $x' = x_{i+j}$

Finds collision after $O(2^{\ell/2})$ steps in expectation

Floyd's Cycle Finding Algorithm



- A cycle denotes a hash collision
- Occurs after $O\left(2^{\ell/2}\right)$ steps by birthday paradox
- First attack phase detects cycle
- Second phase identifies collision



- Analogy: Cycle detection in linked list
- Can traverse "linked list" by computing H

Small Space Birthday Attack

- Can be adapted to find "meaningful collisions" if we have a large message space $O\left(2^{\ell}\right)$

- **Example**: $S = S_1 \cup S_2$ with $|S_1| = |S_2| = 2^{\ell-1}$
  - S_1 = Set of positive recommendation letters
  - S_2 = Set of negative recommendation letters

- **Goal**: find $z_1 \in S_1$, $z_2 \in S_2$, such that $H(z_1) = H(z_2)$

- Can adapt previous attack by assigning unique binary string $\mathrm{b}(\mathrm{x}) \in \{0,1\}^{\ell}$ of length to each $x \in S$

$$\mathrm{x}_{\mathrm{i}} = H(\mathrm{b}(\mathrm{x}_{\mathrm{i-1}}))$$

Targeted Collision (e.g., Password Cracking)

- Attacker is given y=H(pwd)
- Goal find x' s.t. H(x') = y

- There is an attack which requires
  - Precomputation Time: $O(|PASSWORDS|)$
  - Space: $|PASSWORDS|^{2/3}$
  - On input y finds pwd in Time: $|PASSWORDS|^{2/3}$
- Cracking costs amortize over many users…
- Other time-memory tradeoffs are possible…
- **Defense 1:** y=H(pwd|salt)   [password salting]
- **Defense 2:** Make sure that H is moderately expensive to compute (MHFs)

Targeted Collision (e.g., Password Cracking)

- Attacker is given y=H(x)
- Goal find x' s.t. H(x') = y
- Precomputation (sketch)
  - Store $s = 2^{\ell/3}$ pairs (SP_i,EP_i) where $EP_i = H^t(SP_i)$ and t = $2^{\ell/3}$
- Let $y=y_0$
- For i=1,2…., $2^{\ell/3}$
  - $y_i = H(y_{i-1})$
  - For each j s.t $EP_j=y_i$
    - Check if y is in the hash chain (SP_i,EP_i)
    - Yes → Found desired x'

Space: $2^{\ell/3}$
Precomputation Time: $2^{2\ell/3}$

Total #j's $= \frac{st^2}{2^\ell} < O(1)$

Total Runtime $= O(t) = O(2^{\ell/3})$

Success Rate $\approx \frac{1}{4t}$

Targeted Collision (e.g., Password Cracking)

- Attacker is given y=H(x)
- Goal find x' s.t. H(x') = y
- Precomputation (sketch)
  - Store $4st = 4 \times 2^{2\ell/3}$ pairs $\left(SP_i^j, EP_i^j\right)$ where $EP_i^j = Ht(c_j \oplus SP_i)$ and t = $2^{\ell/3}$
- Let $y=y_0$
- For i=1,2…., $2^{\ell/3}$
  - $y_i^j = H(c_j \oplus y_{i-1})$
  - Foreach j s.t $EP_i^j = y_i^j$
  - Check if y is in the hash chain (SP_i,EP_i)
    - Yes → Found desired x'

Space: $2^{2\ell/3}$

Precomputation Time: $2^{\ell} = 2^{2\ell/3} 2^{\ell/3}$

Repeat for each j < t

Total Runtime $= O(t \times t) = O(2^{2\ell/3})$

Success Rate > 0.63

Targeted Collisions (Other Applications)

- Define $H(K) = F_k(x)$

- Suppose attacker obtains a pair $x, F_k(x)$ (chosen plaintext attack)
- There is a key recovery attack with
  - Precomputation Time: $|\mathcal{K}|$
  - Space: $|\mathcal{K}|^{2/3}$
  - Cracking Time: $|\mathcal{K}|^{2/3}$
- Precomputation costs amortize if we are attacking multiple different keys
  - As long as we have $x, F_{k'}(x)$ we don't need to repeat precomputation phase

Next Class

- Read Katz and Lindell 5.5-5.6
- Random Oracle Model + Applications of Hashing.