Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра защиты информации

Дисциплина: Защита информации в информации в информационных системах

ОТЧЕТ
по лабораторной работе № 4
Вариант 1

Студент:

Преподаватель:

МИНСК 2024

Цель: Целью данной работы является разработка программы для защищенного обмена файлами, которая использует криптографические возможности, предоставляемые Microsoft CryptoAPI.

1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Алгоритм **CALG_RC2** (или RC2) — это симметричный блочный шифр, разработанный для замены DES (Data Encryption Standard). RC2 был разработан Роном Ривестом (Ron Rivest) в 1987 году для компании RSA Data Security. Этот шифр стал известен своей гибкостью, поскольку поддерживает переменную длину ключа, что позволяет выбирать оптимальный уровень безопасности в зависимости от требований.

Основные характеристики RC2:

1. **Переменная длина ключа**: В отличие от DES, который использует фиксированный 56-битный ключ, RC2 поддерживает длину ключа от 1 до 128 бит. Это даёт большую гибкость в выборе уровня безопасности.
2. **Блочный шифр**: RC2 работает с блочными данными фиксированного размера, обычно 64 бита (8 байт). Данные обрабатываются блоками, и для каждого блока применяются операции замены и перестановки, что повышает безопасность шифрования.
3. **Три раунда шифрования**: Хотя RC2 использует меньшее количество раундов, чем более современные шифры, его достаточно для обеспечения хорошей безопасности, особенно при использовании длинных ключей.
4. **Ключевая зависимость**: На безопасность шифра влияют как длина ключа, так и способ его генерации. Это может быть фактором, почему RC2 требует осторожного подхода при реализации для обеспечения должного уровня защиты.
5. **Режимы работы**: RC2 поддерживает несколько стандартных режимов шифрования, включая **ECB** (Electronic Codebook), **CBC** (Cipher Block Chaining) и другие. Эти режимы определяют, как обрабатываются блоки данных и как применяется ключ.

Как работает RC2:

1. **Инициализация ключа**: При применении RC2 к данным используется секретный ключ. Длина ключа может быть от 1 до 128 бит (в большинстве случаев используется 128 битный ключ). Алгоритм сначала обрабатывает этот ключ, чтобы привести его к нужному формату.
2. **Шифрование**: Данные разбиваются на блоки по 64 бита. Каждый блок шифруется с использованием выбранного ключа и различных внутренних операций, таких как замены и перестановки. Эти операции происходят за несколько раундов.

3. **Дешифрование**: При дешифровании используется тот же ключ, что и при шифровании. Дешифровка заключается в применении операций, противоположных шифрованию, чтобы вернуть данные в исходный вид.

Шифрование с использованием RC2:

Алгоритм RC2 применяет несколько этапов:

- **Инициализация**: Перед шифрованием выполняется инициализация состояния ключа, которая подготавливает данные к последующим операциям.
- **Основной процесс**: Алгоритм работает с блоками данных, используя различные операции преобразования (замена, перестановка) для каждого блока данных.
- **Финальный результат**: После обработки всех блоков получается зашифрованный текст, который можно передать или хранить.

Пример: Шифрование с использованием RC2:

1. Данные шифруются с использованием первого ключа.
2. Результат дешифруется с использованием второго ключа.
3. Результат снова шифруется с первым ключом. Этот процесс представляет собой **тройное шифрование**, обеспечивающее улучшенную безопасность по сравнению с классическим DES.

Применения RC2:

RC2 был широко использован в старых системах, включая протоколы безопасности, такие как **SSL** и **TLS**, а также в различных коммерческих приложениях. Однако, с ростом вычислительных мощностей и развитием более безопасных алгоритмов, таких как **AES** и **3DES**, RC2 был постепенно вытеснен. Сегодня RC2 используется ограниченно, и в большинстве современных систем предпочтение отдается более эффективным и безопасным алгоритмам.

Преимущества и недостатки:

- **Преимущества**:
  - Поддержка переменной длины ключа, что позволяет настроить безопасность.
  - Сравнительно быстрый в операциях шифрования и дешифрования.
  - Хорошо работает в старых системах с ограниченными ресурсами.
- **Недостатки**:
  - Слабая безопасность с короткими ключами, что делает его менее подходящим для современных приложений.
  - Устаревший алгоритм, который постепенно вытесняется другими, более безопасными решениями, такими как **AES**.
  -

2 ХОД РАБОТЫ

Содержимое файла input.txt представлена на рисунке 1.1.

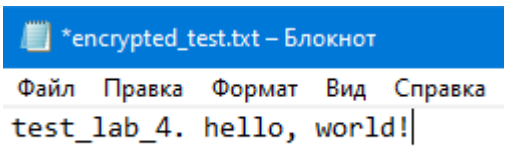

Рисунок 1.1 – input.txt

Консольный интерфейс программы представлен на рисунке 1.2

```
Введите тип операции (encrypt/decrypt): 1
Введите путь к файлу для шифрования: D:/encrypted_test.txt
Файл D:/encrypted_test.txt зашифрован в D:/encrypted_test.txt
Сессионный ключ сохранен в D:/session_key.txt
Введите тип операции (encrypt/decrypt): 2
Файл D:/encrypted_test.txt расшифрован в D:/decrypted_test.txt
Введите тип операции (encrypt/decrypt):
```

Рисунок 1.2 – Графический интерфейс

Выбираем операцию Зашифровать, далее выбираем путь для нужного файла. Затем, если требуется расшифровать, то программа автоматически использует сессионный ключ для файла.

Результат представлен на рисунке 1.3

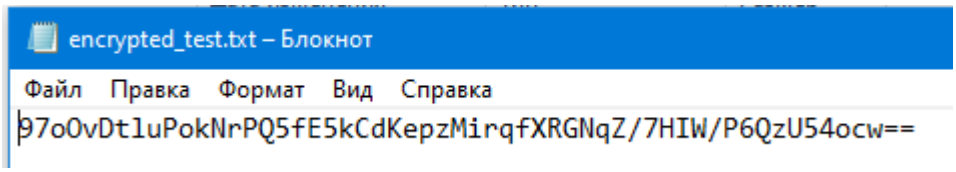

Рисунок 1.3 – Результат шифрования

Расшифровку выполняем по аналогии с шифрованием.
Результат расшифровки представлен на рисунке 1.5.

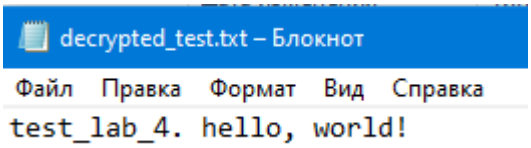

Рисунок 1.5 – Результат расшифровки файла

Листинг кода:

```python
from Crypto.Cipher import ARC2
from Crypto.Util.Padding import pad, unpad
from Crypto.Hash import SHA256
import base64
import os
import secrets


def generate_session_key():
    """Генерация случайного сессионного ключа для RC2 (16 байт)."""
    try:
        return secrets.token_bytes(16)  # 16 байт для RC2
    except Exception as e:
        print(f"Ошибка при генерации сессионного ключа: {e}")


def generate_key(session_key):
    """Генерация ключа для RC2 на основе сессионного ключа."""
    try:
        hasher = SHA256.new()
        hasher.update(session_key)
        return hasher.digest()[:16]  # RC2 использует 16 байтовый ключ
    except Exception as e:
        print(f"Ошибка при генерации ключа: {e}")


def encrypt_file(file_path, session_key):
    """Шифрование файла с использованием RC2."""
    try:
        key = generate_key(session_key)
        cipher = ARC2.new(key, ARC2.MODE_CBC)
        iv = cipher.iv

        with open(file_path, 'rb') as f:
            plaintext = f.read()

        ciphertext = iv + cipher.encrypt(pad(plaintext, cipher.block_size))

        # Кодируем зашифрованные данные в Base64
        encoded_ciphertext = base64.b64encode(ciphertext)

        # Путь для сохранения зашифрованного файла с расширением .txt
        encrypted_file_path = 'D:/encrypted_test.txt'

        with open(encrypted_file_path, 'wb') as f:
            f.write(encoded_ciphertext)
        print(f"Файл {file_path} зашифрован в {encrypted_file_path}")

        # Сохраняем сессионный ключ в файл в формате Base64
        session_key_path = 'D:/session_key.txt'
        with open(session_key_path, 'w') as f:
            f.write(base64.b64encode(session_key).decode('utf-8'))
        print(f"Сессионный ключ сохранен в {session_key_path}")

    except FileNotFoundError:
        print(f"Ошибка: Файл {file path} не найден.")
    except Exception as e:
        print(f"Ошибка при шифровании файла: {e}")


def decrypt_file():
    """Расшифровка файла с использованием RC2."""
    try:
```

```
        # Читаем сессионный ключ из файла
        session_key_path = 'D:/session_key.txt'
        if not os.path.exists(session_key_path):
            print(f"Файл с сессионным ключом {session_key_path} не
существует.")
            return

        with open(session_key_path, 'r') as f:
            session_key = base64.b64decode(f.read().strip())

        key = generate_key(session_key)

        # Путь для чтения зашифрованного файла
        encrypted_file_path = 'D:/encrypted_test.txt'

        if not os.path.exists(encrypted_file_path):
            print(f"Файл {encrypted_file_path} не существует.")
            return

        with open(encrypted_file_path, 'rb') as f:
            encoded_ciphertext = f.read()

        # Декодируем данные из Base64
        ciphertext = base64.b64decode(encoded_ciphertext)
        iv_length = ARC2.block_size

        iv = ciphertext[:iv_length]  # Получаем IV

        # Создаем новый экземпляр шифратора с IV
        cipher = ARC2.new(key, ARC2.MODE_CBC, iv)
        plaintext = unpad(cipher.decrypt(ciphertext[iv_length:]),
cipher.block_size)

        # Путь для сохранения расшифрованного файла с расширением .txt
        decrypted_file_path = 'D:/decrypted_test.txt'

        with open(decrypted_file_path, 'wb') as f:
            f.write(plaintext)
        print(f"Файл {encrypted_file_path} расшифрован в
{decrypted_file_path}")

    except FileNotFoundError:
        print(f"Ошибка: Файл {encrypted_file_path} не найден.")
    except ValueError as ve:
        print(f"Ошибка при расшифровке: {ve}")
    except Exception as e:
        print(f"Ошибка при расшифровке файла: {e}")


if __name__ == "__main__":
    while True:
        try:
            algorithm = 'RC2'  # Устанавливаем алгоритм как RC2

            session_key = generate_session_key()

            operation = input("Введите тип операции (encrypt/decrypt):
").lower()
            if operation == "1":
                file_path = input("Введите путь к файлу для шифрования: ")
                encrypt_file(file_path, session_key)
            elif operation == "2":
                decrypt_file()
```

```
            else:
                print("Неизвестная операция. Пожалуйста, используйте
'encrypt' или 'decrypt'.")
        except Exception as e:
            print(f"Ошибка: {e}")
```

3 ЗАКЛЮЧЕНИЕ

В ходе работы была реализована программа для защищенного обмена файлами с использованием криптографических алгоритмов Microsoft CryptoAPI, обеспечивающая шифрование и дешифрование файлов. Программа продемонстрировала эффективное использование симметричного и асимметричного шифрования для защиты данных и безопасного обмена информацией.