Cryptography
CS 555

Topic 14: Random Oracle Model, Hashing Applications

Recap

- HMACs
- Birthday Attack
- Small Space Birthday Attack
- Precomputation Attack

Today's Goals:

- Random Oracle Model
- Applications of Hash Functions

(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)

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

When Collision Resistance Isn't Enough

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

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

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

- **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

Dropbox

$H(m_1)$

m_1

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

$H(m_1,m_2,m_3)$

Dropbox

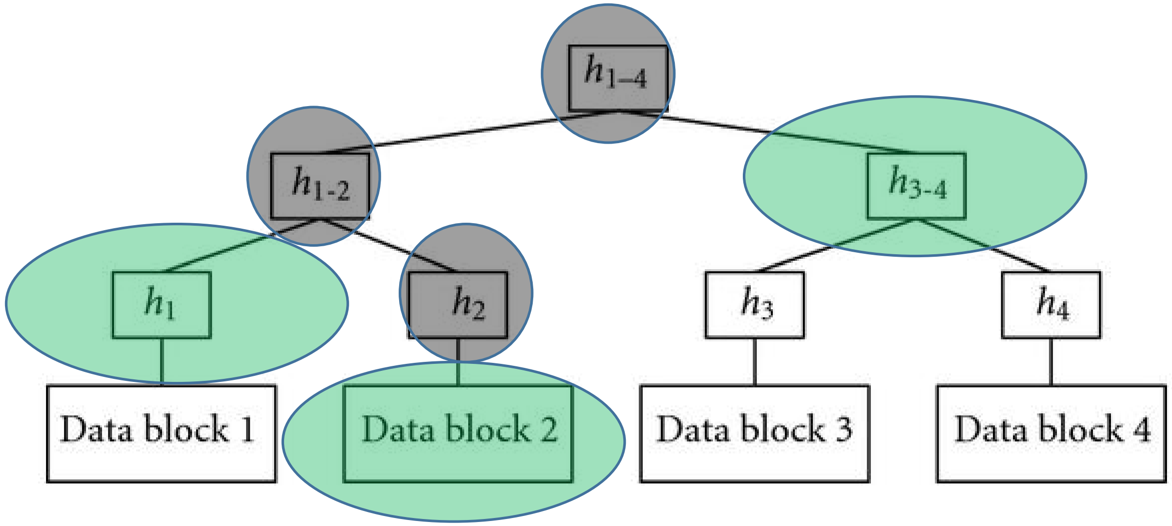
m_1,m_2,m_3

Send file 1

m_1'

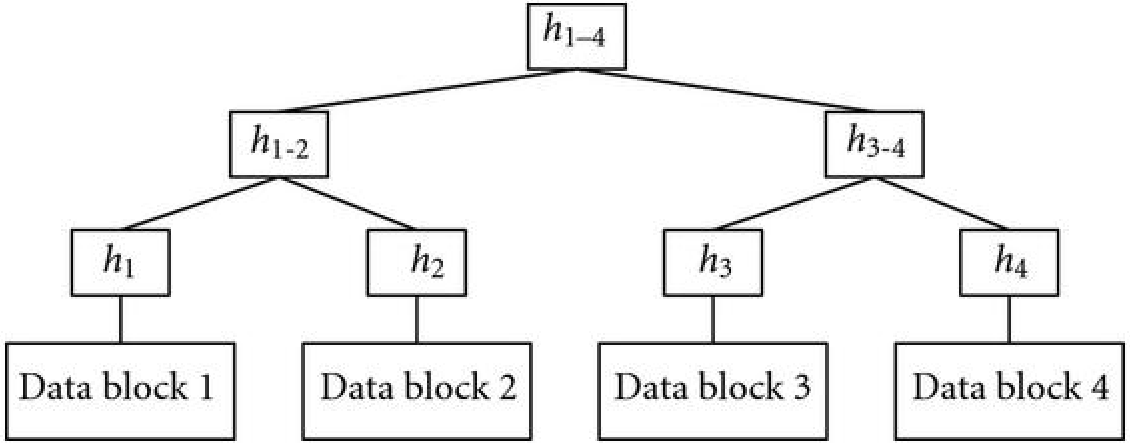
Merkle Trees

- **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_{1-4}

m_1, m_2, m_3, m_4

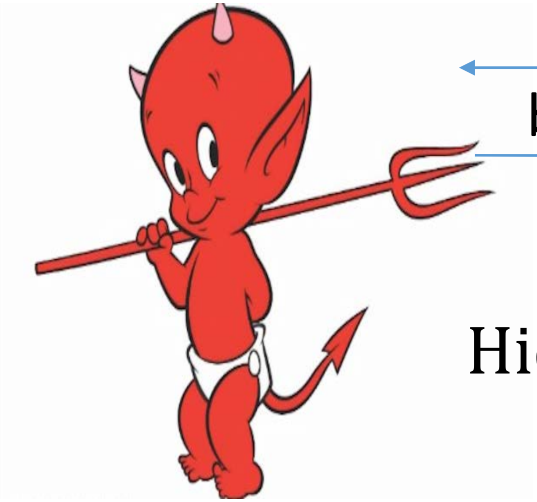
Send file 2

m_2', h_1, h_{3-4}

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



m_0, m_1 $\longrightarrow$

$\longleftarrow$ **commit(r,m_b)**

b’ $\longrightarrow$

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

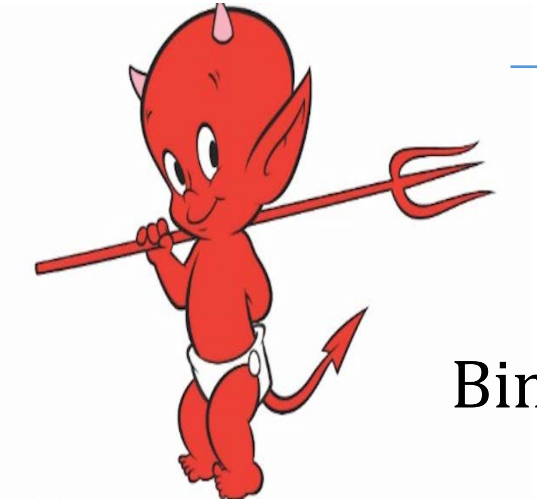
r = Gen(.)

Bit b

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

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

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



r_0, r_1, m_0, m_1 $\longrightarrow$

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

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

Secure Commitment Scheme

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

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

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

- **Binding**

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

$$\Pr\left[\text{Binding}_{A,Com}(n) = 1\right] \leq \mu(n)$$

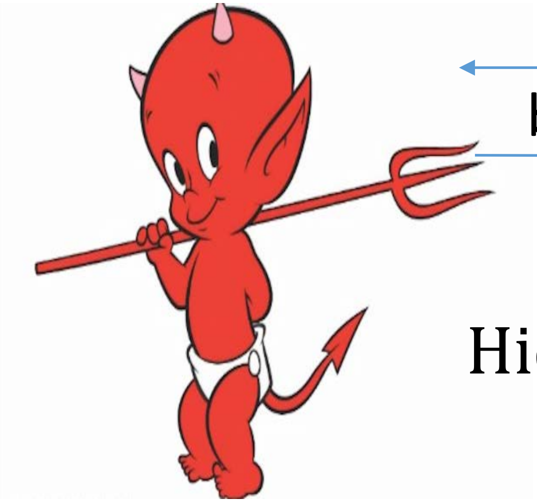
Commitment Scheme in Random Oracle Model

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

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

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

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



m_0, m_1

$H(r, m_b)$

b'

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

r = Gen(.)

Bit b

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

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

Other Applications

- Password Hashing

- Key Derivation

Next Class

- Read Katz and Lindell 6.1
- Stream Ciphers