CSCI471/971
Modern Cryptography

Cryptographic Hash Functions
& Message Authentication Codes

Rupeng Yang
SCIT UOW

RoadMap

- Week 1-2: Preliminaries
- Week 3: How to protect the message confidentiality
- Week 4: How to protect the message integrity

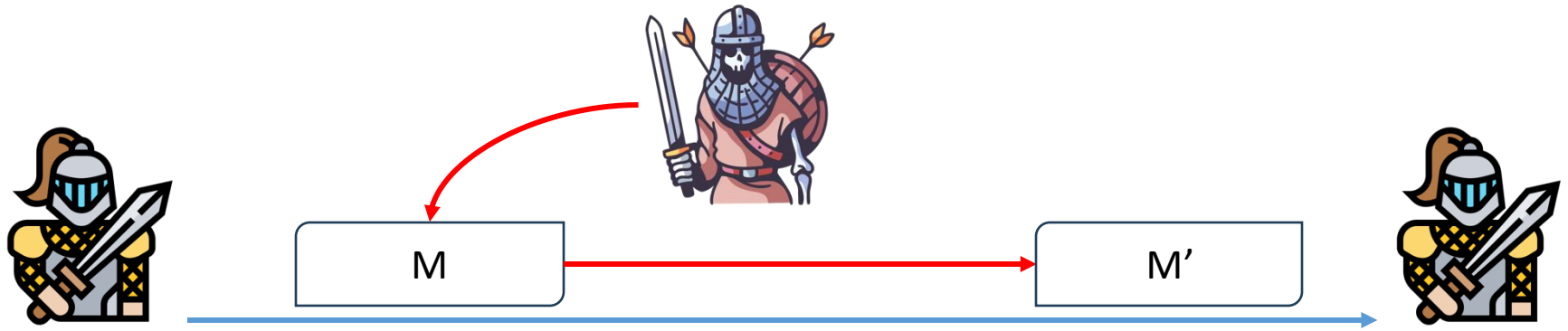
Roadmap

Classical Ciphers

One-Time Pad

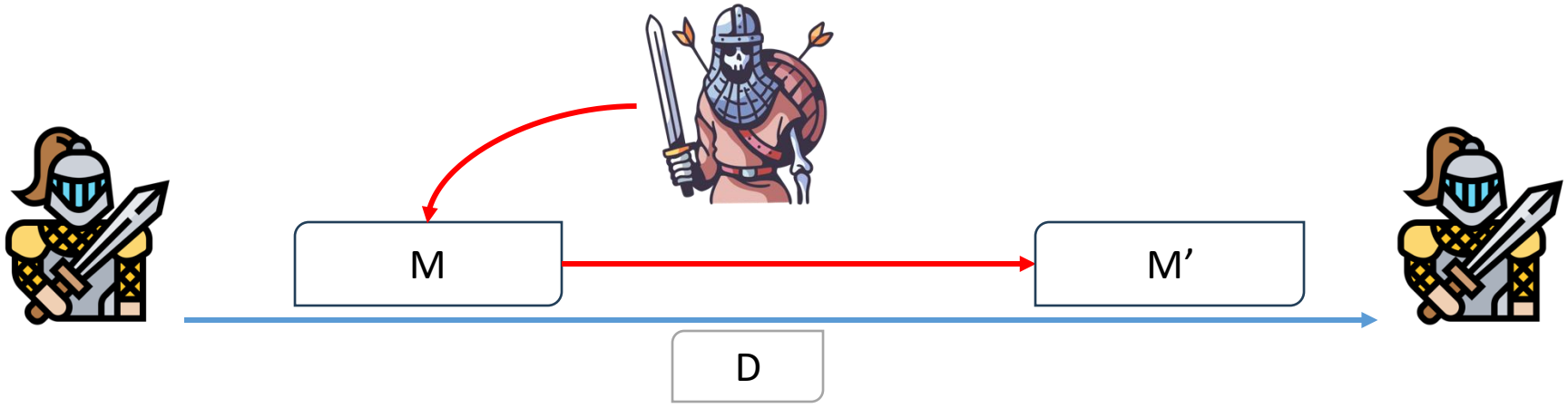
What is integrity?

- Integrity is to assure that the content of the message has not been altered during transmission.
  - Actually, we cannot prevent the adversary from modifying the message.
  - We can only ensure that the any modification on the message can be detected.



What is integrity?

- Integrity is to assure that the content of the message cannot be altered without being detected.
- We first consider a weaker notion of integrity, where the receiver is given a short digest about the message.
  - Why this security guarantee is meaningful?
- This can be guaranteed by using a cryptographic hash function.



Cryptographic Hash Functions

Hash Functions

- A hash function (algorithm) is denoted by $h: \{0, 1\}^* \rightarrow \{0, 1\}^n$, where n is a security parameter.

- Given x, it is easy to compute h(x).
- x can be of arbitrary length while h(x) has a fixed length.
  - There exist different inputs x_1 and x_2 such that y=h(x_1)=h(x_2).
  - (x_1,x_2) is called a collision for the hash function h.
- We require the hash function to be collision resistant, i.e., it is computationally difficult to find any collision for h.
- If h is collision resistant, then we can use the output of h to check the **integrity** of the input.

Hash Functions (Security Definitions)

- A hash function (algorithm) is denoted by $h: \{0, 1\}^* \rightarrow \{0, 1\}^n$

Adversary's Target: Given a hash function h, find $x \neq x'$ such that $h(x)=h(x')$.

Collision Resistance:

Given a hash function $h: \mathcal{X} \rightarrow \mathcal{Y}$, there is no efficient adversary to find $x, x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x')=h(x)$ with a non-negligible probability.

Collision Resistance: The birthday attack

- The Problem: Given a hash function h:$\mathcal{X} \rightarrow \mathcal{Y}$ find $x' \neq x$ and $h(x') = h(x)$.

- The (ε,q)-Algorithm:

  Choose $\mathcal{X}_0 \subseteq_R \mathcal{X}$: $|\mathcal{X}_0| = q$

  For each $x_i \in \mathcal{X}_0$ (Go through the elements in order.)

  $y_i \leftarrow h(x_i)$

  If $y_i = y_{i'}$ for $i' < i$

  return $(x_i, x_{i'})$

  return "Failure"

- What chance do we have?

$$\varepsilon = 1 - \prod_{i=1}^{q-1} \left(1 - \frac{i}{|\mathcal{Y}|} \right)$$

- This is also related to the birthday paradox.
  - How many people do you need in a room before the probability of any two sharing a birthday is at least ½? 23!
  - In the above, we can roughtly have $q = |\gamma|^{\frac{1}{2}} \, for \, \varepsilon = 0.5$
  - h: {0, 1}* $\rightarrow$ {0, 1}^{n} · We only need about q=2^{64} for n=128

Hash Functions

- A hash function (algorithm) is denoted by $h: \{0, 1\}^* \rightarrow \{0, 1\}^n$, where n is a security parameter.

h(x) cannot be too short.

- Recommended message digest lengths (in bits):
- 128 (MD5),
- 160 (SHA-1),
- 224/256/384/512 (SHA-2),
- 224/256/384/512 (SHA-3)

Secure Hash Algorithm (SHA)

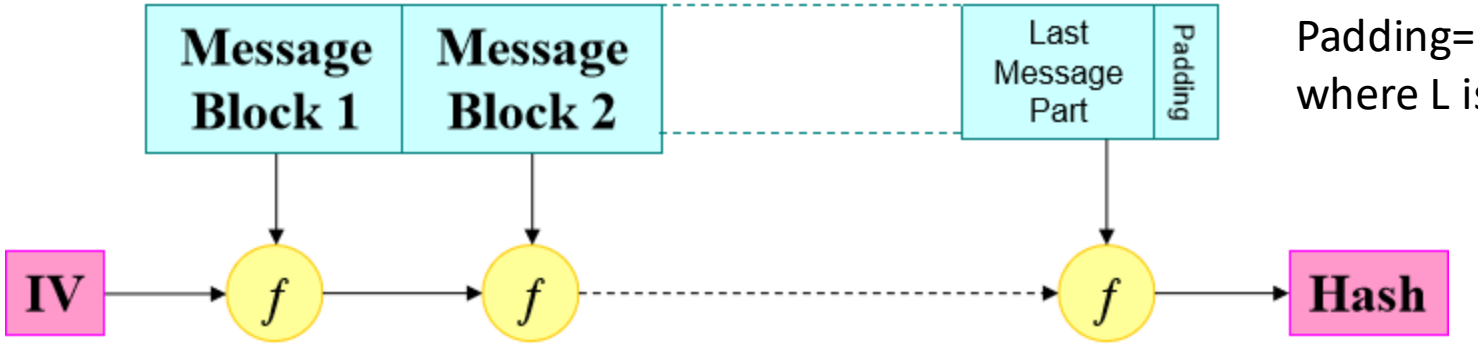
- SHA was originally designed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993
- It was revised in 1995 as SHA-1
  - Produces 160-bit hash values
  - Was broken in 2017
- In 2002 NIST produced a revised version SHA-2 that defined three new versions of SHA with hash value lengths of 256, 384, and 512
  - SHA-2 and SHA-1 share a similar design.
- In 2012, the third generation SHA-3 was standardized
  - SHA-3 uses a different structure.
  - It is intended to complement SHA-2.

Design of SHA-1

- The design works in two steps:
  - Design a compressing function that takes a fixed-length input and returns a shorter, fixed-length output.
  - Upgrade the compressing function to support an arbitrary-long message using the so-called Merkle-Damgård Iterative Structure.

  - We focus on the second step.

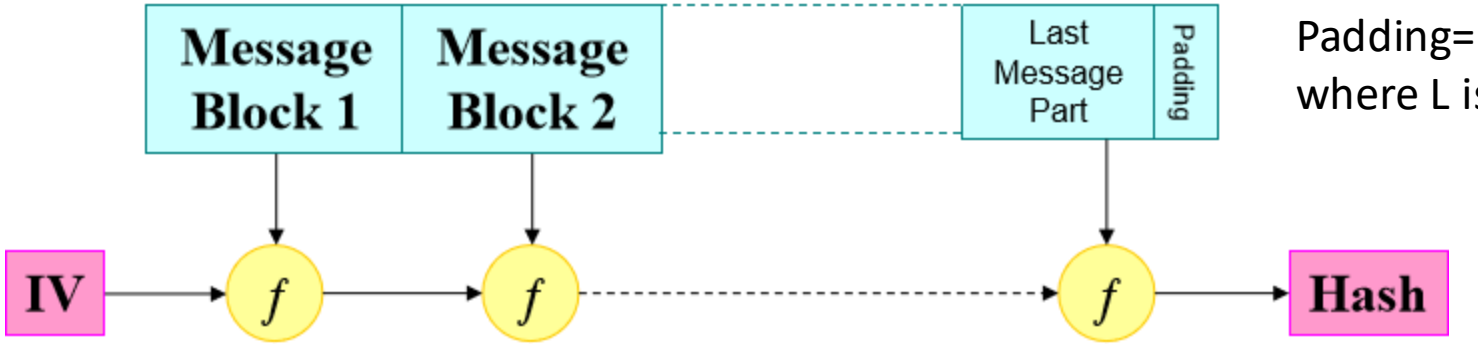
Merkle-Damgård Iterative Structure



Padding=L || 100…0
where L is the length of the message

- The function f takes a fixed-length input and returns a shorter, fixed-length output.
- The Merkle-Damgård structure constructs a hash function that takes an arbitrary-length input and returns a fixed-length output from f.
  1. The message length is appended to the message and the message is properly padded
  2. The function f takes
     - a message block and
     - the previous hash result (the algorithm starts with a fixed initialization vector IV)
  3. The value after the last block is taken to be the hash value for the entire message
- If the function f is collision-resistant, then the hash function will also be collision-resistant

Merkle-Damgård Iterative Structure



Padding=L || 100…0
where L is the length of the message

- If the function f is collision-resistant, then the hash function will also be collision-resistant. Now, given two messages:
  - If the final hash values are identical, then the last message parts, the message lengths, and the "last but one" outputs of f will be identical.
    - Why?
  - We can repeat the above arguments to show that all message blocks are identical.

Hash Functions (Alternative Security Definitions)

1.Collision Resistance:

Given a hash function h:$\mathcal{X}\rightarrow\mathcal{Y}$, there is no efficient adversary to find x, x′$\in\mathcal{X}$ such that x′≠x and h(x′)=h(x).

2.Second Pre-Image Resistance:

Given a hash function h:$\mathcal{X}\rightarrow\mathcal{Y}$ and a uniform x$\in\mathcal{X}$, there is no efficient adversary to find x′$\in\mathcal{X}$ such that x′≠x and h(x′)=h(x).

3. Pre-Image Resistance/Onewayness:

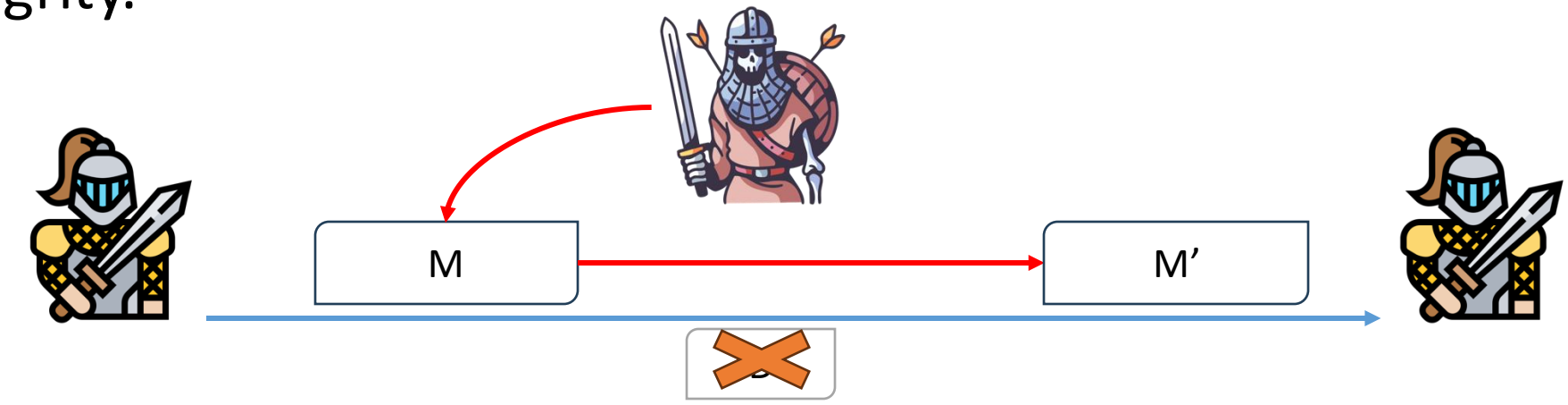
Given a hash function h:$\mathcal{X}\rightarrow\mathcal{Y}$ and y$\in\mathcal{Y}$, where y=h(x) for a uniform x, there is no efficient adversary to find x’$\in\mathcal{X}$ such that y=h(x’).

What are hash functions used for?

- For File Fingerprint
  - Generate the hash value of a file and check if the file has been modified. This allows us to
    - Check the integrity of the file
    - Deduplication
- For Password storage.
  - Server stores hash values of passwords instead of the passwords.
  - Client also sends the hash value of his/her password.
  - This can protect the confidentiality of the password.
- For building advanced cryptographic schemes
  - Message authentication code
  - Digital signature
- For proof-of-work
  - To find an input (of specific form) that can be hashed to a value with N leading 0s.
  - This is used in bitcoin.

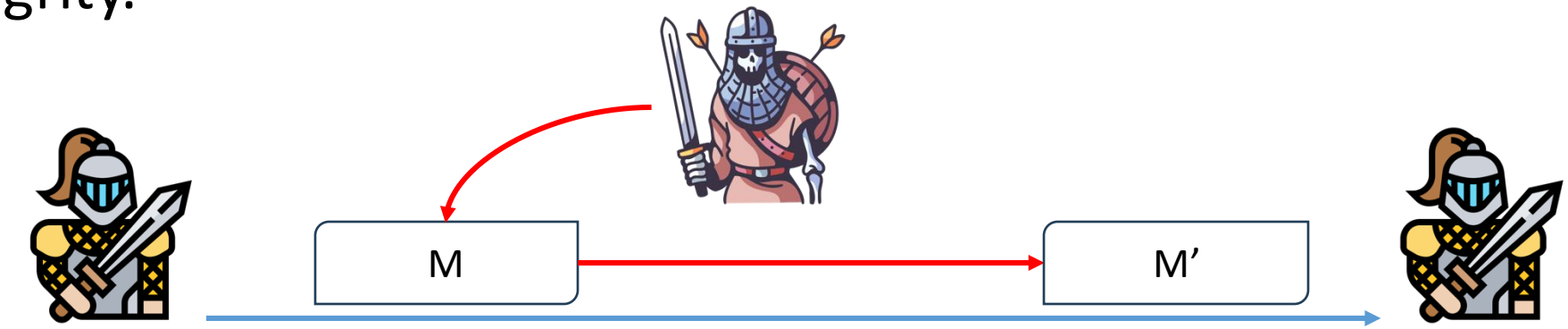
What is integrity?

- Integrity is to assure that the content of the message cannot be altered without being detected.
- We next consider the standard integrity, where the sender and the receiver does not share any auxiliary information about the message.
  - Yet, a secret key is shared between the sender and the receiver.
- Message authentication code is used to protect the standard integrity.



What is integrity?

- Integrity is to assure that the content of the message cannot be altered without being detected.
- We next consider the standard integrity, where the sender and the receiver does not share any auxiliary information about the message.
  - Yet, a secret key is shared between the sender and the receiver.
- Message authentication code is used to protect the standard integrity.

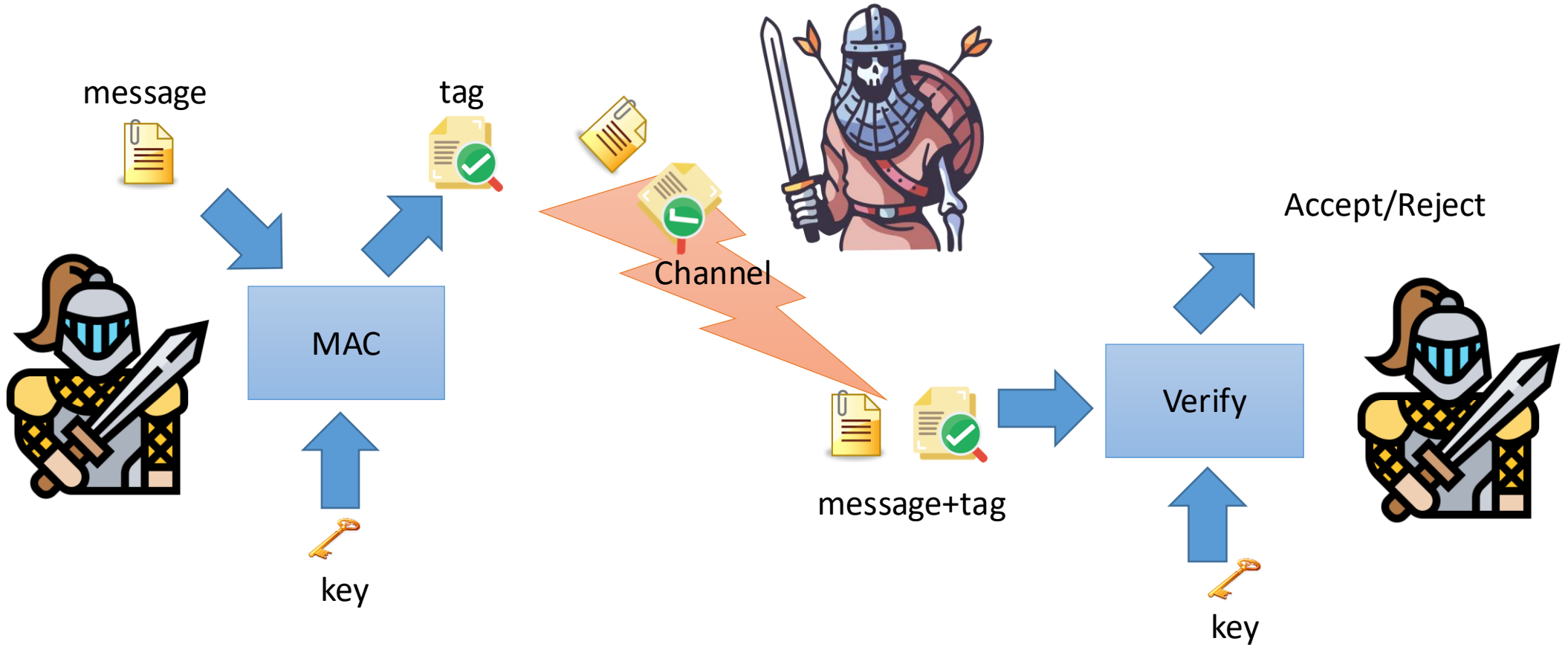


Message Authentication Codes

Message Authentication Code

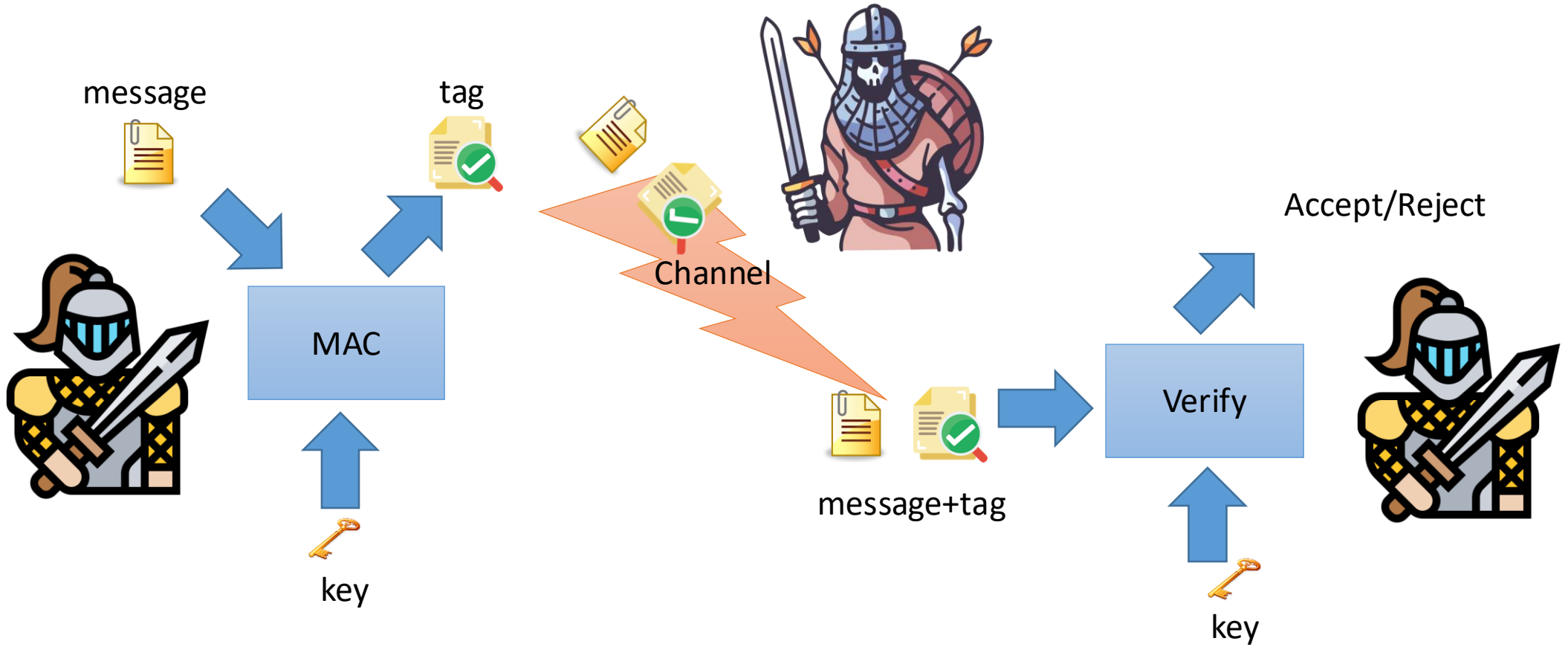
- A symmetric-key cryptosystems for message integrity and authenticity
- Produce a cryptographic checksum (Use the checksum to verify)
- Common constructions from
  - Hash function
  - Block cipher

Message Authentication Code



Can we use a hash function as the MAC algorithm directly?

Message Authentication Code



Can we use a hash function as the MAC algorithm directly?
A secret key is essential in this model.

Message Authentication Code

Application Scenario:

Sender and receiver generate and share a secret key K. To transmit a message M, the sender first generate the tag T for M using the key. Then it sends the message M and the tag T to the receiver. The receiver verifies if the the tag T is a valid tag for the message M using the same key.

Usually, the verification is done by generating a new tag T' for the message M, and checks if T=T'.

Definition of Message Authentication Code

- KeyGen(λ): Taking as input a security parameter λ, the key generation algorithms returns a key K.
- MAC(M, K): Taking as input a message and a key K, the deterministic message authentication code generation algorithm returns a tag denoted by T.

$$T=MAC(M, K)$$

- The correctness is guaranteed by the fact that the MAC algorithm is deterministic.

Security Model for MAC

- Our objective is to prevent the adversary from modifying the message without being detected.
- The adversary can modify both the message and the tag, since it controls the communication channel.
- Thus, we require that the adversary cannot generate a valid message tag pair if it modifies the message
- **Security Goal**
  - **Unforgeable**: It is hard to generate a valid tag for a *new* message.

Security Model for MAC

- Adversary's capability
  - Known tag attack: The adversary knows some messages and the associated tags for them.
  - Chosen Message Attack: Attacker is allowed to choose some messages, and receives the corresponding tags.

Security Model for MAC: Unforgeability under Chosen Message Attacks

Algorithms KeyGen, MAC are public.

1. Run KeyGen to get k

2.2 Run MAC(k, m) to get t

2.1 Attacker sends m to challenger

2.3 Send t to the Attacker

Attacker wins if MAC(k,m*)=t*
and m* is not queried in Step 2.

3. Attacker returns (m*,t*)

A message authentication code Scheme is Secure if ***NO*** efficient attacker can win with a probability of 1/poly(λ).

Security Model for MAC: Unforgeability under Chosen Message Attacks

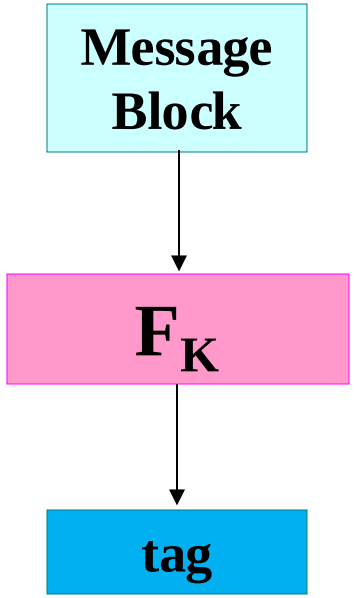
- Does the security definition sufficient?
- What if the adversasry reuses one of the message/tag pairs it has seen before.
  - This is called **replay attack** and the attack cannot be prevented if the sender and receiver are stateless.
  - We can solve the problem by some application-specific methods:
    - Sharing a synchronized state between sender and receiver.
    - Using timestamps.
    - Using the challenge/response model.
    - …
  - The current definition is chosen by tradeoffing among security, simpleness, generality, etc.
  - Also, it is important to know what is not guaranteed by a cryptosystem.

Construction from Blockcipher

MAC(K,M):

1. Output T=F(K, M)

Here, F is a blockcipher.



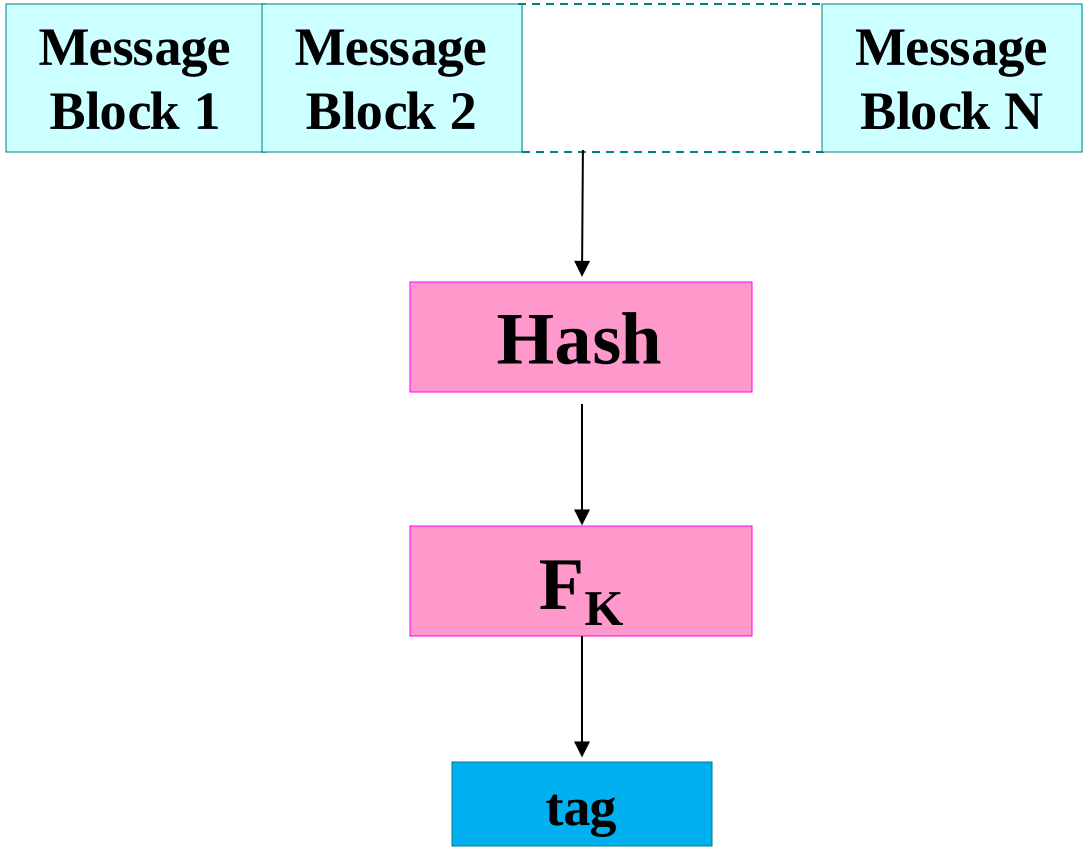
Why the MAC is secure:

- The underlying blockcipher is assumed to be a **pseudorandom function (PRF)**, i.e., outputs of the blockcipher is indistinguishable from random values.
- Previous tags will just be some random values, i.e., no one could learn any information about the key from the tags.
- The task of generating a valid tag for a new message is equivalent to generating a specific random value.
- Therefore, successfully attacking this MAC scheme is as hard as guessing a random value of (e.g.) 128 bits.

Can we use this MAC scheme directly for long messages?

- NO! Please explore the reason in A1.

Construction from Hash and Blockcipher: Hash and MAC



MAC(K,M):

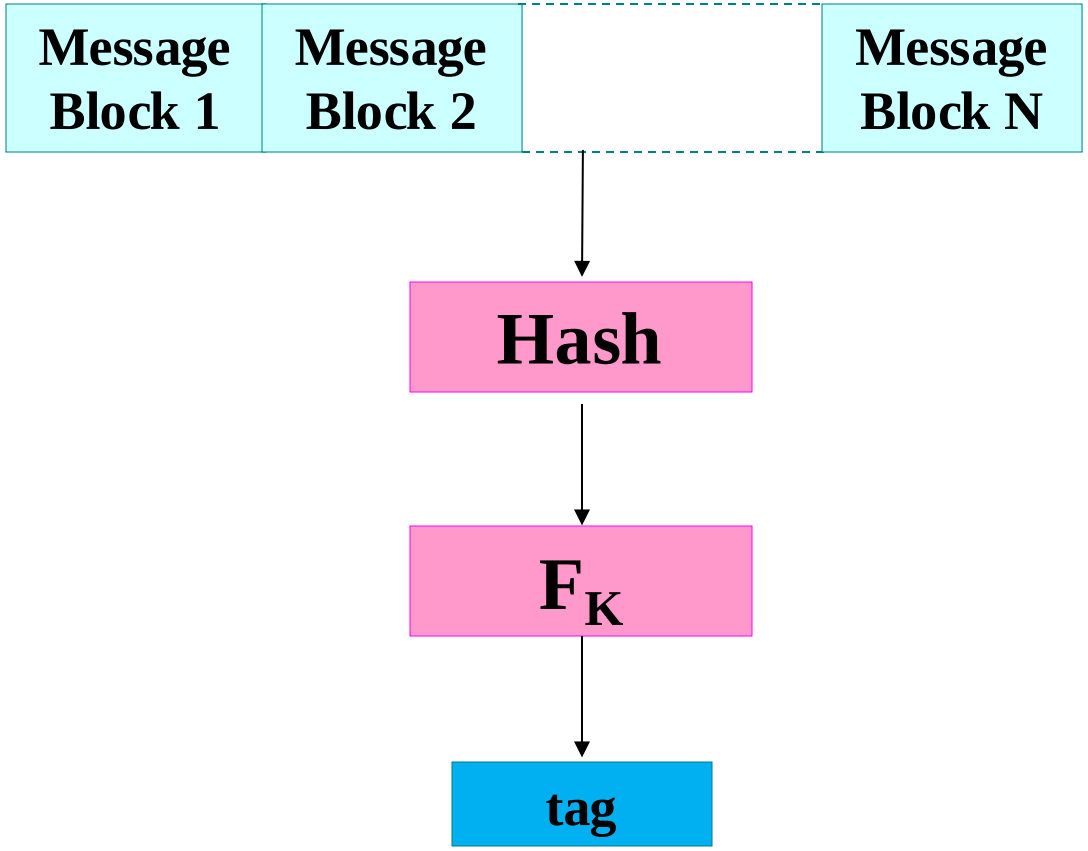
1. W=H(M)
2. Output T=F(K, W)

Here, H is a hash function and F is a blockcipher.

Why the MAC is secure:

- Any new message will be mapped to a new digest W.
  - Why?

Construction from Hash and Blockcipher: Hash and MAC



MAC(K,M):

1. W=H(M)
2. Output T=F(K, W)

Here, H is a hash function and F is a blockcipher.

Why the MAC is secure:

- Any new message will be mapped to a new digest W.
  - Why?
- The adversary cannot generate a valid tag for a new digest W.
  - Why?

Construction from Hash and Blockcipher: Hash and MAC

- Drawback of Hash-and-MAC
  - It requires implementing two cryptographic primitives
  - There is often a mismatch between the output length of hash functions and the block length of block ciphers
- Design of HMAC
  - HMAC has been chosen as the mandatory-to-implement MAC for IP security
  - HMAC has also been issued as a NIST standard (FIPS 198)

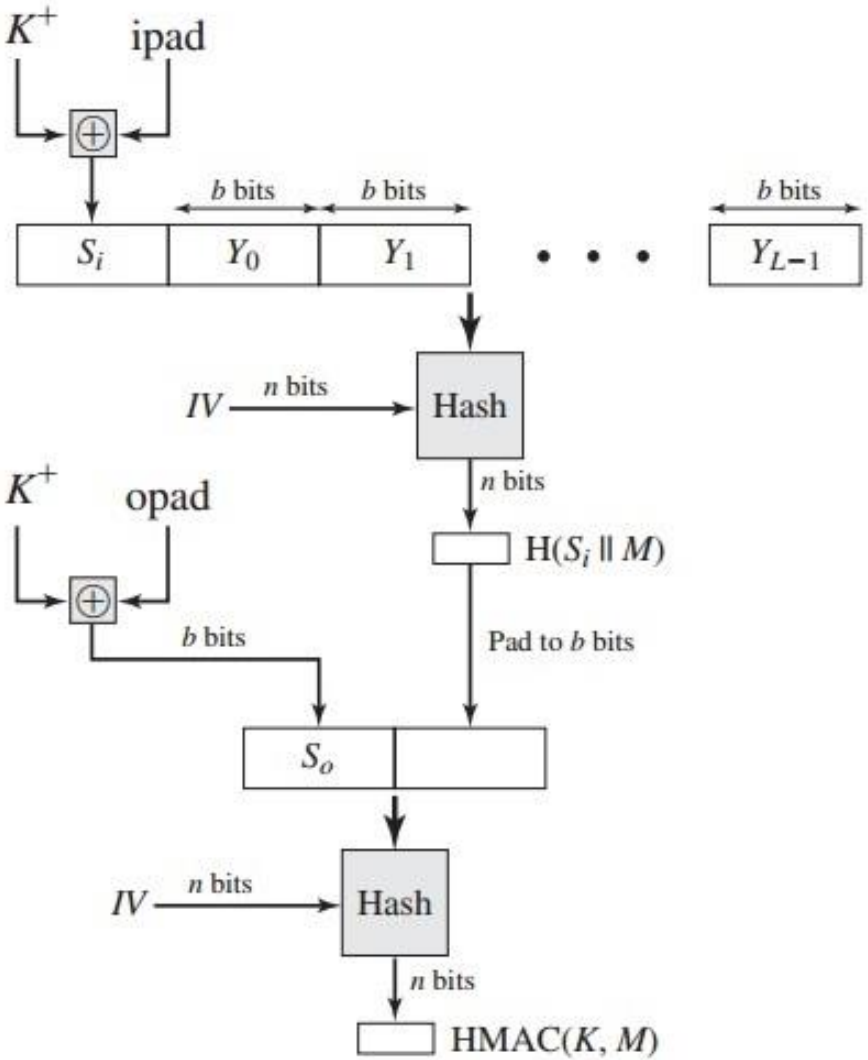


Figure 12.5 HMAC Structure

- K^+ is derived from K by padding with zeros on the left.
- ipad = [0X36 * blocksize]
- opad = [0X5C * blocksize]

- Security of HMAC
  - Security of Hash
  - H(K || M) can be modeled as a blockcipher $F_K(M)$

- The key used in the first step is not necessary if a secure hash function is used. But this key has saved the HMAC when the MD5 hash function was suddenly broken.

Encryption with IND-CCA Security

Security Model of Symmetric-Key Encryption (IND-CCA)

Setup: The challenger chooses a random key K.

Phase 1: The adversary can choose any M for encryption queries and learns the encrypted result; it can also choose any CT for decryption queries and learns the decryption result.

Challenge: The adversary can choose any two different messages M_0 and M_1. The challenger chooses a random b and computes the challenge ciphertext CT*=Enc(M_b, K), which is given to the adversary.

Phase 2: The adversary can choose any M for encryption queries and choose any CT different from CT* for decryption queries.

Guess: The adversary returns the guess c' and wins if b'=b.

We say that the encryption is secure if no P.P.T adversary can win with a probability of ½+1/poly(λ).

Difficulty in Achieving IND-CCA Security

- We have seen a few chosen-ciphertext attacks that can break the security of IND-CPA secure SKE schemes.
- The attackers can obtain inforamtion about the encrypted message in a ciphertext by slightly modifying the ciphertext.
- How can we defend against the attacks?
  - What if the adversary cannot generate a ciphertext?
  - How can we prevent the adversary from generating a valid ciphertext?
  - Note that, we only need to prevent the adversary from generating a new valid ciphertext.

Constructing IND-CCA Secure SKE

Let $\Pi_E = (\mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme and let $\Pi_M = (\mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code, where in each case key generation is done by simply choosing a uniform n-bit key. Define a private-key encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

- Gen': on input 1^n, choose independent, uniform $k_E, k_M \in \{0,1\}^n$ and output the key (k_E, k_M).
- Enc': on input a key (k_E, k_M) and a plaintext message m, compute $c \leftarrow \mathsf{Enc}_{k_E}(m)$ and $t \leftarrow \mathsf{Mac}_{k_M}(c)$. Output the ciphertext $\langle c, t \rangle$.
- Dec': on input a key (k_E, k_M) and a ciphertext $\langle c, t \rangle$, first check if $\mathsf{Vrfy}_{k_M}(c, t) \stackrel{?}{=} 1$. If yes, output $\mathsf{Dec}_{k_E}(c)$; if no, output $\perp$.

Why the solution is secure:

- The adversary can only obtain $\perp$ from the decryption queries.
  - Why
- Then the security is guaranteed by the IND-CPA security of Π_E.

Introduction to Modern Cryptography. - Jonathan Katz and Yehuda Lindell

How to implement a hash function and a MAC algorithm in practice

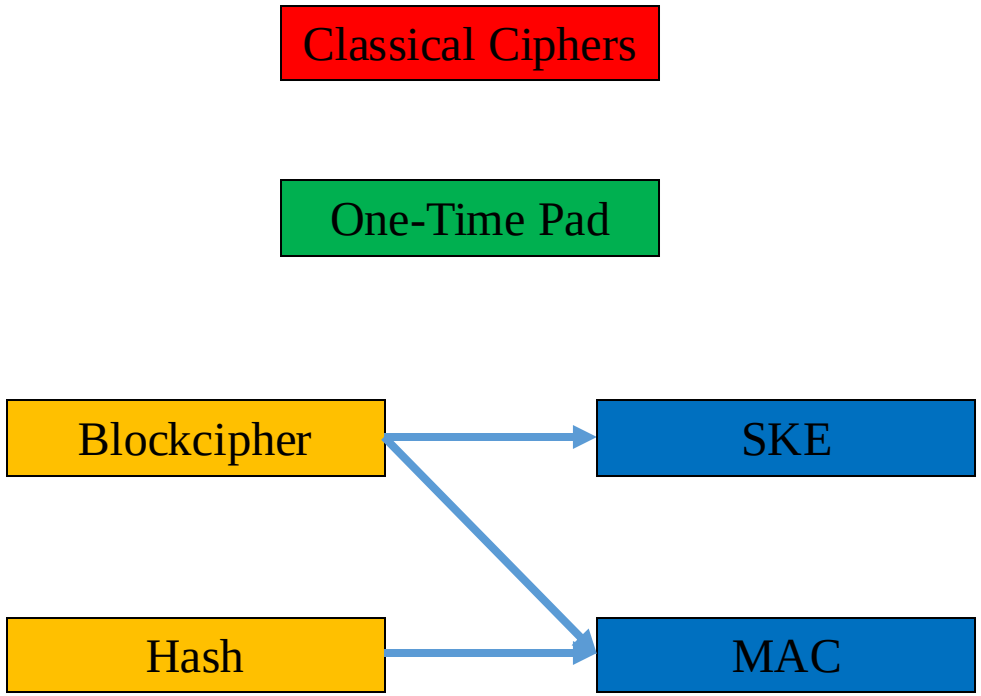
```
from cryptography.hazmat.primitives import hashes, hmac
import os
import binascii

# Read the message
f = open("plaintext.png", mode="rb")
data = f.read()

# Generate the digest
digest = hashes.Hash(hashes.SHA256())
digest.update(data)
y=digest.finalize()
print(binascii.b2a_hex(y))

# Generate the secret key
key = os.urandom(32)

# Generate the MAC
h = hmac.HMAC(key, hashes.SHA256())
h.update(data)
tag= h.finalize()
print(binascii.b2a_hex(tag))
```

```
iMac:codes orbbyrp$ python3.11 mac.py
b'922a35ef3fed02c620a5c856820e373e36afa70b41b177e2d5d33329f7f26b57'
b'd2cf9f72aa5a52e8bb5f1897427ab0e278d529b46660ef04d1764dcd827428ab'
iMac:codes orbbyrp$ shasum -a 256 plaintext.png
922a35ef3fed02c620a5c856820e373e36afa70b41b177e2d5d33329f7f26b57  plaintext.png
```

The codes are implemented using the pyca/cryptography library (https://cryptography.io/).

- This is a python library depending on OpenSSL

Roadmap



Summary

- Cryptographic Hash
  - Syntax
  - Collision Resistance
    - Birthday Attack
  - Merkle-Damgård Structure
  - Alternative hash security properties
  - Applications of Hash
- Message authentication code
  - Application scenarios
  - Definition
    - Syntax and correctness
    - Security goals
    - Adversary's capabilities
    - The security definition
    - Replay attack
  - Constructions
    - Construction from blockcipher
      - Security for one message block*
      - Insecurity for multiple messages
    - Hash and MAC
      - Security*
    - HMAC
- IND-CCA SKE
  - Construction from SKE + MAC
    - Security*

In both SKE and MAC, we will assume that all honest parties hold a secret key. Is this necessary?