Length Extension Attacks

Attacker doesn't know k||x

- **Length extension attack**: Given $H(\overset{\nearrow k||x}{x})$ and the length of $\overset{k||x}{x}$, but not *x*, an attacker can create $\underline{H(\overset{k||x||m}{x \,||\, m})}$ for any *m* of the attacker's choosing
  - [Length extension attack - Wikipedia](#)
- SHA-256 (256-bit version of SHA-2) is vulnerable
- SHA-3 is not vulnerable

sender

K

X →

H(K || X) →

Tell us: Implementation of Hash matters

HMAC. SHA3

receiver

K → recalculate

→ X → H(K, X)

→ H(K||X) → compare

if mach, x is not modified

else

An attack happened

Attacker

x

H(K||x)

x || m →

H(K || x || m) →

without knowing k, x

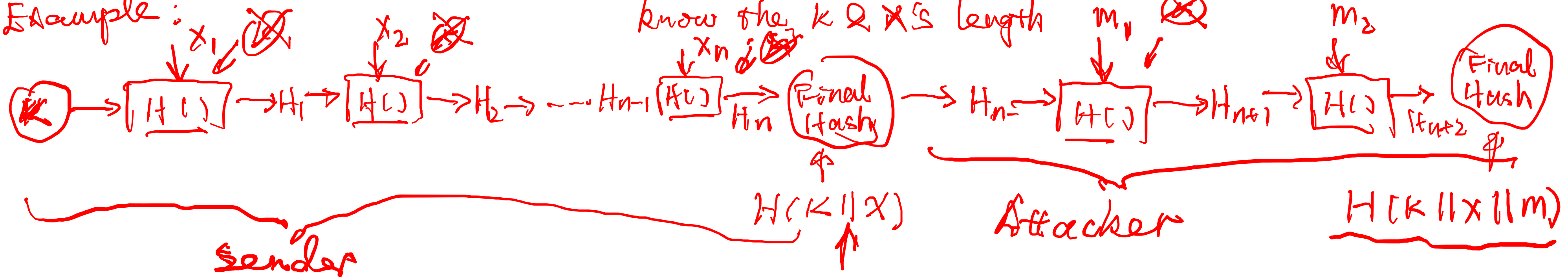
x || m

H(K || x || m) ← H ← recalculate

x = { x1, x2, -- xn }

Example:

know the k & x's length



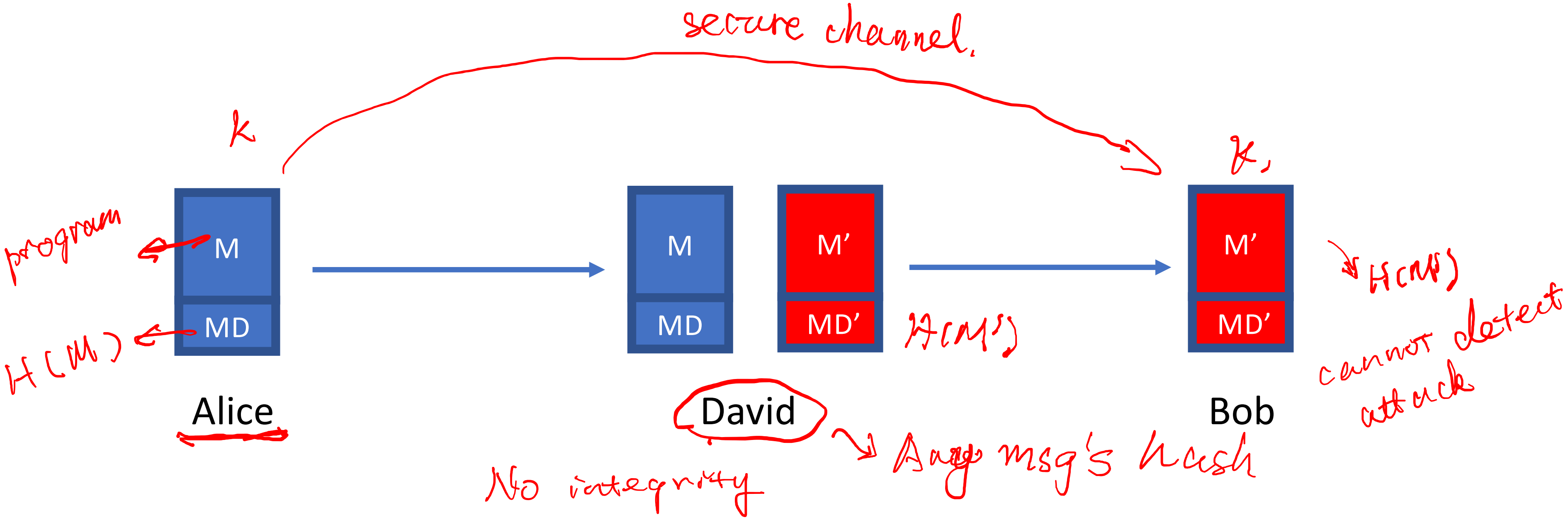
Does hashes provide integrity?

- It depends on your threat model
- Scenario
  - Mozilla publishes a new version of Firefox on some download servers
  - Alice downloads the program binary
  - How can she be sure that nobody tampered with the program?
- Idea: use cryptographic hashes
  - Mozilla hashes the program binary and publishes the hash on its website
  - Alice hashes the binary she downloaded and checks that it matches the hash on the website
  - If Alice downloaded a malicious program, the hash would not match (tampering detected!)
  - An attacker can't create a malicious program with the same hash (collision resistance)
- Threat model: We assume the attacker cannot modify the hash on the website
  - We have integrity, as long as we can communicate the hash securely

Do hashes provide integrity?

- It depends on your threat model
- Scenario
  - Alice and Bob want to communicate over an insecure channel
  - David might tamper with messages
- Idea: Use cryptographic hashes
  - Alice sends her message with a cryptographic hash over the channel
  - Bob receives the message and computes a hash on the message
  - Bob checks that the hash he computed matches the hash sent by Alice
- Threat model: David can modify the message *and the hash*
  - No integrity!

Man-in-the-middle attack



cannot detect attack

No integrity

Any msg's hash

Do hashes provide integrity?

- It depends on your threat model
- If the attacker can modify the hash, hashes don't provide integrity
- Main issue: Hashes are *unkeyed* functions
  - There is no secret key being used as input, so any attacker can compute a hash on any value

Solutions

- A message digest created using a secret symmetric key is known as a Message Authentication Code (MAC), because it can provide assurance that the message has not been modified
- The sender can also generate a message digest and then encrypt the digest using the private key of an asymmetric key pair, forming a digital signature. The signature must then be verified by the receiver through comparing it with a locally generated digest

Hashes: Summary

- Map arbitrary-length input to fixed-length output
- Output is deterministic
- Security properties
  - One way: Given an output *y*, it is infeasible to find any input *x* such that *H*(*x*) = *y*.
  - Second preimage resistant: Given an input *x*, it is infeasible to find another input *x'* ≠ *x* such that *H*(*x*) = *H*(*x'*). weak collision ↔ 5th
  - Collision resistant: It is infeasible to find any pair of inputs *x'* ≠ *x* such that *H*(*x*) = *H*(*x'*). strong collision → 6st
  - Randomized output uniform
- Some hashes are vulnerable to length extension attacks
- Hashes don't provide integrity (unless you can publish the hash securely)

reduce collision

→ MAC, Digital signature

Message Authentication Code

Message authentication code (MAC)

- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some key
  - not be reversible
  - $MAC_M = F(K_{AB}, M)$

symmetric

- appended to message as a signature
- receiver performs same computation on message and checks it matches the MAC

validating
non-repudiation of origin

- provides assurance that message is unaltered and comes from sender

why

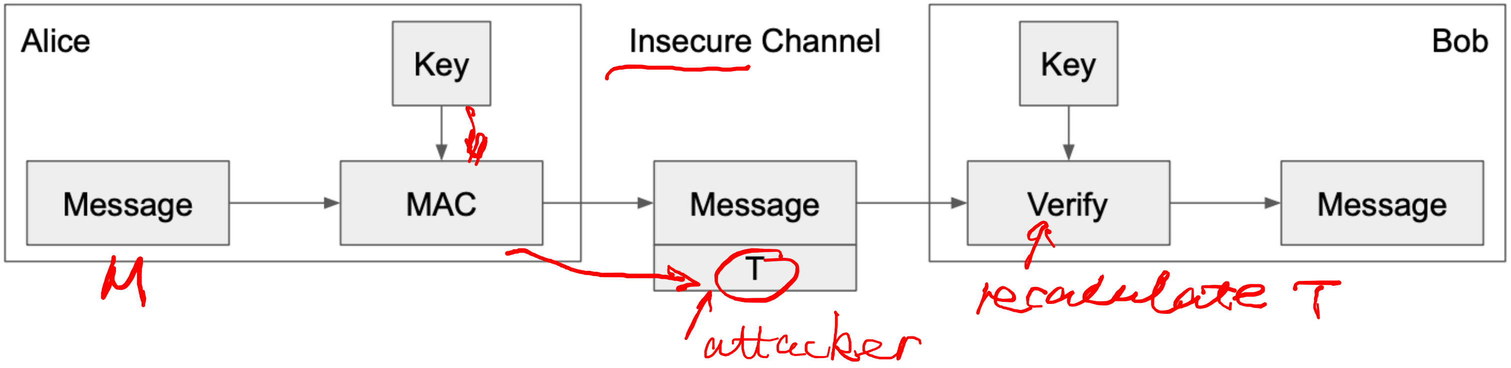
integrity

only sender & receiver have the key

MACs: Usage

attacker

- Alice wants to send *M* to Bob, but doesn't want David to tamper with it
- Alice sends *M* and *T* = MAC(*K*, *M*) to Bob
- Bob receives *M* and *T*
- Bob computes MAC(*K*, *M*) and checks that it matches *T*
- If the MACs match, Bob is confident the message has not been tampered with (integrity)



MACs: Definition

- Two parts:
  - KeyGen() → *K*: Generate a key *K* PRNG
  - MAC(*K*, *M*) → *T*: Generate a tag *T* for the message *M* using key *K*
    - HMAC.
    - Inputs: A secret key and an arbitrary-length message
    - Output: A fixed-length **tag** on the message
- Properties
  - **Correctness**: Determinism
    - Note: Some more complicated MAC schemes have an additional Verify(*K*, *M*, *T*) function that don't require determinism, but this is out of scope
  - **Efficiency**: Computing a MAC should be efficient fast cost effective.
  - **Security**: existentially unforgeable under chosen plaintext attack

Attacker: plaintext & cyphertext

M T

Mid-term Exam

- Nov. 6, 2024 (Wednesday), 12:00 pm – 12:50 pm, in class
- Closed book, but you're allowed to bring one cheat sheet (1 A4-sized paper)
- Chapter 1 – 3
- Will have a review class & quiz. on Nov. 1st , during class