Reading for L2

▼ Table of Content

2.1: Confidentiality with symmetric encryption

- The universal technique for providing confidentiality for transmitted or stored data is symmetric encryption.

Symmetric encryption

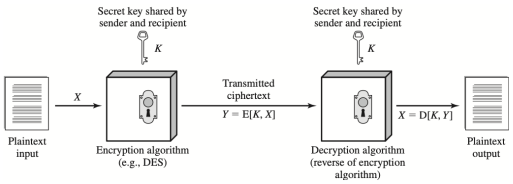


Figure 2.1 **Simplified Model of Symmetric Encryption**

- Symmetric encryption(conventional encryption, single-key encryption) was the only type of encryption in use prior to the introduction of public key encryption. It remains the more widely used of the two types of encryption.

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

- **Plaintext:** This is the original message or data that is fed in 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:** It takes the ciphertext and the secret key and produces the original plaintext.

There are two requirements for secure use of symmetric encryption:

1. **Strong Encryption Algorithm**: The algorithm should prevent an attacker, even with access to multiple ciphertexts and their corresponding plaintexts, from decrypting the ciphertext or discovering the encryption key.
2. **Secure Key Distribution and Protection**: The sender and receiver must securely obtain and maintain the secret key. If an attacker learns the key, they can decrypt all communication encrypted with that key.

There are two main methods for attacking symmetric encryption:

1. **Cryptanalysis**: This attack exploits the weaknesses of the encryption algorithm itself, potentially using knowledge of the plaintext structure or ciphertext-plaintext pairs. The goal is to deduce the key or plaintext. If successful, all messages encrypted with that key are compromised.
2. **Brute-Force Attack**: This method involves trying every possible key until the correct one is found. On average, half of all possible keys must be tested before success. However, recognizing the plaintext from the ciphertext is not always straightforward, especially if the message is compressed or consists of non-text data. To make brute-force more effective, attackers often need knowledge of the expected plaintext and tools to automatically recognize meaningful data.

Symmetric Block Encryption Algoritms

The most widely used symmetric encryption algorithms are **block ciphers**, which encrypt fixed-size blocks of plaintext and produce ciphertext blocks of the same size. These ciphers process longer plaintexts as a series of these fixed-size blocks. Key block cipher algorithms include **Data Encryption Standard (DES)**, **triple DES**, and **Advanced Encryption Standard (AES)**. Further technical details of these algorithms are covered in Chapter 20.

DES:

The **Data Encryption Standard (DES)**, adopted in 1977, was the most widely used encryption algorithm until recently. DES encrypts 64-bit plaintext blocks using a 56-bit key, producing 64-bit ciphertext blocks. However, there are two main concerns with DES:

1. **Algorithm Weaknesses**: Despite extensive cryptanalysis, no major vulnerabilities in the DES algorithm have been found.
2. **Key Length**: The 56-bit key provides only about 72 quadrillion possible keys (2^56), which is inadequate given the processing power of modern computers. With advances in hardware, such as multicore processors and

supercomputers, DES can be broken much more quickly. For example, a single PC could break DES in about a year, and supercomputers could do so in about an hour.

As a result, key sizes of 128 bits or larger are considered practically unbreakable using brute-force methods. Alternatives to DES, such as **triple DES** and **AES**, offer stronger encryption and are discussed later in the text.

Triple DES:

Triple DES (3DES) extends the life of DES by applying the DES algorithm three times with either two or three unique keys, resulting in key lengths of 112 or 168 bits. It was first standardized for financial applications in 1985 and became part of the official DES standard in 1999.

3DES has two main advantages:

1. **Stronger Security**: With its 168-bit key, 3DES is resistant to brute-force attacks, unlike DES.
2. **Proven Algorithm**: The DES algorithm, used in 3DES, has been thoroughly analyzed and is considered highly resistant to cryptanalysis.

However, **3DES has significant drawbacks**:

- **Slower Performance**: The algorithm is relatively slow in software, as it requires three times the number of operations compared to DES.
- **Small Block Size**: Both DES and 3DES use a 64-bit block size, which is less efficient and secure compared to larger block sizes.

Despite these limitations, 3DES remains widely used due to its strong security.

AES:

Due to the limitations of 3DES, it was deemed unsuitable for long-term use. In 1997, NIST initiated a search for a replacement, issuing a call for proposals for the **Advanced Encryption Standard (AES)**. AES was required to provide security at least as strong as 3DES, with improved efficiency. Key specifications included:

- Symmetric block cipher with a 128-bit block size.

- Support for key lengths of 128, 192, and 256 bits.
- Evaluation based on security, efficiency, memory usage, and hardware/software compatibility.

After reviewing 15 proposals, NIST narrowed it down to five, ultimately selecting **Rijndael** as the AES algorithm. AES was finalized as FIPS PUB 197 in November 2001 and is now widely used in commercial products. Further details on AES are provided in Chapter 20.

Practical security issues:

In symmetric encryption, large units of data, such as email messages or network packets, must be divided into fixed-length blocks for encryption by a block cipher. The simplest method for handling multiple blocks is **Electronic Codebook (ECB) mode**, where each block of plaintext is encrypted independently using the same key. This results in a sequence of ciphertext blocks.

However, **ECB mode is not secure for large messages** because patterns in the plaintext can be exploited by attackers. For instance, if the message contains predictable fields, the attacker might use known plaintext-ciphertext pairs to aid decryption.

To address these weaknesses, **modes of operation** have been developed. These alternatives provide better security for encrypting larger datasets. Each mode has its own advantages, and the details are covered in Chapter 20.

Stream Ciphers

A **block cipher** processes data one fixed-size block at a time, producing an output block for each input block. In contrast, a **stream cipher** encrypts data continuously, processing one element at a time. While block ciphers are more commonly used, stream ciphers are more suitable for certain applications.

A typical stream cipher encrypts data one byte at a time, although it can also operate on smaller or larger units. It uses a **pseudorandom bit generator** to produce a keystream, which is combined with the plaintext using a bitwise **XOR** operation. The keystream appears random without knowledge of the key.

Stream ciphers can be as secure as block ciphers if designed properly, with the advantage of being **faster** and requiring less code. Stream ciphers are often better for applications involving continuous data streams, such as data communications or web browsing, while block ciphers are more suited for applications that handle discrete blocks of data, like file transfers or email.

Both types of ciphers can be used for various applications, depending on the specific needs.

2.2

Authentication Using Symmetric Encryption

While symmetric encryption might seem sufficient for **authentication**, it isn't ideal on its own. If the sender and receiver share a secret key, the sender could encrypt a message that only the receiver could decrypt, ensuring authenticity. Adding features like error-detection codes, sequence numbers, and timestamps could further prevent tampering and ensure proper sequencing and timely delivery.

However, **symmetric encryption alone** has vulnerabilities. For example, in **ECB mode**, an attacker could reorder the ciphertext blocks, and each block would still decrypt correctly. This reordering could change the meaning of the entire message. While sequence numbers could help, they typically aren't applied to each block of plaintext, leaving block reordering as a potential threat. Therefore, additional mechanisms are needed for effective data authentication.

Message Authentication without Message Encryption

This section discusses **message authentication approaches** that don’t rely on encryption. In these methods, an **authentication tag** is added to the message for verification, but the message itself remains unencrypted and can be read without the authentication process. While these methods ensure message authenticity, **confidentiality** is not provided since the message is not encrypted.

However, it's possible to combine authentication and confidentiality by encrypting both the message and its authentication tag, though typically, authentication and encryption are handled separately.

Three scenarios where **message authentication without confidentiality** is useful include:

1. **Broadcast Applications**: When a message needs to be sent to multiple recipients (e.g., network downtime notifications), it's more efficient to broadcast the message in plaintext with an authentication tag, leaving one destination to handle authentication.
2. **Selective Authentication**: When one side of a communication is overwhelmed and can't decrypt all incoming messages, random messages are authenticated to reduce processing load.
3. **Program Integrity Verification**: A computer program can be authenticated in plaintext by attaching an authentication tag, allowing it to be executed without decryption, saving processing resources.

In summary, both **authentication** and **encryption** play key roles in meeting different security needs.

Message authentication code (MAC)

One authentication technique uses a **message authentication code (MAC)**, which is generated with a shared secret key between two parties (A and B). When A sends a message to B, A calculates the MAC as a function of the message and the shared key: **MAC = F(KAB, M)**. A then sends both the message and the MAC to B. Upon receiving the message, B performs the same calculation using the shared key to generate a new MAC, which is compared to the received MAC.

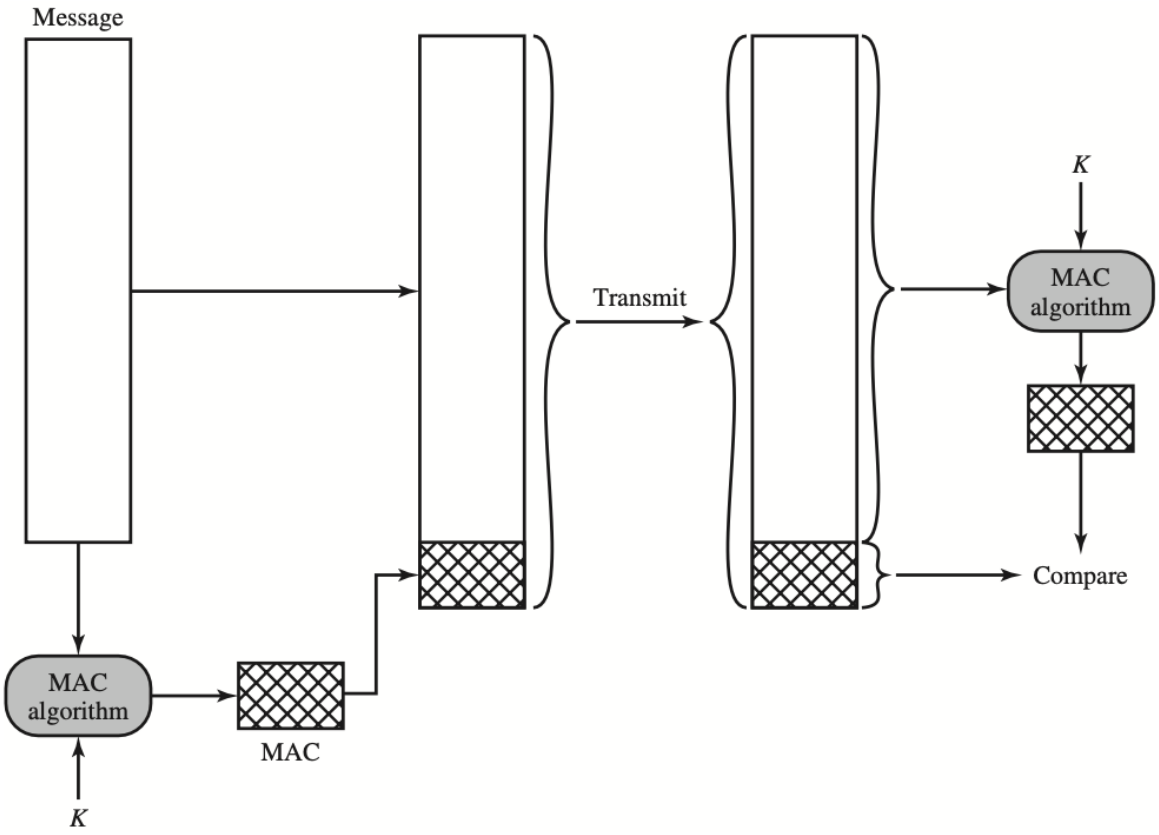
If the codes match, it provides three assurances:

1. **Message Integrity**: The message hasn’t been altered. If an attacker changes the message but doesn’t know the secret key, they can’t correctly alter the MAC to match the modified message.
2. **Sender Authentication**: The message is from the alleged sender because only the sender and receiver know the secret key, making it impossible for an

attacker to generate a valid MAC.

3. **Correct Sequence**: If the message includes a sequence number, the receiver can ensure the message is in the proper order since attackers can't modify the sequence number without invalidating the MAC.

Various algorithms, such as **DES** or **AES**, can be used to generate the MAC. AES is now preferred due to its stronger security. The MAC is typically a small block (16 or 32 bits), but small sizes are less secure today because they're more vulnerable to **collisions**. Unlike encryption, the authentication algorithm doesn't need to be reversible, and it is less vulnerable to being broken due to its mathematical properties.



One-way hash function

An alternative to the **message authentication code (MAC)** is the **one-way hash function**. Like a MAC, a hash function takes a variable-size message and

produces a fixed-size output (the message digest). However, unlike the MAC, a hash function **does not** require a secret key. The message is often padded to a fixed length, and the padding includes the length of the original message to increase security against attacks.

There are several ways to authenticate a message using a hash function:

1. **Symmetric Encryption**: The message digest is encrypted with a shared key between sender and receiver (similar to a MAC), ensuring authenticity.
2. **Public-Key Encryption**: The message digest is encrypted using the sender's private key, providing both **authentication** and a **digital signature**. This method avoids key distribution issues.
3. **Keyed Hash MAC**: The most common approach, which combines a secret key with the message in the hash function. The sender computes the hash of the concatenated message and key, then sends the message and hash to the receiver. The receiver uses the same secret key to recompute the hash and verify the message.

The **keyed hash MAC** is particularly popular because it avoids the computational overhead of encryption. **Encryption** has several drawbacks:

- It's slow, especially when processing a steady stream of messages.
- It's costly, particularly when encryption hardware is needed at all network nodes.
- It may be optimized for larger data sizes, making it inefficient for small messages.
- Encryption algorithms may be patented.

In the **keyed hash MAC** method (Figure 2.5c), the secret key is used both as a **prefix** and **suffix** to the message for added security. If only one part of the message is keyed, it becomes less secure. **HMAC**, a more complex version of this method, is described in Chapter 21 and has become the standard approach for keyed hash MACs.

Secure Hash Functions

First we discus the requirements for a secure hash function. Then we discuss specific algorithms.

Hash Function Requirements

A **hash function** is designed to produce a unique "fingerprint" or digest of data, such as a file or message. For message authentication, a hash function must have the following properties:

1. **Flexible Input Size**: It can process data of any size.
2. **Fixed-Length Output**: It generates a fixed-size output regardless of input size.
3. **Efficiency**: It should be easy to compute in both hardware and software.
4. **One-Way (Preimage Resistant)**: Given a hash output, it's computationally infeasible to reverse-engineer the original input.
5. **Second Preimage Resistance**: It’s difficult to find a different input that produces the same hash output.
6. **Collision Resistance**: It’s computationally infeasible to find any two different inputs that produce the same hash value.

The first three properties make a hash function practical for message authentication. The remaining properties protect against various attacks:

- **One-Way Property**: Ensures that attackers cannot easily reverse the hash to discover the secret key used in message authentication (e.g., in the keyed hash MAC method).
- **Second Preimage Resistance**: Prevents attackers from finding a different message that hashes to the same value, which could be used to forge messages.
- **Collision Resistance**: Ensures that two different messages cannot generate the same hash, preventing malicious modifications (e.g., in digital signatures).

A **weak hash function** satisfies the first five properties, while a **strong hash function** satisfies all six, providing better security, including protection against forgeries where an attacker might trick a party into signing a different message with the same hash.

In addition to authentication, a message digest ensures **data integrity** by flagging errors if any part of the message is altered, similar to a frame check sequence.

Security of Hash Functions

There are two main ways to attack a secure hash function: **cryptanalysis** (exploiting logical weaknesses in the algorithm) and **brute-force attacks** (trying all possible inputs).

The strength of a hash function against brute-force attacks depends on the length of the hash code produced. For a hash code of length *n*:

- **Preimage resistance** (difficulty of finding the original input from the hash): requires 2^n attempts.
- **Second preimage resistance** (difficulty of finding a different input with the same hash): requires 2^n attempts.
- **Collision resistance** (difficulty of finding two inputs with the same hash): requires 2^(n/2) attempts.

For collision resistance, the strength is determined by 2^(n/2). For example, MD5, with its 128-bit hash, would require a machine designed for collision searches to find a collision in about 24 days. This makes a 128-bit hash considered weak today. A 160-bit hash would require over 4,000 years to find a collision with the same machine, but with current technology, even a 160-bit hash could be vulnerable in a shorter time frame.

Other Applications of Hash Functions

Here are two other examples of secure hash function applications:

- **Passwords:** Chapter 3 will explain a scheme in which a hash of a password is stored by an operating system rather than the password itself.
  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 the hash value for a file, H(F), for each file on a system and secure the hash values (e.g., on a write-locked drive or write-once
  optical disk 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.

20.1: Symmetric encryption principles

Cryptography

Cryptographic systems are generically classified along three independent dimensions:

1. The type of operations used for transforming plaintext to ciphertext:
Encryption algorithms rely on two main principles: **substitution** (where elements in the plaintext are replaced with other elements) and **transposition** (where the order of elements in the plaintext is rearranged). The key requirement is that these processes must be reversible, meaning no information is lost. Most encryption systems, called **product systems**, use multiple rounds of substitution and transposition to increase security.

2. The number of keys used: When sender and receiver use the same key = symmetric (single-key, secret-key, conventional) encryption. Different keys = asymmetric (two-key, public-key) encryption.
3. The way in which the plaintext is processed
: A **block cipher** encrypts data in fixed-size blocks, producing one output block for each input block. A **stream cipher**, on the other hand, encrypts data one element at a time, continuously processing the input as it goes along.

Cryptanalysis

Cryptanalysis is the process of trying to uncover the plaintext or key used in encryption. The approach depends on the encryption method and the information available to the cryptanalyst.

The simplest scenario is a **ciphertext-only attack**, where the attacker has only the ciphertext. In this case, the attacker might use a brute-force method, trying every possible key, but this becomes impractical if the key space is large. Instead, the attacker may apply statistical analysis or make educated guesses about the type of plaintext (e.g., English text or a specific file format) to help break the encryption.

More information can make attacks easier. For example (Table 20.1):

- **Known-plaintext attacks**: The attacker has both plaintext and its corresponding ciphertext, helping to deduce the key.
- **Probable-word attacks**: The attacker may know certain key words or patterns in a message (e.g., a specific phrase in an accounting file).
- **Chosen-plaintext attacks**: The attacker can choose specific plaintexts to encrypt, revealing more about the encryption process.

There are also **chosen-ciphertext** and **chosen-text** attacks, though they are less common.

For an encryption scheme to be considered **computationally secure**, the effort to break it should either:

- Exceed the value of the encrypted information, or
- Take more time than the information is useful for.

In most cases, **brute-force attacks** are the fallback method. If there are no weaknesses in the algorithm, the attacker would need to try half of all possible keys to break the cipher, though this process can be very time-consuming and costly.

Feistel Cipher Structure

Video: Feistel Cipher - Computerphile

Many symmetric block encryption algorithms, like DES, use a structure called the **Feistel cipher.** In this structure, the plaintext is divided into two halves, and the encryption process involves multiple rounds of substitution and permutation. Each round uses a subkey derived from the main key, and applies a round function

(often involving XOR) to transform the data. After each round, the halves of the data are swapped.

Key parameters in designing a symmetric block cipher include:

- **Block size**: Larger blocks offer more security but slower processing. A 128-bit block is common.
- **Key size**: Larger keys improve security but slow down the encryption/decryption speed. A 128-bit key is typical.
- **Number of rounds**: More rounds increase security; 16 rounds is a standard choice.
- **Subkey generation and round function**: Greater complexity here enhances resistance to cryptanalysis.

Other design considerations:

- **Speed**: For software implementations, speed is important as hardware solutions may not be feasible.
- **Ease of analysis**: While making an algorithm difficult to break is crucial, having an algorithm that’s easy to analyze helps in evaluating its security.

Decryption in a Feistel cipher is essentially the reverse of encryption: the same rounds are performed, but with the subkeys applied in reverse order, which means encryption and decryption can share the same algorithm.

20.5 CIPHER BLOCK MODES OF OPERATION

A symmetric block cipher encrypts data one block at a time. For example, DES and 3DES use 64-bit blocks. If the plaintext is longer, it's divided into 64-bit blocks, with padding added to the last block if needed. To apply a block cipher to different scenarios, NIST SP 800-38A defines **five modes of operation** that cover almost all encryption needs. These modes can be used with any symmetric block cipher, including AES and 3DES. The modes are summarized in table 20.3, with key ones described further.

Mode	Description	Typical Application
Electronic Code book (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed s bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

Electronic codebook (ECB) mode

Video: Electronic codebook (ECB) mode

The simplest way to proceed, Electronic Codebook (ECB) mode encrypts plaintext one block at a time, using the same key for each block. Each unique plaintext block results in a unique ciphertext block, similar to a "codebook" where each plaintext pattern maps to its corresponding ciphertext.

However, ECB has security weaknesses: if the same block of plaintext appears multiple times in the message, it will always produce the same ciphertext. This predictability can be exploited by a cryptanalyst, especially in structured messages with repetitive patterns. Such regularities could allow an attacker to infer information or manipulate blocks.

To improve security, a method is needed where repeated plaintext blocks result in different ciphertext blocks, even if the plaintext is the same.

Cipher block chaining (CBC) mode

Video: Cipher block chaining (CBC) mode

In **Cipher Block Chaining (CBC)** mode, the encryption of each plaintext block is combined with the previous ciphertext block using XOR before being encrypted

with the same key. This chaining process ensures that even if the same plaintext block is repeated, the ciphertext will be different. This adds security by preventing repeating patterns from being exposed.

For decryption, each ciphertext block is decrypted and XORed with the previous ciphertext block to recover the plaintext. The process relies on an **Initialization Vector (IV)** for the first block, which is XORed with the first plaintext block before encryption. On decryption, the IV is used again to retrieve the plaintext.

The IV must be shared between the sender and receiver and should be protected. If an attacker can manipulate the IV, they could alter specific bits in the first plaintext block, compromising the security of the system. Thus, the IV should be securely transmitted, potentially using ECB mode, to prevent predictable changes by an adversary.

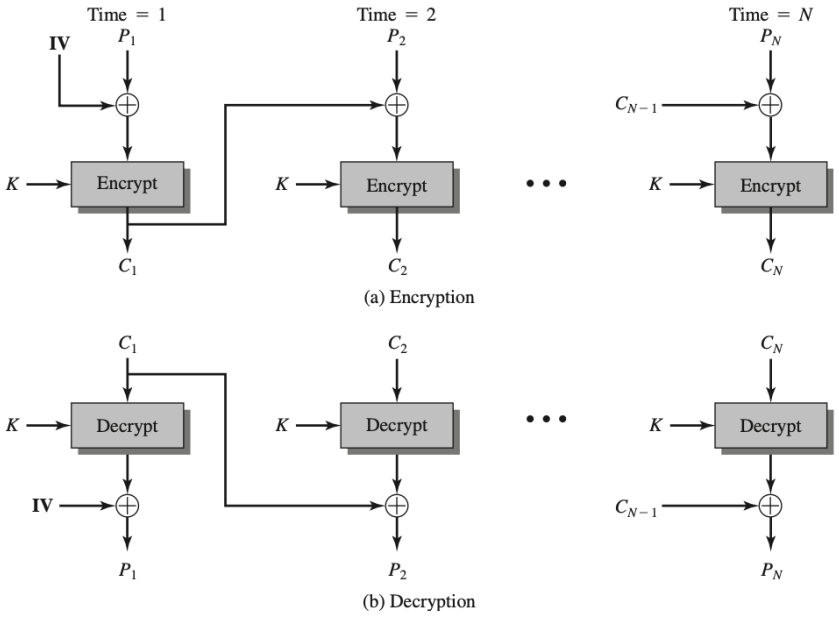


Figure 20.7 **Cipher Block Chaining (CBC) Mode**

Cipher Feedback Mode

Video: Cipher Feedback (CFB) mode

Video: Cipher Feedback Mode - Applied Cryptography

The **Cipher Feedback (CFB)** mode allows a block cipher to function as a **stream cipher**. This eliminates the need for message padding and enables real-time encryption of data, making it ideal for character-oriented streams (like transmitting individual characters). A key feature of a stream cipher is that the **ciphertext** is the same length as the **plaintext**, so no transmission capacity is wasted.

In **CFB encryption**, a shift register is initialized with a **Initialization Vector (IV)**. The leftmost bits of the encrypted IV are XORed with the plaintext to produce the ciphertext. Then, the shift register is updated by shifting the bits and adding the ciphertext into the register. This process repeats for each block of plaintext.

For **decryption**, the same process is used, but the received ciphertext is XORed with the encrypted output instead of using a decryption function. The encryption function is applied the same way as in encryption, ensuring the recovery of the plaintext.

CFB allows for efficient and secure encryption of streams, with no need to pad or wait for full blocks to be processed.

Counter (CTR) mode

Video: Counter (CTR) mode

The **Counter (CTR) mode** of encryption, proposed early on and gaining popularity for applications like **ATM** network security and **IPSec**, uses a counter equal to the block size. The counter is initialized to a value and incremented for each subsequent block. For encryption, the counter is encrypted and then XORed with the plaintext to produce the ciphertext, without chaining between blocks. Decryption follows the same process, using the encrypted counter values to recover the plaintext.

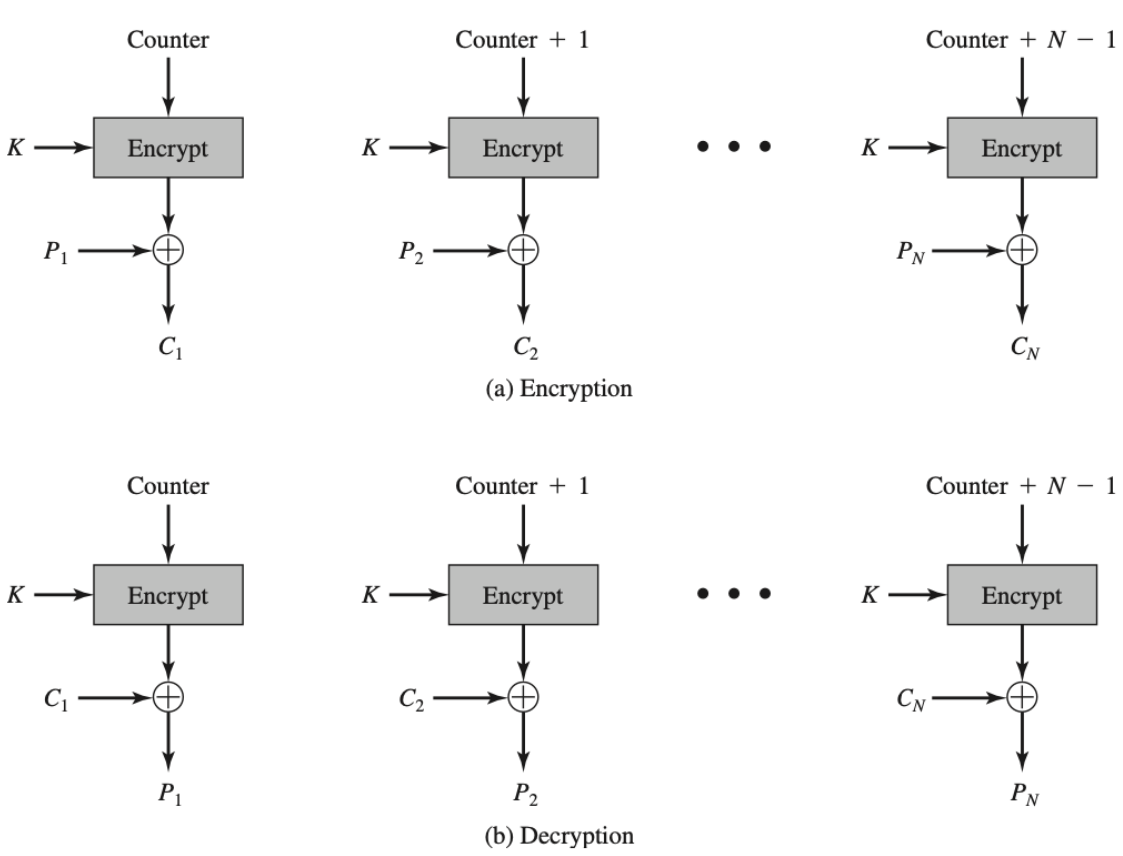


Figure 20.9 **Counter (CTR) Mode**

Advantages of CTR mode:

1. **Hardware Efficiency**: It allows parallel encryption/decryption of multiple blocks, unlike other modes that require sequential processing.
2. **Software Efficiency**: CTR mode benefits from parallelism in processors, improving performance on modern hardware.
3. **Preprocessing**: The encryption algorithm can be precomputed, speeding up encryption when plaintext or ciphertext is available.
4. **Random Access**: Each block can be processed independently, which is useful for applications requiring access to individual blocks (e.g., decrypting specific ciphertext blocks).
5. **Provable Security**: CTR mode is considered at least as secure as other encryption modes.

6. **Simplicity**: Only the encryption algorithm needs to be implemented, not the decryption algorithm, making it simpler, especially when encryption and decryption algorithms differ, as with AES.

Overall, CTR mode is efficient, flexible, and secure, making it suitable for high-performance encryption tasks.