Need for a Cryptographic Checksum

Symmetric Encryption

* Conventional symmetric encryption, A 🡪 B: Ek[M] **provides confidentiality**, as only A and B share K, and a **certain degree** of origin authentication.
* Only if B can understand the message, then it can confirm that indeed the message M has come from A.
* **Does not provide a signature** (A signature is unique) because B can also encrypt using the same key.
* Sender A could deny sending the message.

Asymmetric Encryption

* Public-key (asymmetric) encryption, A 🡪 B: Ekub[M]
* **Does not provide origin repudiation**
* **Provides confidentiality.**
* Checksum can provide digital signing for asymmetric encryption.
* A 🡪 B: M||Ekra [h(M)]. Encrypt the hash using A’s private key. Only A has KRa so only A could have sent the message. The authenticity of the sender, A, is confirmed using KUa. Need to make sure KUa is trustworthy and signature is dated. Also provides a means to check message integrity. And prevents replay of messages.
* Although provides no confidentiality.

Digest Functions

* AKA checksum, hash value or fingerprint
* Given a message M or arbitrary length, a Message Digest function, H, produces a fixed-size output, h.
* Many to one mapping.

MD Functions – Properties

* Compression: H can be applied to any sized M, and a fixed size output, h is produced.
* One-way property: easy to compute H(x), but hard to computer x such that H(x) = h
* Weak collision resistance: Given x, it is difficult to find y != x such that H(y) = H(x), output set is smaller than Strong collision resistance.
* Strong collision resistance: Given x, it is difficult to find y != x such that H(y) = H(x), output set is larger.
* Signature forgery is possible is weak collision resistance property is not met. For example: A🡪B M||Ekra[H(M)], if an attacker intercepts the message, and finds a M’ such that H(M) == H(M’), then the attacker can send the wrong message and B would get it thinking it is correct as the original hash would match M’ hash.
* Repudiation if strong collision property is not met: Not exactly sure on the example.

Construction Methods

* MAC (Message Authentication Code): Block cipher based
* Can be any block cipher function.
* Uses K and MACing function, f, to generate a checksum MAC = fk(M)
* Short digest length



* Send M||MAC, M
* Receiver computes the MAC of the message and compares with the one sent. If same M has not been tampered with.
* Provides **integrity protection**
* Provides **origin authentication**
* Does not provide **Non-repudiation.**
* If a **timestamp or random number (contributed by b)** is sent then the message is **fresh. Meaning not re-sent.**
* Using a timestamp to ensure freshness is not easy because **time is not central**. (look at the podcast)
* The size of the MAC is important. The more the bits the better.
* Finding collisions cost **2^n/2**. Where n is the bit length of the MAC.

HMAC – Hash MACing

* Keyed hash function
* H = any hash function such as SHA-256
* K+ is the key padded out to block size
* Ipad is a string repeating the byte 0x36
* Opad is a string by repeating the byte 0x5c
* HMAC is good because it means **anyone cannot just generate it** without the secret key.

Authenticated Encryption

* Provides message authentication and confidentiality.
* Approaches:
  + 1. Hash-then-encrypt: E(K, (M||H(M)))
    2. MAC-then-encrypt: E(K2, (M||MAC(K1, M))), need to decrypt once.
    3. Encrypt-then-MAC: C = E(K2, M), T = MAC(K1, C): More secure then MAC-then-encrypt. Need to decrypt twice.
    4. Encrypt-and-MAC: C = E(K2, M), Tag = MAC(K1, M), M is encrypted although MAC is not (non-confidential).

Q. Why are there 2 keys resulting from MACing and what is K2 used for? Could possibly be there receiver’s key.