

Федеральное государственное автономное образовательное учреждение высшего
образования
«Национальный исследовательский университет ИТМО»
Факультет программной инженерии и компьютерной техники

Лабораторная работа 2
«Блочное симметричное шифрование»

Вариант № 2в

Группа: Р34102

Выполнил: Лапин А.А.

Проверил:
Рыбаков С.Д.

Санкт-Петербург
2024г.

Оглавление

Введение	2
1 Текст задания	3
2 Структура проекта	4
2.1 Описание основных компонентов	4
2.1.1 idea.py	4
2.1.2 pcbc.py	4
2.1.3 main.py	4
2.1.4 test_idea_pcbc_cipher.py	4
3 Листинги разработанной программы с комментариями	5
3.1 idea.py	5
3.2 pcbc.py	8
3.3 main.py	10
4 Результаты работы программы	12
4.1 Пример 1: Шифрование	12
4.1.1 Исходный текст (input.txt)	12
4.1.2 Команда для шифрования	12
4.1.3 Результат	12
4.2 Пример 2: Дешифровка	12
4.2.1 Исходный текст ('encrypted.bin'):	12
4.2.2 Команда для дешифровки	12
4.2.3 Результат	12
4.2.4 Файл decrypted.txt	12
5 Заключение	14

Введение

Цель работы: изучение структуры и основных принципов работы современных алгоритмов блочно-го симметричного шифрования, приобретение навыков программной реализации блочных симметричных шифров.

Глава 1 Текст задания

Реализовать систему симметричного блочного шифрования, позволяющую шифровать и дешифровать файл на диске с использованием заданного блочного шифра в заданном режиме шифрования.

Глава 2 Структура проекта

Структура проекта представлена следующим образом:

```
Cryptography/
├─ lab1.2/
│   ├── src/
│   │   ├── idea.py
│   │   ├── pcbc.py
│   │   └─ main.py
│   ├── input.txt
│   ├── encrypted.bin
│   ├── decrypted.txt
│   └─ tests/
│       └─ test_idea_pcbc_cipher.py
```

2.1 Описание основных компонентов

2.1.1 idea.py

Реализация алгоритма IDEA (International Data Encryption Algorithm). Предоставляет методы для выполнения основных операций шифрования и дешифрования блоков данных, а также управления ключевыми раундами.

2.1.2 pcbc.py

Реализация режима шифрования PCBC (Propagating Cipher Block Chaining). Обеспечивает безопасность шифрования за счет использования цепочки блоков и добавления пропагирующего эффекта ошибок при дешифровке.

2.1.3 main.py

Основной файл для шифрования и дешифровки файлов. Предоставляет интерфейс командной строки для выполнения операций шифрования и дешифровки, а также управления ключами и инициализационным вектором.

2.1.4 test_idea_pcbc_cipher.py

Набор тестов для проверки корректности реализации алгоритма IDEA и режима шифрования PCBC. Использует библиотеку ‘pytest’ для организации и выполнения тестовых случаев.

Глава 3 Листинги разработанной программы с комментариями

3.1 idea.py

```
1 import struct
2 from typing import Union, Sequence, List
3 UINT16_MASK: int = (1 << 16) - 1 # 0xFFFF
4 def is_uint16(value: int) -> bool:
5     return 0 <= value < (1 << 16)
6
7 class IDEA:
8
9     NUM_ROUNDS = 8
10    KEYS_PER_ROUND = 6
11
12
13    def __init__(self, key):
14        if len(key) != 16:
15            raise ValueError("Key must be 16 bytes long")
16        self.key = key
17        self.round_keys = self.generate_round_keys(key)
18        self.decryption_keys = self.generate_decryption_keys()
19
20    def mul(self, x, y):
21        """
22        Performs multiplication modulo (2^16 + 1) for the IDEA algorithm.
23
24        In the context of IDEA:
25        - 0 is interpreted as 2^16 (65536 or 0x10000)
26        - A result of 0x10001 (65537) is considered equivalent to 0
27        """
28        assert is_uint16(x)
29        assert is_uint16(y)
30        # Константа для модульного умножения
31        MODULUS = 0x10000 # 2^16
32
33        # Обработка специального случая: 0 представляется как 2^16
34        if x == 0:
35            x = MODULUS
36        if y == 0:
37            y = MODULUS
38
39        # Выполнение модульного умножения
40        result = (x * y) % (MODULUS + 1)
41
42        # Если результат равен 2^16, возвращаем 0
43        return 0 if result == MODULUS else result
44
45    def mul_inv(self, x):
```

```

46     """
47     Calculate the multiplicative inverse of a number modulo 0x10001 (65537).
48
49     The multiplicative inverse of x is a number y such that (x * y) % 65537 ==
50     1.
51     If x is 0, the function returns 0 as defined by the IDEA algorithm.
52
53     Args:
54         x (int): The number to find the inverse of.
55
56     Returns:
57         int: The multiplicative inverse of x modulo 65537, or 0 if x is 0.
58
59     Raises:
60         ValueError: If no inverse exists for the given x.
61     """
62     assert is_uint16(x)
63     if x == 0:
64         return 0
65     else:
66         return pow(x, 0xFFFF, 0x10001) # By Fermat's little theorem
67         # 65537 (0x10001) is a Fermat prime number (2^16 + 1).
68         # By Fermat's Little Theorem: x^65536 ≡ 1 (mod 65537) for x ≠ 0.
69         # Therefore, x^65535 ≡ x^(-1) (mod 65537), which is the multiplicative
70         # inverse.
71
72 def add_inv(self, x):
73     """
74     Calculate the additive inverse of a number modulo 2^16 (65536).
75
76     The additive inverse of x is a number y such that (x + y) % 65536 == 0.
77     """
78     assert is_uint16(x)
79     return (-x) & UINT16_MASK
80
81 def add(self, x, y):
82     """
83     Perform addition modulo 2^16 (65536).
84     """
85     assert is_uint16(x)
86     assert is_uint16(y)
87     return (x + y) & UINT16_MASK
88
89
90 def generate_round_keys(self, key):
91     """
92     Generate 52 16-bit round keys from the 128-bit master key.
93     """
94     assert len(key) == 16
95
96     round_keys = []
97     key_bits = int.from_bytes(key, byteorder='big')
98     assert 0 <= key_bits < (1 << 128)
99
100     # Append the 16-bit prefix onto the suffix to yield a uint144

```

```

101         key_bits = (key_bits << 16) | (key_bits >> 112)
102
103         for i in range(self.NUM_ROUNDS * self.KEYS_PER_ROUND + 4):
104             offset = (i * 16 + i // 8 * 25) % 128
105             val = (key_bits >> (128 - offset)) & UINT16_MASK
106             assert is_uint16(val)
107             round_keys.append(val)
108         assert len(round_keys) == 52
109         return round_keys
110
111     def generate_decryption_keys(self):
112         """
113         Generate the decryption keys from the encryption keys.
114
115         This method creates a set of 52 16-bit decryption keys by inverting and
116         rearranging the encryption keys. The process ensures that decryption
117         with these keys will undo the encryption process.
118
119         Returns:
120             list: A list of 52 16-bit integers representing the decryption keys.
121
122         Raises:
123             ValueError: If invalid key components are encountered during the process
124
125         """
126         encrypt_keys = self.round_keys
127         assert len(encrypt_keys) % 6 == 4
128         decrypt_keys: List[int] = []
129         K1 = self.mul_inv(encrypt_keys[-4])
130         K2 = self.add_inv(encrypt_keys[-3])
131         K3 = self.add_inv(encrypt_keys[-2])
132         K4 = self.mul_inv(encrypt_keys[-1])
133         K5 = encrypt_keys[-6]
134         K6 = encrypt_keys[-5]
135         decrypt_keys.extend([K1, K2, K3, K4, K5, K6])
136
137         for i in range(1, self.NUM_ROUNDS):
138             j: int = i * self.KEYS_PER_ROUND
139             K1 = self.mul_inv(encrypt_keys[-j - 4])
140             K2 = self.add_inv(encrypt_keys[-j - 2])
141             K3 = self.add_inv(encrypt_keys[-j - 3])
142             K4 = self.mul_inv(encrypt_keys[-j - 1])
143             K5 = encrypt_keys[-j - 6]
144             K6 = encrypt_keys[-j - 5]
145             decrypt_keys.extend([K1, K2, K3, K4, K5, K6])
146
147         C1 = self.mul_inv(encrypt_keys[0])
148         C2 = self.add_inv(encrypt_keys[1])
149         C3 = self.add_inv(encrypt_keys[2])
150         C4 = self.mul_inv(encrypt_keys[3])
151         decrypt_keys.extend([C1, C2, C3, C4])
152         assert len(decrypt_keys) == len(encrypt_keys)
153         return decrypt_keys
154
155     def encrypt_block(self, block):
156         return self._crypt(block, "encrypt")

```



```

157 def decrypt_block(self, block):
158     return self._crypt(block, "decrypt")
159
160 def _crypt(self, block: bytes, direction: str) -> bytes:
161     assert len(self.key) == 16
162     assert direction in ("encrypt", "decrypt")
163
164     if len(block) != 8:
165         raise ValueError("Block size must be 8 bytes")
166
167     X1, X2, X3, X4 = struct.unpack('!4H', block)
168     round_keys = self.decryption_keys if direction == "decrypt" else self.
        round_keys
169     for round in range(self.NUM_ROUNDS):
170         k = round * self.KEYS_PER_ROUND
171         Y1 = self.mul(X1, round_keys[k])
172         Y2 = self.add(X2, round_keys[k + 1])
173         Y3 = self.add(X3, round_keys[k + 2])
174         Y4 = self.mul(X4, round_keys[k + 3])
175
176         T1 = Y1 ^ Y3
177         T2 = Y2 ^ Y4
178         T3 = self.mul(T1, round_keys[k + 4])
179         T4 = self.add(T2, T3)
180         T5 = self.mul(T4, round_keys[k + 5])
181         T6 = self.add(T3, T5)
182
183         X1 = self.add(Y1 ^ T5, 0)
184         X2 = self.add(Y3 ^ T5, 0)
185         X3 = self.add(Y2 ^ T6, 0)
186         X4 = self.add(Y4 ^ T6, 0)
187
188     # Final transformation
189     k = 48
190     C1 = self.mul(X1, round_keys[k])
191     C2 = self.add(X3, round_keys[k + 1])
192     C3 = self.add(X2, round_keys[k + 2])
193     C4 = self.mul(X4, round_keys[k + 3])
194
195     return struct.pack('!4H', C1, C2, C3, C4)

```

3.2 pcbc.py

```

1 from idea import IDEA
2
3 class PCBC:
4     """
5     Implements the Propagating Cipher Block Chaining (PCBC) mode of operation.
6     """
7     def __init__(self, cipher: IDEA):
8         """Initialize PCBC with a block cipher object."""
9         self.cipher = cipher
10
11     def pad(self, data: bytes) -> bytes:
12         """
13         Apply PKCS#7 padding to the input data.

```

```

14     Ensures the data length is a multiple of the block size (8 bytes).
15     """
16     padding_len = 8 - (len(data) % 8)
17     return data + bytes([padding_len] * padding_len)
18
19 def xor_blocks(self, block1: bytes, block2: bytes) -> bytes:
20     return bytes(b1 ^ b2 for b1, b2 in zip(block1, block2))
21
22 def unpad(self, data: bytes) -> bytes:
23     """
24     Remove PKCS#7 padding from the decrypted data.
25     Raises ValueError if padding is invalid.
26     """
27     padding_len = data[-1]
28     if padding_len < 1 or padding_len > 8:
29         raise ValueError("Invalid padding")
30     return data[:-padding_len]
31
32 def encrypt(self, plaintext: bytes, iv: bytes) -> bytes:
33     """
34     Encrypt the plaintext using PCBC mode.
35
36     Args:
37         plaintext: The data to encrypt.
38         iv: Initialization vector (8 bytes).
39
40     Returns:
41         Encrypted ciphertext.
42     """
43     plaintext = self.pad(plaintext)
44     ciphertext = b''
45     prev_ciphertext = iv
46     prev_plaintext = bytes(8)
47     for i in range(0, len(plaintext), 8):
48         block = plaintext[i:i + 8]
49
50         # PCBC mode encrypt: encrypt(plaintext_block XOR (prev_ciphertext XOR
51         #                       prev_plaintext))
52         ciphertext_block = self.xor_blocks(block, self.xor_blocks(
53             prev_ciphertext, prev_plaintext))
54
55         encrypted = self.cipher.encrypt_block(ciphertext_block)
56         ciphertext += encrypted
57
58         prev_ciphertext = encrypted
59         prev_plaintext = block
60     return ciphertext
61
62 def decrypt(self, ciphertext: bytes, iv: bytes) -> bytes:
63     """
64     Decrypt the ciphertext using PCBC mode.
65
66     Args:
67         ciphertext: The data to decrypt.
68         iv: Initialization vector (8 bytes).
69
70     Returns:

```

```

69         Decrypted plaintext with padding removed.
70
71     Raises:
72         ValueError if ciphertext length is not a multiple of 8 bytes.
73     """
74     if len(ciphertext) % 8 != 0:
75         raise ValueError("Ciphertext length must be multiple of 8 bytes")
76     plaintext = b''
77     prev_plaintext = bytes(8)
78     prev_ciphertext = iv
79     for i in range(0, len(ciphertext), 8):
80         block = ciphertext[i:i + 8]
81
82         decrypted = self.cipher.decrypt_block(block)
83
84         # PCBC mode decrypt: (prev_plaintext XOR prev_ciphertext) XOR decrypt(
85             ciphertext_block)
86         plaintext_block = self.xor_blocks(self.xor_blocks(prev_plaintext,
87             prev_ciphertext), decrypted)
88         plaintext += plaintext_block
89
90         prev_ciphertext = block
91         prev_plaintext = plaintext_block
92
93     return self.unpad(plaintext)

```

3.3 main.py

```

1  import sys
2  import argparse
3  from typing import Callable
4  from pcbc import PCBC
5  from idea import IDEA
6
7
8  def encrypt_file(pcbc: PCBC, input_file: str, output_file: str, iv: bytes) -> None:
9      """Encrypt the contents of input_file and write to output_file."""
10     try:
11         with open(input_file, 'rb') as f_in, open(output_file, 'wb') as f_out:
12             plaintext = f_in.read()
13             ciphertext = pcbc.encrypt(plaintext, iv)
14             f_out.write(ciphertext)
15             print(f"File encrypted and saved to {output_file}")
16     except Exception as e:
17         print(f"Encryption error: {e}")
18         sys.exit(1)
19
20
21  def decrypt_file(pcbc: PCBC, input_file: str, output_file: str, iv: bytes) -> None:
22      """Decrypt the contents of input_file and write to output_file."""
23     try:
24         with open(input_file, 'rb') as f_in, open(output_file, 'wb') as f_out:
25             ciphertext = f_in.read()
26             plaintext = pcbc.decrypt(ciphertext, iv)
27             f_out.write(plaintext)
28             print(f"File decrypted and saved to {output_file}")

```

```

29     except Exception as e:
30         print(f"Decryption error: {e}")
31         sys.exit(1)
32
33
34 def main() -> None:
35     parser = argparse.ArgumentParser(description="IDEA cipher with PCBC mode")
36     parser.add_argument('mode', choices=['encrypt', 'decrypt'], help="Operation mode")
37     parser.add_argument('input_file', help="Path to input file")
38     parser.add_argument('output_file', help="Path to output file")
39     parser.add_argument('key', help="Encryption/decryption key (16 bytes)")
40
41     args = parser.parse_args()
42
43     key = args.key.encode('utf-8')
44     if len(key) != 16:
45         print("Error: Key must be 16 bytes")
46         sys.exit(1)
47
48     try:
49         idea = IDEA(key)
50     except ValueError as e:
51         print(f"Key error: {e}")
52         sys.exit(1)
53
54     pcbc = PCBC(idea)
55     iv = b'\x00' * 8 # Initialization Vector
56
57     operation: Callable[[PCBC, str, str, bytes], None] = encrypt_file if args.mode
58         == 'encrypt' else decrypt_file
59     operation(pcbc, args.input_file, args.output_file, iv)
60
61 if __name__ == "__main__":
62     main()

```

Глава 4 Результаты работы программы

4.1 Пример 1: Шифрование

4.1.1 Исходный текст (input.txt)

```
1 Lorem Ipsum - это текст-"рыба", часто используемый в печати и вэб-дизайне. Lorem  
  Ipsum является стандартной "рыбой" для текстов на латинице с начала XVI века. В  
  то время некий безымянный печатник создал большую коллекцию размеров и форм  
  шрифтов, используя Lorem Ipsum для распечатки образцов. Lorem Ipsum не только  
  успешно пережил без заметных изменений пять веков, но и перешагнул в электронный  
  дизайн. Его популяризации в новое время послужили публикация листов Letraset с  
  образцами Lorem Ipsum в 60-х годах и, в более недавнее время, программы  
  электронной вёрстки типа Aldus PageMaker, в шаблонах которых используется Lorem  
  Ipsum.
```

4.1.2 Команда для шифрования

```
1 python src/main.py encrypt input.txt encrypted.bin 16bytekeyforIDEA
```

4.1.3 Результат

Вывод в консоль:

```
1 File encrypted and saved to encrypted.bin
```

4.2 Пример 2: Дешифровка

4.2.1 Исходный текст ('encrypted.bin'):

Двоичные данные, не представляемые в текстовом формате

4.2.2 Команда для дешифровки

```
1 python src/main.py decrypt encrypted.bin decrypted.txt 16bytekeyforIDEA
```

4.2.3 Результат

Вывод в консоль:

```
1 File decrypted and saved to decrypted.txt
```

4.2.4 Файл decrypted.txt

1 Lorem Ipsum – это текст-"рыба", часто используемый в печати и вэб-дизайне. Lorem Ipsum является стандартной "рыбой" для текстов на латинице с начала XVI века. В то время некий безымянный печатник создал большую коллекцию размеров и форм шрифтов, используя Lorem Ipsum для распечатки образцов. Lorem Ipsum не только успешно пережил без заметных изменений пять веков, но и перешагнул в электронный дизайн. Его популяризации в новое время послужили публикация листов Letraset с образцами Lorem Ipsum в 60-х годах и, в более недавнее время, программы электронной вёрстки типа Aldus PageMaker, в шаблонах которых используется Lorem Ipsum.

Глава 5 Заключение

В ходе выполнения лабораторной работы была реализована система симметричного блочного шифрования на основе алгоритма IDEA в режиме PCBC. Программа успешно шифрует и дешифрует файлы, обеспечивая надежную защиту данных. Полученные результаты подтверждают корректность реализации алгоритма и режима шифрования.