

Advanced Encryption Standard (AES)

Prepared by Dr. Sumaya Alkubaisi

FNT625
Cybersecurity Concepts

Lecture 7



Lecture

7

Learning Goals

By the end of this lecture, you will be able to:

- Explain what AES is and why it became the global standard for symmetric encryption.
- Understand the core structure of AES, including its main transformation steps used during encryption rounds.
- Differentiate between AES-128, AES-192, and AES-256, and identify when each is used in practice.
- Describe why DES became obsolete and how AES overcomes DES limitations.
- Recognize where AES is applied in banking, FinTech, and digital payment systems to secure sensitive data.

What is AES?

AES is a symmetric block cipher that:

- Encrypts data in 128-bit fixed-size blocks.
- Supports 128, 192, or 256-bit keys.
- Performs multiple rounds of transformations to convert plaintext into ciphertext.
- AES is based on strong mathematical structures (finite fields) rather than Feistel design, making it more secure and efficient.
- AES was adopted in 2001 to replace DES and provide long-term security.

Features of AES

AES provides:

- **High security:** Resistant to known cryptographic attacks.
- **Flexibility:** Multiple key lengths for different security needs.
- **Efficiency:** Fast in both hardware and software.
- **Global standard:** Used in banking, Wi-Fi security, VPNs, SSL/TLS, and cloud services.

AES is suitable for modern devices, from servers to smartphones and IoT.

Lecture

7

AES vs DES (Quick Comparison)

Feature

Block Size

Key Size

Structure

Security

Speed

DES

64-bit

56-bit

Feistel Network

Weak today

Slow on software

AES

128-bit

128/192/256-bit

Substitution-Permutation
Network (SPN)

Very strong

Fast on both

AES Basics & Parameters

AES Block and Key Sizes

- AES processes data in 128-bit blocks.
- It supports three key sizes:
 - AES-128 → 128-bit key
 - AES-192 → 192-bit key
 - AES-256 → 256-bit key
- Larger keys provide stronger security but with slightly more computation.

AES Basics & Parameters

Number of Rounds

- AES performs a series of repeated transformation rounds on the data:
 - 10 rounds for AES-128
 - 12 rounds for AES-192
 - 14 rounds for AES-256
- Each round increases confusion and diffusion, making it harder for attackers to reverse the encryption without the key.

AES performs the following steps during encryption:

1.Initial Round: AddRoundKey

2.Repeated Rounds:

- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey

3.Final Round: (no MixColumns)

SubBytes → ShiftRows → AddRoundKey

This structured process provides strong security in each transformation layer.

The State Array (Core Concept)

- AES does not operate directly on a long number.
- Plaintext is arranged into a 4×4 byte matrix called **the State** (blocks of 128 bits = 16 bytes)
- All AES transformations work on this State matrix throughout the encryption process.

B0	B1	B2	B3
B4	B5	B6	B7
B8	B9	B10	B11
B12	B13	B14	B15

Lecture

7

AES Data Flow Summary

AES encryption flows as:

- Input Plaintext → State Matrix
- Apply Initial Round
- Apply Multiple Rounds (SubBytes, ShiftRows, MixColumns, AddRoundKey)
- Apply Final Round
- Output = Ciphertext (128 bits)

This gives a structured, repeatable, and reversible process for converting data securely.

AES Round Transformations (Overview)

Each AES round applies four main transformations to the State:

- SubBytes – Byte substitution for non-linearity (confusion)
- ShiftRows – Row shifting to spread bytes (diffusion)
- MixColumns – Mixes data within each column (diffusion)
- AddRoundKey – XOR with key material (security binding to the key)

These four steps work together to make plaintext unrecognizable without the key.

SubBytes (Concept)

- A non-linear substitution step where each byte in the State is replaced using an S-Box (Substitution Box).
- The S-Box is designed to resist cryptanalysis by introducing non-linearity.
- Each input byte is replaced by a new byte according to a lookup table.

Purpose: Add confusion, making the relationship between plaintext and ciphertext complex.

Lecture

7

SubBytes (Example)

Using a portion of the AES S-Box:

If a byte = 0x3A: (the input determines the lookup position)

Row = 3, Column = A (the value found there is 0xF1)

S-Box output = 0xF1

Example transformation:

Before SubBytes: 3A → After SubBytes: F1

		Columns							
		0	1	2	3	...	A	...	F
R	0	xx	xx	xx	xx		xx		xx
o	1	xx	xx	xx	xx		xx		xx
w	2	xx	xx	xx	xx		xx		xx
s	3	xx	xx	xx	xx	→ (F1)			xx
		...							
F									

- This simple replacement is what makes AES resistant to linear attacks.

ShiftRows (Concept)

- A row-wise shifting of the State matrix to spread bytes across columns.
- Rows are shifted left by different offsets:
 - Row 0: No shift
 - Row 1: Shift left by 1
 - Row 2: Shift left by 2
 - Row 3: Shift left by 3

Purpose: Increase diffusion and prevent alignment of bytes across rounds.

ShiftRows (Example)

Before ShiftRows:

| Row 0 | a | b | c | d |

| Row 1 | e | f | g | h |

| Row 2 | i | j | k | l |

| Row 3 | m | n | o | p |

After row-wise shifting:

- Row 0: no shift → a b c d
- Row 1: shift left by 1 → f g h e
- Row 2: shift left by 2 → k l i j
- Row 3: shift left by 3 → p m n o

After ShiftRows:

| A | B | C | D |

| F | G | H | E |

| K | L | I | J |

| P | M | N | O |

Now bytes originally in the same column are spread across different columns.

MixColumns (Concept)

- A column transformation where each column of the State is transformed using a fixed matrix.
- Achieved using math over a finite field ($GF(2^8)$).
- This mixes each byte with others in the same column.

Purpose: Add strong diffusion across 4 bytes in each column.

Effect: Changing one byte in the input column affects all four bytes in the output column, giving AES strong diffusion.

MixColumns (Simplified Example)

- For one column bytes (a, b, c, d):
- Column before: [D4 BF 5D 30] → Column after: [04 66 81 E5]

How it works:

1. AES treats the column as a vector and multiplies it by a fixed 4×4 matrix in GF(2⁸):

02	03	01	01
01	02	03	01
01	01	02	03
3	01	01	02

2. Each new byte is computed as a combination of the original bytes using GF(2⁸) multiplication (multiplies the corresponding byte **modulo the irreducible polynomial**) and XOR:

$$\text{new_a} = (02 \cdot a) \oplus (03 \cdot b) \oplus (01 \cdot c) \oplus (01 \cdot d)$$
$$\text{new_b} = (01 \cdot a) \oplus (02 \cdot b) \oplus (03 \cdot c) \oplus (01 \cdot d)$$

...and similarly for new_c and new_d.

AddRoundKey (Concept)

- The only step that directly uses the encryption key.
- The State is XORed with a round key derived from the main key.
- XOR ensures reversibility — applying the same key again reverses the effect.

Purpose: Bind encryption directly to the secret key.

AddRoundKey (Example)

State byte: 0x3C

Round key byte: 0xA1

XOR result:

$$3C \oplus A1 = 9D$$

- This is done for all 16 bytes of the State matrix.
- Without the correct round key, decryption is impossible.

AES Initial Round (Round 0)

Before the first round begins, AES performs only:

- AddRoundKey
- The plaintext block is XORed with the initial round key.
- This ensures that no plaintext bits enter the rounds unprotected.
- This “pre-whitening” step immediately binds data to the key.

Lecture

7

AES Standard Round (Rounds 1 to N_r-1)

- Each standard AES round (except the last) includes all four transformations: N_r = Number of rounds:

Step

SubBytes

ShiftRows

MixColumns

AddRoundKey

Role

Adds non-linearity (**confusion**)

Spreads bytes across rows (**diffusion**)

Mixes bytes within columns (**diffusion**)

XOR with the subkey (**security**)

- 10 (AES-128)
- 12 (AES-192)
- 14 (AES-256)

AES Final Round (Last Round)

The last round removes MixColumns:

- SubBytes
- ShiftRows
- AddRoundKey

Why skip MixColumns?

- Prevents unnecessary mixing that would complicate correct decryption alignment.
- Maintains a clean invertible structure.

AES Encryption Process

AES-128 example (10 rounds):

Step

Round 0

Rounds 1–9

Round 10

Applied

AddRoundKey

SubBytes → ShiftRows → MixColumns →
AddRoundKey

SubBytes → ShiftRows → AddRoundKey

- Every round increases confusion & diffusion until ciphertext appears statistically random.

AES Decryption (Overview)

- AES decryption reverses each step using inverse transformations:

Encryption Step

SubBytes

ShiftRows

MixColumns

AddRoundKey

Decryption Step

InvSubBytes

InvShiftRows

InvMixColumns

AddRoundKey (same XOR)

- Note: AddRoundKey is the same for encryption & decryption because XOR is its own inverse.

Lecture

7

AES Decryption Reverse Rounds

Decryption applies the reversed order:

1. Final Round:

- AddRoundKey
- InvShiftRows
- InvSubBytes

2. Rounds 1–9:

- AddRoundKey
- InvMixColumns
- InvShiftRows
- InvSubBytes

3. Initial: AddRoundKey (same logic)

Lecture

7

Key Expansion (Why Needed?)

- AES does not use the original key directly in all rounds.
To strengthen security, AES derives a new key for each round — called Round Keys — using a process called Key Expansion.

Why is this important?

Prevents attackers from reversing the cipher with only 1 key.

Adds complexity so patterns do not repeat across rounds.

Ensures each encryption round behaves differently → stronger security.

In Simple Words: Key Expansion stretches the original key into many sub-keys to make AES harder to break.

Lecture

7

What is Key Expansion? (Simple Overview)

- Key Expansion = a function that generates multiple Round Keys from the original key.
- For AES-128:
- Original Key Size: 128 bits
- Round Keys Needed: 11 keys (1 for initial + 10 rounds)
- Output: 44 “words” (each word = 4 bytes)

Concept Summary:

Original Key → processed through small transformations → produces new keys for every round.

Key Expansion (Concept)

Key Expansion uses 3 core operations:

Operation

RotWord

SubWord

Rcon

Description

Cyclic shift of 4-byte word

Applies S-Box to 4-byte word

Adds round constants to introduce non-linearity

Process:

- Original key = first round key
- Next keys generated word-by-word using previous words

Lecture

7

Key Expansion (Concept)

Why these 3?

RotWord → mixes order (diffusion)

SubWord → adds nonlinearity (confusion)

Rcon → ensures each round key is unique

These small steps dramatically increase security.

Lecture

7

Example: Key Expansion Basics

- AES-128 key = 16 bytes = 4 words (W_0, W_1, W_2, W_3)
- AES-128 requires 44 words total (4 words $\times$ 11 round keys: 1 initial + 10 rounds)
- So we need to generate 40 more words ($W_4 \dots W_{43}$) from the original 4 words.

Step1: Take the last word:

“Last word” means the previously generated word, initially this is the last word of the original key, i.e., **W_3** .

➤ **result = W_3**

Lecture

7

Example: Key Expansion Basics

- AES-128 key = 16 bytes = 4 words (W_0, W_1, W_2, W_3)

Step2: Apply RotWord:

Rotate bytes of the word left by 1 position: $[a_0 \ a_1 \ a_2 \ a_3] \rightarrow [a_1 \ a_2 \ a_3 \ a_0]$

➤ **result: RotWord(W_3)**

Step3: Apply SubWord:

Substitute each byte using the AES S-box. This adds non-linearity.

➤ **result = SubWord(RotWord(W_3))**

Lecture

7

Example: Key Expansion Basics

- AES-128 key = 16 bytes = 4 words (W0, W1, W2, W3)

Step4: XOR with Rcon:

- Rcon is a round constant that changes each round.

result = temp $\oplus$ Rcon1

Step5: XOR with previous word to form next word:

- Here, “previous word” means the word **4 positions back** in the key schedule (the word from the previous column of the key, not the last word you just rotated).

To generate W4: $W4 = W0 \oplus \text{result}$

- So the formula is: $\text{result} = \text{SubWord}(\text{RotWord}(W3)) \oplus \text{Rcon1}$
 $W4 = W0 \oplus \text{result}$

Why Did AES Replace DES?

- DES (Data Encryption Standard) was used for many years, but as computers became faster, its 56-bit key became too small to resist brute-force attacks. Attackers could try all possible keys and break DES in a short time. AES was introduced to solve these weaknesses.
- AES provides stronger security because it uses longer keys (128, 192, or 256 bits) and a more advanced encryption structure.
- It is faster, more secure, and suitable for modern systems such as mobile banking, cloud computing, and internet payments.
- In summary: DES became easy to break and outdated, while AES offers long-term, reliable, and high-performance protection for today's data.

AES vs DES (Conceptual Differences)

- DES and AES are both symmetric encryption algorithms, but AES was designed as a modern replacement.
- DES uses a 56-bit key and works on 64-bit data blocks, which limits its security and speed. While AES uses larger keys and operates on 128-bit blocks, providing stronger protection and better performance.
- AES uses a more modern encryption design that is harder to attack with cryptographic techniques. It is secure against all known practical attacks, while DES can be broken using affordable hardware.
- Because of this, DES is no longer recommended, and AES is now the global standard for securing sensitive information.

AES-128 vs AES-192 vs AES-256 Security

- AES comes in three key sizes: 128-bit, 192-bit, and 256-bit. All three are secure, but the longer the key, the harder it is to break.
- **AES-128** already offers very strong protection and is widely used for fast operations such as online banking.
- **AES-192** offers a middle-level option, providing stronger security than AES-128 with slightly more computation. It is used when extra security is needed without the full cost of AES-256.
- **AES-256** provides an even higher security margin and is preferred for long-term protection, such as securing stored financial records.

How Secure Is AES Today?

- AES is currently viewed as highly secure, no one has been able to break AES when it is implemented correctly.
- It is trusted by governments, the military, banks, and global tech companies.
- Any security problems usually come from weak passwords, poor key management, or incorrect implementation—not from the AES algorithm itself.
- There is one future concern: quantum computing. A powerful quantum computer may reduce the time needed to try keys.
- For this reason, security experts suggest using AES-256 for long-term data protection, as it offers stronger resistance to possible future quantum attacks.

Where Banks Use AES

Banks use AES to protect financial information across many systems:

- When customers log in to online or mobile banking, AES encrypts their personal data and transactions so that no one can read them if intercepted.
- It also protects PIN codes and card information used at ATMs and payment terminals.
- AES is used inside core banking databases to protect stored financial records, customer details, and transaction history.
- International payment systems, including SWIFT messages, rely on AES to secure communication between banks.
- In short, AES protects data while it is stored, transferred, or processed inside the banking environment.

Lecture

7

AES in FinTech and Digital Payments

- FinTech services rely heavily on AES because it is fast, secure, and works across all platforms.
- Mobile payment apps such as Apple Pay and Google Pay use AES to encrypt card details and payment tokens.
- Digital-only banks and financial apps use AES to secure communication between users' devices and their servers.
- AES also protects private keys stored in crypto wallets, and cloud-based financial platforms use AES-256 to encrypt customer data stored on cloud servers.
- For FinTech companies, AES is a perfect fit because it provides top-level security without slowing down digital services.

Lecture

7

AES Application for Protecting Financial Data or Messages

Objective : Understand how AES works to encrypt and decrypt data securely, such as bank card information or personal messages, in a practical and straightforward way, with a note that this can be implemented using Python.

Simplified Steps

1. Choose the data to protect

Example: Virtual card number: 1234-5678-9012-3456

Example: Short message: "Hello, how are you?"

2. Choose a secret key

- The key is a word or combination of letters and numbers that only you know.
- The key should be strong (e.g., 16 or 32 characters).
- Python: The key can be created using encryption library functions such as:

Crypto.Random.get_random_bytes(n) in **PyCryptodome**

os.urandom(16) or **os.urandom(32)** in **cryptography**

AES in FinTech and Digital Payments

3. Encrypt the data (Encryption)

The original data is converted into encrypted text that no one else can understand.

Example:

- Original data: 1234-5678-9012-3456
- After encryption → Encrypted text: F1A9D3B4C2E7...
- Python: Encryption can be done using:

AES.new() in **PyCryptodome**

encryptor.update(plaintext) + encryptor.finalize() in **cryptography**

4. Transfer or store the encrypted data

- The encrypted text can be safely sent over the internet or stored in a database.
- Anyone intercepting it will not be able to understand it.

AES in FinTech and Digital Payments

5. Decrypt the data (Decryption)

- Use the same secret key to convert the encrypted text back into the original data.

Example:

- Encrypted text: F1A9D3B4C2E7...
- After decryption with the key → Original data: 1234-5678-9012-3456
- Python: Decryption can be done using:

`cipher.decrypt(ciphertext)` in **PyCryptodome**

`decryptor.update(ciphertext) + decryptor.finalize()` in **cryptography**

Lecture

7

Recap

- AES replaced DES because DES's 56-bit key became too weak against modern brute-force attacks.
- AES is a symmetric block cipher that encrypts data in 128-bit blocks and supports 128, 192, and 256-bit keys for different security levels.
- The encryption process uses multiple rounds of transformation to ensure strong diffusion and confusion of data.
- AES-128 is commonly used for fast data protection, while AES-256 is preferred for long-term and high-security needs, especially in financial systems.
- AES is widely used in online banking, ATMs, mobile payments, FinTech apps, cloud banking, and digital wallets to ensure confidentiality and data protection.

References

- Stallings, W. Cryptography and Network Security, 8th Ed., 2023
- NIST FIPS-197: Advanced Encryption Standard (AES), 2001
- Khan, M. (2022). Cybersecurity Tools
- Daemen & Rijmen. The Design of Rijndael, Springer, 2002
- Recent Study: S. Chen (2023) – AES performance & security in modern systems