# Cybersecurity

## Professor: F. d'Amore

---

**A.Y. 2022/23**

**Table of Contents**

## List of Figures

## List of Tables

# 1 Introduction and terminologies

# 2 Symmetric Encryption

Secret key cryptography involves the use of a single key. Given a message (the plaintext) and the key, encryption produces unintelligible data which is about the same length as the plaintext was.
Secret key cryptography is sometimes referred to as **conventional cryptography** or **symmetric cryptography**.
Secret key encryption schemes require that both the party that does the encryption and the party that does the decryption share a secret key. We will discuss two types of secret key encryption schemes:

- **Stream Ciphers**: This uses the key as a seed for a pseudorandom number generator, produces a stream of pseudorandom bits, and $\oplus$ s (bitwise exclusive ors) that stream with the data. Since $\oplus$ is its own inverse, the same computation performs both encryption and decryption.
- **Block Ciphers**: This takes as input a secret key and a plaintext block of fixed size (older ciphers used 64-bit blocks, modern ciphers use 128-bit blocks). It produces a ciphertext block the same size as the plaintext block. To encrypt messages larger than the blocksize, the block cipher is used iteratively with algorithms called *modes of operation*. A block cipher also has a decryption operation that does the reverse computation.

So, block ciphers encrypt data in blocks of set lengths, while stream ciphers do not and instead encrypt plaintext one byte at a time. The two encryption approaches, therefore, vary widely in implementation and use cases.
Here we will have some prerequisite concepts and then we will dive into these two encryption algorithms

## 2.1 Finite Fields

## 2.2 Per-Round Keys

## 2.3 S-boxes and Bit Shuffles

Write S-boxes and Bit Shuffle

## 2.4 Feistel Cipher

It is very important to make the encryption algorithm revered so that the decryption is possible. One method (used by AES) of having a cipher is to make all components reversible. With DES, the S-boxes are clearly not reversible since they map 6-bit inputs to 4-bit outputs. So instead, DES is designed to be reversible using a clever technique known as a Feistel cipher.

A Feistel cipher builds reversible transformations out of one-way transformations by only working on half the bits of the input value at a time. let us assume a 64-bit input block. (Figure 1) shows both how encryption and decryption work. In a Feistel cipher there is some irreversible component that scrambles the input. We'll call that component the mangler function.

In encryption for round n, the 64-bit input to round $n$ is divided into two 32-bit halves called $L_n$ and $R_n$. Round n generates as output 32-bit quantities $L_{n+1}$ and $R_{n+1}$. The concatenation of $L_{n+1}$ and $R_{n+1}$ is the 64-bit output of round n and, if there's another round, the input to round n+1. $L_{n+1}$ is simply $R_n$. To compute $R_{n+1}$, do the following. $R_n$ and $K_n$ are input to the mangler function, which takes as input 32 bits of data plus some bits of key to produce a 32-bit output. The 32-bit output of the mangler is XORed with $L_n$ to obtain $R_{n+1}$.

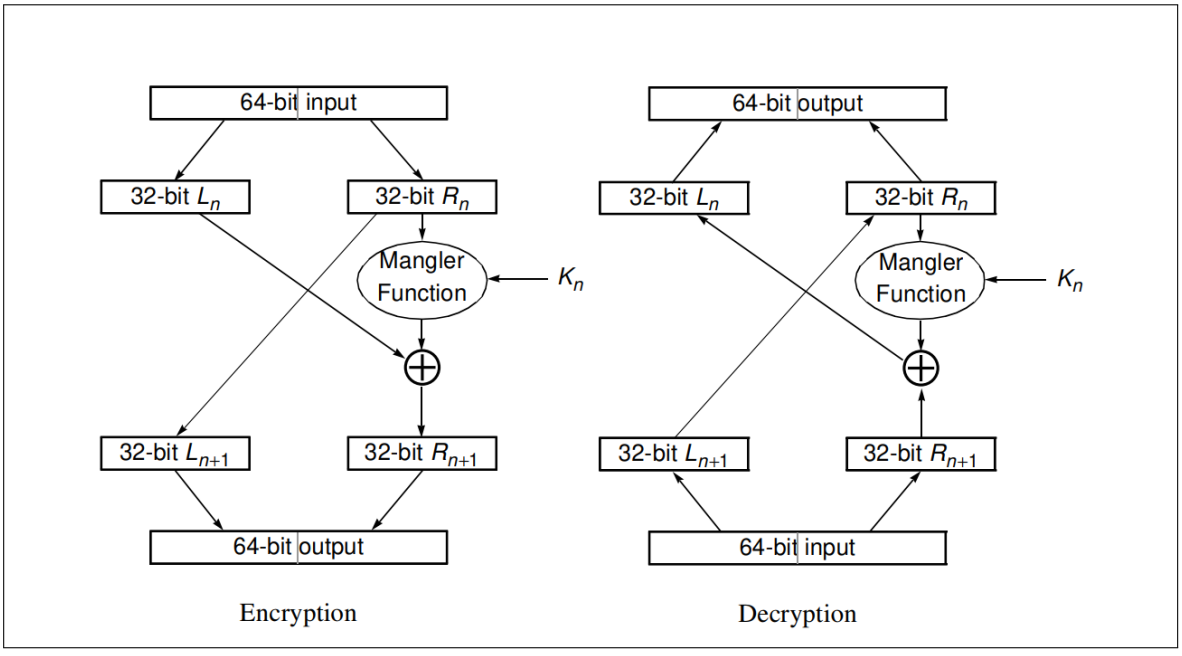


Figure 1: Feistel Cipher

## 2.5 Padding and Ciphertext Stealing

Padding and ciphertext stealing are techniques used in cryptography to handle messages or plaintext that are not an exact multiple of the block size in block cipher algorithms. Both techniques serve the purpose of ensuring proper encryption and decryption while maintaining the integrity and length of the ciphertext.

**Padding**:

Padding is the process of adding extra bits or bytes to the plaintext before encryption to ensure it aligns with the block size required by the encryption algorithm. The most commonly used padding scheme is called PKCS#7 (Public Key Cryptography Standard #7).

In PKCS#7 padding, if the plaintext is shorter than the block size, padding bytes are added to the end of the plaintext. The value of each padding byte is set to the number of padding bytes added. For example, if two bytes are needed to reach the block size, both padding bytes will have a value of 0x02.

During decryption, the receiver knows the number of padding bytes based on the value of the last byte in the decrypted block. The padding bytes are then removed to obtain the original plaintext.

Padding is effective when the plaintext length is an exact multiple of the block size, but it introduces additional bytes to the ciphertext, which might impact certain protocols or applications where maintaining the exact plaintext length is crucial.

**Ciphertext Stealing**:

Ciphertext stealing is an alternative approach to handle the last partial block of plaintext without explicitly padding it. It is commonly used when the plaintext length is not a perfect multiple of the block size.

In ciphertext stealing, the last block of plaintext is divided into two parts: a truncated part and a stolen part.

The truncated part consists of the initial bytes of the last block that can fill up a complete block size. This truncated part is encrypted like any other block and becomes

the second-to-last block of ciphertext.

The stolen part consists of the remaining bytes in the last block. It is then combined with the second-to-last block of ciphertext to create the final block of ciphertext.

During decryption, the last block of ciphertext is decrypted as usual. The stolen part is separated from the decrypted last block, and it is combined with the second-to-last block of plaintext to recover the original message.

Ciphertext stealing allows for encryption and decryption of messages of varying lengths without explicitly padding the last block. It ensures that the ciphertext remains the same length as the original plaintext, which can be desirable in certain scenarios.

Figure (Figure 2) below illustrates the CBC-CS1-Encrypt algorithm for the case that $P_n^*$ is a partial block, i.e., d $\leq$ b. The bolded rectangles contain the inputs and outputs. The dotted rectangles provide alternate representations of two blocks in order to illustrate the role of the "stolen" ciphertext. In particular, the string of the b-d rightmost bits of $C_{n-1}$ , denoted $P_{n-1}^{**}$, becomes the padding for the input block to the final invocation of the block cipher within the execution of CBC mode. The ciphertext that is returned in Step 5 above omits $C_{n-1}^{**}$, because it can be recovered from $C_n$ during decryption.

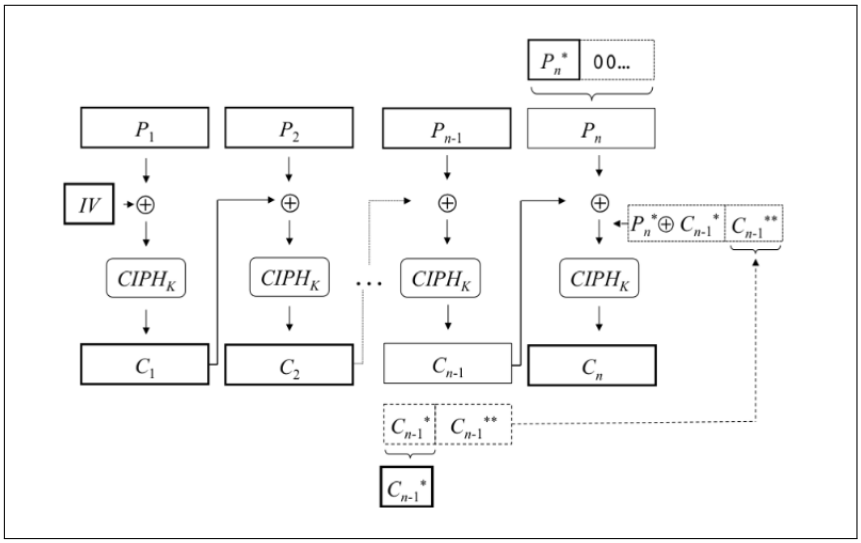


Figure 2: CBC CS1 Encrypt

## 2.6 Stream Ciphers

Idea: try to simulate one-time pad

A stream cipher encrypts a continuous string of binary digits by applying time-varying transformations on plaintext data. Therefore, this type of encryption works bit-by-bit, using keystreams to generate ciphertext for arbitrary lengths of plain text messages. The cipher combines a key (128/256 bits) and a nonce digit (64-128 bits) to produce the **keystream** — a pseudorandom number XORed with the plaintext to produce ciphertext. While the key and the nonce can be reused, the keystream has to be unique for each encryption iteration to ensure security. Stream encryption ciphers achieve this using feedback shift registers to generate a **unique nonce (number used only once) to create the keystream**.

Encryption schemes that use stream ciphers are less likely to propagate system-wide errors since an error in the translation of one bit does not typically affect the entire plaintext block. Stream encryption also occurs in a **linear, continuous manner**, making it simpler and faster to implement. On the other hand, stream ciphers lack diffusion since each plaintext digit is mapped to one ciphertext output. Additionally, they do not validate authenticity, making them vulnerable to insertions. If hackers break the encryption algorithm, they can insert or modify the encrypted message without detection. Stream ciphers are mainly used to encrypt data in applications where the amount of plain text cannot be determined and in low latency use-cases.

## 2.7 Types of Stream Ciphers

Stream ciphers fall into two categories:

**Synchronous Stream Ciphers:**

- In a synchronous stream cipher, the keystream block is generated independently of the previous ciphertext and plaintext messages. This means that **each keystream block is generated based only on the key** and does not depend on any previous blocks.
- The most common stream cipher modes use pseudorandom number generators (PRNGs) to create a string of bits, which is combined with the key to form the keystream.
- The keystream is then XORed with the plaintext to generate the ciphertext.

Self-Synchronizing/Asynchronous Stream Ciphers:

- A self-synchronizing stream cipher, also known as ciphertext autokey, **generates the keystream block as a function of both the symmetric key and the fixed-size (N-bits) previous ciphertext block**.
- By altering the ciphertext, the content of the next keystream is changed. This property allows self-synchronizing ciphers to detect active attacks because any modification to the ciphertext will affect the decryption of subsequent blocks, making it easier to detect tampering.
- Asynchronous stream ciphers also provide limited error propagation. If there is a single-digit error in the ciphertext, it can affect at most N bits (the size of the previous ciphertext block) in the next keystream block

.

## 2.8 Stream Ciphers in practice

popular encryption schemes that use stream ciphers include:

### 2.8.1 A5/1

### 2.8.2 Rivest Cipher (RC4)

### 2.8.3 Salsa20

## 2.9 Block Ciphers

Block ciphers convert data in plaintext into ciphertext in fixed-size blocks. The block size generally depends on the encryption scheme and is usually in octaves (64-bit or 128-bit blocks). If the plaintext length is not a multiple of 8, the encryption scheme uses **padding** to ensure complete blocks. For instance, to perform 128-bit encryption on a 150-bit plaintext, the encryption scheme provides two blocks, 1 with 128 bits and one with the 22 bits left. 106 Redundant bits are added to the last block to make the entire block equal to the encryption scheme's ciphertext block size.

While Block ciphers use symmetric keys and algorithms to perform data encryption and decryption, they also require an **initialization vector (IV)** to function. An initialization vector is a pseudorandom or random sequence of characters used to encrypt the first block of characters in the plaintext block. The resultant ciphertext for the first block of characters acts as the initialization vector for the subsequent blocks. Therefore, the symmetric cipher produces a unique ciphertext block for each iteration while the IV is transmitted along with the symmetric key and does not require encryption.

Block encryption algorithms offer **high diffusion**; that is, if a single plaintext block were subjected to multiple encryption iterations, it resulted in a unique ciphertext block for each iteration. This makes the encryption scheme relatively tamper-proof since it is difficult for malicious actors to insert symbols into a data block without detection. On the other hand, block ciphers have a **high error propagation rate** since a bit of change in the original plaintext results in entirely different ciphertext blocks.

## 2.10 Block Ciphers in practice

### 2.10.1 Data Encryption Standard (DES)

DES is a symmetric block cipher using 64 bit blocks and 56 bits key.
The choice of using 56 bits for keys in DES was a compromise between security

and practicality. While a longer key would provide stronger security, the designers of DES also needed to consider the limitations of computing technology at the time. The inclusion of 8 parity bits reduced the effective key size to 56 bits, which some experts even then considered inadequate for robust security. However, the decision to use a 56-bit key was likely influenced by a balance between security requirements and the feasibility of implementing and processing longer keys with the available hardware during that era.

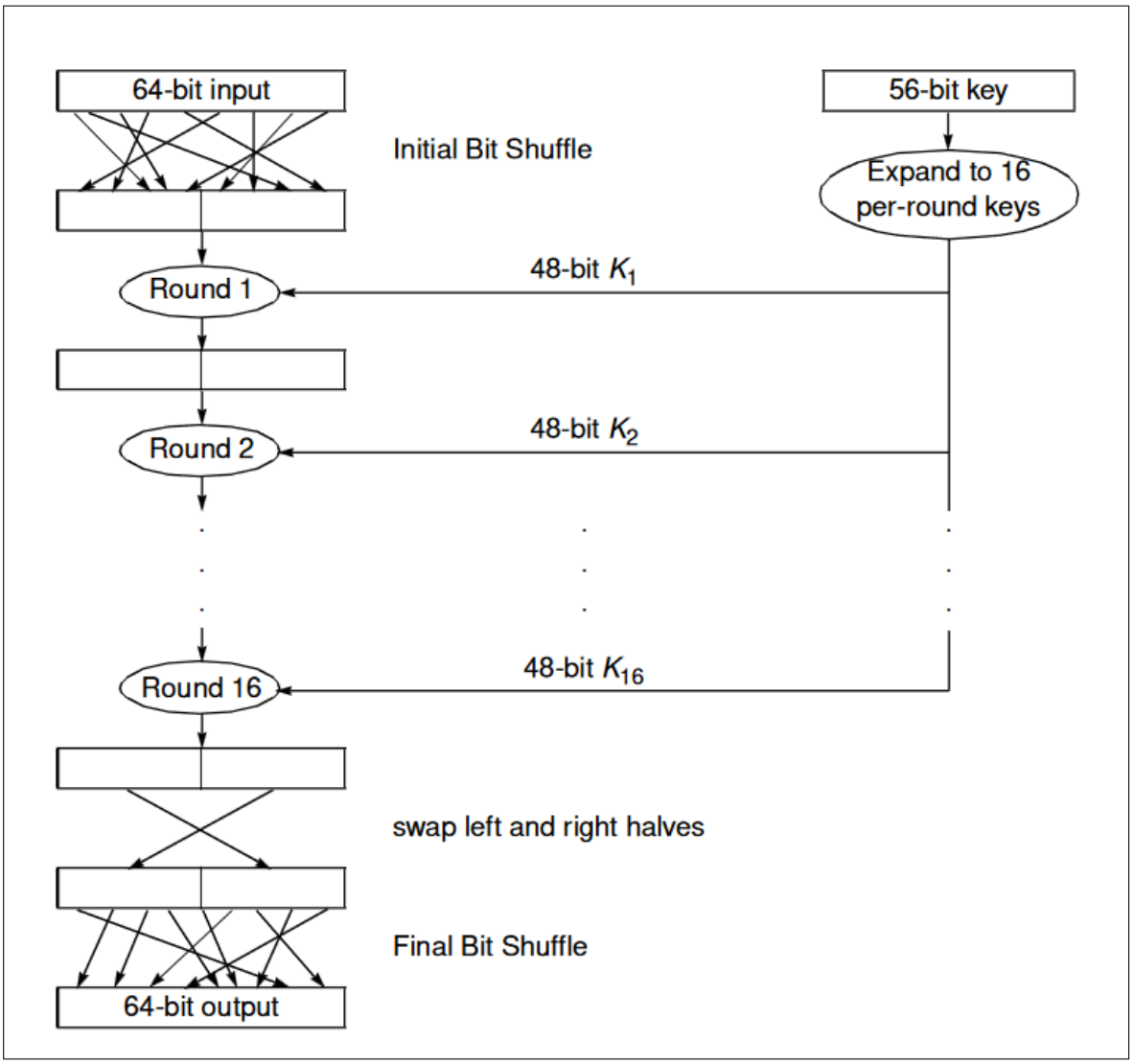


Figure 3: Basic Structure of DE

### 2.10.2 DES Overview

First, the 64-bit input is subjected to an initial bit-shuffle. Then, 56-bit key is expanded into sixteen 48-bit per-round keys by taking a 48-bit subset of the 56-bit

input key for each of the keys. Each round takes a 64-bit input with a 48-bit key and produces a 64-bit cipher block. After the sixteenth round, the output has its halves swapped and then is subjected to another bit-shuffle which happens to be the inverse of the initial bit-shuffle. This swapping after the final round does not add any cryptographic strength but has a side benefit. As noted in Feistel Ciphers, Swapping the output make the encryption and description identical except for the key schedule

### 2.10.3 DES Complementary Notes

**The Mangler Function**

**Undesirable Symmetries**

**DES Variants**

Add DES Mangler Function

Add DES Undesirab

Add DES Variantss

### 2.10.4 Advanced Encryption Standard (AES)

AES is a symmetric block cipher using 128 bit blocks and 128, 192 or 256 bits key with the resulting variants called AES-128, AES-192, and AES-256.

AES is similar to DES in that there is a key expansion algorithm which takes the key as an input and expands it into a bunch of round keys, and the algorithm executes a series of rounds that mangle a plaintext block into a ciphertext block. (Figure 4). With DES each round it takes a 64-bit input with a 48-bit key and outputs 64-bit that (along with the next round key) is fed to the next round. With AES, each round takes a 128-bit input and a 128-bit key and produces a 128-bit output which is the input to the next round. Unlike DES, AES is not Feistel cipher. (The Feistel cipher structure is characterized by the division of the plaintext into two halves and the repeated application of a round function that involves both halves. Read more here: 2.4)

In accordance to do as many rounds as needed to make the exhaustive search cheaper than any other form of cryptanalysis, AES does more round with bigger keys. AES-128 has 10 rounds, AES-196 has 12 rounds and AES-256 has 14 rounds. If AES had a 64-bit version, it would have eight rounds, which would be comparable to the sixteen rounds in DES.

Another diffference between DES and AES is that DES operates on bits where AES

operates on octets (bytes)

In AES, the key (128, 192, or 256 bits) is expanded into a series of 128-bit round keys, where there is one more round key than there are rounds. AES encryption or decryption consists of $\oplus$ing a round key into the intermediate value at the beginning of the process, at the end of the process, and between each pair of rounds.

In terms of Substitution and Permutation and cryptographic transformations, DES relies on predefined tables for S-boxes and P-boxes, which may appear arbitrary without understanding their rationale. In contrast, AES provides simpler mathematical formulas for substitutions and permutations, openly stating the reasons behind their design choices.

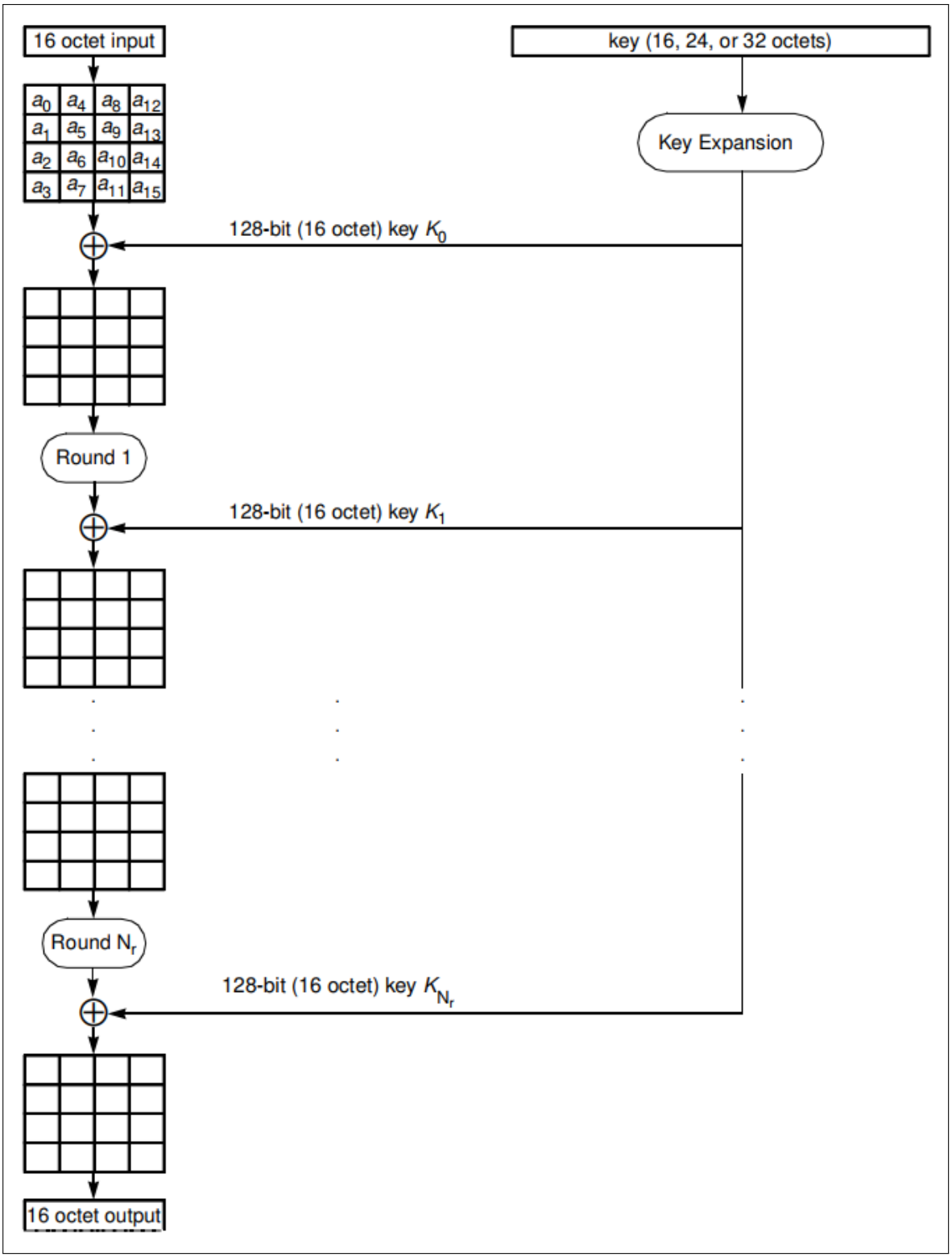


Figure 4: Basic Structure of AE

### 2.10.5 AES Complementary Notes

**(AES's S-box)**

**Key expansion**

Add AES Box

Add AES Key expansion

## 2.11 Block Ciphers Modes of Operations

Block ciphers operate on fixed-length blocks, typically 64 or 128 bits. The reason for using fixed-length blocks is twofold: first, messages can have varying lengths, and second, encrypting the same plaintext with the same key should always produce the same output.

To address the need for encrypting messages of **arbitrary** lengths, several modes of operation have been invented. These modes allow block ciphers to provide confidentiality for messages of any length. These modes define how the block cipher is applied to multiple blocks and how they are combined to encrypt or decrypt the entire message.

### 2.11.1 Electronic Cook Book (ECB)

a full 128 bits), and encrypt each block with the secret key (Figure 5). The other side receives the encrypted blocks and decrypts each block in turn to recover the original message (Figure 6).

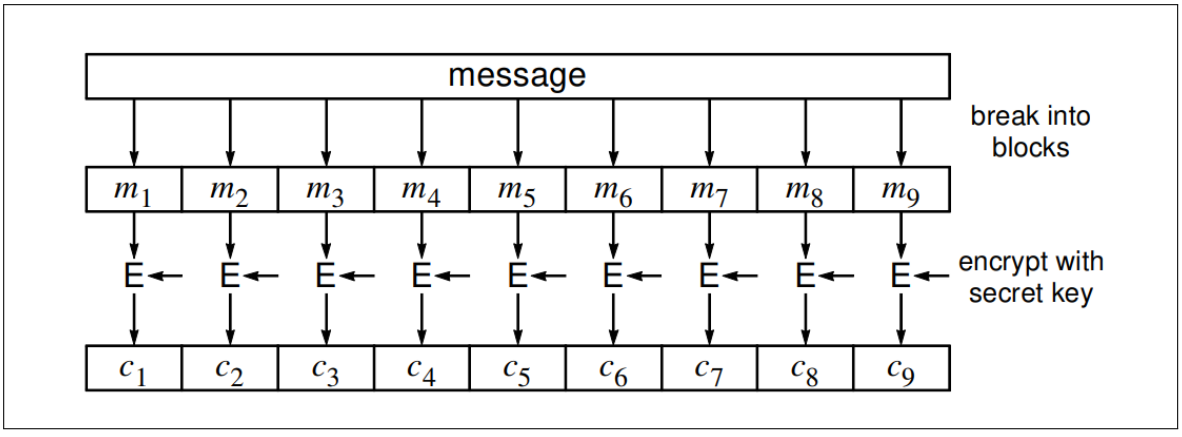


Figure 5: Electronic Code Book Encryption

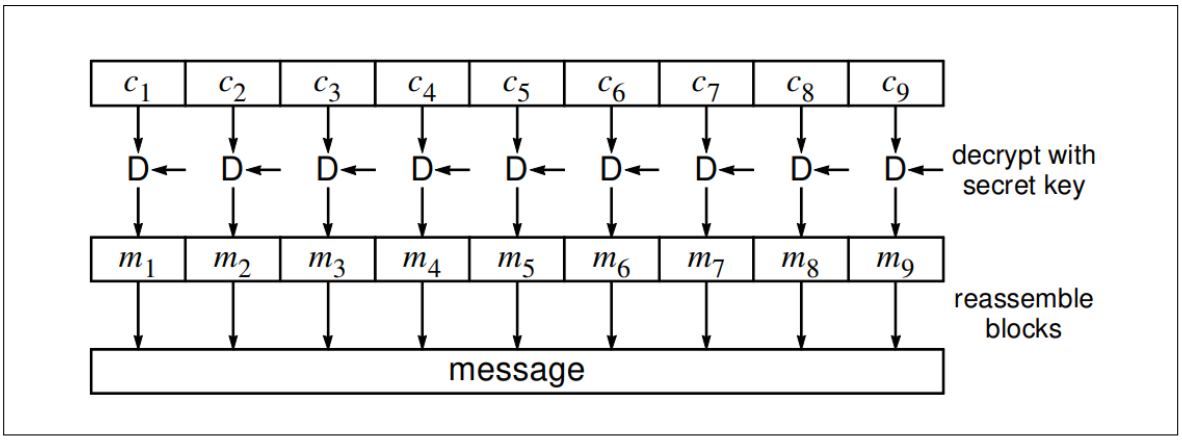


Figure 6: Electronic Code Book Decryption

ECB has two serious flaws. Patterns in the ciphertext, such as repeated blocks, leak information, and nothing prevents someone from rearranging, deleting, modifying, or duplicating blocks.

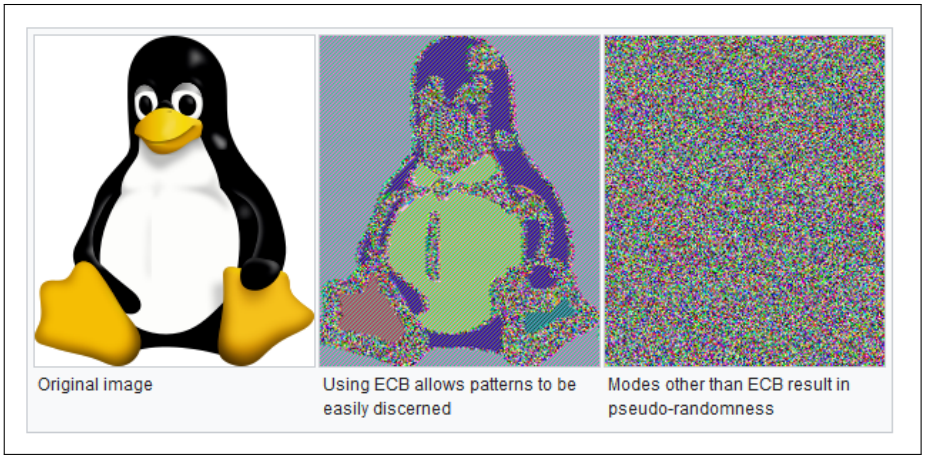


Figure 7: ECB Lack of Diffusion

### 2.11.2 Cipher Block Chaining (CBC)

CBC generates its own "random numbers" for all but the first block. It uses $c_i$ as $r_{i+1}$. In other words, it takes the previous block of ciphertext and uses that as the "random number" that will be $\oplus$'d into the next plaintext block. To avoid leaking the information that two messages encrypted with the same key have the same first plaintext blocks, CBC selects one random number that gets $\oplus$'d into the first block of plaintext and transmits it along with the data.

This initial random number is known as an **IV (initialization vector)** A randomly chosen IV guarantees that even if the same message is sent repeatedly, the ciphertext

will be completely different each time.

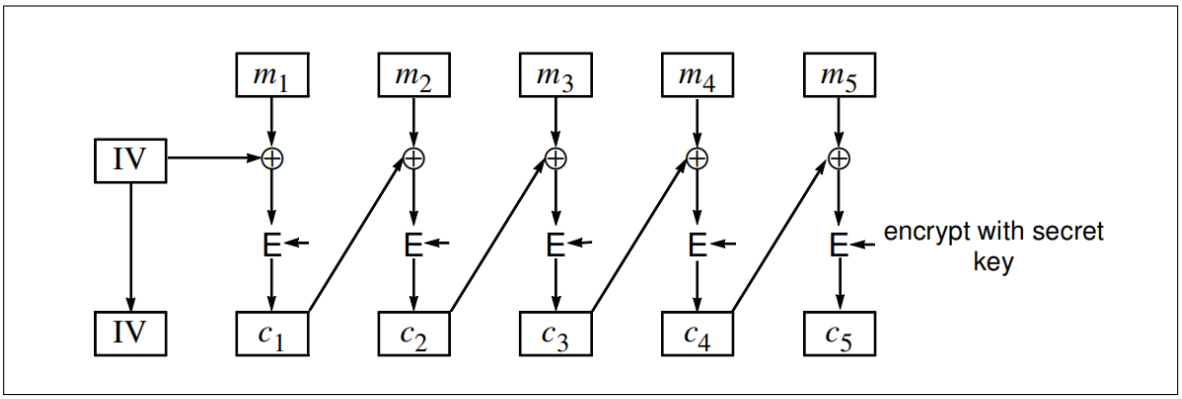


Figure 8: Cipher Block Chaining Encryption

**Properties of CBC:**

- What would happen if they changed a block of the ciphertext, say, the value of ciphertext block $c_n$? Changing $c_n$ has a predictable effect on $m_{n+1}$ because $c_n$ gets $\oplus$'d with the decrypted $c_{n+1}$ to yield $m_{n+1}$ For instance, changing bit 3 of $c_n$ changes bit 3 of $m_{n+1}$. Modifying $c_n$ also garbles block $m_n$ to some unpredictable value.
- Asynchronous stream cipher
- Conceals plaintext patterns
- Plaintext cannot be easily manipulated
- No parallel implementation known for encryption
- message must be padded to a multiple of the cipher block size (or we may use Ciphertext Stealing)
- a plaintext can be recovered from just two adjacent blocks of ciphertext
  as a consequence, decryption can be parallelized. usually a message is encrypted once, but decrypted many times

### 2.11.3 XEX (XOR Encrypt XOR)

Write dow
XEX

### 2.11.4 XTS (XEX with Ciphertext Stealing)

XTS is a clever variant of XEX that encrypts multiblock messages that need not be a multiple of the cryptographic blocksize while still keeping the ciphertext the same size as the plaintext. XTS accomplishes this goal (of being length-preserving despite a message not being a multiple of the cryptographic blocksize) through a very clever but somewhat complicated trick known as ciphertext stealing.

Ciphertext stealing pads the final block $m_n$ with as many of the bits of the previous block of ciphertext ($c_{n-1}$) as necessary to make block $m_n$ be full sized (in our example, $128 - 93 = 35$ bits need to be stolen from $c_{n-1}$). The padded block $m_n$ is then encrypted. But the ciphertext is now longer than the plaintext, since the last block of ciphertext is now a full-sized block. To solve that problem, we swap ciphertext blocks $c_n$ and $c_{n-1}$ and truncate the final ciphertext block (which was the encryption of block $m_{n-1}$) to be the size of the original block $m_n$. In our example, where the original block mn was 93 bits long, the final ciphertext block will be 93 bits. Now the ciphertext is the same size as the message, but how do we decrypt either of the last two blocks? To obtain block $m_n$, decrypt $c_{n-1}$ (using implicit IVs and Bmods associated with block $m_n$). The result will be plaintext $m_n$ with appended padding, but you need to know how much of the result is message and how much is padding. The size of plaintext block $m_n$ will be the size of the final ciphertext block (in our example, 93 bits). The padding (the remaining 128-93=35 bits) is stolen ciphertext from $c_{n-1}$.

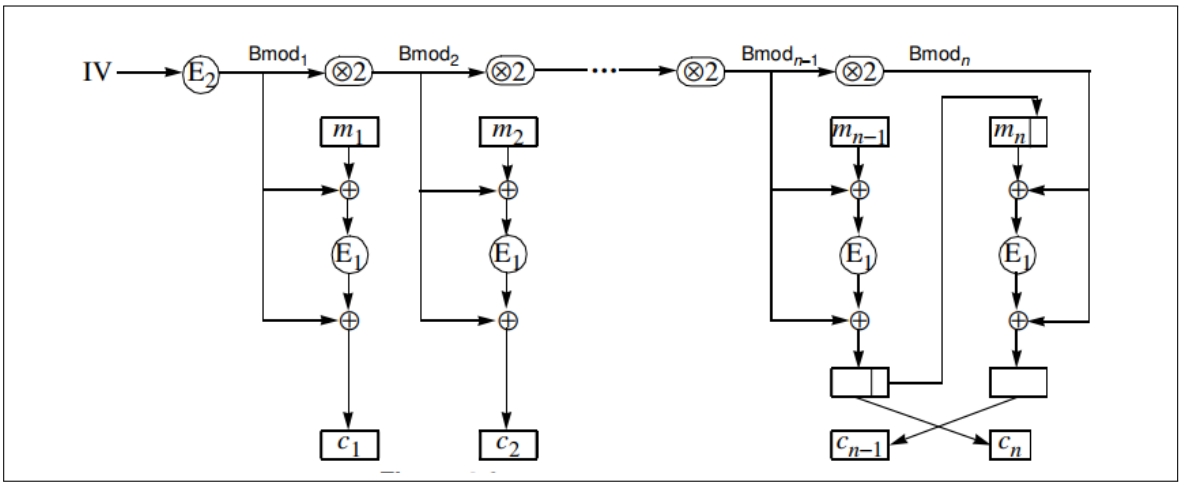


Figure 9: XTS Mode Decryption

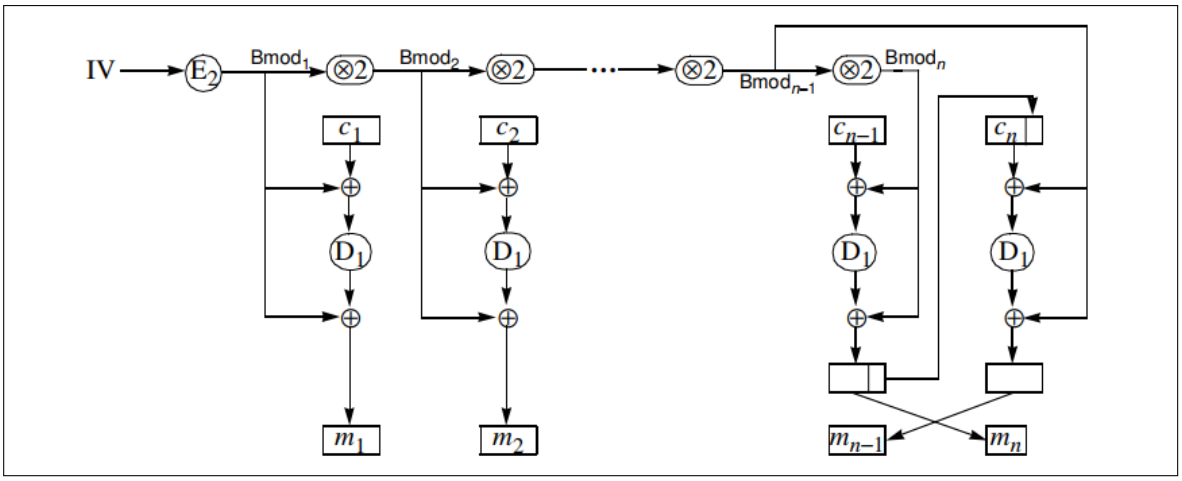


Figure 10: XTS Mode Decryption

### 2.11.5 Cipher Feedback (CFB)

CFB (short for cipher feedback) is an AES block cipher mode like CBC, makes a block cipher into an asynchronous stream cipher (i.e., supports some re-synchronizing after error, if input to encryptor is given through a shift-register) in the sense that for the encryption of a block, $B_i$, the cipher of the previous block, $C_{i-1}$ is required. CFB also makes use of an initialization vector like CBC. The main difference is that in CFB, the ciphertext block of the previous block is encrypted first and then XOR-ed with the block in focus.

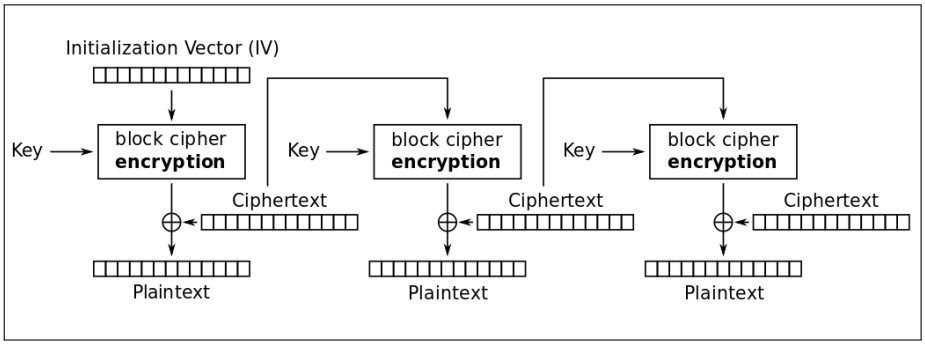


Figure 11: Cipher Feedback (CFB)

### 2.11.6 Output Feedback (OFB)

The output feedback (OFB) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext. Just as with other stream ciphers, flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property

allows many error-correcting codes to function normally even when applied before encryption.

**Properties of OFB:**

- Synchronous stream cipher
- Errors in ciphertext do not propagate
- Pre-processing is possible
- Conceals plaintext patterns
- No parallel implementation known
- Active attacks by manipulating plaintext are possible

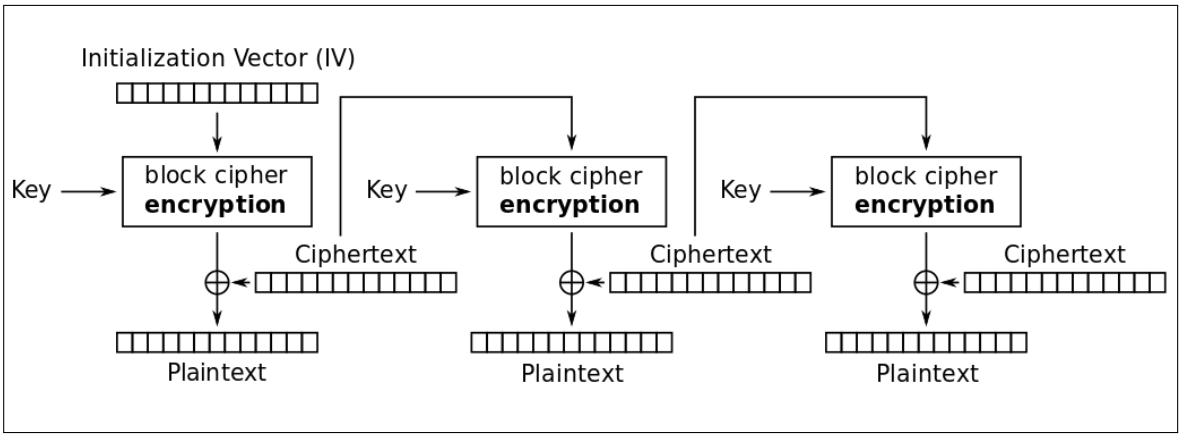


Figure 12: Feedback (OFB)

## 2.12 Ensuring Privacy and Integrity Together

### 2.12.1 CCM (Counter with CBC-MAC)

also known as Integer Counter Mode (ICM) and Segmented Integer Counter (SIC) mode

- turns a block cipher into a stream cipher: it generates the next keystream block by encrypting successive values of a ”counter”
- counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual counter is the simplest and most popular.

- the usage of a simple deterministic input function raised controversial discussions
- has similar characteristics to OFB, but also allows a random access property during decryption
- well suited to operation on a multi-processor machine where blocks can be encrypted in parallel

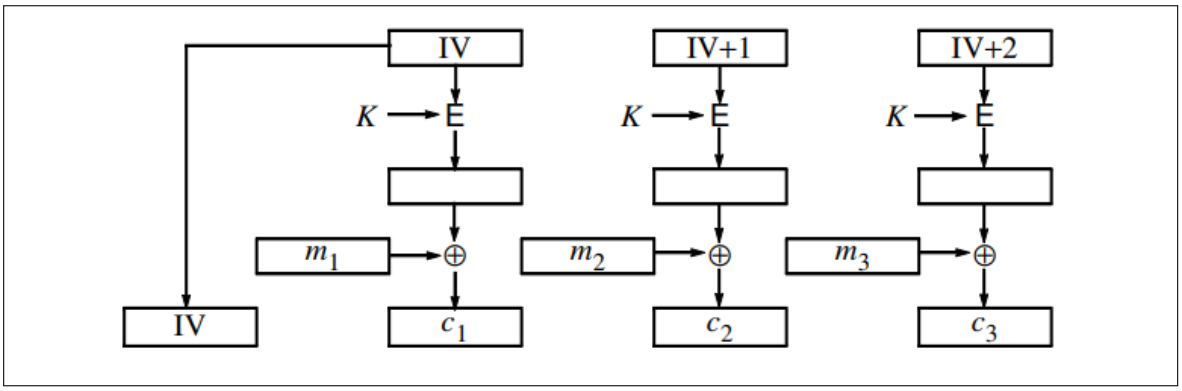


Figure 13: Counter Mode (CTR)

### 2.12.2 GCM (Galois/Counter Mode)

# 3 Data Integrity

A secret key system can be used to generate a cryptographic integrity check known as a MAC (Message Authentication Code). MACs are used to protect the data from modification in transit.
In this chapter, we'll focus on MACs based on secret key encryption functions and MACs based on hash functions.

## 3.1 Definition

- *Authentication Algorithm* (A): $A$ is the algorithm used to generate an authentication tag for a given message using a secret authentication key (k). It takes the message (m) as input and produces the authentication tag $A_k(m)$.
- *Verification Algorithm* (V): $V$ is the algorithm used to verify the authenticity and integrity of a message. It takes the received message $(m)$ and its corresponding authentication tag $(A_k(m))$ as input and outputs either *accept* or *reject* based on whether the authentication tag is valid or not.
- *Authentication Key* $(k)$: The authentication key $(k)$ is a secret key shared between Alice and Bob. It is used by the authentication algorithm $(A)$ to generate the authentication tag and by the verification algorithm $(V)$ to validate the tag.
- *Message Space (usually binary strings)*: The message space refers to the set of possible messages that can be authenticated using CBC-MAC. Typically, these messages are represented as binary strings.
- *Message Format*: Every message exchanged between Alice and Bob is a pair $(m, A_k(m))$, where $m$ is the actual message and $A_k(m)$ is its corresponding authentication tag generated by applying the authentication algorithm $(A)$ with the authentication key $(k)$.

## 3.2 MAC based on CBC Mode Encryption

CBC-MAC computes a MAC of a message using key K. It encrypts the message in CBC mode, using key K, and uses the last block (called the **residue**, see (Figure

14)) as the MAC for the message. If an attacker modifies any portion of a message, the residue will no longer be the correct value (except with probability 1 in $2^{128}$, assuming 128-bit blocks).

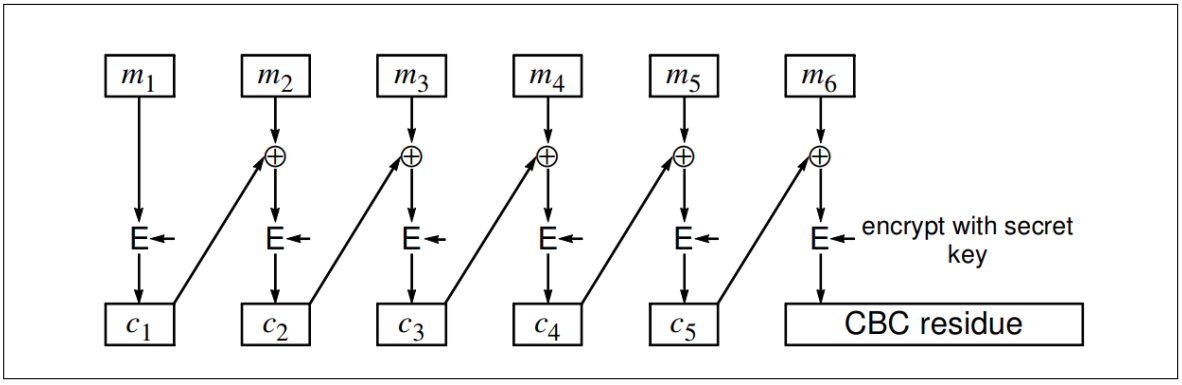


Figure 14: Cipher Block Chaining Residue

## 3.3 Vulnerabilities of CBC-MAC

### 3.3.1 Initialization Vector or IV

When we are working on CBC-MAC or Cipher Block Chaining Message Authentication Code, we generally use the Initialization Vector or IV as zero.
When we use the IV or Initialization Vector as zero, a problem arises. Let's say there are two known messages named 'msg1' and 'msg2'. These two messages will generate two signatures called 'sig1' and 'sig2' independently. Finally,

- E (msg1 XOR 0) = sig1
- E (msg2 XOR 0) = sig2

Now, a message which consists of msg1 and msg2 will generate two signatures. Let's name the concatenated message as msg3 and the signals generated as sig31 and sig32.

- E (msg1 XOR 0) = sig31 = sig1
- E (msg2 XOR sig1) = sig32

We can calculate this without even knowing the key of the encryption.

### 3.3.2 Fixed and Variable-length message

If the block cipher used is secure (meaning that it is a pseudorandom permutation), then CBC-MAC is secure for fixed-length messages. However, by itself, it is not secure for variable-length messages. Thus, any single key must only be used for messages of a fixed and known length. This is because an attacker who knows the correct authentication tag (i.e. CBC-MAC) pairs for two messages (m,t) and (m',t') can generate a third message m" whose CBC-MAC will also be t'. This is simply done by XORing the first block of m' with t and then concatenating m with this modified m'; i.e., by making $m" = m \parallel (m'_1 \oplus t) \parallel m'_2 \parallel \ldots \parallel m'_x$
There are three main ways of modifying CBC-MAC so that it is secure for variable length messages:

1. Input-length key separation
2. Length-prepending
3. Encrypt last block.
4. Use HMACs or CMACs.

## 3.4 MAC based on cryptographic hash

A hash function inputs an arbitrary-sized bitstring and outputs a fixed-size bitstring, ideally so that all output values are equally likely. A cryptographic hash (also known as a message digest) has some extra security properties:
**Preimage Resistance**: It should be computationally infeasible to find a message that has a given pre-specified hash.
**Collision Resistance**: It should be computationally infeasible to find two messages that have the same hash.
**Second Preimage Resistance**: It should be computationally infeasible to find a second message that has the same hash as a given message.

Strong collision resistance: A hash function is said to have strong collision resistance if it is computationally infeasible to find **any two distinct inputs** that produce the

same hash output. In other words, given a hash value h, it is difficult to find any two different messages m1 and m2 such that hash(m1) = hash(m2).
Weak collision resistance (also known as second preimage resistance): A hash function is said to have weak collision resistance if it is computationally infeasible **to find a second input message that produces the same hash output as a given fixed input message.** In simpler terms, it is difficult to find a different message m2 such that hash(m1) = hash(m2) for a known message m1.

The strong collision resistance property implies the weak collision resistance property (strong $\rightarrow$ weak). For the proof we show $not$ Weak $\rightarrow$ $not$ Strong If a hash function is strongly collision resistant, it automatically implies that it is weakly collision resistant. If it were possible to find a second preimage for a fixed input message, it would effectively mean finding a collision between the fixed input message and the second preimage, contradicting the strong collision resistance assumption. Therefore, a hash function that exhibits strong collision resistance will inherently exhibit weak collision resistance as well.
As an example, the mod s function (% s), where s is a large random prime number, is not suitable for cryptographic purposes due to the following reasons:
a) Predictability: The mod s function is deterministic and has a predictable output. Given an input x, the output of x % s will always be the remainder when x is divided by s. This predictability makes it vulnerable to attacks, as an attacker can calculate the output for various inputs and try to identify patterns or predict future outputs.
b) Lack of diffusion: Cryptographic hash functions need to exhibit the property of diffusion, which means that a small change in the input should result in a significantly different output. However, the mod s function does not provide diffusion. If two inputs x1 and x2 are very close to each other (e.g., x2 = x1 + 1), their outputs x1 % s and x2 % s will also be very close to each other, thus lacking the necessary diffusion property.
Limited output space: The output of the mod s function is constrained to the range from 0 to s-1, where s is the prime number. This limited output space makes it

vulnerable to brute-force attacks, where an attacker can exhaustively try different inputs until finding a collision or a preimage.

Lack of resistance to mathematical attacks: The mod s function is vulnerable to various mathematical attacks, such as modular arithmetic properties or number-theoretic attacks. These attacks exploit the specific properties of modular arithmetic and the structure of prime numbers, making the function unsuitable for cryptographic purposes.

## 3.5 The Birthday Paradox

If there are 23 or more people in a room, the odds are better than 50% that two of them will have the same birthday. Analyzing this parlor trick can give us some insight into cryptography.

Let's do this in a slightly more general way. Let's assume $n$ inputs (which would be humans in the birthday example), $k$ possible outputs, and an unpredictable mapping from input to output. With $n$ inputs, there are $\frac{n(n-1)}{2}$ pairs of inputs. For each pair, there's a probability of $\frac{1}{k}$ of both inputs producing the same output value. Therefore, you'll need about $\frac{k}{2}$ pairs in order for the probability to be about $\frac{1}{2}$ that you'll find a matching pair. That means that if $n$ is greater than $\frac{k}{2}$, there's a good chance of finding a matching pair.

## 3.6 MACs based on Hash Functions

## 3.7 SHA-1

## 3.8 Authenticated Encryption (AE)

Authenticated Encryption (AE) is an encryption scheme which simultaneously assures the data confidentiality (also known as privacy: encrypted message is impossible to understand without the knowledge of a secret key) and authenticity (in other words, it is unforgeable: the encrypted message includes an authentication tag that the sender can calculate only if she possesses the secret key)
Many (but not all) AE schemes allow the message to contain ”associated data” (AD) which is not made confidential, but its integrity is protected (i.e., it is readable, but tampering with it will be detected). A typical example is the header of a network packet that contains its destination address. To properly route the packet, all intermediate nodes in the message path need to know the destination, but for security reasons they cannot possess the secret key. Schemes that allow associated data provide authenticated encryption with associated data, or AEAD.

### 3.8.1 Encrypt-then-MAC (E$t$M)

The plaintext is first encrypted, then a MAC is produced based on the resulting ciphertext. The ciphertext and its MAC are sent together. Used in, e.g., IPsec. This is the only method which can reach the highest definition of security in AE, but this can only be achieved when the MAC used is ”strongly unforgeable”. Note that key separation is mandatory (distinct keys must be used for encryption and for the keyed hash), otherwise it is potentially insecure depending on the specific encryption method and hash function used.

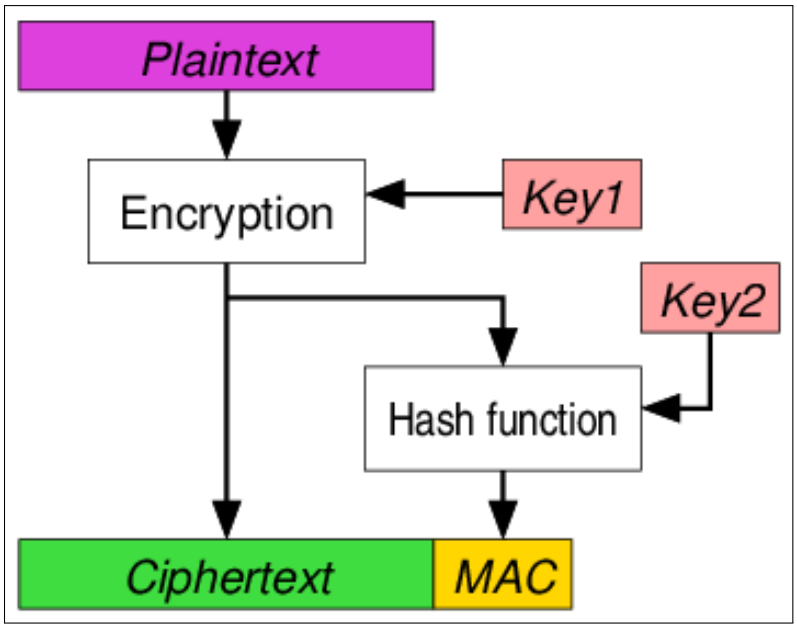


Figure 15: Authenticated Encryption EtM

### 3.8.2 Encrypt-and-MAC (E&M)

A MAC is produced based on the plaintext, and the plaintext is encrypted without the MAC. The plaintext's MAC and the ciphertext are sent together. Used in, e.g., SSH. Even though the E&M approach has not been proved to be strongly unforgeable in itself, it is possible to apply some minor modifications to SSH to make it strongly unforgeable despite the approach.

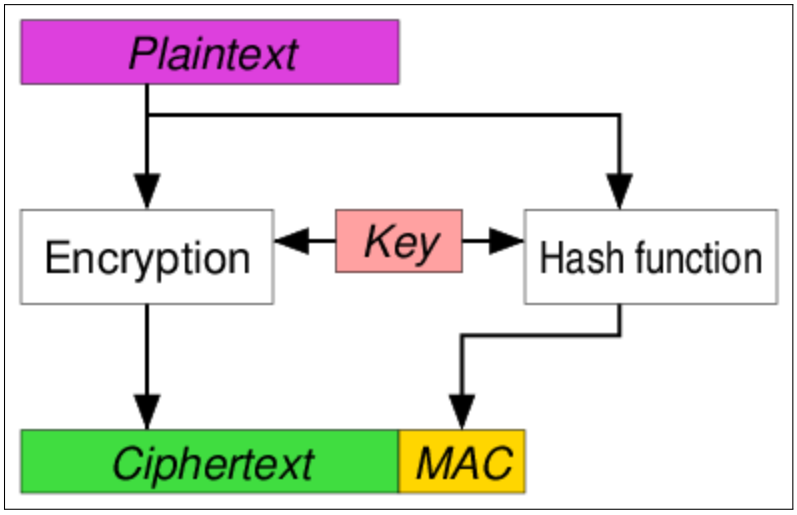


Figure 16: Authenticated Encryption EaM

### 3.8.3 MAC-then-Encrypt (M$t$E)

A MAC is produced based on the plaintext, then the plaintext and MAC are together encrypted to produce a ciphertext based on both. The ciphertext (containing an encrypted MAC) is sent. AEAD is used in SSL/TLS. Even though the MtE approach has not been proven to be strongly unforgeable in itself,[15] the SSL/TLS implementation has been proven to be strongly unforgeable by Krawczyk who showed that SSL/TLS was, in fact, secure because of the encoding used alongside the MtE mechanism. Despite the theoretical security, deeper analysis of SSL/TLS modeled the protection as MAC-then-pad-then-encrypt, i.e. the plaintext is first padded to the block size of the encryption function. Padding errors often result in the detectable errors on the recipient's side, which in turn lead to padding oracle attacks, such as Lucky Thirteen.

AEAD is a cryptographic construction that combines both encryption and authentication in a single operation.

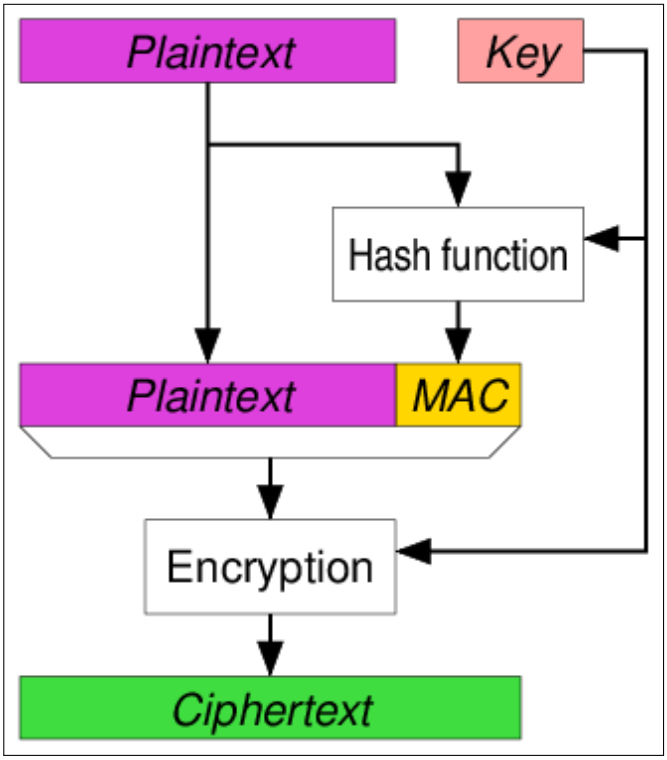


Figure 17: Authenticated Encryption MtE

# 4 Public Key Cryptography

In this chapter we'll have asymmetric techniques that secret and public keys are used instead of a single shared key. In this chapter we'll have:

- RSA, which does encryption and digital signatures
- ElGamal and DSA and ECDSA (elliptic curve DSA), which do digital signatures
- Diffie-Hellman and ECDH (elliptic curve Diffie-Hellman), which establish a shared secret and can also do encryption

The thing that all public key algorithms have in common is the concept that a principal has a pair of related quantities, one private and one public

## 4.1 RSA

RSA is named after its inventors, Rivest, Shamir, and Adleman. The real premise behind RSA's security is the assumption that factoring a big number is hard.

### 4.1.1 Example 1

First, we pick two big prime numbers $p$ and $q$.

1. $p = 2, q = 7$

   Then we will calculate the multiplication of q and p.

2. $p \times q = 14$

   Now, we calculate $\phi$ function: $\phi = (p-1)(q-1)$

3. $\phi = 1 \times 6 = 6$

   Now, we need to choose a value $e$ having the following conditions:

   - $1 < e < \phi(n)$
   - e should be prime with respect to $n$ and $\phi(n)$

4. $e = 5$

   Finally, we choose a number under the following condition: $d \times e(mod\phi(n)) = 1$

5. $d = 11$

   Finally, we have the following numbers:n = 14 ,e = 5 ,d = 11

   Based on RSA, we use $e$ and $n$ as the public-key and $e$ and $d$ as the private-key.

Now, assume we want to send the number $2$ to our friend. Having our fiend's public-key which has been sent to use before (5,14) we first calculate
$2^5 mod 14 = 4$
Now, our encrypted message is $4$. We send $4$ to our friend.
Our friend, with his private-key (11, 14) decrypts the message:
$4^{11} mod 14 = 2$

### 4.1.2 Example 2

### 4.1.3 Properties

- key length is variable

  The longer, the higher security (4096 bits is common)

- Block size also variable

  but $|plaintext| \leq |N|$ (in practice, for avoiding weak cases, $|plaintext| < |N|$)

  $|plaintext| = |N|$

- Slower than DES

  not used in practice for encrypting long messages

  mostly used to encrypt a symmetric key, then used for encrypting the message

## 4.2 Diffie-Hellman

Diffie-Hellman allows two individuals (Alice and Bob) to agree on a shared key even though they can only exchange messages in public.

he security of Diffie-Hellman depends on the difficulty of solving the discrete log problem, which can be informally stated as, "If you know $g$, $p$, and $g^x modp$, what is x?" In other words, what exponent did you have to raise $g$ to, $modp$, to get $g^x$?

So, there are integers $p$ and $g$, where $p$ is a large prime (say 2048 bits) and $g$ is a number less than p with some restrictions that aren't too important for a basic understanding of the algorithm.

The values $p$ and $g$ are known beforehand and can be publicly known, or Alice and Bob can negotiate with each other across the crowded room.

Once Alice and Bob agree on a $p$ and $g$, each chooses a large, say, 512-bit number at random and keeps it secret. Let's call Alice's private Diffie-Hellman key $a$ and Bob's private Diffie-Hellman key $b$. Each raises $g$ to their private Diffie-Hellman number, $modp$, resulting in their public Diffie-Hellman value. So Alice computes $g^a modp$ and Bob computes $g^b modp$, and each informs the other. Finally, each raises the other side's public value to their own private value.

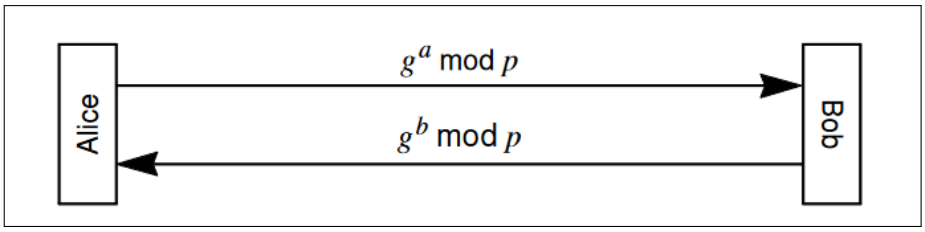


Figure 18: Diffie-Hellman Exchange

Alice raises $g^b modp$ to her private number $a$. Bob raises $g^a modp$ to his private number $b$. So Alice computes $(g^b modp)^a = g^{ba} modp$. Bob computes $(g^a modp)^b = g^{ab} modp$. And, of course, $g^{ba} modp = g^{ab} modp$.

Without knowing either Alice's private number $a$ or Bob's private number $b$, nobody else can calculate $g^{ab} modp$ in a reasonable amount of time even though they can see $g^a modp$ and $g^b modp$.

### 4.2.1 MITM (Meddler-in-the-Middle) Attack

Diffie-Hellman alone does not prevent MITM attacks, if we assume that Alice and Bob do not have credentials (such as public keys) for each other. As a result, this form of Diffie-Hellman is only secure against passive attacks where the intruder just watches the encrypted messages.

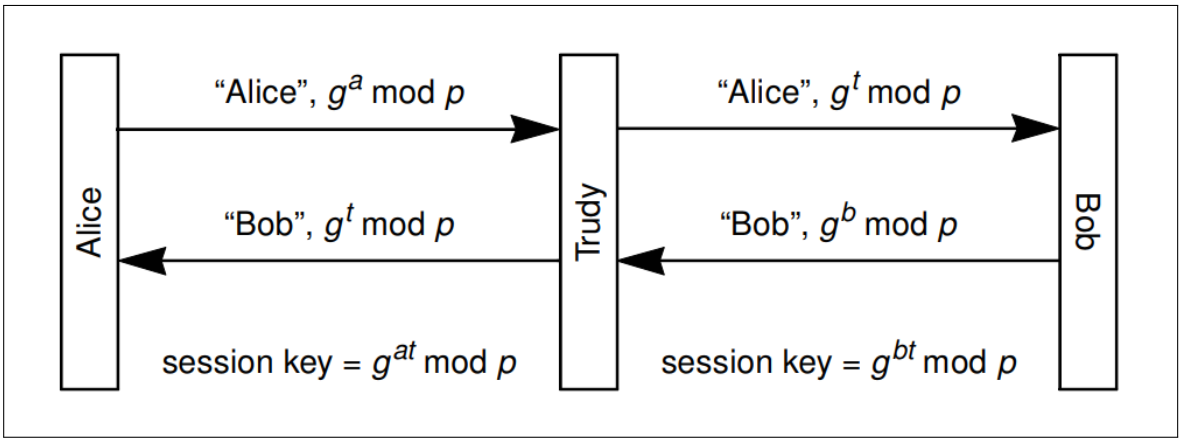


Figure 19: Diffie-Hellman with Trudy in the Middle

### 4.2.2 Defenses Against MITM Attack

- Having a personal permanent Diffie-Hellman value and do a handshake first.
- Use any authenticated Diffie-Hellman exchange techniques
- Use channel bindings

### 4.2.3 Safe Primes and the Small-Subgroup Attack

### 4.2.4 DF Properties

- DH is at most as strong as DL in $Z_p^*$.
- Formal equivalence unknown, though some partial results known.
- Despite 25 years of effort, still considered secure to date.
- Secret integers $a$ and $b$ are discarded at the end of the session, hence Diffie-Hellman key exchange achieves perfect forward secrecy, because no long-term private keying material exists to be disclosed
- Computation time is $O(\log^3 p)$

### 4.2.5 ElGamal Signatures

## 4.3 Digital Signatures

## 4.4 How Secure Are RSA and Diffie-Hellman?

## 4.5 Elliptic Curve Cryptography (ECC)

# 5 Cryptographically Secure Pseudo-Random Number Generators

# 6 Authentication

## 6.1 Passwords

### 6.1.1 Lamport's Hash

### 6.1.2 Strong Password Protocols

## 6.2 Biometric

## 6.3 Authentication by symmetric key

### 6.3.1 One way authentication using nonce (challenge-response)

**ONE-WAY AUTHENTICATION OF ALICE**

arg1

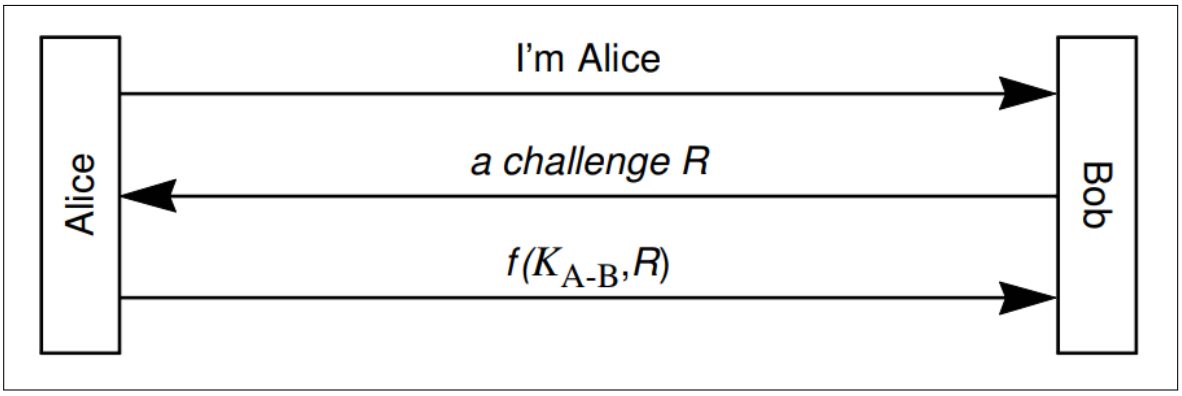


Figure 20: Bob Authenticates Alice Based on a Shared Secret $K_{(A-B)}$

### 6.3.2 One way authentication using timestamps

## 6.4 Authentication with trusted server

# 7 Secret Sharing

# 8 Access Control

# 9 Secure Protocols

# 10 Firewalls

# 11 Email Security

# 12 Web Technologies

# 13 Web Security

# 14 Web Tracking

# 15 Exams

## 15.1 9 January 2023

### 15.1.1 Ex 1. Hashing

1. What are pros and cons of keyed and unkeyed hashing?
2. Do cryptographic hash functions have less collision than non-cryptographic hashing functions? Elaborate.
3. Why are strongly collision-resistant functions also weakly collision-resistant?
4. Is the traditional binary representation of integers a cryptographic hash function? Explain.

**Solution:**

1. Pros and Cons of Keyed and Unkeyed Hashing:

   Keyed Hashing: Keyed hashing, also known as HMAC (Hash-based Message Authentication Code), provides an additional layer of security by incorporating a secret key into the hashing process. Pros of keyed hashing include:

   - Authentication: The use of a secret key ensures that only entities possessing the key can generate or verify the hash, providing authentication of the data.
   - Tamper Resistance: Modifying the data or the key will result in a different hash value, making it difficult for an attacker to tamper with the data without detection.

   Unkeyed Hashing: Unkeyed hashing, also referred to as non-keyed hashing, does not utilize a secret key in the hashing process. Pros of unkeyed hashing include:

   - Simplicity: Unkeyed hash functions are generally simpler to implement and use, as they do not require the management of secret keys.
   - Efficiency: Without the additional overhead of a secret key, unkeyed hash functions can be computationally more efficient.

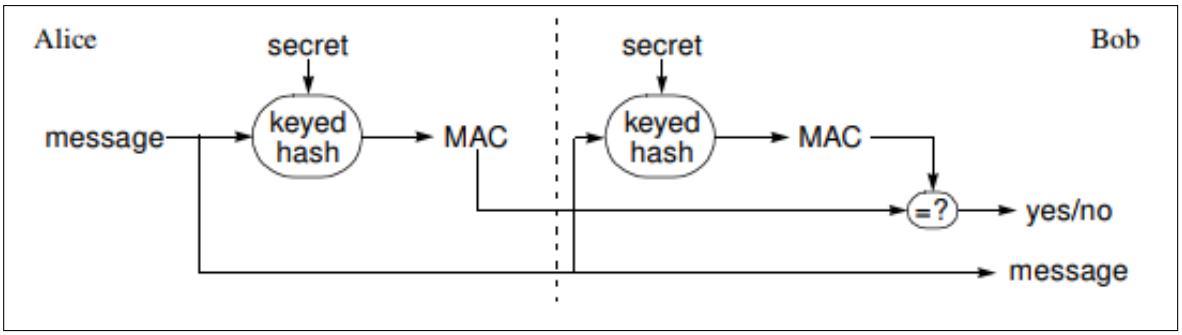


Figure 21: Keyed Hashed Map

2. Cryptographic Hash Functions and Collisions:

   Cryptographic hash functions are designed to have strong collision resistance, meaning it is computationally infeasible to find two different inputs that produce the same hash value (collision). Non-cryptographic hashing functions, on the other hand, may have a higher probability of collisions due to their primary focus on efficiency rather than collision resistance.

   While cryptographic hash functions aim to minimize collisions, it is important to note that collisions are still possible due to the pigeonhole principle, which states that if the number of possible inputs is larger than the number of possible hash outputs, collisions are inevitable. However, the strength of cryptographic hash functions lies in the difficulty of finding such collisions, making them highly useful for applications like data integrity verification, password storage, and digital signatures.

3. Strong Collision Resistance and Weak Collision Resistance:

   Strong collision resistance refers to the property of a hash function where it is computationally infeasible to find any two distinct inputs that produce the same hash value. Weak collision resistance, on the other hand, means that it is computationally difficult to find two specific inputs that result in the same hash value.

   The reason why strongly collision-resistant functions are also weakly collision-resistant is that if an attacker can find a specific pair of inputs that collide, they can demonstrate a general collision by applying additional modifications to those

inputs. In other words, if there exists a pair of inputs A and B such that hash(A) = hash(B), an attacker can create a new pair of inputs A' and B' such that hash(A') = hash(B') by manipulating A and B based on the collision found.

### 15.1.2 Ex 2. Key exchange

Describe at least two methods used for letting two parties exchange a secret key.

**Solution:**

1. Diffie-Hellman Key Exchange: Diffie-Hellman allows two parties, Alice and Bob, to agree on a shared secret key over an insecure channel. They each generate their own public-private key pairs and exchange public keys. By performing mathematical operations with their own private keys and the other party's public key, they can compute a shared secret that can be used as a symmetric key for encryption and decryption.

2. Key Exchange using Public Key Infrastructure (PKI): In a PKI-based key exchange, Alice and Bob each have a pair of public and private keys. They obtain each other's public keys through a trusted certificate authority (CA) or a secure key distribution protocol. Alice can encrypt the secret key with Bob's public key and send it to him. Bob can then decrypt the encrypted key using his private key to obtain the shared secret.

3. Key Transport using Symmetric Encryption: In this method, a trusted third party, called a Key Distribution Center (KDC), is involved. Alice and Bob both share a secret key with the KDC. When they want to exchange a secret key, they request the KDC to securely send the key to each other. The KDC uses symmetric encryption to protect the key during transmission.

4. Pre-shared Key (PSK) Exchange: In scenarios where Alice and Bob already share a secret key, such as a previously established secure channel or a shared secret obtained through an out-of-band method, they can directly use this shared key as the secret key for their communication.

5. Secure Sockets Layer/Transport Layer Security (SSL/TLS) Handshake: SSL/TLS protocols use asymmetric encryption and digital certificates to establish a secure channel between two parties. During the handshake process, the client and server exchange public keys and establish a shared secret key for subsequent symmetric encryption.

### 15.1.3 Ex 3. Authentication

1. Why do servers that authenticate users with Lamport hashing prefer that the user always use the same device?
2. With reference to password-based authentication, describe a (any) protocol to make client and server talk so that the client supplies the password but prevents replay and reflection attacks.
3. Can digital signature (whatever it is) authenticate a user of a web application? Discuss.

**Solution:**

1. Servers that authenticate users with Lamport hashing prefer that the user always use the same device because Lamport hashing is a one-time password scheme that relies on the secrecy of the password and the server's knowledge of the user's password hash. When the user authenticates with the server, the server compares the received password hash with its stored hash. However, if the user switches devices, the stored password hash on the server will not match the hash generated by the new device.

   To address this issue, the server would need to store multiple password hashes corresponding to each device used by the user, which increases complexity and potential security risks. Therefore, to maintain simplicity and security, it is preferable for users to consistently use the same device for authentication with servers that employ Lamport hashing.

2. When it comes to password-based authentication, a protocol that can address replay and reflection attacks is the use of a challenge-response mechanism with

the inclusion of a timestamp.

Here's a high-level description of the protocol:

- The client sends a request to the server to initiate the authentication process.
- The server generates a random challenge and sends it to the client.
- The client combines the challenge with the password, computes a hash of the concatenation (e.g., using a cryptographic hash function), and sends the hash value back to the server.
- The server verifies the received hash by performing the same hash computation using the stored password and the challenge.
- To prevent replay attacks, the server includes a timestamp in the challenge. The client includes this timestamp in the response, and the server verifies that it matches the expected value within an acceptable time window.

This protocol prevents replay attacks because the client cannot simply replay a previously captured response. Additionally, it mitigates reflection attacks because the server includes a challenge that is unique for each authentication attempt.

3. Key Transport using Symmetric Encryption: In this method, a trusted third party, called a Key Distribution Center (KDC), is involved. Alice and Bob both share a secret key with the KDC. When they want to exchange a secret key, they request the KDC to securely send the key to each other. The KDC uses symmetric encryption to protect the key during transmission.

4. Pre-shared Key (PSK) Exchange: In scenarios where Alice and Bob already share a secret key, such as a previously established secure channel or a shared secret obtained through an out-of-band method, they can directly use this shared key as the secret key for their communication.

5. Secure Sockets Layer/Transport Layer Security (SSL/TLS) Handshake: SSL/TLS protocols use asymmetric encryption and digital certificates to establish a secure channel between two parties. During the handshake process, the client and

server exchange public keys and establish a shared secret key for subsequent symmetric encryption.

### 15.1.4 Ex 4. Shamir secret sharing

1. In a Shamir scheme (2, 5) users got the points A=(1, 5), B=(2, 0), C=(3, 2), D=(4, 4) and E=(5, 6). Assume to work on GF(7). Reconstruct and return the secret starting from the points A and B.
2. Reconstruct again the secret starting from D and E, and verify that the secret is the same.

**Solution:** **Reconstruction starting from points A and B**

Given points:

$$A = (1, 5)$$

$$B = (2, 0)$$

To reconstruct the secret, we need to interpolate a polynomial of degree 1 (since we have 2 points).
The polynomial can be represented as:

$$P(x) = a + bx$$

Using the given points, we can set up two equations:

$$5 = a + b \cdot 1$$

$$0 = a + b \cdot 2$$

Solving these equations, we find:

$$a = 2$$

$$b = 3$$

Thus, the polynomial is:

$$P(x) = 2 + 3x$$

To find the secret, we evaluate the polynomial at $x = 0$:

$$P(0) = 2 + 3 \cdot 0 = 2$$

Therefore, the secret is 2.

**Reconstruction starting from points D and E**

Given points:

$$D = (4, 4)$$

$$E = (5, 6)$$

Using the same process, we set up the equations:

$$4 = a + b \cdot 4$$

$$6 = a + b \cdot 5$$

Solving these equations, we find:

$$a = 0$$

$$b = 6$$

Thus, the polynomial is:

$$P(x) = 0 + 6x$$

To find the secret, we evaluate the polynomial at $x = 0$:

$$P(0) = 0 + 6 \cdot 0 = 0$$

Therefore, the secret is 0.

By reconstructing the secret starting from points A and B and points D and E, we obtain the same secret value of 0.

### 15.1.5 Firewalls

1. When iptables is used as a personal firewall is there any case where the routing chain needs to be configured with rules?

2. Suppose iptables is running on host H which is protecting a LAN, from which it can be reached with the IP 192.168.1.254/24. Write appropriate rules so that the only network connection allowed to H is via ssh (two-way talk) 18 the LAN host 192.168.1.2/24 to allow firewall configuration. All other network connections to and from H are prohibited.

**Solution:** Firewall

## 15.2 3 February 2023

### 15.2.1 Ex 1. Authenticity versus authentication

Does one contain the other? Or does the other contain one? Any differences? Carefully explain.

**Solution:**

- Authenticity: Authenticity refers to the quality or state of being genuine, original, or trustworthy. It relates to the assurance that something is what it claims to be and has not been tampered with or altered. Authenticity focuses on the integrity and trustworthiness of an entity, such as a message, data, or object. It ensures that the entity has not been modified, forged, or manipulated in any unauthorized way. Establishing authenticity involves mechanisms that provide evidence of the origin and integrity of the entity, such as digital signatures, checksums, or hashing.
- Authentication: Authentication, on the other hand, is the process of verifying the identity or legitimacy of an entity, such as a user, system, or device. It involves confirming that the claimed identity of the entity is valid and reliable. Authentication is about establishing trust in the identity of the entity and ensuring that it is authorized to access certain resources or perform specific actions. Common authentication methods include passwords, biometrics, digital certificates, or multi-factor authentication.

Authenticity can contribute to the overall trustworthiness of an authentication process by verifying the integrity of the data or message being authenticated. However, authentication itself does not guarantee authenticity. Authenticity is a broader concept that encompasses the trustworthiness of the entire entity, while authentication is a subset that specifically verifies identity.

### 15.2.2 Ex 2. Symmetric encryption/decryption

1. Describe the architecture of OFB, both for encryption and for decryption.

2. Does OFB need to use a randomly generated IV at any encryption? Why?

3. What is a KDF and why is it useful?

4. What is the effect of the following command line string?
   ```
   echo "SilverSurfer" -n | openssl enc -aes-128-cbc -p -out
   out.enc
   ```

5. If in CBC the attacker flips the third bit of the first ciphertext block, and no integrity mechanism is in place, what is the total effect of that in Bob's reconstructed plaintext message? Discuss.

6. If in the CBC it is Alice who inverts the third bit of the first plaintext block, and no integrity mechanism is provided, what is the total effect of this intervention in Bob's reconstructed plaintext message? Discuss.

### 15.2.3 Ex 3. Digital certificates

1. Compare CRLs to OCSP

2. What is OCSP stapling and why is it done?

3. Why don't we need the https protocol but can use plain http in retrieving a certificate?

**Solution:**

### 15.2.4 Ex 4. Digital signatures

1. Alice and Bob digitally sign the same document M, which is a contract. If Alice is the first signer is there a difference whether Bob signs M or M with Alice's signature? Discuss.

2. f the verification of a digital signature fails (without diagnostics, so we don't know the reason for the failure), what is it correct to infer? Discuss.

3. The following sentence contains a poorly posed (and therefore incorrect) question, written solely to confuse ideas.
   If Alice wants to digitally sign a document, but she doesn't know the public key,

how should she do it?

Discuss why such a question should not be asked.

**Solution:**

### 15.2.5 Ex 5. Firewalls

1. Compare blacklisting with whitelisting and discuss the impact of those policies on iptables.
2. Suppose iptables is running on host H, which is protecting a LAN. Write appropriate rules so that
   (a) H does not allow any incoming or outgoing network connections (can only be configured using the local console)
   (b) H allows any connection from the LAN to the Internet, but does not allow opening connections from the Internet to the LAN.

**Solution:**

## 15.3 9 June 2023

### 15.3.1 Ex 1. Symmetric ciphers

1. Describe the scenario of symmetric ciphers and define the concepts of: synchronous/asynchronous stream ciphers, block ciphers and modes of operations.
   *If definition is wrong subsequent questions cannot be correctly answered*
2. Describe the RC4 cipher (both the key generation and the encryption process). What type of cipher is it?
3. Describe and compare OFB and CTR. Can you suggest possible design criteria for their adoption?

**Solution:**

### 15.3.2 Ex 2. Man in the middle

1. Describe the attack Man-In-The-Middle, as well as a possible scenario where it could be run.
2. Alice suspects she is currently being the target of a Man-In-The-Middle attack, and she decides to hire you as a personal adviser. Is it still possible to carry out safe actions? Discuss.

**Solution:**

### 15.3.3 Ex 3. Hashing

1. Define the properties that qualify a hashing function as cryptographic (the more formal, the better).
2. Describe the Merkle-Damgård construction. If the underlying hash function maps 256b blocks into 128b blocks, how many rounds are required for hashing a 140KB file?
3. Discuss and compare possible schemes for keying a hash function.

**Solution:**

### 15.3.4 Ex 4. RSA verification

1. Describe how to verify an RSA digital signature.
2. Explicitly describe the possible causes of a verification failure. For having non-repudiation is the verification needed?

**Solution:**

### 15.3.5 Ex 5. Authentication

1. Discuss the security of the following challenge-based scheme for mutual authentication, where Alice (A) and Bob (B) share a secret key K: (information below is transmitted as clear text)
   A $\rightarrow$ B: (A, NA, B)  NA is a nonce chosen by A
   B $\rightarrow$ A: (B, NB, K(NA), A)  NB is a nonce chosen by B
   A $\rightarrow$ B: (A, K(NB), B)
2. How would you improve the above schema?

**Solution:**

### 15.3.6 Ex 6. Miscellaneous

Provide short answers to the following questions.

1. Compute $5^{12241}$ mod 13
2. Describe as best as you can the meaning of the following command
   `iptables -A INPUT -p tcp -s 0/0-d 195.55.55.78 --sport 513:65535 --dport 22 -m state --state NEW, ESTABLISHED -j ACCEPT`
3. Describe as best as you can the meaning of the following command
   `ssh -L 44044:192.168.1.221:22 user@host.example.com`