CS306: Introduction to IT Security

Fall 2020

Lecture 6: MACs & Hashing

Instructor: **Nikos Triandopoulos**

October 6, 2020

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Assignment Project Exam Help

https://powcoder.com

6.0 Announcements

Add WeChat powcoder

CS306: Other announcements

- HW2 to come by Friday this week
- Road ahead
  - no lecture on October 13 (next week, classes will run on Monday schedule)
  - regular lecture on October 20
  - midterm exam on October 27 (in whatever format)

CS306: Tentative Syllabus

Week	Date	Topics	Reading	Assignment
1	**Sep 1**	**Introduction**	**Lecture 1**	**-**
2	**Sep 8**	**Symmetric-key encryption**	**Lecture 2**	**Lab 1**
3	**Sep 15**	**Perfect secrecy**	**Lecture 3**	**Lab 2, HW 1**
4	**Sep 22**	**Ciphers in practice I**	**Lecture 4**	**Lab 3, HW 1**
5	**Sep 29**	**Ciphers in practice II**	**Lecture 5**	**Lab 4**
6	Oct 6	MACs & hashing		
-	Oct 13	**No class (Monday schedule)**		
7	Oct 20	Public-key cryptography		

CS306: Tentative Syllabus (continued)

Week	Date	Topics	Reading	Assignment
8	Oct 27	**Midterm**	All materials covered	
9	Nov 3	Network/Web security		
10	Nov 10	Software/Database security		
11	Nov 17	Cloud security		
12	Nov 24	AC/Authentication/Privacy		
13	Dec 1	Economics		
14	Dec 8	Legal & ethical issues		
15	Dec 10 (or later)	**Final** (closed "books")	All materials covered*	

* w/ focus on what covered after midterm

Last week

- Ciphers in practice
  - Revision
    - the big picture, computational security, pseudo-randomness, stream ciphers, PRGs
  - Block ciphers, pseudorandom functions
  - Modes of operations
  - DES, AES
- Demo
  - The Caesar and Vigenère ciphers and their cryptanalysis (Afternoon)
  - Pseudo-randomness in practice (Evening)

Today

- Message authentication
  - MACs
  - Replay attacks
  - Constructions
- Cryptographic hashing
  - Hash functions
  - Constructions
- Demo
  - Hash functions in practice

6.1 Message authentication

Recall: Integrity

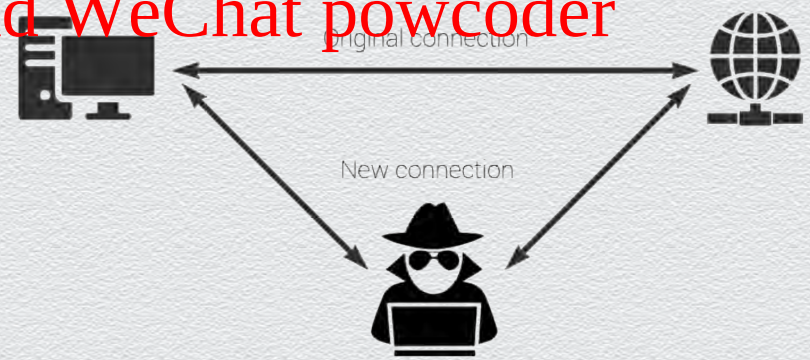
Fundamental security property

- **an asset is modified only by authorized parties**
- "I" in the CIA triad

> *"computer security seeks to prevent **unauthorized** viewing (confidentiality) or **modification (integrity) of data** while preserving access (availability)"*

Alteration

- main threat against integrity of **in-transit** data
- e.g., MITM attack



Man in the middle, Phisher, or annonymous proxy

Security problems studied by modern cryptography

- Classical cryptography: **message encryption**
  - early crypto schemes tried to provide **secrecy / confidentiality**

- Modern cryptography: **wide variety** of security problems

  - today we need to study a large set of **security properties** beyond secrecy

- The sibling of message encryption: **message authentication**
  - **another cornerstone** of any secure system aiming to provide **authenticity & integrity**

Message authentication: Motivation

Information has **value**, but only when it is **correct**

- random, incorrect, inaccurate or maliciously altered data is **useless** or **harmful**
  - **message authentication = message integrity + authenticity**
    - while in transit (or at rest), no message should be **modified** by an outsider
    - no outsider can **impersonate** the stated message sender (or owner)
- it is often necessary / worth to protect critical / valuable data
  - **message encryption**
    - while in transit (or at rest), no message should be **leaked** to an outsider

Example 1

Secure electronic banking

- a bank receives an electronic request to transfer $1,000 from Alice to Bob

Concerns

- who ordered the transfer, Alice or an attacker (e.g., Bob)?

- is the amount the intended one or was maliciously modified while in transit?

  - adversarial Vs. random message-transmission errors
    - standard error-correction is **not sufficient** to address this concern

Example 2

Web browser cookies

- a user is performing an online purchase at Amazon
- a “cookie” contains session-related info, as client-server HTTP traffic is stateless
  - stored at the client, included in messages sent to server
  - contains client-specific info that affects the transaction
    - e.g., the user’s shopping cart along with a discount due to a coupon

Concern

- was such state maliciously altered by the client (possibly harming the server)?

Integrity of communications / computations

Highly important

- any unprotected system cannot be assumed to be trustworthy w.r.t.
  - origin/source of information (due to impersonation attacks, phishing, etc.)
  - contents of information (due to man-in-the-middle attacks, email spam, etc.)
  - overall system functionality

Prevention Vs. detection

- unless system is “closed,” adversarial tampering with its integrity **cannot be avoided**!
- goal: identify system components that are not trustworthy
  - **detect tampering** or **prevent undetected tampering**
    - e.g., avoid “consuming” falsified information

Encryption does not imply authentication

A common misconception

“since ciphertext c hides message m, Mallory cannot meaningfully modify m via c”

Why is this incorrect?

- all encryption schemes (seen so far) are based on one-time pad, i.e., masking via XOR

- consider flipping a single bit of ciphertext c; what happens to plaintext m?
  - such property of one-time pad does not contradict the secrecy definitions

Generally, secrecy and integrity are distinct properties

- encrypted traffic generally provides **no integrity** guarantees

6.2 Message authentication codes (MACs)

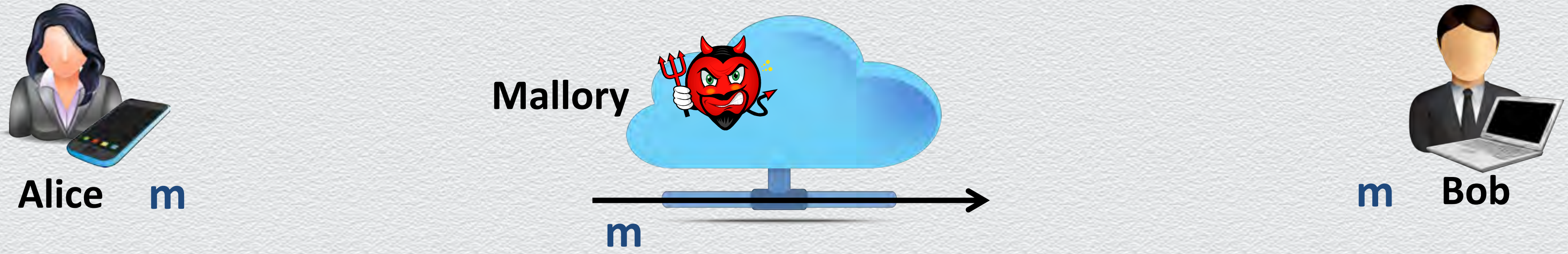
Problem setting: Reliable communication

Two parties wish to communicate over a channel

- Alice (sender/source) wants to send a message m to Bob (recipient/destination)

Underlying channel is unprotected

- Mallory (attacker/adversary) can manipulate any sent messages
- e.g., message transmission via a compromised router



Solution concept: Symmetric-key message authentication

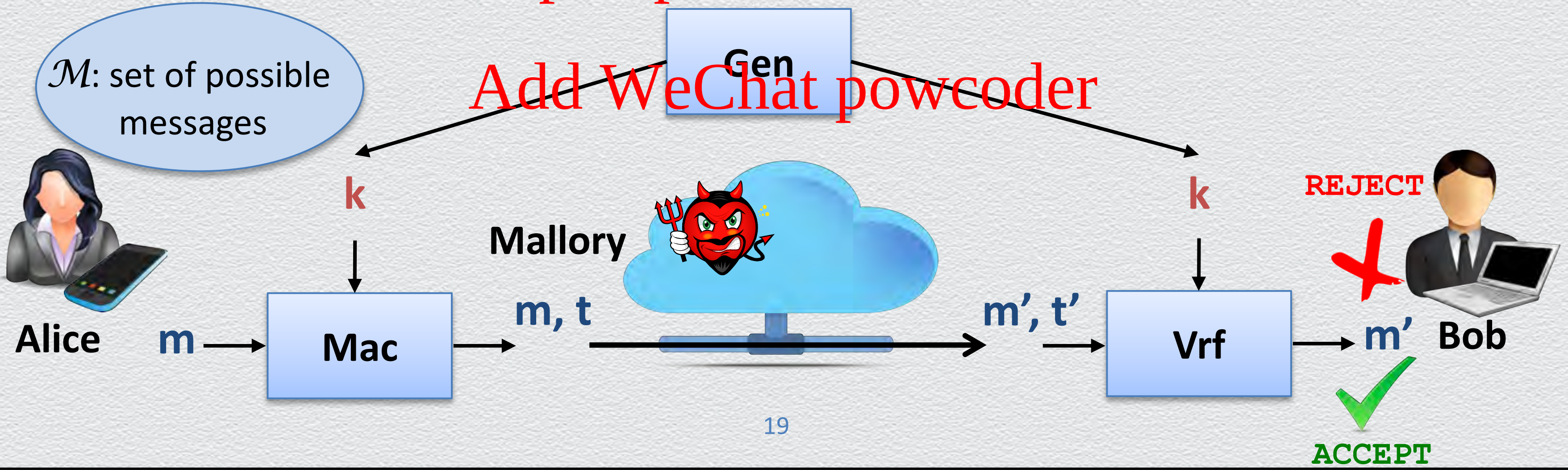
Main idea

- secretly annotate or "sign" message so that it is **unforgeable** while in transit
  - Alice **tags** her message m with **tag t**, which is sent **along** with **plaintext m**
  - Bob **verifies** authenticity of received message using tag t
  - Mallory can manipulate m, t but "**cannot forge**" a fake verifiable pair m', t'
  - Alice and Bob share a **secret key k** that is used for both operations

Security tool: Symmetric Message Authentication Code

Abstract cryptographic primitive, **a.k.a. MAC**, defined by

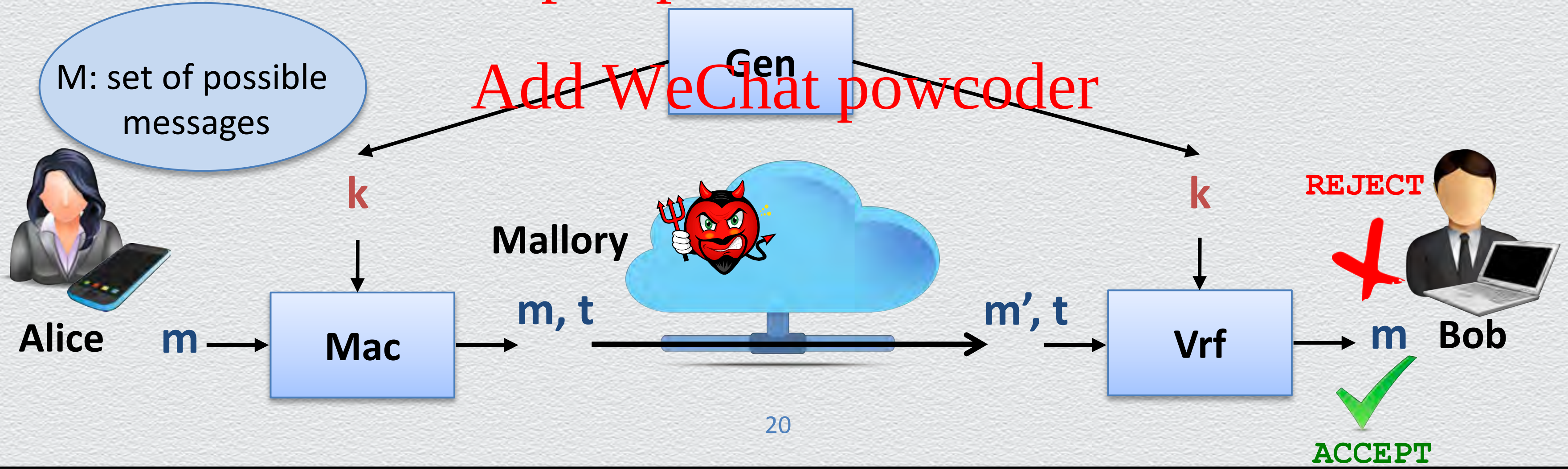
- a **message space** $\mathcal{M}$; and
- a triplet of algorithms **(Gen, Mac, Vrf)**
  - Gen, Mac are probabilistic algorithms, whereas Vrf is deterministic
  - Gen outputs a uniformly random key k (from some key space $\mathcal{K}$)



Desired properties for MACs

By design, any MAC should satisfy the following

- **efficiency**: key generation & message transformations "are fast"
- **correctness**: for all m and k, it holds that $Vrf_k(m, Mac_k(m)) = \texttt{ACCEPT}$
- **security**: one "cannot forge" a fake verifiable pair m', t'



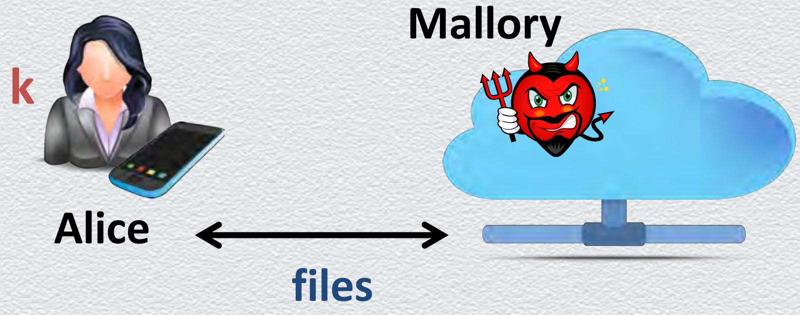
Main application areas

Secure communication

- **verify authenticity of messages** sent among parties
- assumption
  - Alice and Bob **securely generate, distribute and store shared key k**
  - attacker does not learn key k

Secure storage

- **verify authenticity of files** outsourced to the cloud
- assumption
  - Alice **securely generates and stores key k**
  - attacker does not learn key k



Conventions

Random key selection

- typically, Gen selects key k **uniformly at random** from the key space $\mathcal{K}$

Canonical verification

- when Mac is deterministic, Vrf typically amounts to re-computing the tag t
  - $Vrf_k(m, t)$: 1. $t' := Mac_k(m)$ 2. if $t = t'$, output `ACCEPT` else output `REJECT`
- but conceptually the following operations are distinct
  - authenticating m (i.e., running Mac) Vs. verifying authenticity of m (i.e., running Vrf)

MAC security

Attacker **wins** the game if

1. $\mathrm{Vrf}_k(m^*, t^*) = \texttt{ACCEPT}$ &
2. m^* not in $\mathcal{Q}$

MAC scheme
(Gen, Mac, Vrf)

$\mathcal{T}$

Gen $\longrightarrow$ k

$\mathrm{Mac}_k(m_i) \longrightarrow t_i$

m_1

t_1

m_2

t_2

…

m^*, t^*

$\mathcal{A}$ **Mac(k,)**

$\mathcal{Q} = m_1, m_2, \ldots$

The MAC scheme is **secure** if any PPT $\mathcal{A}$ wins the game only negligibly often.

Assignment Project Exam Help

https://powcoder.com

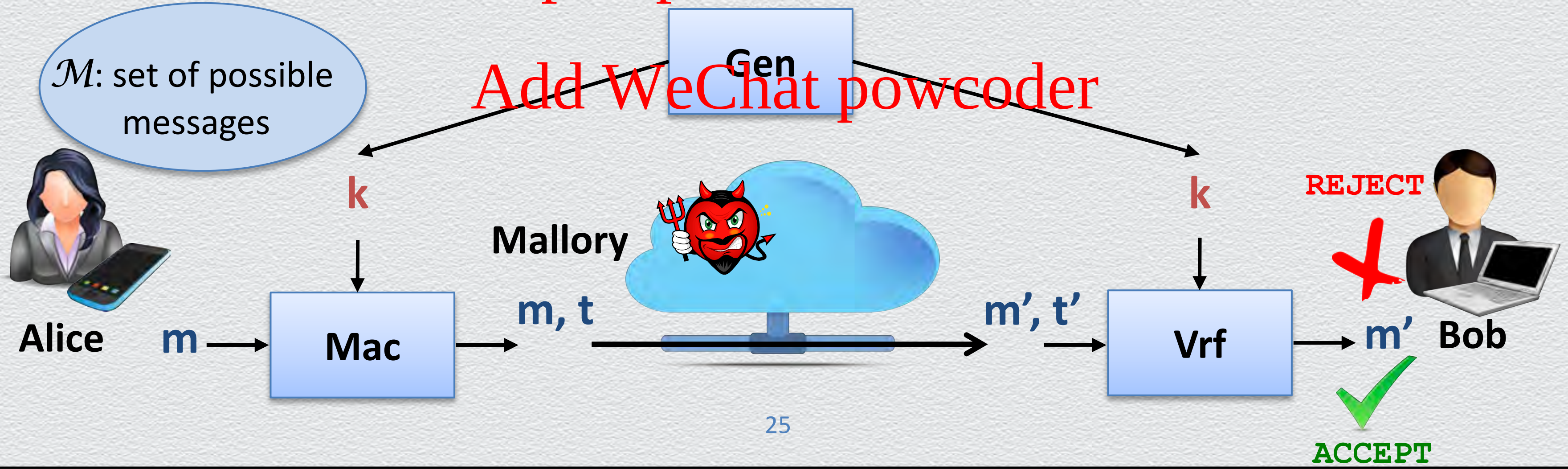
6.3 Replay attacks

Add WeChat powcoder

Recall: MAC

Abstract cryptographic primitive, **a.k.a. MAC**, defined by

- a **message space** $\mathcal{M}$; and
- a triplet of algorithms **(Gen, Mac, Vrf)**



Recall: MAC security

Attacker **wins** the game if

1. $\mathrm{Vrf}_k(m^*, t^*) = \texttt{ACCEPT}$ &
2. m^* not in $\mathcal{Q}$

MAC scheme
(Gen, Mac, Vrf)

$\mathcal{T}$

Gen $\rightarrow$ k

$\mathrm{Mac}_k(m_i) \rightarrow t_i$

m_1

t_1

m_2

t_2

...

m^*, t^*

$\mathcal{A}$ Mac(k,)

$\mathcal{Q} = m_1, m_2, \ldots$

The MAC scheme is **secure** if any PPT $\mathcal{A}$ wins the game only negligibly often.

Real-life attacker

In practice, an attacker may

- observe a traffic of authenticated (and successfully verified) messages
- manipulate (or often also partially influences) traffic
  - aims at inserting an invalid but verifiable message m^*, t^* into the traffic
    - interesting case: forged message is a new (unseen) one
    - trivial case: forged message is a previously observed one, a.k.a. a **replay attack**
- launch a **brute-force attack** (given that $Mac_k(m) \rightarrow t$ is publicly known)
  - given any observed pair m, t, exhaustively search key space to find the used key k

Threat model

In the security game, Mallory is an adversary $\mathcal{A}$ who is

- "active" (on the wire)
  - we allow $\mathcal{A}$ to **observe** and **manipulate** sent messages

- "well-informed"
  - we allow $\mathcal{A}$ to **request MAC tags** of messages of **its choice**

- "replay-attack safe"
  - we restrict $\mathcal{A}$ to **forge only new** messages

- "PPT"
  - we restrict $\mathcal{A}$ to be **computationally bounded**
  - new messages may be forged undetectably only **negligibly** often

Notes on security definition

Is it a rather strong security definition?

- we allow $\mathcal{A}$ to **query MAC tags for any message**
  - but real-world senders will authenticate only "meaningful" messages
- we allow $\mathcal{A}$ to break the scheme by **forging any new message**
  - but real-world attackers will forge only "meaningful" messages

Yes, it is the right approach…

- message "**meaningfulness" depends on higher-level application**
  - text messaging apps require authentication of English-text messages
  - other apps may require authentication of binary files
  - security definition should better be **agnostic** of the specific higher application

Notes on security definition (II)

Are replay attacks important in practice?

- absolutely yes: a **very realistic & serious threat!**
  - e.g., what if a money transfer order is “replayed”?

Yet, a “replay-attack safe” security definition is preferable

- again, whether replayed messages are valid depends on higher-lever app
- better to delegate to this app the specification of such details
  - e.g., semantics on traffic or validity checks on messages before they’re “consumed”

Eliminating replay attacks

- use of counters (i.e., common shared state) between sender & receiver
- use of timestamps along with a (relaxed) authentication window for validation

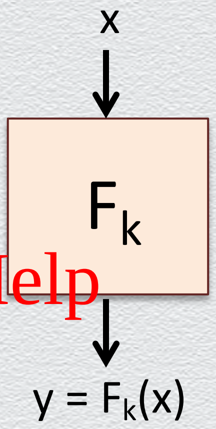
6.4 MAC constructions

Three generic MAC constructions

- fixed-length MAC
  - direct application of a PRF for tagging
  - limited applicability
- domain extension for MACs
  - straightforward secure extension of fix-length MAC
  - inefficient
- CBC-MAC
  - resembles CBC-mode encryption
  - efficient

1. Fixed-length MAC

- based on use of a PRF
  - employ a PRF F_k in the obvious way to compute and canonically verify tags
  - set tag t to be the pseudorandom string derived by evaluating F_k on message m
- secure, provided that F_k is a secure PRF



MAC scheme Π

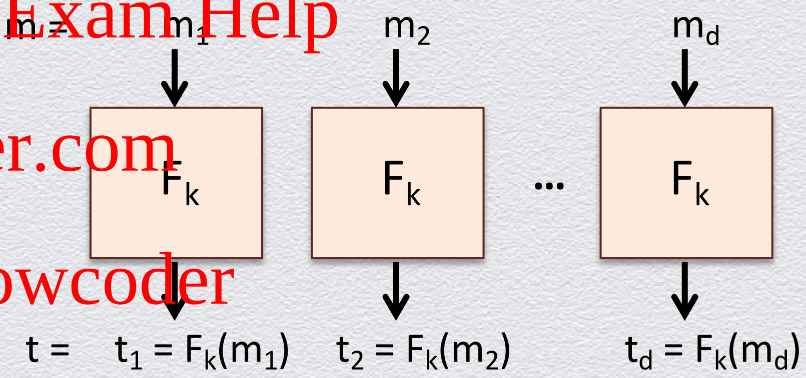
$Gen(1^n)$: $\{0,1\}^n \to k$

$Mac_k(m)$: set $t = F_k(m)$

$Vrfy_k(m,t)$: return 1 iff $t = F_k(m)$

2. Domain extension for MACs (I)

- suppose we have the previous fix-length MAC scheme
- how can we authenticate a message m of arbitrary length?
- naïve approach
  - pad m and view it as d blocks $m_1, m_2, \ldots, m_d$
  - separately apply MAC to block m_i



- security issues
  - reordering attack; verify block index, $t = F_k(m_i||i)$
  - truncation attack; verify message length $\delta = |m|$, $t = F_k(m_i||i||\delta)$
  - mix-and-match attack; randomize tags (using message-specific fresh nonce)

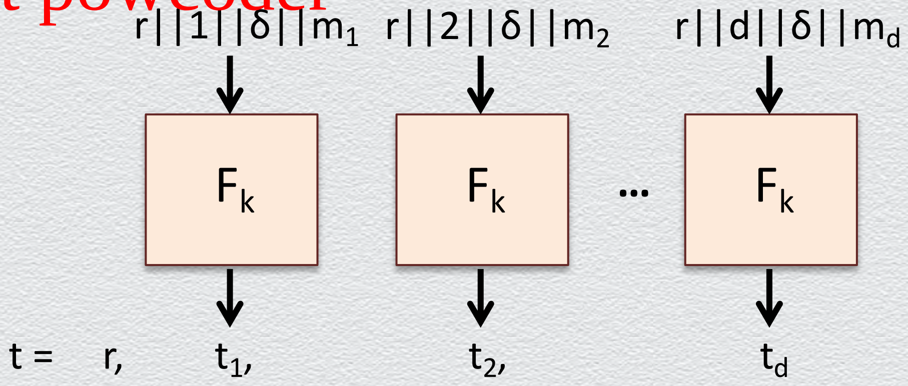
2. Domain extension for MACs (II)

Final scheme

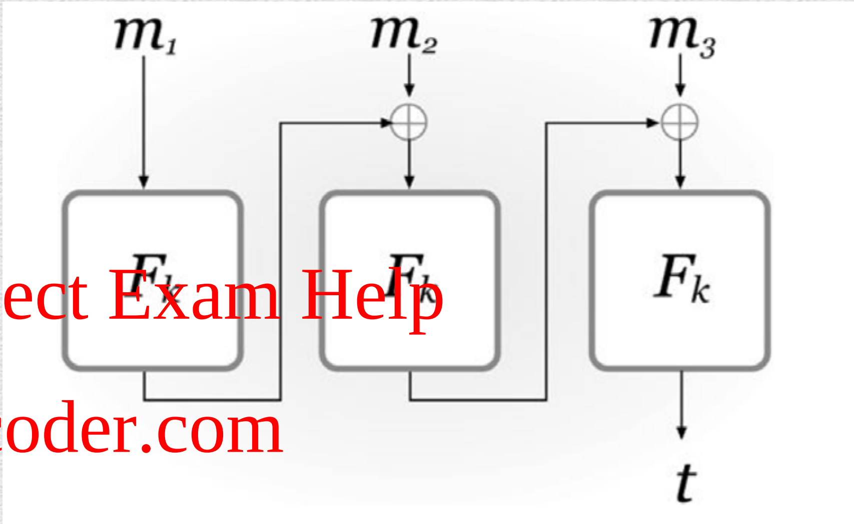
- assumes a secure MAC scheme for messages of size n
- set tag of message m of size δ at most $2^{n/4}$ as follows
  - choose fresh random nonce r of size n/4; view m as d blocks of size n/4 each
  - separately apply MAC on each block, authenticating also its index, δ and nonce r

Security

- extension is secure, if F_k is a secure PRF



3. CBC-MAC



Idea

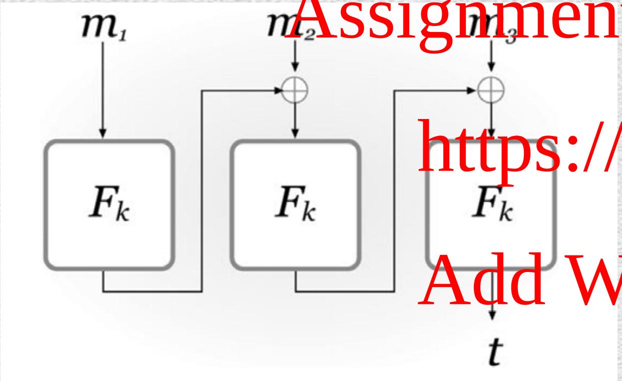
- employ a PRF in a manner similar to CBC-mode encryption

Security

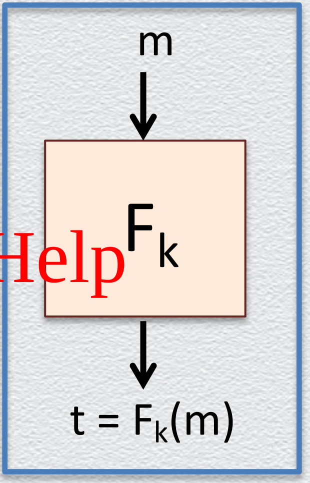
- extension is secure, if
  - F_k is a secure PRF; and
  - only **fixed-length** messages are authenticated
- messages of length equal to any multiple of n can be authenticated
  - but this length need be fixed in advance
  - insecure, otherwise

3. CBC-MAC Vs. previous schemes

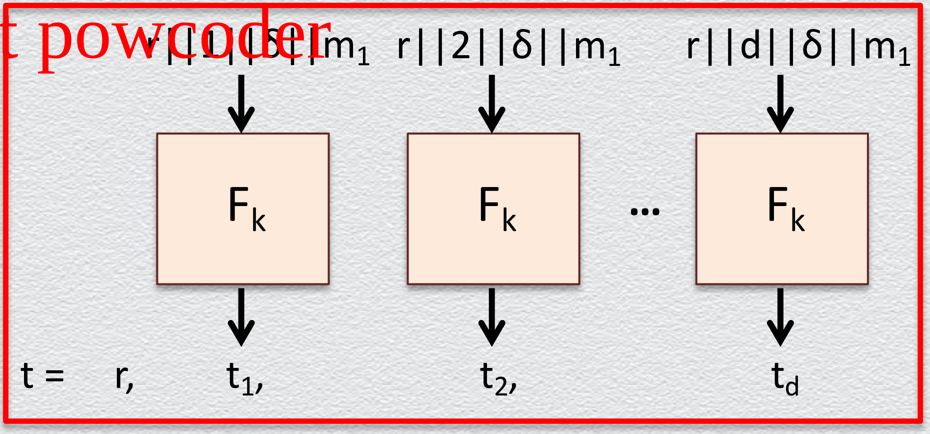
- can authenticate longer messages than basic PRF-based scheme (1)



Scheme (1)



Scheme (2)

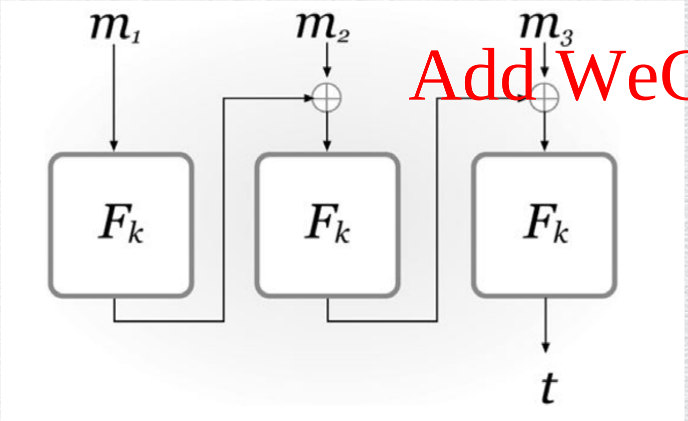


- more efficient than domain-extension MAC scheme (2)

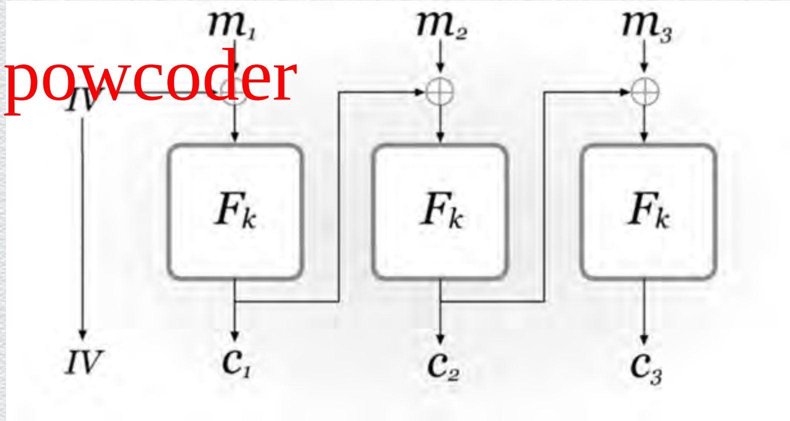
3. CBC-MAC Vs. CBC-mode encryption

- crucially for their security
  - CBC-MAC uses **no IV** (or uses an IV set to 0) and only the **last PRF output**
  - CBC-mode encryption uses a **random IV** and **all PRF outputs**
  - “simple”, innocent modification can be catastrophic…

CBC-MAC



CBC-mode encryption



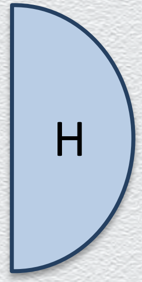
6.5 Hash functions

Cryptographic hash functions

Basic cryptographic primitive

- maps **"objects"** to a **fixed-length** binary strings
- core security property: mapping **avoids collisions**
  - **collision**: distinct objects (x ≠ y) are mapped to the same hash value (H(x) = H(y))
  - although collisions **necessarily exist**, they are **infeasible to find**

input
arbitrarily long string



output
short digest, fingerprint, "secure" description

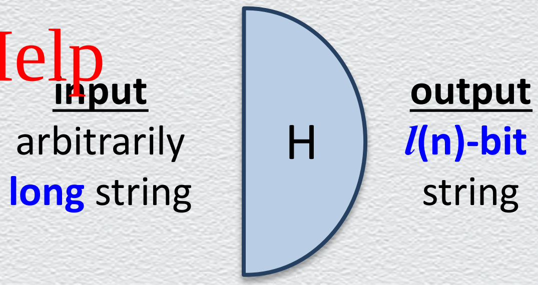
Important role in modern cryptography

- lie between symmetric- and asymmetric-key cryptography
- capture different security properties of "idealized random functions"
- qualitative stronger assumption than PRF

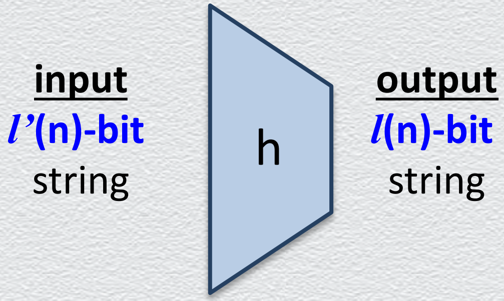
Hash & compression functions

Map messages to short digests

- a **general** hash function H() maps
  - a message of an **arbitrary length** to a ***l*(n)-bit** string

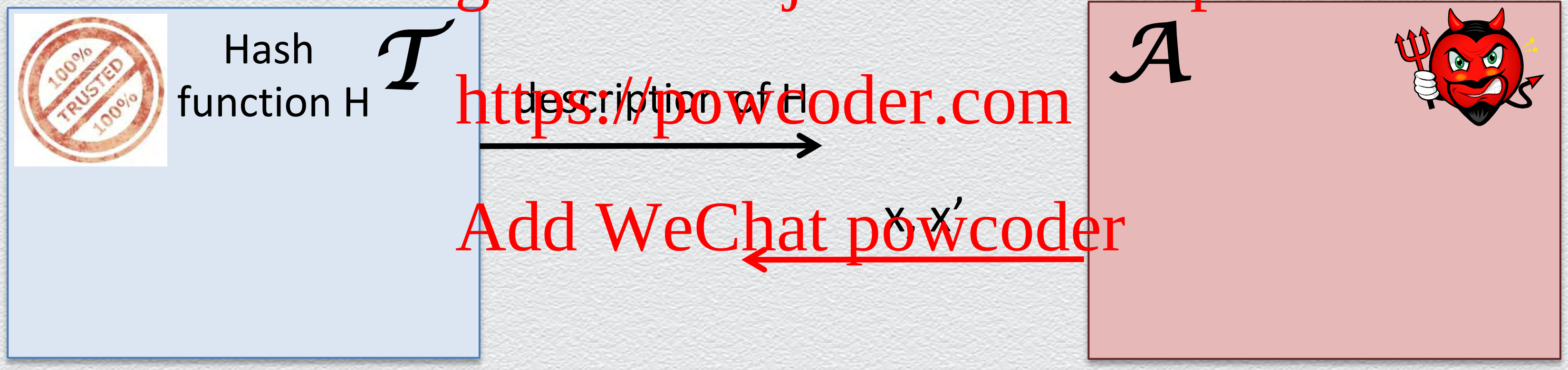


- a **compression** (hash) function h() maps
  - a **long** binary string to a **shorter** binary string
  - an ***l'*(n)-bit string** to a ***l*(n)-bit string**, with ***l'*(n) > *l*(n)**



Collision resistance (CR)

Attacker wins the game if $x \neq x'$ & $H(x) = H(x')$



H is collision-resistant if any PPT $\mathcal{A}$ wins the game only negligibly often.

Weaker security notions

Given a hash function $H: X \to Y$, then we say that H is

- **preimage resistant** (or **one-way**)
  - if given $y \in Y$, finding a value $x \in X$ s.t. $H(x) = y$ happens negligibly often
- **2-nd preimage resistant** (or **weak collision resistant**)
  - if given a **uniform** $x \in X$, finding a value $x' \in X$, s.t. $x' \neq x$ and $H(x') = H(x)$ happens negligibly often
- cf. **collision resistant** (or **strong collision resistant**)
  - if finding two distinct values $x', x \in X$, s.t. $H(x') = H(x)$ happens negligibly often

6.6 Design framework

Domain extension via the Merkle-Damgård transform

General design pattern for cryptographic hash functions

- reduces CR of general hash functions to CR of compression functions

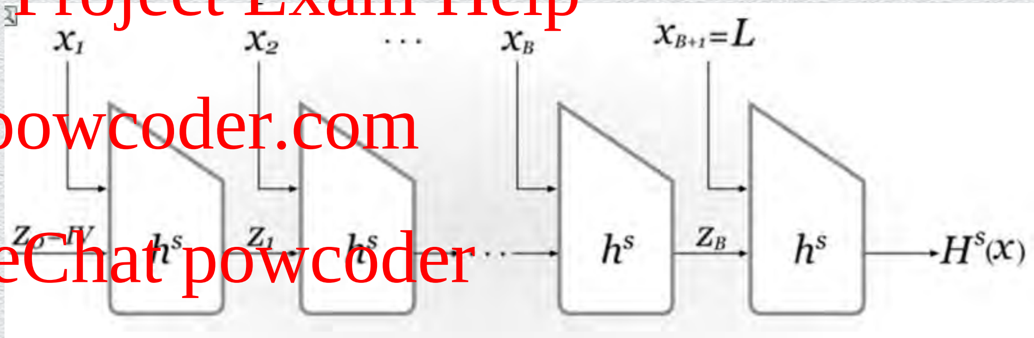
- thus, in practice, it suffices to realize a collision-resistant compression function h
- compressing by 1 single bit is a least as hard as compressing by any number of bits!

Merkle-Damgård transform: Design

Suppose that h: $\{0,1\}^{2n} \longrightarrow \{0,1\}^n$ is a collision-resistant compression function

Consider the general hash function H: $\mathcal{M} = \{x : |x| < 2^n\} \rightarrow \{0,1\}^n$, defined as

<u>Merkle-Damgård design</u>

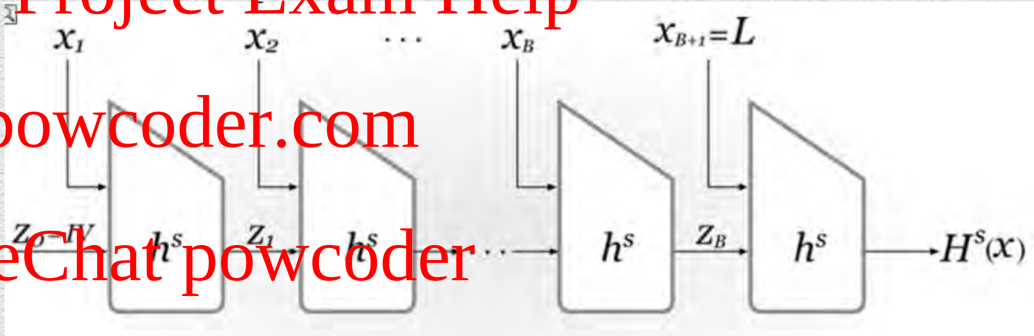


- H(x) is computed by applying h() in a **"chained" manner** over n-bit message blocks

  - pad x to define a number, say B, **message blocks $x_1, \ldots, x_B$**, with $|x_i| = n$
  - set extra, final, message block **x_{B+1} as an n-bit encoding L of $|x|$**
  - starting by initial digest **$z_0 = IV = 0^n$**, output **$H(x) = z_{B+1}$**, where **$z_i = h^s(z_{i-1} || x_i)$**

Merkle-Damgård transform: Security

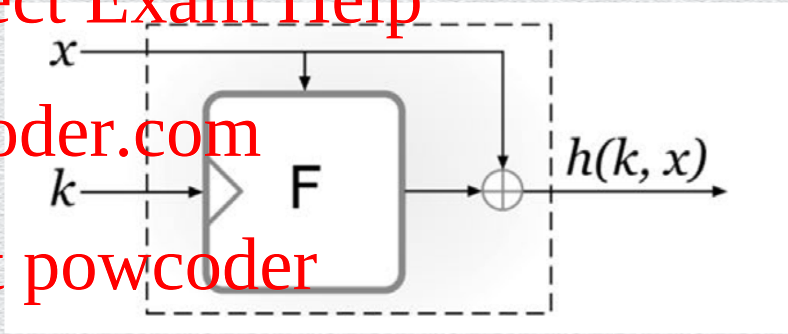
If the compression function h is CR,
then the derived hash function H is also CR!



Compression function design: The Davies-Meyer scheme

Employs PRF w/ key length m & block length n

- define h: $\{0,1\}^{n+m} \rightarrow \{0,1\}^n$ as **$h(x||k) = F_k(x)$ XOR x**



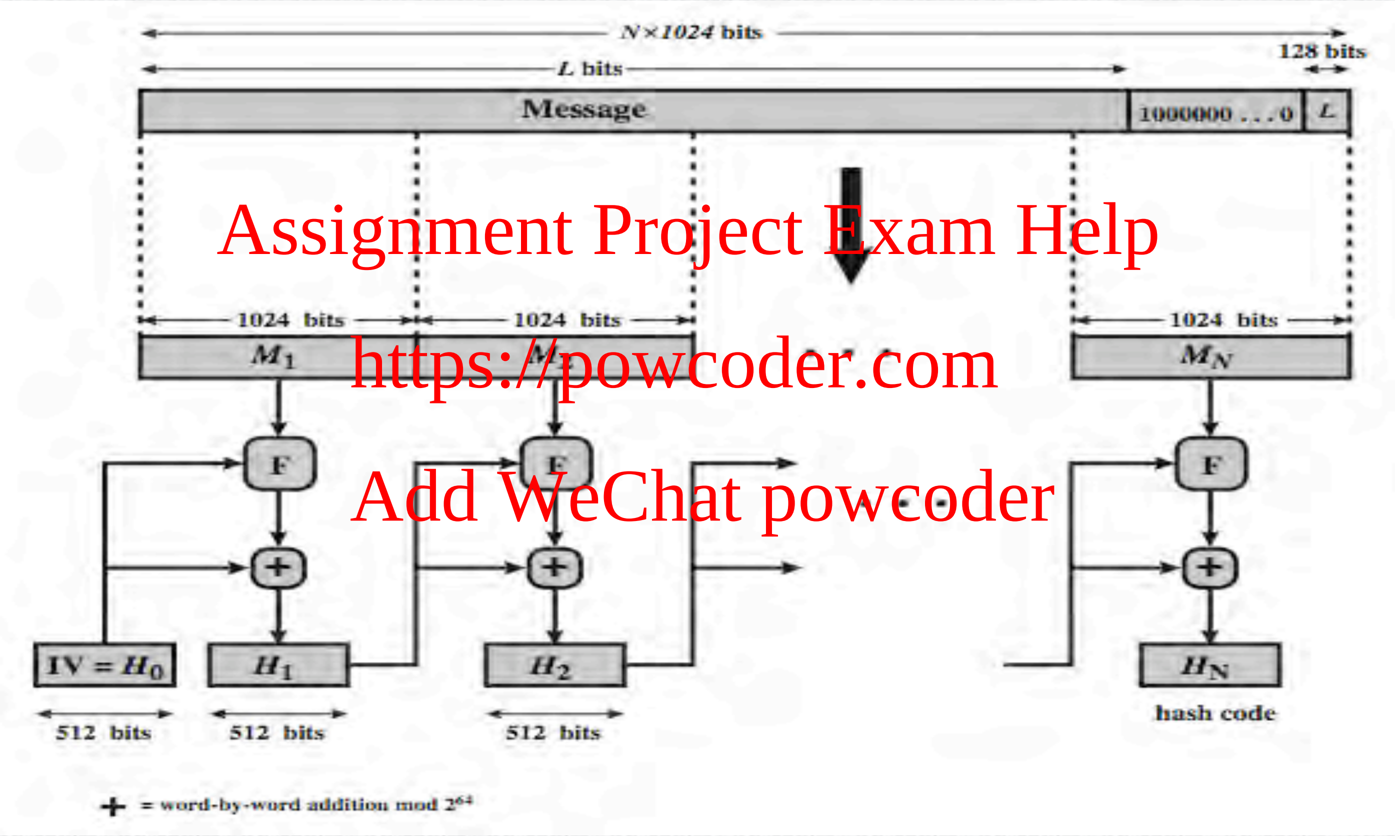
Security

- h is CR, if F is an **ideal cipher**

Well known hash functions

- MD5 (designed in 1991)
  - output 128 bits, collision resistance **completely broken** by researchers in 2004
  - today (controlled) collisions can be found in less than a minute on a desktop PC
- SHA1 – the Secure Hash Algorithm (series of algorithms standardized by NIST)
  - output 160 bits, considered **insecure** for collision resistance
  - **broken** in 2017 by researchers at CWI
- SHA2 (SHA-224, SHA-256, SHA-384, SHA-512)
  - outputs 224, 256, 384, and 512 bits, respectively, **no real security concerns yet**
  - based on Merkle-Damgård + Davies-Meyer generic transforms
- SHA3 (Kessac)
  - **completely new philosophy** (sponge construction + unkeyed permutations)

SHA-2-512 overview



Current hash standards

Algorithm	Maximum Message Size (bits)	Block Size (bits)	Rounds	Message Digest Size (bits)
MD5	2^{64}	512	64	128
SHA-1	2^{64}	512	80	160
SHA-2-224	2^{64}	512	64	224
SHA-2-256	2^{64}	512	64	256
SHA-2-384	2^{128}	1024	80	384
SHA-2-512	2^{128}	1024	80	512
SHA-3-256	unlimited	1088	24	256
SHA-3-512	unlimited	576	24	512