Topic 5: Cryptographic Checksums

Understand the need for cryptographic checksums, how to construct them, and their applications

Source: Main textbook: 2.2, 21.1-21.3

Overview

- Motivation and Definitions
- Cryptographic Hash Functions
- Block Cipher based MAC (Message Authentication Code)
- HMAC (hash function based MAC)
- Authenticated Encryption (designed for very high speed networks, e.g. 5G)
- Conclusion

Threats to Message Security (vs Security Properties)

- Content disclosure (confidentiality)
- Traffic flow analysis (traffic flow confidentiality)
- Sender masquerading/impersonation (entity authentication)
- Content modification (message authentication)
- Sequence modification (message authentication)
- Timing modification (message authentication)
- Repudiation of transmission/origin (digital signature)
- Repudiation of receipt (digital signature + trusted third party service)

These terms are used interchangeably:
Checksum=digest=fingerprint=compressed value=hash value
MAC (Message Authentication Code) = cryptographic checksum

Need for a Cryptographic Checksum

Why do I emphasise '*a certain degree*' here?

- Conventional (symmetric) encryption, A --> B: $E_K[M]$
  - Provides confidentiality, as ONLY A and B share K.
  - Provides ***a certain degree*** of origin authentication, as it could only come from A or B.
  - Does not provide non-repudiation, as
    - A could deny sending the message – repudiation of origin.
    - B could also generate the encryption.

- Another variant, A --> B: $E_K[M||h(M)]$
  - Assuming that h is a compression function.
  - What is the difference between this protocol and the one above?
  - What benefit does this protocol provide vs the one above?

Need for a Cryptographic Checksum

- Public-key encryption, *A* --> *B*: $E_{KUb}[M]$ (using *B*'s public key)
  - Provide confidentiality as only *B* has the decryption key.
  - Does not provide authentication (origin authentication and content integrity), as anybody could get hold of *B*'s public key.

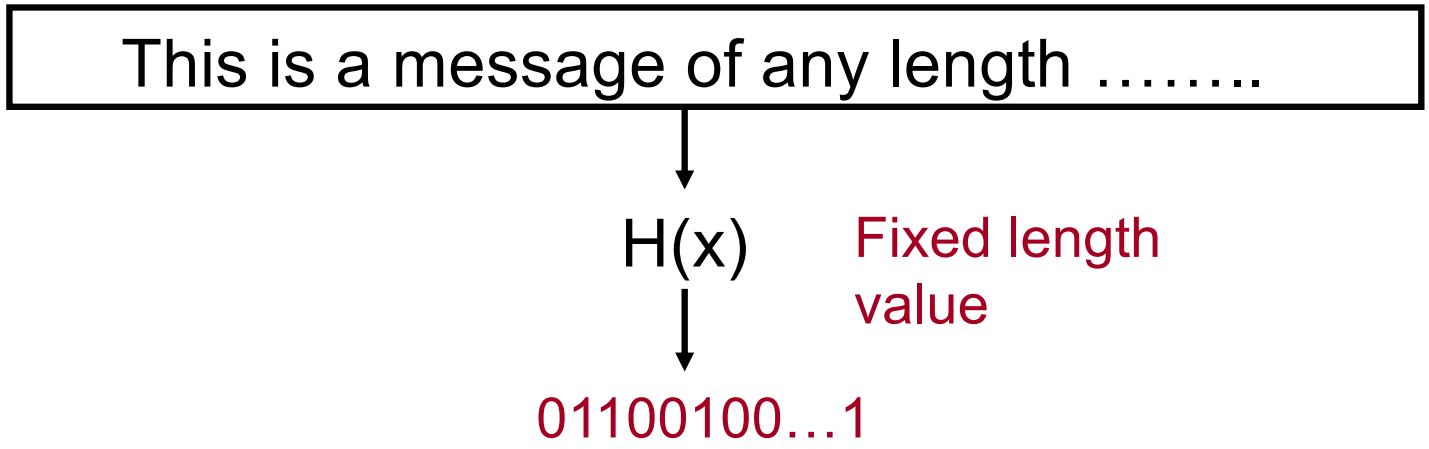
- Digital signing, *A* --> *B*: $M||E_{KRa}[h(M)]$ (using *A*'s private key)
  - Provides non-repudiation (origin authentication and content integrity), as
    - Only *A* has KR_a, so signed item (called checksum) must have come from *A*.
    - Any party can use KU_a to verify the signed item.
    - Provided KU_a is trust-worthy, and the signature is dated.
  - Does not provide confidentiality.

Need for a Cryptographic Checksum

- Cryptographic checksum can be used to achieve
  - Content authentication (= origin + integrity) for any kind of messages including unstructured ones.
  - Non-repudiation of origin is provided if the checksum
    - is uniquely associated to the message originator or
    - certified by a trusted third party.
  - Anti-replay (i.e. achieve freshness)
- It is attractive to applications that only require authentication, e.g.
  - secure broadcast;
  - secure software, or source code, distribution.
- A cryptographic checksum should be
  - hard to forge and tamper-proof.

Hash Functions – Compression Property

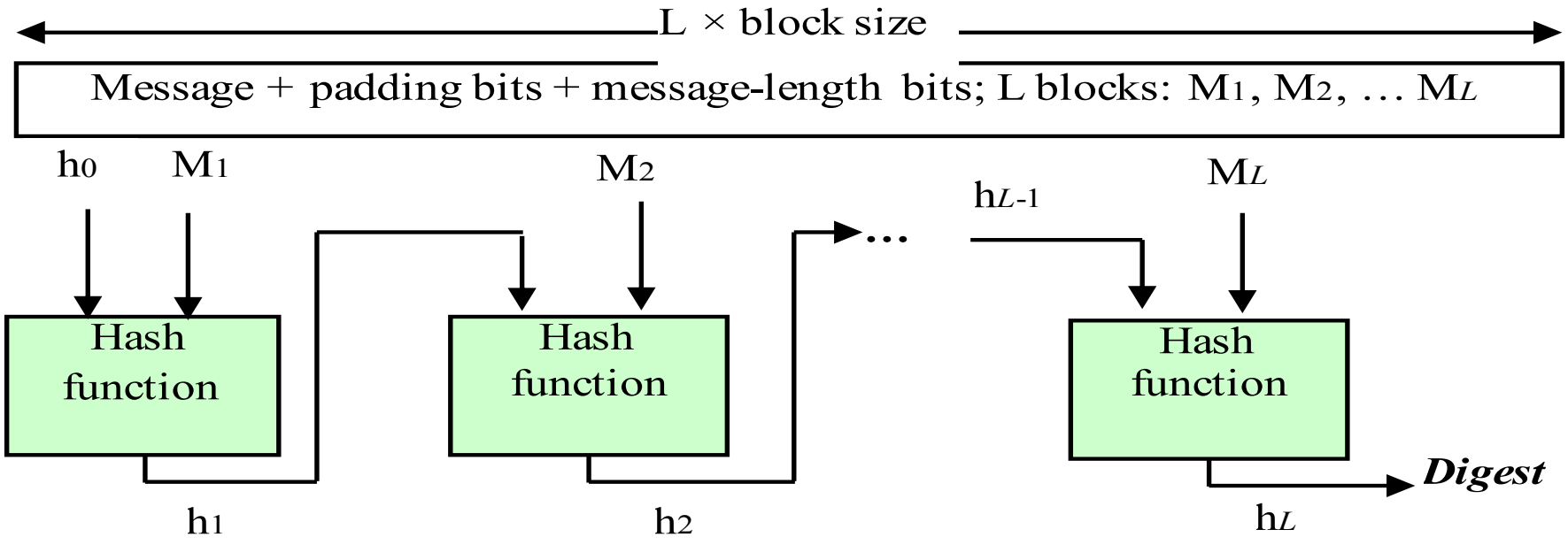
- Given a message *M* of arbitrary length, a hash function, *H,* produces a fixed-sized output, *h* (called a message digest, checksum, hash value, or fingerprint, of *M*), i.e. $h=H(M)$.
- *H* should be a function of all the bits of M.



This is a many-to-one mapping, so collisions are unavoidable, but we should make finding collisions as difficult as possible.

Hash Functions – Compression Property

- Each hash function execution processes a block of M; the output is the input for the next iteration; the output of the last block is the digest for M.



h0 is a constant initial value, and the output of the last block *is the digest* of the entire message.

Hash Functions - Security Requirements/Properties

Cryptographic hash functions are hash functions with these security requirements/properties:

- Preimage resistant (one-way):

  Given $y \in Y$, it is computationally infeasible to find a value $x \in X$ s.t. $H(x) = y$.

- 2nd preimage resistant (weak collision-resistant):

  Given $x \in X$, it is computationally infeasible to find another value $x' \in X$, s.t. $x' \neq x$ and $H(x') = H(x)$.

- Collision resistant (strong collision-resistant):

  It is computationally infeasible to find two distinct values, x' and $x \in X$, s.t. $H(x') = H(x)$.

PS: If H is strong collision-resistant, then it is also weak collision-resistant.

Hash Functions – Importance of the Security Properties

- **Signature forgery if weak collision resistance property is not met.**
  - Assuming that *A* has sent a signed message *M* to *B*, *i.e.* $M||s$ where $s=E_{KRa}[H(M)]$ and *KRa* is A's private key;
    - An attacker intercepts *A's* signature and message;
    - The attacker finds another message *M'* with $H(M)=H(M')$;
    - The attacker now has your signature *s* on the message *M'*.
  - Think about the implication of this attack in real-life!

Hash Functions – Importance of the Security Properties

- **Repudiation if strong collision resistance property is not met.**
  - Assuming that A is to send a signed message M to B
    - A chooses two messages M and M' with $H(M)=H(M')$;
    - A signs M by generating signature $s=E_{KRa}[H(M)]$;
    - A sends B $\boldsymbol{M||s}$;
    - Later A repudiates this signature, saying it was really a signature on the message M'.
  - Think about the implication of this attack, if
    - The communication is for A to make an e-payment; and
    - M is an electronic cheque for £10.
    - M' is an electronic cheque for £1000.

Hash Functions – Commonly used Hash Functions

	SHA-1	SHA-256	SHA-384	SHA-512	SHA3-Family
Hash value size	160	256	384	512	SHA3-224/ SHA3-256/ SHA3-384/ SHA3-512
Block size	512	512	1024	1024	1152/1088/832/ 576
Word size	32	32	64	64	64
Security	80	128	192	256	112/128/192/256

* All sizes are measured in bits; *SHA = Secure Hash Algorithm

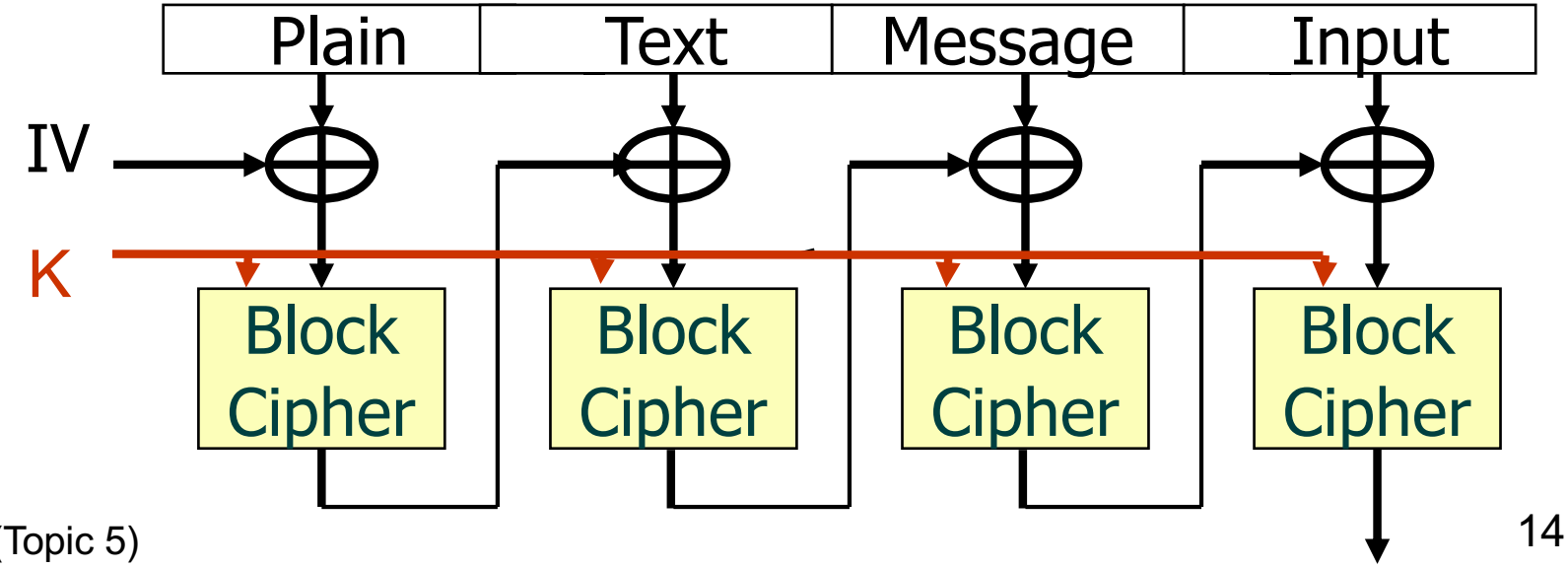
* Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a work-factor of approx $2^{n/2}$

Hash Function Applications

- Secure storage of passwords
- Digital signatures
- Pseudo-random number generations
- Bit commitment (or coin flipping) problem
- Bit coins
- Blockchains
- Digital payment systems
- Digital right management systems
- …

Message Authentication Code (MAC) using Block Ciphers

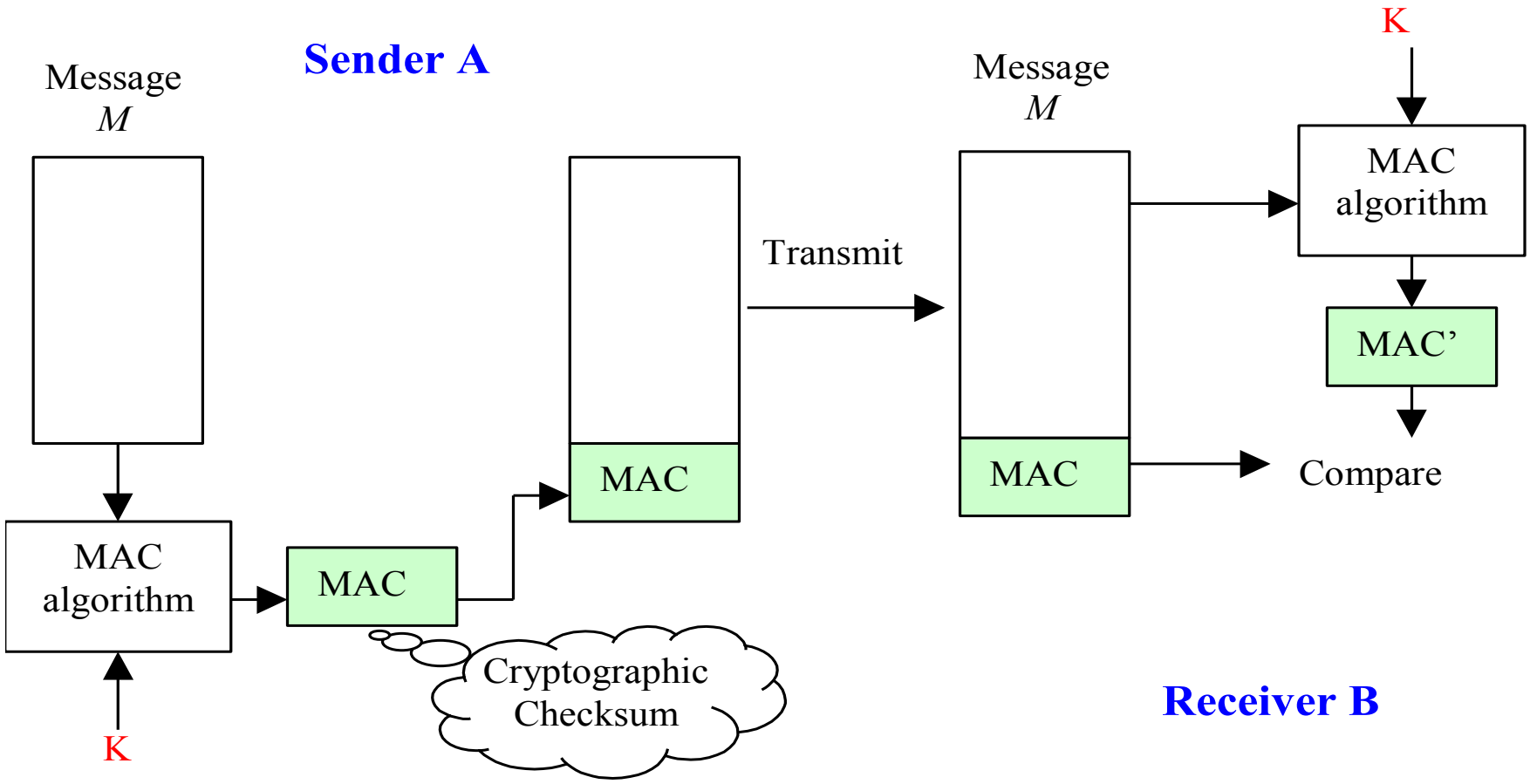
- A cryptographic checksum can also be generated using a symmetric block cipher, i.e. **MAC=$f_K(M)$,** where f_K is a block cipher based digest function.
- An example is CBC-MAC:
  - can be any symmetric block cipher
  - the output of the last block is MAC
- The block cipher based digest function has an embedded key.



MAC in Operation

- **Sender**
  - uses K and a MACing function, f, to generate a checksum, $MAC=f_K(M)$.
  - then sends $M||MAC$, where || is concatenation of data items.
- **Receiver**
  - computes $MAC' = f_{K'}(M')$, where M' is the message received, and K' is receiver's copy of the key.
  - If $MAC=MAC'$, then the message has not been tampered with.

MAC in Operation



MAC in Operation

- If only *A* and *B* know the secret key *K*, and if MAC=MAC', then the receiver can be assured
  - the message has not been altered - integrity protection;
  - the message is from the alleged sender - origin authentication;
  - the message is of the proper sequence if the message includes a sequence number;
  - the message is fresh - *i.e,* not a replay
    - if the message includes a timestamp; or
    - a random number contributed (fully or partially) by B (the recipient).
- The MAC operation described above is applicable to any symmetric key based MAC methods, e.g. hash function based.

HMAC (Keyed Hash Function, or Hash Function based MAC

- KeyedHash = Hash(K||Message)
- HMAC is a keyed hash function, specified as Internet standard

H(K2||H(K1||M))

HMAC(K, M)= H[(K^+ XOR opad) || H[(K^+ XOR ipad) || M)]];

where

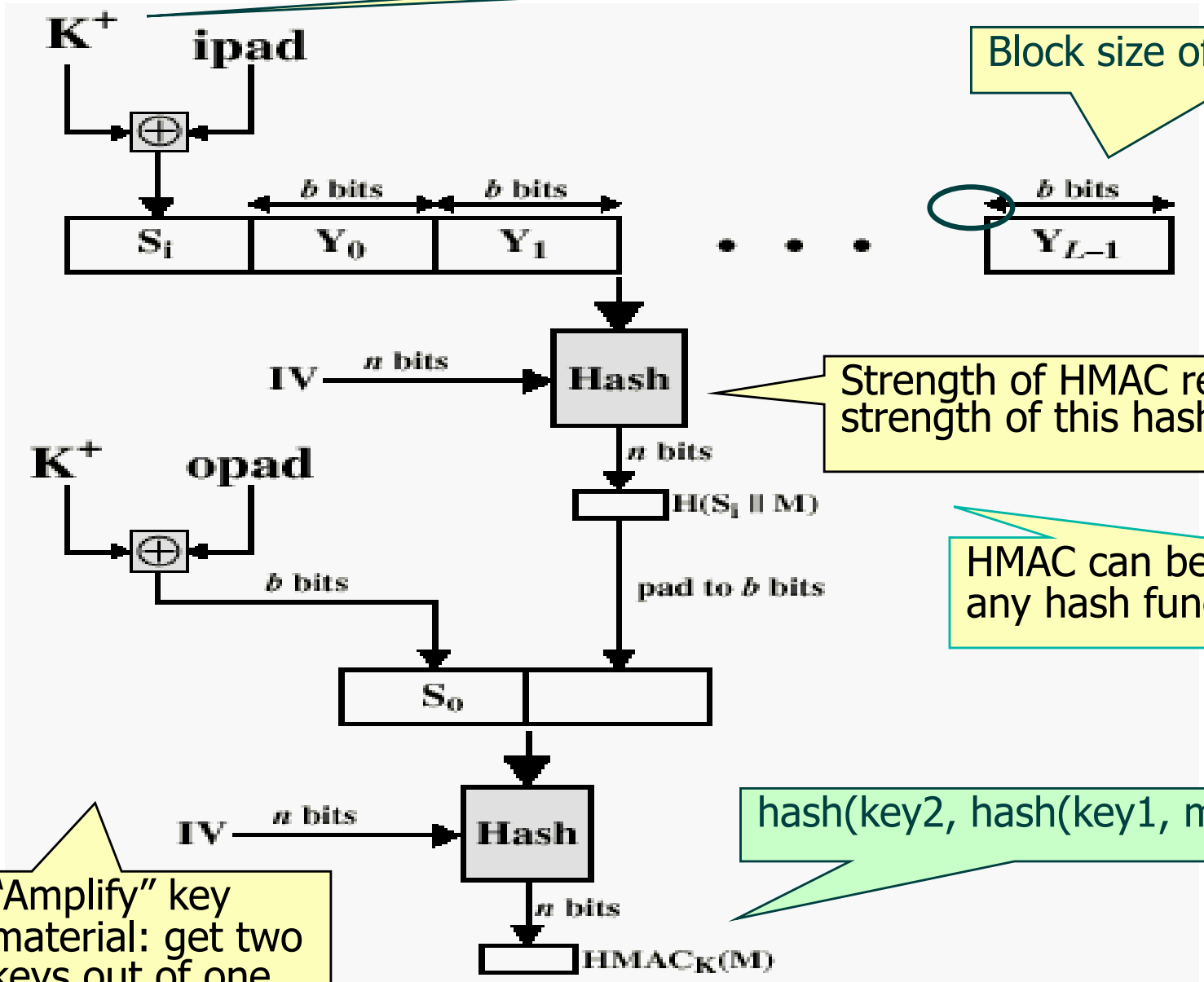
H = any hash function such as SHA-256;

K^+ is the key padded out to block size;

ipad = a string by repeating the byte 0x36 (00110110) as often as necessary;

opad = a string by repeating the byte 0x5c (01011100) as often as necessary.

HMAC


Secret key padded to block size
K+
ipad
b bits
b bits
b bits
Si
Y0
Y1
YL–1
IV
n bits
Hash
Block size of hash function, H.
Strength of HMAC relies on strength of this hash function, H.
K+
opad
n bits
H(Si || M)
b bits
pad to b bits
HMAC can be constructed with any hash function.
S0
IV
n bits
Hash
hash(key2, hash(key1, message))
"Amplify" key material: get two keys out of one.
n bits
HMACK(M)

Security of MACs

- Brute force attacks
  - Finding collisions cost $2^{n/2}$, where n is the bit-length of MAC value. The larger the n value, the more secure it is.
  - Attack key space or MAC by exploiting MACs with known message-MAC pairs.

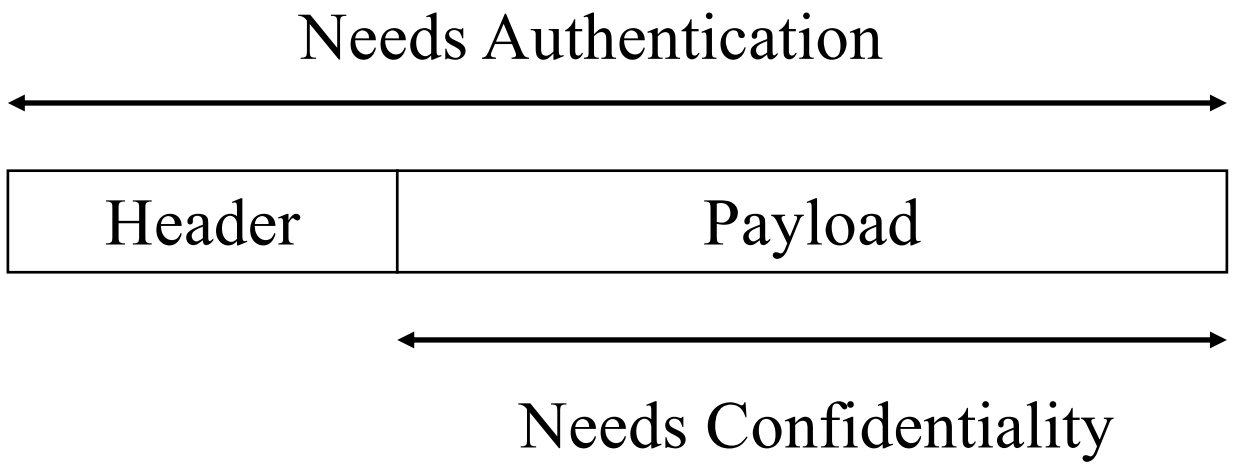
- Worth mentioning: a MAC is not a digital signature; it does not provide non-repudiation without any assistance of a third party.

Authenticated Encryption

- This is for achieving both message authentication and confidentiality.
- Possible approaches:
  - Hash-then-encrypt: E(K, (M||H(M))
  - MAC-then-encrypt (used in SSL): E(K2, (M||MAC(K1, M));
    or Tag= MAC(K1, M), E(K2, (M||Tag)); Tag is for authentication
  - Encrypt-then-MAC (used in IPSec):
    C=E(K2, M), Tag=MAC(K1, C)
  - Encrypt-and-MAC (used in SSH):
    C=E(K2, M), Tag=MAC(K1, M)

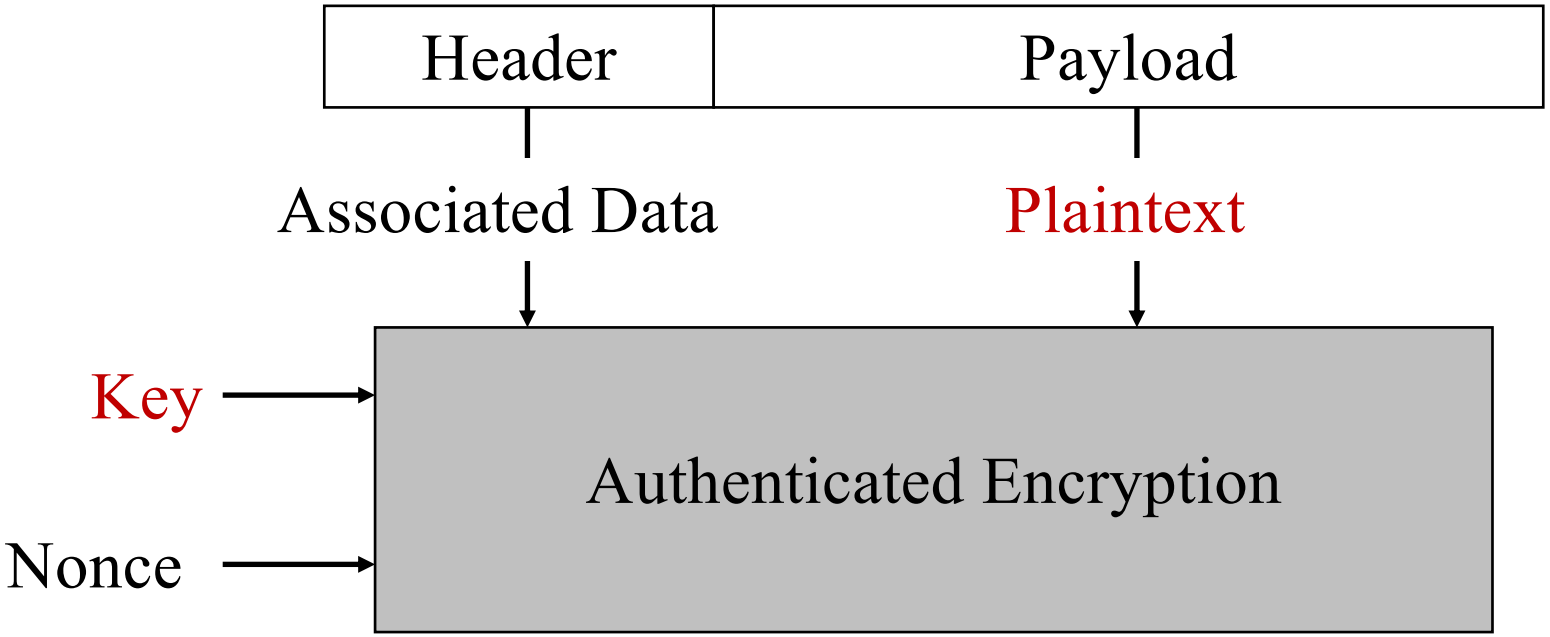
Authenticated Encryption with Associated Data (AEAD)

- In some contexts, such as networks, a message (packet) consists of two parts, a header field and a payload, and the two parts often have different security requirements.
- In addition, the protections should be provided efficiently (with as less overhead as possible).



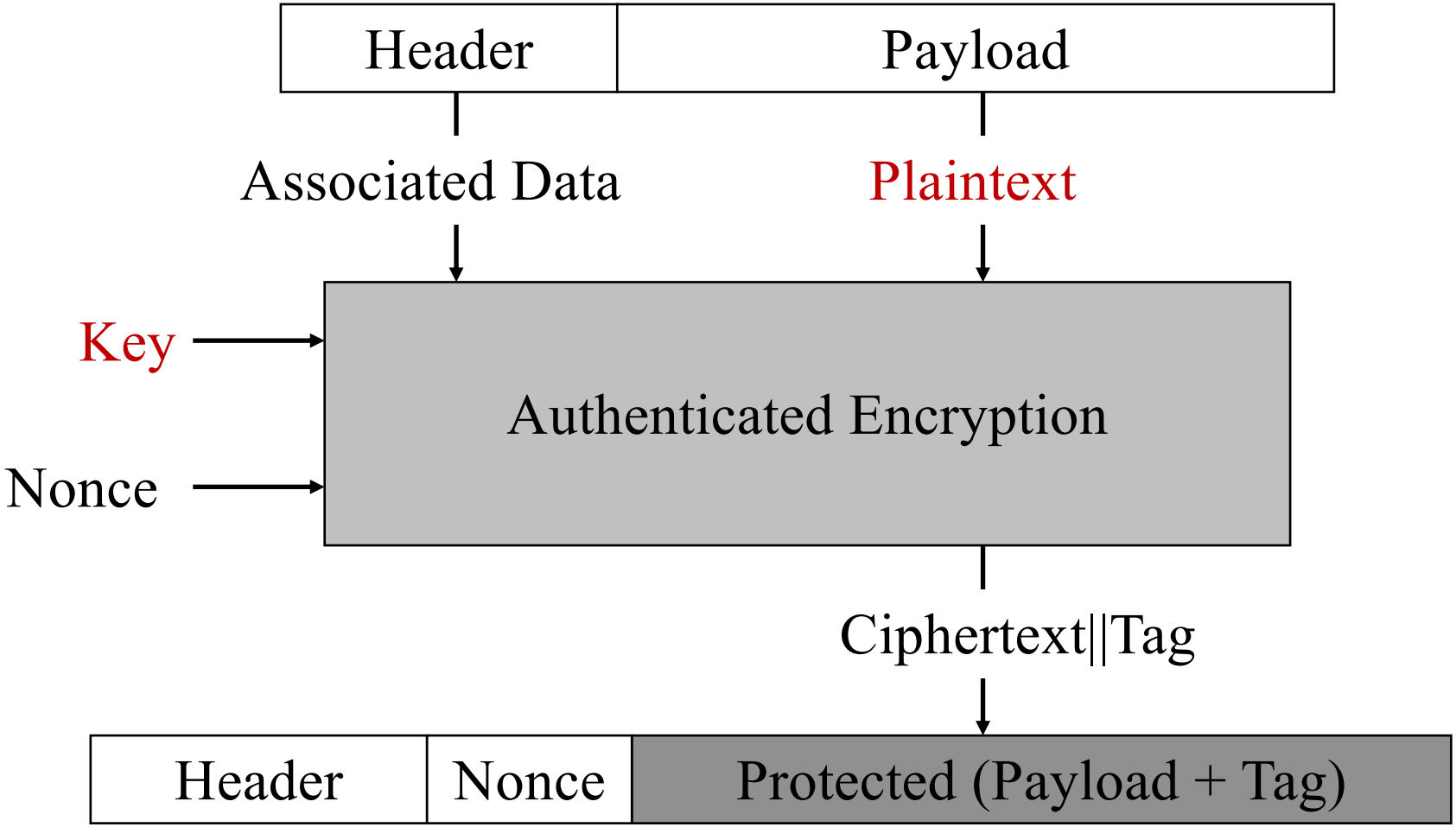
Authenticated Encryption with Associated Data

- Plaintext (Payload) should be encrypted AND authenticated.
- Associated Data (Header) should be authenticated (without encryption).
- Nonce should be distinct for each AEAD operation.



Authenticated Encryption with Associated Data

- Tag also goes through the encryption operation.

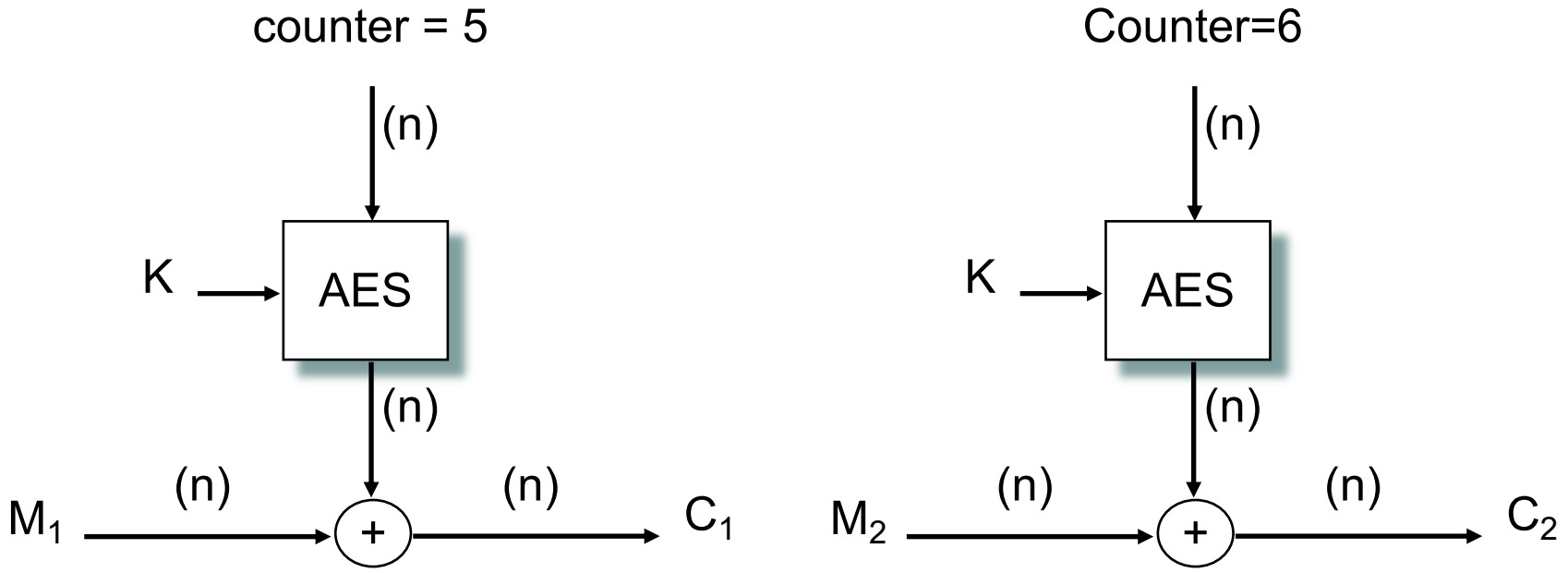


Authenticated Encryption with Associated Data

- Two notable schemes: CCM and GCM
  - CCM: Counter Mode with CBC-MAC (NIST SP 800-38C for WiFi)
  - GCM: Galois/Counter Mode (NIST standard SP 800-38D) (optional materials)

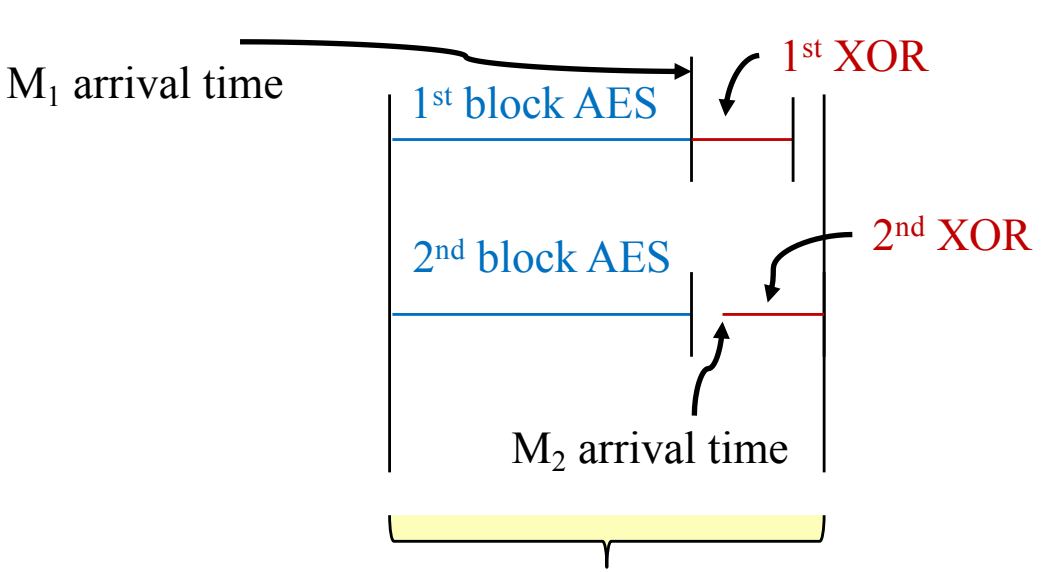
CCM: Counter Mode with CBC-MAC

- CTR encryption example: assuming two plaintext blocks, counter value is initialised to 5, AES is chosen as the block cipher.
  - Counter value increment for each subsequent plaintext block.
  - Parallel/pipelined computation/processing/operation.
  - Encryption/decryption are done using XOR – very fast.

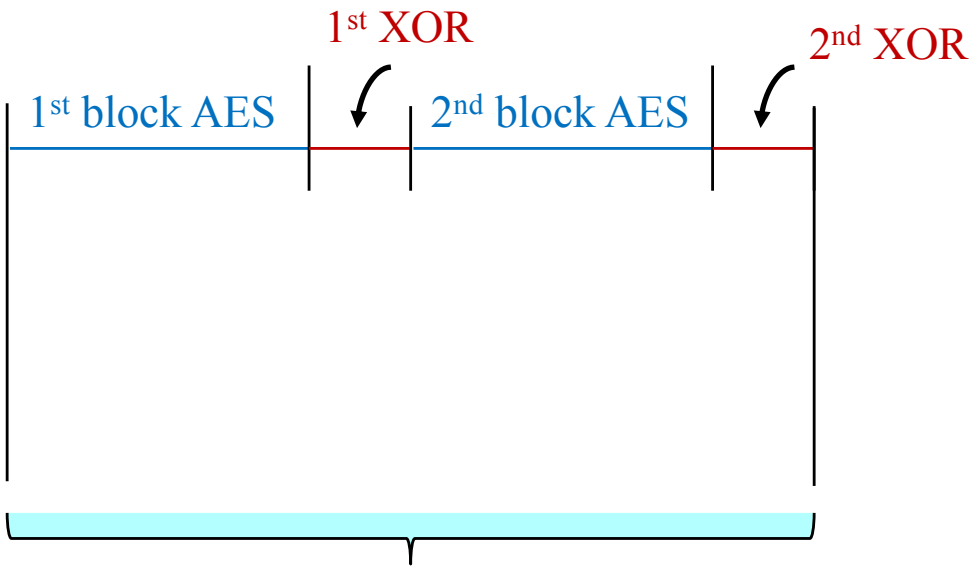


CCM: Parallel/pipelined computation

- Computation time: pipelined vs non-pipelined operations



Using CTR Mode: Total time taken to encrypt the two blocks



Using CBC Mode: Total time taken to encrypt the two blocks

CCM

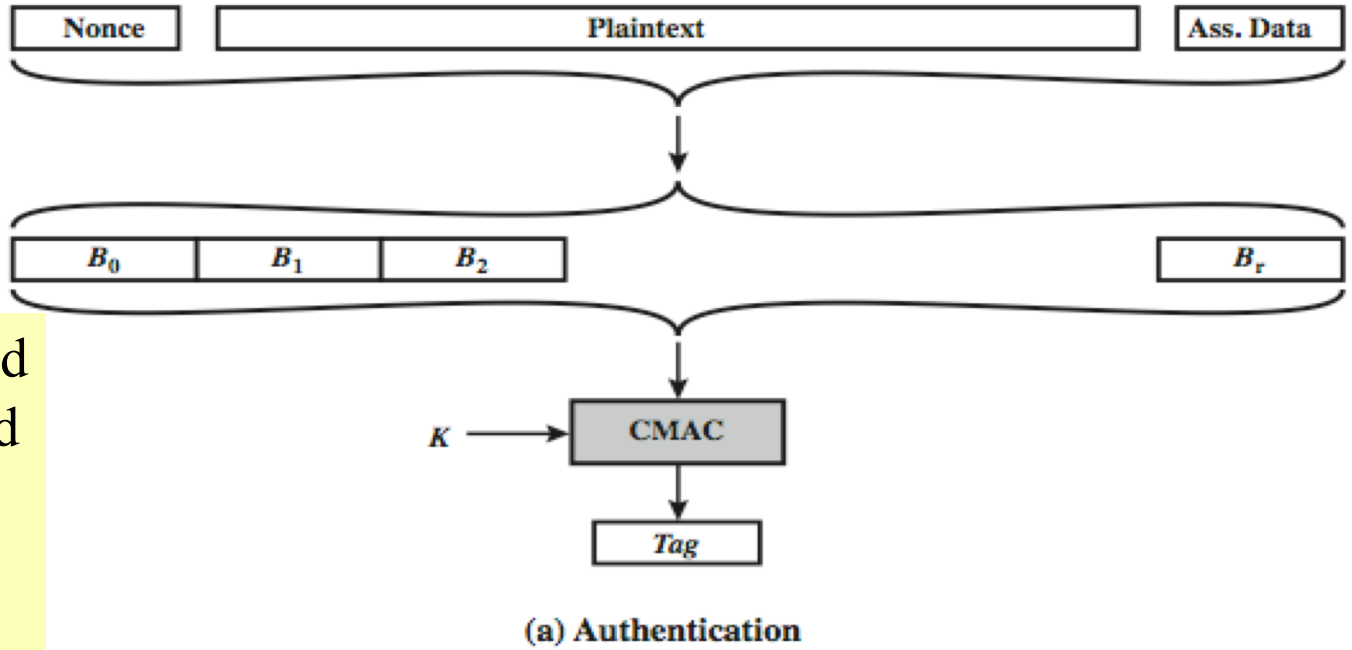
Authentication Tag is generated using CBC-MAC (also referred to as CMAC):

Tag=CMAC(K, M)

where Plaintext=Payload;
M=Nonce||Plaintext||Ass.Data;
Ass.Data=Associated Data (e.g. packet headers)

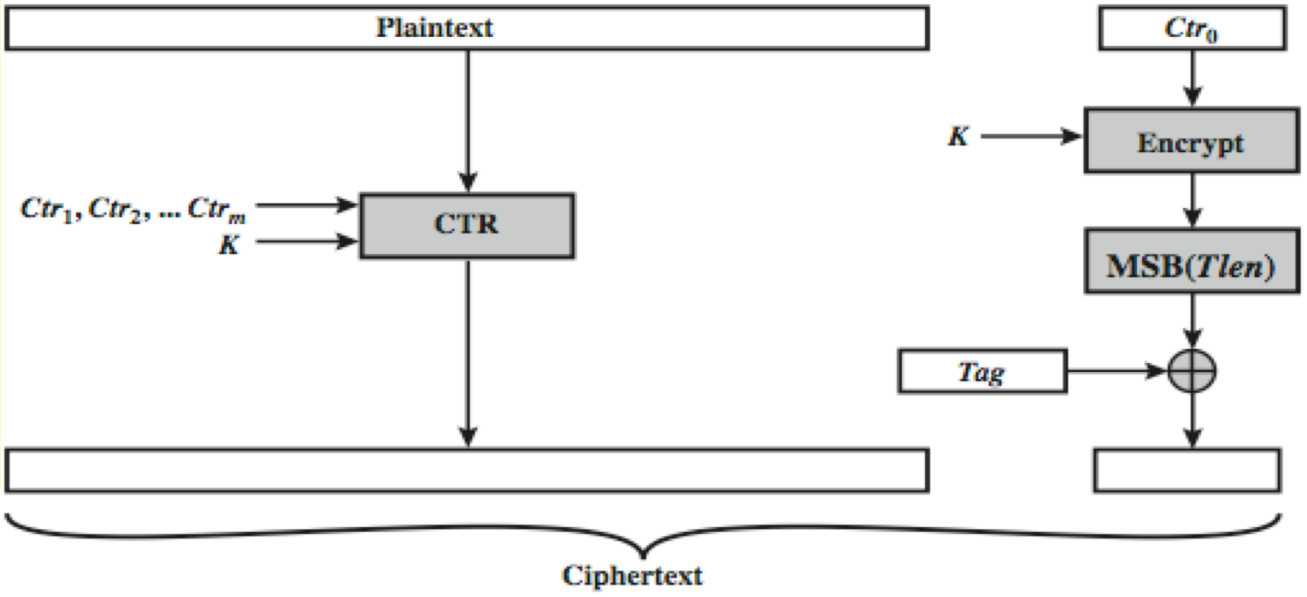
Encryption is by AES CTR mode: E(K, Payload||Tag)

Nonce (random number) to ensure Tag is fresh.



(a) Authentication

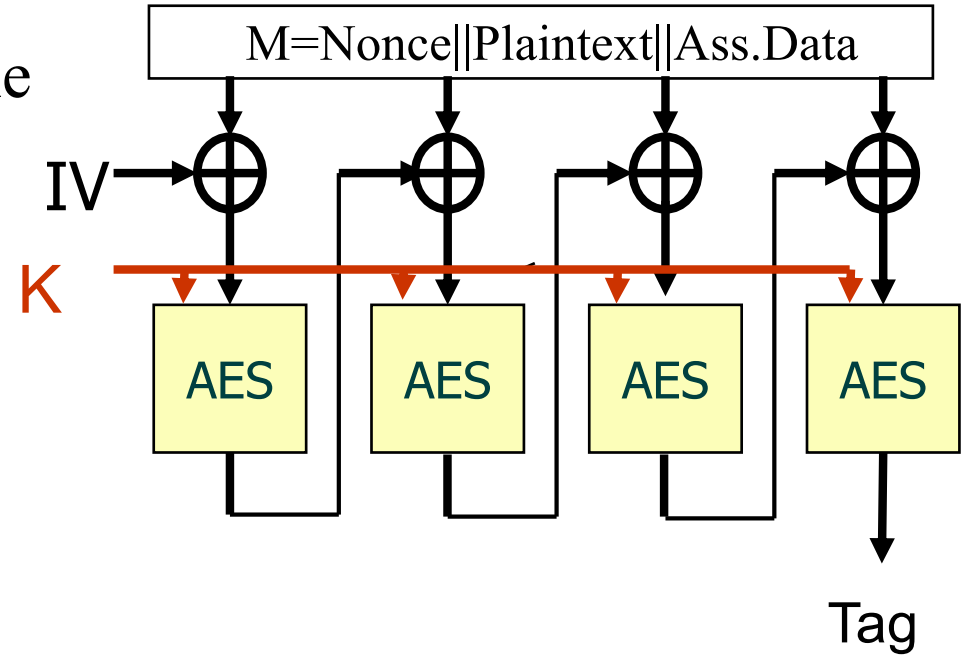
E(K, (M||MAC(K, M))



(b) Encryption

CCM: Counter Mode with CBC-MAC

- Tlen (Tag Length) most significant bits of the output are XORed with the tag to produce an encrypted tag.
- Same key (can be different) for MACing and encryption.
- Plaintext passes 2 block cipher operations (MAC and encryption).
- CBC-MAC is not pipelinable/parallelizable.



- To speed up the performance further, we have GCM.

GCM

- Galois/Counter Mode (GCM)
- GCM is designed to provide authenticated encryption for high speed applications (can do authenticated encryption at > 10 gigabits per second)
- Combines counter mode with Galois mode
- Counter mode can be pipelined and parallelized in hardware implementations
- Rather than using block chaining to generate tags (MACs), GCM uses a Galois field multiplication that is less computationally intensive than CBC-MAC
- Cost half of the number of AES operations used in CCM
- NIST standard SP 800-38D

GCM

- GCM has two functions, authenticated encryption and authenticated decryption.
- **Authenticated encryption**: Given a selected block cipher & K, authenticated encryption function has **3 inputs:**
  - Plaintext, denoted P.
  - Additional authenticated data (AAD), denoted A.
  - An initialization vector, denoted IV (*Nonce* to ensure freshness).
- It generates **two outputs**:
  - A ciphertext, denoted C, which has the same length as P.
  - An authentication tag, denoted as T, *used to protect authenticity of* P, A and IV.

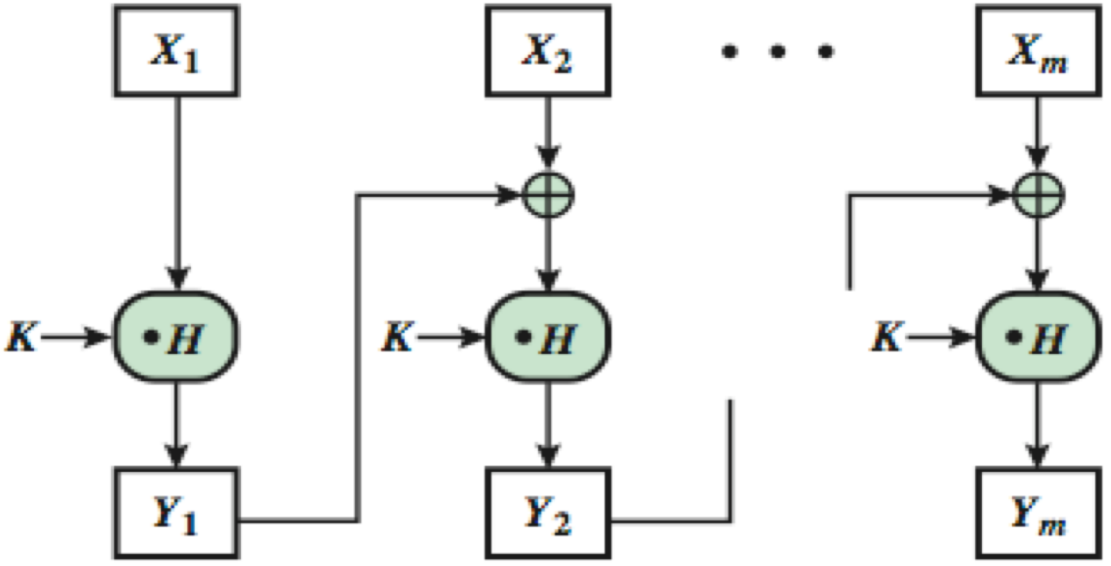
GCM

- **Authenticated decryption**: Given a selected block cipher & K, this function has **4 inputs**: IV, A, C, and T.
- The output is one of the following:
  - plaintext P that corresponds to the ciphertext C, or
  - a special error code, denoted $FAIL$.
- The output P indicates that T is the correct authentication tag for IV, A, and C; otherwise, the output is $FAIL$.

GCM

- Uses two functions:
  - GHASH - a keyed hash function over a binary Galois field.
  - GCTR - a variation of CTR mode (with a particular incrementing function, denoted *inc*).
- GHASH: plaintext xor'ed with feedback and multiplied with ***H*** (a hash subkey) in GF(2^{128}).

Y_0 is a 128-bit length "zero block", 0^{128}. For i=1 ,…, m, $Y_i=(Y_{i-1} \oplus X_i) \cdot H$, where block ***H* is the hash subkey**, generated by applying the block cipher to the "zero" block.
Y_m is the output.



(a) $\text{GHASH}_H(X_1 \| X_2 \| \ldots \| X_m) = Y_m$

GHASH

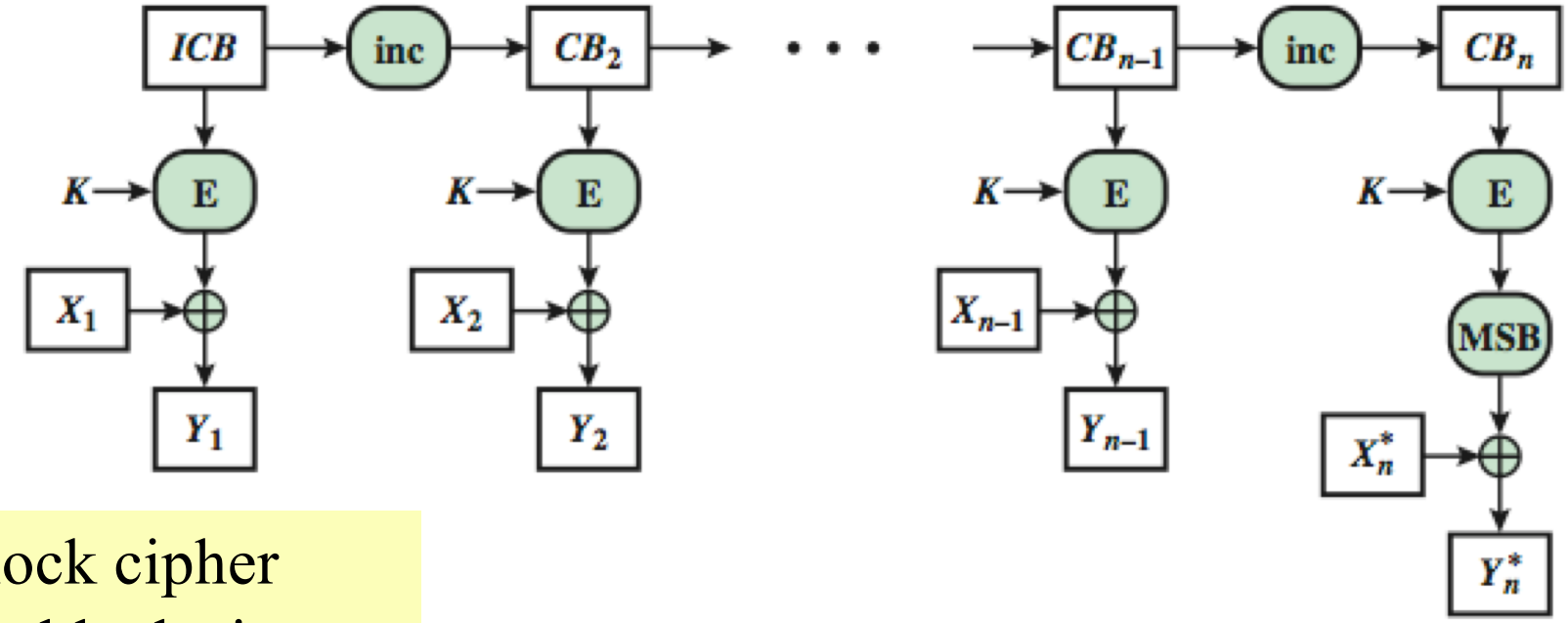
- GHASH function can be expressed as:

$$Y_m = (X_1 \bullet H^m) \text{ XOR } (X_2 \bullet H^{m-1}) \text{ XOR } \ldots \text{ XOR } (X_{m-1} \bullet H^2) \text{ XOR } (X_m \bullet H)$$

where $\bullet$ designates multiplication in $GF(2^{128})$

GCM

- Y_i in the two figures, (a) and (b), are not related.
- ICB=Initial Counter Block.
- MSB$_s(X)$: bit string consisting of s left-most bits of the bit string X.

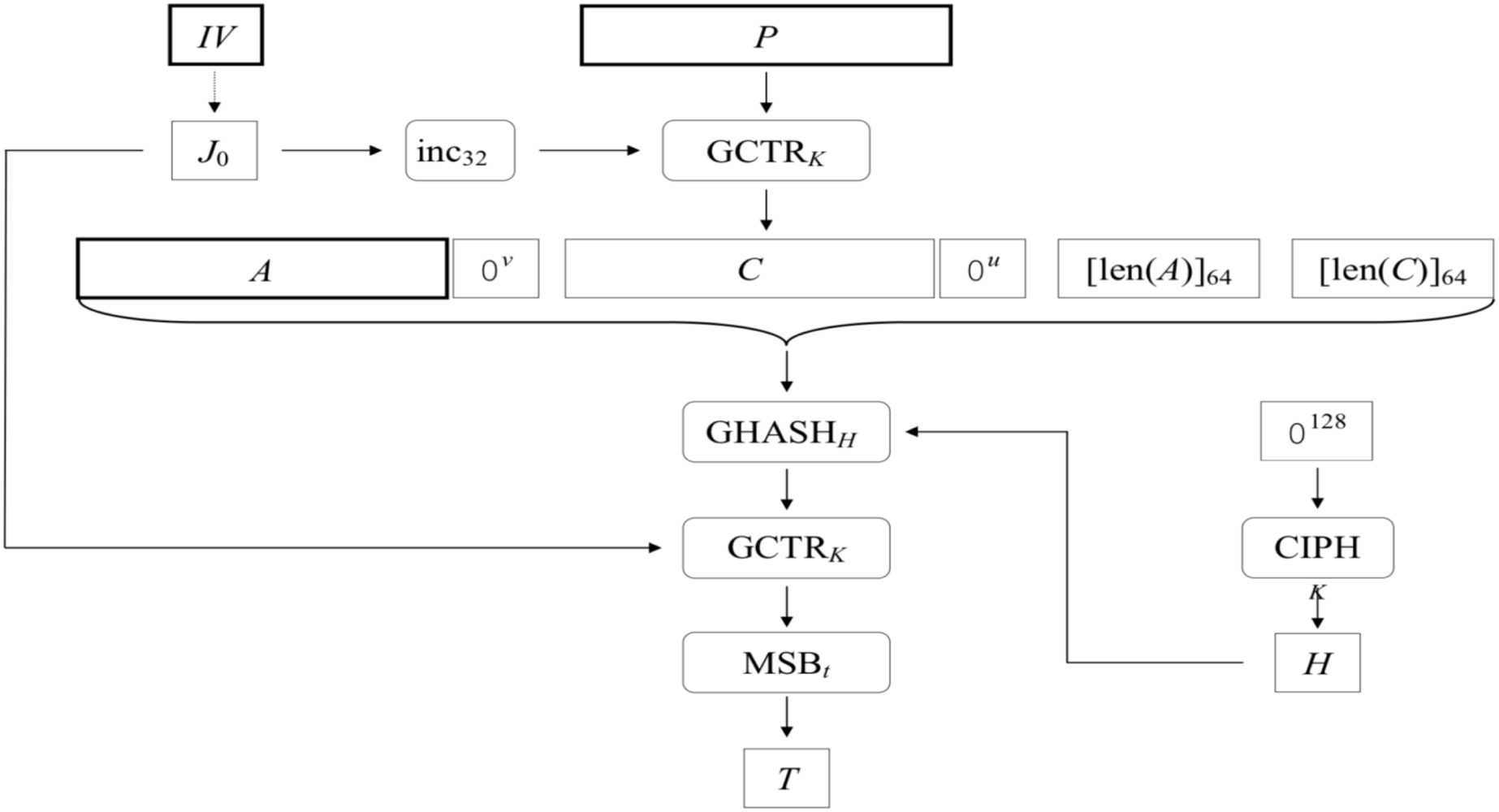


E=CIPH; a block cipher with a 128-bits block size

(b) $\text{GCTR}_K(ICB, X_1 \| X_2 \| \ldots \| X_n^*) = Y_1 || Y_2 || \ldots || Y_{n-1} || Y_n^*$

GCM

- J_0=ICB (Initial Counter Block)



GCM

- *GHASH* compresses an encoding of AAD (A) and the ciphertext (C) into a single block, which is then encrypted to produce the authentication tag, $T=Tag$.
- C can be null, resulting a variant of GCM, called GMAC, generating and verifying tag on non-confidential data.
- J_0 (ICB=Initial Counter Block) is generated from IV.
- Inc_{32}: an increment function; it increments the right-most 32 bits of the string with the remaining bits unchanged.
- *CIPH*: a block cipher with a 128-bits block size, and the key size should be at least 128 bits.

Exercise Question – E5.1

For a hash value to be used as a cryptographic checksum, it must be protected with a secret, as it is clear, from the above, that a hash function does not have an embedded key.

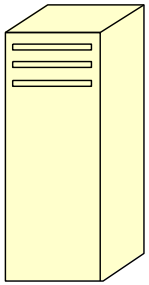
Assuming that a sender, s, is to send a message, M, to a receiver, r. Propose as many different methods as you can to protect the hash value of M to assure the authenticity of M. Comment on the suitability/applicability of each of the methods you propose.

Exercise Question – E5.2

For each of the following applications, please identify what property(ies) the hash function needs to have to ensure the security of the application.

(i) Secure storage of passwords

Psword = myDoB



Password file:

User_A ***

User_B ***

What should we put in there?
What if backup tape is stolen?
What property do we need?

(ii) Protection against viruses

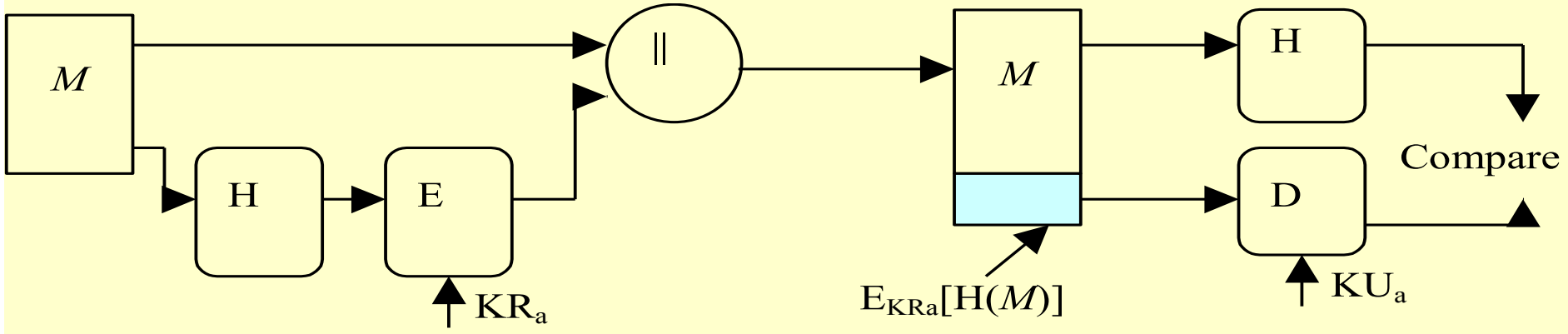
- A software manufacturer wants to ensure that the executable files are received by users without modification.
- They send out each file to users and publish its hash value on an authentic website.

Exercise Question – E5.2 (continue)

(iii) Digital signatures

One party can sign a message, M, and many parties can verify.
Such applications include contract signing, code signing, etc.
A raw signature scheme only signs a few hundred (e.g. 160) bits.
What properties do we need?

Integrity, authentication, and *non-repudiation* are provided.
*This is the essence of the digital signature technique.



Exercise Question – E5.3

- This is a Coin Flipping Over the Telephone problem.

  (i) Assuming there is only one car, and Alice and Bob have to decide who can have this car (only one of them can have it, i.e. they cannot share it). Alice and Bob cannot see each other, and they do not trust each other. So they have decided to make a decision by flipping a coin over the telephone. Design a protocol to support this using a hash function.

  (ii) Identify any factors that you should consider to ensure the security of this protocol.

Exercise Question – E5.3 (hint)

Assumption: Alice and Bob agree that if the outcome is 1 then Bob takes the car, if it is 0 then the car goes to Alice.

Solution 1 (an insecure solution):

- Alice generates a random bit *b*: 0=heads, 1=tails.
- Alice asks Bob: heads or tails?
- Bob sends Alice his choice_B: 'heads' (or 'tails').
- Alice compares *b* with choice_B: if b=choice_B, then outcome=1; if not, outcome=0.
- Alice sends the comparison outcome to Bob.

What is wrong with this solution?
Please fix the flaw.

Conclusions

- Message encryption can not always provide message authentication.
- Message authentication is typically achieved by using a cryptographic checksum.
- A checksum produced from a hash function should be protected with a private key (of a public-key cipher) or a symmetric key.
- A symmetric-key protected checksum may also be produced by using a block cipher in CBC mode, or a keyed hash function such as HMAC.
- Authenticated encryption is for achieving message authentication as well as confidentiality in a way that maximises parallel operations.
- Hash functions are commonly used in many other applications as well.