Laboratorium I - szyfrowanie symetryczne

Zadanie 1: Wizualna analiza wpływu różnych trybów szyfrowania blokowego na obrazu

Celem zadania jest zrozumienie, jak różne tryby szyfrowania blokowego wpływają na zaszyfrowane dane w kontekście dyfuzji i konfuzji, poprzez prostą wizualną analizę obrazów BMP po zaszyfrowaniu.

Polecenia do wykonania

1. **Przygotowanie środowiska:**
   - Upewnij się, że masz zainstalowaną bibliotekę `pycryptodome`. Jeśli nie, zainstaluj ją poleceniem:

     ```
     pip install pycryptodome
     ```

   - Pobierz prosty obraz w formacie BMP, np. `tux-96.bmp`, lub użyj dowolnego innego obrazu BMP o znanej strukturze.
2. **Szyfrowanie obrazu w trybie ECB:**
   - **Wczytanie obrazu:** Napisz skrypt w języku Python, który wczytuje plik BMP w trybie binarnym. Zachowaj nagłówek pliku BMP (pierwsze 54 bajty) bez zmian, ponieważ zawiera on metadane obrazu.
   - **Przygotowanie danych do szyfrowania:** Wyodrębnij dane obrazu (piksele) zaczynając od bajtu 54 do końca pliku. Upewnij się, że dane do szyfrowania są wielokrotnością długości bloku algorytmu AES (16 bajtów). Jeśli nie są, odpowiednio je dopełnij (np. bajtami zerowymi lub za pomocą schematu paddingu PKCS#7).
   - **Szyfrowanie danych:** Wygeneruj klucz szyfrujący o długości 16 bajtów (128 bitów). Możesz użyć dowolnego ciągu bajtów lub funkcji generującej losowy klucz. Użyj algorytmu AES w trybie ECB do zaszyfrowania danych obrazu. Połącz nagłówek pliku BMP z zaszyfrowanymi danymi.
   - **Zapis zaszyfrowanego obrazu:** Zapisz nowy plik BMP z zaszyfrowanymi danymi, np. `tux-96_ecb.bmp`.
3. **Szyfrowanie obrazu w innych trybach:**

   Powtórz kroki z punktu 2, zmieniając jedynie tryb szyfrowania na:
   - **CBC (Cipher Block Chaining):** Wygeneruj wektor inicjalizujący (IV) o długości 16 bajtów. Użyj trybu `AES.MODE_CBC` i przekaż IV do funkcji szyfrującej. Zapisz wynik jako `tux-96_cbc.bmp`.
   - **CFB (Cipher Feedback):** Użyj trybu `AES.MODE_CFB` z tym samym IV. Zapisz wynik jako `tux-96_cfb.bmp`.
   - **OFB (Output Feedback):** Użyj trybu `AES.MODE_OFB` z tym samym IV. Zapisz wynik jako `tux-96_ofb.bmp`.
   - **CTR (Counter):** Użyj trybu `AES.MODE_CTR`. Wygeneruj licznik za pomocą `Crypto.Util.Counter`. Zapisz wynik jako `tux-96_ctr.bmp`.
4. **Analiza wyników:**

   Porównaj wizualnie, jak różne tryby szyfrowania wpływają na widoczność struktury oryginalnego obrazu. Zwróć szczególną uwagę na to, czy kształty i wzory z oryginalnego obrazu są nadal rozpoznawalne po zaszyfrowaniu.
5. **Odszyfrowanie obrazu:**

   Dla jednego z trybów (np. CBC): Napisz kod odszyfrowujący zaszyfrowany obraz. Upewnij się, że używasz tego samego klucza i IV (jeśli dotyczy), co podczas szyfrowania. Zapisz odszyfrowany obraz i porównaj go z oryginałem, aby potwierdzić poprawność procesu szyfrowania i odszyfrowywania.

Podpowiedzi

- Wczytanie pliku jako danych binarnych i wyodrębnienie nagłówka (54 bajty) oraz danych do szyfrowania:

```
with open('tux-96.bmp', 'rb') as f:
    byteblock = f.read()

header = byteblock[:54]
data = byteblock[54:]
```

- Dopełnienie danych do wielokrotności 16 bajtów (rozmiaru bloku AES). Można użyć modułu Crypto.Util.Padding:

```
from Crypto.Util.Padding import pad
data_padded = pad(data, 16)
```

- Wygenerowanie klucz szyfrujący o długości 16 bajtów (128 bitów):

```
key = b'aaaabbbbccccdddd'  # Przykładowy klucz
```

- Użycie algorytmu AES w trybie ECB do zaszyfrowania danych:

```
from Crypto.Cipher import AES
cipher = AES.new(key, AES.MODE_ECB)
ciphertext = cipher.encrypt(data_padded)
```

- Połącz nagłówek z zaszyfrowanymi danymi:

```
encrypted_image = header + ciphertext
```

- Szyfrowanie w innych trybach
  - Tryb CBC (Cipher Block Chaining):

    ```python
    iv = b'0000000000000000'  # Przykładowy IV

    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(data_padded)
    ```

  - Tryb CFB (Cipher Feedback): W tym trybie nie jest wymagane dopełnienie danych:

    ```python
    cipher = AES.new(key, AES.MODE_CFB, iv)
    ciphertext = cipher.encrypt(data)
    ```

  - Tryb OFB (Output Feedback):

    ```python
    cipher = AES.new(key, AES.MODE_OFB, iv)
    ciphertext = cipher.encrypt(data)
    ```

  - Tryb CTR (Counter): ten tryb wymaga generatora licznika:

    ```python
    from Crypto.Util import Counter
    ctr = Counter.new(128)
    cipher = AES.new(key, AES.MODE_CTR, counter=ctr)
    ciphertext = cipher.encrypt(data)
    ```

5. **Odszyfrowanie obrazu (opcjonalnie):** Dla jednego z trybów (np. CBC)

```
# Przygotuj obiekt cipher
cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted_data_padded = cipher.decrypt(ciphertext)
```

```
    # Usuń dopełnienie z odszyfrowanych danych:
    from Crypto.Util.Padding import unpad
    decrypted_data = unpad(decrypted_data_padded, 16)

    # Połącz nagłówek z odszyfrowanymi danymi:
    decrypted_image = header + decrypted_data

    # Zapisz odszyfrowany obraz do pliku:
    with open('tux-96_dec.bmp', 'wb') as f:
        f.write(decrypted_image)
```

Zadanie 2: Zmiana bajtów w plikach

Napisz skrypt w Pythonie, za pomocą którego będzie można dokonać zmian wybranych bajtów w pliku. Wykorzystaj poniższy fragment kodu, w którym zdefiniowano funkcję `modify_btes()`:

```
def modify_bytes(input_filename, output_filename, byte_modifications):
    """
    Funkcja do modyfikacji wybranych bajtów w pliku.

    :param input_filename: Nazwa wejściowego pliku (zaszyfrowanego obrazu BMP)
    :param output_filename: Nazwa wyjściowego pliku z wprowadzonymi zmianami
    :param byte_modifications: Lista krotek zawierających pozycję bajtu i nową wartość bajtu
                               [(position1, new_byte1), (position2, new_byte2), ...]
    """
    # Wczytanie zaszyfrowanego obrazu
    with open(input_filename, 'rb') as f:
        file_data = bytearray(f.read())
```

Zadanie 3: Jak zmiany w szyfrogramie wpływają na tekst jawny?

Celem zadania jest pokazanie, jak zmiany w zaszyfrowanych danych wpływają na wynik odszyfrowywania, poprzez eksperymenty z modyfikacją bajtów w zaszyfrowanych plikach obrazów BMP. Korzystając z wcześniej wprowadzonej funkcji `modify_bytes`, przeprowadź eksperyment polegający na modyfikacji wybranych bajtów w zaszyfrowanym pliku obrazu.

Wybierz kilka bajtów w zaszyfrowanym pliku, które chcesz zmodyfikować, pamiętając, aby modyfikować bajty w sekcji danych obrazu (po nagłówku BMP), aby nie uszkodzić struktury pliku. Następnie, używając tego samego klucza i IV, które były użyte podczas szyfrowania, odszyfruj zmodyfikowany plik. Porównaj oryginalny odszyfrowany obraz z odszyfrowanym obrazem po modyfikacji.

Do pracy niezbędne będzie przygotowanie funkcji, która będzie deszyfrować zaszyfrowane obrazy:

```
def decrypt_image(mode_name, mode, key, iv_or_counter=None):
    # Wczytanie zaszyfrowanego obrazu
    # Zachowanie nagłówka BMP (pierwsze 54 bajty)
    # Inicjalizacja szyfru do odszyfrowywania
    # Odszyfrowanie danych
    # Usunięcie dopełnienia, jeśli dotyczy
    # Połączenie nagłówka z odszyfrowanymi danymi
    # Zapis odszyfrowanego obrazu
```

Wykorzystaj poniższy program:

```
# Główna część skryptu
def main():
    # Klucz i IV muszą być takie same jak podczas szyfrowania
    key = b'aaaabbbbccccdddd'  # Użyj tego samego klucza co w szyfrowaniu

    # Odszyfrowanie w trybie ECB
```

```
    decrypt_image('ECB', AES.MODE_ECB, key)

    # Wektor inicjalizujący (IV) taki sam jak podczas szyfrowania
    iv = b'0000000000000000'  # Użyj tego samego IV co w szyfrowaniu

    # Odszyfrowanie w trybie CBC
    decrypt_image('CBC', AES.MODE_CBC, key, iv_or_counter=iv)

    # Odszyfrowanie w trybie CFB
    decrypt_image('CFB', AES.MODE_CFB, key, iv_or_counter=iv)

    # Odszyfrowanie w trybie OFB
    decrypt_image('OFB', AES.MODE_OFB, key, iv_or_counter=iv)

    # Generowanie licznika dla trybu CTR (musi być taki sam jak podczas szyfrowania)
    ctr = Counter.new(128)
    decrypt_image('CTR', AES.MODE_CTR, key, iv_or_counter=ctr)

if __name__ == '__main__':
    main()
```