

Message Authentication Code

Prepared by Dr. Ayman Khallel

IT373

Information Security

Lecture 8



Course content

1. Overview of security concepts and definitions
2. Symmetric cryptography; modern encryption algorithms (DES, AES)
3. Modern data encryption standards
4. Block cipher cryptanalysis and usage
5. Modern stream ciphers
6. Asymmetric cryptography; public key cryptography,
7. Cryptographic Hash Functions
8. Message authentication Code
9. Digital signature and authentication protocols
10. Electronic mail security, IP security, web security.

Agenda

- Network Communication Attacks: Types and Overview
- Message Authentication Functions
- Message Encryption
- Public key cryptography
- Cryptographic hash functions, MAC
- HMAC, CMAC, GCM, and CCM
- Pseudorandom number generation



Network Communication Attacks: Types and Overview

1. Disclosure

Release of message contents to any person or process not possessing the appropriate cryptographic key

2. Traffic analysis

Discovery of the pattern of traffic between parties

3. Masquerade

Insertion of messages into the network from a fraudulent source

4. Content modification

Changes to the contents of a message, including insertion, deletion, transposition, and modification

5. Sequence modification

- Any modification to a sequence of messages between parties, including insertion, deletion, and reordering

6. Timing modification

- Delay or replay of messages

7. Source repudiation

- Denial of transmission of message by source

8. Destination repudiation

- Denial of receipt of message by destination

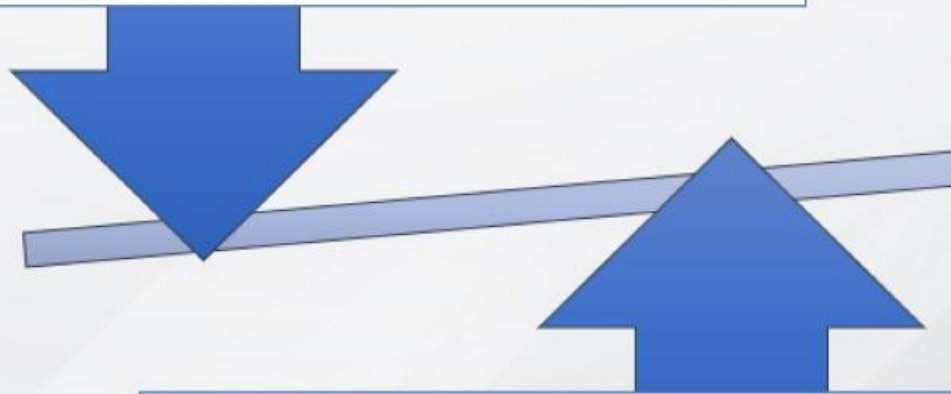


Message Authentication Functions

- Two levels of functionality:

Lower level

- There must be some sort of function that produces an authenticator

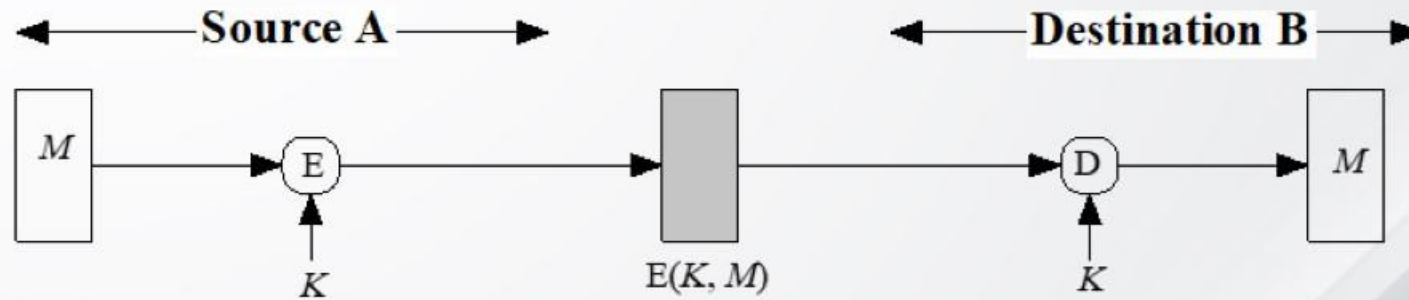


Higher-level

- Uses the lower-level function as a primitive in an authentication protocol that enables a receiver to verify the authenticity of a message

- Hash function
 - A function that maps a message of any length into a fixed-length hash value which serves as the authenticator
- Message encryption
 - The ciphertext of the entire message serves as its authenticator
- Message authentication code (MAC)
 - A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

Message encryption



(a) Symmetric encryption: confidentiality and authentication

- **Confidentiality:**
 - Message M is encrypted using a secret key K shared by sender A and receiver B.
 - Only A and B know the key K , ensuring that no third party can decrypt the message.
- **Authentication:**
 - Since only A and B share the secret key K , B can be assured that the message came from A.
 - If the ciphertext is valid, B knows that none of the bits have been altered.
- **Challenges with Authentication:**
 - Without additional measures, B may accept any random bit pattern as valid plaintext.
 - To avoid this, only a subset of bit patterns is considered legitimate plaintext.
 - Example: If only one bit pattern in 10^6 is valid, the chance of random ciphertext producing valid plaintext is 10^{-6} .

Message encryption

- **Example**
 - **Caesar cipher with a shift of one ($K = 1$)**
 - **Legitimate Ciphertext:**
 - Nbsftfbupbutboeepftfbupbutboemjuumfmbnctfbujwz
 - **decrypts to:**
 - **mareseat~~oats~~anddoeseat~~oats~~andlittlelambse~~at~~ivy**
 - Mean: “Mares eat oats, and does eat oats, and little lambs eat ivy”
 - **Random Ciphertext:**
 - ZuvrsoevgqxIzwigamdvnmhpmccxiuureosfbcebtqxsxq
 - **decrypts to**
 - YtuqrndufpwkyvhfzlcumlgoIbbwhttqdnreabdasppwrwp
 - **Ensuring Message Authenticity in Symmetric Encryption**
 - **Challenge in Authenticating Ciphertext:**
 - Determining if incoming ciphertext decrypts to intelligible plaintext can be difficult.
 - Issues arise when the plaintext is non-textual, such as binary object files or digitized X-rays.
 - An opponent could disrupt communication by sending random content as though it were from a legitimate source.

Solution: Error-Detecting Code (FCS/Checksum):

- **Appended to Plaintext:**
 - A function F generates an **Error-Detecting Code (FCS)** for the plaintext message M .
 - The FCS is appended to M before encryption.
- **At the Destination:**
 - After decryption, B verifies the FCS by applying the same function F .
 - If the calculated FCS matches the incoming FCS, the message is considered authentic.
- **Why It Works:**
 - It's highly unlikely that a random sequence of bits would result in the correct FCS, ensuring message authenticity.

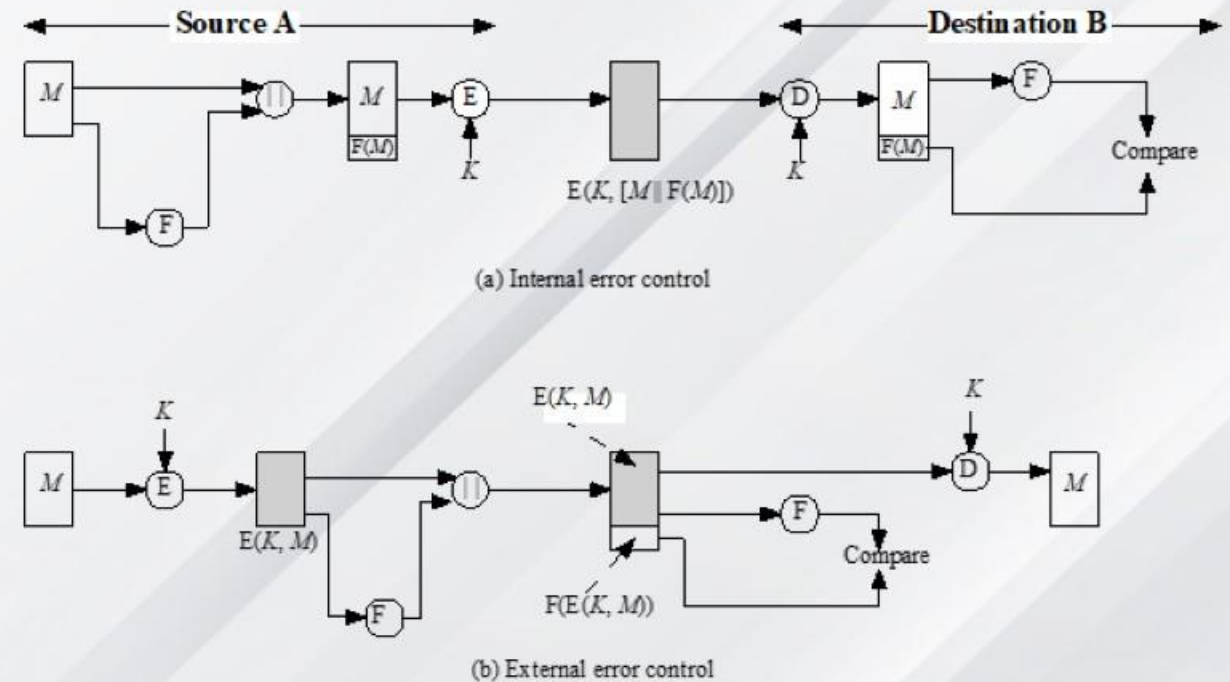


Figure 12.2 Internal and External Error Control

Strengthening Authentication with Structured Communication

•Error-Control Code and Structuring:

- An **error-control code** is one example of added structure to ensure message authenticity.
- Any structure added to the transmitted message strengthens authentication capabilities.

•Layered Protocols in TCP/IP:

- The **TCP/IP protocol architecture** provides inherent structure to transmitted messages.
- The format of a **TCP segment** includes a header, which contains useful information such as the **sequence number** and a **checksum**.

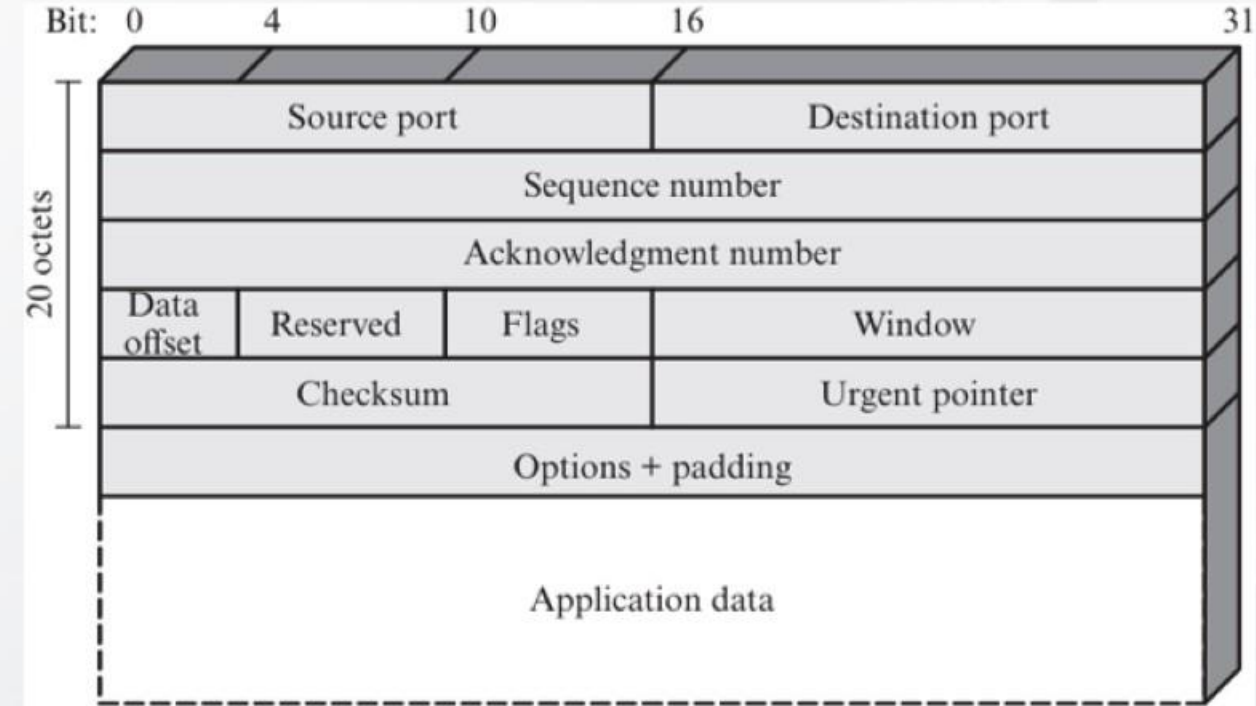


Figure 12.3 TCP Segment

Figure 12.3 TCP Segment

Public-Key Encryption

- The straightforward use of public-key encryption provides confidentiality but not authentication
- To provide both confidentiality and authentication, A can encrypt M first using its private key which provides the digital signature, and then using B's public key, which provides confidentiality
- Disadvantage is that the public-key algorithm must be exercised four times rather than two in each communication

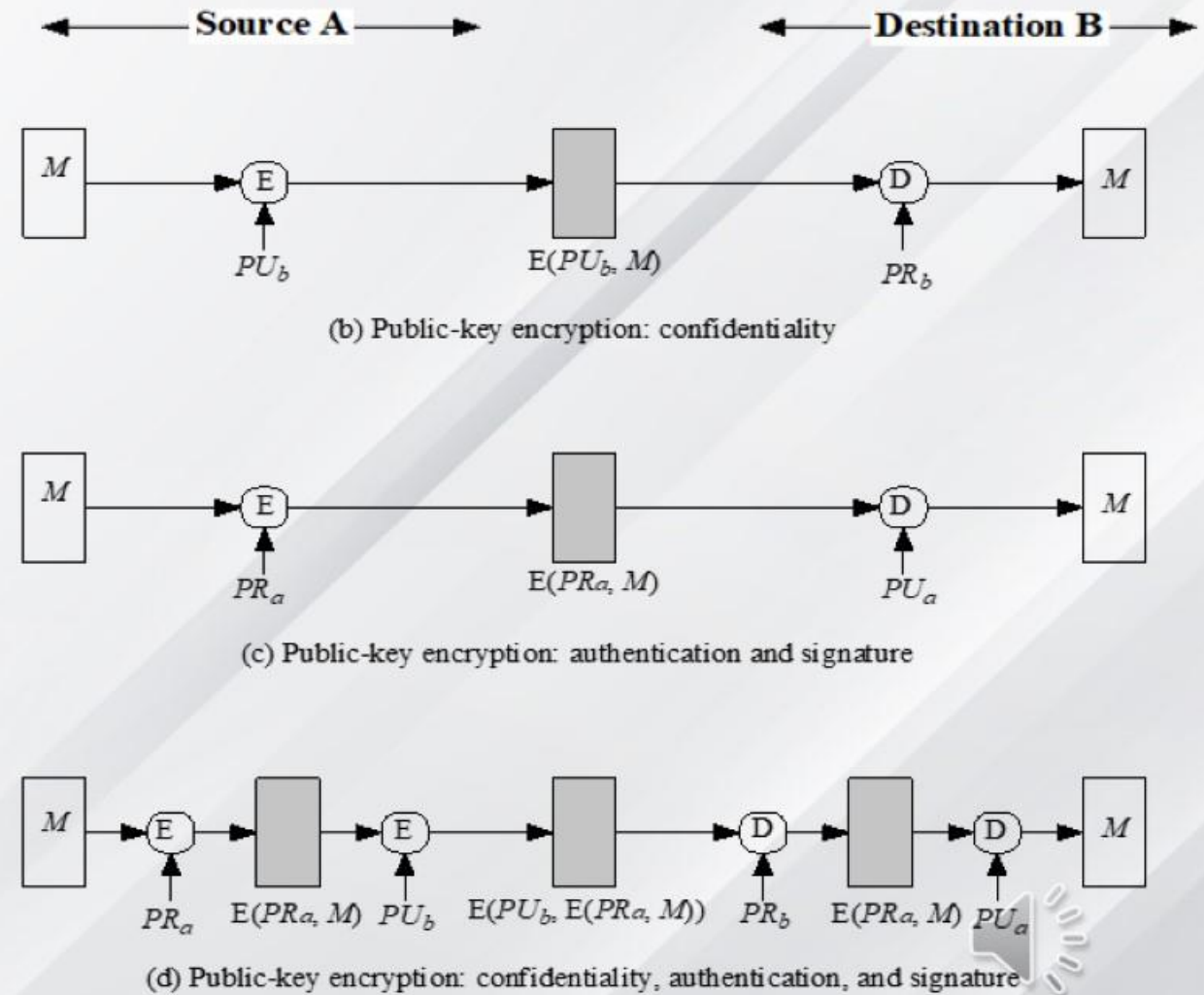


Figure 12.1 Basic Uses of Message Encryption

Message Authentication Code (MAC)

- Authenticate messages using a cryptographic checksum (MAC).
- Process:**
 - Sender (A) and Receiver (B) share a **secret key (K)**.
 - MAC Calculation:**
 - Sender computes: **MAC = C(K, M)**
 - M**: Input message
 - C**: MAC function
 - K**: Shared secret key
 - Sender transmits **M + MAC** to Receiver.
 - Validation by Receiver**
 - Receiver uses **K** to compute MAC for received message (**M'**) and compares it with the received **MAC**.

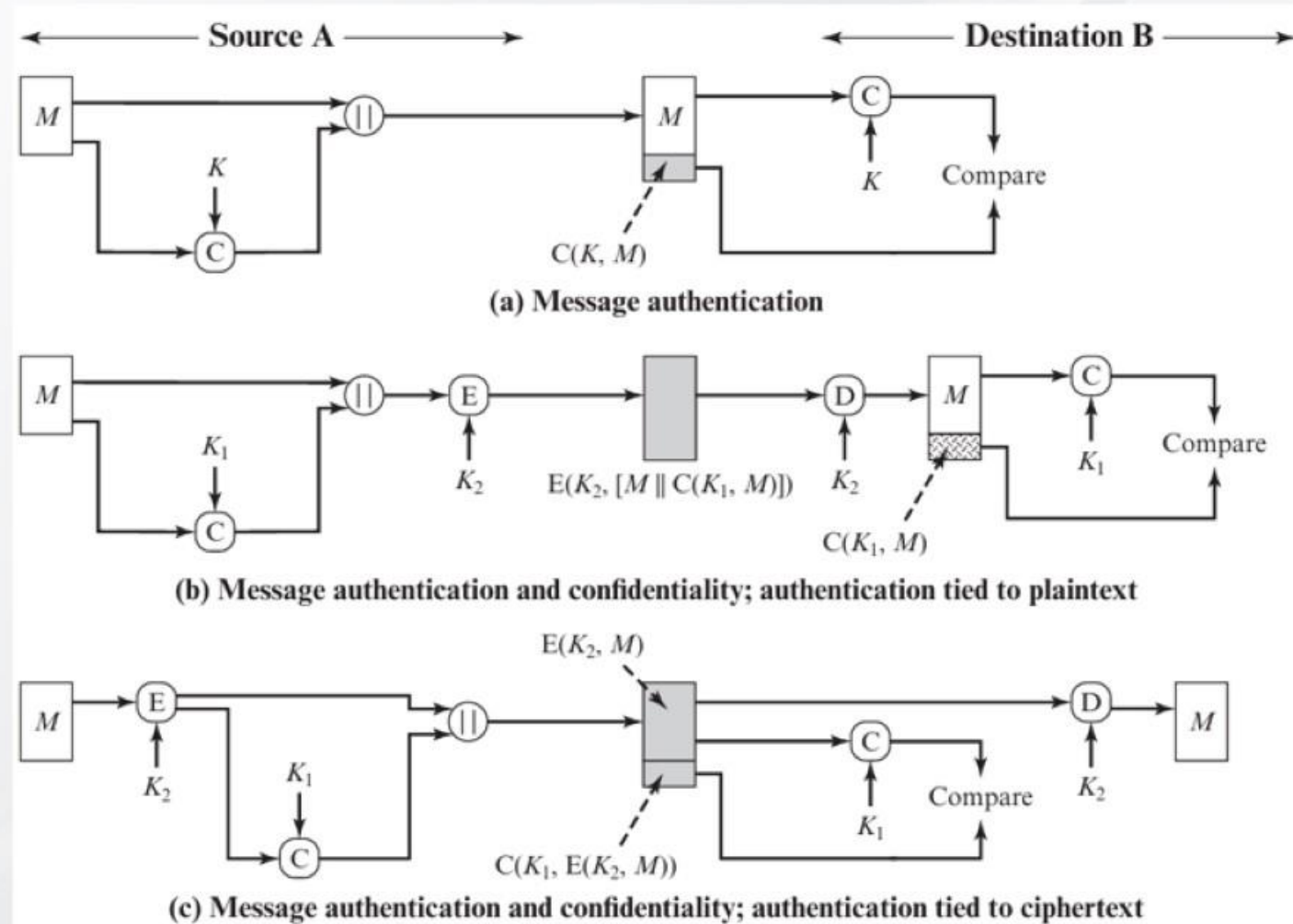


Figure 12.4 Basic Uses of Message Authentication code (MAC)

Why Use Message Authentication Code (MAC) Instead of Symmetric Encryption?

1. Broadcast Authentication:

- Message sent to multiple destinations in plaintext with MAC.
- A single system with the secret key verifies authenticity, alerting others if compromised.

2. Selective Authentication:

- In high-load systems, only selected messages are authenticated, saving decryption time.

3. Program Integrity Verification:

- Attach MAC to programs to validate integrity without repeatedly decrypting them.

Why Use Message Authentication Code (MAC) Instead of Symmetric Encryption?

4. **Non-Confidential Authentication:**

- Example: SNMPv3 prioritizes authentication without requiring confidentiality.

5. **Architectural Flexibility:**

- Authentication at the application level; confidentiality at a lower layer (e.g., transport layer).

6. **Extended Protection:**

- MAC protects messages beyond decryption, ensuring integrity even after transit.

Requirements for MACs

Taking into account the types of attacks, the MAC needs to satisfy the following:

The first requirement deals with message replacement attacks, in which an opponent is able to construct a new message to match a given MAC, even though the opponent does not know and does not learn the key

The second requirement deals with the need to thwart a brute-force attack based on chosen plaintext

The final requirement dictates that the authentication algorithm should not be weaker with respect to certain parts or bits of the message than others



Brute-Force Attack

- Requires known message-tag pairs
 - A brute-force method of finding a collision is to pick a random bit string y and check if $H(y) = H(x)$

Two lines of attack:

- Attack the key space
 - If an attacker can determine the MAC key then it is possible to generate a valid MAC value for any input x
- Attack the MAC value
 - Objective is to generate a valid tag for a given message or to find a message that matches a given tag



Cryptanalysis

- Cryptanalytic attacks seek to exploit some property of the algorithm to perform some attack other than an exhaustive search
- An ideal MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort
- There is much more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs

MACs Based on Hash Functions: HMAC

- There has been increased interest in developing a MAC derived from a cryptographic hash function
- Motivations:
 - Cryptographic hash functions such as MD5 and SHA generally execute faster in software than symmetric block ciphers such as DES
 - Library code for cryptographic hash functions is widely available
- HMAC has been chosen as the mandatory-to-implement MAC for IP security
- Has also been issued as a NIST standard (FIPS 198)

HMAC Design Objectives

RFC 2104 lists the following objectives for HMAC:

To use, without modifications, available hash functions

To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required

To preserve the original performance of the hash function without incurring a significant degradation

To use and handle keys in a simple way

To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function



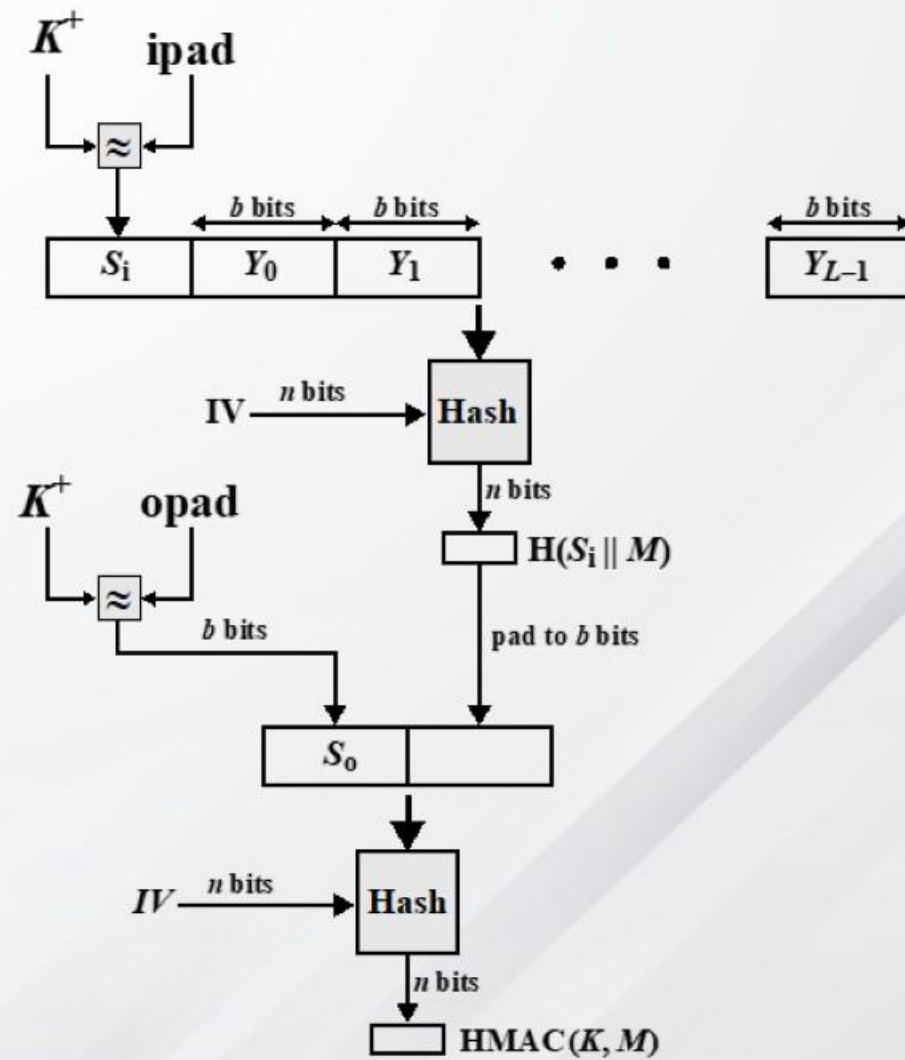


Figure 12.5 HMAC Structure

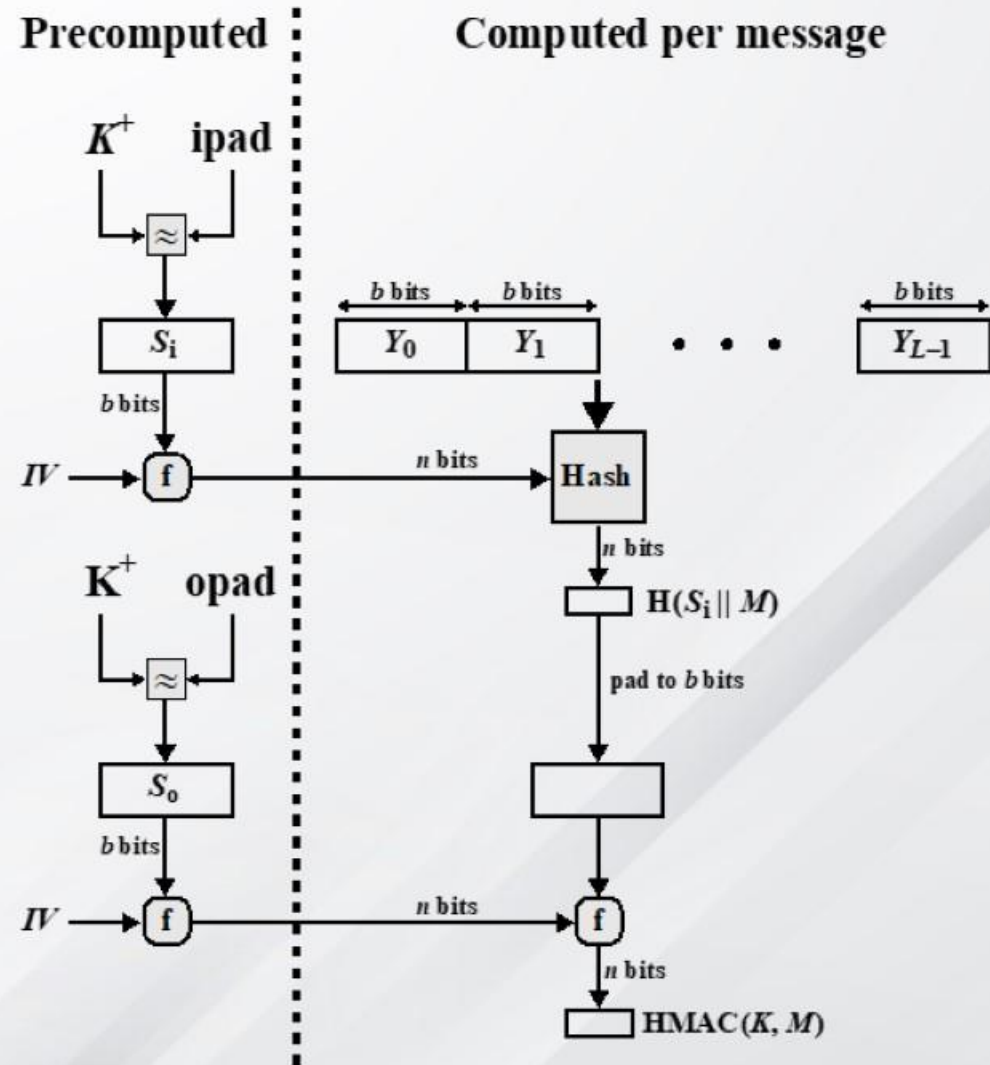


Figure 12.6 Efficient Implementation of HMAC

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



Data Authentication Code (DAC)

- Widely used MAC for many years.
- Replaced by newer and stronger algorithms due to discovered security weaknesses.
- **How DAA Works:**
 - Data is divided into **64-bit blocks** (e.g., $D_1, D_2, \dots, D_N$).
 - Final block padded with zeros if needed to form a full 64-bit block.
- **Calculation of DAC:**
 - DES encryption algorithm E with secret key K is used.
 - **DAC** is calculated using this process.

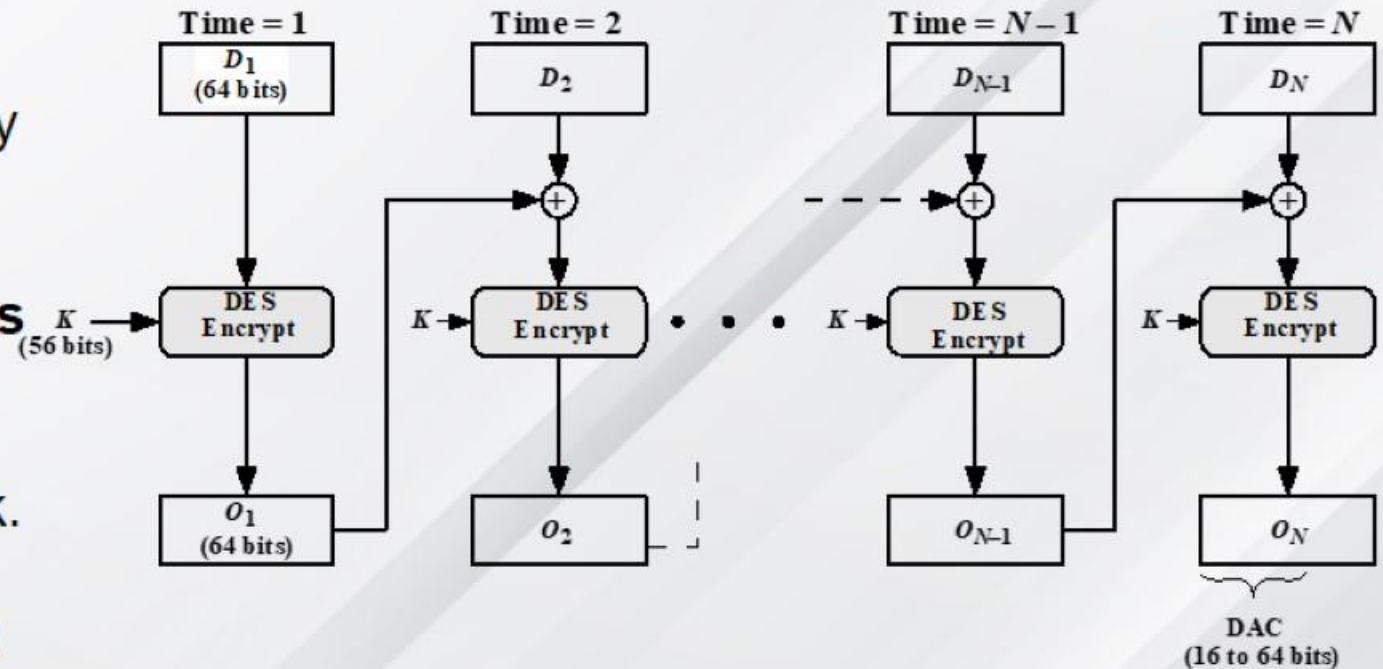
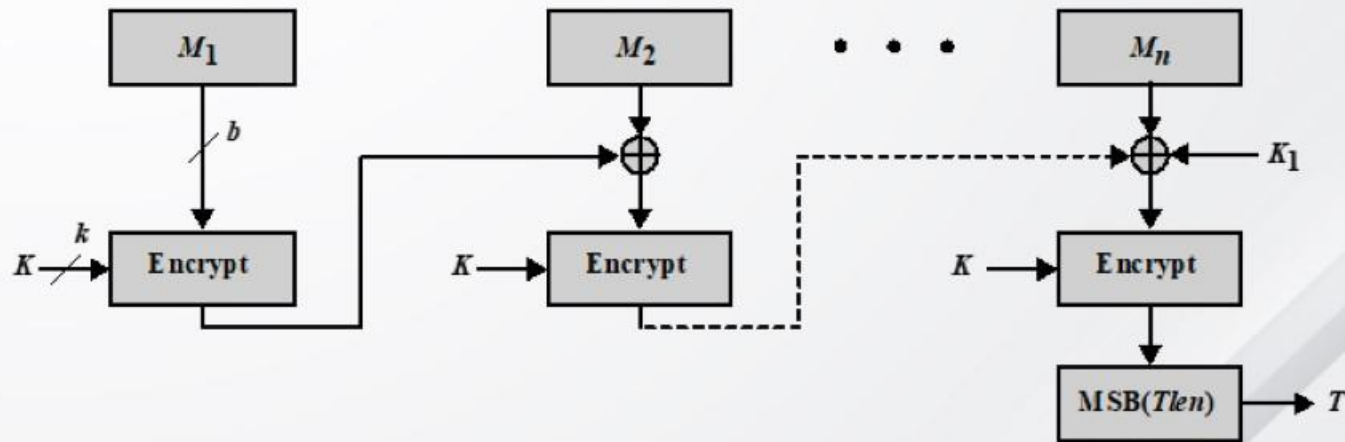
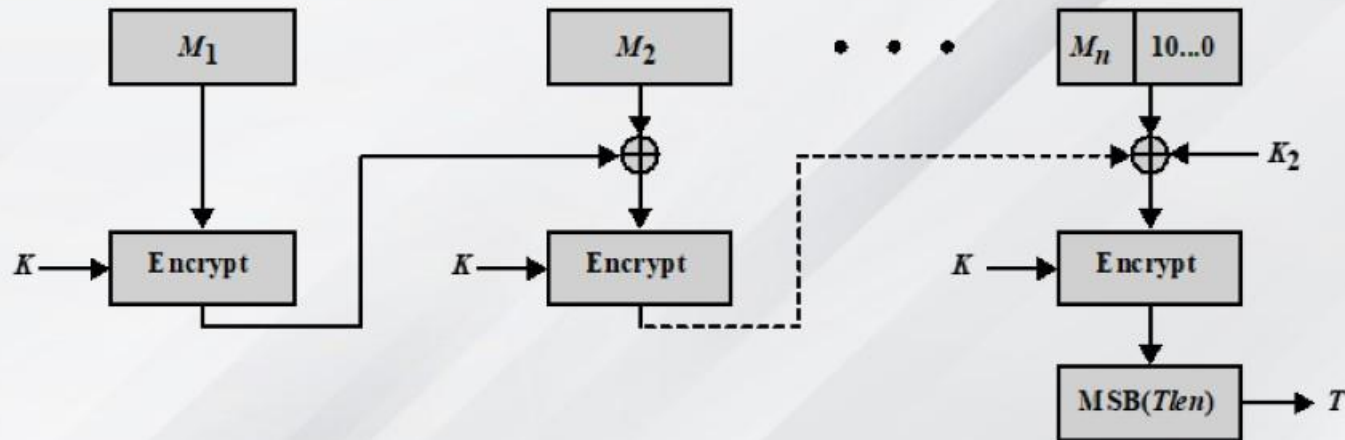


Figure 12.7 Data Authentication Algorithm (FIPS PUB 113)

Cipher-based Message Authentication Code (CMAC)



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

Figure 12.8 Cipher-Based Message Authentication Code (CMAC)

Authenticated Encryption (AE)

- A term used to describe encryption systems that simultaneously protect confidentiality and authenticity of communications
- Approaches:
 - Hashing followed by encryption
 - Authentication followed by encryption
 - Encryption followed by authentication
 - Independently encrypt and authenticate
- Both decryption and verification are straightforward for each approach
- There are security vulnerabilities with all of these approaches

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
 - 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:

Data that will be both authenticated and encrypted

This is the plaintext message P of the data block

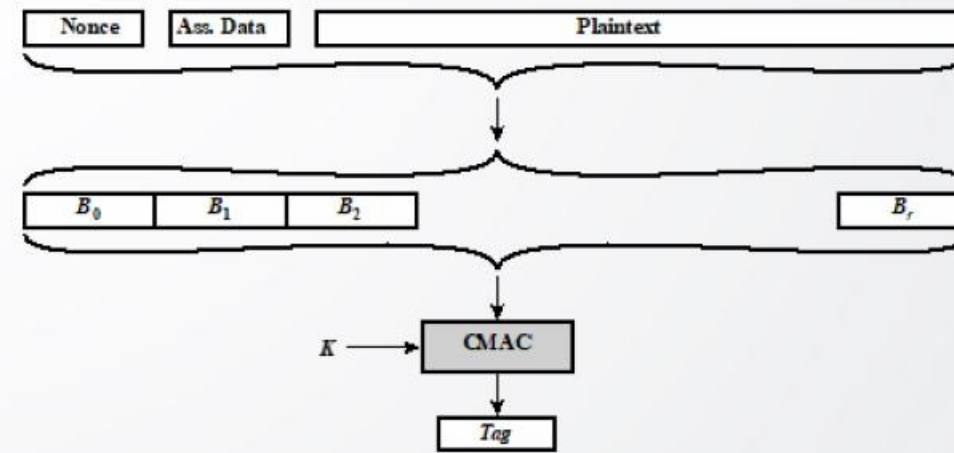
Associated data A that will be authenticated but not encrypted

An example is a protocol header that must be transmitted in the clear for proper protocol operation but which needs to be authenticated

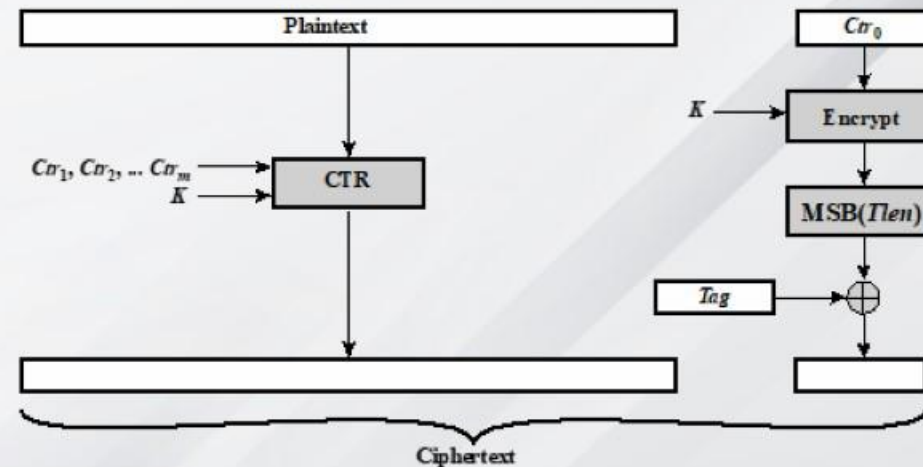
A nonce N that is assigned to the payload and the associated data

This is a unique value that is different for every instance during the lifetime of a protocol association and is intended to prevent replay attacks and certain other types of attacks





(a) Authentication

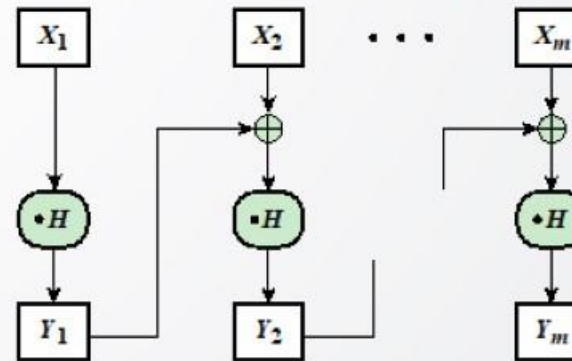


(b) Encryption

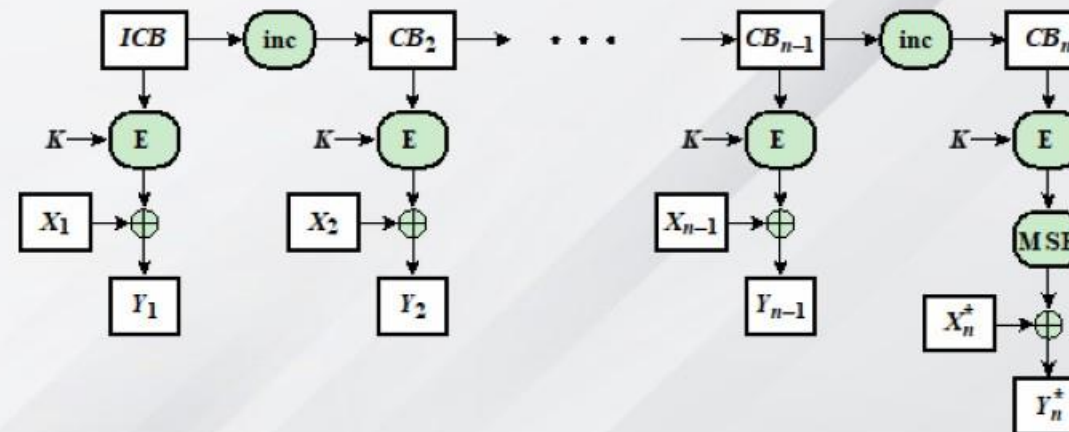
Figure 12.9 Counter with Cipher Block Chaining-Message Authentication Code (CCM)

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

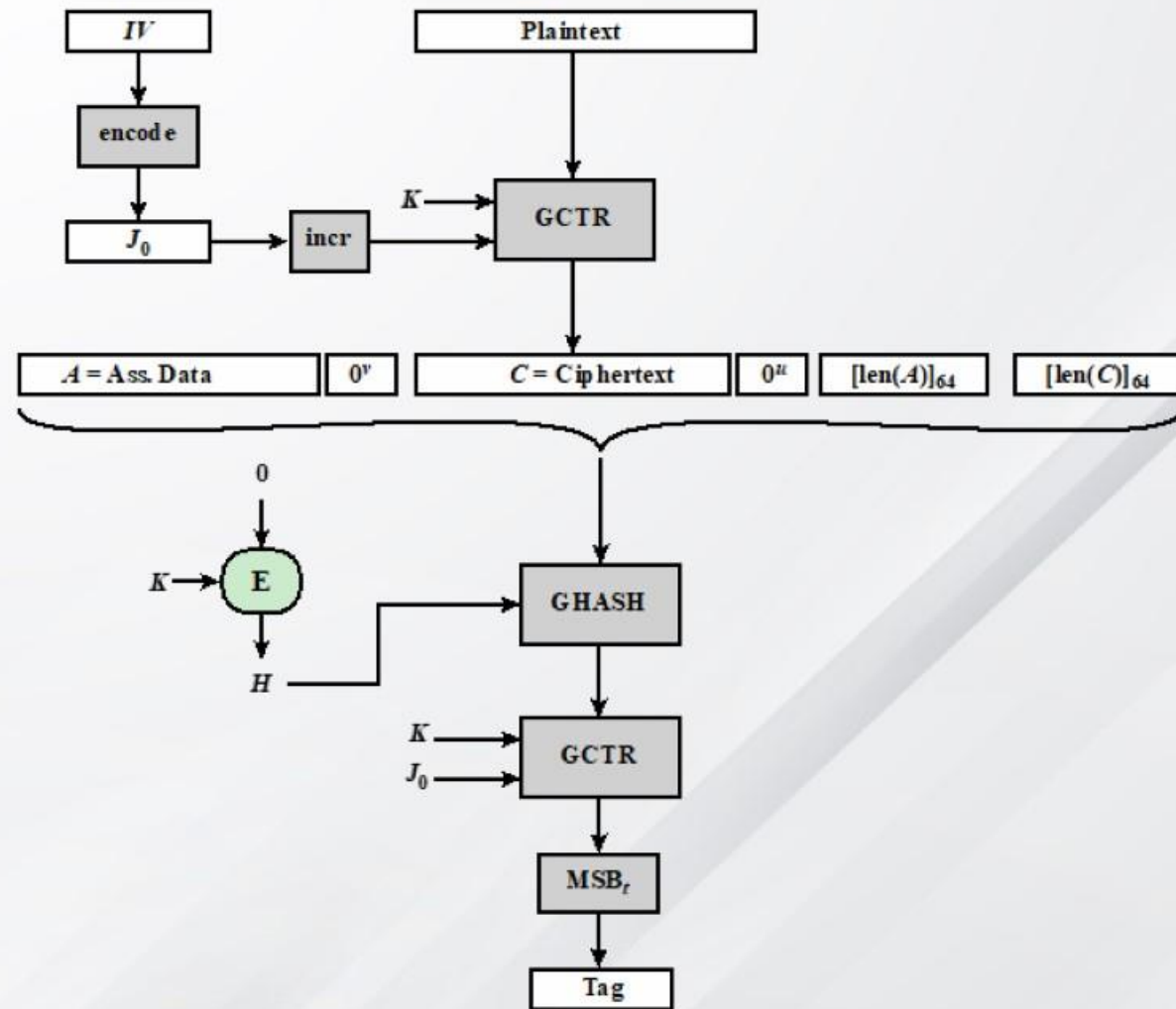
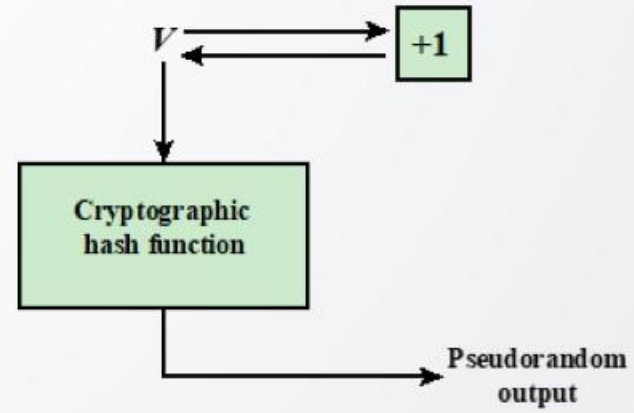


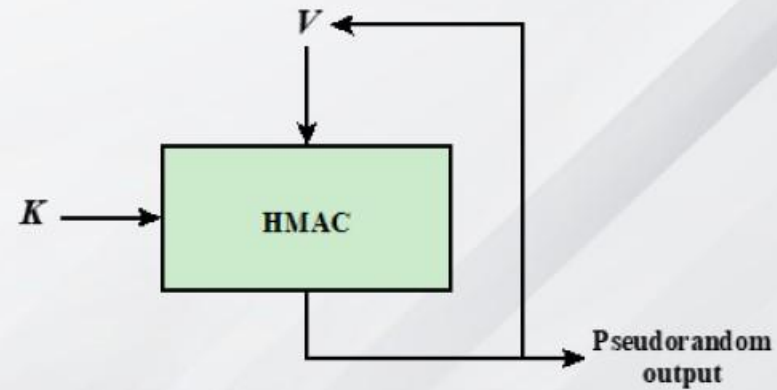
Figure 12.11 Galois Counter - Message Authentication Code (GCM)

Pseudorandom Number Generation Using Hash Functions and MACs

- Essential elements of any pseudorandom number generator (PRNG) are a seed value and a deterministic algorithm for generating a stream of pseudorandom bits
 - If the algorithm is used as a pseudorandom function (PRF) to produce a required value, the seed should only be known to the user of the PRF
 - If the algorithm is used to produce a stream encryption function, the seed has the role of a secret key that must be known to the sender and the receiver
- A hash function or MAC produces apparently random output and can be used to build a PRNG



(a) PRNG using cryptographic hash function



(b) PRNG using HMAC

Figure 12.14 Basic Structure of Hash-Based PRNGs (SP 800-90)

$m = \lceil n/\text{outlen} \rceil$ $w_0 = V$ $W = \text{the null string}$ For $i = 1$ to m $w_i = \text{MAC}(K, w_{i-1})$ $W = W \parallel w_i$ Return leftmost n bits of W	$m = \lceil n/\text{outlen} \rceil$ $W = \text{the null string}$ For $i = 1$ to m $w_i = \text{MAC}(K, (V \parallel i))$ $W = W \parallel w_i$ Return leftmost n bits of W	$m = \lceil n/\text{outlen} \rceil$ $A(0) = V$ $W = \text{the null string}$ For $i = 1$ to m $A(i) = \text{MAC}(K, A(i-1))$ $w_i = \text{MAC}(K, (A(i) \parallel V))$ $W = W \parallel w_i$ Return leftmost n bits of W
NIST SP 800-90	IEEE 802.11i	TLS/WTLS

Figure 12.15 Three PRNGs Based on HMAC



pythonProject8 | HMAC.py

Project | pythonProject8 | F:\Users\ayymn\PycharmProjects\pythonProject8\venv\Scripts\python.exe

```
6
7 # Define the message
8 message = b'This is a confidential message.'
9
10 # Generate HMAC using MD5
11 hmac_md5 = hmac.new(secret_key, message, hashlib.md5).hexdigest()
12 print(f"HMAC-MD5: {hmac_md5}")
13
14 # Generate HMAC using SHA1
15 hmac_sha1 = hmac.new(secret_key, message, hashlib.sha1).hexdigest()
16 print(f"HMAC-SHA1: {hmac_sha1}")
17
18 # Generate HMAC using SHA256 (SHA-2)
19 hmac_sha256 = hmac.new(secret_key, message, hashlib.sha256).hexdigest()
20 print(f"HMAC-SHA256: {hmac_sha256}")
21
22 # Generate HMAC using SHA512 (SHA-2)
23 hmac_sha512 = hmac.new(secret_key, message, hashlib.sha512).hexdigest()
```

Run: HMAC

F:\Users\ayymn\PycharmProjects\pythonProject8\venv\Scripts\python.exe F:/Users/ayymn/PycharmProjects/pythonProject8/HMAC.py

HMAC-MD5: 85f2487c25777f249f417e31dcf64d54

HMAC-SHA1: d6fc20d5ddf328ecc8edeae8386e5bfe3d17b130

HMAC-SHA256: e9067bc217dcd5053a320d364016eec812aad4991d845a04883fb2e1c67c8e65

HMAC-SHA512: 7db2f8657985f71693f3d60eb6733cd60d5b03cfab70e5706ab40660357f7cf947f86172e6eca0de7fcf8a84136d45f613ee71caa2b06bd42b0a92d03

Process finished with exit code 0

Run | TODO | Problems | Terminal | ExcelReader | Python Console | Event Log | ExcelEditor

8:1 CRLF UTF-8 4 spaces 0% / 0% Python 3.9 (pythonProject8) 395 of 1979M

pythonProject8 AES GCM.py

Project

- pythonProject8
- venv
- AEC CBC D.py
- AEC CBC E.py
- AEC CFB Dec.py
- AEC CFB Enc.py
- AES EBC.py
- AES GCM.py
- DEC EBC.py
- hash.py
- HMAC.py
- input_videos.mp4
- main.py
- part_1.mp4
- part_1TEMP_MPY_wvf_snd.mp4
- RSA.py
- X255.py
- External Libraries
- Scratches and Consoles

```
32 print(f"Tag: {tag.hex()}")
33 print(f"Nonce: {nonce.hex()}")
34
35 # Decrypt the ciphertext
36 decryptor = Cipher(algorithms.AES(key), modes.GCM(nonce, tag), backend=backend).decryptor()
37
38 # Include associated data (must match the encryption AAD)
39 decryptor.authenticate_additional_data(associated_data)
40
41 # Decrypt the ciphertext
42 decrypted_text = decryptor.update(ciphertext) + decryptor.finalize()
43
44 print(f"Decrypted Text: {decrypted_text.decode('utf-8')}")
45
46 # Verify that decryption matches the original plaintext
47 assert decrypted_text == plaintext, "Decryption failed!"
48
```

Run AES GCM

F:\Users\ayymn\PycharmProjects\pythonProject8\venv\Scripts\python.exe "F:/Users/ayymn/PycharmProjects/pythonProject8/AES GCM.py"

Ciphertext: 8cbc286e10f9fc3c0b6f8870ace5e69b9f520d01a3d80fcd1e157d544b26f8dbccac5a02

Tag: 049937b7bab07c48dcd96c80f34d789

Nonce: b6e9727046fd37462241d889

Decrypted Text: Sensitive data that needs encryption

Process finished with exit code 0

Run TODO Problems Terminal ExcelReader Python Console

5:22 (7 chars) CRLF UTF-8 4 spaces 0% / 1% Python 3.9 (pythonProject8) 261 of 1979M

Summary

- List and explain the possible attacks that are relevant to message authentication
- Define the term *message authentication code*
- List and explain the requirements for a message authentication code
- Present an overview of HMAC
- Present an overview of CMAC
- Explain the concept of authenticated encryption
- Present an overview of CCM
- Present an overview of GCM
- Understand how a hash function or a message authentication code can be used for pseudorandom number generation





Thank You

