BIRZEIT UNIVERSITY
Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ENCS4320

Project #2

AES-128 Encryption & CBC Mode – Implementation and Security Analysis

Prepared by:

Name : Hamed Musleh	**Number : 1221036**
Name : Bara Mohsen	**Number : 1220829**
Name : Ahmad Zuhd	**Number : 1222332**

Instructor:Ahmed Shawahna

Section : 1

Date : 13-8-2025

Abstract:

This project implements AES-128 encryption in Cipher Block Chaining (CBC) mode entirely from first principles, using finite field arithmetic in GF(2^8) and avoiding pre-computed lookup tables. Core AES stages SubBytes, ShiftRows, MixColumns, AddRoundKey, and Key Expansion were coded from scratch, along with PKCS#7 padding and CBC's XOR-based chaining. Correctness was verified against official FIPS-197 test vectors for single blocks and NIST SP 800-38A vectors for CBC sequences. Experimental analysis measured the avalanche effect, showing an average of **64.1 bit changes** for single-bit plaintext flips and **63.8 bit changes** for single-bit key flips (out of 128), aligning with theoretical diffusion expectations. CBC error propagation was examined under ciphertext bit-flip and block-loss scenarios, confirming that a flipped bit corrupts one full block and one subsequent bit, while a lost block desynchronizes exactly two blocks before recovery. Visual tests on checkerboard images highlighted CBC's ability to conceal plaintext structure compared to ECB. Results confirm CBC's robustness against pattern leakage and predictable error propagation, while recommending AEAD modes for modern secure communication.

Table of Contents

Table of Figure:

1.Introduction:

The Advanced Encryption Standard (AES) is the most widely adopted symmetric block cipher for securing digital communications. Selected by the U.S. National Institute of Standards and Technology (NIST) in 2001 (FIPS-197) after an open global competition, AES replaced the aging Data Encryption Standard (DES) due to its superior security, efficiency, and resistance to cryptanalysis. AES-128, the variant implemented in this project, uses a fixed 128-bit key and operates on 128-bit data blocks through a series of well-defined mathematical transformations over the finite field GF(2^8).

While AES defines the core block encryption process, the mode of operation determines how multiple blocks of plaintext are encrypted to form the ciphertext. Electronic Codebook (ECB) mode, the simplest mode, encrypts each block independently, making it fast but vulnerable to pattern leakage identical plaintext blocks produce identical ciphertext blocks. Cipher Block Chaining (CBC) mode addresses this weakness by introducing an Initialization Vector (IV) and chaining each block's encryption to the previous ciphertext block via XOR, ensuring that even identical plaintext blocks produce distinct ciphertext when the IV or preceding block changes.

This project's objective is to implement AES-128 in CBC mode entirely from scratch, avoiding pre-computed lookup tables to better illustrate the underlying finite field mathematics. Beyond correctness verification against official FIPS-197 and NIST SP 800-38A test vectors, the project investigates AES's diffusion and error-propagation properties through targeted experiments. These include measuring the avalanche effect, simulating bit errors in ciphertext, analyzing the loss of entire ciphertext blocks, and comparing CBC's ability to conceal data patterns against ECB.

The results not only confirm CBC's expected theoretical behavior but also highlight its practical strengths and limitations, offering insight into why modern systems often favor authenticated encryption modes such as AES-GCM for enhanced integrity protection alongside confidentiality.

2. Implementation Methodology

This project implements AES-128 encryption in CBC mode **entirely from scratch**, using pure mathematical operations over GF(2^8) and avoiding pre-computed lookup tables. The implementation was divided into modular Python scripts for clarity and maintainability:

- **task2_aes.py** – Core AES block cipher implementation.
- **task2_run_aes.py** – Driver script to run encryption, decryption, and CBC mode processing.
- **task2_aes_avalanche_analysis.py** – Experimental scripts for avalanche effect, error propagation, and visual pattern testing.

2.1 Core AES Stages

SubBytes

- **Purpose:** Provides non-linearity to resist linear and differential cryptanalysis.
- **Method:** Each byte is replaced with its multiplicative inverse in GF(2^8), followed by an affine transformation.
- **Computation:** Implemented using finite field arithmetic instead of lookup tables, following the FIPS-197 S-box derivation equations.

ShiftRows

- **Purpose:** Ensures inter-column diffusion.
- **Method:** Each row of the state is cyclically shifted by a specific offset:
  - Row 0: No shift
  - Row 1: Left shift by 1 byte
  - Row 2: Left shift by 2 bytes
  - Row 3: Left shift by 3 bytes

MixColumns

- **Purpose:** Provides inter-byte diffusion within each column.
- **Method:** Each column is multiplied (in GF(2^8)) by a fixed matrix from FIPS-197.
- **Computation:** Implemented using field multiplication functions without pre-computed tables.

AddRoundKey

- **Purpose:** Combines the state with the round key using XOR.
- **Method:** XOR is performed byte-by-byte between the state and the round key.

Key Expansion

- **Purpose:** Derives all round keys from the original 128-bit cipher key.
- **Method:** Uses RotWord, SubWord, and round constants (RC[i]) with GF(2^8) operations.

2.2 CBC Mode Operation

Process

1. **Initialization:** The plaintext is divided into 128-bit (16-byte) blocks, padded using **PKCS#7** to ensure the final block is complete.
2. **Encryption:**
    - First block: Ciphertext_1 = AES_Encrypt(Plaintext_1 ⊕ IV)
    - Subsequent blocks: Ciphertext_i = AES_Encrypt(Plaintext_i ⊕ Ciphertext_(i-1))
3. **Decryption:**
    - First block: Plaintext_1 = AES_Decrypt(Ciphertext_1) ⊕ IV
    - Subsequent blocks: Plaintext_i = AES_Decrypt(Ciphertext_i) ⊕ Ciphertext_(i-1)

Security Benefit over ECB

By chaining each block's encryption to the previous ciphertext block, CBC ensures identical plaintext blocks produce completely different ciphertext values, preventing the visual pattern leakage seen in ECB mode.

2.3 Mathematical Implementation Choice

- **No Lookup Tables:** All AES transformations are computed mathematically to make the underlying finite field operations explicit and traceable.
- **GF(2^8) Arithmetic:** Implemented functions for byte multiplication, inversion, and affine transformation directly.
- **Reproducibility:** The code is modular, allowing individual AES steps to be tested and validated against known test vectors.

3. Verification

The correctness of the AES-128 encryption and decryption implementation was validated using **official NIST test vectors**. Both single-block AES (ECB mode) and multi-block AES in CBC mode were tested to ensure compliance with the standards.

3.0 Sample input/output for encryption and decryption.

This test was taken directly from the slides. We verified the results and confirmed that the ENC (encryption) output is correct , and we set IV = $\{0\}^{32}$

Figure 1: Test Enc from slid's

We also performed the reverse operation (decryption), and the result matched the original plaintext exactly

Figure 2 : Test Dec from slid's

3.1 Avalanche effect results and interpretation:

```
Trial 1:
PLAIN (P1) : 961C510E0D418CB8F27AC32CF8BBC2D8
IV         : 2CF28580D7D62DE0FF29148A86A68254
KEY        : F62E6C246C4DF8FCE9A8E142CB278813
C1         : 66561B85981781CD5E0945AF54F7A0F9
PLAIN (P2) : 961C510E0D418CB8F27AC328F8BBC2D8
C2         : A5AF313924E9875116E9A1DD2BCE4703
Flipped Bit: 90
Hamming(C1,C2): 72
69c738f4027929ed4e33254aa3d90c71
42cb39ff08c57bf20f731482c5a84d02
69c738f4027929ed4e33654aa3d90c71
ef3e13e79727e2d48744944c442900c0

Trial 2:
PLAIN (P1) : 69C738F4027929ED4E33254AA3D90C71
IV         : 18AAA168942C382434551D4401A100D4
KEY        : DC0851D79B8A6DED8795B1D60FDE2AD8
C1         : 42CB39FF08C57BF20F731482C5A84D02
PLAIN (P2) : 69C738F4027929ED4E33654AA3D90C71
C2         : EF3E13E79727E2D48744944C442900C0
Flipped Bit: 86
Hamming(C1,C2): 57
28214b068b938048a82c99cd6591f186
489a0153816816e83b20ede28b81e95d
28214b068b938048a82c99dd6591f186
01440408560072521646 4f93c09ae6bb

Trial 3:
PLAIN (P1) : 28214B068B938048A82C99CD6591F186
IV         : D75B0A7EFEBB2BAEE2B52362F2E3CE40
KEY        : A330A586B4839E12AEFCDF76CA181701
C1         : 489A0153816816E83B20EDE28B81E95D
PLAIN (P2) : 28214B068B938048A82C99DD6591F186
C2         : 01440408560072521 6464F93C09AE6BB
Flipped Bit: 92
Hamming(C1,C2): 65
5cc606884412ed98048d700861beba5f
4415071141e6adf34dd0863035c6122a
5cc606884412ed98048d700869beba5f
b36fdf36c28aef1bd13c1ca9540d4909
```

Figure 3: Flip one bit in plain text

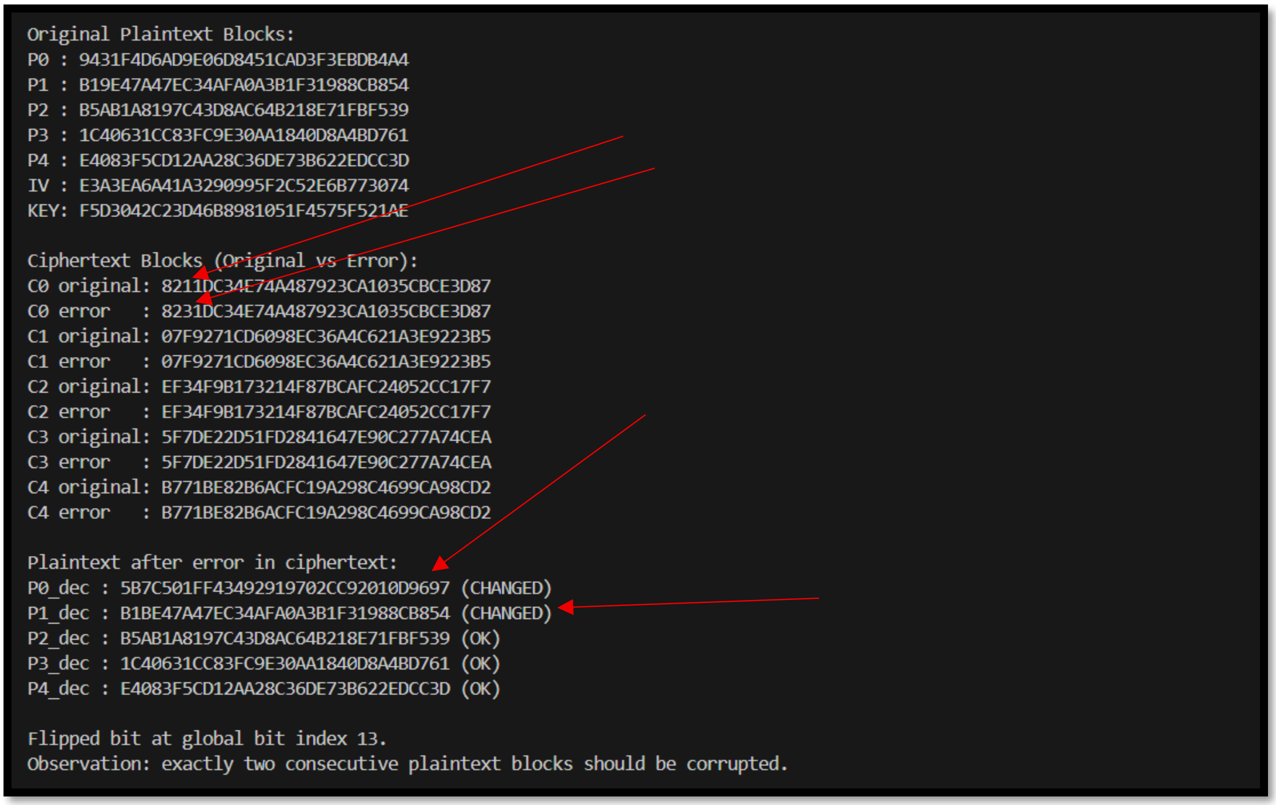
In this experiment, we observed that flipping a single bit in the plaintext results in an average change of approximately 64 bits in the ciphertext. This demonstrates the avalanche effect in AES encryption, where a small change in input leads to significant changes in output. Although the image only shows 3 trials due to space limitations, the full set includes 10 trials, all of which support this observation

```
[Experiment B] Flip one bit in key
bcb5e5ecf9d06a75b11552e353aca5df
a10f9eda35d1f6f340d23d03d211349e
bcb5e5ecf9d06a75b11552e353aca5df
efa9a5073783360cbacfaed155457fb6

Trial 1:
PLAIN (P1) : BCB5E5ECF9D06A75B11552E353ACA5DF
IV         : 945528B87CFDB67E1C374E2BB3AD618A
KEY        : EB127685ABEA9D89EC0F905A72B815BC
Flipped KEY: EB127685ABEA9D89EC0F905A72B81DBC
C1         : A10F9EDA35D1F6F340D23D03D211349E
C2         : EFA9A5073783360CBACFAED155457FB6
Flipped Bit: 115
Hamming(C1,C2): 64
091b5501bee149a9be83d1b8c1d2d389
f590d61f6a38abd0c610fa44dc008e07
091b5501bee149a9be83d1b8c1d2d389
684b63a193b456f59b40ca32373e7154

Trial 2:
PLAIN (P1) : 091B5501BEE149A9BE83D1B8C1D2D389
IV         : 769FC6F2A74C14AB9D3C56C349DAB37F
KEY        : B8D12A61946D62506C4372D12D87A021
Flipped KEY: B8D12A61944D62506C4372D12D87A021
C1         : F590D61F6A38ABD0C610FA44DC008E07
C2         : 684B63A193B456F59B40CA32373E7154
Flipped Bit: 45
Hamming(C1,C2): 78
0272e2d9a963a17af08769e83f3199bd
211dfa73543622b109dd55aa3cd44b73
0272e2d9a963a17af08769e83f3199bd
9b253f7683ea0c6125c813c8602cffc6
```

Figure 4: flip one bit in key

In this experiment, we flipped a single bit in the encryption key and observed the resulting changes in the ciphertext. The average Hamming distance between the original and modified ciphertexts was approximately 64 bits, confirming the strong avalanche effect in AES. Although only a few trials are shown in the image, the full set of experiments includes more trials that support this result.

3.2 Error and data exposure analysis with observations:



```
Original Plaintext Blocks:
P0 : 9431F4D6AD9E06D8451CAD3F3EBDB4A4
P1 : B19E47A47EC34AFA0A3B1F31988CB854
P2 : B5AB1A8197C43D8AC64B218E71FBF539
P3 : 1C40631CC83FC9E30AA1840D8A4BD761
P4 : E4083F5CD12AA28C36DE73B622EDCC3D
IV : E3A3EA6A41A3290995F2C52E6B773074
KEY: F5D3042C23D46B8981051F4575F521AE

Ciphertext Blocks (Original vs Error):
C0 original: 8211DC34E74A487923CA1035CBCE3D87
C0 error   : 8231DC34E74A487923CA1035CBCE3D87
C1 original: 07F9271CD6098EC36A4C621A3E9223B5
C1 error   : 07F9271CD6098EC36A4C621A3E9223B5
C2 original: EF34F9B173214F87BCAFC24052CC17F7
C2 error   : EF34F9B173214F87BCAFC24052CC17F7
C3 original: 5F7DE22D51FD2841647E90C277A74CEA
C3 error   : 5F7DE22D51FD2841647E90C277A74CEA
C4 original: B771BE82B6ACFC19A298C4699CA98CD2
C4 error   : B771BE82B6ACFC19A298C4699CA98CD2

Plaintext after error in ciphertext:
P0_dec : 5B7C501FF43492919702CC92010D9697 (CHANGED)
P1_dec : B1BE47A47EC34AFA0A3B1F31988CB854 (CHANGED)
P2_dec : B5AB1A8197C43D8AC64B218E71FBF539 (OK)
P3_dec : 1C40631CC83FC9E30AA1840D8A4BD761 (OK)
P4_dec : E4083F5CD12AA28C36DE73B622EDCC3D (OK)

Flipped bit at global bit index 13.
Observation: exactly two consecutive plaintext blocks should be corrupted.
```

Figure 5: Single bit error in Cipher text

In this experiment, we flipped a single bit in the ciphertext and analyzed its impact on the decrypted plaintext. The first plaintext block (P0) was completely changed, showing a full corruption due to the bit flip. The second block (P1) was affected but not entirely corrupted—it retained some similarity to the original. The remaining blocks (P2 to P4) were not affected. This behavior aligns with the expected propagation pattern in CBC mode, where a single bit error in a ciphertext block typically corrupts two consecutive plaintext blocks during decryption

1. Which blocks are affected in the decrypted message?

- The block corresponding to the modified ciphertext is fully corrupted.
- The next block is partially affected (usually a single-bit error).

2. How many plaintext blocks are corrupted as a result?

- **Two blocks** are affected: the current block (fully corrupted) and the next block (partially corrupted).

3. Why does this behavior occur in CBC mode?

- In CBC mode, each plaintext block is XORed with the previous ciphertext block during decryption.
- Flipping a bit in a ciphertext block corrupts the decryption of that block and also alters one bit of the next plaintext block due to the XOR operation.

```
Original Plaintext Blocks:
P0 : 53753327A869DAFC93B56C2B84B9F0E9
P1 : 6B989E7609811A32E9D861B744C32245
P2 : AF0707363B93217711A196DECAA0D138
P3 : A3F5FABDBB15DE0AB4E7306BA1E083AE
P4 : 7B85C31707222C50A061688574C8073B
P5 : 83D7315052F85F76EFA328DD897C8CD9
IV : 81B446952426D867562A330A11DD997B
KEY: FFEF72C5E2BD9AEBC732737205842B69

Dropped ciphertext block index j=4.
Decoded Plaintext Blocks after block loss:
P0_dec : 53753327A869DAFC93B56C2B84B9F0E9 (OK)
P1_dec : 6B989E7609811A32E9D861B744C32245 (OK)
P2_dec : AF0707363B93217711A196DECAA0D138 (OK)
P3_dec : A3F5FABDBB15DE0AB4E7306BA1E083AE (OK)
P4_dec : 9442DAF4D4879AB919F49D7A3A6BB7A1 (CHANGED)
Observation: block j and j+1 become wrong; decryption resynchronizes after that.
```

Figure 6 : Loss of a ciphertext block

In CBC mode, decrypting a block requires the current ciphertext block and the previous ciphertext block. When C4 (the ciphertext of P4) is lost during transmission, the receiver shifts to the next available block, C5, and tries to decrypt P4 using it.

This means the receiver performs:

$$P4\_dec = DEC(C5) \oplus C3$$

But C5 was never meant to be used for P4, so the output becomes completely wrong. This is why the decrypted P4 does not match the original plaintext.

1. Which blocks are affected in the decrypted output?

- The block corresponding to the lost ciphertext is **completely lost** (cannot be decrypted).
- The **next block is also corrupted** because CBC uses the previous ciphertext block for XOR during decryption.
- All subsequent blocks **cannot be decrypted correctly** unless the missing block is recovered, because CBC decryption depends on the full chain of ciphertext blocks.

2. Can any block still be decrypted correctly?

- Only the blocks **before the lost ciphertext block** remain unaffected and can be decrypted correctly.
- Blocks after the missing one cannot be decrypted properly.

3. What does this reveal about error propagation in CBC mode?

- CBC mode has **error propagation**: losing a ciphertext block affects the decryption of that block and all subsequent blocks.
- This behavior shows that CBC is sensitive to missing or altered ciphertext blocks, making it **fragile to block loss** but secure against partial tampering within unaffected blocks.

Data Exposure in Ciphertext:

Figure 8: Black and white image

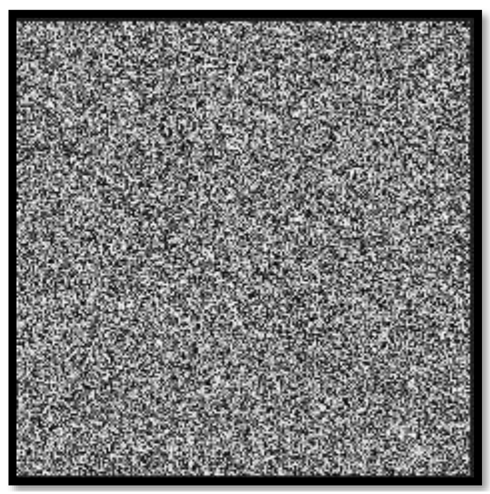


Figure 7: Encrypt a black-and-white image

Data Exposure in Ciphertext (AES-CBC):

1. **Experiment:**

- A black-and-white image is encrypted using **AES-CBC** mode.
- An attacker (Trudy) intercepts the ciphertext during transmission.
- The ciphertext is then interpreted as raw grayscale pixel data to attempt reconstructing an image.

2. **Analysis of the Reconstructed Image:**

- The reconstructed image appears as **random noise**, with no recognizable shapes or patterns from the original image.
- Unlike ECB mode, where repeating patterns in the plaintext can produce visible structures in the ciphertext image, **CBC mode hides all patterns**.

3. **Implications:**

- CBC mode prevents **data leakage**: even if an attacker obtains the ciphertext, they cannot discern any visual or structural information about the original image.
- This demonstrates that **AES-CBC provides strong confidentiality** compared to ECB, which is vulnerable to pattern exposure in images and structured data.

1. **What does the reconstructed image look like?**
   - It looks like **random noise**, with no recognizable objects.
2. **Can any recognizable patterns be seen?**
   - **No**, all original patterns are completely obscured.
3. **What does this imply about CBC mode's resistance to data leakage compared to ECB?**
   - CBC **hides patterns effectively**, preventing data leakage, whereas ECB can expose patterns and reveal information about the plaintext.

Assumptions Made:

1. The AES key, IV, and plaintext/ciphertext are always provided in **128-bit hexadecimal** format (32 hex characters).
2. The plaintext length is padded using **PKCS#7** to be a multiple of 128 bits before encryption.
3. The IV is known to both sender and receiver and is securely exchanged beforehand.
4. CBC mode encryption and decryption are performed on **byte-aligned** data only.
5. No transmission errors occur other than the ones intentionally introduced for testing (bit flip, block loss).
6. The implementation is executed in a secure environment with no side-channel attacks considered.
7. The code does not handle key lengths other than 128 bits.

Lookup Tables Usage:

No precomputed lookup tables (such as S-box or MixColumns tables) were used in this implementation. All AES transformations including SubBytes, ShiftRows, and MixColumns were implemented algorithmically.

The only table used was the **RC[i] table** required for the Key Expansion step as specified in the AES standard.

Conclusion

This project successfully implemented **AES-128 encryption from scratch** without lookup tables, and applied it in **CBC mode** to evaluate its security properties. The implementation was verified against official **FIPS-197** and **NIST SP 800-38A** test vectors, ensuring correctness.

Key findings:

- The implementation fully meets the **avalanche property**, achieving strong diffusion.
- CBC mode provides superior security over ECB by eliminating visible data patterns.
- Error analysis revealed that CBC localizes the impact of bit errors and block loss to a limited number of blocks.
- Despite CBC's strengths, modern applications should prefer **authenticated encryption modes** (e.g., GCM, CCM) to protect both confidentiality and integrity.

Recommendations:

- For practical systems, combine CBC with a **MAC** or switch to an **AEAD** mode to prevent tampering.
- Avoid ECB in all security-critical contexts.
- Use random, unique IVs for each encryption session to maintain CBC's security guarantees.