MONASH University
Information Technology

FIT2093 INTRODUCTION TO CYBER SECURITY

Week 5 Lecture

Information Integrity & Authentication

Principles for INTEGRITY


Outline

Integrity & Message Authentication

- Public-key Cryptographic Integrity Techniques
  - digital signatures
  - hash functions
- Symmetry-key Cryptographic Integrity Techniques
  - message authentication codes (MAC)
    - from block ciphers
    - from hash functions
  - authenticated encryption (AE), sign-then-encrypt

MONASH University

Public-key Cryptography for Integrity & AUTHentication

Recap: INT/AUTH goals/strategy

- INT/AUTH Aims:
  - **Integrity (INT)**: Received data not tempered with (**modified**)
  - **Authenticity (AUTH)**: Received data not fake (**fabricated**)

Problem: Insecure channel / storage

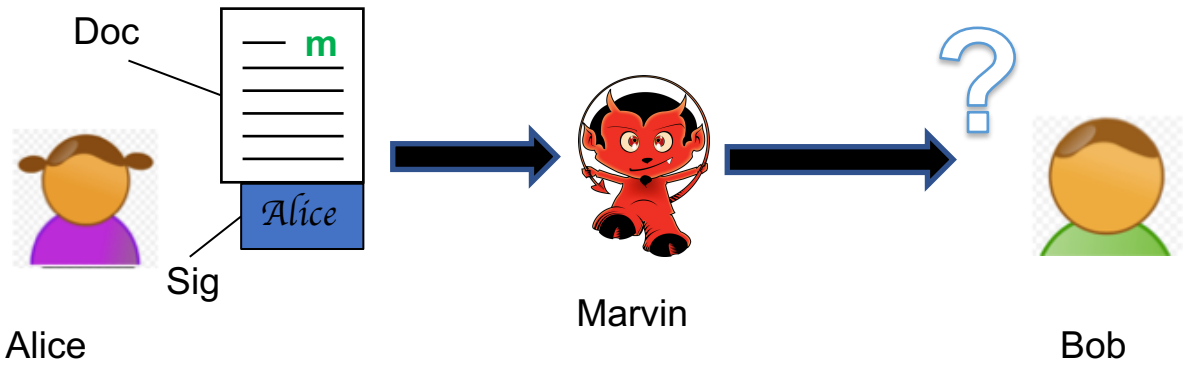
Strategy

- *accept* fact: channel insecure
  - whatever sent could be modified, fabricated data could be sent
- So: make sure mods/fakes are **detected** by receiver
  - receiver **rejects** the mods/fakes

Data INT/AUTH: How?

- **Requirements**
  - Only authentic sender can sign
    - infeasible for attacker to forge
  - Receiver can efficiently verify

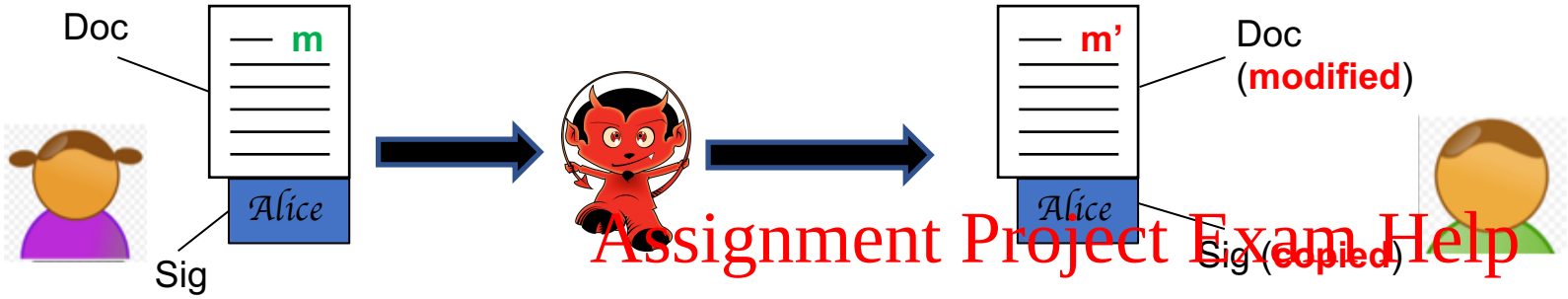
- Paper world analogy: hand-written signatures
  - **Q:** Why not adapt in the digital world?
    - Attach to document an image of handwritten signature
    - But… is it secure?



Q: How could Marvin, intercepting a document m with attached image of Alice's signature, produce a modified document m' accepted by Bob as authentic from Alice?

Data INT/AUTH: How?

- **Replay (copy)** attack with **unchanging (fixed)** signatures:



- Idea: to detect replay (copy) attacks:
  - make the **signature dependent on the signed document**!
  - copied sig. for m will **not be valid** for different m'

- **Q**: Modify ideas of PKE to build such a digital signature?
  - To prevent forgeries: sig. depends on sender's private key
  - To allow **anyone** to verify: verify with sender's public key

Recap: PKE

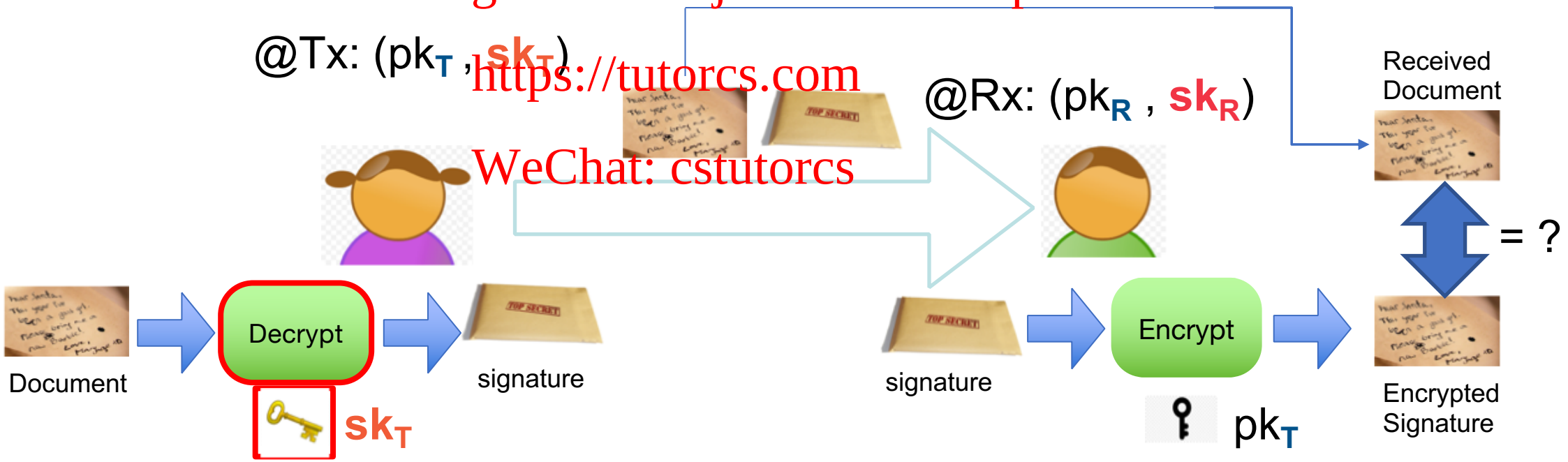
Integrity & Authentication

- Recall from last week: PKE for CONFidentiality
  - Q: use what key to encrypt so only recipient can undo (i.e. decrypt)?

PKC for AUTH & Integrity

Integrity & Authentication

- Idea: “dual” (inverted) use of keys for AUTH
  - **Q:** use what key to sign to ensure only originator can do (the signing)?
  - known as digital signature



Digital Signatures

Integrity & Authentication

- **AUTHentication:** verify msg really **came from sender**
  - use digital signature:
    - only sender could have generated signature,
    - because only sender has her private key

- **INTegrity:** verify msg has **not been modified by unauthorised parties**
  - use digital signature:
    - infeasible for attacker to modify msg together with valid sig
    - without knowing sender's private key


Digital Signatures: Examples

Integrity & Authentication

- **RSA Signature** [1977]
  - AUTHentication variant of RSA encryption
  - security based on **Integer Factorisation** Problem
- **Digital Signature Algorithm (DSA)** [ElGamal 1984, Schnorr 1991, NIST 1991]
  - AUTHentication variant of Diffie-Hellman key exchange/El-Gamal encryption
  - security based on **Discrete Logarithm** Problem
  - **Shorter/faster variant**: Elliptic Curve Digital Signature Algorithm (**ECDSA**) [ANSI 1998, NIST 2000]


Digital Signatures Example: RSA

Integrity & Authentication

- **Key Generation** for the Tx Alice:
  - choose two primes: $p = 5$, $q = 11$
  - get modulus: multiply p and q: $n = p \times q = 55$
  - And $\phi(n) = (p-1) \times (q-1) = 4 \times 10 = 40$
  - find two numbers **$e = 3$** & **$d = 27$** which satisfy

    $e \times d \bmod \phi(n) = 1$

    $(3 \times 27) \bmod 40 = 1$

  - Tx's Alice's public key
    - two numbers:

      (e,n) = (**3, 55**)
    - encryption algorithm: modular exponentiation
  - Tx's private key:

    (d, n) = **(27,55)**


Digital Signatures Example: RSA

Integrity & Authentication

- Alice has a document ***m* = 19** to **sign**:
  - use private key ***d* = 27** to calculate the digital signature of ***m* = 19**:

$$s = m^d \pmod{n}$$

$$= 19^{27} \bmod 55 = (19^3)^9 \bmod 55 = (31 \times 19)^9 \bmod 55$$

$$= (39^3)^3 \bmod 55 = (36 \times 39)^3 \bmod 55 = 29^3 \bmod 55$$

$$= (16 \times 29) \bmod 55$$

$$= 24$$

  - append 24 to 19. Now **(*m*, *s*) = (19, 24)**
  - the doc is 19, and Alice's signature on the doc is 24

Digital Signatures Example: RSA

Integrity & Authentication

- **Verification** by a verifier Bob:
  - receive a pair $(m,s) = (19, 24)$
  - look up the web directory and find out Alice's public key
  - $(e, n) = (3, 55)$
  - calculate $t = s^e \pmod n$

    $= 24^3 \pmod{55}$) = 26x24 mod 55

    $= 19$
  - check whether $t = m$
  - confirm that (**19,24**) is a **genuinely signed document of Alice** if $t = m$

MONASH University

Digital Signatures Example: RSA

Integrity & Authentication

- **Suppose p = 3, q = 11,**
- **Alice's private key (d,n) = (7,33)**
- **Alice's public key (e,n) = (3,33)**

- **Q:**

1) What is Alice's signature s on message m = 2?

2) If Cathy receives message signature pair (m = 4, s = 31) from Alice, does Cathy conclude s is a valid signature by Alice on m?

Activity (5 mins)

1) Click the latest link in the Zoom chat
2) Add your question response to the Ed forum


Digital Signatures & Hash Functions

Integrity & Authentication

- Digital Signature is **slow**, similar to PKE
  - docs (m) to sign are usually **long (MBs-GBs)**
  - so sign a **short (~hundreds of bits, e.g. 256 bit) representation** (**digest/hash value**) of m, not directly on m

- Hash function:
  - convert m to digest d, |m| >> |d|
  - Side benefit: improves UNForgeability security of signatures


Hash Functions

Integrity & Authentication

- **input:** m of **any** length
- **output**: h of **fixed** length L
- |m| >> |h|

- e.g. "**MD5**" (obsolete) has L = 128 bits
  "**SHA-1**" (obsolete) has L = 160 bits

- Use **SHA-2** (L=256 to 512) or **SHA-3** (L=256 to 512) hash functions

Hash Function: One-Way Security


Integrity & Authentication

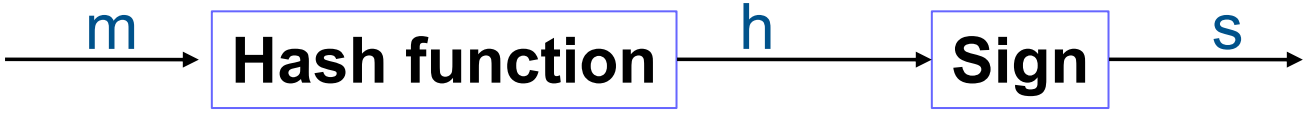
- **(I) One-way function security: (aka preimage-resistance):**

given h = H(m), computationally infeasible to find m such that h = H(m)

  - though easy to compute output h = H(m) from input m

    - i.e. hard to find input for an output

    - needed for password storage security (later), RSA signature security

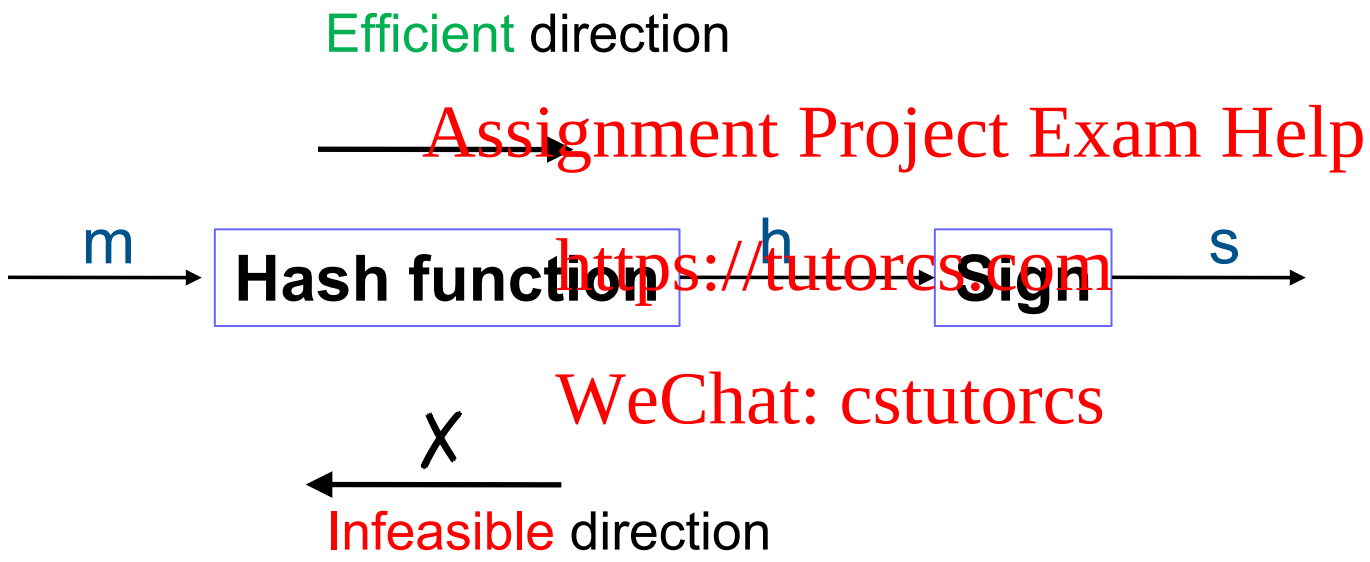


Q: How could an attacker efficiently forge some message m with a valid RSA signature if Hash Function is **not** a one-way function?
Hint: What does RSA signature verification of s compute?

Activity (2 mins)
1) Click the latest link in the Zoom chat
2) Add your question response to the Ed forum

Hash Function: One-Way Security

Integrity & Authentication



Hash Function: COL Security

Integrity & Authentication


- **(II) COL: collision-resistance security**
  - computationally **infeasible** to find **any** pair of **distinct** messages m_1 , m_2 such that $h_1 = H(m_1)$ is **equal** to $h_2 = H(m_2)$

Q: Suppose RSA signature uses a Hash function that is **not collision-resistant.** How could an attacker exploit this to efficiently forge some document and its valid RSA signature?

Activity (2 mins)

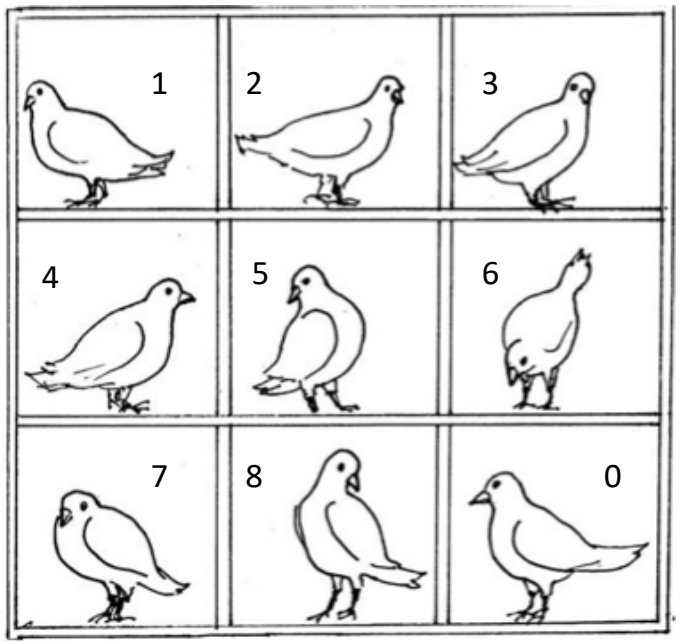
1) Click the latest link in the Zoom chat
2) Add your question response to the Ed forum


Hash Function: COL Security

Integrity & Authentication

- Hash functions: long input to short output , |m| >> |h|
  - more possible input values than possible output values
  - collisions must **exist**
    e.g.

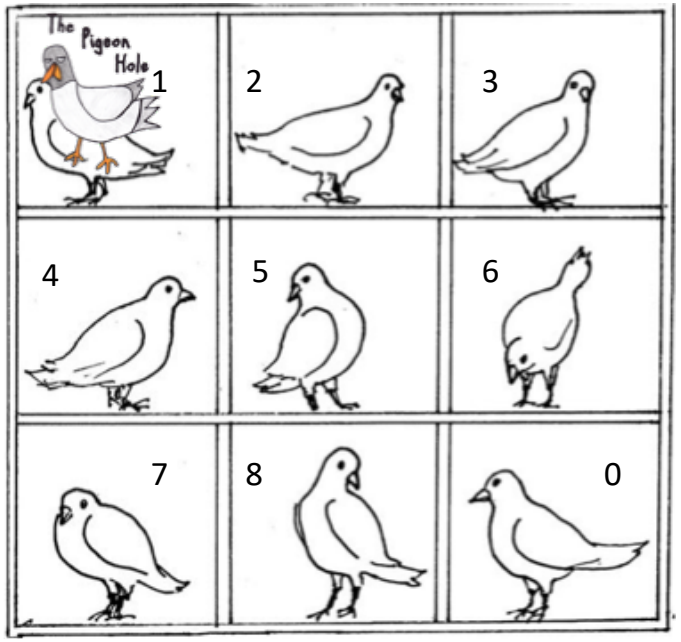
  - the hash table is being filled up by:
    - 1 mod 9 = 1
    - 2 mod 9 = 2
      .
      .
      .
    - 9 mod 9 = 0



Hash Function: COL Security

Integrity & Authentication

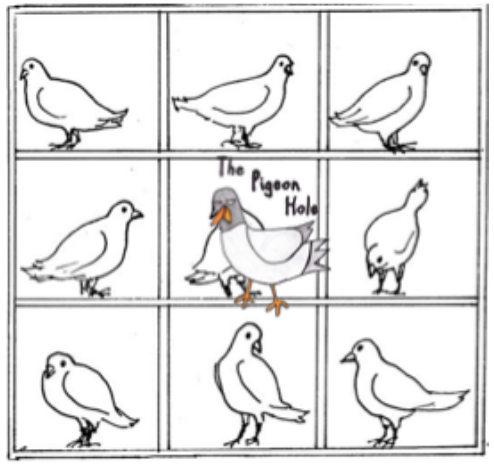

- hash table already full after computing 9 values …
  - further computing e.g.
    10 mod 9 = 1
    gives collision as shown
- **Pigeon hole principle**



Hash Function: COL Security

Integrity & Authentication

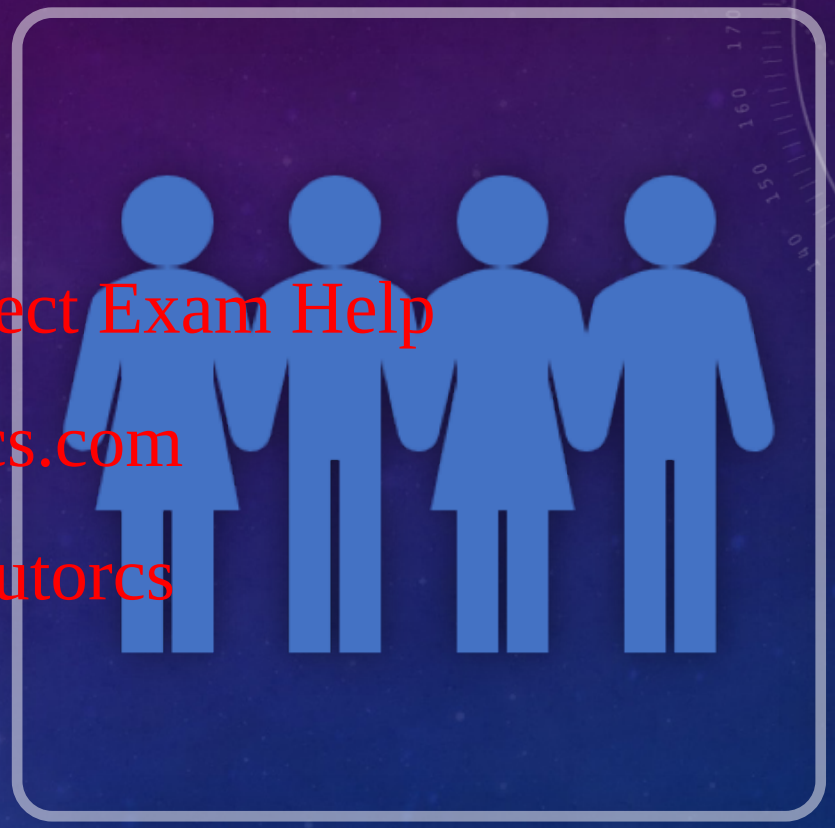
- **Surprising Fact (Birthday "Paradox")**
  - only need $23 \approx \sqrt{365}$ random people in a room to have good chance of finding a pair with **same** birth day & month
  - → **In general**: for any hash function H with n-bit output, can find collision using $\approx \sqrt{2^n} = 2^{n/2}$ random inputs to H
  - e.g. for n = 256, can find collisions with $\approx 2^{128}$ input, so only have security against attacks with run-time < 2^{128} ops



- → expected collision-resistant (COL) security for hash function:
  - should take about $2^{n/2}$ evaluations of H to find COL

*BIRTHDAY PARADOX: EXAMPLE

How many people required in the same room for two people in the room to have the same birthday with about 50% chance?



Assumptions made:

1. 29th February doesn't exist
2. Each birthday is equally likely to come up.
3. Each people in the room are independent from each other. E.g. none of them are twins etc.

The importance of independence is that we can multiply the probability of occurrences of the events together

*RELATING TO COLLISION RESISTANCE

You might think that we will be needing a lot of people to eliminate the chances away, however it is much less than a lot of us would tend to think:
The answer is only 23 people!

We got the answer from hash function concept, which stated that:
Given H(x), it should take about $2^{n/2}$ evaluations of H(x) to find a collision for H

Following the idea, attacker just needs to choose ~ $\sqrt{2^n}$ (which equals $2^{n/2}$) to look for collision in the hashes which will succeeds in high probability.

Particularly for birthday problem, $\sqrt{365}$ = 19 which is considered an approximate number of people needed, 23 people.

We will prove the answer on the next slide by using probability.

*Birthday Paradox Calculation

Instead of proving “How many people for collision (two people having same birthday)”, we’ll look at complement: what’s the probability that no one in the room shares a birthday?

When 2nd person enters the room, his/her birthday must not be the same with the 1st person. To avoid the first birthday the probability of getting different birthday at this point:

$$\frac{364}{365} = 0.9973$$

When 3rd person enters the room, same concept applies, he/she has to avoid first two birthdays, thus 363 possibilities left.

$$0.9973 \times \frac{363}{365} = 0.9918$$

By the time we have 15 people in the room:

$$0.7769 \times \frac{351}{365} = 0.7471$$

That means probability that everyone avoids each other’s birthday has dropped to less than 75%,which also means there is greater than 25% that two people in the room might share the same birthday

When we have 23 people in the room,

$$0.5243 \times \frac{343}{365} = 0.4927$$

That means chance of everyone having unique birthday = 49.27% and the chance of at least two sharing same birthday = 50.73%
Which means there is a high possibility (more than 50%) of getting ‘collision’ in the room.

Digital Signatures: Properties

Integrity & Authentication

- **analogous to handwritten signature**
  - can only be done by the owner

- **Properties:**

  - verifies **author**
    - **UNForgeable**

    - **Undeniable – non-repudiatable – Q: why?**
  - authenticates the **contents** (m) at the time of signature
  - **universally verifiable**:
    - verifiable by third parties, to resolve disputes


Digital Signatures: Security

Integrity & Authentication

- Attack model for Signature Schemes: **EUF-CMA**
- Attack **capability: attacker knows signer's public key pk**
  - **Chosen-Message Attacks** (CMA):
    - Should be unforgeable even if attacker can see many valid signatures on many messages, even messages chosen by attacker.
- Attack **goal:**
  - **Existential UnForgeability (EUF):**
    - Should be infeasible for attacker to produce **any** valid (msg,sig) pair where msg has not been signed by the honest signer (even for randomly looking msg)


Digital Signatures: Security

Integrity & Authentication

- Warning: “Textbook RSA signature” without one-way hash function is NOT EUF-CMA , shouldn’t be used
  - e.g.. easy to create valid forgery signature for random message m
  - (and also some not so random msgs…)
- → Use a **standardised** digital **signature** scheme using a **secure (one-way and collision-resistant)** one-way **hash** function
  - e.g. “RSASSA-PSS” digital signature scheme in PKCS #1 v2.2 standard
  - designed to achieve EUF-CMA security using a secure cryptographic one-way hash function

MONASH University

Symmetric Cryptography for Integrity


Symmetric Crypto for INTegrity

Integrity & Authentication

- Problems with Digital Signature (PKC)
  - PKC is slow
  - **Universal** verifiability may not be needed
    - May be sufficient to allow **receiver** to verify msg came from other side
  - Sender may already share a secret key with other side
    - → can use a MAC instead of signature

- **Message Authentication Code (MAC): symmetric key** analogue of digital signature
  - **fast**
  - similar building blocks/operations as **symmetric** key encryption


MACs vs others

Integrity & Authentication

	Public-Key Crypto (*slow*)	**Symmetric-Key Crypto** (private *key distribution problem*)
CONFidentiality	Public Key Encryption / Key Exchange Protocol	Symmetric Key Encryption
INTegrity	Digital Signature	Message Authentication Code (MAC)


MAC: What?

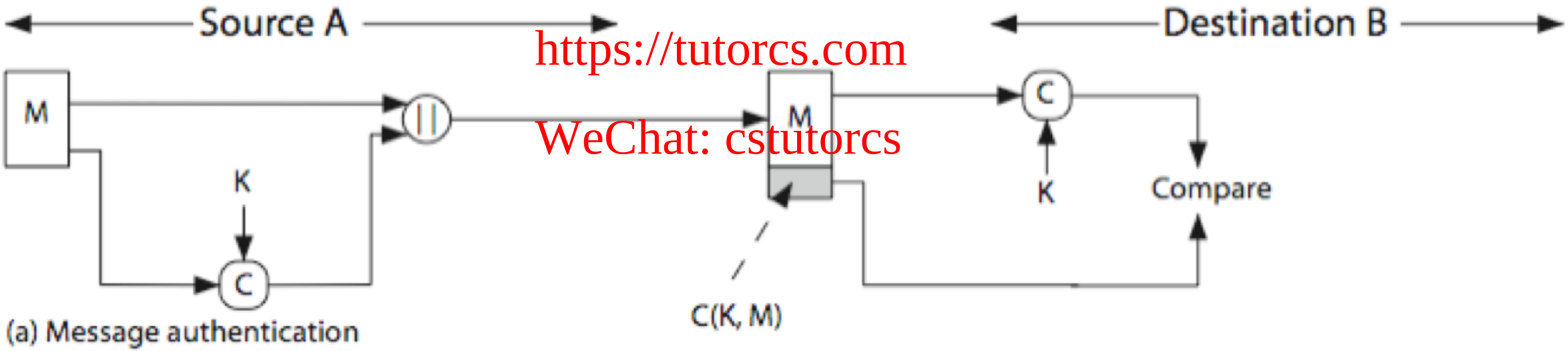
Integrity & Authentication

- MAC can be viewed as a cryptographic checksum
  - $\mathrm{MAC} = C_K(M)$
  - condenses a variable-length message M
  - **using a secret key K**
  - to a fixed-sized authenticator / check code / INT code
- is a many-to-one function
  - potentially many messages can generate the same MAC
  - (***compressing feature similar to hash functions***)
  - but finding those messages should be difficult

Q: What is a main difference between a MAC and a Hash Function?

MAC: Diagram

Integrity & Authentication




MACs: Security

Integrity & Authentication

- Attack model for MACs: **EUF-CMA – similar to signatures (except no pk)**
- Adversarial **capability:**
  - **Chosen-message attacks** (CMA):
    - Should be unforgeable even if attacker can see many valid MAC authenticators on many msgs, even msgs chosen by attacker
- Adversarial **goal:**
  - **Existential unforgeability (EUF):**
    - Should be infeasible for attacker to produce **any** valid (msg,auth) pair where msg has not been MAC'd by the honest sender

      (even for randomly looking msg)


MACs: Security Properties

Integrity & Authentication

- knowing a message m and MAC output s,
  - infeasible to find another message m' with **same** MAC
  - deals with message replacement (MAC copy/replay) attacks

- MACs should be uniformly distributed across the messages
  - deals with need to thwart a brute-force attack based on chosen plaintext

- MAC should depend equally on all bits of the message
  - dictates that the authentication algorithm should not be weaker with respect to certain parts or bits of the message than others

MACs: How to Construct


Integrity & Authentication

- **Two common ways:**
  - using **block ciphers**
    - **special block cipher authentication modes of operation,** e.g.
      - CMAC (CBC mode for authentication)
  - using **cryptographic hash functions**
    - **special hash function authentication modes of operation, e.g.**
      - HMAC


MACs: from Block Ciphers

Integrity & Authentication

- can use any block cipher chaining mode & use final block as a MAC
  - need to do carefully to deal securely with arbitrary message lengths

- Old method (obsolete): **Data Authentication Algorithm (DAA)** was a widely used MAC based on DES-CBC

  - No longer recommended for use as it is too small for sufficient security, and has other problems

- New method: **CMAC (NIST, 2005)**
  - uses two keys K1, K2 derived from a single key K
  - works with any secure block cipher (e.g. AES-128)

MACs: from Block Ciphers

Integrity & Authentication

Q: Why need two cases/keys K1 and K2? What would go wrong if we just had one case (K1=K2)?

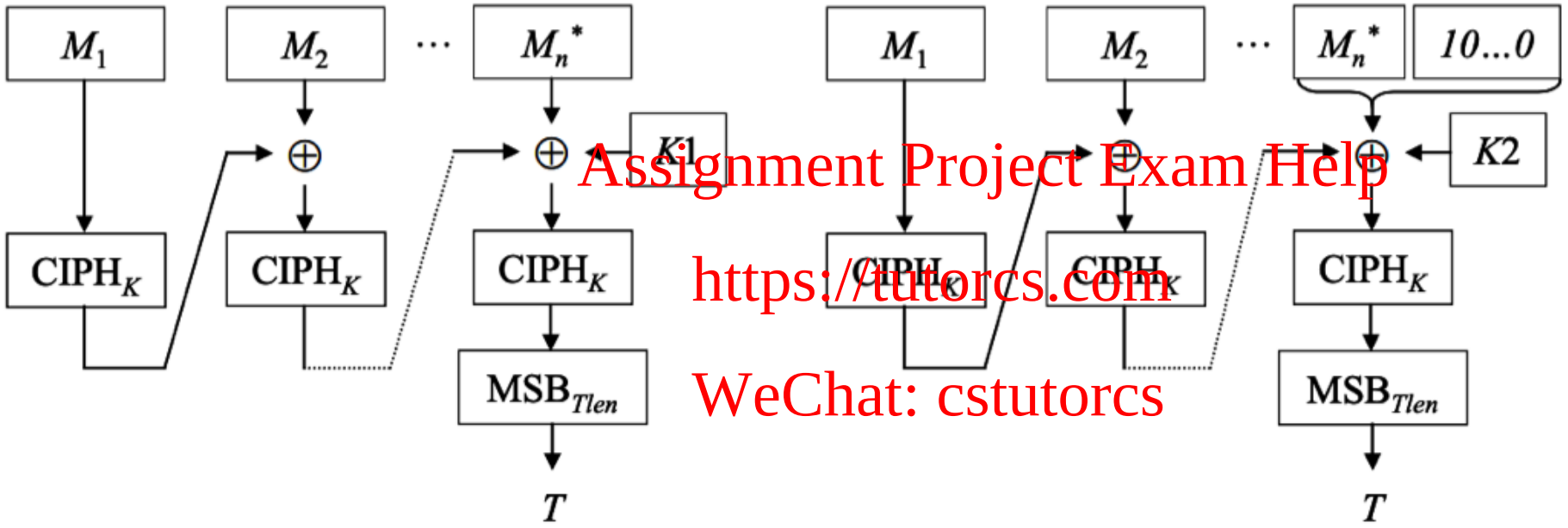


Figure 1: Illustration of the two cases of MAC Generation.

Fig Ref: NIST SP800-38b

The two cases of MAC Generation are illustrated in Figure 1 above. On the left is the case where the message length is a positive multiple of the block size; on the right is the case where the message length is not a positive multiple of the block length.


MACs: from Hash Functions

Integrity & Authentication

- Hash function is **similar** to MAC:
  - one-way
  - for any input length, fixed output length
  - but **no** key input

- Can be used with **both symmetric & public key cryptography**
  - Hashing the message in digital signatures
    - can also be used to design a MAC

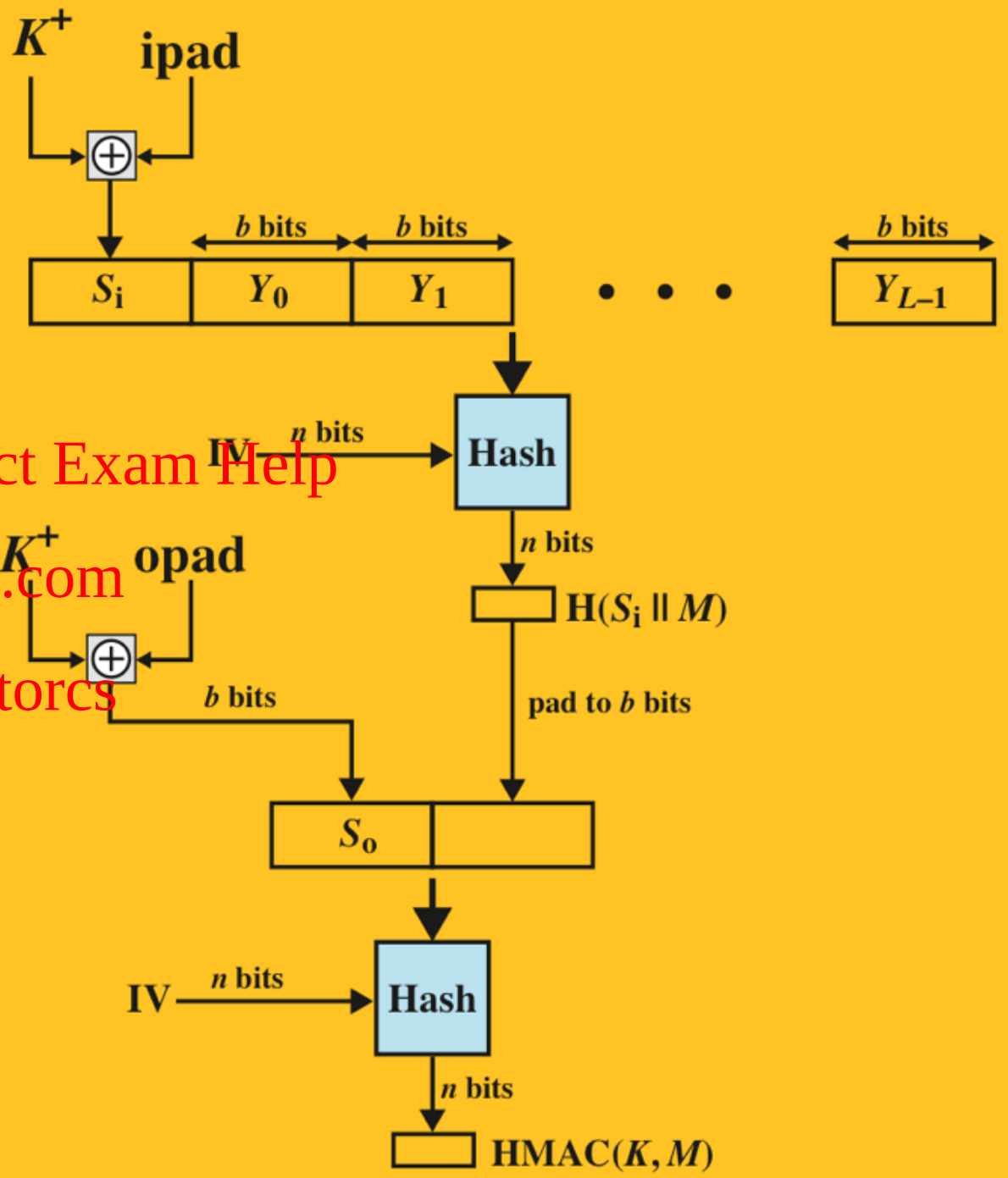

MACs: from Keyed Hash Functions

Integrity & Authentication

- **want a MAC based on a hash function**
  - because hash functions are generally faster than block cipher
  - crypto hash function code is widely available
- hash includes a key along with message
- original proposal:
  - KeyedHash = Hash(Key||m)
  - some security weaknesses were found with this
- eventually led to development of the MAC standard HMAC

MAC Example: HMAC

Integrity & Authentication




MAC Example: HMAC

Integrity & Authentication

- specified as **Internet standard RFC2104**
- **uses hash function on the message:**
  - HMAC(K,M)= Hash[(K+ XOR opad) || Hash[(K+ XOR ipad) || M)]]
    where K+ is the key padded out to size
  - opad, ipad are specified padding constants
- overhead is just hash calculations on 3 more blocks than hashing the message alone
- **any hash function can be used**
  - eg. MD5, SHA-1, RIPEMD-160 (obsolete, not recommended)
  - SHA-2, SHA-3 (recommended)


HMAC: Security

Integrity & Authentication

- proved: security of HMAC relates to that of the underlying hash algorithm
- **attacking HMAC requires either:**
  - brute force attack on key used (2^n)
  - birthday attack (but since keyed would need to observe a very large number of MAC'd messages)
    - Collision resistant attacks, an adversary wishes to find 2 messages that yield the same hash (attack needs ~$2^{n/2}$ MAC'd messages)
- choose hash function used based on speed vs security constraints


MACs: Combining with Encryption

Integrity & Authentication

- for **secure communication,** need **both:**
  - CONFidentiality
  - INTegrity

- can combine: **authentication + encryption**
  - e.g. encrypt-then-MAC
  - use **separate keys** for encrypt & MAC to avoid security vulnerabilities
  - do carefully to **preserve security** of both mechanisms

- In practice: use efficient & secure **authenticated encryption** modes of block ciphers:
- e.g. **GCM authenticated encryption mode [NIST 2007],** *will not cover in detail*)

PKC for INTegrity & AUTH

Integrity & Authentication

- **if public-key encryption / signature is used:**
  - Pub key encryption provides no authentication of sender
    - since anyone potentially knows public-key
  - however if
    - senders **signs** the message using their private-key
    - then **encrypts** with recipient's public key
    - Then we have both CONFidentiality and AUTHentication
  - Cost: two public-key mechanisms

Further Reading

Integrity & Authentication

- Chapter 2 (Section 2.2) and Chapter 21 (Sections 21.1-21.2) of the textbook: *Computer Security: Principles and Practice" by William Stallings & Lawrie Brown,* Prentice Hall, 2015