Assignment on AES-128

Aniket Basak (CrS2401)
Cryptographic and Security Implementation

September 2025

1 Why ECB Mode is Insecure for General Use of AES-128

The Electronic Codebook (ECB) mode of operation with AES-128 is insecure for general use because of several structural weaknesses in how it processes plaintext blocks:

1. Pattern Leakage

AES-128 in ECB mode encrypts each 128-bit block independently with the same key. As a result, identical plaintext blocks produce identical ciphertext blocks. This leads to leakage of patterns in the plaintext structure.

Example: If an image is encrypted in ECB mode, the output ciphertext image still reveals visible outlines of the original image, since repeating pixel blocks remain identical after encryption.

2. Lack of Randomness (No IV)

ECB mode does not use an Initialization Vector (IV) or nonce. Thus, encrypting the same message with the same key always produces the same ciphertext. This makes ECB deterministic and vulnerable to the following attacks:

- Replay attacks: an attacker can resend an old encrypted message.
- Frequency analysis: attackers can analyze repetitions in ciphertext to infer patterns in the plaintext.

3. No Chaining Between Blocks

In ECB, blocks are processed completely independently of one another. Changing a single plaintext block affects only one ciphertext block. This property allows attackers to cut, paste, or rearrange ciphertext blocks without breaking the rest of the encrypted message.

4. Chosen-Plaintext Attack Weakness

Since ECB produces deterministic, block-by-block results, an attacker with the ability to feed chosen plaintexts into the system can construct a "codebook" mapping plaintext blocks to ciphertext blocks. Such a codebook can later be used to recognize or partially decrypt messages without knowledge of the secret key.

Example: Repeated Text

Consider the plaintext message:

`HELLOHELLOHELLOHELLO`

This message is 20 bytes long and will be divided into 16-byte blocks for AES-128 encryption:

```
Block 1: HELLOHELLOHELLO
Block 2: HELLO + padding
```

Notice that the sequence "HELLO" appears multiple times. In ECB mode, every occurrence of the block "HELLO" will always encrypt to the same ciphertext. An attacker who observes the ciphertext can therefore detect these repetitions.

Example: Image Encryption

The weakness of ECB mode is even clearer when encrypting images. If a bitmap image (for example, the classic penguin test image) is encrypted using AES-128 in ECB mode, the resulting ciphertext image still visibly resembles the original. This happens because large areas of the same color encrypt into identical ciphertext blocks, preserving the overall structure of the image even though the pixel values are encrypted.

2 Recommended Alternatives to ECB Mode

For real-world applications of AES-128, several alternative modes of operation are recommended because they eliminate the weaknesses of Electronic Codebook (ECB) mode.

1. Cipher Block Chaining (CBC)

How it works: Each plaintext block is XORed with the previous ciphertext block before encryption. The first block uses a random Initialization Vector (IV).

Why it is better than ECB:

- Identical plaintext blocks produce different ciphertexts if IVs differ.
- Provides diffusion: changing one plaintext block changes all following ciphertext blocks.

Weakness: Still deterministic if the IV is reused, and vulnerable to certain padding oracle attacks if not implemented securely.

2. Counter Mode (CTR)

How it works: Instead of chaining blocks, CTR turns AES into a stream cipher. A counter value (combined with a nonce) is encrypted for each block, and the result is XORed with the plaintext.

Why it is better than ECB:

- Identical plaintext blocks encrypt into different ciphertexts due to the changing counter.
- Fully parallelizable (encryption and decryption can be done in parallel, unlike CBC).
- No padding is required since CTR works like a stream cipher.

Weakness: Nonce misuse is catastrophic: reusing the same nonce and key leaks relationships between plaintexts.

3. Galois/Counter Mode (GCM)

How it works: GCM builds on CTR mode for encryption but also provides authentication using Galois field multiplication. This makes it an Authenticated Encryption with Associated Data (AEAD) scheme.

Why it is better than ECB:

- Provides both confidentiality and authenticity: any tampering with ciphertext can be detected.
- Efficient and parallelizable.
- Widely used in secure protocols such as TLS and IPsec.

Weakness: As with CTR mode, nonce reuse is dangerous and must be avoided.

4. Cipher Feedback Mode (CFB)

How it works: CFB treats the block cipher like a stream cipher by feeding the previous ciphertext block into the encryption function, then XORing the result with the plaintext.

Why it is better than ECB:

- Repeated plaintext blocks do not result in identical ciphertexts.
- Can operate on segments smaller than the block size.

Weakness: Slower compared to CTR and GCM, and less commonly used in modern systems.

3 Explanation of my code

- **State Representation:** The AES state is stored as a 4×4 byte matrix $state$[row][col], and bytes are loaded column-major from the input. That is, for input bytes in[16]:

$$state[r][c] = in[c \cdot 4 + r], \quad 0 \leq r, c < 4$$

  This mapping follows the AES/FIPS standard.

- **Key Expansion:** The `KeyExpansion` function generates all 11 round keys for AES-128 (10 rounds + initial round key), stored in a 176-byte array (11×16 bytes). Each 4-byte word is derived from the previous word(s) using `RotWord`, `SubWord`, and the round constant `Rcon`.

- **AES Operations:**
  - `SubBytes` / `InvSubBytes`: Apply S-box / inverse S-box substitution to each byte.
  - `ShiftRows` / `InvShiftRows`: Rotate rows of the state matrix to the left/right by 1,2,3 bytes respectively.
  - `MixColumns` / `InvMixColumns`: Perform Galois field multiplication to mix each column.
  - `AddRoundKey`: XOR each byte of the state with the corresponding round key byte.

- **Single-block Encryption/Decryption:** Each 128-bit block is encrypted using the standard AES-128 sequence:
  1. Initial `AddRoundKey` with round 0 key.
  2. 9 rounds of `SubBytes`, `ShiftRows`, `MixColumns`, `AddRoundKey`.
  3. Final round: `SubBytes`, `ShiftRows`, `AddRoundKey` (no `MixColumns`).

  Decryption performs the inverse operations in reverse order, including `InvMixColumns` in the first 9 rounds.

- **Galois Field Multiplication:** `gmul(a,b)` multiplies two bytes in $GF(2^8)$ using the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. `xtime(x)` multiplies by 0x02, used in `MixColumns`.

- **ECB Mode and PKCS#7 Padding:**
  - ECB encryption splits plaintext into 16-byte blocks. If the last block is incomplete, PKCS#7 padding is applied: each padding byte has value N, the number of padding bytes ($1 \leq N \leq 16$).
  - Decryption removes padding after verifying correctness; invalid padding results in an error.

- **Test Vector:** The implementation correctly encrypts the provided single-block test vector:

Plaintext: 3243f6a8885a308d313198a2e0370734

Key: 2b7e151628$ae$$d2a6ab$$f7158809c$$f4f3c$

Ciphertext: 3925841d02dc09$fbdc$118597196a0b32

```
Computed ciphertext: 3925841d02dc09fbdc118597196a0b32
Expected ciphertext: 3925841d02dc09fbdc118597196a0b32
Single-block AES-128 encryption test: OK Γ£à
Decrypted back: 3243f6a8885a308d313198a2e0370734

ECB encrypted (hex, 64 bytes):
b5a1996306ed3cef8387c448a0508d218e0fe94016e01d3b878a2c540ae214e553cb10f28dc3c27eb657477ed96cc3a0789ef93a1af2f665dc922ab90f4387c4
ECB decrypted (63 bytes):
This is a test of AES-128 ECB mode. It will use PKCS#7 padding!
```

Figure 1: AES output

The compile command used is:

```
gcc -O2 -std=c11 aes_ecb.c -o aes_ecb
```

Here is a detailed explanation of each part:

- **gcc:** This is the GNU Compiler Collection's C compiler, commonly used on Linux, Windows (via MinGW), and macOS.
- **-O2:**This tells the compiler to optimize the code at level 2:
  - `-O0` = no optimization (default, useful for debugging)
  - `-O1`, `-O2`, `-O3` = increasingly aggressive optimization

  Using `-O2` improves performance (especially for AES loops) without making debugging too difficult.
- **-std=c11:**This enforces that the C11 standard is used, which ensures:
  - Compatibility with modern C features (`stdint.h`, `inline`, etc.)
  - Avoids warnings about certain standard functions or constructs
  - AES code uses `uint8_t` types, which are part of standard C99 and later.
- **aes_ecb.c**
  This is the source code file containing the AES implementation.
- **-o aes_ecb**
  This specifies the output executable name. Without it, GCC defaults to `a.out` on Unix-like systems. Naming it `aes_ecb` makes it clear and convenient.