# Applied Cryptography
# Week 6/ Padding Oracle Attack Lab

Student Name: **Mikheil Davidovi**
Class: **Applied Cryptography**

Submission Date: **November , 2025**

**GITHUB:** https://github.com/mikheildavidovi-pixel/Sangu2025

**Task 1: Understand the Components**

***Analyze the padding_oracle function. How does it determine if padding is valid?***
It first checks that the ciphertext length is a multiple of the block size. It splits the input into iv (first 16 bytes) and ct (remaining bytes), decrypts ct with AES-CBC using iv and KEY, then attempts PKCS#7 unpadding. If unpadding completes without raising ValueError/TypeError, the function returns True (valid padding); if unpadding raises an exception, it returns False. This boolean feedback is the padding oracle.

***What is the purpose of the IV in CBC mode?***
The IV is XORed with the first plaintext block before encryption to introduce randomness. It ensures identical plaintexts encrypted with the same key produce different ciphertexts and prevents pattern leakage across messages.

***Why does the ciphertext need to be a multiple of the block size?***
AES is a block cipher operating on fixed-size blocks (16 bytes). CBC processes data block-by-block, so the ciphertext must be composed of whole blocks; PKCS#7 padding is used to make the plaintext length a multiple of the block size before encryption and is removed after decryption. A ciphertext whose length isn't a multiple of the block size cannot be correctly decrypted or padded.

**Task 2: Implement Block Splitting**

```
def split_blocks(data: bytes, block_size: int = BLOCK_SIZE) -> list[bytes]:
    return [data[i:i + block_size] for i in range(0, len(data), block_size)]
```

**Task 3: Implement Single Block Decryption**

```
def decrypt_block(prev_block: bytes, target_block: bytes) -> bytes:
    block_len = BLOCK_SIZE
    intermediate = [0] * block_len
```

```
    plaintext = [0] * block_len
    for pad_len in range(1, block_len + 1):
        i = block_len - pad_len
        found = False
        for guess in range(256):
            forged = bytearray(block_len)
            for j in range(i + 1, block_len):
                forged[j] = intermediate[j] ^ pad_len
            forged[i] = guess
            test_ct = bytes(forged) + target_block
            if padding_oracle(test_ct):
                intermediate[i] = guess ^ pad_len
                plaintext[i] = intermediate[i] ^ prev_block[i]
                found = True
                break
        if not found:
               raise RuntimeError(f"Failed to find a valid byte at position
{i}")
    return bytes(plaintext)
```

**Task 4: Implement Full Attack**

```
def padding_oracle_attack(ciphertext: bytes) -> bytes:
    blocks = split_blocks(ciphertext, BLOCK_SIZE)
    if len(blocks) < 2:
        raise ValueError("Ciphertext must contain at least IV + 1 block")
    plaintext = bytearray()
    for i in range(1, len(blocks)):
        prev_block = blocks[i - 1]
        target_block = blocks[i]
        pt_block = decrypt_block(prev_block, target_block)
        plaintext.extend(pt_block)
    return bytes(plaintext)
```

**Task 5: Implement Plaintext Decoding**

```
def unpad_and_decode(plaintext: bytes) -> str:
    try:
```

```
        unpadder = padding.PKCS7(BLOCK_SIZE * 8).unpadder()
        unpadded = unpadder.update(plaintext) + unpadder.finalize()
    except Exception as e:
        return f"[!] Error during unpad/decode: {e}"
    try:
        return unpadded.decode("utf-8", errors="replace")
    except Exception as e:
        return f"[!] Decode error: {e}"
```

**Final plaintext:**

**This is a top secret message. Decrypt me if you can!**