

Message Authentication and Hashing

Prepared by Dr. Sumaya Alkubaisi

FNT625
Cybersecurity Concepts

Lecture 9



Lecture

9

Learning Goals

By the end of this lecture, you will be able to:

- Explain hashing and its properties
- Understand message authentication & MACs
- Perform basic HMAC reasoning
- See how hashing enables digital signatures
- Identify secure vs insecure hash functions
- Understand hashing roles in fintech, blockchain, and API security

Message Authentication

Message Authentication means verifying that a message is genuine and hasn't been altered.

It focuses on two main goals:

- **Authenticity** – making sure the message really came from the claimed sender.
- **Integrity** – ensuring the message wasn't changed during transmission.
- **Freshness (optional)** – confirming the message is new and not a replay of an old message.

Examples:

- When a bank sends a transaction code, the system must ensure it truly came from the bank.

Message Authentication - Example

Imagine your friend sends you a message saying: “Meet me at 5.”

Message authentication ensures that:

- It was really sent by your friend (not someone pretending to be them).
- The message wasn’t changed on the way (so it doesn’t become “Meet me at 7”).
- It’s a fresh message and not an old one being resent to confuse you.

Simply:

Message authentication makes sure the message is real, untouched, and current.

Lecture

9

Why Do We Need Message Authentication?

Without authentication, attackers can:

- Modify transactions
- Inject fake messages
- Manipulate API calls
- Replay old commands
- Impersonate senders

Modern systems (banking apps, blockchains, payments, e-commerce) depend on strong authentication to prevent fraud.

Authentication vs Encryption

Encryption → protects confidentiality (hiding the content)

Authentication → protects integrity + verifies sender

A message can be:

- Encrypted but not authenticated → attacker can change ciphertext
- Authenticated but not encrypted → attacker sees but cannot alter
- Both (recommended) → prevents eavesdropping and tampering at the same time

Authentication Approaches

There are two main ways to authenticate messages:

Message Authentication Codes (MAC):

- Uses a shared secret key between sender and receiver.
- Ensures the message hasn't been changed.
- Symmetric and fast, but doesn't prevent the sender from denying they sent it.

Digital Signatures:

- Uses asymmetric keys (like RSA or ECDSA).
- Also ensures message integrity.
- Provides non-repudiation, meaning the sender cannot deny sending the message.

Key difference: MAC is faster but symmetric; digital signatures are slower but allow non-repudiation.

Components of Authentication

A secure authentication system has four main components:

- 1.Integrity:** Ensures the message hasn't been altered during transmission.
- 2.Authenticity:** Confirms the message really came from the claimed sender.
- 3.Non-repudiation:** If digital signatures are used, the sender cannot deny sending the message.
- 4.Replay protection:** Prevents attackers from resending old messages, using techniques like timestamps, nonces, or sequence numbers.

Example: A payment API checks timestamps and rejects any message that is too old, preventing attackers from reusing previous requests.

Threats to Message Integrity

- **Modification attack:** An attacker changes the message content during transmission, so the receiver gets a different message than intended.
- **Man-in-the-middle (MITM) attack:** The attacker intercepts communication between sender and receiver, possibly altering or injecting messages without either party noticing.
- **Replay attack:** The attacker resends a previously valid message to trick the receiver into accepting it again.
- **Forgery attack:** The attacker creates a fake message that appears valid, aiming to deceive the receiver.
- **Collision attack:** The attacker exploits weaknesses in a hash function to create two different messages that produce the same hash, undermining integrity checks.

Lecture

9

Message Authentication Without Encryption

Authentication doesn't require secrecy of the message—it ensures the message hasn't been altered and comes from a legitimate sender.

Example:

An API sends:

message = "amount=200"

MAC = HMAC-SHA256(message, secret-key)

Even if an attacker can read the amount, they cannot change it to, say, 300 without invalidating the MAC.

The receiver checks the MAC and will reject any tampered message.

So, the message can be public, but authentication still guarantees integrity and authenticity.

Lecture

9

Message Authentication with Encryption

When you use both encryption and authentication together, the message becomes:

- Confidential → no one can read it
- Authenticated → you know who sent it
- Integrity-protected → no one can change it

This gives full protection, modern systems use a method called Encrypt-then-MAC (ETM), which means:

- First encrypt the message → Then compute a MAC on the encrypted data

This is the strongest and safest way because:

- Even if someone sees the encrypted data, they cannot modify it
- The MAC ensures it wasn't tampered with
- The receiver decrypts only after checking the MAC, preventing attacks

Lecture

9

Recap of Authentication Purpose

Message authentication protects against:

- ✓ Altered messages
- ✓ Tampering
- ✓ Spoofing
- ✓ Impersonation
- ✓ Replay attacks

❖ Authentication = “Is this message valid and from the correct sender?”

Lecture

9

Hash Functions

- A hash function is a mathematical function that takes any input (text, file, password, etc.) and converts it into a fixed-size string of characters called a hash or digest. We use hashing to protect, verify, and identify data safely and efficiently.

Example

Input: "Hello"

Output (SHA-256):

A591A6D40B...

This 64 hexadecimal characters (0–9 and A–F) output is the hash value.

Cryptographic Hash Functions

A cryptographic hash function is a special type of hash function designed for security. To be secure, it must have the following properties:

- One-way → cannot reverse; there is no “undo” or decryption.
- Collision-resistant → cannot find two inputs with same output
- Fast to compute: It should quickly generate a hash even for large data.
- Avalanche effect → small input change → big output change

Examples of Common Cryptographic Hash Algorithms :

- SHA-256 (very common, used in Bitcoin, TLS)
- SHA-3 (newer standard)
- BLAKE2 (fast and secure)

Why Hashing Matters in Security

- Hashing is crucial in cybersecurity because it lets us turn any data (password, file, message...) into a fixed-size, unique fingerprint.
- This fingerprint makes it possible to verify integrity, detect changes, and secure sensitive data without exposing the original information.

Hashing is used for:

1) Password Storage

- Systems do NOT store passwords, only their hash — never store raw passwords.
- They store hashes such as:
 $\text{password_hash} = \text{SHA-256}(\text{password} + \text{salt})$
- If someone hacks the database, they see the hash — not the real password — making it harder to steal user accounts.

Why Hashing Matters in Security

2) Digital Signatures

Hashing ensures that what gets signed is small and tamper-proof.

If someone changes even one bit in the document, the hash changes completely.

3) Blockchain

Each block contains a hash of the previous block.

This links blocks together and prevents tampering — changing one block breaks all the following blocks.

4) Integrity Checks

Hashing lets us verify that a file or message hasn't been modified.

If the received hash matches the expected hash → the data is intact.

Why Hashing Matters in Security

5) Malware Detection

Antivirus software uses hashes (“signatures”) of known malware.

If a file’s hash matches a malicious signature, it’s detected immediately.

6) File Verification

When you download software, websites show a hash (like SHA-256).

You compute the hash of your downloaded file;

if it matches → file is safe and unmodified.

7) Message Authentication Codes (MAC)

MACs combine a secret key + hashing.

They ensure that a message is:

- authentic (came from the right sender)
- untouched (not modified during transmission)

Hash Function Examples

- SHA-256 → industry standard (Bitcoin, TLS, digital signatures)
- SHA-1 → broken, no longer secure
- MD5 → broken, easily collidable
- SHA-3 → modern secure alternative

We use SHA-256 in fintech due to regulatory strength.

One-Way Property

- A secure hash makes it impossible to retrieve original data from the hash.

It is one-way, meaning:

- You can easily compute the hash from the data.
- But you cannot reverse the hash to get the original data.

Example:

- Suppose a password is "MySecret123"
 - Its hash might be "ABCD1234"
 - Even if an attacker sees "ABCD1234", they cannot figure out "MySecret123" from it.
-
- ✓ No attacker can reverse it to find the password or message.
 - ✓ Hashing is like a fingerprint — unique, fixed-size, and impossible to reverse to the original object.

Collision Resistance

- A cryptographic hash must make it infeasible to produce collisions.
- A collision happens when two different inputs produce the same hash:

$$\mathbf{M1 \neq M2 \text{ but } H(M1) = H(M2)}$$

Collision resistance means:

- It should be computationally infeasible for an attacker to find such a pair.
- This property ensures the uniqueness of each hash.

Why it matters?

- If collisions are easy, an attacker could replace a legitimate message or file with a malicious one without changing the hash.
- This breaks digital signatures, integrity checks, and security protocols.

Example:

MD5 and SHA-1 are considered broken because researchers have found ways to produce collisions.

Using these algorithms today is unsafe for security-critical applications.

Avalanche Effect

The avalanche effect means:

- A tiny change in input causes a completely different hash output.
- Even changing one bit or one character makes the hash look totally unrelated.

Example:

Input: "hello" → SHA-256 → A591A6D...

Input: "Hello" → SHA-256 → 185F8DB3...

Notice: only the first letter changed from h → H, but the hash changed completely.

Importance in security

Detects tiny changes → any modification changes the hash completely.

Prevents predictability → attackers can't guess hash changes.

Ensures integrity → matching hashes = data intact; differing hashes = altered.

Lecture

9

Salting in Hashing

A salt is a random value added before hashing to prevent:

- Rainbow table attacks
- Hash comparison attacks
- Reuse of identical hashes for identical passwords

Example: Hash = SHA-256(password + random_salt)

❖ Rainbow Table Attack

A rainbow table attack uses a precomputed table of hashes to quickly reverse common passwords. Instead of guessing each password, the attacker looks up the hash in the table to find the original password.

Salting prevents this by making each password hash unique, even if the password is common.

Lecture

9

Message Authentication Code (MAC)

A MAC is a short piece of data (tag) attached to a message to ensure:

- Authenticity → the message came from the expected sender
- Integrity → the message was not altered during transmission

It uses:

- A message
- A shared secret key
- A hash algorithm

Output: “tag”

How MAC Works

1. Sender combines:
 - The message
 - A shared secret key
 - A hash algorithm
2. Computes the MAC → produces a tag → **tag = MAC(message, secret_key)**
3. Sender sends **message + tag** to the receiver
4. Receiver uses the same key and hash algorithm to recompute the MAC:
 - If the tags match → message is authentic and intact
 - If not → message was tampered with

The MAC depends on both the message and the secret key, so an attacker cannot forge a valid MAC without knowing the key.

Lecture

9

When MAC Is Used

MACs secure:

- Banking APIs → Verify that API requests come from legitimate clients (prevent tampering).
- Payment gateways → Ensure transaction data isn't altered during processing.
- SWIFT messages → Confirm authenticity and integrity of international bank messages.
- Secure messaging → Protect messages (like in WhatsApp or Signal) from being modified.
- IoT devices → Authenticate sensor data sent to servers, preventing fake readings.
- VPN protocols → Verify data packets are untampered while traveling over the network.

MACs are fast and strong → making them perfect for systems with large transaction volumes.

Limitations of Basic MAC

1. Shared key must remain secret

- Both sender and receiver use the same secret key.
- If the key leaks, attackers can forge MACs.

2. No non-repudiation

- Unlike digital signatures, a sender cannot be proven to have sent a message.
- Both parties can generate the same MAC, so you can't blame one party.

3. Sender cannot be distinguished

- Since the MAC is computed with a shared key, either party could have created it.

Lecture

9

HMAC (Hash-based MAC)

HMAC = MAC built using a cryptographic hash function.
Most common authentication tool today.

Example: HMAC-SHA-256.

Benefits:

- ✓ Very secure
- ✓ Fast
- ✓ Resistant to length-extension attacks
- ✓ Real-world applications: Used in APIs, TLS, banking apps

Lecture

9

HMAC Example

Algorithm: HMAC-SHA256

Inputs: 1. Message = "amount=100" 2. Secret key = "K123"

Step 1: Sender computes HMAC:

Tag = HMAC_SHA256("K123", "amount=100") → Tag = 4E920A... (example output)

Step 2: Sender sends message + tag to the receiver.

Step 3: Receiver recomputes HMAC using the same key:

- If the tag matches → message is authentic and untampered
- If the tag does not match → message was modified

Example attack:

- Attacker changes message to "amount=1000"
- Receiver computes HMAC → tag does not match
- Receiver detects tampering immediately

Why HMAC Is Strong

HMAC uses:

- Keyed hashing → Only someone with the secret key can generate a valid tag.
- Inner/outer pads → Adds extra mixing to prevent certain attacks.
- Hash function compression → Ensures even small changes in input produce very different tags.

Protects against:

- Forgery → Attackers cannot create a valid tag without the key.
- Tampering → Any change in the message changes the tag.
- Replay attacks → Adding a timestamp prevents attackers from resending old messages.

MAC vs Digital Signature

MAC

Symmetric

Fast

No non-repudiation

Shared secret key

Digital Signature

Asymmetric

Slower

Provides non-repudiation

Public/private key pair

- MAC is preferred for speed.
- Signatures are preferred for identity proof.

Lecture

9

Using Hashing + MAC Together

Modern secure systems combine:

- SHA-256 (hash)
- HMAC (authentication)
- AES (encryption)

Used in:

1. TLS handshake

- Securely establishes encryption keys between client and server.
- HMAC ensures messages during handshake are authentic and intact.

Using Hashing + MAC Together

2. Payment processing

- Protects transaction messages from tampering.
- Ensures that payment instructions are genuine and unmodified.

3. Blockchain validation

- Hashing ensures each block's data is tamper-evident.
- HMAC can verify messages between nodes or secure off-chain data.

4. Token verification (JWT uses HMAC)

- Verifies that a token (JWT) was issued by a trusted source.
- HMAC ensures the payload hasn't been altered.

Lecture

9

Digital Signatures & Hashing

How Hashing Enables Digital Signatures?

1. Hash the message

- Instead of signing the entire message, compute a hash of the message: $H(M)$
- This produces a fixed-size fingerprint of the message.
- Even a tiny change in the message completely changes the hash.

2. Encrypt the hash with the sender's private key

- The encrypted hash becomes the digital signature.
- Only the sender with the private key can create this signature.

3. Receiver verifies using the sender's public key

- Receiver decrypts the signature → gets the hash.
- Receiver also hashes the received message → compares it with the decrypted hash.
- If they match → message is authentic and intact.

This method: **Efficient:** Only the small hash is signed, not the full message.

Secure: Hash ensures tamper detection, encryption ensures authenticity.

Lecture

9

Why Sign the Hash Instead of the Message?

Much faster → hashing produces a small fixed-size digest instead of signing a large message.

Consistent length → signature size depends only on the hash/key, not message length.

Protects integrity → any change in the message changes the hash, which invalidates the signature.

Efficient for large data → no need to process the entire message with expensive signing algorithms.

Used in: RSA, ECDSA, blockchain, and most digital signature systems.

Lecture

9

Integrity Attacks on Hashing

Attackers may try to:

- Find collisions → create two different messages with the same hash.
- Replace message & signature → swap a legitimate message with a tampered one.
- Exploit weak algorithms → use broken hashes like MD5 or SHA-1 to bypass integrity checks.

Defense: Modern systems use SHA-256 or SHA-3 to ensure strong collision resistance and message integrity.

Hashing in Blockchain (Overview)

- **Blockchain:** A secure, decentralized ledger of transactions where each block contains transaction data and a link to the previous one using cryptographic hashes.

Features:

- **Immutable:** Once added, a block cannot be changed without affecting all following blocks.
- **Transparent:** All participants can verify transactions.
- **Tamper-proof:** Hashes and consensus prevent fraud.

Finance Applications:

- Cryptocurrencies (e.g., Bitcoin, Ethereum)
- Cross-border payments and settlements
- Auditing and record-keeping

Lecture

9

Hashing in Blockchain (Overview)

Hashing is core to blockchain security because:

- Each block's hash depends on the previous block → creates a chain of blocks.
- Guarantees immutability → changing one block changes all following hashes.
- Prevents tampering → any modification is immediately detectable.
- Proof-of-work uses hashing → miners repeatedly hash data to meet difficulty targets (e.g., starts with many zeros).

Example: Bitcoin uses SHA-256 to compute block hashes.

Lecture

9

Hash Chains

- Hash chains are a way to link values sequentially using hashes:

$H1 = H(M1)$

$H2 = H(H1 + M2)$

$H3 = H(H2 + M3)$

- Each hash depends on the previous hash + new message → tampering is easy to detect.

Uses:

- TOTP (Time-based One-Time Passwords) → each password derived from previous hash
 - Blockchain → linking blocks securely
 - Version control integrity → ensure files or commits haven't been altered
- Hashing + hash chains ensure security, integrity, and tamper-resistance in blockchain and other systems.

Lecture

9

Hands-On Exercise (Optional)

- Compute SHA-256 hash:

Python example:

```
import hashlib
```

```
print(hashlib.sha256(b"Hello").hexdigest())
```

- You can experiment with changing one letter to see avalanche effect.

Lecture

9

Recap

- Hash functions provide one-way, collision-resistant integrity
- Message Authentication Codes (MAC/HMAC) provide integrity + authenticity
- Digital signatures provide integrity + authenticity + non-repudiation
- SHA-256 is industry standard; MD5 & SHA-1 are insecure
- Hashing is foundational for blockchain, password storage, API security
- HMAC is widely used in fintech systems (API, tokens, transactions)

References

- William Stallings, Cryptography & Network Security, 8th Ed., 2023
- William Stallings, Network Security Essentials, 2022
- Cybersecurity Tools and Techniques (2022)
- Computer Forensics & Cyber Crime (2020)
- NIST FIPS 180-4 (Secure Hash Standard)
- RFC 2104 (HMAC Standard)
- Research: Rivest (MD5), NIST SHA-2 & SHA-3 papers