CSE350: Programming Assignment 2

Ananya Lohani (2019018), Mihir Chaturvedi (2019061)

Advanced Encryption Standard

AES, or Advanced Encryption Standard, is a symmetric encryption algorithm that uses a block cipher to encrypt data in fixed-size blocks of 128 bits. The 128-bit key size is one of the three possible key sizes in AES, and it determines the number of rounds used in the encryption process. For 128-bit AES, the encryption process uses 10 rounds of operations, each consisting of four steps - SubBytes, ShiftRows, MixColumns, and AddRoundKey.

In the SubBytes step, each byte in the block is replaced with a corresponding byte from a fixed substitution table called S-box. In the ShiftRows step, the second row of the block is shifted one byte to the left, the third row is shifted two bytes to the left, and the fourth row is shifted three bytes to the left. In the MixColumns step, each column of the block is multiplied with a fixed matrix to ensure that each byte in the block is affected by the diffusion process. Finally, in the AddRoundKey step, the block is XORed with a portion of the key that is derived from the main key using a key schedule. This process is repeated for 9 rounds, and in the final round, the MixColumns step is skipped. The resulting block is the ciphertext. To decrypt the ciphertext, the inverse operations of each round are applied in reverse order, using the inverse S-box, inverse matrix, and inverse key schedule to generate the round keys.

AES Specifications

- **Number of rounds:** 10
- **Key size:** 128 bits or 16 bytes
- **Plaintext size:** 128 bits or 16 bytes
- **Mode:** CBC (Cipher Block Chaining)

Documentation (Docstrings)

```
|-----------------------------------------------------------------------------------------------------
| AES(key)
| A class that implements the AES encryption algorithm with 10 rounds in CBC mode.
|
| Attributes
| ----------
| KEY_SIZE : int
| the size of the key in bytes (16)
| N_ROUNDS : int
| the number of rounds (10)
| SBOX : list
| the S-box used for SubBytes in encryption
| INV_SBOX : list
| the inverse S-box used for SubBytes in decryption
| RC : list
| the round constants used for round key generation
|
| master_key : bytes
| the master key used for encryption and decryption
| round_keys : list
| the round keys generated from the master key
| gf : galois.Galois
| the Galois field (2**8) used for multiplication in MixColumns
```

```
|
| Methods
| -------
| left_rotate(word) -> bytes :
|   Rotates a word (a 4-byte sequence) to the left by one byte.
| bytes_to_blocks(data) -> list :
|   Converts a byte string into a list of n-byte blocks.
| blocks_to_bytes(blocks) -> bytes :
|   Converts a list of blocks into a byte string.
| xor(a, b) -> bytes :
|   Performs the XOR operation between two byte strings of the same length.
| pad_msg(msg) -> bytes :
|   Pads a message to a length that is a multiple of 128 bits.
| unpad_msg(msg) -> bytes :
|   Removes the padding from a message.
|
| get_round_keys(key) -> list :
|   Generates a list of 11 round keys from the master key.
|
| add_round_key(state, round_key) -> list :
|   Performs the AddRoundKey step of the encryption & decryption process.
| sub_bytes(state, inv=False) -> list :
|   Performs the SubBytes step of the encryption & decryption process.
| shift_rows(state, inv=False) -> list :
|   Performs the ShiftRows step of the encryption & decryption process.
| mix_columns(state, inv=False) -> list :
|   Performs the MixColumns step of the encryption & decryption process.
| encrypt(state, iv) -> bytes :
|   Encrypts a message using the AES algorithm in CBC mode.
| decrypt(state, iv) -> bytes :
|   Decrypts an encrypted message using the AES algorithm in CBC mode.
|
| Methods defined here:
|
| __init__(self, key)
|    Initializes a new instance of the AES class with the given key.
|    Parameters:
|       key (bytes): A byte string representing the encryption key. Must have a length of 16 bytes.
|
| add_round_key(self, state, round_key)
|    Performs the AddRoundKey step of the encryption & decryption process.
|    Parameters:
|       state (list): A matrix representing the current state of the encryption process.
|       round_key (bytes): A list representing the round key to be XORed with the state.
|    Returns:
|       list: The resulting state matrix after the AddRoundKey step.
|
| blocks_to_bytes(self, blocks)
|    Converts a list of blocks into a byte string.
|    Parameters:
|       blocks (list): A list of blocks, where each block is a list of n-byte sequences.
|    Returns:
|       bytes: The byte string obtained by concatenating the blocks.
|
| bytes_to_blocks(self, data, n=4)
|    Converts a byte string into a list of n-byte blocks.
|    Parameters:
|       data (bytes): The byte string to be converted into blocks.
|       n (int): The number of bytes in each block. Default is 4.
|    Returns:
|       list: A list of n-byte blocks.
|
| decrypt(self, ciphertext, iv)
|    Decrypts an encrypted message using the AES algorithm in CBC mode.
|    Parameters:
|       ciphertext (bytes): The byte string representing the message to be decrypted.
|       iv (bytes): The byte string representing the initialization vector. Must have a length of 16 bytes.
|    Returns:
|       bytes: The byte string representing the decrypted message.
|
| encrypt(self, plaintext, iv)
|    Encrypts a message using the AES algorithm in CBC mode.
|    Parameters:
|       plaintext (bytes): The byte string representing the message to be encrypted.
|       iv (bytes): The byte string representing the initialization vector. Must have a length of 16 bytes.
|    Returns:
|       bytes: The byte string representing the encrypted message.
|
| get_round_keys(self, key)
|    Generates a list of 11 round keys from the master key.
|    Parameters:
|       key (bytes): The byte string representing the master key.
|    Returns:
|       list: A list of round keys, where each key is a list of 4 32-bit words.
|
| left_rotate(self, word)
|    Rotates a word (a 4-byte sequence) to the left by one byte.
|    Parameters:
|       word (bytes): A 4-byte sequence to be rotated.
|    Returns:
|       bytes: The rotated 4-byte sequence.
|
| mix_columns(self, state, inv=False)
|    Performs the MixColumns step of the encryption & decryption process.
|    Parameters:
|       state (list): A matrix representing the current state of the encryption process.
|       inv (bool): If True, the inverse MixColumns operation is performed for decryption.
|       Default is False.
|    Returns:
|     list: The resulting state matrix after the MixColumns step.
|
| pad_msg(self, msg)
|    Pads a message to a length that is a multiple of 128 bits.
|    Parameters:
|       msg (bytes): The byte string representing the message.
|    Returns:
|       bytes: The padded message.
|
| shift_rows(self, state, inv=False)
|    Performs the ShiftRows step of the encryption & decryption process.
|    Parameters:
|       state (list): A matrix representing the current state of the encryption process.
|       inv (bool): If True, the inverse ShiftRows operation is performed for decryption.
|       Default is False.
|    Returns:
|       list: The resulting state matrix after the ShiftRows step.
```

```
|
| sub_bytes(self, state, inv=False)
|     Performs the SubBytes step of the encryption & decryption process.
|     Parameters:
|         state (list): A matrix representing the current state of the encryption process.
|         inv (bool): If True, the inverse SubBytes operation is performed for decryption.
|         Default is False.
|     Returns:
|         list: The resulting state matrix after the SubBytes step.
|
| unpad_msg(self, msg)
|     Removes the padding from a message.
|     Parameters:
|         msg (bytes): The byte string representing the padded message.
|     Returns:
|         bytes: The unpadded message.
|
| xor(self, a, b)
|     Performs the XOR operation between two byte strings of the same length.
|     Parameters:
|         a (bytes): The first byte string.
|         b (bytes): The second byte string.
|     Returns:
|         bytes: The byte string resulting from the XOR operation.
|
| ----------------------------------------------------------------------------------------------------------
```

Sample Inputs & Outputs

Plaintext: **`Hello, world!`**

I/O Pair #1

Inputs:

Key:

0xd3	0xcc	0x34	0x60	0xae	0xec	0xe7	0x78	0x33	0xd7	0x16	0x5d	0x2e	0x9a	0x7b	0xba

IV:

0xa9	0xaa	0xf9	0xa3	0x7f	0x57	0x7d	0x93	0x3	0x26	0x9	0x52	0xf6	0x8b	0xf4	0xf

Outputs:

```
Plaintext: Hello, world!
Ciphertext: b'\x03\xec\xeb.\x8c\x95\x99\x12\xf8\xd2\x14\xf9\xf1\xbb\x86J'
Decrypted Ciphertext: Hello, world!
```

Block 1, Encryption Round 1

a0	a1	a2	a3
0x68	0xb1	0xde	0xf0
0x1	0xbf	0x1e	0x5d
0xba	0x26	0x2c	0xdc
0xaf	0xbc	0x62	0x9

Block 1, Encryption Round 9

a0	a1	a2	a3
0x9	0x13	0x27	0xd5
0xde	0x60	0xe0	0xa9
0x7e	0x2d	0x55	0x36
0xf1	0x52	0x5d	0xf1

Block 1, Decryption Round 1

a0	a1	a2	a3
0x9	0x13	0x27	0xd5
0xde	0x60	0xe0	0xa9
0x7e	0x2d	0x55	0x36
0xf1	0x52	0x5d	0xf1

Block 1, Decryption Round 9

a0	a1	a2	a3
0x68	0xb1	0xde	0xf0
0x1	0xbf	0x1e	0x5d
0xba	0x26	0x2c	0xdc
0xaf	0xbc	0x62	0x9

I/O Pair #2

Inputs:

Key:

0x7e	0xca	0x58	0x88	0x41	0xe2	0xbf	0x67	0x44	0xbd	0xec	0xb3	0x83	0x17	0xe5	0xe3

IV:

0x4e	0x68	0x83	0x64	0xdf	0x24	0xa7	0x97	0x2e	0xcd	0x11	0xc0	0x5c	0x4c	0x92	0xb

Outputs:

```
Plaintext: Hello, world!
Ciphertext: b'Z\xf1S\xf3T\xb0 \x8f\x16\x87\x10%B!\x938'
Decrypted Ciphertext: Hello, world!
```

Block 1, Encryption Round 1

a0	a1	a2	a3
0x16	0x57	0x5c	0xc2
0xcb	0xd8	0x2	0x88
0x4b	0x53	0x6d	0x81
0x74	0x9f	0xe0	0xbf

Block 1, Encryption Round 9

a0	a1	a2	a3
0x7b	0xb1	0xa0	0xcf
0x76	0xc0	0x99	0xfa
0x37	0xaa	0x61	0x8a
0x5d	0xbd	0xe2	0x7a

Block 1, Decryption Round 1

a0	a1	a2	a3
0x7b	0xb1	0xa0	0xcf
0x76	0xc0	0x99	0xfa
0x37	0xaa	0x61	0x8a
0x5d	0xbd	0xe2	0x7a

Block 1, Decryption Round 9

a0	a1	a2	a3
0x16	0x57	0x5c	0xc2
0xcb	0xd8	0x2	0x88
0x4b	0x53	0x6d	0x81
0x74	0x9f	0xe0	0xbf