

Cryptography Algorithms

Lecture 10: Message Authentication Code (MAC) and Cryptographic Hash

Second property - Integrity

ดร. รุ่งรัตน์ เวียงศรีพนาวัลย์

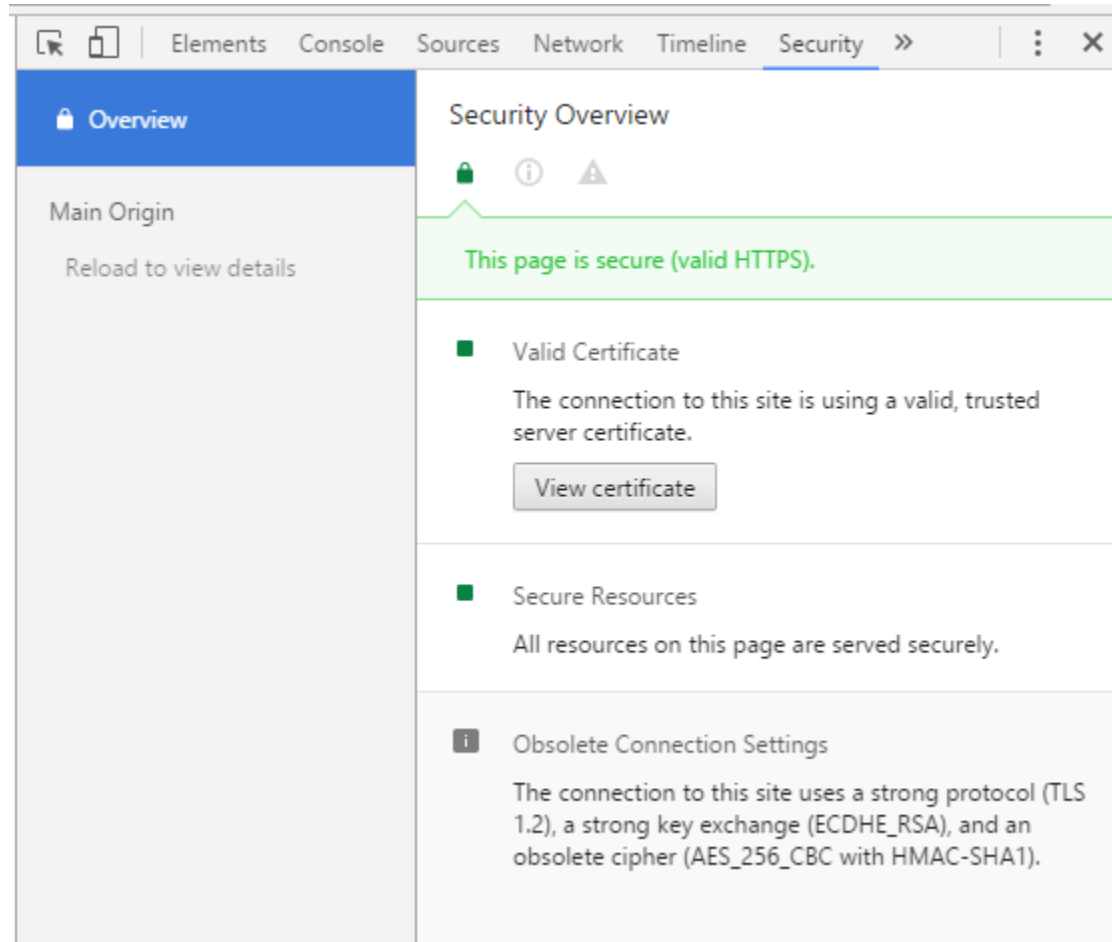
10.08.21

เค้าโครงการบรรยาย

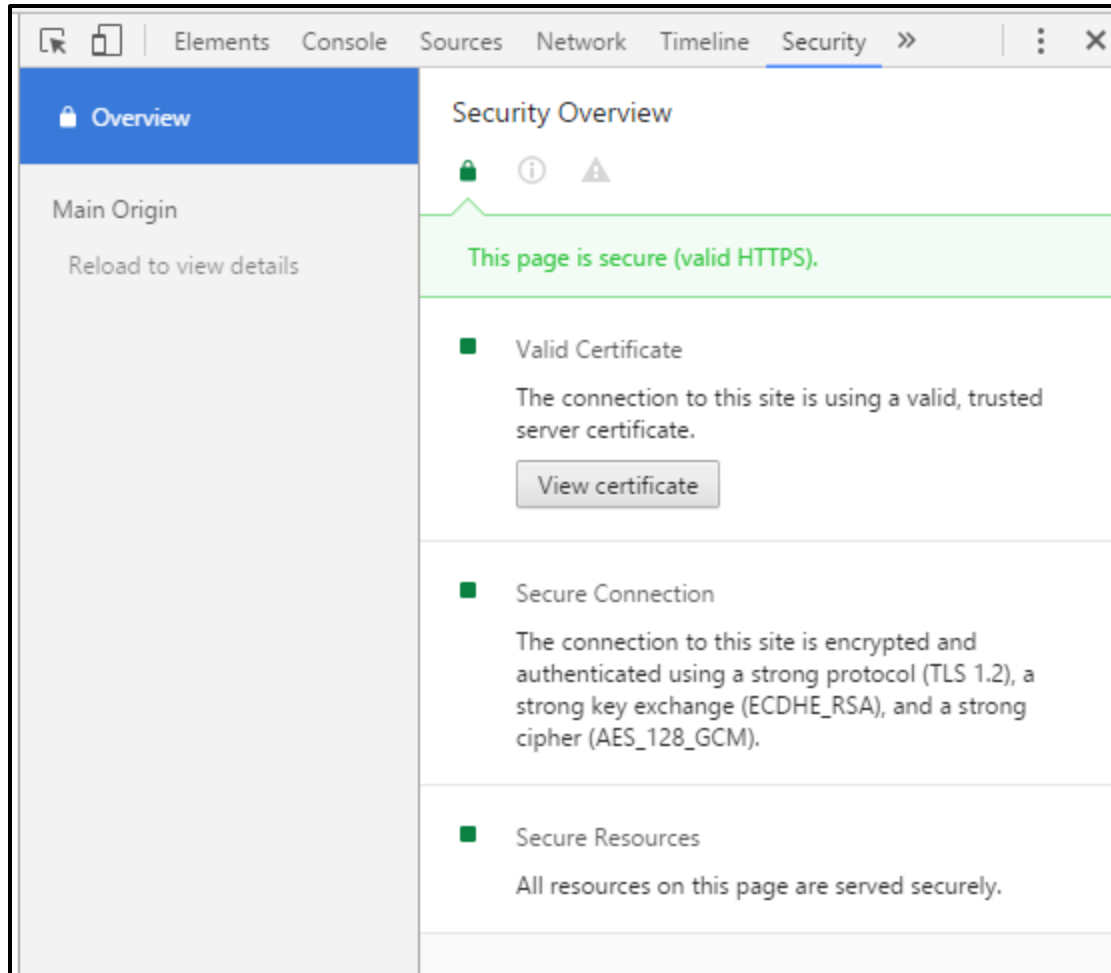
- ▶ What is Message Integrity?
- ▶ How it works?
- ▶ What is Message Authentication Code (MAC)?
- ▶ Unconditional MAC
- ▶ Ways to Generate MAC
 - ▶ Block Cipher Based MAC
 - ▶ Cryptographic Hash Based MAC
 - ▶ HMAC
- ▶ Authenticated Encryption



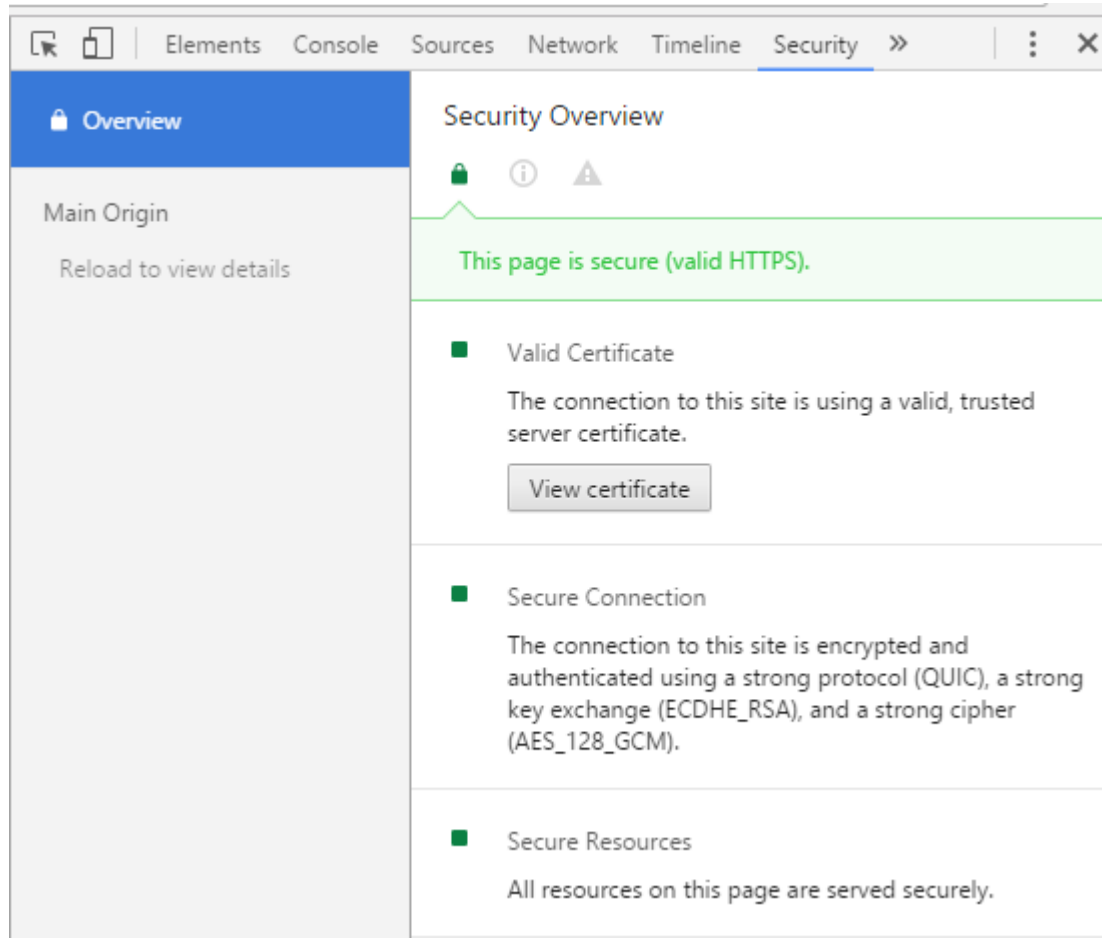
Try get to www.hotmail.com



Try www.paypal.com



Try www.google.com



Message Integrity

- ▶ Threat ของคุณสมบัติ Integrity ของแมสเสจ (message integrity) แบ่งได้เป็นสองประเภทคือ
- ▶ 1. **Un-intentional** ไม่ได้ตั้งใจให้เกิด เช่น เกิดจากสัญญาณรบกวน (noise):
 - ▶ สามารถป้องกันได้โดยการทำ
 - Error detecting codes หรือ Error correcting codes.
 - For example, the parity bit in the ASCII code is for error detection. (8 bits to represent 7-bits of information).
 - เช่น การใช้ parity bit ในการส่งข้อมูลของรหัสแอสกี (ปกติแอสกี เจ็ดบิต แต่ ส่ง 8 บิต โดยบิตที่แปดเป็น บิตที่ใช้เช็คว่าคุณสมบัติข้อมูลมีการเปลี่ยนแปลงหรือไม่)
 - หรือ การทำ CRC เช่น CRC32
- ▶ **Intentional (ตั้งใจ)** มีคนตั้งใจในการเปลี่ยนแปลงข้อความหรือ สร้าง ข้อความที่ไม่ถูกต้อง
 - ▶ Protection in this case is provided by (ในกรณีนี้ป้องกันได้โดยใช้)
 - Message Authentication Codes (MAC).



Tag

- ▶ ปกติวิธีการที่ใช้ในการป้องกัน threat ทั้งสองประเภททำโดยการเติม (append) tag ต่อท้ายไปที่เมสเสจที่ต้องการส่ง
- ▶ บางทีเรียกเรียก tag นี้ว่า checksum
 - ▶ Such a tag is a string of bits, in a binary representation.
 - ▶ แทคนี้จริงๆ แล้ว string ของ บิต (0 หรือ 1)



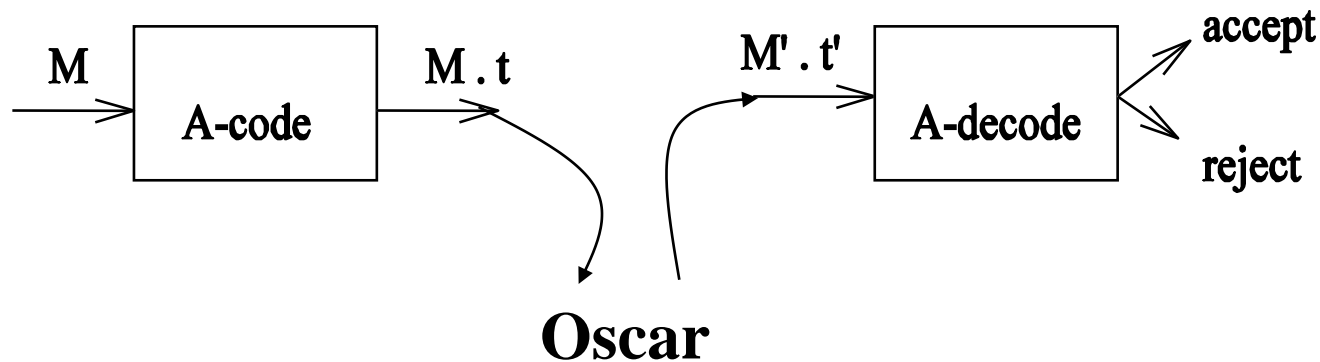
ตัวอย่าง

- ▶ Algorithms for **error detection** do not need a **secret key**.
- ▶ ตัวอย่างต่อไปนี้ คือ ตัวอย่างของ อัลกอริธึมที่ใช้ในการสร้าง parity bit:
 - ▶ ให้แมสเสจแต่ละ block มีขนาด 8 บิต (เทียบได้กับจำนวนเต็ม ในช่วง 0-255)
 - ▶ ให้ parity bits มีขนาด 3 บิต : โดยคือค่าเศษที่นำจำนวนเต็มในแต่ละบล็อกหารด้วยเจ็ด เช่น message 01111101=125.
 - ▶ The parity bits หาจาก $125 \bmod 7 = 6 = 110$.
 - ▶ ดังนั้นข้อมูลที่ส่งออกไปคือ 01111101 110.
- ▶ ถ้ามีความผิดพลาด (error) ที่บิตใดบิตหนึ่งเช่น ผู้รับได้รับ 01011101 110, ซึ่งสามารถตรวจจับความผิดพลาดได้ จากการนำ : $01011101 = 93$ ไป mod ด้วย 7 ซึ่งจะได้
 - ▶ $93 \bmod 7 = 010 \neq 110$



Error correcting/detecting

- ▶ Error-correcting/detecting codes ไม่สามารถป้องกัน
การโจมตีแบบตั้งใจ เช่น การ **spoofing** หรือการแก้ไข
(**modification** หรือ **impersonation**) ข้อมูลที่ส่ง



เนื่องจากอัลกอริทึมที่ใช้ในการคำนวณ ทุกคนทราบ ดังนั้น Oscar สามารถเปลี่ยนจาก
แมสเสจ M เป็น M' และ คำนวณหา tag t' ได้
ซึ่งเมื่อผู้รับได้รับและทำการตรวจสอบจะพบว่า tag t' เป็น tag ที่ถูกต้อง เพราะ
คำนวณมาจากแมสเสจ M' !!!

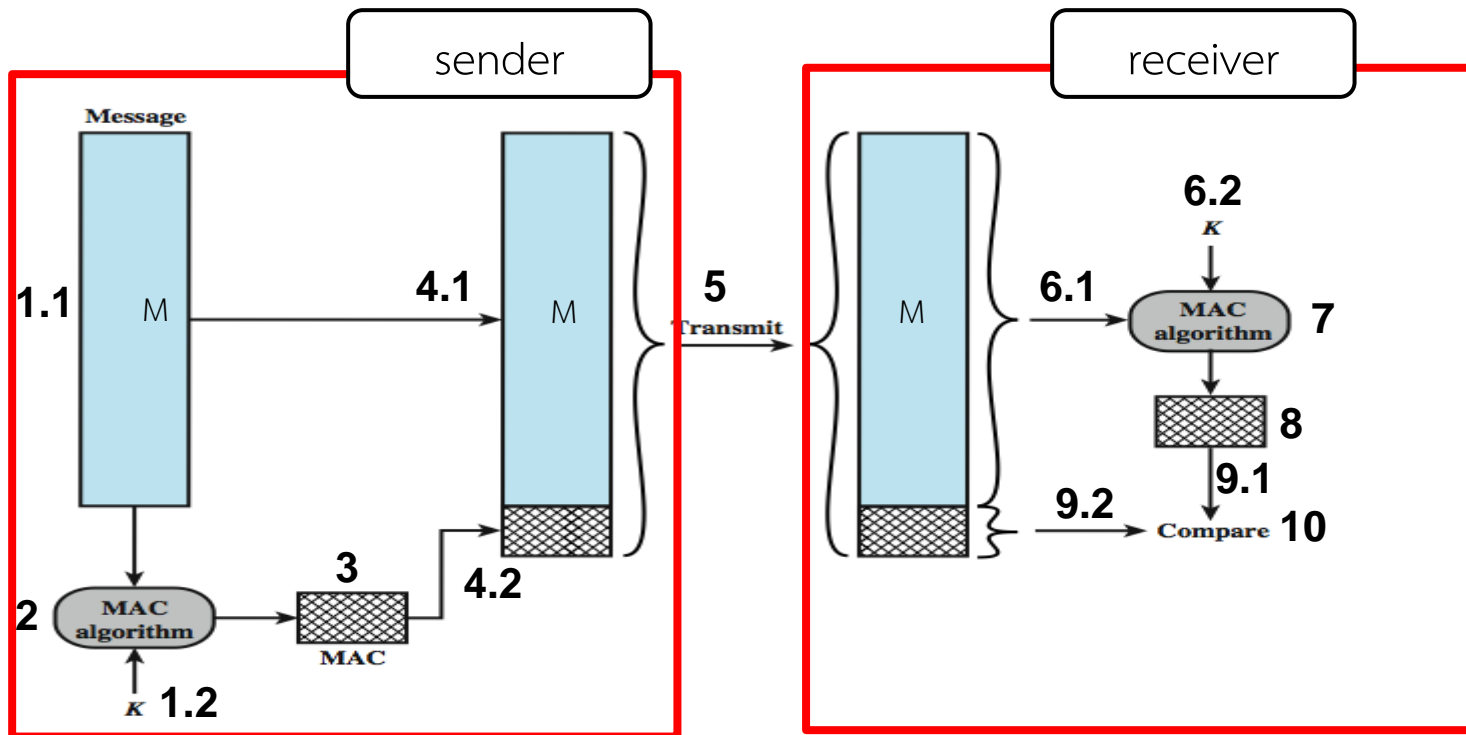
Message Authentication Code (MAC)

- ▶ การใช้ MAC จะมีการตรวจสอบ (verify) ว่า message ที่ได้รับ authentic
 - ▶ เนื้อหาไม่ถูกเปลี่ยนไป **contents unaltered**
 - ▶ มาจาก **authentic source (Sender Authentication)**
 - ▶ timely and in correct sequence
 - ▶ เราสามารถเพิ่ม เวลา และ sequence no. ในการใช้ตรวจสอบ เวลาหรือ ลำดับของแมสเสจที่ส่งระหว่างผู้ส่งและผู้รับได้ (ช่วยกัน replay attack) (You can also add time and sequence number to verify a message's timeliness and sequence relative to other messages flowing between two parties.)
 - ▶ วิธีการหนึ่งที่ใช้กันมากในการ implement message authentication คือ การเติม tag หรือ checksum ลงไป ซึ่ง tag หรือ checksum ที่ใช้ใน message authentication จะถูกเรียกว่า **Message Authentication Code** หรือ เรียกย่อๆ ว่า MAC
 - ▶ MAC ป้องกัน active attacks ในขณะที่ Encryption ป้องกัน passive attacks.
-



Message Authentication Codes

- Sender and receiver share a secret key K .



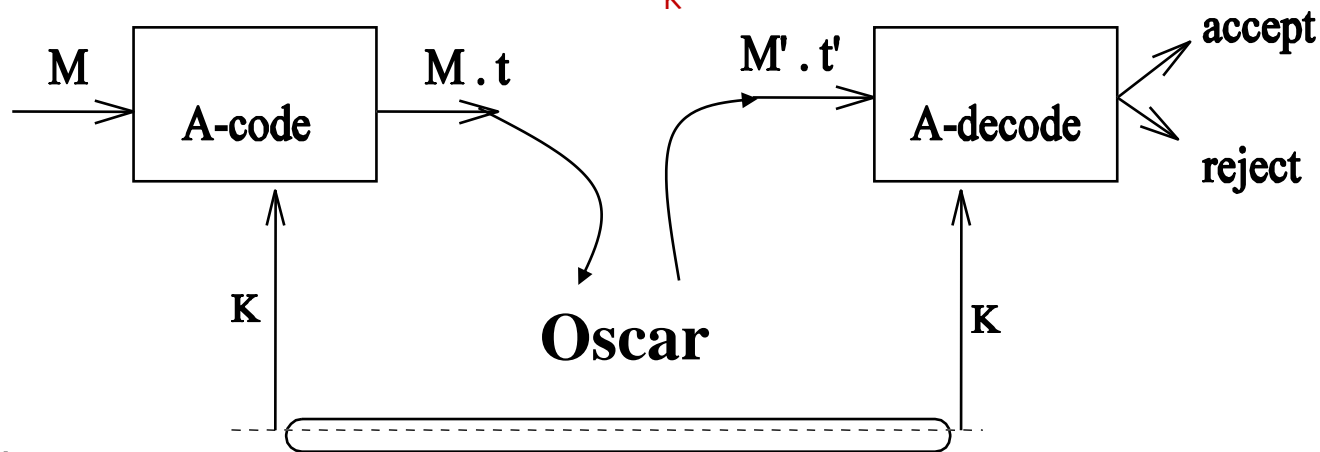
Message Authentication Code Cont.

- ▶ *MAC* ปลอดภัย (secure) ถ้าการปลอม (forging) $(M, \text{MAC}_K(M))$, ยาก (hard) (แมสเสจถูกแก้ไขแต่ตรวจจับไม่ได้)
- ▶ การสร้าง MAC สามารถทำได้ทั้งในกรณี unconditional security หรือ practical security.
- ▶ แต่ MAC สำหรับ unconditional security ใช้จำนวนบิตมาก (many key bits)
 - ▶ Unconditional security ในกรณีนี้หมายถึงว่า โอกาสที่ผู้โจมตี จะประสบความสำเร็จ (ปลอมแปลงแมสเสจได้) จะน้อยกว่า หนึ่งในสอง



Message Authentication Code

- ▶ ผู้ส่ง (transmitter) และผู้รับ (receiver) จะมีการ แชร์ secret key (K) ก่อน
- ▶ ในการส่งแมสเสจ M , ผู้ส่งจะคำนวณค่า MAC (t) และทำการต่อมันไปที่ message (M) ก่อนส่งไป ดังนั้นสิ่งที่ส่งคือ $(M.t)$ ($t = \text{MAC}_K(M)$).



- ▶ ผู้รับได้รับ แมสเสจ $(M.X)$.
 - ▶ นำ คีย์ K และ แมสเสจ M เพื่อคำนวณค่า $\text{MAC}_K(M)$
 - ▶ ทำการเปรียบเทียบ ค่าที่ได้กับ X .
 - ▶ ถ้าค่าทั้งสองเหมือนกัน แสดงว่า แมสเสจที่ได้รับนั้นเป็นแมสเสจที่ผู้ส่งส่งมาจริง (authentic)
- ▶ MAC ถูกเรียกอีกชื่อหนึ่งว่า cryptographic checksum.

An unconditionally secure MAC

- ▶ ให้ แมสเสจ (M) เป็นเซตของข้อมูล
 - ▶ ตัวเลขระหว่าง 0 and $p-1$, เมื่อ p เป็นจำนวนเฉพาะ (prime)
- ▶ แมสเสจแต่ละแมสเสจจะถูกแบ่งเป็น บล็อกขนาด $\log_2 p$
- ▶ ผู้ส่ง (transmitter) และ ผู้รับ (receiver) เลือก/สร้าง/แลกเปลี่ยน (choose/establish/exchange) คีย์ K
 - ▶ โดยที่ K เป็นคู่ของตัวเลข $K=(a,b)$ เมื่อ $0 \leq a, b \leq p-1$. (a และ b มีค่าอยู่ระหว่าง 0 ถึง $p-1$)
- ▶ คำนวณค่า MAC จาก
 - ▶ $MAC_K(M) = aM + b \bmod p$
- ▶ แม้ enemy เห็น $(M, MAC_K(M))$ แต่การที่ไม่ทราบ K ทำให้ไม่สามารถเปลี่ยน M และ คำนวณ recalculate cryptographic checksum ได้
- ▶ วิธีการนี้เป็นตัวอย่างหนึ่งของ Authentication-code หรือที่เรียกว่า (A-code).



ตัวอย่าง: ให้ $p=11$, $K=(a,b)=(2,3)$.

- ▶ หา MAC ของ message 5, ดังนั้น

$$MAC_K(5) = 2*5 + 3 \bmod 11 = 2$$

- ▶ Attacker ไม่ทราบค่า K , แต่จะทราบค่า M กับ MAC (2).
- ▶ ดังนั้นสามารถลองค่า ' a ' ได้ 11 ค่า ซึ่งจะสัมพันธ์กับค่า ' b ' ซึ่งจะได้ค่า key ทั้งหมด 11 คู่ $(0,2), (1,8), (2,3), \dots$
- ▶ ซึ่งถ้า attacker ต้องการเดา โอกาสที่จะเดาถูกคือ $1/11$.
- ▶ และค่า tag จะเป็นค่า 1 ใน 11 จำนวนเสมอ
- ▶ ในกรณีที่ attacker สามารถดักจับ message M' ซึ่งค่า MAC คำนวณมาจากคีย์เดียวกัน เช่นให้ $M'=7$ ดังนั้น $MAC_K(7)=5$,
 - ▶ ดังนั้น attacker สามารถหาค่าคีย์ได้จาก สมการ 2 สมการดังนี้

$$a*5 + b \bmod 11 = 2, \quad a*7 + b \bmod 11 = 5$$

ซึ่งจะหาค่าคีย์ได้คือ $K=(2,3)$.

ดังนั้นต้องเปลี่ยนคีย์ในทุก message

Unconditional MAC

► In general it can be proven that:

**Unconditionally secure authentication systems
require**

a large amount of key amount*

and so are

I M P R A C T I C A L .

(unconditional secure authentication ต้องใช้ คีย์เป็นจำนวนมากจึงไม่ practical)

***คีย์ต้องเปลี่ยน ทุก เมสเสจ เลยต้องใช้มาก

Unconditional MAC Cont.

To illustrate,

- ▶ จงออกแบบ ระบบ Message Authentication System ที่มีคุณสมบัติ unconditional security ซึ่งโอกาสที่ intruder จะประสบความสำเร็จที่ความน่าจะเป็น ($P \leq 0.01$)
- ▶ ให้ $p=101$, and $K=(a,b) = (5,11)$.
 - ▶ ซึ่ง M จะเป็น message ที่อยู่ใน range 0-100 ซึ่งจะใช้ 7 บิต
 - ▶ และ Message ที่ส่ง $M||MAC$ จะมีขนาด 14 บิต
 - ▶ คีย์ K จะมีขนาด 14 บิต (a,b) อย่างละ 7 บิต
- ▶ จากการที่ $MAC_K(M) = aM + b \bmod p$
ถ้า $M = 90$, $MAC_K(90) = 5 * 90 + 11 \bmod 101 = 0111001$.
ดังนั้น message ที่ส่งคือ 1011010 0111001.

The chance of the enemy's success is $P=1/101 < 1/100$.

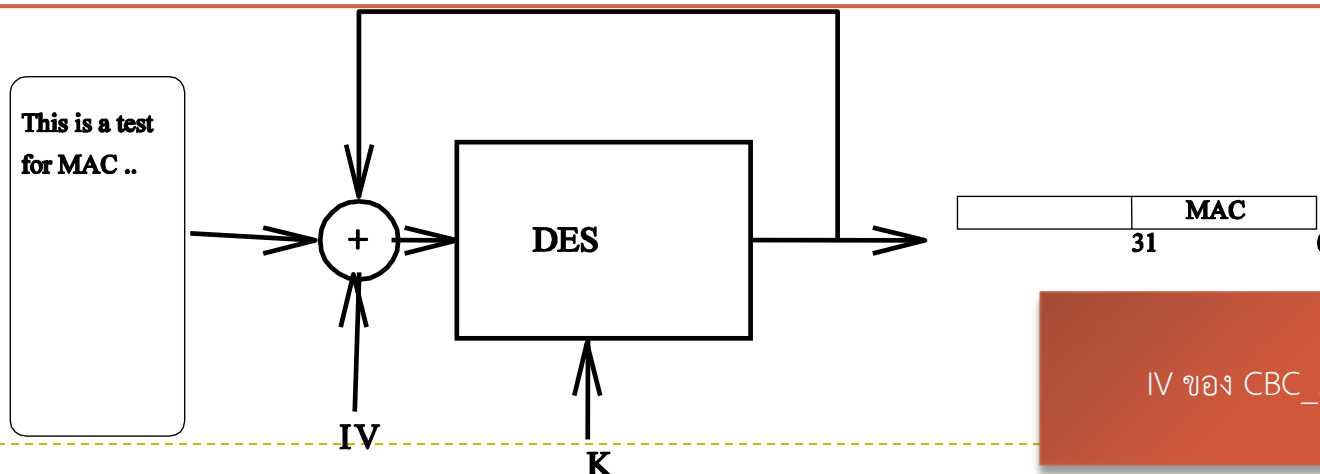


Unconditional MAC Cont.(2)

- ▶ Unconditional MAC ไม่ **efficient** มากๆ ในเรื่องของ
 - ▶ จำนวนบิต (number of bits) ที่ต่อท้ายแมสเสจ
 - ▶ จำนวนคีย์ (amount of key information) โดยเฉพาะในกรณีที่แต่ละแมสเสจต้องใช้คีย์ไม่ซ้ำกันเลย
 - ▶ คล้ายๆ กับ one-time pad ใน encryption ซึ่งเป็น unconditionally secure encryption. ที่คีย์ ความยาวของคีย์ต้องเท่ากับความยาวของแมสเสจที่ส่ง
 - ▶ ใน MAC แบบ **practical security** มันเป็นไปได้เสมอที่จะปลอม (forge) message, แต่ต้องใช้การคำนวณเป็นจำนวนมาก (amount of computation required to do so is **large**) ในแบบที่ attacker ไม่มี resource พอที่จะทำได้ (to the extent it is impractical for the attacker to do so.)

2. ชนิดของ MAC: Block Cipher based MAC

- ▶ เป็น MAC ชนิดหนึ่งที่ใช้มากที่สุด
 - ▶ Data Authentication Algorithm
- ▶ อัลกอริทึมบล็อกไซเฟอร์ในโหมด CBC สามารถนำมาสร้าง checksum ได้เพราะ ciphertext ใน CBC โหมด จะถูก feed กลับไปที่ encryption algorithm
- ▶ เช่น การใช้ 32-bit MAC ที่สร้างมาจาก CBC mode ของ DES. ไม่ว่า แอสเสจ M จะมีขนาดเท่าไร (arbitrary length) $MAC_K(M)$ จะเป็น 32 บิตท้ายของบล็อกสุดท้าย



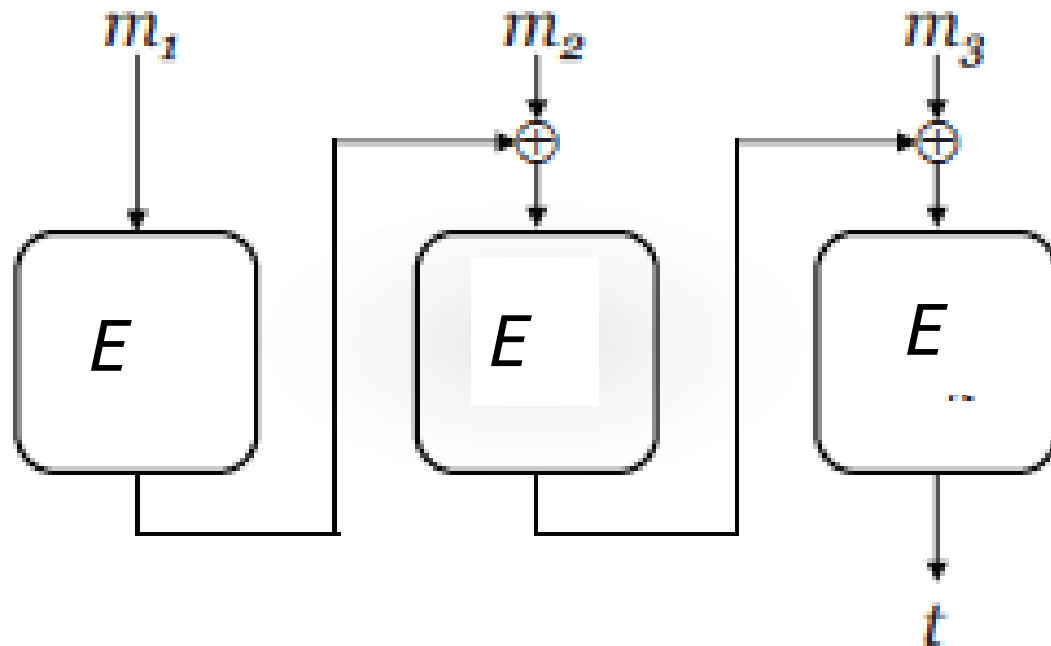
IV ของ CBC_MAC จะมีค่าเป็น 0

คำถาม: attack ต่อ MAC แบบนี้เกิดขึ้นได้อย่างไรบ้าง

- ▶ Question One: cost หรือ โอกาสของ enemy ในการปลอม (forge) message เป็นเท่าไร?
- ▶ Attack 1: โจมตีไปที่ key space
 - ▶ Enemy ใช้ exhaustive key search ในการหาคีย์.
 - ▶ แล้วทำการแก้ไข message อย่างที่ต้องการ
 - ▶ หลังจากนั้นทำการคำนวณ MAC สำหรับเมสเสจที่ต้องการจากคีย์ที่หาได้นี้
- ▶ ในกรณี DES นี้ ต้องลองทั้งหมดประมาณ 2^{64} operation (จริงๆ 2^{56})
- ▶ Attack 2: โจมตีโดยการลองสุ่มเลือก MAC
 - ▶ enemy เลือก message.
 - ▶ สุ่มเลือก 32-bit string และทำการต่อท้าย เมสเสจ
 - ▶ โอกาสที่จะสำเร็จ $\geq 1/2^{32}$. (เพราะ tag มีทั้งหมด ได้ 2^{32} ตัว)

Basic CBC-MAC –Fixed length

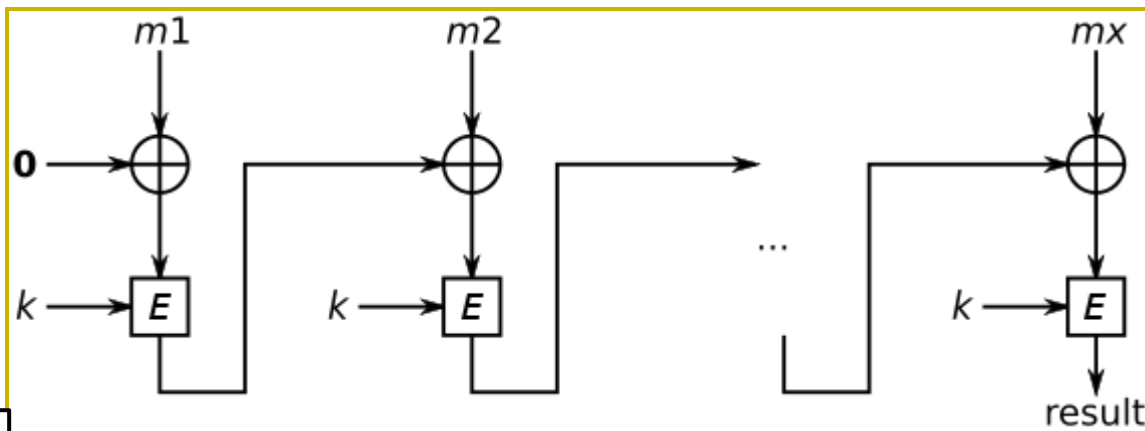
CBC_MAC ที่เกิดขึ้นถ้าจำนวน Block คงที่(Fix length Message) ในการส่งข้อมูลทุกครั้งจะปลอดภัย แต่ถ้าจำนวนบล็อกเปลี่ยนแปลงขึ้นอยู่กับขนาด Message จะไม่ปลอดภัย (Variable Length Message)



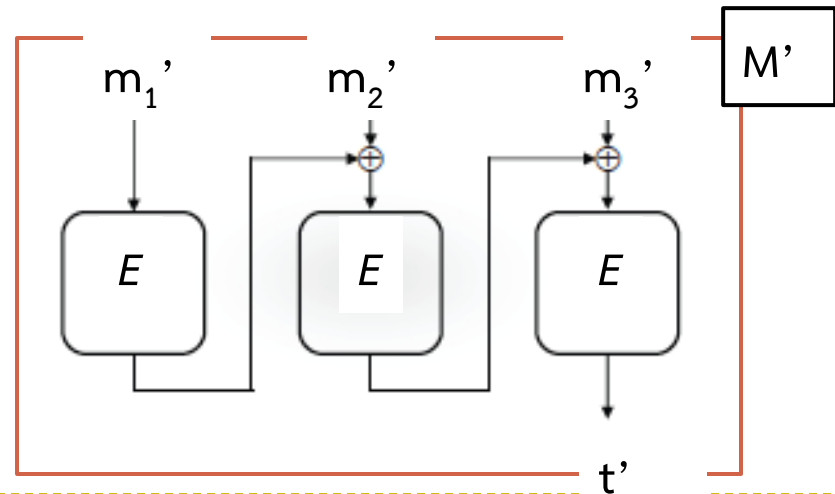
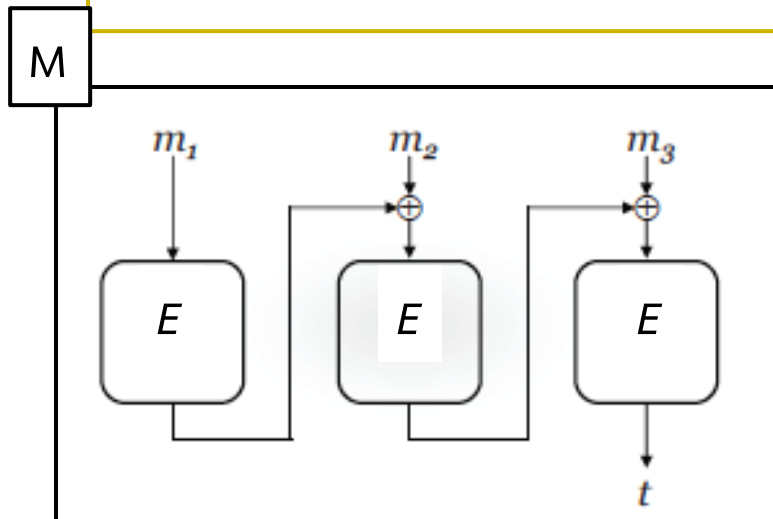
ขนาดไม่
เปลี่ยน
ปลอดภัย

Variable Length CBC-MAC

- ไม่ปลอดภัย (Variable Length Message) ซึ่งอธิบายได้ดังนี้



ไม่ปลอดภัย



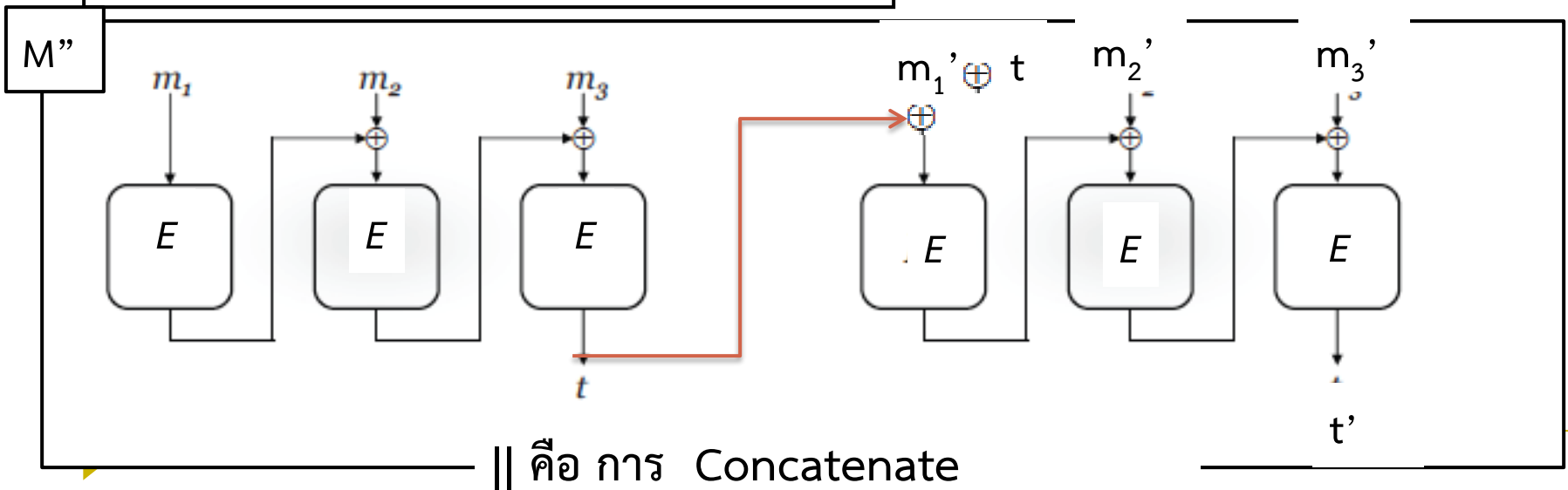
Problem: Variable Length CBC-MAC

- ▶ ไม่ปลอดภัย (Variable Length Message) ซึ่งอธิบายได้ดังนี้
- ▶ ให้ msg 2 msgs คือ M และ M' ซึ่งเมื่อแบ่งเป็นบล็อกแล้วประกอบไปด้วย $m_1 m_2 m_3$ และ $m'_1 m'_2 m'_3$
- ▶ และให้ $t = \text{MAC}_k(M)$ และ $t' = \text{MAC}_k(M')$

การโจมตี

1. ผู้โจมตีสร้าง message ใหม่ M''

$$M'' = M || m'_1 \oplus t || m'_2 || m'_3$$
2. ผู้โจมตีส่ง M'', t'
 เนื่องจาก $\text{MAC}(M') = t'$



Variable Length CBC-MAC ไม่ปลอดภัยอย่างไร

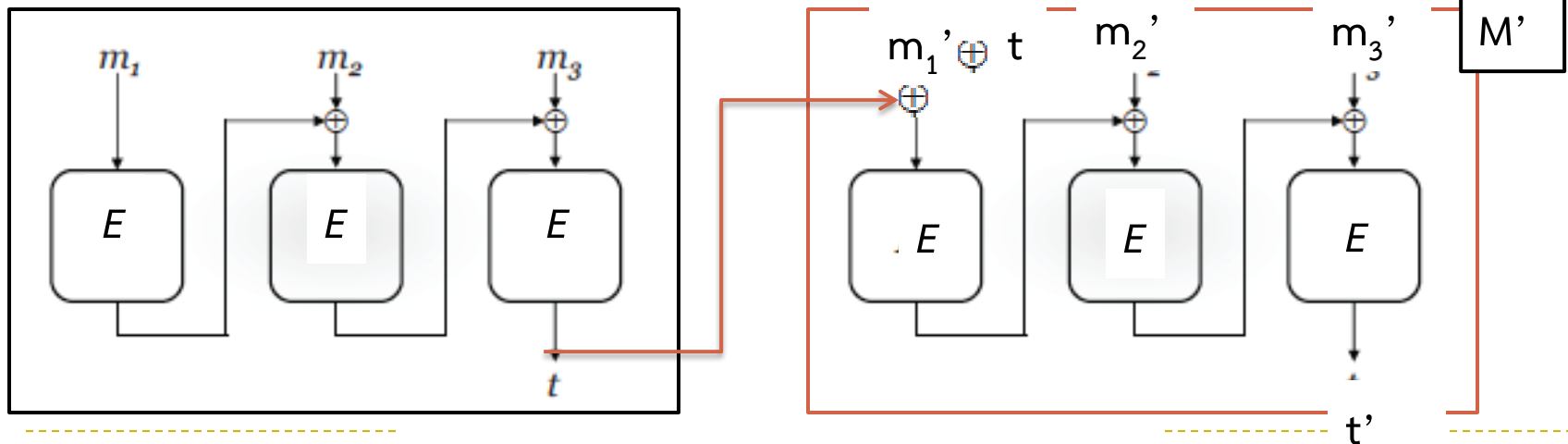
- ▶ พิสูจน์ ว่า สามารถสร้าง MAC ของ Message M'' ได้จาก Message M , M' , t และ t'
- ▶ $M'' = M || m_1' \text{ xor } t || m_2' || m_3'$
- ▶ $MAC(M'') = E(E(E(\text{MAC}(M) \text{ xor } m_1' \text{ xor } t) \text{ xor } m_2') \text{ xor } m_3')$

$$= E(E(E(t \text{ xor } m_1' \text{ xor } t) \text{ xor } m_2') \text{ xor } m_3')$$

$$= E(E(E(m_1') \text{ xor } m_2') \text{ xor } m_3')$$

$$= MAC(M') = t'$$

M



|| คือ การ Concatenate

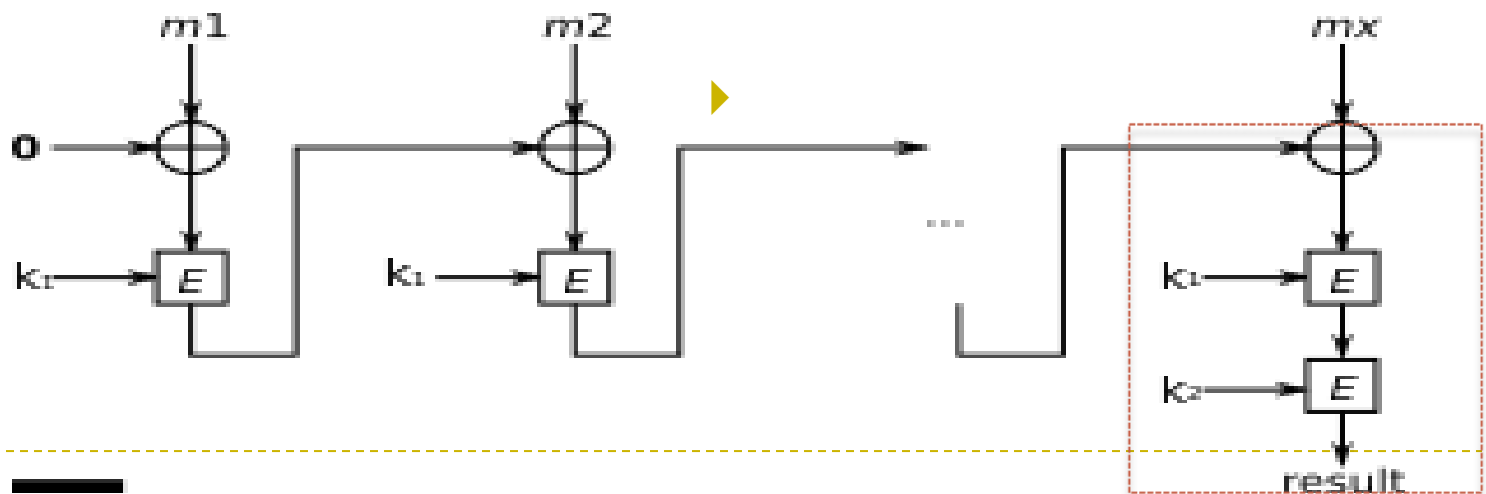
การแก้ปัญหา CBC_MAC variable length

► Prepend message:

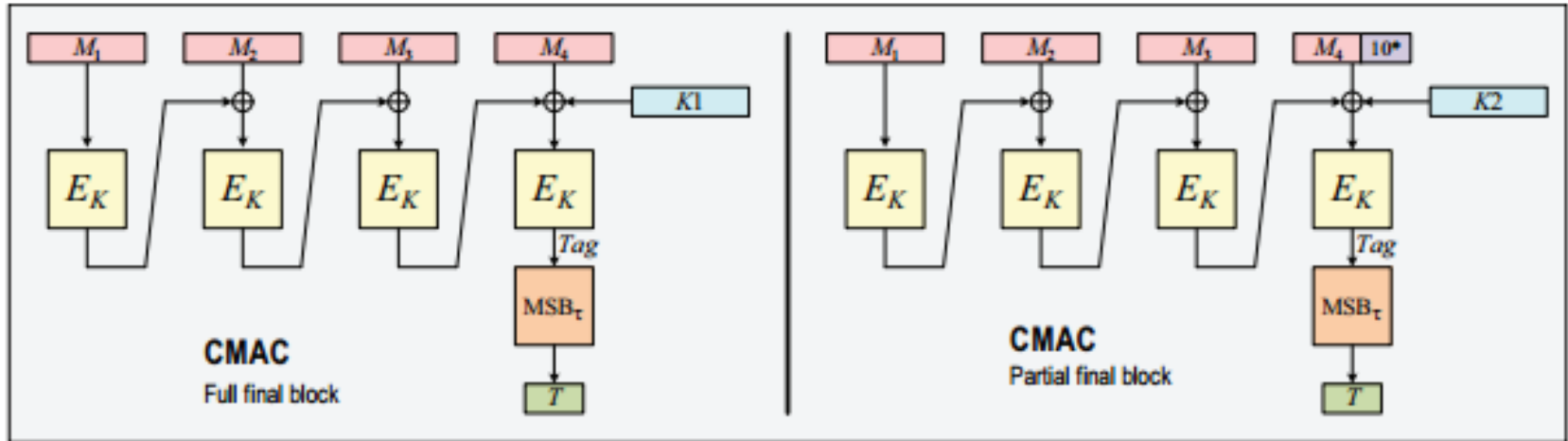
- เติมความยาวของ Message $|M|$ จำนวน n -bit string ไปหน้าข้อความก่อนทำการทำ CBC-MAC ตามปกติ

► Encrypt Last Block (ECBC-MAC)

- เข้ารหัสบล็อกสุดท้ายด้วยคีย์อีกคีย์หนึ่ง
- NIST standard called CMAC
- OMAC
- CCM encryption mode (used in 802.11i)



Cipher-based Message Authentication Code (CMAC)



```

10 algorithm CMACK(M)
11    $K1 \leftarrow \text{dbl}(E_K(0^n))$ 
12    $K2 \leftarrow \text{dbl}(K1)$ 
13   if  $|M| \in \{n, 2n, 3n, \dots\}$ 
14     then  $K' \leftarrow K1, P \leftarrow M$ 
15     else  $K' \leftarrow K2, P \leftarrow M10^i$  where  $i = n - 1 - (|M| \bmod n)$ 
16    $M_1 \dots M_m \leftarrow P$  where  $|M_1| = \dots = |M_m| = n$ 
17   for  $i \leftarrow 1$  to  $m$  do  $C_i \leftarrow E_K(M_i \oplus C_{i-1})$ 
18    $T \leftarrow MSB_\tau(C_m)$ 
19   return  $T$ 

```

CMAC mode

OMAC (One key MAC) [Wiki]

The CMAC tag generation process is as follows:

Divide message into b -bit blocks $m = m_1 \parallel \dots \parallel m_{n-1} \parallel m_n$ where m_1, \dots, m_{n-1} are complete blocks.

(The empty message is treated as 1 incomplete block.)

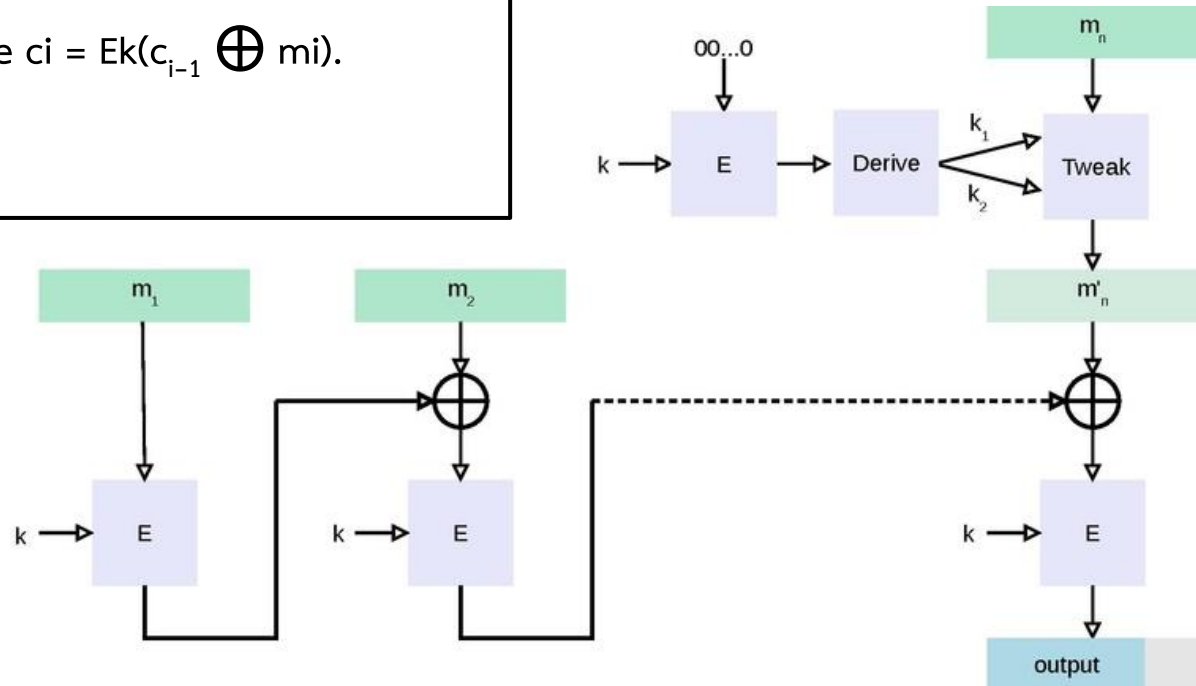
If m_n is a complete block then $m'_n = k_1 \oplus m_n$ else $m'_n = k_2 \oplus (m_n \parallel 10\dots0_2)$. - tweak

Let $c_0 = 00\dots02$.

For $i = 1, \dots, n-1$, calculate $c_i = \text{Ek}(c_{i-1} \oplus m_i)$.

$c_n = \text{Ek}(c_{n-1} \oplus m'_n)$

Output $t = \text{msbl}(c_n)$.



2. ชนิดของ MAC: ใช้ Cryptographic Hash function

- ▶ Three ways to authenticate message using hash functions
สร้าง message authentication code
- ▶ 1. Using symmetric encryption
 - ▶ If the sender and receiver share the encryption key, then authenticity is assured
- ▶ 2. Using public key encryption
 - ▶ Two advantages
 - ▶ provides a **digital signature** as well as message authentication
 - ▶ does not require the **distribution of keys** to communicating parties.
(explained in later lecture)
- ▶ 3. Using secret value



2. ชนิดของ MAC: ใช้ Cryptographic Hash function Cont.

- ▶ These two approaches
 - ▶ เข้ารหัสเฉพาะค่า Hash ของ message
 - ▶ เลยทำงานได้เร็วกว่าการสร้าง MAC จากการเข้ารหัสทั้ง Message
 - ▶ have an advantage over approaches that encrypt the entire message (เช่น ใน CBCMAC) เพราะใช้ computation resource น้อยกว่า..
- ▶ แต่ยังช้ากว่าวิธีที่ 3



2. Three ways to authenticate message using hash functions

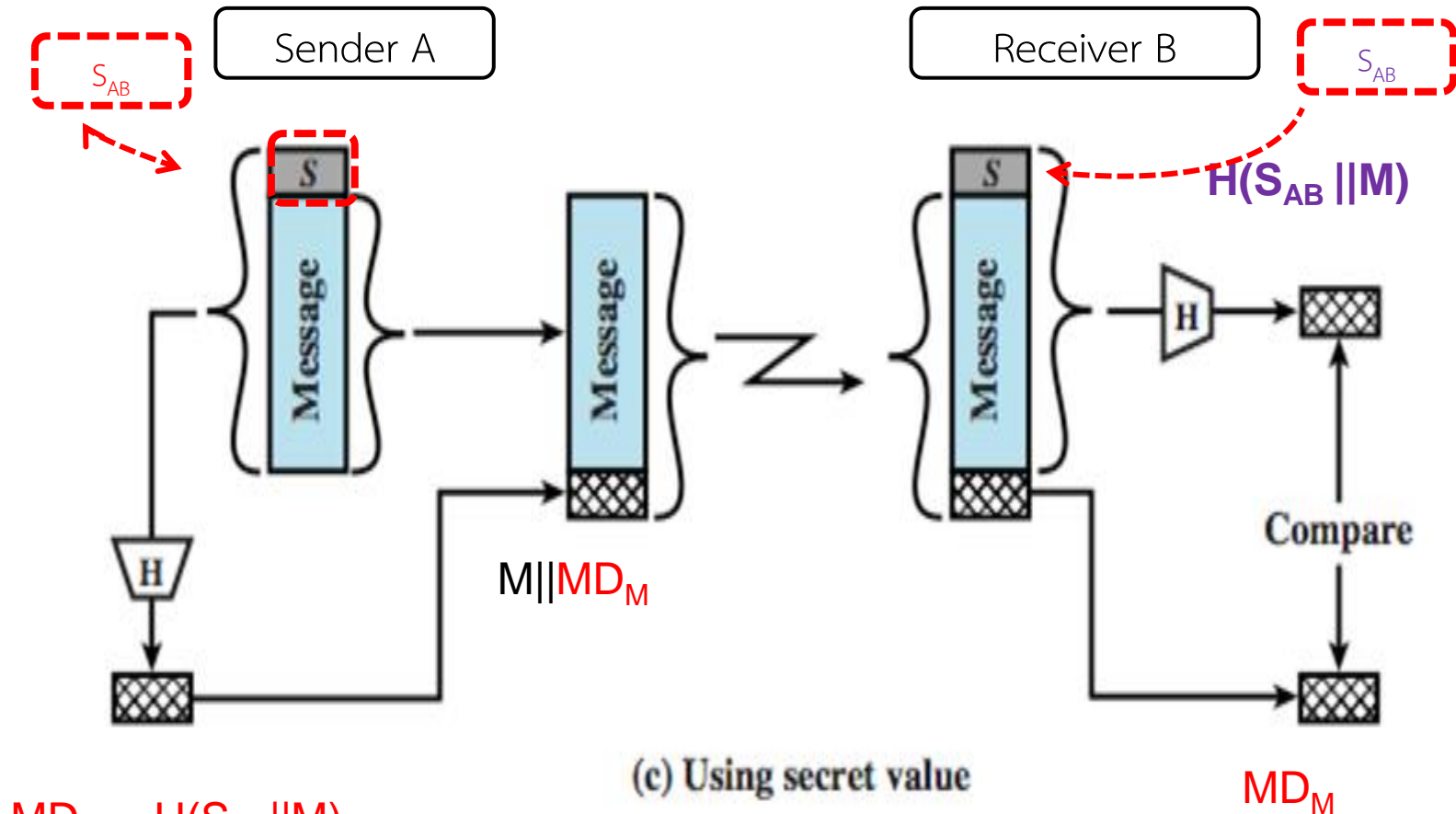
วิธีที่ 3: ใช้ secret value

- ▶ ใช้ hash function แต่ไม่ต้องใช้ encryption ในการทำ message authentication.
- ▶ ให้ A และ B เป็นผู้ที่รับส่งข้อมูลกัน (two communicating parties), ซึ่งในกรณีนี้ A และ B, จะต้องมีการแชร์ common secret value S_{AB} . ไว้ก่อน
 - ▶ เมื่อ A มี message ที่ต้องการส่งไปหา B,
 - ▶ A นำ secret value มาต่อกับ แอสเสจ และ ใช้ hash function หาค่า hash ของ :

$$MD_M = H(S_{AB} || M).$$
 - ▶ A ส่ง $[M || MD_M]$ ไปให้ B.
 - ▶ เพราะว่า B รู้ S_{AB} ,
 - ▶ B นำ secret value ที่ตัวเองมีมาต่อกับแอสเสจที่ได้รับ
 - ▶ ทำการคำนวณ $H(S_{AB} || M)$ และ ตรวจสอบว่าเท่ากับ MD_M . หรือไม่
- ▶ เนื่องจาก secret value ไม่ได้ถูกส่งไปกับ message attacker แม้ว่าจะดักฟัง message ได้ไป แต่ไม่สามารถ modify intercepted message ได้
- ▶ ดังนั้น ตราบใดที่ secret value รู้กันเฉพาะ A และ B attacker จึงไม่สามารถสร้าง แอสเสจปลอมได้

2. Three ways to authenticate message using hash functions

วิธีที่ 3: ใช้ secret value (ภาพ)



Sometimes is called MAC Based on a keyed Hash Functions

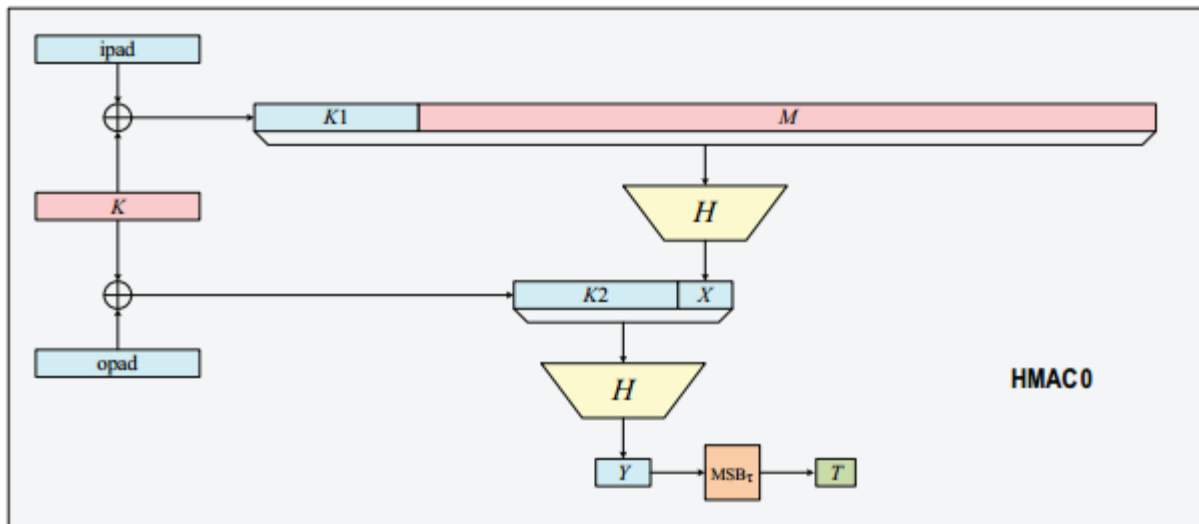
Most widely used MAC on the Internet.

HMAC (1996)

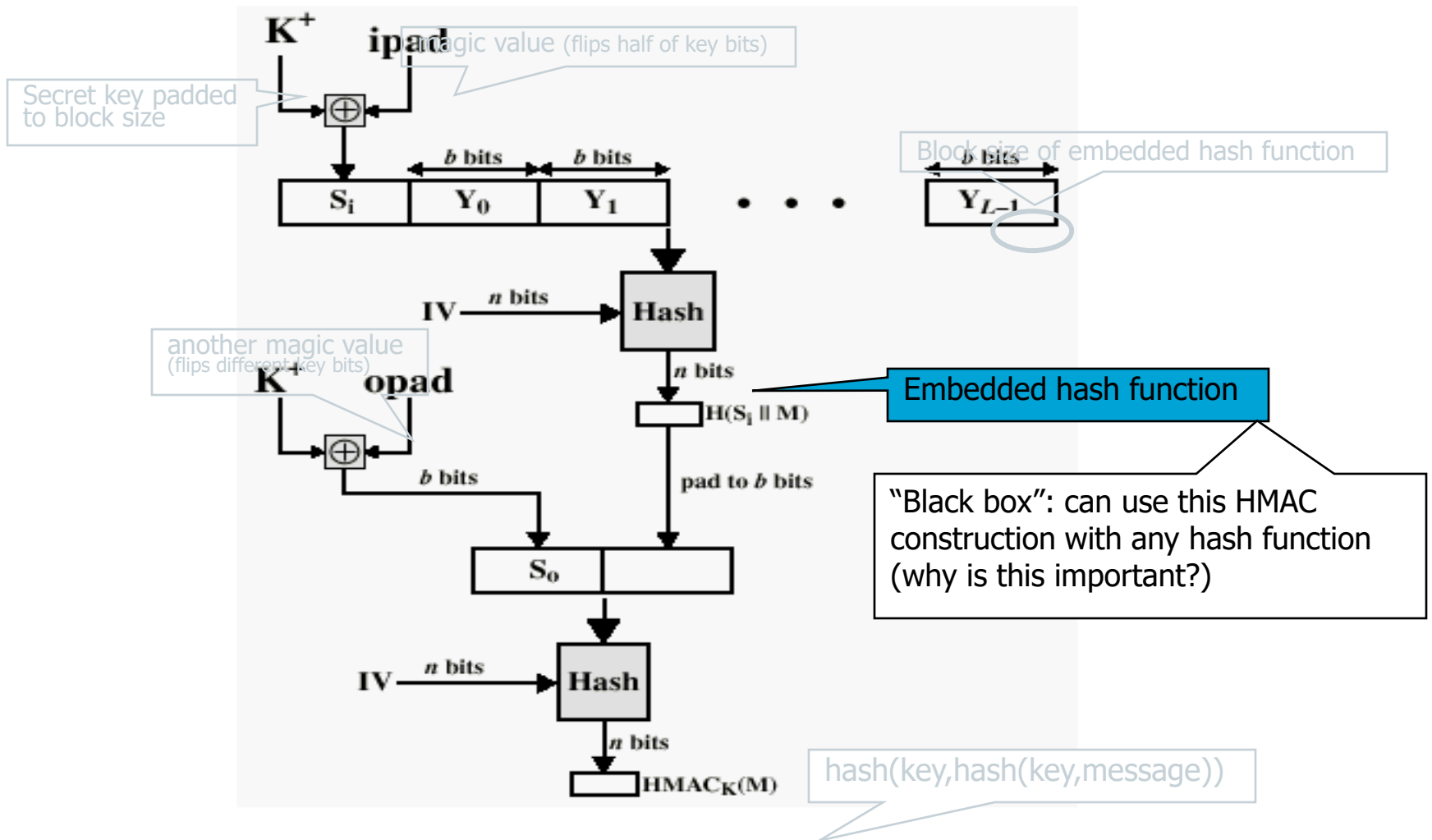
Building a **MAC** out of a hash function:

Standardized method: HMAC

$$S(k, m) = H(k \oplus \text{opad} \parallel H(k \oplus \text{ipad} \parallel m))$$



Structure of HMAC



Most widely used MAC on the Internet.

HMAC (1996)

Building a **MAC** out of a hash function:

Standardized method: HMAC

$$S(k, m) = H(k \oplus \text{opad} || H(k \oplus \text{ipad} || m))$$

H คือ Hash Function เช่น SHA-1

ipad is a pad value of 36 hex repeated to fill block **ipad= 0x3636...3636**

opad is a pad value of 5C hex repeated to fill block **opad=0x5c5c5c...5c5c**

M is the message input to HMAC (including any padding)

ตัวอย่างที่ใช้ HMAC-SHA1 HMAC-MD5

ใช้ใน Applications: IPSec and TLS protocols

RFC 2104. FIPS PUB 198 generalizes. and standardizes the use of HMACs. HMAC-SHA-1 and HMAC-MD5.

Security of HMAC



- ▶ Depends in some way on the cryptographic strength of the underlying hash function
- ▶ Appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC
- ▶ Generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-tag pairs created with the same key

AE Constructions

Cipher + MAC = security



History

Pre 2000: Crypto API's provide *separate* MAC and encrypt primitives

- ▶ Example: Microsoft Cryptographic Application Programming Interface (MS-CAPI) provided HMAC and CBC + IV
- ▶ Every project had to combine primitives in their own way

2000: Authenticated Encryption

- ▶ Bellare and Namprempre in Crypto, 2000
- ▶ Katz and Yung in FSE, 2000



Authenticated Encryption (AE)

- ▶ A term used to describe encryption systems that simultaneously protect confidentiality and authenticity of communications
- ▶ Approaches:
 - ▶ Hash-then-encrypt: $E(K, (M \parallel h))$
 - ▶ MAC-then-encrypt: $T = \text{MAC}(K1, M), E(K2, [M \parallel T])$
 - ▶ Encrypt-then-MAC: $C = E(K2, M), T = \text{MAC}(K1, C)$
 - ▶ Encrypt-and-MAC: $C = E(K2, M), T = \text{MAC}(K1, M)$
- ▶ Both decryption and verification are straightforward for each approach
- ▶ There are security vulnerabilities with all of these approaches

Standards

GCM:	CTR mode encryption then CW-MAC
CCM:	CBC-MAC then CTR mode (802.11i)
EAX:	CTR mode encryption then CMAC

All are nonce-based.

All support *Authenticated Encryption with Associated Data (AEAD)*.

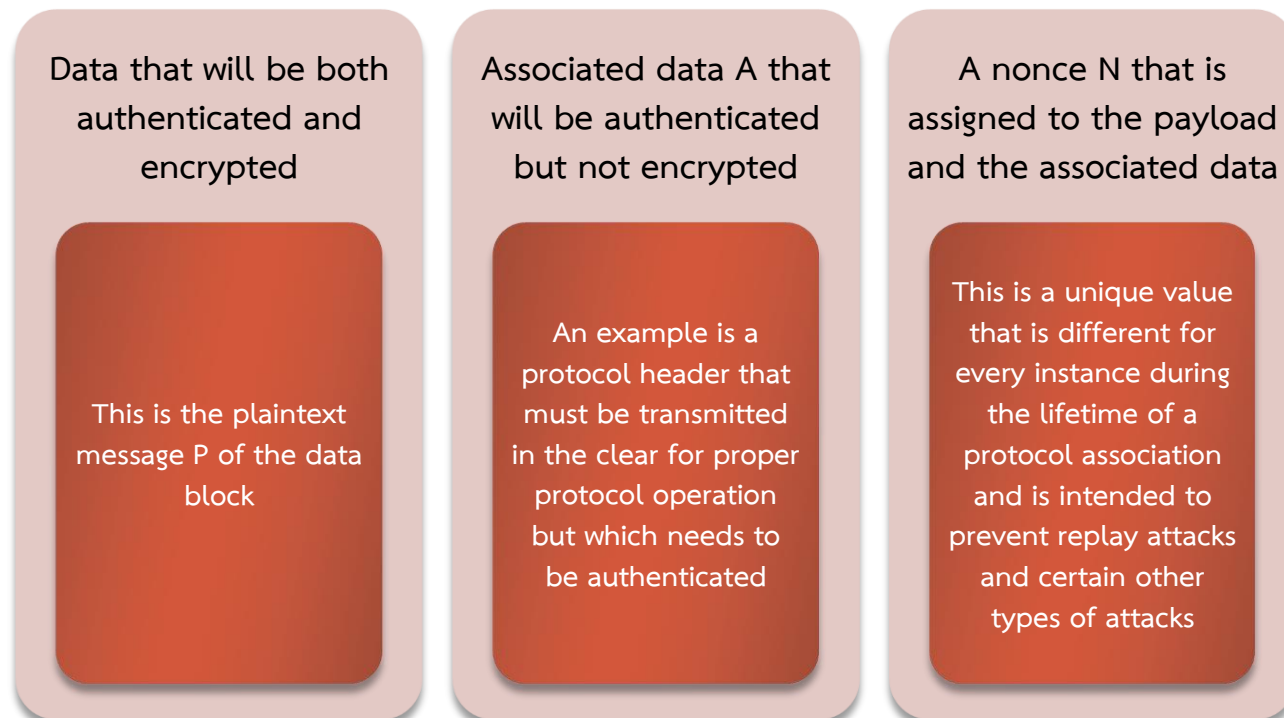


Counter with Cipher Block Chaining-Message Authentication Code (CCM)

- ▶ Was standardized by NIST specifically to support the security requirements of IEEE 802.11 WiFi wireless local area networks
- ▶ Variation of the encrypt-and-MAC approach to authenticated encryption
 - ▶ Defined in NIST SP 800-38C
- ▶ Key algorithmic ingredients:
 - ▶ AES encryption algorithm
 - ▶ CTR mode of operation
 - ▶ CBCMAC => CMAC authentication algorithm
- ▶ Single key K is used for both encryption and MAC algorithms



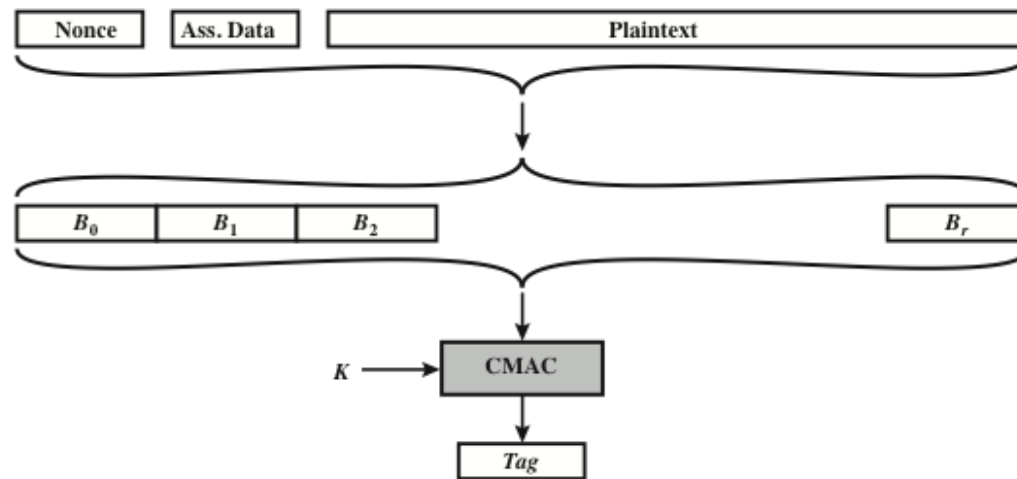
- ▶ The input to the CCM encryption process consists of three elements:



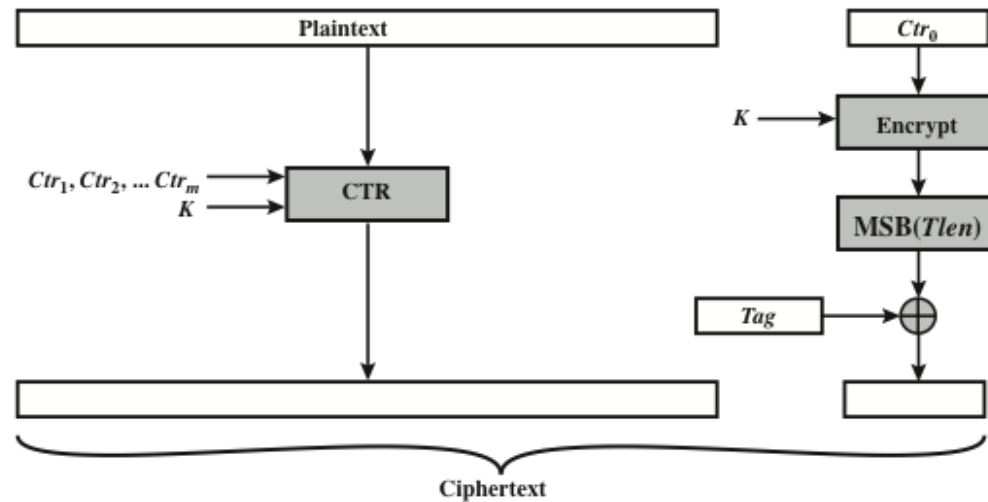
An example API (OpenSSL)

```
int AES_GCM_Init(AES_GCM_CTX *ain,  
                unsigned char *nonce, unsigned long noncelen,  
                unsigned char *key, unsigned int klen )
```

```
int AES_GCM_EncryptUpdate(AES_GCM_CTX *a,  
                          unsigned char *aad, unsigned long aadlen,  
                          unsigned char *data, unsigned long datalen,  
                          unsigned char *out, unsigned long *outlen)
```



(a) Authentication



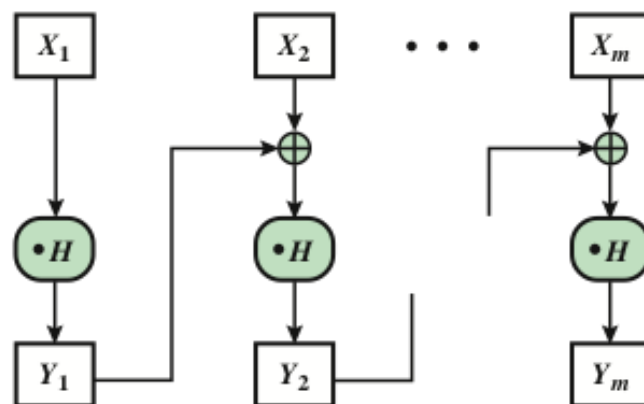
(b) Encryption

Figure 12.9 Counter with Cipher Block Chaining-Message Authentication Code (CCM)

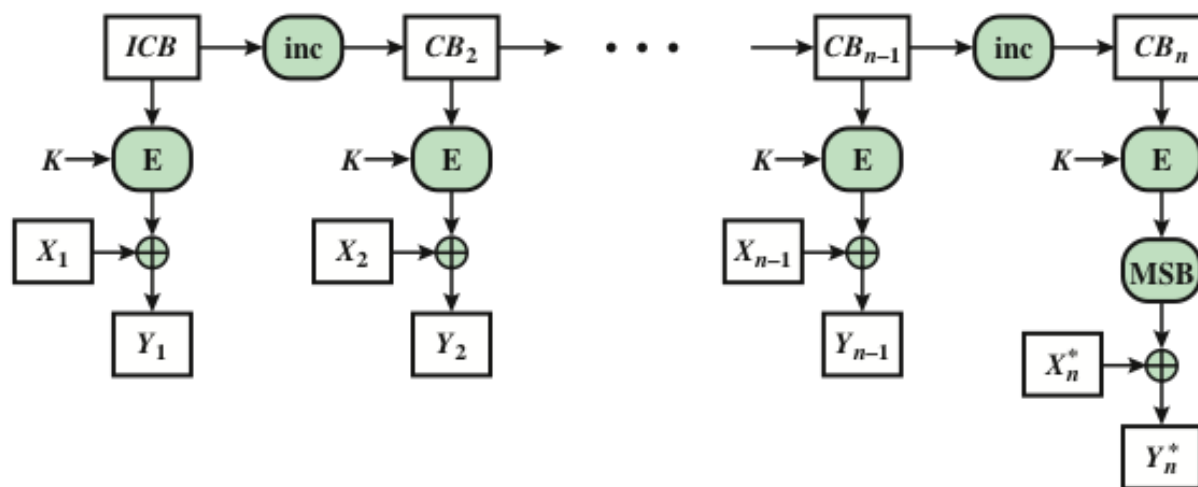
Galois/Counter Mode (GCM)

- ▶ NIST standard SP 800-38D
- ▶ Designed to be parallelizable so that it can provide high throughput with low cost and low latency
 - ▶ Message is encrypted in variant of CTR mode
 - ▶ Resulting ciphertext is multiplied with key material and message length information over $GF(2^{128})$ to generate the authenticator tag
 - ▶ The standard also specifies a mode of operation that supplies the MAC only, known as GMAC
- ▶ Makes use of two functions:
 - ▶ GHASH - a keyed hash function
 - ▶ GCTR - CTR mode with the counters determined by simple increment by one operation





(a) $\text{GHASH}_H(X_1 \parallel X_2 \parallel \dots \parallel X_m) = Y_m$



(b) $\text{GCTR}_K(\text{ICB}, X_1 \parallel X_2 \parallel \dots \parallel X_n) = Y_1 \parallel Y_2 \parallel \dots \parallel Y_n$

Figure 12.10 GCM Authentication and Encryption Functions

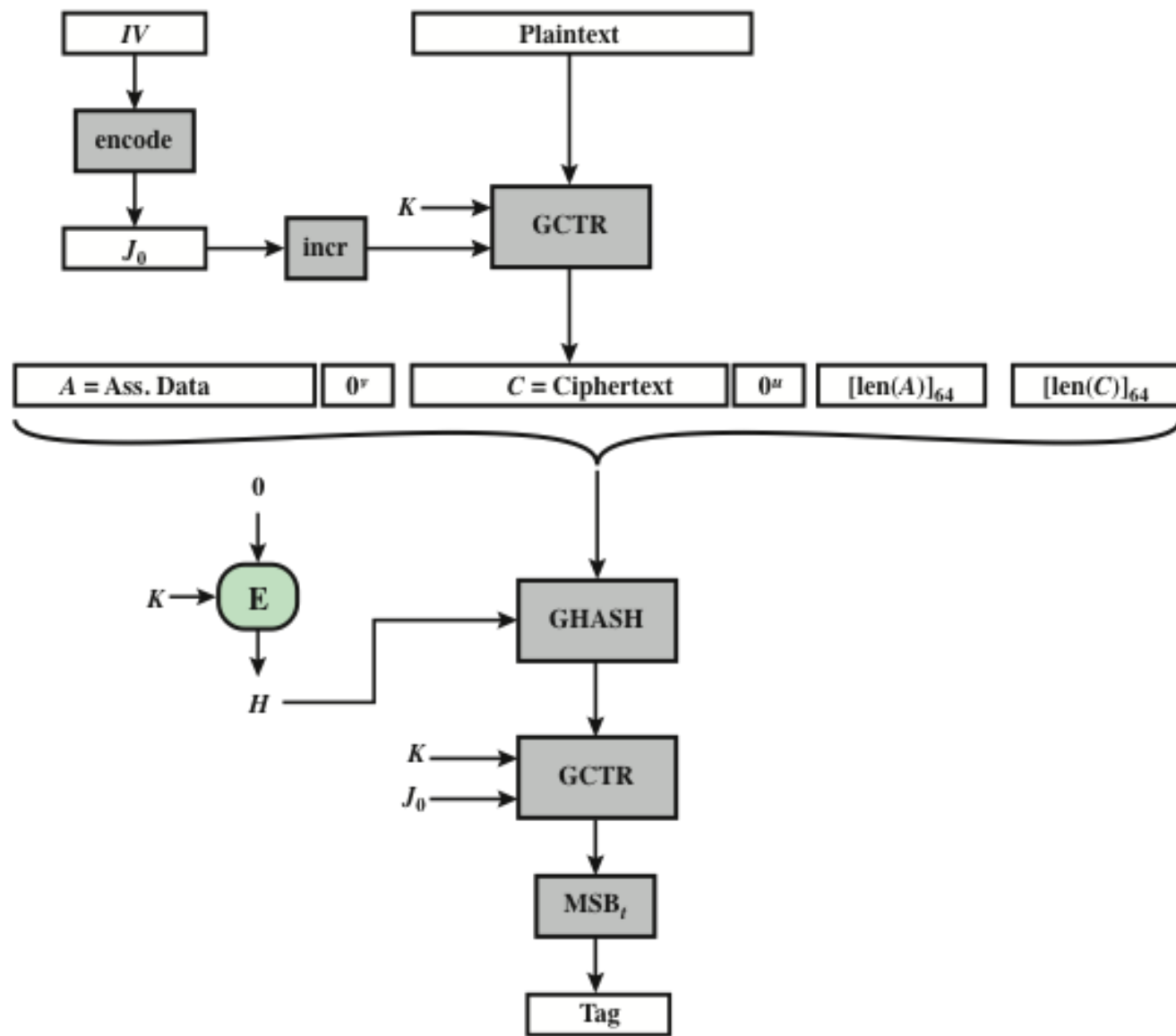


Figure 12.11 Galois Counter - Message Authentication Code (GCM)

Performance

From Crypto++ 5.6.0 [Wei Dai]

AE Cipher	Code Size	Speed (MB/sec)	Raw Cipher	Raw Speed
AES/GCM	Large	108	AES/CTR	139
AES/CCM	smaller	61	AES/CBC	109
AES/EAX	smaller	61	AES/CMAC	109
AES/OCB*	small	129	HMAC/SHA1	147

* OCB mode may have patent issues. Speed extrapolated from Ted Kravitz's results.



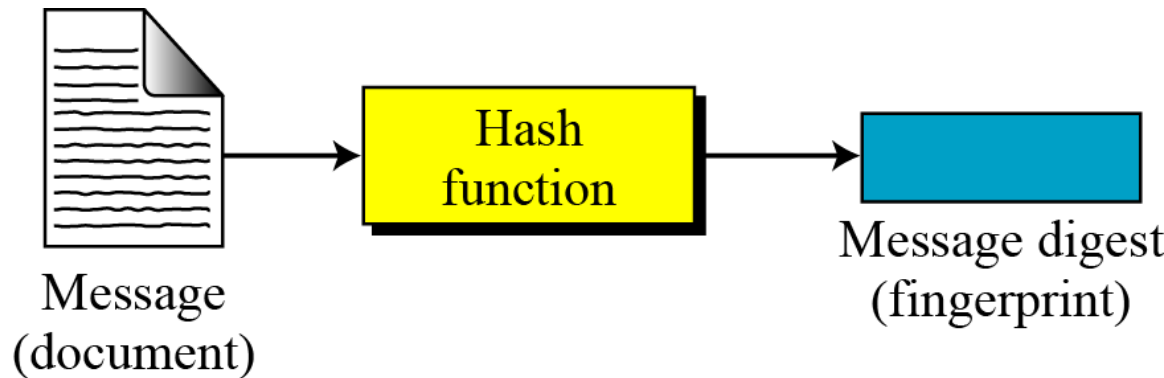
Summary

Encrypt-then-MAC	MAC-then-Encrypt	Encrypt-and-MAC
<ul style="list-style-type: none"> ▶ Provides integrity of CT ▶ Plaintext integrity ▶ If cipher is malleable, we detect invalid CT ▶ MAC provides no information about PT since it's over the encryption 	<ul style="list-style-type: none"> ▶ No integrity of CT ▶ Plaintext integrity ▶ If cipher is malleable, can change message w/o detection ▶ MAC provides no information on PT since encrypted 	<ul style="list-style-type: none"> • No integrity on CT • Integrity of PT can be verified • If cipher is malleable, contents of CT can be altered; should detect at PT level • May reveal info about PT in the MAC (e.g., MAC of same messages are the same)

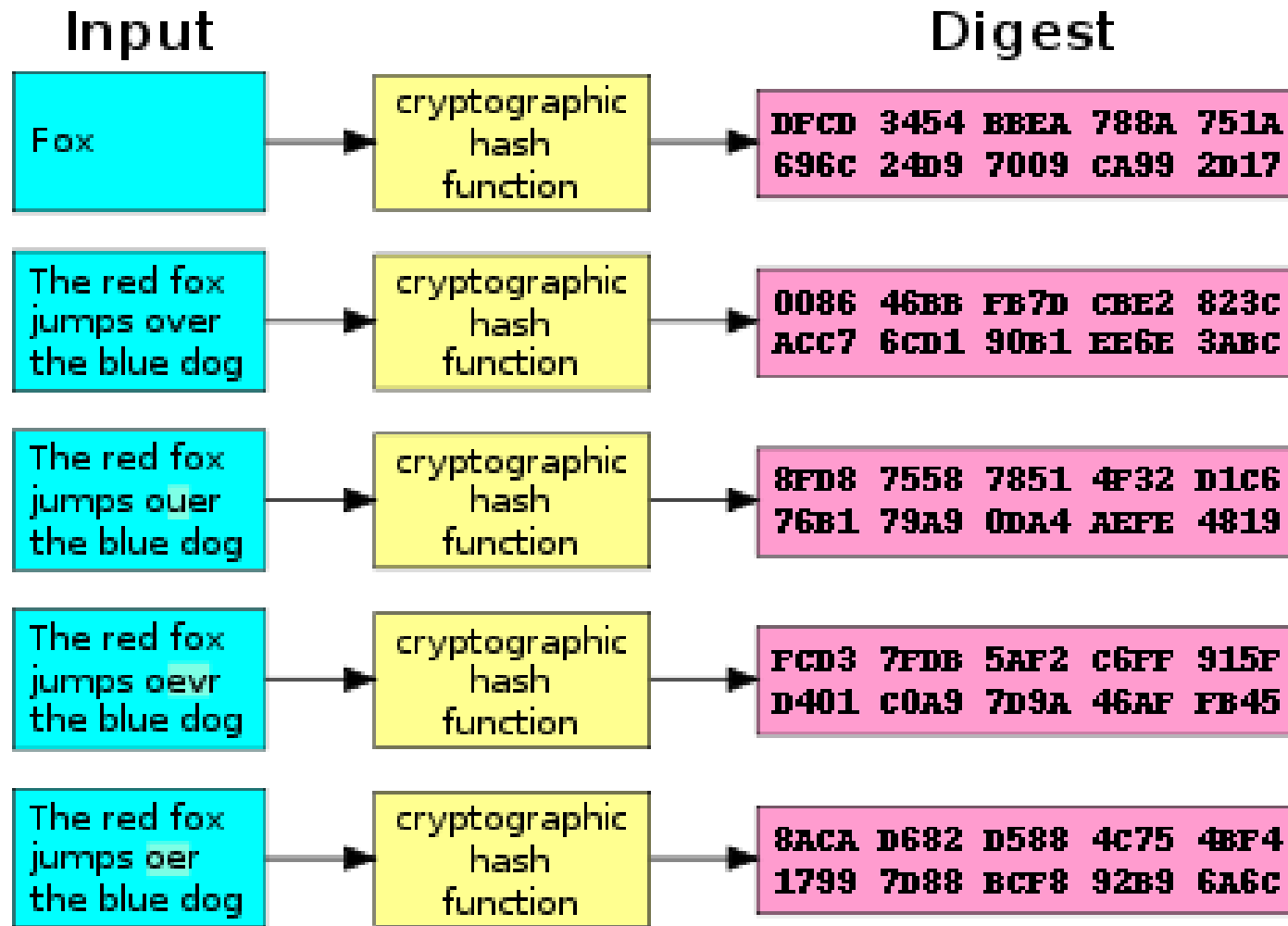


Outline

- ▶ Hash functions.
 - ▶ Cryptographic requirements.
- ▶ Signing with hash functions.
- ▶ Assessing security.
 - ▶ The birthday attack.
- ▶ Techniques for hash functions.
 - ▶ Block ciphers.
 - ▶ Modular arithmetic.
 - ▶ Dedicated or designed.



จากรูป ในช่อง Digest มีสิ่งใดบ้างที่เหมือนกัน??



http://upload.wikimedia.org/wikipedia/commons/thumb/2/2b/Cryptographic_Hash_Function.svg/375px-

► Cryptographic_Hash_Function.svg.png

Hash Function

- ▶ A *hash function* H
 - ▶ accepts a message X of arbitrary size and
 - ▶ outputs a block of fixed size,
 - ▶ called a *message digest* or *hash value*.
- ▶ The idea is that, in general,
 - ▶ $H(X)$ is much smaller than X .
 - ▶ Example: The message is an arbitrary file and the digest is a 128 bit block.

Collision

- ▶ มันเป็นไปได้ที่ msg สอง msg ที่แตกต่างกันจะให้ค่า Hash เดียวกัน
- ▶ เรียกเหตุการณ์ที่เกิดขึ้นนี้ว่า การชนกัน (*collision*)

▶ Example:

$$H(X) = X \bmod 10$$

$$H(56) = H(96) = H(156)$$

- ▶ ใน cryptography ต้องการ hash function ที่ทนทานต่อการชนกัน (*collision resistance*).
- ▶ นั่นคือ, มัน *computationally difficult* ในการหา *inputs* x และ y ที่ $H(x)=H(y)$.



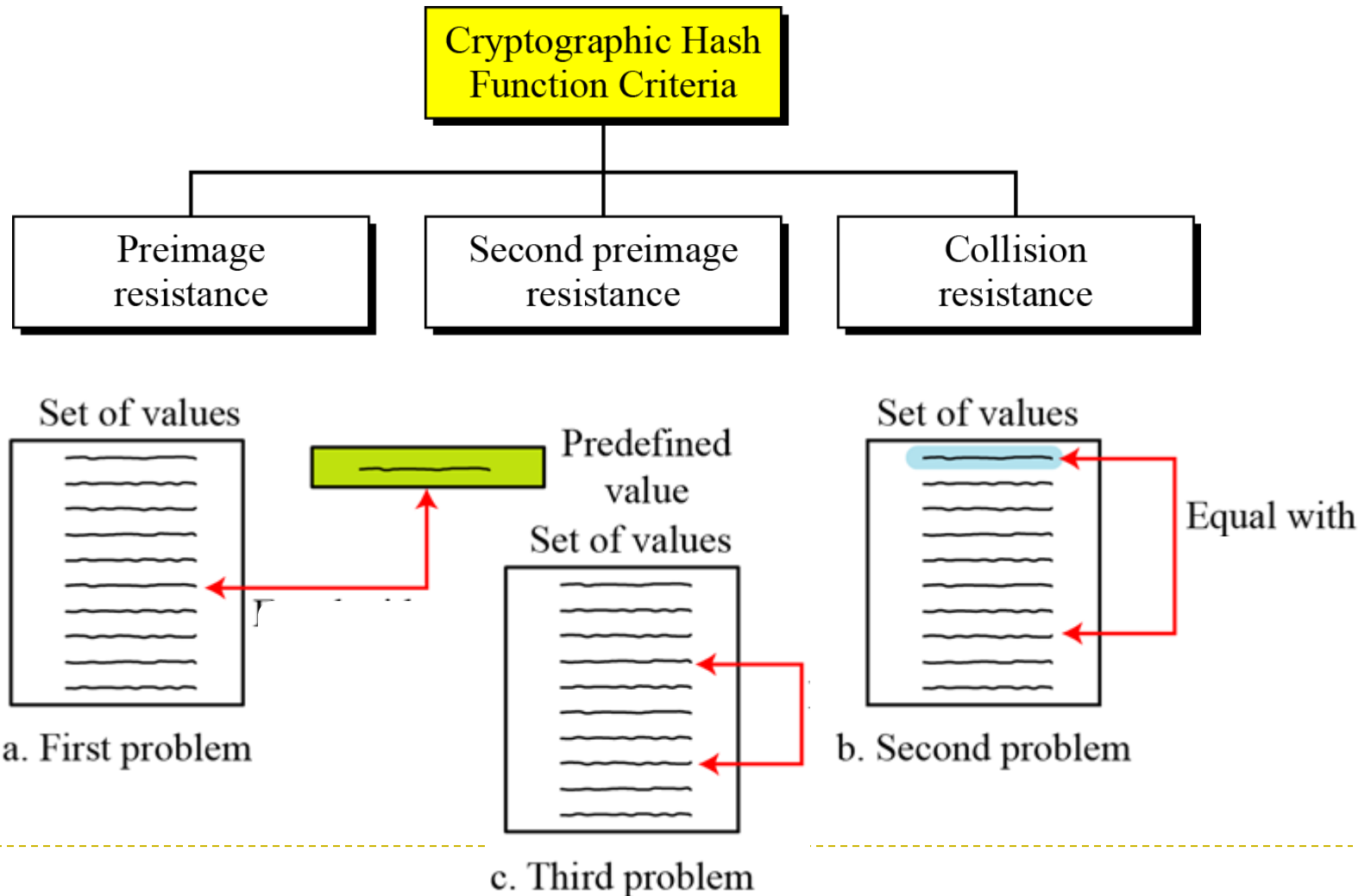
Cryptographic Hash Functions

- cryptographic hash function ต้องมีคุณสมบัติดังนี้:
1. It can be applied to **any size input**. (ใช้ได้กับ input ทุกขนาด)
 2. The **output must be of fixed size**. (output จะต้องมีความยาวเท่าเดิม (fix))
 3. One-way: **Easy to calculate** but **hard to invert** (ง่ายในการคำนวณแต่ยากในการ invert)
 4. Pre-image resistant: For any **given Y**, it is **difficult** to **find** an **X** such that $H(X)=Y$. (เมื่อทราบค่า Y มันยากที่จะหาค่า X ที่ $H(X)=Y$)
 5. Second Pre-image resistant: Given X_1 it should be **difficult** to **find** another X_2 such that $H(X_1)=H(X_2)$. (เมื่อทราบค่า X_1 มันยากที่จะหาค่า X_2 ที่ $H(X_1)=H(X_2)$)
 6. Collision resistant: It is **computationally infeasible** to **find** messages **X** and **Y** with $X \neq Y$ such that $H(X)=H(Y)$. (มัน **computationally infeasible** ที่จะหา message **X** และ **Y** ซึ่ง $X \neq Y$ ที่ $H(X) = H(Y)$)

► Properties 3 and 4 are closely related

Cryptographic Hash Function

► Property 4,5 and 6



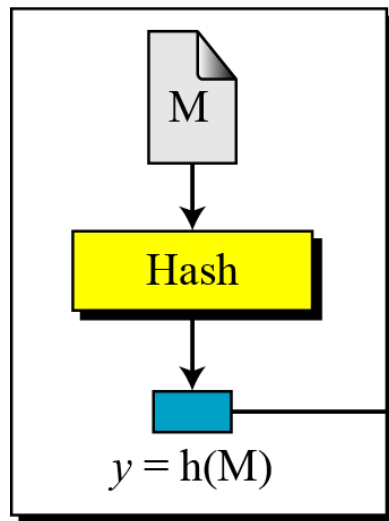
Preimage Resistance: Diagram

Preimage Attack

Given: $y = h(M)$

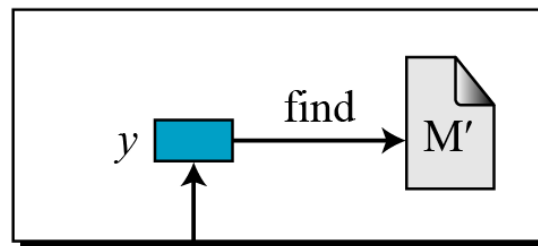
Find: M' such that $y = h(M')$

M: Message
Hash: Hash function
 $h(M)$: Digest



Alice

Given: y
Find: any M' such that
 $y = h(M')$



Eve

To Bob

Second Preimage Resistance

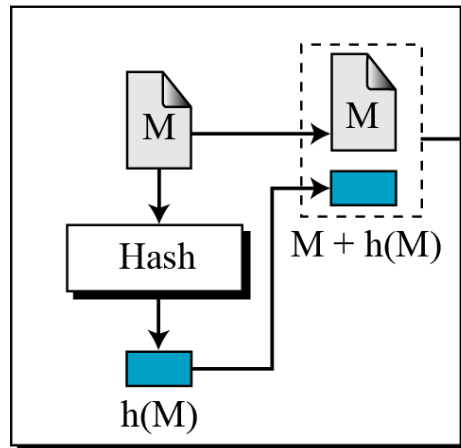
Second Preimage Attack

Given: M and $h(M)$

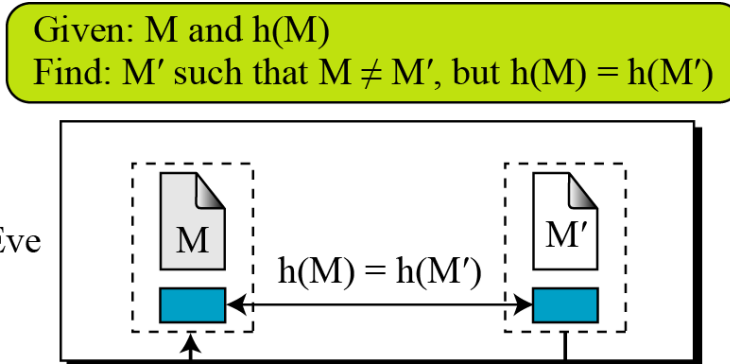
Find: $M' \neq M$ such that $h(M) = h(M')$

M : Message
Hash: Hash function
 $h(M)$: Digest

Alice



Eve



To Bob

Collision Resistance

Collision Attack

Given: none

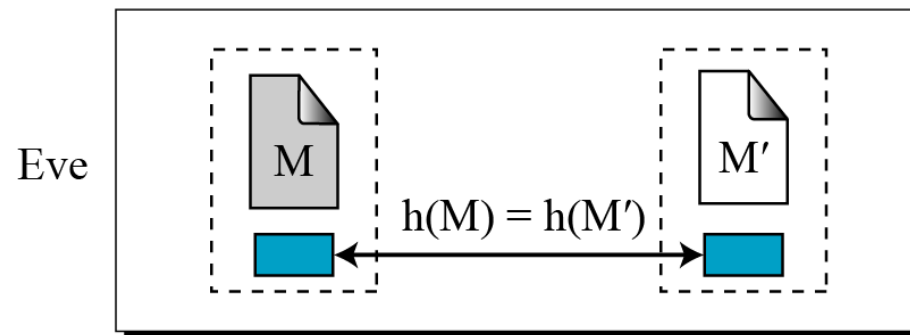
Find: $M' \neq M$ such that $h(M) = h(M')$

M: Message

Hash: Hash function

$h(M)$: Digest

Find: M and M' such that $M \neq M'$, but $h(M) = h(M')$



คำอธิบายเพิ่มเติม การชนกัน (Collisions)

- ▶ ถ้าขนาดของ message (message space) ใหญ่กว่า ขนาดของผลลัพธ์ที่ได้จากการ Hash (digest space) เราจะหา collisions ได้เสมอ (the pigeonhole principle).
- ▶ ตัวอย่างเช่น:
 - ▶ ให้ message เป็นสมาชิกของเซต $Z_p^* = \{1, 2, \dots, p-1\}$, และ digest เป็นสมาชิกของเซต $Z_q^* = \{1, 2, \dots, q-1\}$ ซึ่ง q เป็นจำนวนเฉพาะและ $p > q$.
 - ▶ เลือก g โดยที่ $g \in Z_q^*$.
- ▶ ให้ hash function $h(x) = g^x \bmod q$.

Example: Collision

▶ ถ้า $p=15$, $q=11$, และ $g=3$.

▶ จากการที่ $h(x) = g^x \bmod q$.

$$3^2 = 9 \bmod 11$$

$$3^3 = 5 \bmod 11$$

$$3^4 = 4 \bmod 11$$

$$3^5 = 1 \bmod 11$$

$$3^6 = 3 \bmod 11$$

$$3^7 = 9 \bmod 11$$

→ Collision เกิดขึ้นเมื่อ $x = \underline{\quad}$ and $x = \underline{\quad}$


▶ ถ้าให้ 4 บิตแทน messages ดังนั้น $h(0010) = h(0111)$

▶ ดังนั้น ฟังก์ชันนี้เป็นฟังก์ชัน cryptographic hash ที่ไม่ดี

▶ คำถาม เราจะรู้ได้อย่างไรว่า collision จะมีโอกาสเกิดขึ้นมากน้อยเพียงใด และ ฟังก์ชันนี้มี security แค่ไหน ?? see security assessment



เทคนิคในการทำ Hashing



ส่วนใหญ่จะออกแบบตาม Merkle-Damgard Construction (Iterated Hash Function)

แบ่งได้เป็นสามกลุ่มใหญ่คือ (three main categorie)

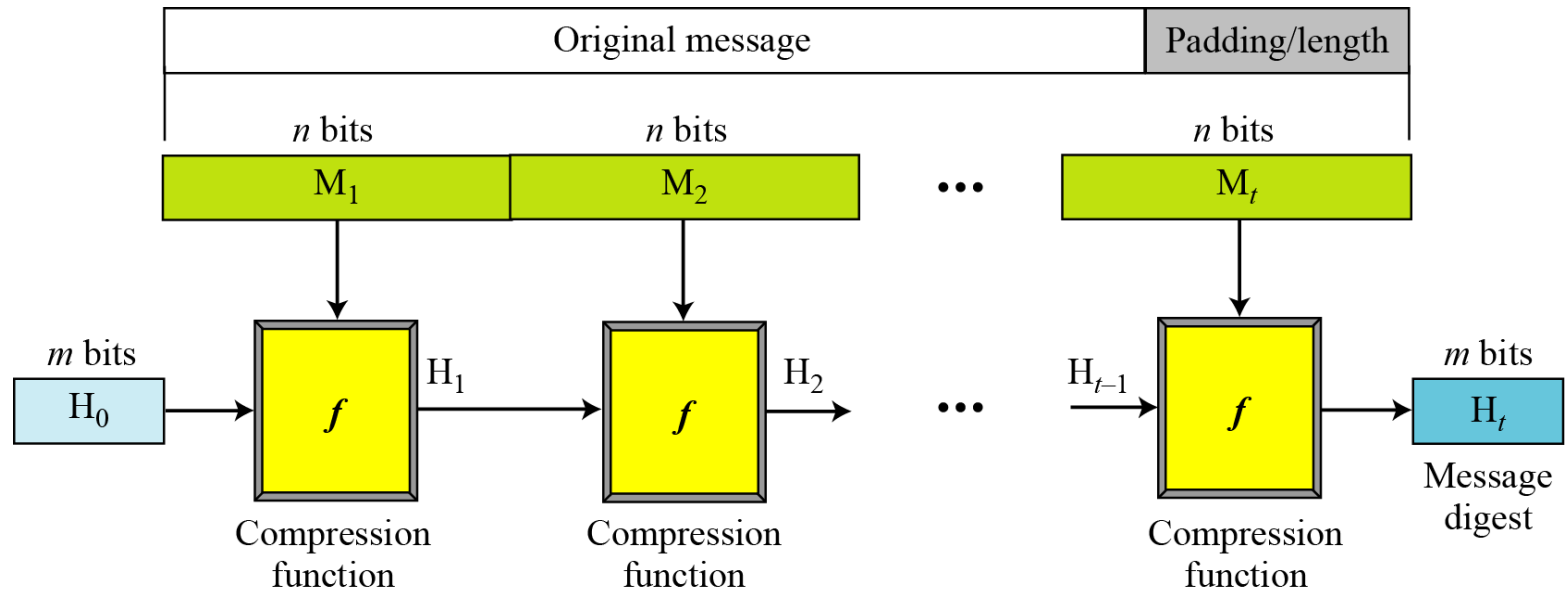
- เทคนิคที่ใช้ block ciphers (symmetric key).
- เทคนิคที่ใช้ modular arithmetic.
- การออกแบบโดยเฉพาะ (Dedicated or designed hash functions (Others)).

Merkle-Damgard Construction

- ▶ แบ่งแมสเสจออกเป็นบล็อกขนาดคงที่
 - ▶ ถ้า message ไม่ครบบล็อกให้ทำการ pad
 - ▶ ต้องเลือกใช้วิธีการ pad ที่ดี มิฉะนั้นอาจทำให้เกิด Collision ได้ง่าย
 - ▶ ในแต่ละบล็อก ใช้ compression function f เดียวกัน
 - ▶ Compression function f ต้องมีคุณสมบัติ one-way function
 - ▶ Input ของ compression function คือ output ของ compression function ของบล็อกก่อนหน้า และ m ของแต่ละบล็อก
 - ▶ สำหรับบล็อกแรกค่าที่ใช้แทน ค่าของ compression function ของบล็อกก่อนหน้า ขึ้นอยู่กับอัลกอริทึม เช่น อาจใช้ความยาวของ message
 - ▶ Output ของ compression function ในแต่ละบล็อกจะเป็น input ของ compression function ของ บล็อกถัดไป
 - ▶ ค่า output ของ Hash บล็อกสุดท้ายคือคำตอบ
-



Merkle-Damgard Diagram



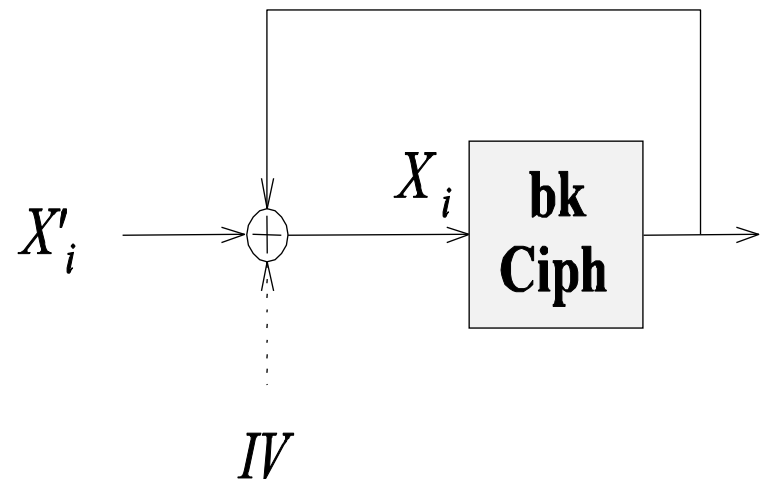
1. Hashing based on Block Cipher

- ▶ เป็นวิธีที่ง่ายที่สุด
- ▶ จริงแล้ว วิธีนี้ก็คือ การสร้าง message authentication code (CBC-MAC) นั่นเอง
- ▶ ซึ่ง ทั้งสองฝั่งต้องทราบคีย์
- ▶ ไม่มีคีย์ทำวิธีนี้ไม่ได้ (เป็น keyed Hash Function)
- ▶

$$X = X_1, X_2, \dots, X_n$$

$$Y_i = E_K(X_i \oplus Y_{i-1})$$

$$H(X) = Y_n$$



1. Hashing based on Block Cipher Cont. I

- ▶ วิธีการนี้ ถูก standardized และใช้ใน banking authentication ใน ANSI 9.9. ANSI 9.19 และ ISO 873-1.

- ▶ ข้อเสีย:
 - ▶ ผู้รับต้องมีคีย์ (**recipient must have the key**)
 - ▶ ถ้า ผู้ไม่ประสงค์ดีรู้ คีย์ จะปลอมแปลงแมสเสจง่ายมาก ไม่มีคุณสมบัติ authentication เลย



1. Hashing based on Block Cipher Cont. II

▶ ปัญหา

▶ สำหรับ n -bit block Y ใดๆ ถ้าวาง n -bit blocks X ซึ่งมีลำดับของแมสเสจคือ X_1, X_2, \dots, X_w

▶ ถ้า n -bit block X_{w+1} มีค่าเท่ากับ $D_K(Y) \oplus Y_w$

▶ $X_{w+1} = D_K(Y) \oplus Y_w$

▶ เมื่อ Y_w คือ hash value ของ X_1, X_2, \dots, X_w .

▶ Message ใหม่ ที่มีจำนวน $w+1$ บล็อกจะให้ค่า hash เหมือนกับ ค่า Hash ของแมสเสจ Y เนื่องจาก xor จะ cancel ค่า $Y_w \Rightarrow$ Collision

▶ การใช้ block ciphers ในการสร้าง secure hash functions ไม่ใช่เรื่องง่าย หลายๆ scheme broken เรียบร้อยแล้ว

2. Hash functions based on modular arithmetic: An example.

- ▶ Quadratic Congruential Manipulation Detection Code (QCMDC) (Jueneman (1983))

- ▶ เป็น *keyed hash function*.

- ▶ Compression Function เป็น สมการ Quadratic

- ▶ แบ่ง message ออกเป็น block ขนาด m บิต

- ▶ เลือก M ที่เป็นจำนวนเฉพาะ และ มีค่า $\geq 2^{m-1}$. ขึ้นไป

- ▶ เครื่องหมาย + คือ integer addition.

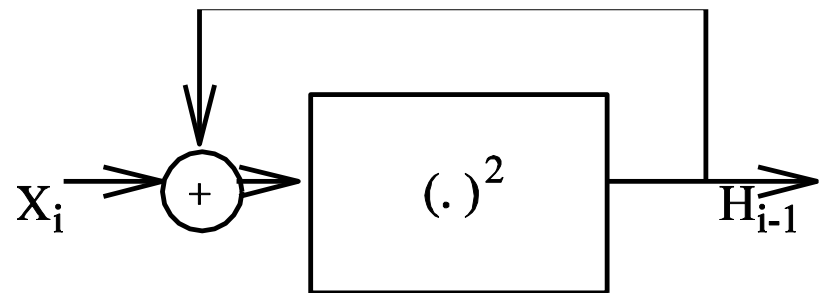
- ▶ H_0 เป็นค่าสุม *secret initial value* ซึ่งคือ คีย์นั่นเอง (the key).

- ▶ M is a prime satisfying $M \geq 2^{m-1}$.

- ▶ สมการคือ

$$H_i = (H_{i-1} + X_i)^2 \mod M$$

- ▶ H_n is the hash value.



- ▶ This scheme is **broken**. (Coppersmith)

3. Designed hash functions: MD5

- ▶ MD5 (คิดโดย Ron Rivest 1992).
- ▶ เป็น Crypto Hash Function ที่รู้จักกันมาก (best known hash algorithm).
 - ▶ ใช้ Merkle-Damgard Construction
 - ▶ ประมวลผล Input แต่ละบล็อกขนาด **512-bit**
 - ▶ สร้าง **message digest (Hash value)** ขนาด **128-bit** เสมอ
 - ▶ MD5 พัฒนามาจาก MD4. (ช้ากว่า MD4 แต่ security สูงกว่ามาก)
 - ▶ ออกแบบมาเพื่อใช้กับ เครื่องแบบ 32 บิต
- ▶ Collisions ใน MD5 ใน practical time ถูกค้นพบในปี 2004
- ▶ ปี 2005 มีผู้นำเสนอวิธีการการหา collision ใน MD5 ที่เร็วกว่าวิธีการใน 2004 มาก
- ▶ Completely broken!!!! นำมาใช้กับ Digital Signature ไม่ได้



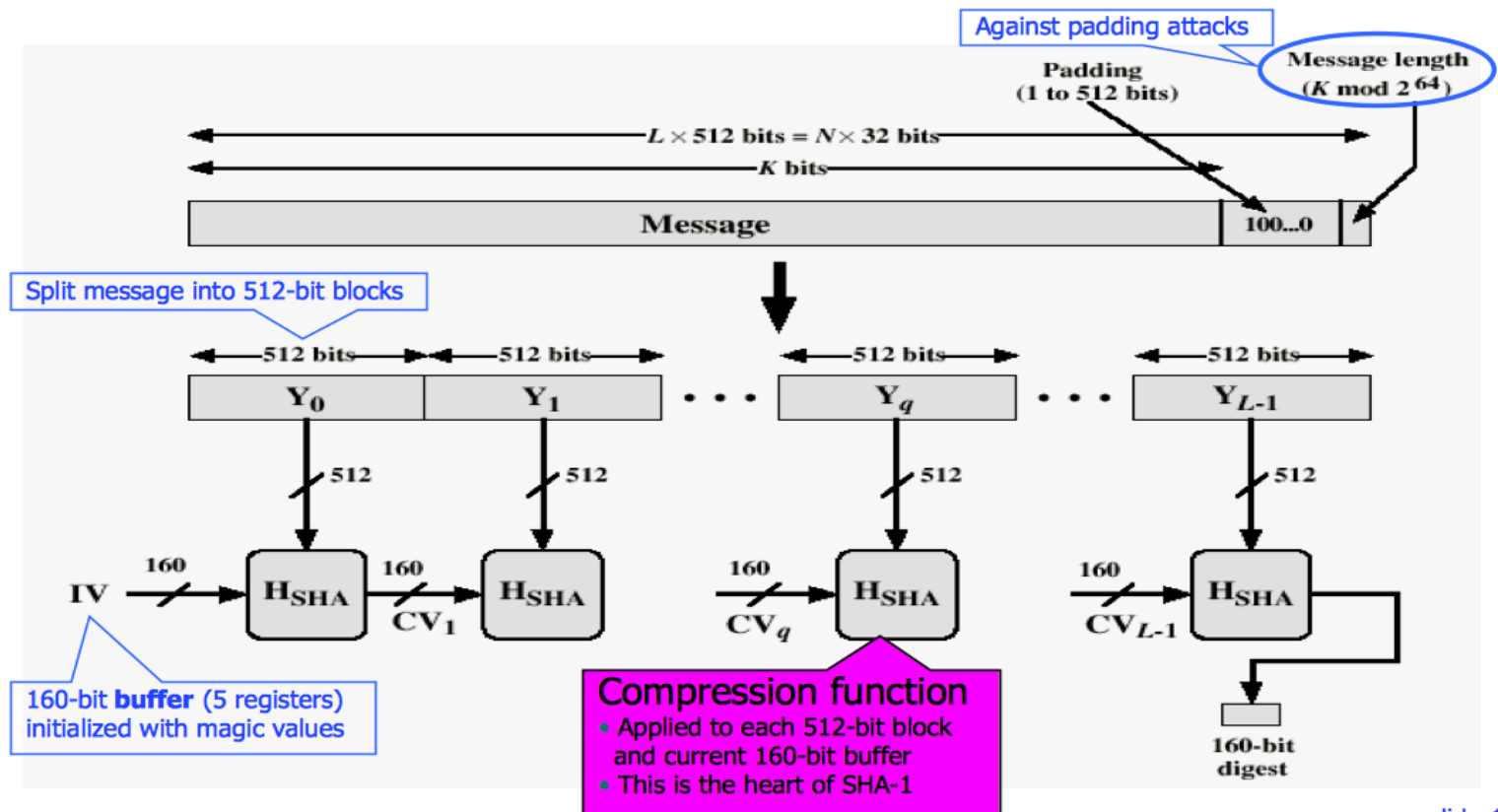
2. Designed hash functions: SHA-1

- ▶ เป็น Crypto Hash อัลกอริทึมที่นำเสนอ โดย NIST เพื่อใช้กับ DSA standard.
- ▶ พัฒนามาจาก SHA-0.
- ▶ ให้ output **160 bit message** digest วิธีการที่ใช้จะคล้ายๆ กับ MD5.
- ▶ On the next page we show a single SHA-1 operation.
- ▶ The plus boxes are addition mod 2^{32} .
- ▶ The non-linear function and K_t vary for different operations.



SHA-1 Cont.

Basic Structure of SHA-1



slide 1

คำอธิบาย

▶ ใน SHA-1 message input แต่ละบล็อกต้องมีขนาด **512-บิต** (เช่นเดียวกับใน MD5).

▶ ในแต่ละ compression function (แต่ละรอบ)

▶ ประกอบไปด้วย 4 sub-round

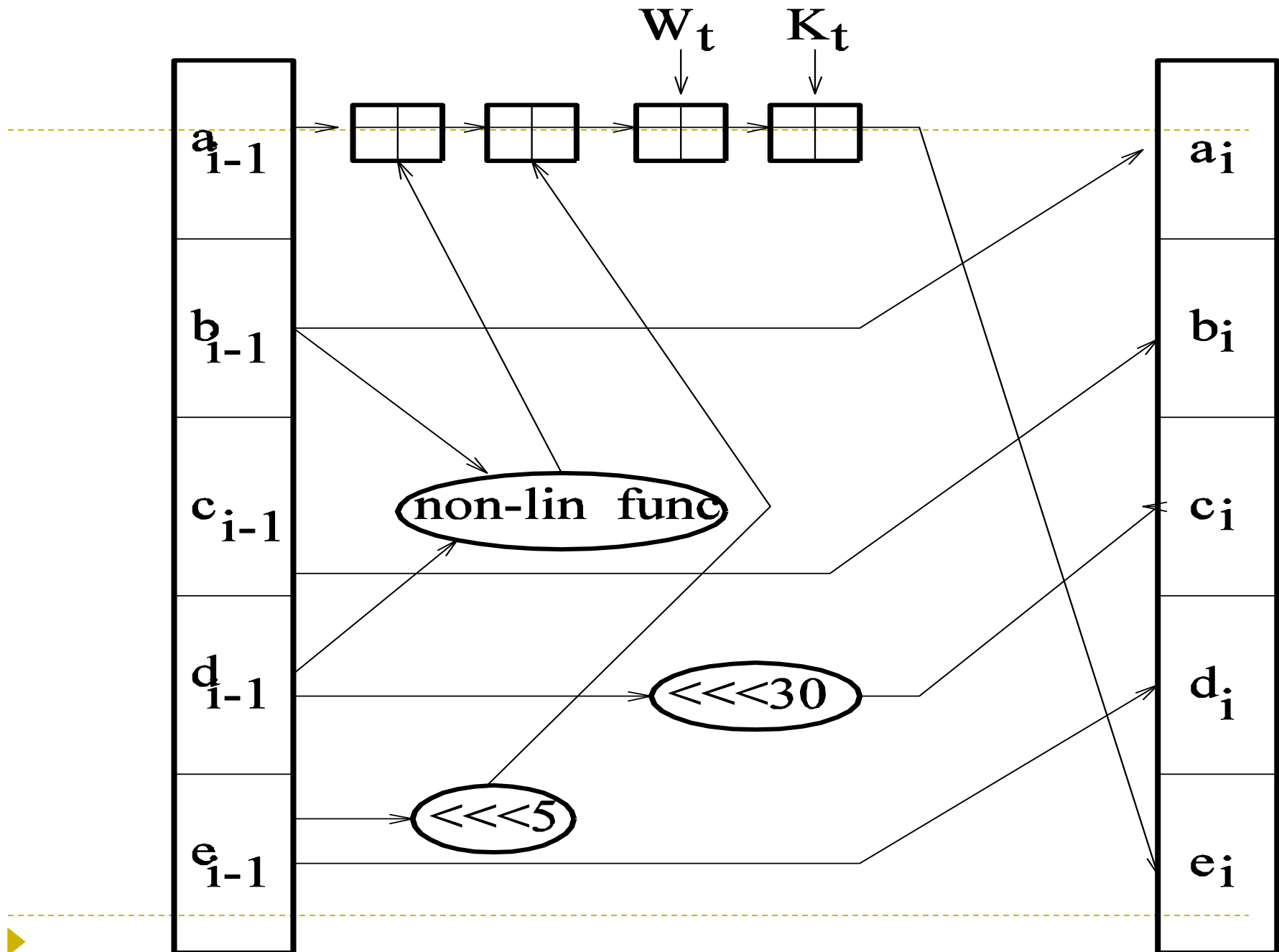
▶ แต่ละ sub-round มี 20 operation

▶ nonlinear function ใช้ในแต่ละรอบจะแตกต่างกัน

รับ input จากบล็อกก่อนหน้านี้ ผ่านทาง register 5 ตัวคือ a b c d และ e

▶ $W_t, i=0,1 \dots 79$ จะมีขนาด 32 บิต blocks สร้างจาก input ในส่วนที่เป็น
แมสเสจ 512 บิต





SHA-1: ปัญหา

- ▶ 2005 SHA-1 มีผู้หา SHA1 collision ได้ ใน 2^{69} operation ซึ่งเร็วกว่าการทำ Brute force (2^{80} operation)
- ▶ ปลายปี 2005 ถูกหา collision ได้ใน 2^{63} operation
- ▶ ดังนั้น SHA-1 ไม่ปลอดภัย ถ้างานที่ทำนั้นที่ต้องไม่มี Collision
- ▶ แต่ถ้า ต้องการเพียงคุณสมบัติ one-wayness (ยังใช้ได้อยู่)
- ▶ จริงๆ แล้ว NIST มีแผนจะให้เลิกใช้ SHA-1 ภายในปี 2010 ตั้งแต่ปี 2005
- ▶ Nov 2013 ไมโครซอฟต์ประกาศ *
 - ▶ จะเลิก support SHA-1 ใน root certificate program โดยจะเริ่มทำ วันที่ 1 มค 2016 และจะทำให้เสร็จก่อนวันที่ 1 มค 2017
 - ▶ Certificate ที่ Sign โดยใช้ SHA-1 วินโดวส์จะไม่ support ตั้งแต่วันที่ 1 มค 2016

**[http://www.zdnet.com/google-accelerates-end-of-sha-1-support-certificate-authorities-nervous-](http://www.zdnet.com/google-accelerates-end-of-sha-1-support-certificate-authorities-nervous-7000033159/)

SHA-1: ปัญหา Cont.

- ▶ Sep 2014 google ประกาศว่า
 - ▶ จะเลิก accept (SSL ของ เว็บไซต์ ที่ใช้ SHA-1 ว่าปลอดภัย ใน Chrome โดยจะค่อยๆ เลิกเป็น phase แต่จะทำให้เสร็จภายในปี 2017
- ▶ Mozilla มีแผนที่จะเลิก accept เว็บไซต์ที่ใช้ SHA-1 ใน Certificate เช่นเดียวกัน
- ▶ NIST ปรับปรุง FIPS 180-2 ในปี 2002 และ
 - ▶ ทำการเพิ่ม SHA อีก 3 เวอร์ชัน และเรียกรวมกันว่า SHA2
 - ▶ SHA-256, SHA-384, SHA-512
 - ▶ ตัวเลข 256/384/512 แสดงถึงขนาด(เป็นบิต) ของ hash value
 - ▶ ใช้ basic structure เดียวกับ SHA-1 แต่มี security ที่ดีกว่า
 - ▶ ไม่มีรายงานการ attack SHA2 ที่ สำคัญ
- ▶ ปัจจุบัน มี SHA-3 หรือเรียกอีกชื่อว่า Keccak NIST ออกมาตรฐาน ในปี 2014 มาเพื่อเป็นทางเลือก แต่ SHA-2 เพราะยังไม่ถูก attack สำคัญ



Other hash functions

- ▶ HAVAL.
- ▶ RIPEMD-(160).
- ▶ Snefru.
- ▶ Tiger.
- ▶ WHIRLPOOL. (NESSIE*** endorsed hash)
 - ▶ developed by Vincent Rijmen & Paulo Barreto
 - ▶ input is processed in 512-bit blocks.
 - ▶ based on the use of a block cipher,
 - ▶ The block cipher W has a similar structure and uses the same elementary functions as AES, but with a 512-bit block and key size.
 - ▶ produces 512-bit hash



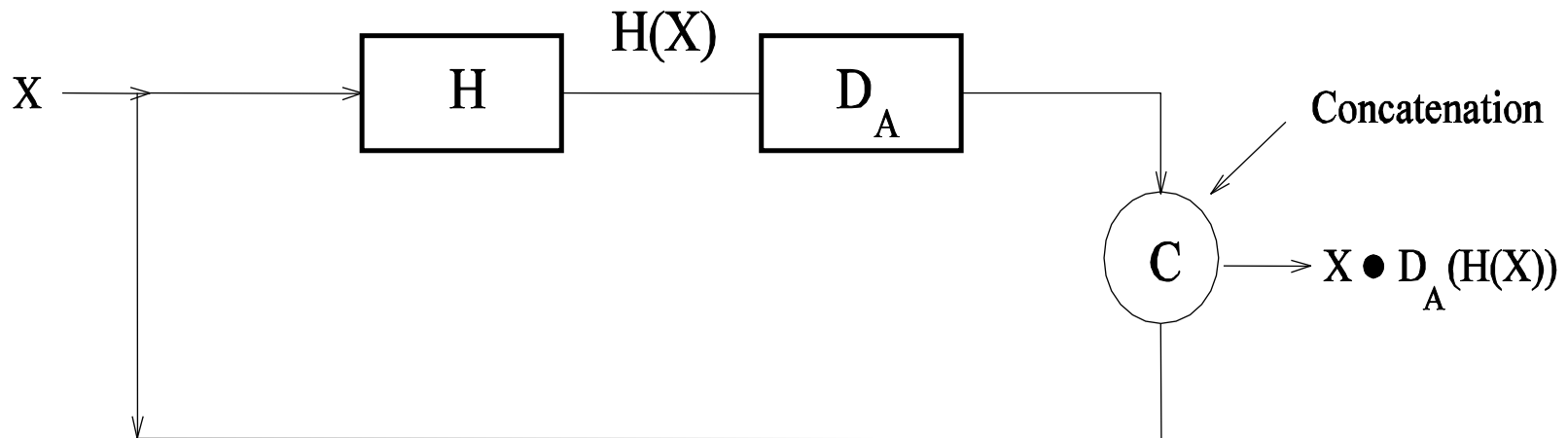
การทำ Cryptographic Hash Function ไปใช้ประโยชน์

- ▶ ใช้ใน Digital Signature
- ▶ ใช้สร้าง Message Integrity (Message Authentication Code)
- ▶ ใช้สร้าง Integrity ของ ข้อมูล
- ▶ ใช้ในการเข้ารหัส password

Hash Function ใน Digital Signature

1. Signing with Hash Function

- ▶ main applications ของ hash function คือ **digital signatures**.
- ▶ แทนที่จะ Sign message โดยตรง message จะถูกนำไปเข้า Hash Function ก่อน แล้วจึงนำไปทำการ sign (ใช้ $H(X)$ เป็น input แทน X)

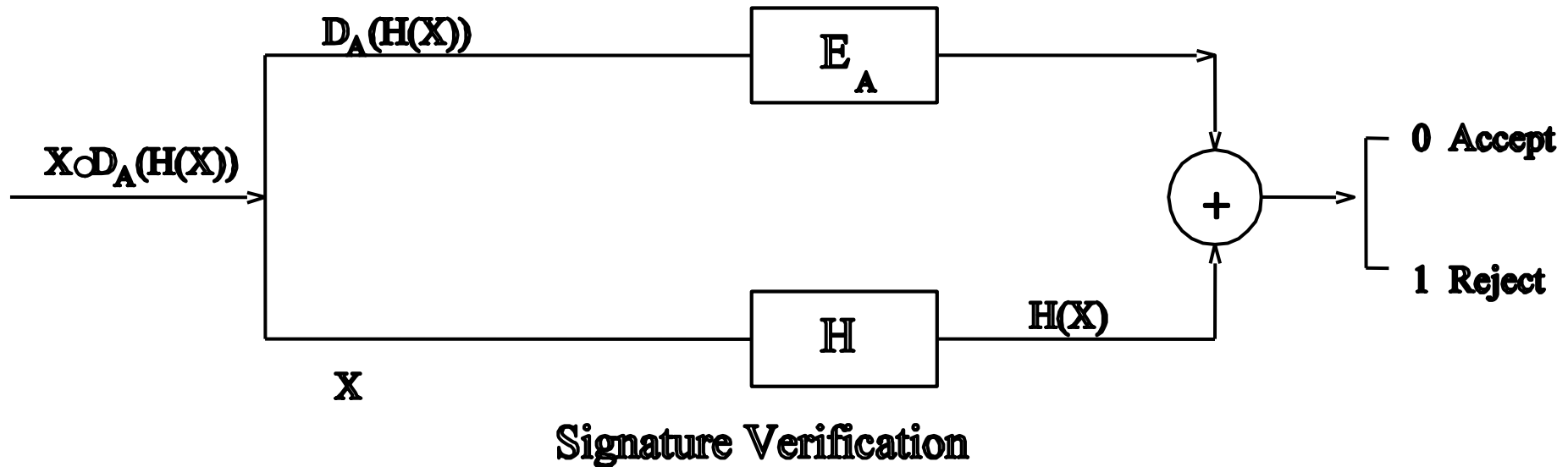


Signature Generation

Hash Function ใน Digital Signature

2. Verification

- ▶ ในการ verify หลักการคือ นำ Message ที่ได้รับมา ไปหาค่า HASH ก่อน จึงนำมาเปรียบเทียบ (เช็ค/ตรวจสอบ) กับค่า hash ที่ส่งมา



Hash Function ใน Digital Signature

3. Security of Hash function: In the context of signatures

▶ ต้องการทุกคุณสมบัติ 1-6

Cryptographic hash properties **1, 2 and 3** are required for **efficient signature generation**.

▶ Cryptographic hash properties **4,5 and 6** are required to **stop attackers forging** signatures. (ต้องการคุณสมบัติสามอันหลังนี้เพื่อป้องกันการปลอมลายเซ็น)

▶ Example

- ▶ Consider that Oscar, a malicious person, can generate $X \neq X'$ with $H(X) = H(X')$ such that Alice is happy to sign X and produce $S_A(X)$.
- ▶ Now Oscar can use $(X', S_A(X))$ as a signed message



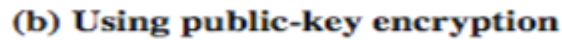
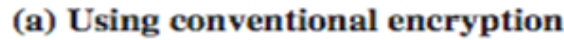
-
- ▶ For property 4, 5 and 6 to hold, the size of the output space (message digest/hash value) must be **large**, currently the **standard is 2^{160}** . That is, the hash value must be at least 160 bits.
 - ▶ A **strong signature scheme** and a **secure hash function may produce a weak signature scheme**. An example of this was demonstrated and shown by Coppersmith in combining RSA and the CCITT X.509 hash function
-

2. Three ways to authenticate message using hash functions

สร้าง message authentication code

- ▶ 1. Using symmetric encryption
 - ▶ If the sender and receiver share the encryption key, then authenticity is assured
- ▶ 2. Using public key encryption
 - ▶ Two advantages
 - ▶ provides a **digital signature** as well as message authentication
 - ▶ does not require the **distribution of keys** to communicating parties.
(explained in later lecture)
- ▶ These two approaches have an advantage over approaches that encrypt the entire message (เช่น ใน CBCMAC) เพราะใช้ computation resource น้อยกว่า..
- ▶ 3. Using secret value

วิธีที่ 1 และ 2 ใช้การเข้ารหัส



2. Three ways to authenticate message using hash functions

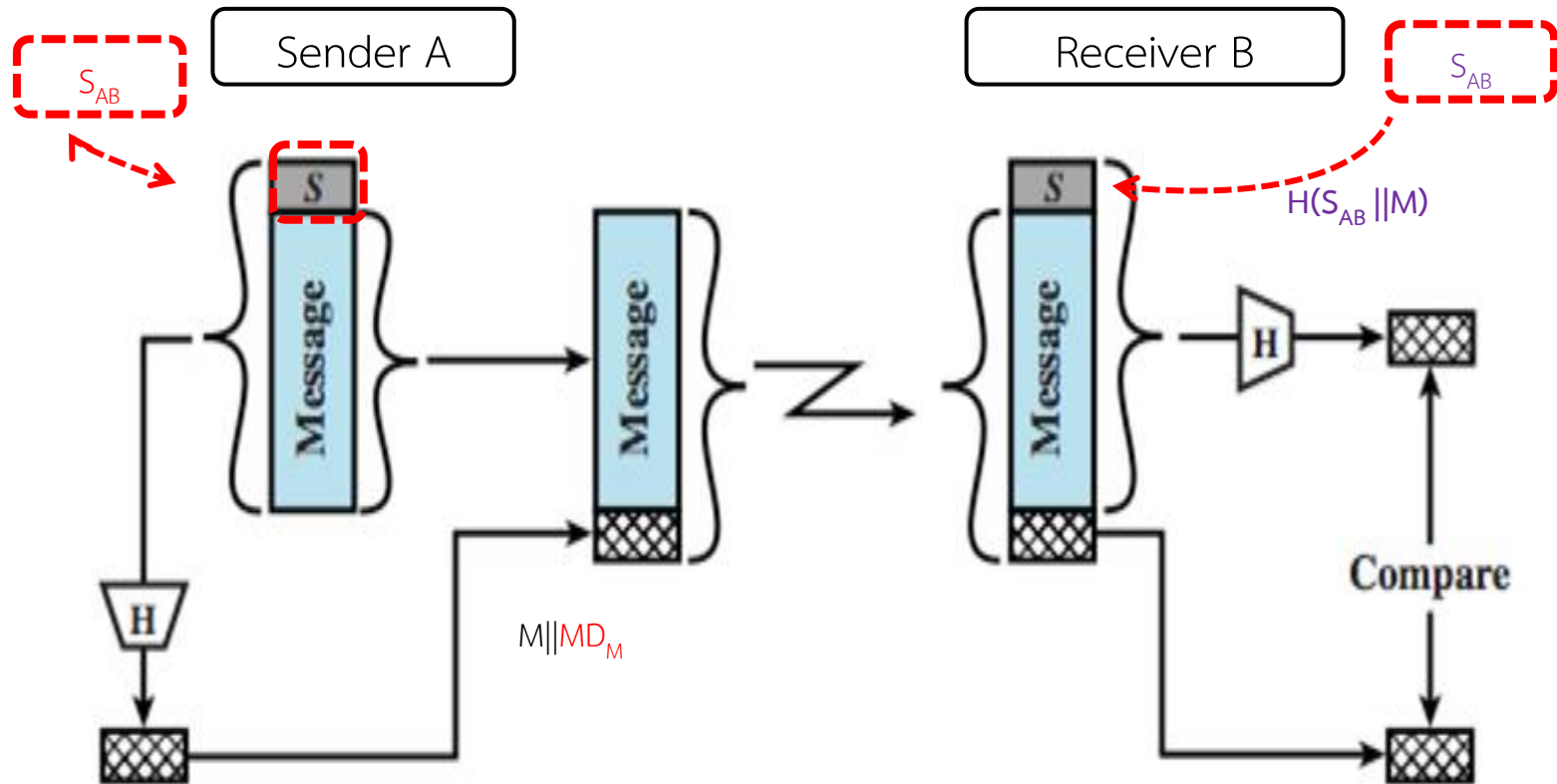
วิธีที่ 3: ใช้ secret value

- ▶ ใช้ hash function แต่ไม่ต้องใช้ encryption ในการทำ message authentication.
- ▶ ให้ A และ B เป็นผู้ที่รับส่งข้อมูลกัน (two communicating parties), ซึ่งในกรณีนี้ A และ B, จะต้องมีการแชร์ common secret value S_{AB} . ไว้ก่อน
 - ▶ เมื่อ A มี message ที่ต้องการส่งไปหา B,
 - ▶ A นำ secret value มาต่อกับ แเมสเสจ และ ใช้ hash function หาค่า hash ของ :

$$MD_M = H(S_{AB} || M).$$
 - ▶ A ส่ง $[M || MD_M]$ ไปให้ B.
 - ▶ เพราะว่า B รู้ S_{AB} ,
 - ▶ B นำ secret value ที่ตัวเองมีมาต่อกับแเมสเสจที่ได้รับ
 - ▶ ทำการคำนวณ $H(S_{AB} || M)$ และ ตรวจสอบว่าเท่ากับ MD_M . หรือไม่
- ▶ เนื่องจาก secret value ไม่ได้ถูกส่งไปกับ message attacker แม้ว่าจะดักฟัง message ได้ไป แต่ไม่สามารถ modify intercepted message ได้
- ▶ ดังนั้น ตราบใดที่ secret value รู้กันเฉพาะ A และ B attacker จึงไม่สามารถสร้าง แเมสเสจปลอมได้

2. Three ways to authenticate message using hash functions

วิธีที่ 3: ใช้ secret value (ภาพ)



(c) Using secret value

$$MD_M = H(S_{AB} || M).$$

MD_M

Sometimes is called MAC Based on a keyed Hash Functions

Most widely used MAC on the Internet.

HMAC (1996)

Building a **MAC** out of a hash function:

Standardized method: HMAC

$$\mathbf{S(k, m) = H(k \oplus \text{opad} || H(k \oplus \text{ipad} || m))}$$

H คือ Hash Function เช่น SHA-1

ipad is a pad value of 36 hex repeated to fill block **ipad= 0x3636...3636**

opad is a pad value of 5C hex repeated to fill block **opad=0x5c5c5c...5c5c**

M is the message input to HMAC (including any padding)

ตัวอย่างที่ใช้ HMAC-SHA1 HMAC-MD5

ใช้ใน Applications: IPSec and TLS protocols

RFC 2104. FIPS PUB 198 generalizes. and standardizes the use of HMACs. HMAC-SHA-1 and HMAC-MD5.

Hash Function: Security assessment

Birthday Attack

- ▶ This terminology arises from the Birthday paradox.
- ▶ Question:
 - ▶ What is the **smallest size class** such that **the chance of two students having the same birthday is at least $\frac{1}{2}$** ?
 - ▶ ถ้าค่าความน่าจะเป็น 0.5 จะต้องมึนักศึกษาทั้งหมดกี่คนในห้อง ถึงจะทำให้โอกาสที่นักศึกษาในห้องสองคนที่ถูกสุ่มเลือกขึ้นมา เกิดวันเดียวกัน
 - ▶ *In a group of how many people (select randomly) that at least two will share a birthday with probability at least $\frac{1}{2}$?*

Birthday Attack Cont. I

- ▶ คำตอบคือ เพียง 23 คนเท่านั้น (น้อยกว่าที่คาดคิด => paradox)
- ▶ Birthday Paradox สามารถนำมาประยุกต์ใช้การหาโอกาสที่จะเกิด Collision ใน Hash Function
- ▶ สมมติให้ จำนวน hash value (message digests) ที่เป็นไปได้มีทั้งหมด m ค่า
- ▶ Suppose there are m possible hash values. ถ้าเราสุ่มเลือก message มา k ค่า ทำการหาค่า hash value ของ message จำนวน k ค่านั้น ความน่าจะเป็น (probability) ที่อย่างน้อยจะเกิดการชนกันของ message หนึ่งคู่คือ

$$P(m, k) > 1 - e^{\frac{-k(k-1)}{2m}} = \varepsilon$$

Birthday Attack Cont. II

$$k \approx \sqrt{2m \ln \left(\frac{1}{1-\varepsilon} \right)}$$

$$P(m, k) > 1 - e^{\frac{-k(k-1)}{2m}} = \varepsilon$$

- ▶ Calculate this for $\varepsilon=0.5$, $m=365$... $k=22.49$...

$$k \gg 1.17\sqrt{m}$$

- ▶ The formula suggests that with **sqrt(m) evaluations of the hash function** there is a good chance (about 50%) of a collision being found.

Birthday Attack Cont. III

- ▶ จะสังเกตได้ว่า โอกาสที่จะประสบความสำเร็จใน attack นี้ ขึ้นอยู่กับปัจจัยสองอย่างเท่านั้นคือ
 - ▶ จำนวน message digest (m) ที่เป็นไปได้ทั้งหมด
 - ▶ เช่น MD5 มีขนาด 128 บิต ดังนั้น มี message digest ได้ทั้งหมด 2^{128} ค่า
 - ▶ จำนวน message ที่เรานำมาหาค่า hash (k)
 - ▶ มันจะไม่ขึ้นอยู่กับ โครงสร้างของ hash function นั่นคือ size ของ digest space ถูกใช้เป็น lower bound ของความน่าจะเป็นในการประสบความสำเร็จ (probability of success.)
-



Birthday Attack Cont. IV

- ▶ We can use this **lower bound** to find the required **size of digest space**, that is the number of bits of the digest, if we assume certain values for the level of feasible computation.
- ▶ เราสามารถใช้ lower bound ในการหาว่าจำนวนบิตของ message digest ควรมีจำนวนอย่างน้อยเท่าไร จึงจะปลอดภัยจาก attacker ที่มี computational power ที่เราคาดเดาค่าไว้ได้



Security Assess Hash Function กรณี Collision Resistant

- ▶ จาก $k \gg 1.17\sqrt{m}$ ที่ความน่าจะเป็น 0.5
- ▶ เมื่อ k = จำนวน Message ที่ผู้โจมตีสามารถนำมาหาค่า Digest ได้
- ▶ m คือจำนวน message digest ทั้งหมด
- ▶ ดังนั้น การที่เมื่อผู้โจมตี สุ่ม message มา สองค่า โอกาสที่ message สองค่า นั้นจะให้ค่า Hash เดียวกัน (ชนกัน) ด้วยความน่าจะเป็นที่ 0.5 จะเกิดขึ้นเมื่อ k มีค่าอย่างน้อย $= 1.17 * m^{1/2}$
- ▶ ถ้า m มีค่ามาก สามารถละ 1.17 ได้ ดังนั้น $k \gg \sqrt{m}$
- ▶ ดังนั้น ถ้าเราทราบว่า message digest มีได้ทั้งหมด m ตัว เพื่อไม่ให้เกิด Collision k ต้องมีค่า น้อยกว่า $m^{1/2}$



การเปรียบเทียบ Birthday paradox กับ Hash property

- ▶ ดังนั้น Security ของ Cryptographic Hash function ในกรณี ที่ความน่าจะเป็นที่จะเกิดของแต่ละกรณี คือ 0.5
- ▶ สำหรับ ในกรณี Pre-image และ Second Pre-image นั้น
size of digest = size ของ k

ปัญหา	Size of Digest	Lower bound of the security
Pre-image Resistant	N	N
Second-Preimage Resistant	N	N
Collusion Attack	N	$(N)^{1/2}$

การทราบ lower bound security ของ Cryptographic Hash function ให้ ประโยชน์ดังต่อไปนี้

- ▶ ตัวอย่าง A cryptographic hash function uses a digest of 64 bits. How many digests does Eve need to create to find two messages with the same digest with the probability 0.5?
- ▶ โจทย์ของตัวอย่างนี้ หมายความว่า **output** ของ **Cryptographic Hash function** นี้ $|H(M)| = 64 \text{ bit}$ ถ้า **Eve** อยาก เบรก **Hash** นี้ จะต้องสร้าง **Message Digest** อย่างน้อยทั้งหมด ก็ **messages** จึงจะเกิดการชนกัน

▶ คำตอบ

Collusion Attack	N	$(N)^{1/2}$
------------------	---	-------------

- ▶ $2^{64/2} = 2^{32}$
- ▶ คำถามที่ 2 ถ้า ใน 1 วินาที Eve สามารถ หาค่า Message Digest ได้ทั้งหมด 2^{20} ค่า Eve จะใช้เวลานานเท่าไรในการ Break Cryptographic Hash function นี้

Additional Slide

- ▶ Because of the birthday attack,
 - ▶ the length of hash outputs in general should double the key length of block ciphers
- ▶ SHA-224 matches the 112-bit strength of triple-DES
- ▶ SHA-256, SHA-384, SHA-512 match the new key lengths (128,192,256) in AES



