Homework 2 Released

- Due: Thursday, September 28th at 9 AM (beginning of class)
- Please Typeset Your Solutions (LaTeX, Word etc…)
- You may collaborate, but must write up your own solutions in your own words

Cryptography
CS 555

Week 5:

- Loose Ends
- Cryptographic Hash Functions
- HMACs
- Generic Attacks
- Random Oracle Model
- Applications of Hashing

Readings: Katz and Lindell Chapter 5, Appendix A.4

Fall 2017

Recap

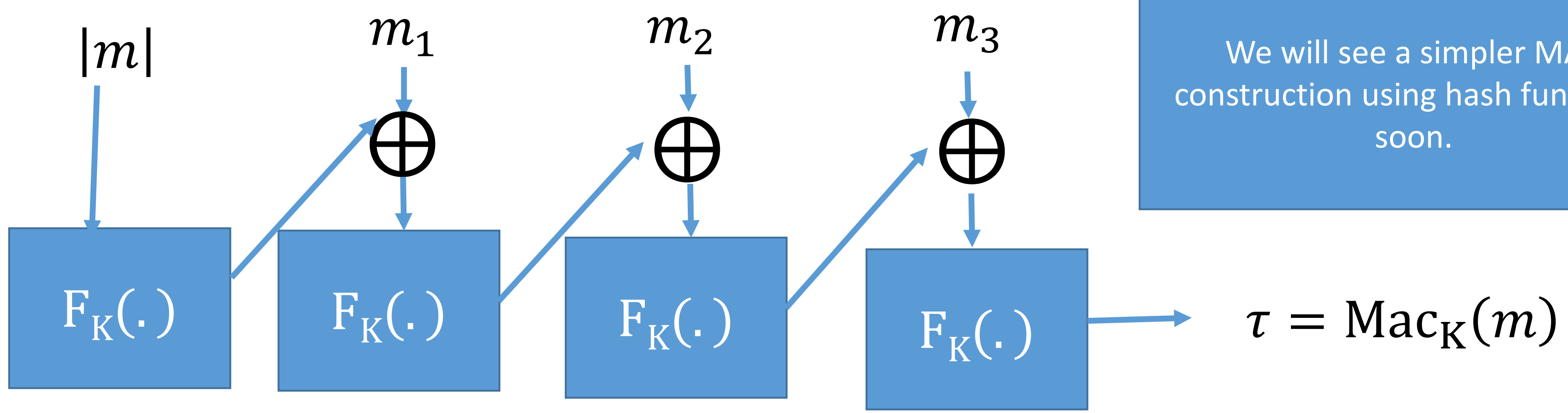
- Message Authentication Codes
  - Integrity vs Confidentiality

$$\mathrm{Mac_k}(m) = \mathrm{F_K}(m)$$

  - Extension to unbounded messages and pitfalls (block re-ordering, truncation)
  - ~~CBC-MAC~~
- Authenticated Encryption + CCA-Security
  - ~~Encrypt and Authenticate [SSH]~~
  - Authenticate then Encrypt [TLS] (Caution Required)
  - Encrypt **then** Authenticate!

$$Enc_K(m) = \langle \mathrm{c}, \mathrm{Mac}'_{K_M}(\mathrm{c}) \rangle \text{ where } \mathrm{c} = \mathrm{Enc}'_{K_E}(m)$$

CBC-MAC



Caveat: Tricky Padding Issues arise if m is not a multiple of the block-length. See textbook.

We will see a simpler MAC construction using hash functions soon.

Advantages over Previous Solution

- Both MACs are secure
- Works for unbounded length messages
- Canonical Verification
- Short Authentication tag
- ~~Parallelizable~~

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

Building Authenticated Encryption

Theorem: (Encrypt-then-authenticate) Let $\mathrm{Enc}'_{K_E}(m)$ be a CPA-Secure encryption scheme and let $\mathrm{Mac}'_{K_M}(m)$ be a secure MAC. Then the following construction is an authenticated encryption scheme.

$$Enc_K(m) = \langle \mathrm{c}, \mathrm{Mac}'_{K_M}(\mathrm{c}) \rangle \text{ where } \mathrm{c} = \mathrm{Enc}'_{K_E}(m)$$

Proof Intuition: Suppose that we have already shown that any PPT attacker wins $\mathrm{Encforge}_{A,\Pi}$ with negligible probability.

Why does CCA-Security now follow from CPA-Security?

CCA-Attacker has decryption oracle, but cannot exploit it! Why?

Always sees ⊥ “invalid ciphertext” when he query with unseen ciphertext

Proof Sketch

1. Let **ValidDecQuery** be event that attacker submits new/valid ciphertext to decryption oracle

2. Show Pr[**ValidDecQuery**] = negl(n) for any PPT attacker
   - **Hint:** Follows from strong security of MAC since
   $$Enc_K(m) = \langle \mathrm{c}, \mathrm{Mac}'_{K_M}(\mathrm{c}) \rangle$$
   - This also implies unforgeability.

3. Show that attacker who does not issue valid decryption query wins CCA-security game with probability ½ + negl(n)
   - Hint: otherwise we can use A to break CPA-security
   - Hint 2: simulate decryption oracle by always returning ⊥ when given new ciphertext

Secure Communication Session

- Solution? Alice transmits $c_1 = Enc_K(m_1)$ to Bob, who decrypts and sends Alice $c_2 = Enc_K(m_2)$ etc…
- Authenticated Encryption scheme is
  - Stateless
  - For fixed length-messages
- We still need to worry about
  - Re-ordering attacks
    - Alice sends 2n-bit message to Bob as $c_1 = Enc_K(m_1)$, $c_2 = Enc_K(m_2)$
  - Replay Attacks
    - Attacker who intercepts message $c_1 = Enc_K(m_1)$ can replay this message later in the conversation
  - Reflection Attack
    - Attacker intercepts message $c_1 = Enc_K(m_1)$ sent from Alice to Bob and replays to c_1 Alice only

Secure Communication Session

- Defense
  - Counters ($CTR_{A,B}$,$CTR_{B,A}$)
    - Number of messages sent from Alice to Bob ($CTR_{A,B}$) --- initially 0
    - Number of messages sent from Bob to Alice ($CTR_{B,A}$) --- initially 0
    - Protects against Re-ordering and Replay attacks
  - Directionality Bit
    - $b_{A,B} = 0$ and $b_{B,A} = 1$ (e.g., since $A < B$)

- Alice: To send m to Bob, set $c = Enc_K(b_{A,B} \parallel CTR_{A,B} \parallel m)$, send c and increment $CTR_{A,B}$
- Bob: Decrypts c, (if $\perp$ then reject), obtain $b \parallel CTR \parallel m$
  - If $CTR \neq CTR_{A,B}$ or $b \neq b_{A,B}$ then reject
  - Otherwise, output m and increment $CTR_{A,B}$

Authenticated Security vs CCA-Security

- Authenticated Encryption → CCA-Security (by definition)

- CCA-Security does not necessarily imply Authenticate Encryption
  - But most natural CCA-Secure constructions are also Authenticated Encryption Schemes
  - Some constructions are CCA-Secure, but do not provide Authenticated Encryptions, but they are less efficient.
- Conceptual Distinction
  - CCA-Security the goal is secrecy (hide message from active adversary)
  - Authenticated Encryption: the goal is integrity + secrecy

Week 5: Topic 1: Cryptographic Hash Functions

Hash Functions

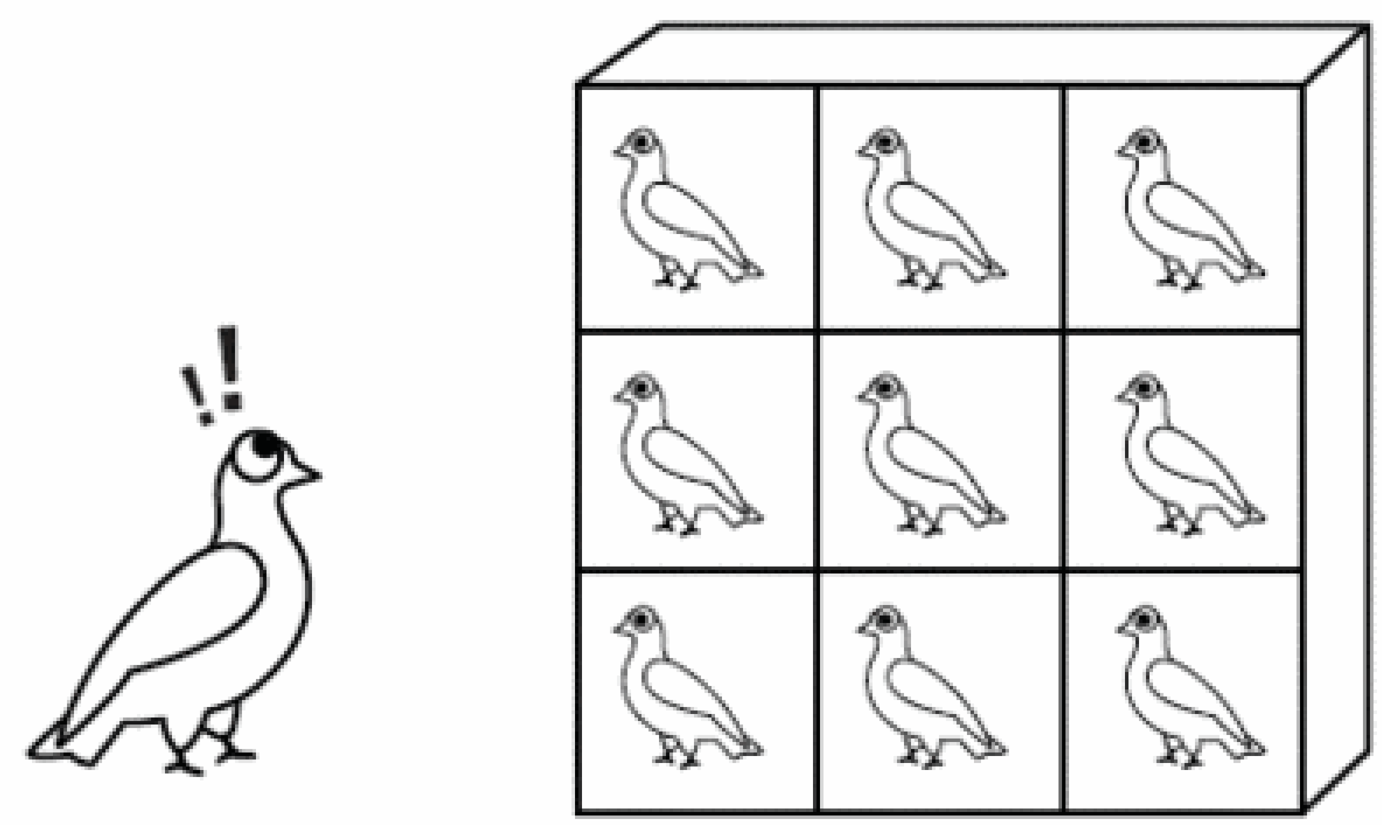
$$H(x)=y$$

Long Input: x

Short Output: y

Pigeonhole Principle

“You cannot fit 10 pigeons into 9 pigeonholes”



Hash Collisions

By Pigeonhole Principle there must exist x and y s.t.

$$H(x) = H(y)$$

Classical Hash Function Applications

- Hash Tables
  - O(1) lookup*
- “Good hash function” should yield “few collisions”

* Certain terms and conditions apply

Collision-Resistant Hash Function

Intuition: Hard for computationally bounded attacker to find x,y s.t. H(x) = H(y)

How to formalize this intuition?

- **Attempt 1:** For all PPT A,

$$\Pr\left[A_{x,y}(1^n) = (x, y)\ s.t\ H(x) = H(y)\right] \leq negl(n)$$

- **The Problem:** Let x,y be given s.t. H(x)=H(y)

$$A_{x,y}(1^n) = (x, y)$$

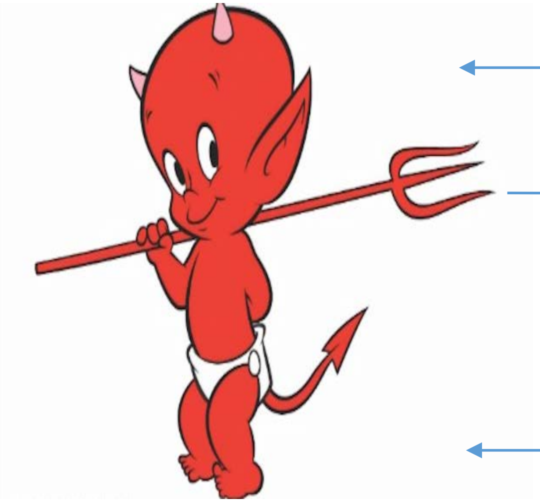
- We are assuming that |x| > |H(x)|. Why?
  - H(x)=x is perfectly collision resistant! (but with no compression)

Keyed Hash Function Syntax

- Two Algorithms
  - $\mathrm{Gen}(1^n; R)$ (Key-generation algorithm)
    - Input: Random Bits R
    - Output: ~~Secret~~ key s
  - $H^s(m)$ (Hashing Algorithm)
    - Input: key s and message $\mathrm{m} \in \{0,1\}^*$ (unbounded length)
    - Output: hash value $H^s(m) \in \{0,1\}^{\ell(n)}$

- Fixed length hash function
  - $m \in \{0,1\}^{\ell'(n)}$ with $\ell'(n) > \ell(n)$

Collision Experiment ($HashColl_{A,\Pi}(n)$)



s

x_1, x_2

$$HashColl_{A,\Pi}(n) = \begin{cases} 1 & if\ H^s(x_1) = H^s(x_2) \\ 0 & otherwise \end{cases}$$

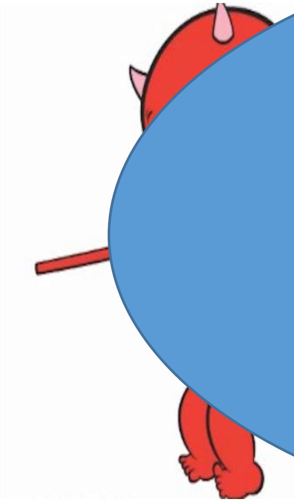
$s = \text{Gen}(1^n; R)$

Definition: (Gen,H) is a collision resistant hash function if

$$\forall PPT\ A\ \exists \mu\ \text{(negligible) s.t}$$

$$\Pr\left[HashColl_{A,\Pi}(n) = 1\right] \leq \mu(n)$$

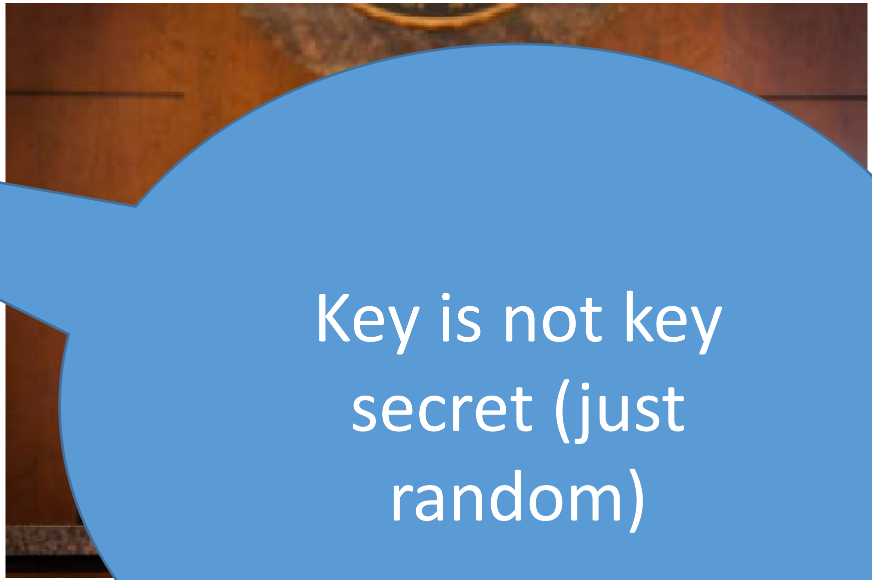
Collision Experiment ($HashColl_{A,\Pi}(n)$)



For simplicity we will sometimes just say that H (or H^s) is a collision resistant hash function

$= H^s(x_2)$

ise



Key is not key secret (just random)

Definition: (Gen,H) is a collision resistant hash function if

$$\forall PPT\ A\ \exists \mu \text{ (negligible) s.t}$$

$$\Pr\left[HashColl_{A,\Pi}(n)=1\right] \leq \mu(n)$$

Theory vs Practice

- Most cryptographic hash functions used in practice are un-keyed
  - Examples: MD5, SHA1, SHA2, SHA3
- Tricky to formally define collision resistance for keyless hash function
  - There is a PPT algorithm to find collisions
  - We just usually can't find this algorithm ☺

Formalizing Human Ignorance:
Collision-Resistant Hashing without the Keys

Phillip Rogaway

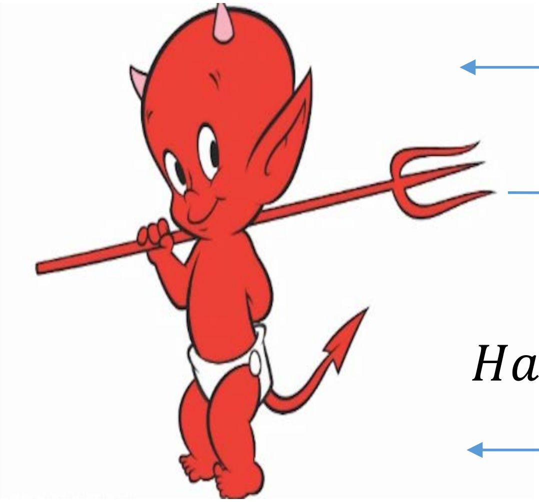
Department of Computer Science, University of California,
Davis, California 95616, USA, and
Department of Computer Science, Faculty of Science,
Chiang Mai University, Chiang Mai 50200, Thailand
rogaway@cs.ucdavis.edu

31 January 2007

Abstract. There is a foundational problem involving collision-resistant hash-functions: com-

Weaker Requirements for Cryptographic Hash

- Target-Collision Resistance



s,x

x’

$$HashTgtColl_{A,\Pi}(n) = \begin{cases} 1 & if\ H^s(x') = H^s(x) \\ 0 & otherwise \end{cases}$$

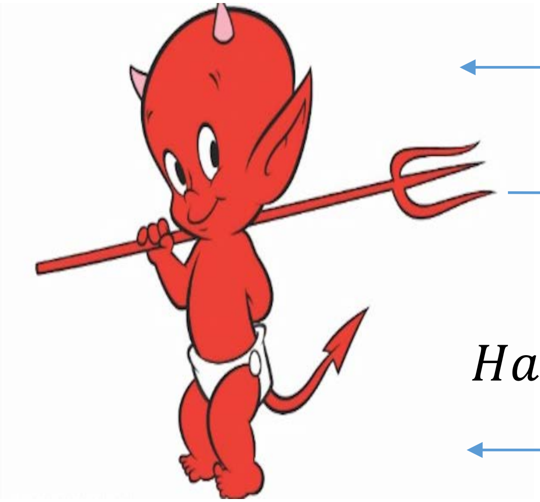
$s = \mathrm{Gen}(1^n; R)$

$x \in \{0,1\}^n$

Question: Why is collision resistance stronger?

Weaker Requirements for Cryptographic Hash

- Preimage Resistance (One-Wayness)



s, y

x

$$HashPreImgRes_{A,\Pi}(n) = \begin{cases} 1 & if\ H^s(x) = y \\ 0 & otherwise \end{cases}$$

$s = \mathrm{Gen}(1^n; R)$

$y \in \{0,1\}^{\ell(n)}$

Question: Why is collision resistance stronger?

Merkle-Damgård Transform

- Most cryptographic hash functions accept fixed length inputs
- What if we want to hash arbitrary length strings?

Construction: (Gen,h) fixed length hash function from 2n bits to n bits

$$H^s(x_1, \dots, xd) = h^s\big(h^s\big(h^s\big(\dots h^s(0^n \parallel x_1)\big) \parallel x_{d-1}\big) \parallel x_d\big)$$

Merkle-Damgård Transform

Construction: (Gen,h) fixed length hash function from 2n bits to n bits

$H^s(x)$ =

1. Break x into n bit segments $x_1,..,x_d$ (pad last block by zeros if needed)
2. $z_0 = 0^n$ (initialization)
3. For i = 1 to d+1
   1. $z_i = h^s(z_{i-1} \parallel x_i)$
4. Output z_{d+1}

Merkle-Damgård Transform

Theorem: If (Gen,h) is collision resistant then so is (Gen,H)

Proof: Show that any collision in H^s yields a collision in h^s. Thus a PPT attacker for (Gen,H) can be transformed into PPT attacker for (Gen,h).

Suppose that

$$H^s(x) = H^s(x')$$

(note x and x' may have different lengths)

Merkle-Damgård Transform

Theorem: If (Gen,h) is collision resistant then so is (Gen,H)

Proof: Suppose that

$$H^s(x) = H^s(x')$$

Case 1: |x|=|x'| (proof for case two is similar)

$$H^s(x) = z_{d+1} = h^s(z_d \parallel x_d) = H^s(x') = z'_{d+1} = h^s(z'_d \parallel x'_d)$$

$$z_d \parallel x_d =? \; z'_d \parallel x'_d$$

No → Found collision

Yes?

$$h^s(z_{d-1} \parallel x_{d-1}) = h^s(z'_{d-1} \parallel x'_{d-1})$$

Merkle-Damgård Transform

Theorem: If (Gen,h) is collision resistant then so is (Gen,H)

Proof: Suppose that

$$H^s(x) = H^s(x')$$

Case 1: |x|=|x'| (proof for case two is similar)

If for some i we have $z_i \parallel x_i \neq z_i' \parallel x_i'$ then we will find a collision

But x and x' are different!

Week 5: Topic 2: HMACs and Generic Attacks

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)

ri

|| i ||

and ℓ as n/4 bi

- **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 $Mac_{\langle k,S\rangle}$ 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

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

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

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

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

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!

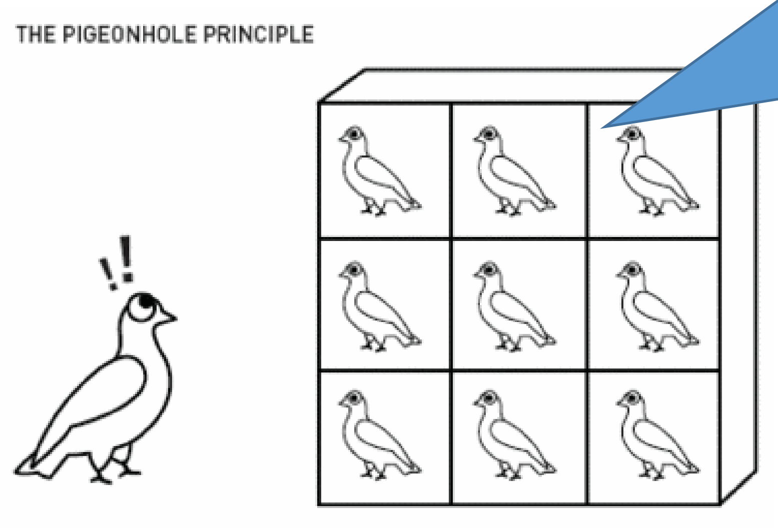
- HMAC is efficient and unbroken
  - CBC-MAC was not widely deployed because it as “too slow”
  - Instead practitioners often used heuristic constructions (which were breakable)

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 $2^\ell+1$ 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

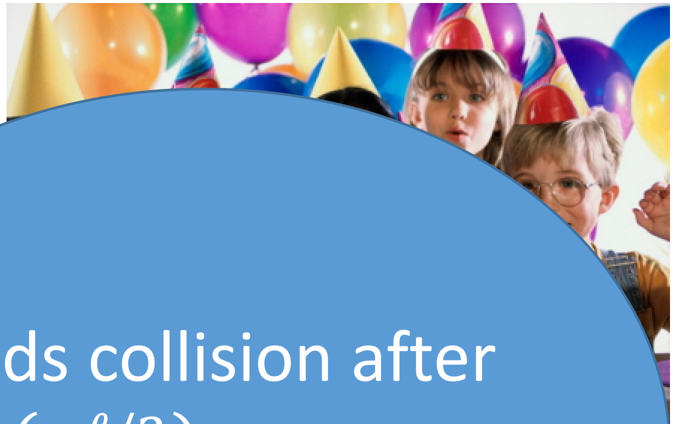
- Ideal Hashing Algorithm
  - Random function H from $\{0,1\}^*$ to $\{0,1\}^\ell$
  - Suppose attacker has oracle access to H(.)

- **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 $x' = x_{i+j}$

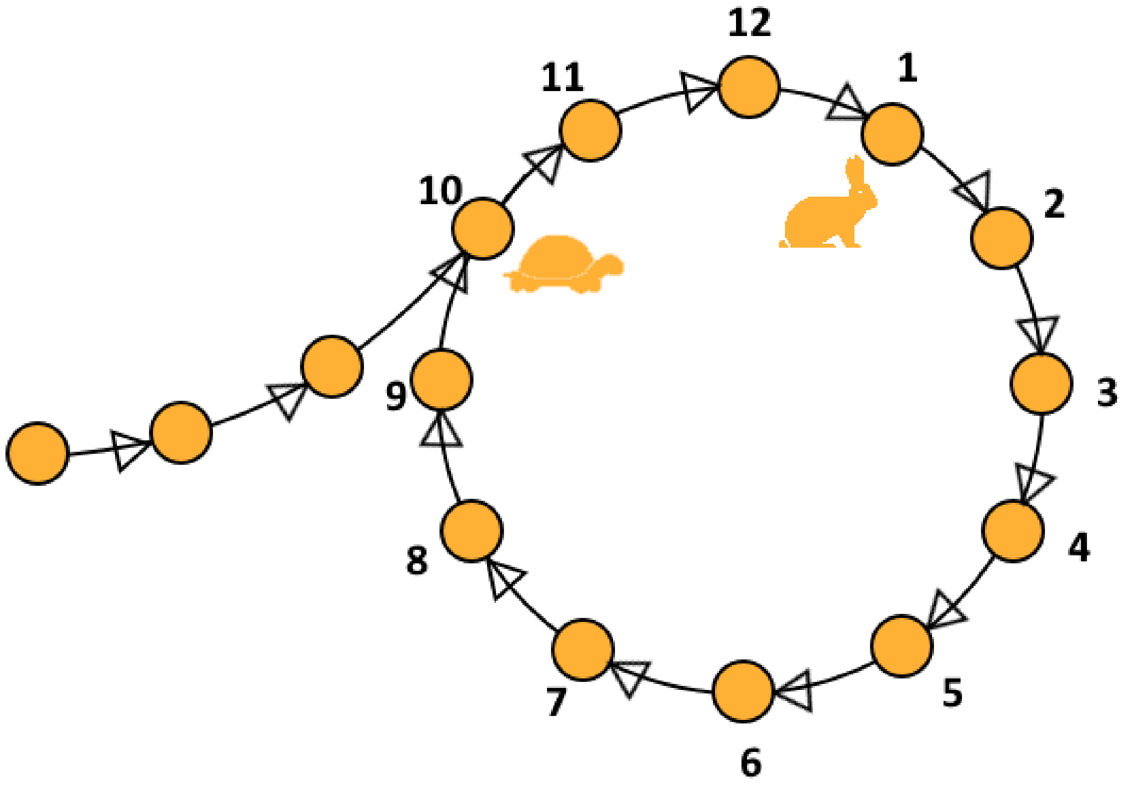
Small Space Birthday Attack



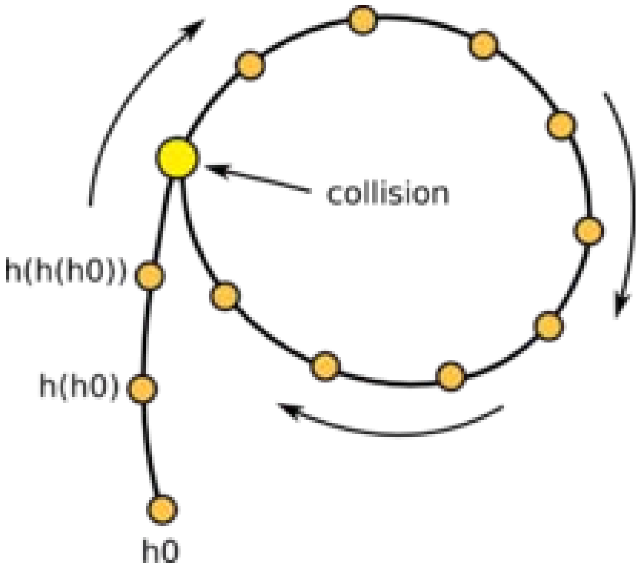
- **Attack 2:** Select random x_0, define $x_i = H(x_{i-1})$
  - Initialize: $x=x_0$ and $x'=x_0$
  - Repeat for i=1,2,…
    - $x:=H(x)$ now $x = x_i$
    - $x':=H(H(x'))$ now $x' = x_{2i}$
    - If $x=x'$ then break
  - Reset $x=x_0$ and set $x'=x$
  - Repeat for j=1 to i
    - If $H(x) = H(x')$ then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

Finds collision after $O\left(2^{\ell/2}\right)$ 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 $\mathrm{O}\left(2^{\ell}\right)$

- **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 $EP_i = Ht(SP_i)$ 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,EPi)
    - 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,EPi)
    - 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

Week 5: Topic 3:
Random Oracle Model +
Hashing Applications

(Recap) Collision-Resistant Hash Function

Intuition: Hard for computationally bounded attacker to find x,y s.t. H(x) = H(y)

How to formalize this intuition?

- **Attempt 1:** For all PPT A,

$$\Pr\left[A_{x,y}(1^n) = (x,y)\ s.t\ H(x) = H(y)\right] \leq negl(n)$$

- **The Problem:** Let x,y be given s.t. H(x)=H(y)

$$A_{x,y}(1^n) = (x,y)$$

- We are assuming that |x| > |H(x)|. Why?
  - H(x)=x is perfectly collision resistant! (but with no compression)

(Recap) Keyed Hash Function Syntax

- Two Algorithms
  - $\mathrm{Gen}(1^n; R)$ (Key-generation algorithm)
    - Input: Random Bits R
    - Output: ~~Secret~~ key s
  - $H^s(m)$ (Hashing Algorithm)
    - Input: key s and message $\mathrm{m} \in \{0,1\}^*$ (unbounded length)
    - Output: hash value $H^s(m) \in \{0,1\}^{\ell(n)}$

- Fixed length hash function
  - $m \in \{0,1\}^{\ell'(n)}$ with $\ell'(n) > \ell(n)$

When Collision Resistance Isn't Enough

- **Example**: Message Commitment
  - Alice sends Bob: $\mathrm{H}^s(r \parallel m)$ (e.g., predicted winner of NCAA Tournament)
  - Alice can later reveal message (e.g., after the tournament is over)
    - Just send r and m (note: r has fixed length)
    - Why can Alice not change her message?
  - In the meantime Bob shouldn't learn *anything* about m

- **Problem**: Let (Gen,H') be collision resistant then so is (Gen,H)

$$H^s(x_1, \dots, xd) = H'^s(x_1, \dots, xd) \parallel x_d$$

When Collision Resistance Isn't Enough

- **Problem**: Let (Gen,H') be collision resistant then so is (Gen,H)

$$H^s(x_1, \ldots, xd) = H'^s(x_1, \ldots, xd) \parallel x_d$$

- (Gen,H) definitely does not hide all information about input $(x_1, \ldots, xd)$

- **Conclusion**: Collision resistance is not sufficient for message commitment

The Tension

- **Example**: Message Commitment
  - Alice sends Bob: $\mathrm{H}^s(r \parallel m)$ (e.g., predicted winners of NCAA Final Four)
  - Alice can later reveal message (e.g., after the Final Four is decided)
  - In the meantime Bob shouldn't learn anything about m

This is still a reasonable approach in practice!

- No attacks when instantiated with any reasonable candidate (e.g., SHA3)
- Cryptographic hash functions seem to provide "something" beyond collision resistance, but how do we model this capability?

Random Oracle Model

- Model hash function H as a truly random function
- Algorithms can only interact with H as an oracle
  - **Query**: x
  - **Response**: H(x)
- If we submit the same query you see the same response
- If x has not been queried, then the value of H(x) is uniform

- **Real World:** H instantiated as cryptographic hash function (e.g., SHA3) of fixed length (no Merkle-Damgård)

Back to Message Commitment

- **Example**: Message Commitment
  - Alice sends Bob: $\mathrm{H}(r \parallel m)$ (e.g., predicted winners of NCAA Final Four)
  - Alice can later reveal message (e.g., after the Final Four is decided)
    - Just send r and m (note: r has fixed length)
    - Why can Alice not change her message?
  - In the meantime Bob shouldn't learn *anything* about m

- **Random Oracle Model**: Above message commitment scheme is secure (Alice cannot change m + Bob learns nothing about m)
- **Information Theoretic Guarantee** against any attacker with q queries to H

Random Oracle Model: Pros

- It is easier to prove security in Random Oracle Model
- Suppose we are simulating attacker A in a reduction
  - **Extractability**: When A queries H at x we **see this query** and learn x (and can easily find H(x))
  - **Programmability**: We can set the value of H(x) to a value of our choice
    - As long as the value is correctly distribute i.e., close to uniform
- Both **Extractability** and **Programmability** are useful tools for a security reduction!

Random Oracle Model: Pros

- It is easier to prove security in Random Oracle Model

- Provably secure constructions in random oracle model are often much more efficient (compared to provably secure construction is “standard model”

- Sometimes we only know how to design provably secure protocol in random oracle model

Random Oracle Model: Cons

- Lack of formal justification
- Why should security guarantees translate when we instantiate random oracle with a real cryptographic hash function?

- We can construct (contrived) examples of protocols which are
  - Secure in random oracle model…
  - But broken in the real world

Random Oracle Model: Justification

"A proof of security in the random-oracle model is significantly better than no proof at all."

- **Evidence of sound design** (any weakness involves the hash function used to instantiate the random oracle)
- **Empirical Evidence for Security**

  "there have been no successful real-world attacks on schemes proven secure in the random oracle model"

Hash Function Application: Fingerprinting

- The hash h(x) of a file x is a unique identifier for the file
  - Collision Resistance → No need to worry about another file y with H(y)=H(y)

- Application 1: Virus Fingerprinting

- Application 2: P2P File Sharing

- Application 3: Data deduplication

Tamper Resistant Storage

$H(m_1)$

m_1

Dropbox

m_1'

Tamper Resistant Storage

File Index	Hash
1	$H(m_1)$
2	$H(m_2)$
3	$H(m_3)$

Disadvantage: Too many hashes to store

Dropbox

m_1, m_2, m_3

Send file 1

m_1'

Tamper Resistant Storage

Disadvantage: Need all files to compute hash m_1,m_2,m_3

$H(m_1,m_2,m_3)$

m_1,m_2,m_3

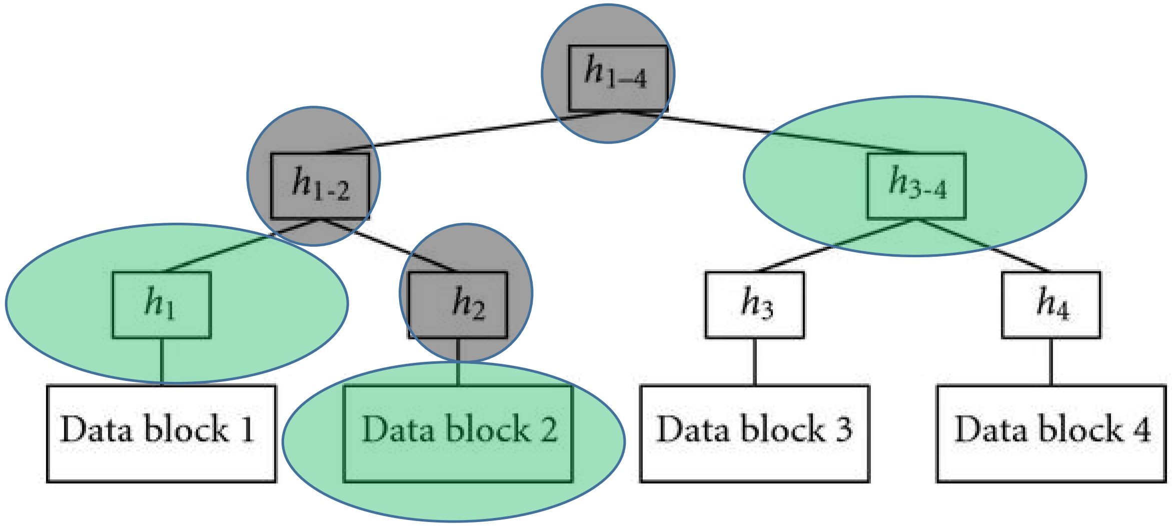
Send file 1

m_1'

Dropbox

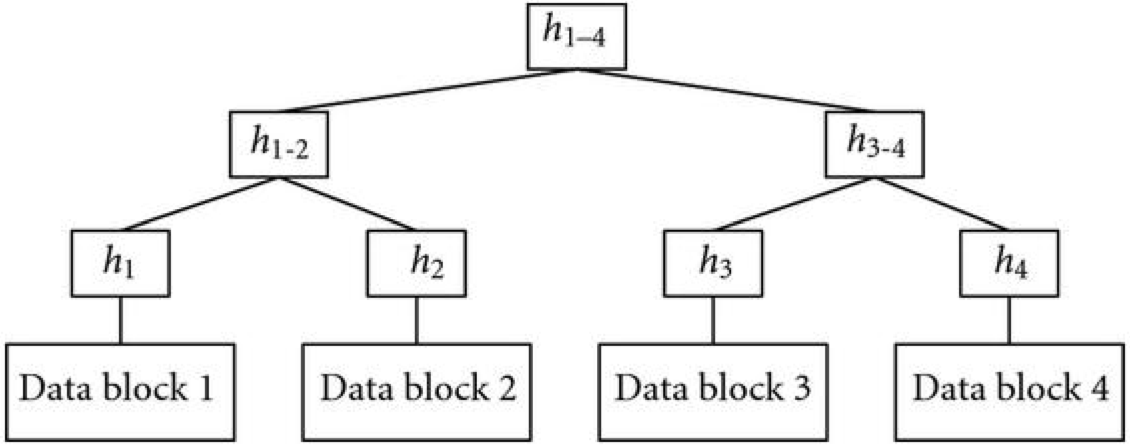
Merkle Trees

- **Proof of Correctness for data block 2**



- **Verify that root matches**
- **Proof consists of just log(n) hashes**
  - **Verifier only needs to permanently store only one hash value**

Merkle Trees



Theorem: Let (Gen, h^s) be a collision resistant hash function and let $H^s(m)$ return the root hash in a Merkle Tree. Then H^s is collision resistant.

Tamper Resistant Storage

Root: $H_{1\text{-}4}$

m_1, m_2, m_3, m_4

Send file 2

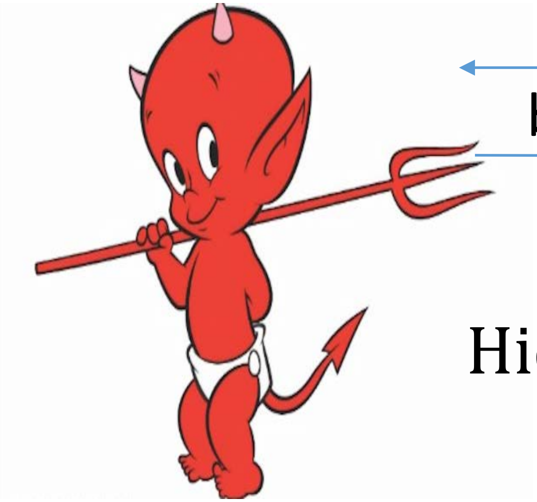
$m_2', h_1, h_{3\text{-}4}$

Dropbox

Commitment Schemes

- Alice wants to commit a message m to Bob
  - And possibly reveal it later at a time of her choosing
- Properties
  - Hiding: commitment reveals nothing about m to Bob
  - Binding: it is infeasible for Alice to alter message

Commitment Hiding $\left(\mathrm{Hiding}_{A,Com}(n)\right)$



m_0, m_1 →

← **commit(r,m_b)**

b' →

$$\mathrm{Hiding}_{A,Com}(n) = \begin{cases} 1 & \text{if } b = b' \\ 0 & otherwise \end{cases}$$

r = Gen(.)

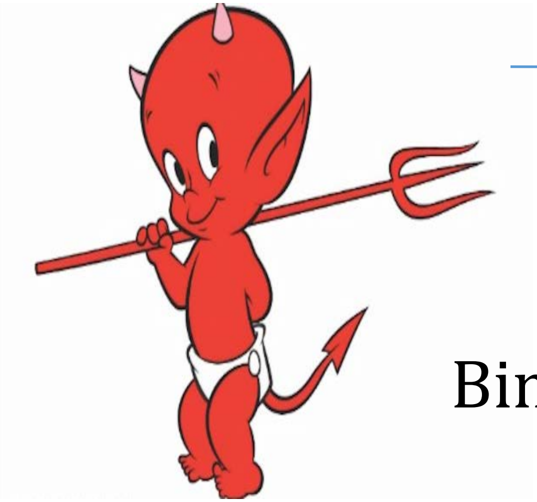
Bit b

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}$$

$$\Pr\left[\mathrm{Hiding}_{A,Com}(n) = 1\right] \le \frac{1}{2} + \mu(n)$$

Commitment Binding ($\text{Binding}_{A,Com}(n)$)

r_0, r_1, m_0, m_1



$$\text{Binding}_{A,Com}(n) = \begin{cases} 1 & \text{if } \mathbf{commit(r_0,m_0) = commit(r_1,m_1)} \\ 0 & otherwise \end{cases}$$

$$\forall PPT\ A\ \exists \mu \text{ (negligible) s.t}$$
$$\Pr[\text{Binding}_{A,Com}(n) = 1] \leq \mu(n)$$

Secure Commitment Scheme

- **Definition:** A secure commitment scheme is **hiding** and **binding**
- **Hiding**

$$\forall PPT\ A\ \exists\mu\ (\text{negligible})\ \text{s.t}$$
$$\Pr\left[\text{Hiding}_{A,Com}(n) = 1\right] \le \frac{1}{2} + \mu(n)$$

- **Binding**

$$\forall PPT\ A\ \exists\mu\ (\text{negligible})\ \text{s.t}$$
$$\Pr\left[\text{Binding}_{A,Com}(n) = 1\right] \le \mu(n)$$

Commitment Scheme in Random Oracle Model

- **Commit**(r,m):=H(m|r)

- **Reveal**(c):= (m,r)

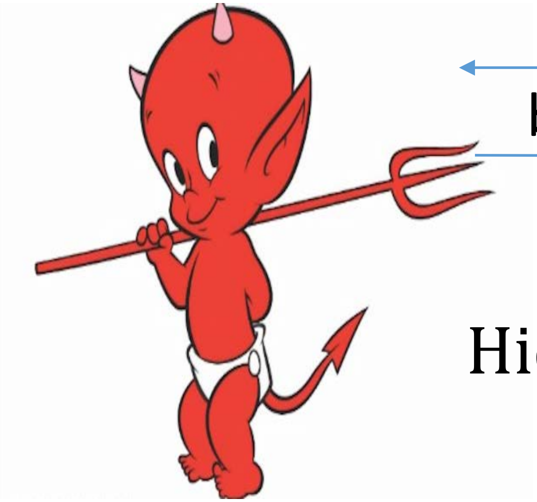
Theorem: In the random oracle model this is a secure commitment scheme.

Commitment Hiding $\left(\text{Hiding}_{A,Com}(n)\right)$

m_0,m_1

$H(r,m_b)$

b’



$$\text{Hiding}_{A,Com}(n) = \begin{cases} 1 & if\, b = b' \\ 0 & otherwise \end{cases}$$

r = Gen(.)

Bit b

$$\forall \cancel{PPT}\ A\ making\ q(n)\ queries\ \text{s.t}$$

$$\Pr\left[\text{Hiding}_{A,Com}(n) = 1\right] \le \frac{1}{2} + \frac{q(n)}{2^{|r|}}$$

Other Applications

- Password Hashing

- Key Derivation

Next Week

- Stream Ciphers
- Block Ciphers
- Feistel Networks
- DES, 3DES
- Read Katz and Lindell 6.1-6.2

Revisit: Building Authenticated Encryption

Attempt 3: (Authenticate-then-encrypt) Let $\mathrm{Enc}'_{K_E}(m)$ be a CPA-Secure encryption scheme and let $\mathrm{Mac}'_{K_M}(m)$ be a secure MAC. Let $K = (K_E, K_M)$ then

$$Enc_K(m) = \left\langle \mathrm{Enc}'_{K_E}(m \parallel t) \right\rangle \text{ where } \quad \mathrm{t} = \mathrm{Mac}'_{K_M}(m)$$

Doesn't **necessarily** work

- Approach is still used in TLS
- Some practitioners still advocate for this methodology

Building Authenticated Encryption

Attempt 3: (Authenticate-then-encrypt) Let $\mathrm{Enc}'_{K_E}(m)$ be a CPA-Secure encryption scheme and let $\mathrm{Mac}'_{K_M}(m)$ be a secure MAC. Let $K = (K_E, K_M)$ then

$$\mathrm{Enc}_{\mathrm{K}}(m) = \left\langle \mathrm{Enc}'_{K_E}(m \parallel t) \right\rangle \text{ where } \ \mathrm{t} = \mathrm{Mac}'_{K_M}(m)$$

A Bad Example:

$$\mathrm{Enc}_{\mathrm{K}}(m) = \left\langle r, \mathrm{F}_{K_E}(r) \oplus (\mathrm{ECC(m)} \parallel t) \right\rangle$$
$$\mathrm{ECC(m)} = w \text{ (codeword for error correction)}$$
$$\mathrm{Decode(w')} = m \text{ if w}' \text{and w are `mostly the same}'$$

Source: The Order of Encryption and Authentication for protecting communication by Hugo Krawczyk
https://eprint.iacr.org/2001/045

Building Authenticated Encryption

$$\mathrm{Enc_K}(m) = \langle r, \mathrm{F}_{K_E}(r) \oplus (\mathrm{ECC(m)} \parallel t) \rangle \text{ where } \begin{array}{l} \mathrm{t} = \mathrm{Mac}'_{K_M}(m) \\ \mathrm{w} = \mathrm{ECC(m)} \end{array}$$

$Dec_K(\langle r, s \rangle)$

- $w \parallel t \leftarrow \mathrm{F}_{K_E}(r) \oplus s$
- $m \leftarrow Decode(w)$
- $output = \begin{cases} m & \text{if } \mathrm{t} = \mathrm{Mac}'_{\mathrm{K_M}}(m) \\ \perp & \text{otherwise} \end{cases}$

Source: The Order of Encryption and Authentication for protecting communication by Hugo Krawczyk
https://eprint.iacr.org/2001/045

Building Authenticated Encryption

$$\mathrm{Enc_K}(m) = \langle r, \mathrm{F}_{K_E}(r) \oplus (\mathrm{ECC(m)} \parallel t) \rangle \text{ where } \begin{array}{l} \mathrm{t = Mac'_{K_M}}(m) \\ \mathrm{w = ECC(m)} \end{array}$$

$Dec_K(\langle r, s \rangle)$

- $w \parallel t \leftarrow \mathrm{F}_{K_E}(r) \oplus s$
- $m \leftarrow Decode(w)$
- $output = \begin{cases} m & \text{if } \mathrm{t = Ma} \\ \perp & \mathrm{o} \end{cases}$

Key Point: Error Correcting Code allows attacker to flip a few bits of s without altering message m.

$$\langle r, s \oplus 10^{n-1} \rangle = \langle r, \mathrm{F}_{K_E}(r) \oplus (w' \parallel t) \rangle$$

Source: The Order of Encryption and Authenticatio
https://eprint.iacr.org/2001/045

Building Authenticated Encryption

$$\mathrm{Enc_K}(m) = \langle r, \mathrm{F}_{K_E}(r) \oplus (\mathrm{ECC(m)} \parallel t) \rangle \text{ where } \quad \mathrm{t} = \mathrm{Mac}'_{K_M}(m)$$

$$\mathrm{w} = \mathrm{ECC(m)}$$

Worst Case: Chosen ciphertext attack allows attacker to completely recover plaintext.

Key Point: Error Correcting Code allows attacker to flip a few bits of s without altering message m.

$$\langle r, s \oplus 10^{n-1} \rangle = \langle r, \mathrm{F}_{K_E}(r) \oplus (w' \parallel t) \rangle$$

Source: The Order of Encryption and Authenticatio
https://eprint.iacr.org/2001/045