Speaker's Notes: Symmetric Authentication and Hash Functions

Total Time: ~15 minutes

1. Hash Functions & Collision-Intractability (0-3:00)

[BOARD] Write:

> $h : \{0,1\}^* \to \{0,1\}^k$

- Hash function generator $\mathcal{H}$ produces function $h : \{0,1\}^* \to \{0,1\}^k$
- **Public function** - no secret key
- Maps arbitrary length input to fixed k-bit output

Use case:

- Store data m in insecure location
- Compute authenticator $h(m)$, store in secure location
- Later: retrieve m', compute $h(m')$, compare
- Also called *message digest* or *message fingerprint*

[BOARD] Add:

> **Second preimage:** given h, m → find $m' \neq m$ where $h(m) = h(m')$
>
> **Collision:** find ANY $m \neq m'$ where $h(m) = h(m')$

- Collision attack easier than second preimage
- Best security: make even collision attack infeasible

Definition - Collision Intractability:

- Game: Run $\mathcal{H}$ on input k → get function h
- Give h to adversary A
- A outputs m, m' with $m \neq m'$ and $h(m) = h(m')$
- $\mathcal{H}$ collision intractable if any PPT algorithm has negligible success probability

Key proof:

- Collision-intractability implies one-way
- If can invert hash, can find collision
- Therefore: cannot invert → certainly cannot find collision

[TRANSITION] "Now that we have hash functions mapping arbitrary inputs to fixed output, we need to build them efficiently. How do we construct collision-intractable hash for arbitrarily long inputs? Use Merkle-Damgård..."

---

2. Domain Extension - Merkle-Damgård Construction (3:00-5:30)

[BOARD] Write:

> **Given:** $f : \{0,1\}^m \to \{0,1\}^k$ where $m > k + 1$
>
> **Build:** $h : \{0,1\}^* \to \{0,1\}^k$

- Only need to build fixed-length collision-intractable function
- Merkle-Damgård extends to arbitrary length

Construction steps:

1. **Split into blocks**: input $x \to v$-bit blocks x_i (where $v = m - k - 1$)
    - Pad last block with 0s
    - Add extra block x_{n+1} with number of padding 0s
2. **Initialize**: $z_1 := 0^k||1||x_1$ (use "1" separator bit)
3. **Iterate**: For $i = 2, \ldots, n+1$: compute $z_i := f(z_{i-1})||0||x_i$ (use "0" separator)
4. **Output**: $h(x) := f(z_{n+1})$

- Separator bits prevent collision attacks
- Padding length prevents extension attacks

Modern alternative:

- SHA-3 uses **sponge design**
- Prevents length extension attacks better
- State maintained but output doesn't fully reveal internal state

[TRANSITION] "Hash functions alone insufficient for authentication - adversary could modify both message and hash. Need shared secret key. Enter MACs..."

---

3. Message Authentication Codes (MACs) (5:30-8:00)

Problem setup:

- Hash codes alone don't prevent MiM adversary from modifying both m and authenticator
- Need **shared secret key** between sender and receiver

[BOARD] Write:

> MAC scheme = (G, A, V)

Definition:

- G: key generation (outputs key K)
- A: authentication algorithm - $s = A_K(m)$ (produces MAC)
- V: verification algorithm - $V_K(s, m) \in \{\text{accept}, \text{reject}\}$
- **Correctness**: Always $V_K(A_K(m), m) = \text{accept}$

Flow:

1. Sender: compute authenticator s from message m and secret key k
2. Send (m, s) on channel, receive (m', s')
3. Receiver: test if m', s' valid using same key k

Also called: Keyed Hash Functions

[TRANSITION] "How do we measure MAC security? What attacks must it resist? Define CMA security..."

---

4. CMA Security for MACs (8:00-10:00)

[BOARD] Write:

> **CMA** = Chosen Message Attack

Security game:

- Adversary E has oracle with key K
- E may query oracle with any messages m → receives $A_K(m)$
- E **wins**: outputs NEW message m_0 and authenticator s_0 where $V_K(s_0, m_0) = \text{accept}$
- Must forge MAC for message never queried

Formal definition:

$$\text{MAC is } (t, q, \varepsilon, \mu)\text{-CMA-secure if:}$$

- Any adversary with running time ≤ t
- Making ≤ q queries
- On messages totaling ≤ μ bits
- Wins with probability ≤ ε

Key point: Even with many MAC examples, cannot forge new valid MAC

[TRANSITION] "How do we build CMA-secure MACs in practice? Two main approaches: CBC-MAC (with fixes) and HMAC. First, CBC-based..."

---

5. CBC-MAC and EMAC (10:00-12:30)

[BOARD] Write:

> **CBC-MAC:** Use final CBC ciphertext block as MAC

Basic CBC-MAC:

- Encrypt in CBC mode with $IV = 0$
- Output = final ciphertext block

[BOARD] Draw CBC chain ending with MAC

Critical vulnerability:

- **Only secure if no message is prefix of another**
- Have MAC for m → can compute MAC for $m||x$ for any extension x
- Length extension attack!

Solution: EMAC (Encrypted MAC)

[BOARD] Write:

> $\text{EMAC}_{K_1,K_2}(m) = E_{K_2}(\text{CBC-MAC}_{K_1}(m))$

Two-key construction:

1. Compute CBC-MAC using key K_1
2. Encrypt that MAC using key K_2 (single block encryption)
3. Output result

Why this works:

- Even if attacker extends message, cannot predict final encryption with K_2
- Two independent keys prevent extension attacks

Security theorem:

$$\varepsilon = 2\varepsilon' + \frac{2(\mu/k)^2 + 1}{2^k}$$

- If block cipher is (t', q', ε')-secure PRF with block length k
- Then EMAC is (t, q, ε, μ) CMA-secure

[TRANSITION] "CBC-MAC relies on block ciphers. Alternative approach: build MAC from hash functions directly. Most widely used: HMAC..."

---

6. HMAC (12:30-15:00)

[BOARD] Write:

> $\mathrm{HMAC}_K(m) = H((K \oplus \mathrm{opad}) \,||\, H((K \oplus \mathrm{ipad}) \,||\, m))$

Construction details:

- Based on any collision-intractable hash H (e.g., SHA-1, SHA-256)
- $\mathrm{ipad} = 363636\ldots36$ (hex constant)
- $\mathrm{opad} = 5C5C5C\ldots5C$ (hex constant)

[BOARD] Draw nested hash structure:

> **Outer:** $H(K \oplus \mathrm{opad} \,||\, [\text{inner hash}])$
>
> **Inner:** $H(K \oplus \mathrm{ipad} \,||\, m)$

Why this design:

- Hash applied **twice** with key XOR'd with different constants
- Inner hash processes message with key
- Outer hash processes inner result with different key derivation
- Prevents attacks exploiting internal hash structure

Security properties:

- Secure in **random oracle model**
- Widely deployed: TLS, IPsec, SSH
- Efficient - no block cipher needed
- Works with any Merkle-Damgård hash

Advantages:

- Faster than encryption-based MACs in many cases
- No export restrictions (not encryption)
- Simple to implement on top of existing hash functions

Summary & Key Takeaways (15:00-15:30)

[BOARD] Write summary:

> 1. **Hash:** $\{0,1\}^* \to \{0,1\}^k$ (collision-intractable)
> 2. **MD construction:** extend fixed to arbitrary length
> 3. **MAC** = keyed authentication
> 4. **CMA security:** can't forge after seeing examples
> 5. **EMAC** = two-key CBC-MAC
> 6. **HMAC** = nested hash with key

Final points:

- Hash functions: compression with collision resistance
- MACs: symmetric authentication with shared key
- Two main constructions: block cipher-based (EMAC) vs hash-based (HMAC)
- Both CMA-secure when properly constructed
- Choice depends on available primitives and performance needs