


ECE 4309

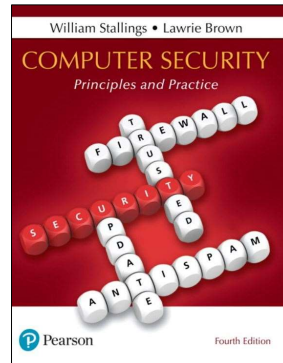
# Basics of cryptography – part 1

Dr. Valerio Formicola



## Computer Security: Principles and Practice

Fourth Edition



### Chapter 2

Cryptographic Tools



Copyright © 2018, 2015, 2012 Pearson Education, Inc. All Rights Reserved

If this PowerPoint presentation contains mathematical equations, you may need to check that your computer has the following installed:

- 1) MathType Plugin
- 2) Math Player (free versions available)
- 3) NVDA Reader (free versions available)

An important element in many computer security services and applications is the use of cryptographic algorithms. This chapter provides an overview of the various types of algorithms, together with a discussion of their applicability. For each type of algorithm, we will introduce the most important standardized algorithms in common use. For the technical details of the algorithms themselves, see Part Four.

We begin with symmetric encryption, which is used in the widest variety of contexts, primarily to provide confidentiality. Next, we examine secure hash functions and discuss their use in message authentication. The next section examines public-key encryption, also known as asymmetric encryption. We then discuss the two most important applications of public-key encryption, namely digital signatures and key management. In the

case of digital signatures, asymmetric encryption and secure hash functions are combined to produce an extremely useful tool.


Finally, in this chapter, we provide an example of an application area for cryptographic algorithms by looking at the encryption of stored data.

# Cryptography


*It's a technic that can be used in various ways to provide all CIA properties, Confidentiality, Integrity and Availability.*

*Techniques used in cryptography with different purposes:*

- *Symmetric encryption*
- *Secure message hashing (hash functions)*
- *Asymmetric encryption*



# Confidentiality with symmetric encryption

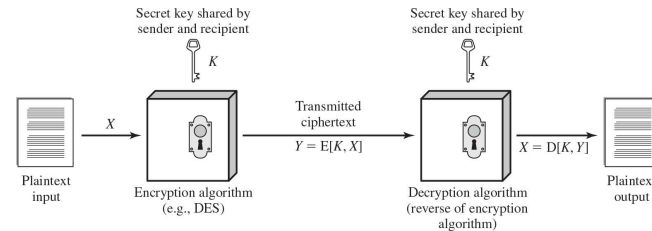


# Confidentiality

- It can be achieved if we avoid anybody able to observe a message, is still not able to understand the content of the message
  - e.g., the attacker might capture it on a transmission medium, like cable or air (type of attack called as *eavesdropping* attack, a type of passive attack)



**Figure 2.1 Simplified Model of Symmetric Encryption**



Note: a plaintext message is not necessarily text. It is anything that you can use with proper decoding; for example, a text with letters or a multimedia form, or simply commands or instructions

A symmetric encryption scheme has five ingredients (Figure 2.1):

- **Plaintext:** This is the original message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

The plain text is fed in to the algorithm as input. Encryption Algorithm, example, A E S. The input secret key K is shared by sender and recipient to the encryption algorithm. The encryption algorithm produces the output as ciphertext,  $Y = E(K, X)$ . Decryption algorithm, reverse of encryption algorithm. It takes the ciphertext and the secret key K, shared by sender and recipient as the input and produces the original plain text.



## Symmetric Encryption

- The universal technique for providing confidentiality for transmitted or stored data
- Also referred to as conventional encryption or *single-key encryption*
- Two requirements for secure use:
  - Need a strong encryption algorithm
  - Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure

The universal technique for providing confidentiality for transmitted or stored data is symmetric encryption. This section introduces the basic concept of symmetric encryption. This is followed by an overview of the two most important symmetric encryption algorithms: the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES), which are block encryption algorithms. Finally, this section introduces the concept of symmetric stream encryption algorithms.

Symmetric encryption, also referred to as conventional encryption or single-key encryption, was the only type of encryption in use prior to the introduction of public-key encryption in the late 1970s. Countless individuals and groups, from Julius Caesar to the German U-boat force to present-day diplomatic, military, and commercial users, have used symmetric encryption for secret communication. It remains the more widely used of the two types of encryption.

There are two requirements for secure use of symmetric encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an

opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form:  
The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.

2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

## What can be attacked in such a schema?

- The attacker and anybody else usually knows how the encryption/decryption algorithm works, so that's not a secret
  - Algorithms are mostly standard and public
- The attacker might very well intercept and study the encrypted messages
  - that's why we need a mechanism that it doesn't allow to reverse the process

## Attacking Symmetric Encryption

### Cryptanalytic Attacks

- Rely on:
  - Mathematical “nature” of the algorithm
  - Some knowledge of the general characteristics of the plaintext
    - e.g., all emails start with Hello X ...
  - Ideal for the crypto-analyst: Some sample plaintext-ciphertext pairs (aka, *ciphertext analysis*)
- Exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or the key being used
  - If successful, all future and past messages encrypted with that key are compromised

### Brute-Force Attacks

- Try all possible keys on some ciphertext until an intelligible translation into plaintext is obtained
  - On average half of all possible keys must be tried to achieve success
- An attacking tool shows results of decryption with a wrong key

A cryptographic algorithm is said to be **breakable** if a crypto-analyst can systematically recover the original message without knowing the key

- Example of attack in cryptanalysis: **frequency analysis** (also known as **counting letters**) is the study of the frequency of letters or groups of letters in a ciphertext.
- Example of attack in cryptanalysis: **Dictionary attack** to use names and common words with small substitutions; **credential stuffing** to use databases of users and passwords on different accounts.

There are two general approaches to attacking a symmetric encryption scheme. The first attack is known as **cryptanalysis**. Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used. If the attack succeeds in deducing the key, the effect is catastrophic: All future and past messages encrypted with that key are compromised.

The second method, known as the **brute-force attack**, is to try every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

**Table 2.1 Comparison of Three Popular Symmetric Encryption Algorithms**

**Block ciphers:** The most commonly used symmetric encryption algorithms.

A block cipher processes the plaintext input in fixed-size blocks and produces a block of ciphertext of equal size for each plaintext block

	DES	Triple DES	AES
Plaintext block size (bits)	64	64	128
Ciphertext block size (bits)	64	64	128
Key size (bits)	56	112 or 168	128, 192, or 256

DES = Data Encryption Standard

AES = Advanced Encryption Standard

The most commonly used symmetric encryption algorithms are block ciphers. A block cipher processes the plaintext input in fixed-size blocks and produces a block of ciphertext of equal size for each plaintext block. The algorithm processes longer plaintext amounts as a series of fixed-size blocks. The most important symmetric algorithms, all of which are block ciphers, are the Data Encryption Standard (DES), triple DES, and the Advanced Encryption Standard (AES); see Table 2.1. This subsection provides an overview of these algorithms. Chapter 20 presents the technical details.

## Data Encryption Standard (DES)



- Until recently was the most widely used encryption scheme
  - FIPS PUB 46
  - Referred to as the Data Encryption Algorithm (DEA)
  - Uses 64 bit plaintext block and 56 bit key to produce a 64 bit ciphertext block



- Strength concerns:
  - Concerns about the algorithm itself
    - DES is the most studied encryption algorithm in existence.
    - Quite stable in practice but very old (i.e., very studied)
  - Concerns about the use of a 56-bit key
    - The speed of commercial off-the-shelf processors makes this key length woefully inadequate

Until recently, the most widely used encryption scheme was based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as FIPS PUB 46 (Data Encryption Standard, January 1977). The algorithm itself is referred to as the Data Encryption Algorithm (DEA). DES takes a plaintext block of 64 bits and a key of 56 bits, to produce a ciphertext block of 64 bits.

Concerns about the strength of DES fall into two categories: concerns about the algorithm itself, and concerns about the use of a 56-bit key. The first concern refers to the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. Over the years, there have been numerous attempts to find and exploit weaknesses in the algorithm, making DES the most-studied encryption algorithm in existence. Despite numerous approaches, no one has so far reported a fatal weakness in DES.

A more serious concern is key length. With a key length of 56 bits, there are  $2^{56}$  possible keys, which is approximately  $7.2 \times 10^{16}$  keys. Given the speed of commercial off-the-shelf processors, this key length is woefully inadequate. A paper from Seagate Technology [SEAG08] suggests that a rate of one billion ( $10^9$ ) key

combinations per second is reasonable for today's multicore computers. Recent offerings confirm this. Both Intel and AMD now offer hardware-based instructions to accelerate the use of AES. Tests run on a contemporary multicore Intel machine resulted in an encryption rate of about half a billion encryptions per second [BASU12]. Another recent analysis suggests that with contemporary supercomputer technology, a rate of  $10^{13}$  encryptions/s is reasonable [AROR12].

**Table 2.2 Average Time Required for Exhaustive Key Search**

Key Size (bits)	Cipher	Number of Alternative Keys	Time Required at	Time Required at
			10 <sup>9</sup> decryptions/s	10 <sup>12</sup> decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	$2^{56} \mu s = 1.125$ years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	$2^{127} \mu s = 5.3 \times 10^{21}$ years	$5.3 \times 10^{17}$ years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	$2^{167} \mu s = 5.8 \times 10^{33}$ years	$5.8 \times 10^{29}$ years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	$2^{191} \mu s = 9.8 \times 10^{40}$ years	$9.8 \times 10^{36}$ years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	$2^{255} \mu s = 1.8 \times 10^{60}$ years	$1.8 \times 10^{56}$ years

- Note: book as some error. The number of decryptions are per seconds (not per micro-seconds)

Table 2.2 shows how much time is required for a brute-force attack for various key sizes. As can be seen, a single PC can break DES in about a year; if multiple PCs work in parallel, the time is drastically shortened. And today's supercomputers should be able to find a key in about an hour. Key sizes of 128 bits or greater are effectively unbreakable using simply a brute-force approach. Even if we managed to speed up the attacking system by a factor of 1 trillion ( $10^{12}$ ), it would still take over 100,000 years to break a code using a 128-bit key.



## Triple DES (3DES)

- Repeats basic DES algorithm three times using either two or three unique keys
- First standardized for use in financial applications in ANSI standard X9.17 in 1985
- Attractions:
  - 168-bit key length overcomes the vulnerability to brute-force attack of DES
  - Underlying encryption algorithm is the same as in DES
- Drawbacks:
  - Algorithm is sluggish in software (slow when implemented)
  - Uses a 64-bit block size
    - Too many fragmentations for larger data

The life of DES was extended by the use of triple DES (3DES), which involves repeating the basic DES algorithm three times, using either two or three unique keys, for a key size of 112 or 168 bits. Triple DES (3DES) was first standardized for use in financial applications in ANSI standard X9.17 in 1985. 3DES was incorporated as part of the Data Encryption Standard in 1999, with the publication of FIPS PUB 46-3.

3DES has two attractions that assure its widespread use over the next few years. First, with its 168-bit key length, it overcomes the vulnerability to brute-force attack of DES. Second, the underlying encryption algorithm in 3DES is the same as in DES. This algorithm has been subjected to more scrutiny than any other encryption algorithm over a longer period of time, and no effective cryptanalytic attack based on the algorithm rather than brute force has been found. Accordingly, there is a high level of confidence that 3DES is very resistant to cryptanalysis. If security were the only consideration, then 3DES would be an appropriate choice for a standardized encryption algorithm for decades to come.

The principal drawback of 3DES is that the algorithm is relatively sluggish in software. The original DES was designed for mid-1970s hardware implementation and does not produce efficient software code. 3DES, which

requires three times as many calculations as DES, is correspondingly slower. A secondary drawback is that both DES and 3DES use a 64-bit block size. For reasons of both efficiency and security, a larger block size is desirable.

## Advanced Encryption Standard (AES)

- Needed a replacement for 3DES
  - 3DES was not reasonable for long term use
- NIST called for proposals for a new AES in 1997
  - Should have a security strength equal to or better than 3DES
  - Significantly improved efficiency
  - Symmetric block cipher
  - 128 bit data and 128/192/256 bit keys
- Selected Rijndael in November 2001
  - Published as FIPS 197
- AES is now widely available in commercial products.

Because of its drawbacks, 3DES is not a reasonable candidate for long-term use. As a replacement, NIST in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which should have a security strength equal to or better than 3DES and significantly improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits. Evaluation criteria included security, computational efficiency, memory requirements, hardware and software suitability, and flexibility.

In a first round of evaluation, 15 proposed algorithms were accepted. A second round narrowed the field to 5 algorithms. NIST completed its evaluation process and published a final standard (FIPS PUB 197) in November of 2001. NIST selected Rijndael as the proposed AES algorithm. AES is now widely available in commercial products. AES is described in detail in Chapter 20.

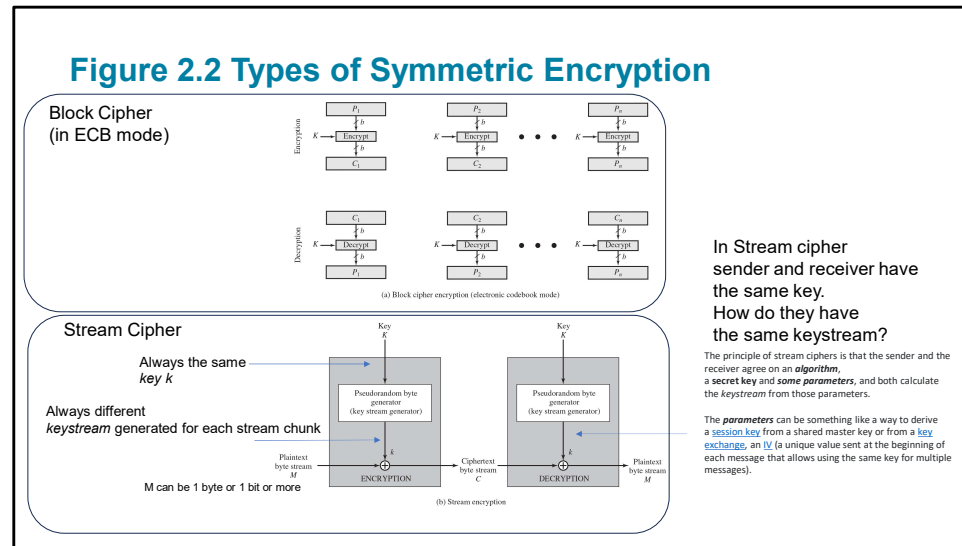


Figure 2.2a shows the ECB mode. A plaintext of length  $nb$  is divided into  $n$   $b$ -bit blocks ( $P_1, P_2, \dots, P_n$ ). Each block is encrypted using the same algorithm and the same encryption key, to produce a sequence of  $n$   $b$ -bit blocks of ciphertext ( $C_1, C_2, \dots, C_n$ ).

For lengthy messages, the ECB mode may not be secure. A cryptanalyst may be able to exploit regularities in the plaintext to ease the task of decryption. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext-ciphertext pairs with which to work.

To increase the security of symmetric block encryption for large sequences of data, a number of alternative techniques have been developed, called modes of operation. These modes overcome the weaknesses of ECB; each mode has its own particular advantages.

Figure 2.2b is a representative diagram of stream cipher structure. In this structure a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. A

pseudorandom stream is one that is unpredictable without knowledge of the input key and which has an apparently random character (see Section 2.5). The output of the generator, called a keystream, is combined one byte at a time with the plaintext stream using the bitwise exclusive- OR (XOR) operation.

Diagram a, block cipher encryption and decryption, electronic codebook mode. The plain text is divided into blocks from  $P_{sub\ 1}$  to  $P_{sub\ n}$ . Each block is encrypted one at a time to produce the cipher block. The same key is used to encrypt each block. The block  $P_{sub\ n}$  is encrypted to produce the cipher block  $c_{sub\ n}$ . Diagram b, stream encryption. The encryption and decryption is depicted in two blocks. The plain text is fed to the encryption. An input is shared by a key  $K$ , to pseudorandom byte generator, key stream generator and the pseudorandom byte  $k$  is fed to the encryption algorithm. The encryption algorithm produces the output as ciphertext byte stream  $C$ . The ciphertext is fed to the decryption algorithm. An input is shared by a key  $K$ , to pseudorandom byte generator, key stream generator and the pseudorandom byte  $k$  is fed to the decryption and it produces the final output as plain text byte stream  $M$ .

## Practical Security Issues

- Typically, symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block
- Electronic codebook (ECB)** mode is the simplest approach to multiple-block encryption
  - Each block (of same size) of plaintext is encrypted using the same key
  - Cryptanalysts may be able to exploit regularities in the plaintext
    - E.g., the beginning of a message might follow some patterns like a headline in an email, etc.
- Modes of operation
  - Alternative techniques developed to increase the security of symmetric block encryption for large sequences:
    - Cipher Block Chaining (CBC)
    - Cipher Feedback Mode (CFB)
    - Output Feedback Mode (OFB)
    - Counter Mode (CM)
  - Overcomes the weaknesses of ECB

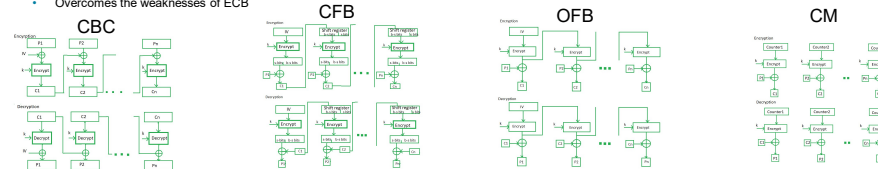
### Advantages of using ECB –

- Faster way of encryption in parallel mode.

- Simple way of the block cipher.

### Disadvantages of using ECB –

- Prone to cryptanalysis since there is a direct relationship between plaintext and ciphertext.



Typically, symmetric encryption is applied to a unit of data larger than a single 64-bit or 128-bit block. E-mail messages, network packets, database records, and other plaintext sources must be broken up into a series of fixed-length block for encryption by a symmetric block cipher. The simplest approach to multiple-block encryption is known as electronic codebook (ECB) mode, in which plaintext is handled *b bits at a time and each block of plaintext* is encrypted using the same key. Typically  $b = 64$  or  $b = 128$

For lengthy messages, the ECB mode may not be secure. A cryptanalyst may be able to exploit regularities in the plaintext to ease the task of decryption. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext-ciphertext pairs to work with.

To increase the security of symmetric block encryption for large sequences of data, a number of alternative techniques have been developed, called **modes of operation**. These modes overcome the weaknesses of ECB; each mode has its own particular advantages. This topic is explored in Chapter 20.

## Block & Stream Ciphers

- Block Cipher
  - Processes the input one block of elements at a time
  - Produces an output block for each input block
  - Can reuse keys
  - More common
- Stream Cipher
  - Processes the input elements continuously
  - Produces output one element at a time
  - Primary advantage is that they are almost always faster and use far less code
  - Encrypts plaintext one byte at a time
  - Pseudorandom stream is one that is unpredictable without knowledge of the input key and/or initialization parameters

A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along. Although block ciphers are far more common, there are certain applications in which a stream cipher is more appropriate. Examples are given subsequently in this book.

A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time.

With a properly designed pseudorandom number generator, a stream cipher can be as secure as block cipher of comparable key length. The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers. The advantage of a block cipher is that you can reuse keys. For applications that require encryption/decryption of a stream of data, such as over a data communications channel or a browser/Web link, a stream cipher might be the better alternative. For applications that deal with blocks of data, such as file transfer, e-mail, and database, block ciphers may be more appropriate. However, either type of cipher can be used in virtually any application.


## Applications for Block ciphers

1. **Data Encryption:** Block Ciphers are widely used for the encryption of private and sensitive data such as passwords, credit card details and other information that is transmitted or stored for a communication. This encryption process converts a plain data into non-readable and complex form. Encrypted data can be decrypted only by the authorised person with the private keys.
2. **File and Disk Encryption:** Block Ciphers are used for encryption of entire files and disks in order to protect their contents and restrict from unauthorised users. The disk encryption softwares such as BitLocker, TrueCrypt also uses block cipher to encrypt data and make it secure.
3. **Virtual Private Networks (VPN):** Virtual Private Networks (VPN) use block cipher for the encryption of data that is being transmitted between the two communicating devices over the internet. This process makes sure that data is not accessed by unauthorised person when it is being transmitted to another user.
4. **Secure Sockets Layer (SSL) and Transport Layer Security (TLS):** SSL and TLS protocols use block ciphers for encryption of data that is transmitted between web browsers and servers over the internet. This encryption process provides security to confidential data such as login credentials, card information etc.
5. **Digital Signatures:** Block ciphers are used in the digital signature algorithms, to provide authenticity and integrity to the digital documents. This encryption process generates the unique signature for each document that is used for verifying the authenticity and detecting if any malicious activity is detected.




## Applications for Stream ciphers

- Stream ciphers are often used for their speed and simplicity of implementation in hardware, and in applications where plaintext comes in quantities of unknowable length like a secure wireless connection. If a block cipher (not operating in a stream cipher mode) were to be used in this type of application, the designer would need to choose either transmission efficiency or implementation complexity, since block ciphers cannot directly work on blocks shorter than their block size.
- Mostly used for Real Time applications, e.g., media stream cryptography in DVD/BD players, etc.



# Message authentication and hash functions



## Message encryption alone cannot provide authentication

- Merely encrypting a message content, does not guarantee that the message is generated by an authentic source.
  - It's only for **confidentiality**
- A malicious actor might intercept the sequence of blocks in ECB encrypted by legitimate user and retransmit the blocks in a different order, hence changing the meaning of the message.
  - (aka, *message reordering attack*)
  - In general, we need more protection against *active attacks*

## Message Authentication

- We need to add more information to the original message to guarantee it is integer and authentic:
  - Contents have not been altered
  - From authentic source
  - Timely and in correct sequence
- Two mechanisms are usually combined:
  - Integrity: we guarantee that contents of message/data have not been altered
  - Authenticity: data comes from authentic source
- How to obtain: Additional information in a *Message Tag* preceding or following the original message

Original message

Tag

Encryption protects against passive attack (eavesdropping). A different requirement is to protect against active attack (falsification of data and transactions). Protection against such attacks is known as message or data authentication.

A message, file, document, or other collection of data is said to be authentic when it is genuine and came from its alleged source. Message or data authentication is a procedure that allows communicating parties to verify that received or stored messages are authentic. The two important aspects are to verify that the contents of the message have not been altered and that the source is authentic. We may also wish to verify a message's timeliness (it has not been artificially delayed and replayed) and sequence relative to other messages flowing between two parties. All of these concerns come under the category of data integrity as described in Chapter 1.

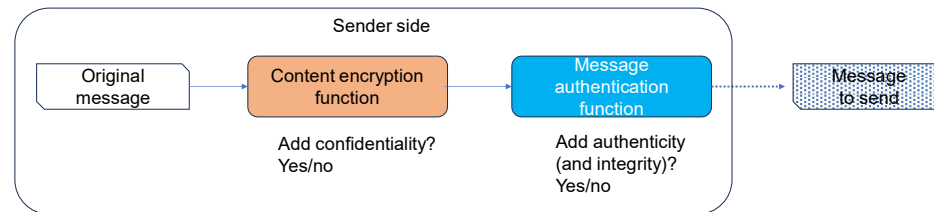
It would seem possible to perform authentication simply by the use of symmetric encryption. If we assume that only the sender and receiver share a key (which is as it should be), then only the genuine sender would be able to encrypt a message successfully for the other participant, provided the receiver can recognize a valid message. Furthermore, if the message includes an error-detection code and a sequence number, the receiver

is assured that no alterations have been made and that sequencing is proper. If the message also includes a timestamp, the receiver is assured that the message has not been delayed beyond that normally expected for network transit.

In fact, symmetric encryption alone is not a suitable tool for data authentication. To give one simple example, in the ECB mode of encryption, if an attacker reorders the blocks of ciphertext, then each block will still decrypt successfully. However, the reordering may alter the meaning of the overall data sequence. Although sequence numbers may be used at some level (e.g., each IP packet), it is typically not the case that a separate sequence number will be associated with each *b-bit block of plaintext*. Thus, *block reordering is a threat*.

## Observation: Authentication Vs Confidentiality as a functions

- It is possible to combine authentication and confidentiality in a single algorithm by encrypting a message plus its authentication tag
- Typically, message authentication is provided as a separate function from message encryption



## Case 1 (not used): Message authentication with symmetric encryption

- Sender and receiver are the only ones having the key and they know each other.
  - Apparently, we might achieve authenticity this way
  - However, ...
- Pure encryption of messages does not guarantee:
  - Protection from *reordering attacks*: the attacker simply stays in the middle of a communication and changes the sequence of data messages (1-2-3 -> 3-1-2)
  - Protection from *replay/delay attacks*: the attacker replays packets even if encrypted, to generate the same data at later time
- In practice: symmetric encryption alone is not a suitable tool for data authentication

## Case 2 (limited cases): Message Authentication Without Confidentiality

- We guarantee the message is sent from authentic source and it's not altered, but we don't encrypt the content of the message
  - Only authenticity and integrity, but not confidentiality
- Situations in which message authentication without confidentiality may be preferable include:
  - There are a number of applications in which the same message is broadcast to a number of destinations, like alarms or public messages from an authoritative source (e.g., police department)
  - An exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages
    - Only a few messages are encrypted in the content, most likely the ones that contain more critical information
  - Authentication of a computer program in plaintext is an attractive service

Because the approaches discussed in this section do not encrypt the message, message confidentiality is not provided. As was mentioned, message encryption by itself does not provide a secure form of authentication. However, it is possible to combine authentication and confidentiality in a single algorithm by encrypting a message plus its authentication tag. Typically, however, message authentication is provided as a separate function from message encryption. [DAVI89] suggests three situations in which message authentication without confidentiality is preferable:

1. There are a number of applications in which the same message is broadcast to a number of destinations. Two examples are notification to users that the network is now unavailable, and an alarm signal in a control center. It is cheaper and more reliable to have only one destination responsible for monitoring authenticity. Thus, the message must be broadcast in plaintext with an associated message authentication tag. The responsible system performs authentication. If a violation occurs, the other destination systems are alerted by a general alarm.
2. Another possible scenario is an exchange in which one side has a heavy load and cannot afford the time to



decrypt all incoming messages. Authentication is carried out on a selective basis, with messages being chosen at random for checking.

3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources. However, if a message authentication tag were attached to the program, it could be checked whenever assurance is required of the integrity of the program.

Thus, there is a place for both authentication and encryption in meeting security requirements.

## Case 3 (used for some cases): authentication with no confidentiality protection, i.e., message is in clear

1. **Approach 1:** Use of plain MAC (Message Authentication Code):
  - Compute a Tag that depends on the content of original message, plus a sequence number (*anti-reordering attack protection*)
  - Encrypt the Tag using a symmetric key (shared key)
  - Append to the original message (in clear)
  - Receiver will use secret key to decrypt the encrypted MAC tag and compare to the actual MAC of the message received
2. **Approach 2:** One-way hash function:
  - As before, compute a Tag but this time use a *one-way hash function* of the original message + original length information + padding
  - Encrypt the hash:
    - (2A) Using a symmetric key (shared key)
    - (2B) Using an asymmetric key (aka, public key)
  - Append to the original message (in clear)
  - Receiver will use the key (public or shared) to decrypt the encrypted Hash tag and compare to the actual Hash of message received
3. **Approach 3:** Keyed hash MAC:
  - You evaluate the Hash of a message concatenated to a secret key (shared key):  $H(K || M || K)$
  - Append the keyed Hash tag to the original message
  - Receiver will concatenate the received message with secret key, recalculate the keyed hash and compare with received Hash tag

## Approach 1: Message Authentication Using a Message Authentication Code (MAC)

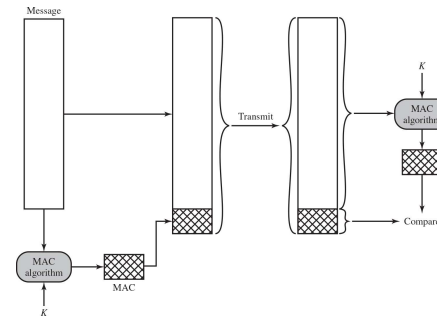
The message authentication code (MAC) is a complex function of the message and the secret key  $K_{AB}$  exchanged between sender and receiver:  $MAC_M = F(K_{AB}, M)$ .

MAC guarantees **integrity AND authentication**. Use of a key as an input to the function  $F$  guarantees authentication.  
Use of a message content  $M$  guarantees **integrity**.

Symmetric key is still used but only for the MAC tag

- 1 - secret key is used to encrypt a MAC tag (authenticity)
- 2 - no alteration of message is possible (integrity because nobody has secret key but legitimate ends of the communication)
- 3 - **if the message contains a sequence number**, any alteration of the sequence number will be detected as an integrity violation; moreover, sequence number will protect from *replay attacks* (next slide)

AES is usually used for authentication with MAC. Recently another called CMAC.



One authentication technique involves the use of a secret key to generate a small block of data, known as a message authentication code, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key  $K_{AB}$ . When A has a message to send to B, it calculates the message authentication code as a complex function of the message and the key:  $MAC_M = F(K_{AB}, M)$ . The message plus code are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new message authentication code. The received code is compared to the calculated code (Figure 2.3). If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the code, then the receiver's calculation of the code will differ from the received code. Because the attacker is assumed not to know the secret key, the attacker cannot alter the code to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret

key, no one else could prepare a message with a proper code.

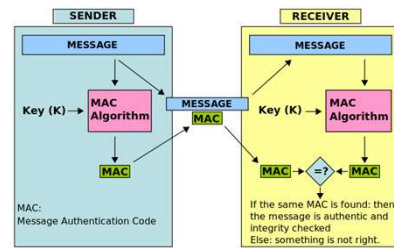
3. If the message includes a sequence number (such as is used with X.25, HDLC, and TCP), then the receiver can be assured of the proper sequence, because an attacker cannot successfully alter the sequence number.

A number of algorithms could be used to generate the code. The now withdrawn NIST publication FIPS PUB 113 (Computer Data Authentication , May 1985), recommended the use of DES. However, AES would now be a more suitable choice. DES or AES is used to generate an encrypted version of the message, and some of the bits of ciphertext are used as the code. A 16- or 32-bit code used to be typical but would now be much too small to provide sufficient collision resistance, as we will discuss shortly.

The process just described is similar to encryption. One difference is that the authentication algorithm need not be reversible, as it must for decryption. It turns out that because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.

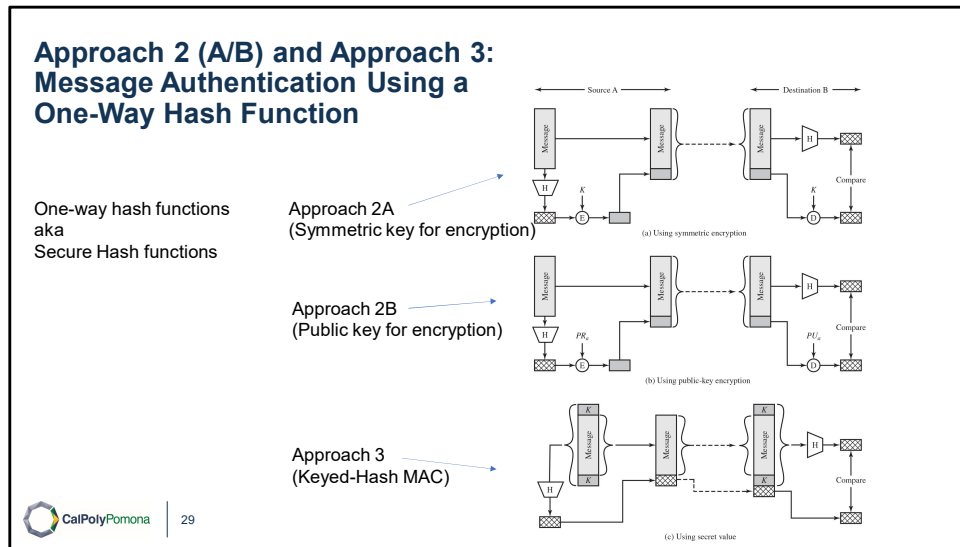
The steps are as follows. The first block labeled message is shared to M A C algorithm and a key K is shared to M A C algorithm. Step 2. The first block transmits the message to the second block with M A C and M A C is shared to the M A C in the second block. Step 3. The second block transmits the message to the third block with M A C. Step 4. The message in the third block is transmitted to M A C algorithm. A key is shared to the algorithm. The output from the algorithm is compared by M A C from the third block.

## Approach 1 (more)



In this example, the sender of a message runs it through a MAC algorithm to produce a MAC data tag. The message and the MAC tag are then sent to the receiver. The receiver in turn runs the message portion of the transmission through the same MAC algorithm using the same key, producing a second MAC data tag. The receiver then compares the first MAC tag received in the transmission to the second generated MAC tag. If they are identical, the receiver can safely assume that the message was not altered or tampered with during transmission (data integrity).

However, to allow the receiver to be able to detect replay attacks, the message itself must contain data that assures that this same message can only be sent once (e.g. time stamp, sequence number or use of a one-time MAC). Otherwise an attacker could – without even understanding its content – record this message and play it back at a later time, producing the same result as the original sender.



Unlike the MAC, a hash function does not also take a secret key as input. To authenticate a message, the message digest is sent with the message in such a way that the message digest is authentic. Figure 2.5 illustrates three ways in which the message can be authenticated using a hash code. The message digest can be encrypted using symmetric encryption (part a); if it is assumed that only the sender and receiver share the encryption key, then authenticity is assured. The message digest can also be encrypted using public-key encryption (part b); this is explained in Section 2.3. The public-key approach has two advantages: It provides a digital signature as well as message authentication; and it does not require the distribution of keys to communicating parties.

These two approaches have an advantage over approaches that encrypt the entire message in that less computation is required. But an even more common approach is the use of a technique that avoids encryption altogether. Several reasons for this interest are pointed out in [TSUD92]:

- Encryption software is quite slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.

- Encryption hardware costs are non-negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- An encryption algorithm may be protected by a patent.

Figure 2.5c shows a technique that uses a hash function but no encryption for message authentication. This technique, known as a keyed hash MAC, assumes that two communicating parties, say A and B, share a common secret key K. This secret key is incorporated into the process of generating a hash code. In the approach illustrated in Figure 2.5c, when A has a message to send to B, it calculates the hash function over the concatenation of the secret key and the message:

$MD_M = H(K \parallel M \parallel K)$ . It then sends  $[M \parallel MD_M]$  to B. Because B possesses K, it can recompute  $H(K \parallel M \parallel K)$  and verify  $MD_M$ . Because the secret key itself is not sent, it should not be possible for an attacker to modify an intercepted message. As long as the secret key remains secret, it should not be possible for an attacker to generate a false message.

Note that the secret key is used as both a prefix and a suffix to the message. If the secret key is used as either only a prefix or only a suffix, the scheme is less secure. This topic is discussed in Chapter 21. Chapter 21 also describes a scheme known as HMAC, which is somewhat more complex than the approach of Figure 2.5c and which has become the standard approach for a keyed hash MAC.

Diagram a, using symmetric encryption. The first block labeled message is shared to Hash function H algorithm M A C algorithm. A key K, is shared to the encryption algorithm. The first block transmits the message to the second block and the encrypted message is shared to the encrypted message in the second block. The process is represented as source A. The second block with encrypted message transmits the message to the third block with encrypted message. The message in the third block is transmitted to hash function and M A C. The encrypted message is fed to the decryption algorithm. A key is shared to the decryption algorithm. The output from the decryption algorithm is compared by M A C from the third block. This process is labeled, Destination B.

Diagram b, using public key encryption. The first block labeled message is shared to Hash function H algorithm M A C algorithm. A public key  $P_R$  sub a, is shared to the encryption algorithm. The first block transmits the message to the second block and the encrypted message is shared to the encrypted message in the second block. The process is represented as source A. The second block with encrypted message transmits the message to the third block with encrypted message. The message in the third block is transmitted to hash function and M A C. The encrypted message is fed to the decryption algorithm. A public key  $P_U$  sub a is shared to the decryption algorithm. The output from the decryption algorithm is compared by M A C from the third block. This process is labeled, Destination B.

Diagram c, using secret value. The first block labeled, message with keys at the top and bottom is transmitted to hash function, H. The hash function is fed to M A C and the message from the first block is transmitted to second block with M A C. The M A C is fed to the M A C in the second block. The message from the second block is transmitted to message in the third block with keys at the top and bottom. The M A C from the second block is fed to M A C in the third block. The message is transmitted to the hash function H. The output from H is fed to M A C. The M A C from the third block and the M A C from the hash function are compared.



### Secure Hash Functions: To Be Useful for Message Authentication, a Hash Function $H$ Must Have the Following Properties:

1. Can be applied to a block of data of any size
2. Produces a fixed-length output
3.  $H(x)$  is relatively easy to compute for any given  $x$
4. **One-way or pre-image resistant**
  - Computationally infeasible to find  $x$  such that  $H(x) = h$ , i.e., nobody should be able to find the inverse function of  $H$  because, otherwise  $H^{-1}(y) = x$ ;  $x$  might be  $K || M || K$  and the key  $K$  would be revealed if you also have  $M$
5. **Second pre-image resistant or weak collision resistant**
  - Computationally infeasible to find  $y \neq x$  such that  $H(y) = H(x)$
  - Note: here you get the hash of a message, but you cannot create another one that has the same hash of the first (*anti-forgery protection*)
6. **Collision resistant or strong collision resistance**
  - Computationally infeasible to find any pair  $(x, y)$  such that  $H(y) = H(x)$
  - Note: Here you don't have any starting message; the property states you should not be able to create two messages with the same hash



30

The purpose of a hash function is to produce a “fingerprint” of a file, message, or other block of data. To be useful for message authentication, a hash function  $H$  must have the following properties:

1.  $H$  can be applied to a block of data of any size.
2.  $H$  produces a fixed-length output.
3.  $H(x)$  is relatively easy to compute for any given  $x$ , making both hardware and software implementations practical.
4. For any given code  $h$ , it is computationally infeasible to find  $x$  such that  $H(x) = h$ . A hash function with this property is referred to as one-way or preimage resistant.
5. For any given block  $x$ , it is computationally infeasible to find  $y \neq x$  with  $H(y) = H(x)$ . A hash function with this property is referred to as second preimage resistant. This is sometimes referred to as weak collision resistant.

6. It is computationally infeasible to find any pair  $(x, y)$  such that  $H(x) = H(y)$ . A hash function with this property is referred to as collision resistant. This is sometimes referred to as strong collision resistant.

The first three properties are requirements for the practical application of a hash function to message authentication.

The fourth property is the one-way property: It is easy to generate a code given a message, but virtually impossible to generate a message given a code. This property is important if the authentication technique involves the use of a secret value (Figure 2.5c). The secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value: If the attacker can observe or intercept a transmission, the attacker obtains the message  $M$  and the hash code  $MD_M = H(K \parallel M \parallel K)$ . The attacker then inverts the hash function to obtain  $K \parallel M \parallel K = H^{-1}(MD_M)$ . Because the attacker now has both  $M$  and  $K \parallel M \parallel K$ , it is a trivial matter to recover  $K$ .

The fifth property guarantees that it is impossible to find an alternative message with the same hash value as a given message. This prevents forgery when an encrypted hash code is used (Figures 2.5a and b). If this property were not true, an attacker would be capable of the following sequence: First, observe or intercept a message plus its encrypted hash code; second, generate an unencrypted hash code from the message; third, generate an alternate message with the same hash code.

A hash function that satisfies the first five properties in the preceding list is referred to as a weak hash function. If the sixth property is also satisfied, then it is referred to as a strong hash function. A strong hash function protects against an attack in which one party generates a message for another party to sign. For example, suppose Bob gets to write an IOU message, send it to Alice, and she signs it. Bob finds two messages with the same hash, one of which requires Alice to pay a small amount and one that requires a large payment. Alice signs the first message and Bob is then able to claim that the second message is authentic.

## Security of Hash Functions

- There are two approaches to attacking a secure hash function:
  - Cryptanalysis
    - Exploit logical weaknesses in the algorithm
  - Brute-force attack
    - Strength of hash function depends solely on the length of the hash code produced by the algorithm
- MD5 which generates 128 bit hash, has been found to be breakable and it's not secure anymore
- SHA most widely used hash algorithm family. Currently SHA-2 (256, 384, 512 bit hashes) are the most used and SHA-3 will be in the future
- Additional secure hash function applications:
  - Passwords
    - Hash of a password is stored by an operating system
  - Intrusion detection
    - Store  $H(\text{File})$  for each file on a system and secure the hash values

As with symmetric encryption, there are two approaches to attacking a secure hash function: cryptanalysis and brute-force attack. As with symmetric encryption algorithms, cryptanalysis of a hash function involves exploiting logical weaknesses in the algorithm.

The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm. For a hash code of length  $n$ , the level of effort required is proportional to the following:

Preimage resistant  $2^n$

Second preimage resistant  $2^n$

Collision resistant  $2^{n/2}$

If collision resistance is required (and this is desirable for a general-purpose secure hash code), then the value  $2^{n/2}$  determines the strength of the hash code against brute-force attacks. Van Oorschot and Wiener [VANO94] presented a design for a \$10 million collision search machine for MD5, which has a 128-bit hash length, that could find a collision in 24 days. Thus a 128-bit code may be viewed as inadequate. The next step up, if a hash

code is treated as a sequence of 32 bits, is a 160-bit hash length. With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision. With today's technology, the time would be much shorter, so that 160 bits now appears suspect.

In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA). SHA was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993. When weaknesses were discovered in SHA, a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1. SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1. In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010. As discussed in Chapter 21, researchers have demonstrated that SHA-1 is far weaker than its 160-bit hash length suggests, necessitating the move to the newer versions of SHA.

We have discussed the use of hash functions for message authentication and for the creation of digital signatures (the latter is discussed in more detail later in this chapter). Here are two other examples of secure hash function applications:

- **Passwords:** Chapter 3 explains a scheme in which a hash of a password is stored by an operating system rather than the password itself. Thus, the actual password is not retrievable by a hacker who gains access to the password file. In simple terms, when a user enters a password, the hash of that password is compared to the stored hash value for verification. This application requires preimage resistance and perhaps second preimage resistance.
- **Intrusion detection:** Store  $H(F)$  for each file on a system and secure the hash values (e.g., on a CD-R that is kept secure). One can later determine if a file has been modified by recomputing  $H(F)$ . An intruder would need to change  $F$  without changing  $H(F)$ . This application requires weak second preimage resistance

## In summary

- **MAC function:**
  - Calculates an output Tag from the message in combination with a key
  - Hence provides integrity and authenticity
- **One-way hash function or Secure-hash function:**
  - message Tag is calculated only using the message content, not a key
  - Hence, it provides integrity but not authenticity
  - You need to combine with encryption for authenticity (symmetric or public encryption) to obtain authenticity
  - It has applicable to any blocks size
    - In contrast to stream ciphers that cannot be applied to blocks shorter than the key size
  - Generates the same output, whatever the input size
  - It's not computationally stressful
  - It can have several levels of security based on the resistance level:
    - preimage resistance: for essentially all pre-specified outputs, it is computationally infeasible to find any input that hashes to that output; i.e., given  $y$ , it is difficult to find an  $x$  such that  $h(x) = y$ .
    - second-preimage resistance: for a specified input, it is computationally infeasible to find another input which produces the same output; i.e., given  $x$ , it is difficult to find a second input  $x' \neq x$  such that  $h(x) = h(x')$ .
    - collision resistance (strong resistance): it is computationally infeasible to find any two distinct inputs  $x, x'$  that hash to the same output; i.e., such that  $h(x) = h(x')$