Lec 11: Hash and MAC

CSED415: Computer Security

Spring 2024

Seulbae Kim

POSTECH

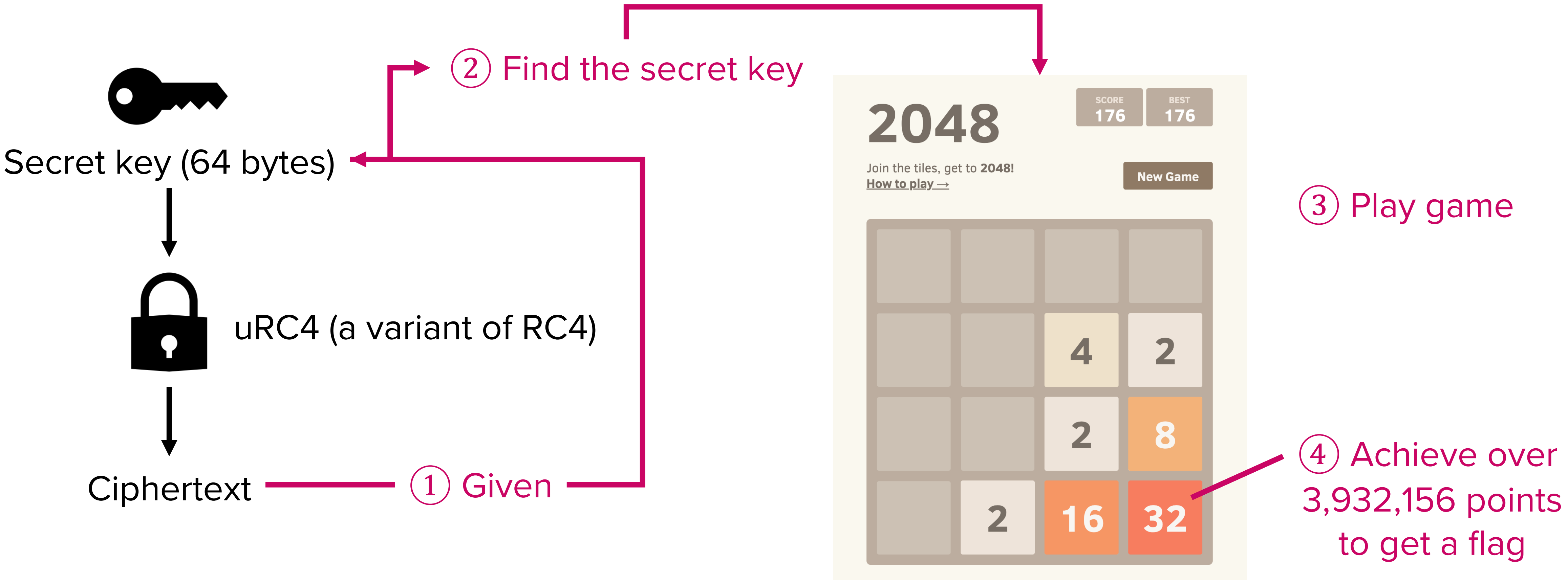
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Administrivia

- Lab 03 is out!
  - Due Sunday, April 7
  - Breaking a faulty cryptographic scheme and a game

Lab 03 overview

Phase 1: uRC4 service (server.py)	Phase 2: target



Cryptography roadmap

Goal \ Scheme	Symmetric Key	Asymmetric Key
Confidentiality	✅ One Time Pad (OTP) ✅ Block ciphers (DES, AES) ✅ Stream ciphers	✅ ElGamal encryption ✅ RSA encryption
Integrity & Authentication	• Message Authentication Code (MAC)	• Digital signature

Tools
- ✅ Secure key exchange
- Hash

Hash Functions

Missing integrity

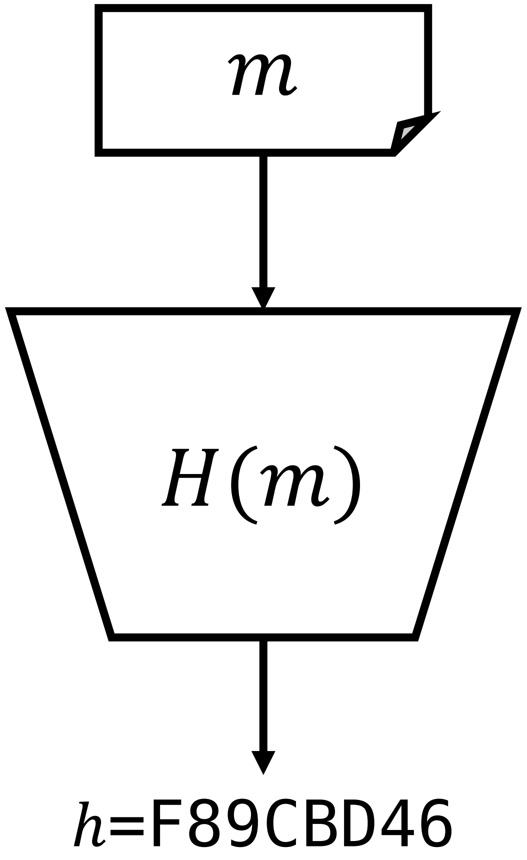
- Enc/decryption does not provide integrity (Lec 9 and 10)

- How can we allow Alice and bob verify that their messages have not been tampered with?
  - i.e., how to verify $c_1 == c_2$?

Hash functions

POSTECH

- Hash function H
  - Takes a message m of arbitrary length
  - Creates a message digest h of fixed length
    - h is also called hash, hash value, hash digest, …
- Required properties
  - Correctness: Deterministic outcomes
    - Hashing m should always produce the same h
  - Efficiency: Efficient to compute $H(m)$



Arbitrary length input to fixed length output

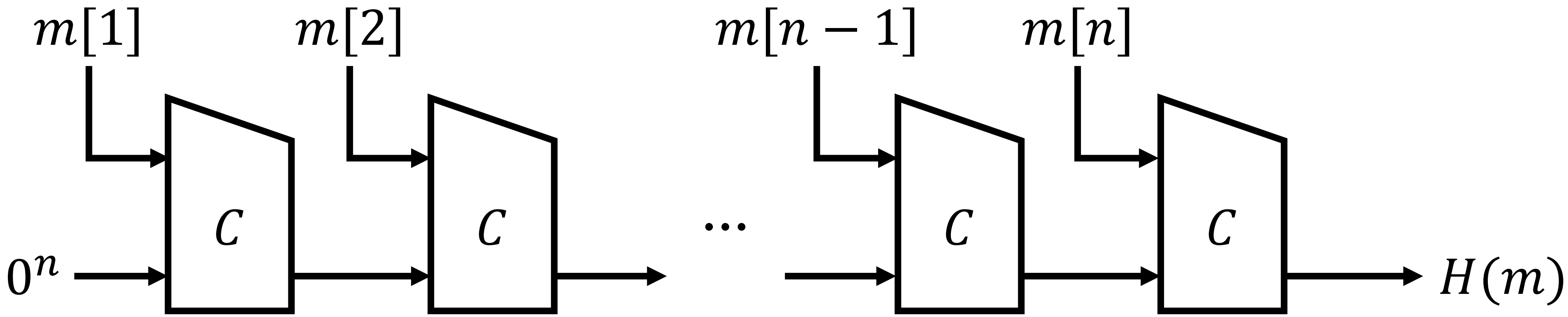
- MD5: 128-bit hash function (produces 16-byte hash digests)
  - “a” → 0cc175b9c0f1b6a831c399e269772661
  - “aa” → 4124bc0a9335c27f086f24ba207a4912
  - “a”*2048 → b7ea2d21ad2ef3e28085d30247603e0b

arbitrary length input

fixed length output

Merkle-Damgård Transform (1979)

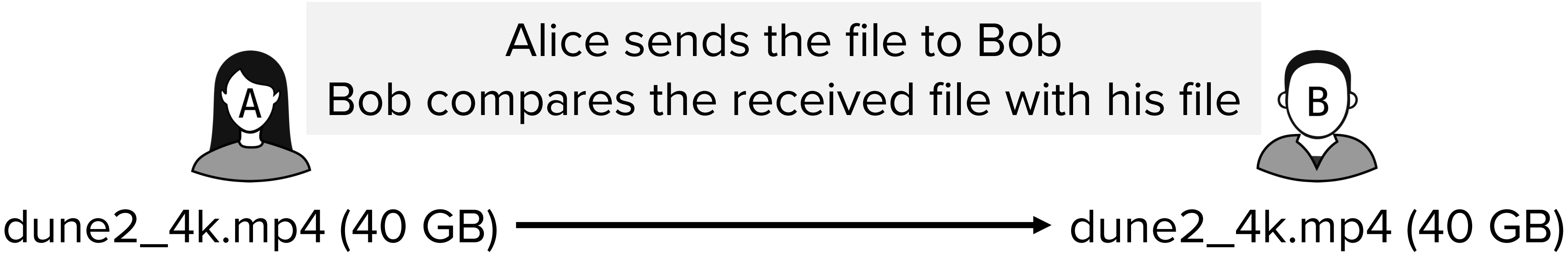
- Used by many hash functions, including MD, SHA-1 and SHA-2 families
  - Build hash function H by chaining a compression function C



C always outputs a fixed length output

Typical usage of hash function

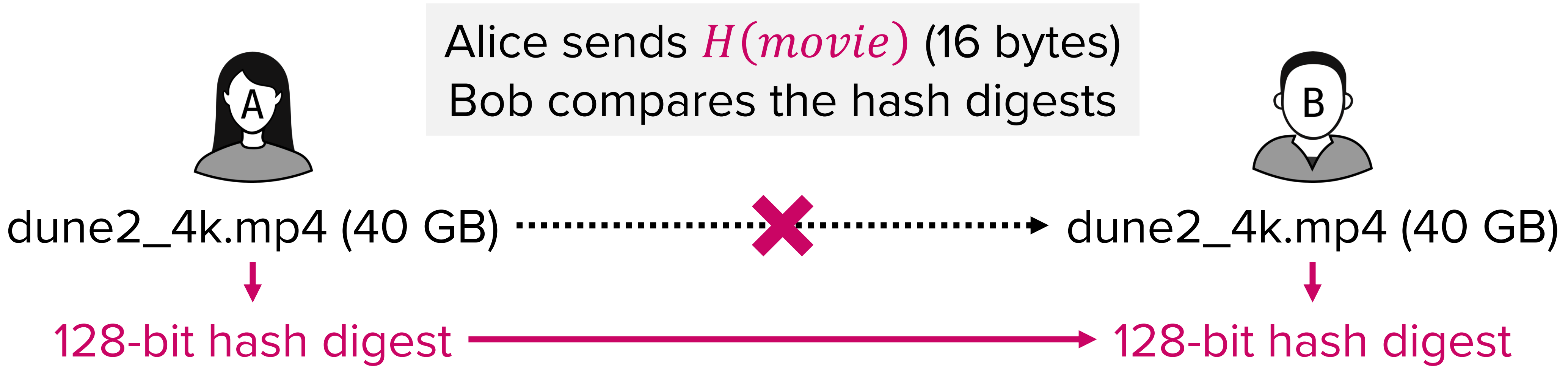
- Scenario: File integrity verification
  - Alice and Bob both downloaded a 40-GB movie file from the internet
  - They want to verify if the two files are identical
  - Naïve way:



→ Waste of bandwidth and computational powers

Typical usage of hash function

- Scenario: File integrity verification
  - Alice and Bob both downloaded a 40-GB movie file from the internet
  - They want to verify if the two files are identical
  - Using hash:



Cryptographic hash functions

- Hash with additional requirements for security
  - One-wayness (OW)
    - For any given hash value h,
      it is computationally infeasible to find m such that $H(m) = h$
  - Collision resistance (CR)
    - It is computationally infeasible to find a pair of plaintexts m_1 and m_2
      such that $H(m_1) = H(m_2)$

One-wayness (OW)

- Informally:
  - Given an output h, it is infeasible to find input m such that $H(m) = h$
- Formally:
  - H is one-way, if for all polynomial time adversary A who randomly selects m' from the plaintext domain,

$$Adv_H^{ow}(A) = Prob[H(m') = h] \text{ is negligible}$$

"Advantage"

- Common misconception (beware):
  - "A hash function is one-way because the mapping is many-to-one"
    → Wrong! Totally different concept from OW

OW examples

OW: Is $Adv_H^{ow}(A) = Prob[H(m') = h]$ negligible?

- Is $H(m) = 0$ a one-way hash function?

  No. A can easily find multiple m's. $Prob[H(m') = 0] = 1$

- Is the following summation checksum one-way?

Message		Ascii-hex format
C S E D	→	43 53 45 44
4 1 5 0	→	34 31 35 30
Checksum:		77 84 7A 74

No. A can easily find "CSED4150" (and other m's) given 77847A74

OW examples

OW: Is $Adv_H^{ow}(A) = Prob[H(m') = h]$ negligible?

- If H and G are length-preserving hash functions that are OW, is $F(x) = H(x) \oplus G(x)$ one-way?

Collision resistance (CR)

- Collision: Two different inputs results in the same output
  - $m_1 \neq m_2$ and $H(m_1) = H(m_2)$

- Can a hash function have no collision?
  - No. If the input domain is larger than 2^n for a n-bit hash function, there must be collisions (by the pigeonhole principle)
  - Collision resistance is not about having no collisions.
    It makes **finding collisions infeasible** for adversaries

Collision resistance (CR)

- Informally:
  - It is computationally infeasible to find a pair of plaintexts m_1 and m_2 such that $H(m_1) = H(m_2)$
- Formally:
  - H is **collision-resistant**, if for all polynomial time adversary A,

$Adv_H^{cr}(A) = Prob[H(m_1) = H(m_2)]$ is negligible where $m_1 \neq m_2$

"Advantage"

CR example

POSTECH

CR: is $Adv_H^{cr}(A) = Prob[H(m_1) = H(m_2)]$ negligible where $m_1 \neq m_2$?

- Let $H: \{0,1\}^{256} \rightarrow \{0,1\}^{128}$ be defined by

$$H(x) = H(x_L || x_R) = AES(x_L) \oplus AES(x_R)$$

(|| means concatenation)

  - Q) Is H collision-resistant?

A generic attack for finding collisions

- Birthday problem
  - Choose a group of N random people
  - What's the probability that at least one pair of individuals have the same birthday?
- Birthday pardox:
  - If N=23, the odds that two people share the same birthday is 50%
    - Event E: Birthdays of 23 people are different → ${}_{365}P_{23}$
    - Possible outcomes: Each people have 365 choices → 365^{23}
    - $P(E) = \frac{{}_{365}P_{23}}{365^{23}} \approx 0.492$
    - Therefore, prob. that at least two people share the b-day: $1 - P(E) \approx 50\%$

A generic attack for finding collisions

- Birthday attack
  - Similarly, the probability of detecting a hash collision is much higher than the expectation (e.g., brute-forcing)
  - Approximation
    - When there are 2^n possible data,
      if we have $\sqrt{2^n}$ data, the probability of collision is > 50%
    - In other words, finding a collision of a n-bit hash function requires $\sqrt{2^n}$ trials
    - 365 days → $n = 9$ bits → $\sqrt{2^9} = 22.67$ → approximately 23 trials for 50% chance

A generic attack for finding collisions

- Collision-resistance of a n-bit hash function is bounded by $\sqrt{2^n}$
- Cryptanalysis of hash functions

Function	n	Trials needed by birthday attack	Existing attacks
MD4	128	2^{64}	< sec
MD5	128	2^{64}	1 min
SHA-1	160	2^{80}	2^{69} trials (2005)
SHA-1	160	2^{80}	$2^{63.1}$ trials (2017)
SHA-256	256	2^{128}	-

Attacks requiring less trials than B-day attack are considered feasible attacks

MD5: An old standard without CR

- Developed by Ron Rivest in 1991
- Generates 128-bit hash digests
- Various severe weaknesses have been discovered
- Chosen-prefix collisions attacks (Marc Stevens, et al.)
  - Start with two arbitrary plaintexts m_1 and m_2
  - One can compute suffixes s_1 and s_2 such that $md5(m_1||s_1) = md5(m_2||s_2)$ in 250 trials
  - Using this approach, a pair of different files (e.g., jpeg) with the same MD5 hash value can be computed

Collision in practice – MD5 is completely broken

- Download ship.jpg and plane.jpg from https://natmchugh.blogspot.com/2015/02/create-your-own-md5-collisions.html

```
import hashlib

f1 = open("ship.jpg", "rb").read()
f2 = open("plane.jpg", "rb").read()

print(hashlib.md5(f1).hexdigest())
print(hashlib.md5(f2).hexdigest())
```

Both files are hashed to `253dd04e87492e4fc3471de5e776bc3d`

CR vs OW

- Does collision-resistance imply one-wayness?

- Does one-wayness imply collision-resistance?

CR vs OW

- Does collision-resistance imply one-wayness?
  - It does not

    e.g., $H(x) = x$ is CR, but not OW

- Does one-wayness imply collision-resistance?
  - It does not

    e.g., $H(x)$ is a good hash, which is OW.

    Notation: $x = x_0 x_1 x_2 \dots x_n$ (x_i: i-th bit of x)

    $G(x) = H(x_0 x_1 \dots x_{n-1})$ (ignores the last bit)

    → $G(x)$ is still OW. Hard to find x from $G(x)$

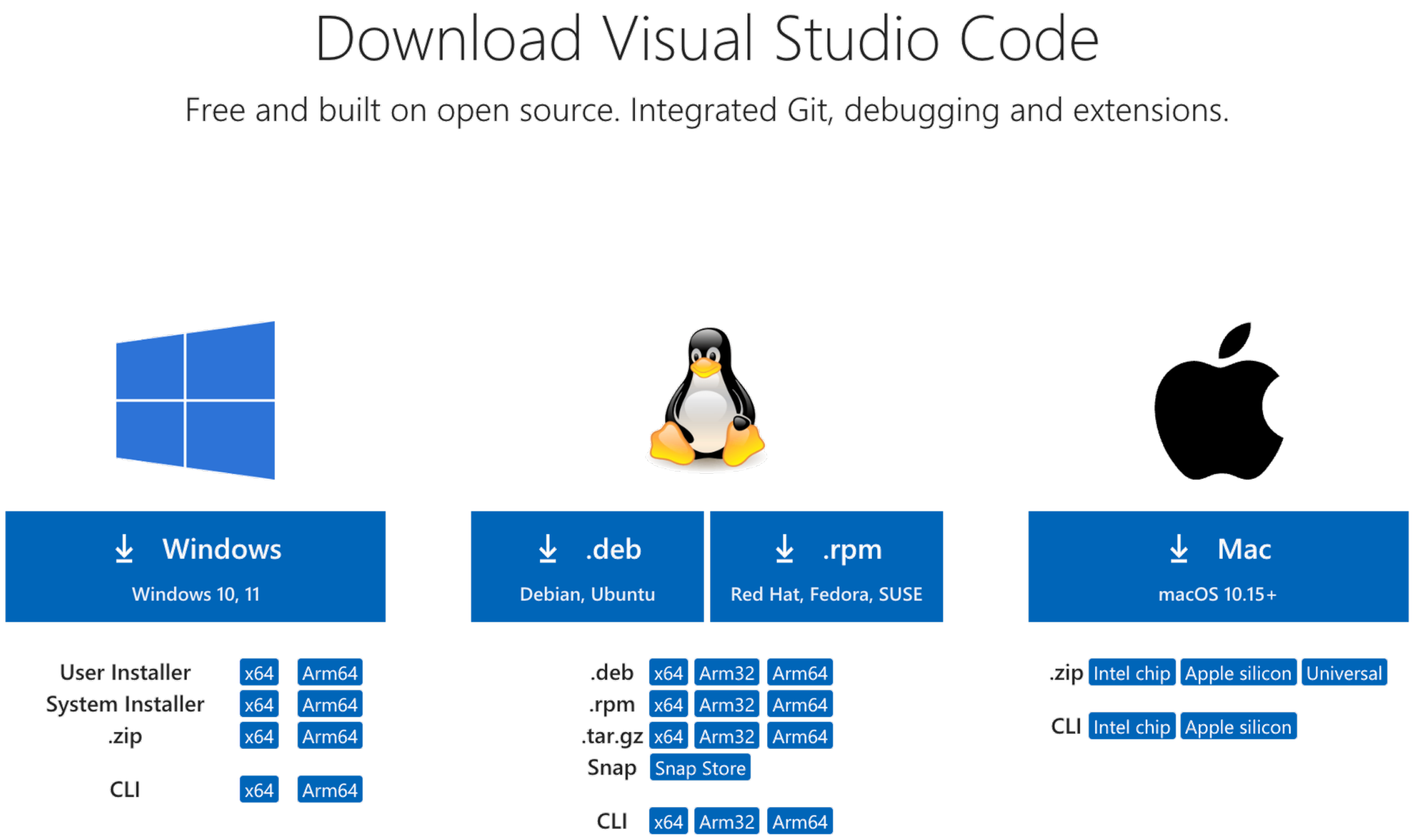
    → $G(x)$ is not CR. $H(x_0 x_1 \dots x_{n-1} 0) = H(x_0 x_1 \dots x_{n-1} 1)$

Using hash functions for integrity

- Scenario
  - Microsoft publishes a new version of vscode
  - Alice downloads the installer
  - How does she verify that nobody tampered with the installer?

Using hash functions for integrity

- e.g., vscode download site



Using hash functions for integrity

- e.g., vscode download site

SHA-256 hashes

Windows User Installer (x64)	7bda1c7dfc670489155db2f8fc1f48c209b92fb6145a320d677dccf0bce921b6
Windows System Installer (x64)	c49f51562a99e19412d968a81ad653960c4861e95f7cd04e49e15c42e139a9ee
Windows .zip (x64)	564d545cc1099bcb48c7eb5b5efb292d7dea2e02a37d8bd84a907e171f3092ce
Windows CLI (x64)	c306eb45d0ef485885308090c66f1a0328aece3ccdb4cc1554a7b3ad54f639e7
Windows User Installer (Arm64)	c91bd092b71c3d948bb8f32fc5f83e454f4ec90eee7b0e9cf58decf880fea54e
Windows .zip (Arm64)	a63c75550322fca979e672d09cc46385d02d1e7a9d07f12b2b078af4f4005478
Windows System Installer (Arm64)	63178497481ddf816396566904e99b4b3a817637f1c9170255fa294babed9f79
Windows CLI (Arm64)	0d8ded98088669219b52784f48c0b4f2364dbefd104c87dcfbf048827880fe8a
Linux .deb (x64)	3340b2649e486adfde2452418599acb64c1dc3998087d715d244f10302a89b94
Linux .rpm (x64)	841f72255270b647c657f6a20728d271cf08f94a07b7625fc91b548545efac8b
Linux .tar.gz (x64)	c2e97cdc63ff1bcbfbb10c227b5398623d21f21e487108fa1d740dabe7d37985
Linux CLI (x64)	1cb4ee01e6941b369c69253f12ff0eed15071221c7f16858a49694cd981bfb6c

Using hash functions for integrity

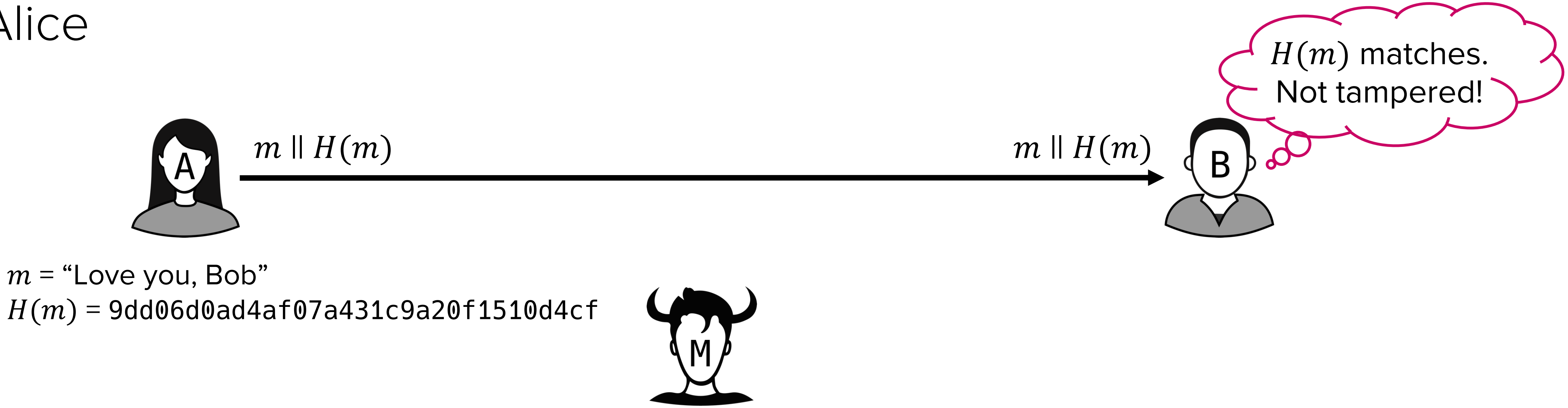
- Method
  - Microsoft hashes the installer binary with SHA-256 and publishes the hash on its website
  - Alice hashes the installer binary she downloaded with SHA-256 and checks if the hash matches the hash on the website
- Security
  - If Alice downloaded a malicious program, the hash would not match
  - An attacker cannot create a malicious program with the same hash as the original installer (SHA-256 is collision-resistant)

Using hash functions for integrity

- Another scenario
  - Alice and Bob want to communicate over an insecure channel and verify integrity of their messages
  - Mallory can tamper with the messages

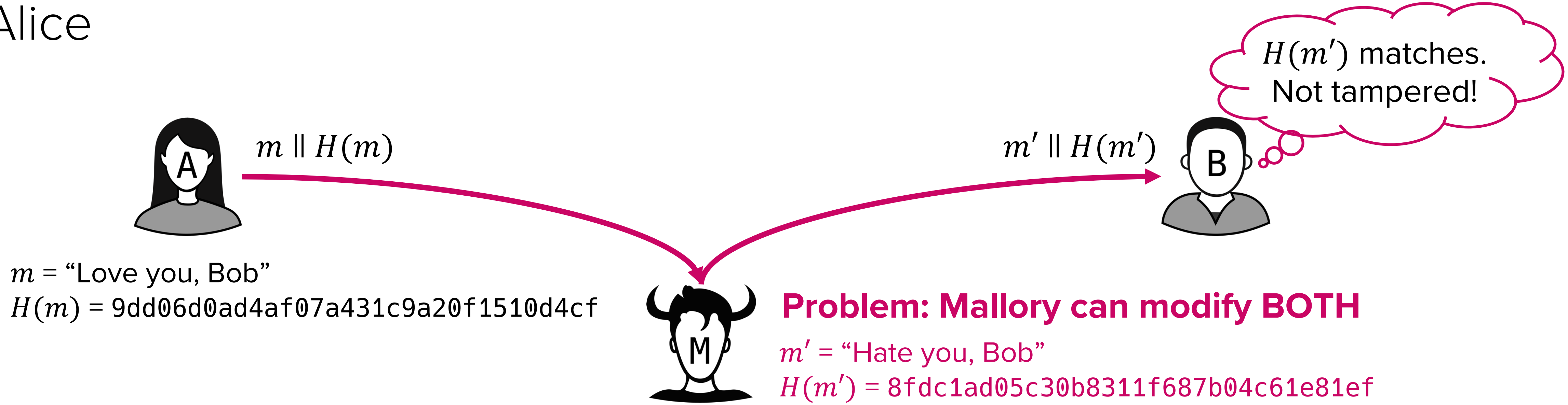
Using hash functions for integrity

- Method
  - Alice sends her message with its hash digest over the channel
  - Bob receives the message and computes a hash of the message
  - Bob verifies that the hash he computed matches the hash sent by Alice



Using hash functions for integrity

- Method
  - Alice sends her message with its hash digest over the channel
  - Bob receives the message and computes a hash of the message
  - Bob verifies that the hash he computed matches the hash sent by Alice



Do hash functions provide integrity?

- Depends on the threat model
  - MS website → hash cannot be modified by Mallory
  - Communication → Mallory can modify hash

- Main issue: Hash functions are unkeyed functions
  - No secret key is used as input for hash functions, so any attacker can compute the hash of any value

How do we utilize hash to design schemes that provide integrity?

Cryptography roadmap

Goal \ Scheme	Symmetric Key	Asymmetric Key
Confidentiality	✅ One Time Pad (OTP) ✅ Block ciphers (DES, AES) ✅ Stream ciphers	✅ ElGamal encryption ✅ RSA encryption
Integrity & Authentication	• Message Authentication Code (MAC)	• Digital signature

Tools

✅ Secure key exchange

✅ Hash

Message Authentication Code (MAC)

Goal: Providing integrity

POSTECH

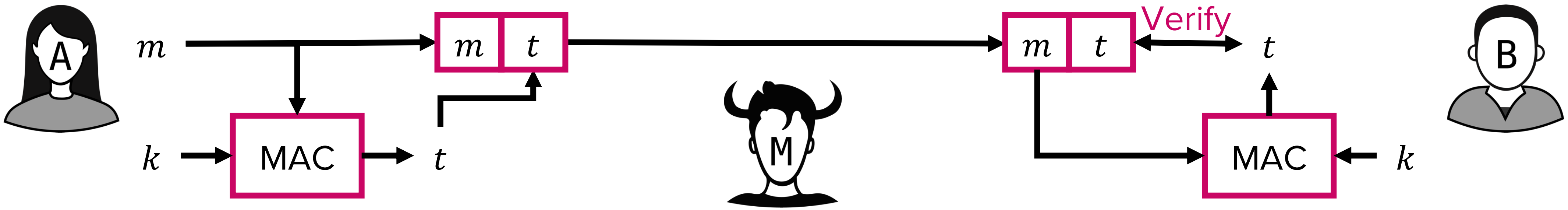
- Reminder: We are in the symmetric-key setting
  - Alice and Bob share a secret key
  - Attacker does not know the key

- Idea: Attach some piece of information to verify that someone with the key is the sender of a message

Message Authentication Code (MAC)

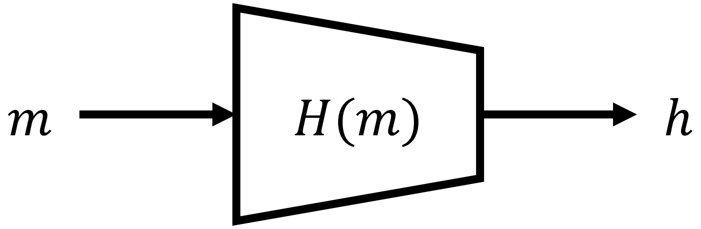
POSTECH

- Designed to provide both integrity and authenticity
- Setting
  - Alice sends message m and tag $t = MAC(k, m)$ where k: secret key
  - Bob recomputes $MAC(k, m)$ and verifies if the result matches t
  - If the MACs match, Bob is confident that m has not been tampered with

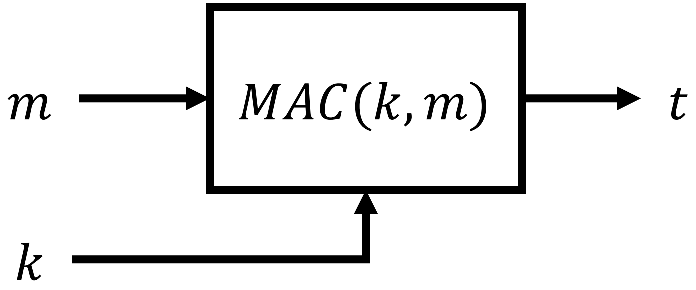


Hash function vs MAC

- Hash: Keyless



- MAC: Keyed

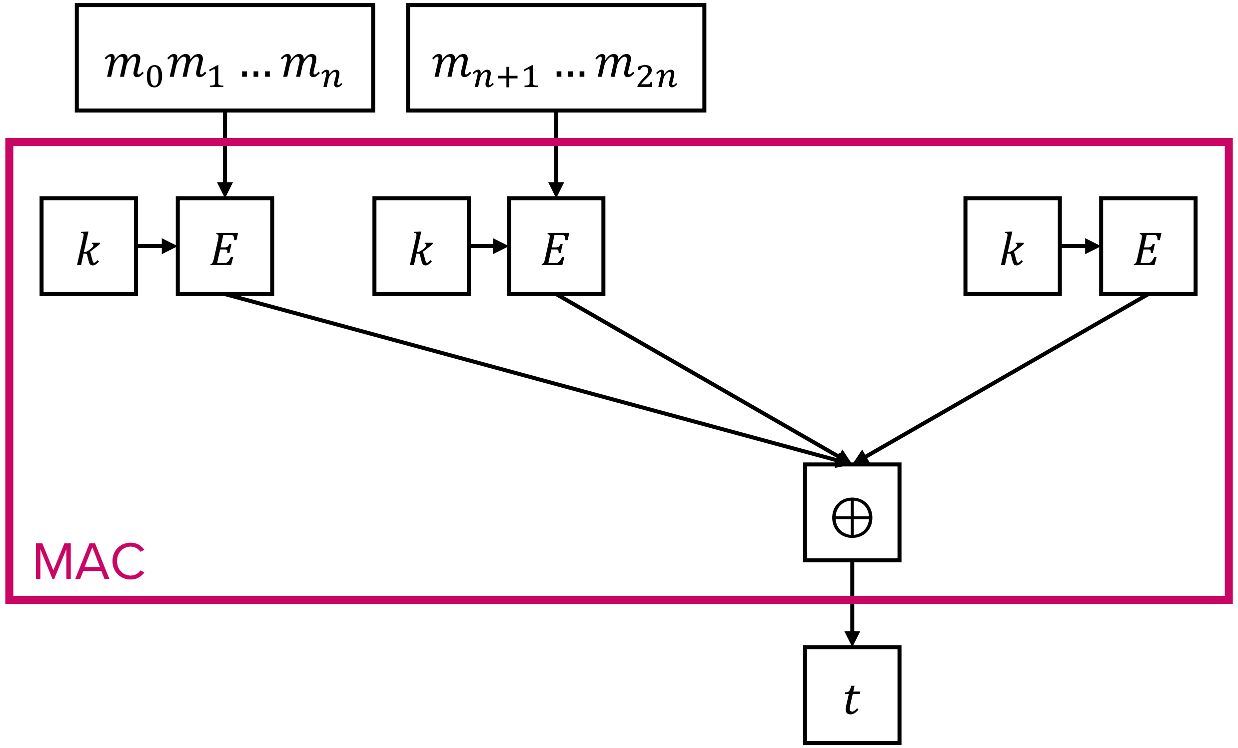


Evaluating the security of MAC

- “Unforgeability”: MAC is unforgeable under chosen msg if
  - A polynomial time adversary can see some number of (m, t) pairs
  - Without knowing the key k,
    it is infeasible to find a message m and its MAC tag t
    such that $t = MAC(k, m)$

Evaluating the security of MAC

- Example: Block-cipher-based MAC
  - E is a n-bit block cipher using key k



Is this MAC unforgeable?

(0000...01111....1)

1. Adversary selects plaintext $0^n||1^n$ and obtains $t = MAC(k, 0^n||1^n)$
2. Adversary found $m = 1^n||0^n$ and its tag t such that $t = MAC(k, 1^n||0^n)$

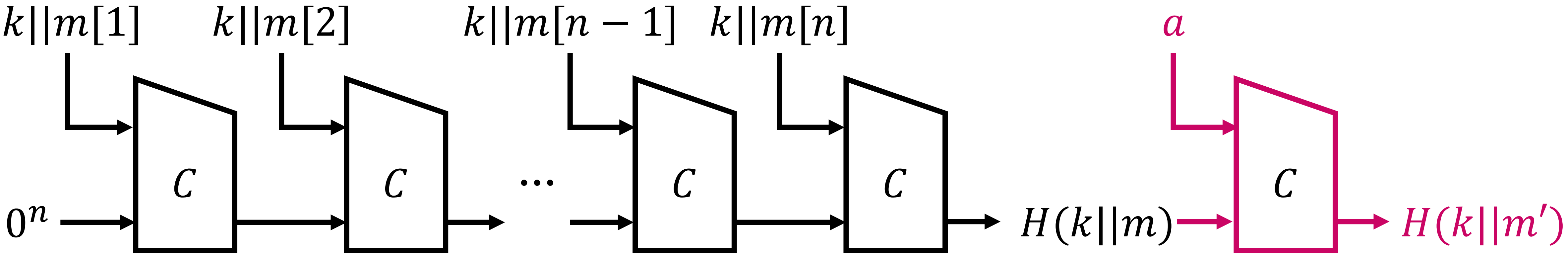
→ Not unforgeable (i.e., no integrity)

Constructing MAC using hash functions

- Secret prefix construction: $H(k||m)$ (|| means concatenation)
- Secret suffix construction: $H(m||k)$
- Nested MAC (NMAC): $H\left(k_1 \,||\, H(k_2||m)\right)$
- Hash-based MAC (HMAC): $H\left(k' \oplus opad \,||\, H(k' \oplus ipad \,||\, m)\right)$

Constructing MAC using hash functions

- Secret prefix construction: $H(k||m)$
  - Recall: Merkel-Damgård transform



[Length extension attack]
Given: m and $H(k||m)$.
Attacker can append a to m to get $m' = m||a$
Attacker can use $H(k||m)$ to initialize the computation of $H(k||m') = H(k||m||a)$

Constructing MAC using hash functions

- Secret suffix construction: $H(m||k)$
  - No known attack for secret suffix construction
  - However, its unforgeability is not proven

Constructing MAC using hash functions

- Nested MAC: $H\left(k_1 \mid\mid H(k_2 \mid\mid m)\right)$
  - Nesting two hashes prevents a length extension attack
  - If two keys (k_1 and k_2) are different, NMAC is provably secure (unforgeable)
  - Issues with NMAC
    - Need two different keys (weaker security)
    - Two keys need to be the same length as hash digest (constraint)

Constructing MAC using hash functions

- Hash-based MAC (HMAC): $H\left(k' \oplus opad \mid\mid H(k' \oplus ipad \mid\mid m)\right)$
  - Improvement over NMAC
  - k': n-bit version of k where n is the length of hash digest
    - If k is smaller than n bits, $k' = k||0^{n-|k|}$, i.e., pad k with 0's to make it n bits
    - Otherwise, $k' = H(k)$, i.e., hash k to make it n bits
  - Two different keys can be derived from k'
    - Outer pad ($opad$): 0x5c repeated until the length becomes n bits
    - Inner pad ($ipad$): 0x36 repeated until the length becomes n bits
  - Two rounds of hashing with two keys

Evaluating the security of HMAC

- Hash-based MAC (HMAC):
  - $H\left(k' \oplus opad \mid\mid H(k' \oplus ipad \mid\mid m)\right)$
  - HMAC is unforgeable under chosen message attack
    - A polynomial attacker cannot create m and valid $t = HMAC(k, m)$ without knowing the secret key k (proof omitted)

- HMAC is one of the most widely standardized and used cryptographic constructs

Cryptography roadmap

POSTECH

Goal \ Scheme	Symmetric Key	Asymmetric Key
Confidentiality	✅ One Time Pad (OTP) ✅ Block ciphers (DES, AES) ✅ Stream ciphers	✅ ElGamal encryption ✅ RSA encryption
Integrity & Authentication	✅ Message Authentication Code (MAC)	• Digital signature

Can we achieve both?

Tools
- ✅ Secure key exchange
- ✅ Hash

Authenticated Encryption

Confidentiality and integrity/authenticity goals

- Encryption schemes provide confidentiality, but not integrity
- MACs provide integrity/authenticity, but not confidentiality

→ Can we achieve both?

Authenticated encryption (AE)

- Definition
  - A scheme that simultaneously guarantees confidentiality and integrity of a message

- Existing building blocks for AE:
  - $E(k_1, m)$ and $D(k_1, m)$
    - e.g., AES
  - $MAC(k_2, m)$
    - e.g., HMAC

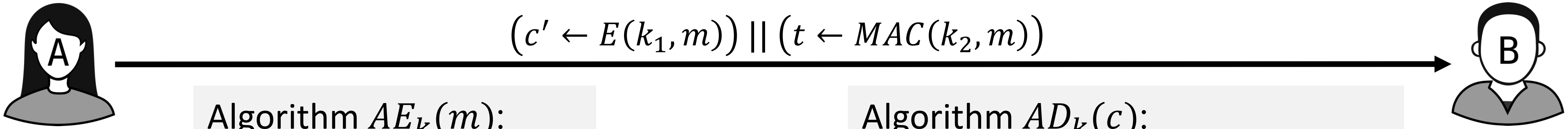
Building AE from existing primitives

POSTECH

1. Encrypt-and-MAC

Secure?

$\left(c' \leftarrow E(k_1, m)\right) \,||\, \left(t \leftarrow MAC(k_2, m)\right)$



$m, k = k_1 \,||\, k_2$

Algorithm $AE_k(m)$:
- $c' \leftarrow E(k_1, m)$
- $t \leftarrow MAC(k_2, m)$
- $c \leftarrow c' || t$
- Return c

Algorithm $AD_k(c)$:
- $m \leftarrow D(k_1, c)$
- If $\left(t = MAC(k_2, m)\right)$ Return m
- Else Return $NULL$

$k = k_1 \,||\, k_2$

No. Vulnerable to chosen-plaintext attacks ☹

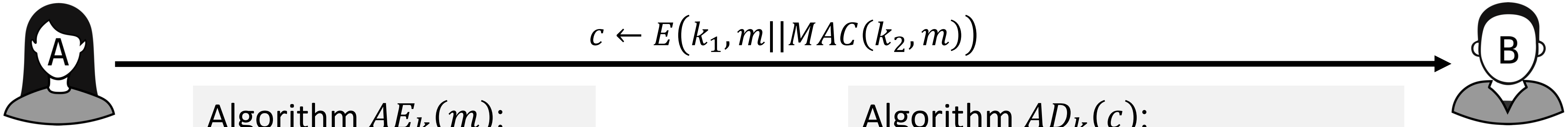
t is exposed as is. Attacker can observe t to check the equality of messages

Building AE from existing primitives

2. MAC-then-Encrypt

Secure?

$c \leftarrow E\big(k_1, m||MAC(k_2, m)\big)$



$m, k = k_1 \,||\, k_2$

Algorithm $AE_k(m)$:
- $t \leftarrow MAC(k_2, m)$
- $c \leftarrow E(k_1, m||t)$
- Return c

Algorithm $AD_k(c)$:
- $m||t \leftarrow D(k_1, c)$
- If $\big(t = MAC(k_2, m)\big)$ Return m
- Else Return $NULL$

$k = k_1 \,||\, k_2$

No longer vulnerable to chosen-plaintext attacks ☺

Integrity (unforgeability) is not guaranteed for some encryption schemes even if a good MAC is used ☹

→ Attackers can forge messages that are accepted by AD_k:

e.g., $E'(k, m) = E(k, m) \,||\, 0 = c'$

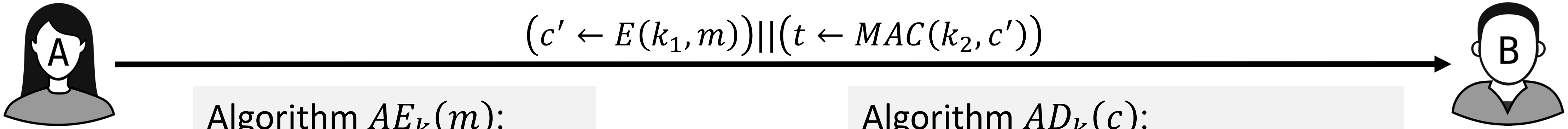
$D'(k, c') = D'(k, c \,||\, 0) = D(k, c)$

Building AE from existing primitives

POSTECH

3. Encrypt-then-MAC

Secure?



$(c' \leftarrow E(k_1, m)) || (t \leftarrow MAC(k_2, c'))$

$m, k = k_1 \,||\, k_2$

Algorithm $AE_k(m)$:
- $c' \leftarrow E(k_1, m)$
- $t \leftarrow MAC(k_2, c')$
- $c \leftarrow c' \,||\, t$
- Return c

Algorithm $AD_k(c)$:
- $c' || \, t \leftarrow c$
- $m \leftarrow D(k_2, c')$
- If $\left(t = MAC(k_2, c')\right)$ Return m
- Else Return $NULL$

$k = k_1 \,||\, k_2$

Not vulnerable to chosen-plaintext attacks ☺

Unforgeability is algo guaranteed ☺
(proof omitted)

Can check MAC first before decrypting (efficiency!)

Cryptography roadmap

Goal \ Scheme	Symmetric Key	Asymmetric Key
Confidentiality	✅ One Time Pad (OTP) ✅ Block ciphers (DES, AES) ✅ Stream ciphers	✅ ElGamal encryption ✅ RSA encryption
Integrity & Authentication	✅ Message Authentication Code (MAC)	• Digital signature
CIA at the same time	✅ Authenticated encryption	

Tools

✅ Secure key exchange

✅ Hash

Questions?