Symmetric Authentication and Hash Functions

Definition of collision-intractable hash functions. Then a selection of: construction from discrete log, proof that collision-intractable implies one-way, construction and proof that we can get any size input from fixed size input. Finally, MAC schemes, definition of CMA security, CBCMAC and EMAC security result for EMAC. Maybe a brief mention of HMAC.

The "Elevator Pitch" (Synthesis)

Prompt: Explain the core purpose of symmetric authentication systems and hash functions in 3 sentences. Focus on the *problem they solve* and *why collision resistance matters more than preimage resistance*.

> [Write your answer here]

Core Vocabulary & Syntax (Total Recall)

> Define each term precisely using the source material. Include the mathematical notation or context where applicable.

Hash functions, collision-intractability and proof that this implies one-way

> A hash function is defined by a *generator* $\mathcal{H}$, which on input a security parameter k outputs the description of a function that maps any bit string to a string of length k.
>
> $$h : \{0,1\}^* \to \{0,1\}^k$$
>
> By "description" we mean information allowing us to compute the function efficiently (formally speaking, in polynomial time in k and length of the input string). Note that there is no secret key involved here - the function we are using is public.

> One use case is where a data string m is to be stored in an insecure location. We then compute the authenticator $h(m)$ and store it in a secure location. Later, we retrieve a (possibly different) string m', we then compute $h(m')$ and compare to $h(m)$. This is why we want a fixed length output: it's easier to store a short fixed size value securely. The value $h(m)$ is also called a *message digest* or a *message fingerprint*.

> What properties do we need for such a function in order for the system to be secure?
>
> A *second preimage attack*:
> Given the function h and the message m, an adversary must find a different message m' such that $h(m) = h(m')$.

A *collision attack*: The adversary must find *any single* pair of messages $m \neq m'$, such that $h(m) = h(m')$ – a **collision**. (If they can trick the honest user into storing m, they can later change this to m' without being detected).

Second preimage attack $\geq$ collision attack, using the same amount of time and with the same success probability: just choose m yourself and then run whatever algorithm you have for a second preimage attack. Therefore, the best security is obtained, if even a collision attack is infeasible.

Collision intractability:
Consider the game where we run $\mathcal{H}$ on input k to get function h. We give h as input to adversary algorithm A, who outputs 2 strings m, m'.
We say that A has success if $m \neq m'$ and $h(m) = h(m')$.
We say that $\mathcal{H}$ is collision intractable, if any PPT (probabilistic, polynomial-time) algorithm A has success with negligible probability (as a function of k).

If we can invert a hash function, we can also find a collision:

1. The inversion algorithm can choose a random m and give $h(m)$ as input to A,
2. A returns m'. If A has succeeds (so $h(m') = h(m)$), and if $m' \neq m$, then we have a collision. If not, we can keep repeating this until we do.

A collision for h can be found in time t plus one evaluation of h and with probability at least $\epsilon/2 - 2^{-k-1}$ if m is $2k$ bits long. (This comes from the fact that $P[m \text{ has no siblings that would map to the same input}] \leq 2 - k$)

Therefore, because being able to invert a function $\implies$ finding a collision, if we cannot find don't have a way to find a collision $\implies$ we certainly cannot invert the hash function.

Domain extension (Merkle-Damgård construction)

When constructing collision intractable hash functions, it is always enough build collision-intractable functions that map a fixed number of bits, say m bits to k bits, as long as $m > k$.

This is because if there exists a collision-intractable hash function generator $\mathcal{H}'$ producing functions with finite input length $m > k$, then there exists a collision-intractable generator $\mathcal{H}$ that produces functions taking arbitrary length inputs.

One way to do this is the Merkle-Damgård construction, which underlies many popular hash functions including MD5, SHA-1, and SHA-2. This starts from the given $f : \{0,1\}^m \to \{0,1\}^k$ (where $m > k + 1$), and builds a hash function $h : \{0,1\}^* \to \{0,1\}^k$. It follows these steps:

1. Split into Blocks - Split the input message x into v-bit blocks x_i (where $v = m - k - 1$), padding the last block with \$0\$s if needed. Add an extra block x_{n+1} containing the number of \$0\$s used in padding.
2. Initialization - Set $z_1 := 0^k||1||x_1$ (using a "1" separator bit).

3. Iterative Hashing - For $i = 2, \ldots, n+1$, compute $z_i := f(z_{i-1})||0||x_i$ (using a "0" separator bit).
   Each iteration combines the result of applying f to the previous state (k bits), a separator bit, and the next message block (v bits) to produce a new m-bit input for the next round.
4. Output - $h(x) := f(z_{n+1})$

SHA-3 uses a newer method, the sponge design.
Here, the construction maintains a state which are carried forward from the previous iteration. The output does not fully match the internal state of the construction at the end, so you couldn't just continue where you left off (get $\hat{h}(m||x)$) (prevents *length extension attacks* possible with some hash functions).

MAC

How do we make sure that a message the receiver gets is the same as the one sent by the sender?

Yeah, hash codes - but what if the MiM adversary can modify both the messages and the authenticators?
(This will be the case when a sender communicates with a receiver over an insecure channel.)

When the sender and receiver can share a secret key:

1. An authenticator s can be computed from both data m and the secret key k.
2. m, s is sent on the channel, and m', s' is received.
3. The receiver can, using the *same* key k, test if m', s' look OK.

Of course, this should always be the case if $m', s' = m, s$, and we hope that anyone who does not know k cannot come up with a fake pair m', s' that the receiver would accept. Such systems can be built using *Message Authentication Codes* (MAC's) also called *Keyed Hash Functions*.

> The version where sender and receiver cannot have a shared secret, (like public-key crypto,) is *Digital Signature Schemes* (out of scope now).

A *secret-key authentication system* consists of three probabilistic algorithms (G, A, V).

G which outputs a key K, usually it works by simply choosing K as a random bit string of a certain length.
Algorithm A gets input a message m and the key K and produces an authenticator value $s = A_K(m)$ (aka a MAC).
Finally algorithm V gets as input an authenticator s, a message m and key K, and outputs $V_K(s, m)$ which is equal to *accept* or *reject*.

It is required that we always have $V_K(A_K(m), m) = accept$.

CMA Security

> Given a system (G, A, V), adversary E plays the following game with an oracle that holds key K from G:
>
> E may send any number of messages m to the oracle, receiving back $A_K(m)$s (a chosen message attack, CMA).
> E wins by **outputting a new message** m_0 and authenticator s_0 where $V_K(s_0, m_0) = accept$.

> Note: For deterministic MAC schemes (where each message has one valid MAC), the adversary can verify candidates by querying the oracle. Otherwise it does not have access to V_K.

> **Definition**: A MAC scheme is (t, q, ε, μ) CMA-secure if any adversary running in time $\leq t$, making $\leq q$ queries on messages totaling $\leq \mu$ bits, wins with probability $\leq \varepsilon$.

CBC-MAC and EMAC (MACs from Block Ciphers)

CBC-MAC Construction

> **Basic idea**: Encrypt the message in CBC mode (with $IV = 0$), and use the *final ciphertext block* as the MAC. This gives a fixed-length output regardless of message length.

> **Problem**: Only secure if no message is ever a prefix of another message.
> *Why?* If you have MAC for message m, you can compute MAC for $m||x$ for any extension x.
>
> **Attempted fixes**:
>
> - Prepending length: *Secure* but impractical (need to know message length before starting).
> - Appending length: Practical but *insecure* (attackable even with this fix).

EMAC (Encrypted MAC) - The Solution

> **Construction**: Use *two* independent keys K_1, K_2:
>
> $$\text{EMAC}_{K_1,K_2}(m) = E_{K_2}(\text{CBC-MAC}_{K_1}(m))$$
>
> 1. Compute CBC-MAC of message using key K_1
> 2. Encrypt that MAC using key K_2 (single block encryption)
> 3. Output the result

> **Security Result**:
> If the block cipher is a (t', q', ε')-secure PRF with block length k, then EMAC is (t, q, ε, μ) CMA-secure where:
>
> $$\varepsilon = 2\varepsilon' + \frac{2(\mu/k)^2 + 1}{2^k}$$
>
> (As long as μ isn't too large, EMAC inherits most of the cipher's strength.)

> **Proof intuition**:

1. Replace E_{K_1} and E_{K_2} with truly random functions (distinguishing advantage $\leq 2\varepsilon'$).
2. Show EMAC with random functions is close to a totally random function (advantage $\leq 2(\mu/k)^2/2^k$).
3. Against a totally random function, adversary wins with probability $\leq 1/2^k$ (must guess correct value for new message).
4. Triangle inequality gives total bound.

HMAC (MACs from Hash Functions)

Construction: Based on any collision-intractable hash function H (e.g., SHA-1):

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

where $\text{ipad} = 363636\ldots36$ and $\text{opad} = 5C5C5C\ldots5C$ (in hex).

Intuition: Apply the hash function *twice* with the key XOR'd with different constants.

- Inner hash: combines key with one constant and the message
- Outer hash: combines key with different constant and inner hash result

Why not just $H(K||m)$?
Many hash functions (like SHA-1) use iterative compression functions. Applying the key twice with different XOR values prevents attacks that exploit the internal structure.

Security:

- Provably secure in the *random oracle model* (where the hash function is modeled as a truly random function)
- In practice: secure if hash is collision-resistant AND compression function is secure as MAC
- Popular due to efficiency and no reliance on block ciphers (which may have export restrictions)

Logic Flow / Mechanism (Process)

A. Discrete Log-Based Hash Function Construction

You are given the discrete log hash function: $h(m_1, m_2) = \alpha^{m_1}\beta^{m_2} \bmod p$

Task: Reconstruct the proof that if discrete log is hard, then h is collision-intractable.

1. Assume an adversary finds a collision: $(m_1, m_2) \neq (m_1', m_2')$ with $h(m_1, m_2) = h(m_1', m_2')$.
2. [_____] (What equation do you now have?)
3. [_____] (What case analysis must you perform?)
4. [_____] (How do you isolate α in terms of β?)

5. Conclude: You have computed the discrete logarithm of α base β, contradicting the hardness assumption.

B. Merkle-Damgård Domain Extension (Case: $m - k > 1$)

You have a collision-intractable function $f : \{0,1\}^m \to \{0,1\}^k$ where $m > k$.

Task: Fill in the missing steps of the construction that produces $h : \{0,1\}^* \to \{0,1\}^k$.

1. Set $v = m - k - 1$. Split input x into v-bit blocks: $x_1, \ldots, x_n$.
2. [_____] (What is the padding step?)
3. Define the starting state: $z_1 = 0^k$ 1 x_1. For $i = 2, \ldots, n+1$, define [_____].
4. [_____] (What is the final output?)
5. **Proof sketch:** If $h(x) = h(x')$ for $x \neq x'$, show that you can work backwards through the sequences z_i and z'_i to find a collision for f. Why is the single bit in position $k+1$ critical?

C. Proving Collision-Intractable Implies One-Way (Lemma 11.2)

Task: Complete the logical chain.

1. Assume you have an algorithm A that inverts h with probability ϵ in time t.
2. [_____] (What is the collision-finding algorithm?)
3. [_____] (Define "only child" and compute the probability that a random m is an only child.)
4. [_____] (Why does $P[C|G] \geq 1/2$ when m is not an only child?)
5. Conclude: $P[\text{collision}] \geq (\epsilon - 2^{-k})/2$, which is non-negligible if ϵ is non-negligible.

The "Exam Trap" (Distinctions)

Collision Resistance vs. Preimage Resistance

Instruction: Distinguish between these two security properties using the criteria below.

Criterion	Collision Resistance	Preimage Resistance
Definition	[Your answer]	[Your answer]
Adversary's goal	[Your answer]	[Your answer]
Which is stronger?	[Your answer]	[Your answer]
Why is collision resistance the standard?	[Your answer]	[Your answer]

Criterion	Collision Resistance	Preimage Resistance
Relationship	If you can do X, can you do Y?	[Your answer]

Hash Functions in Theory vs. Practice

Instruction: Explain why the theoretical definition of collision-intractable hash functions (Definition 11.1) requires a *generator* H rather than a single fixed function. Then contrast this with how SHA-3 is used in practice.

> [Write your answer here]

Exam Simulation

Question 1 (Presentation + Follow-up)

Scenario: You randomly select "Hash Functions" as your oral exam topic. You have 18 minutes to present.

Prompt:

- **Part A (Presentation):** Walk through the discrete log-based hash function construction. Explain why the collision-intractability of $h(m_1, m_2) = \alpha^{m_1}\beta^{m_2} \bmod p$ follows from the hardness of discrete log. Include the proof strategy.
- **Part B (Follow-up Question):** "Your construction only handles 2(k-1)-bit inputs. How do you extend this to arbitrary-length inputs, and why is the single bit in the Merkle-Damgård construction necessary?"

Question 2 (Proof Reconstruction)

Prompt: State and prove **Lemma 11.2**: "If a hash function $h : \{0,1\}^{2k} \to \{0,1\}^k$ can be inverted with probability ϵ in time t, then a collision can be found in time t plus one evaluation of h with probability at least $\epsilon/2 - 2^{-k-1}$."

Your proof should:

1. Describe the collision-finding algorithm.
2. Define the event "only child" and compute its probability.
3. Explain why $P[C|G] \geq 1/2$ (where C is collision and G is the event that m is not an only child and A succeeds).
4. Conclude with the final probability bound.

Question 3 (Conceptual Depth)

Prompt: "The random oracle model assumes the hash function behaves like a random function. Explain:

1. What is a random oracle, and how does it differ from a concrete hash function like SHA-256?
2. Why does the birthday paradox (Theorem 11.4) imply that $k \geq 160$ bits?
3. What does it mean for an application to be 'secure in the random oracle model,' and what are its limitations?"

Source Map

Source	Location	Coverage
CryptographyV6.pdf	Page 134	Definition 11.1: Collision-intractable hash functions
CryptographyV6.pdf	Page 135	Section 11.2.1: Discrete log and RSA-based constructions
CryptographyV6.pdf	Page 135	Section 11.2.2: Lemma 11.2 (collision-intractable implies one-way)
CryptographyV6.pdf	Pages 136–138	Section 11.2.3: Merkle-Damgård domain extension (Theorem 11.3)
CryptographyV6.pdf	Pages 138–139	Section 11.2.4: Hash functions in practice (SHA-1, SHA-3, birthday paradox)
CryptographyV6.pdf	Pages 139–140	Section 11.2.5: Random oracle model

---

Critical Gaps & Missing Content

Source data incomplete for:

- Detailed proofs of RSA-based hash functions (Exercise 11.1 is stated but not fully worked).
- Formal security definitions for MAC schemes (Section 11.3 is referenced but not included in the provided excerpt).
- CBCMAC and EMAC constructions and their security proofs (mentioned in learning objectives but not in provided text).
- HMAC construction and security analysis (mentioned in learning objectives but not in provided text).

Recommendation: Consult the full CryptographyV6.pdf for Sections 11.3 and 11.4 to complete your preparation on MAC schemes.