

ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Направление подготовки бакалавриата
09.03.04 — Программная инженерия

Отчет по курсу
«Криптографические средства»
ЛАБОРАТОРНАЯ РАБОТА №3
Вариант №6

Выполнил:
студент 3 курса группы 22307
Меньщиков Михаил

Петрозаводск — 2022

Содержание

1	Формулировка задания	3
2	Описание метода решения	4
2.1	Шифрование приватного ключа RSA	5
2.2	Дешифрация приватного ключа RSA	6
2.3	Шифрование документа	6
2.4	Дешифрация документа	7
2.5	Генерация цифровой подписи документа	7
2.6	Верификация цифровой подписи документа	7
3	Примеры кода программы	9
3.1	Импортируемые функции библиотеки Crypto	9
3.2	Генерация ключей RSA	9
3.3	Шифрование с помощью RSA	10
3.4	Дешифрация с помощью RSA	10
3.5	Дешифрвация приватного ключа RSA	11
3.6	Генерация ключа AES	11
3.7	Шифрование с помощью AES	12
3.8	Дешифрация с помощью AES	12
3.9	Создание цифровой подписи	13
3.10	Верификация цифровой подписи	13
4	Тестовые данные	14
4.1	Генерация ключей RSA	14
4.2	Шифрование данных	14
4.3	Дешифрация данных	17

1 Формулировка задания

При помощи функций криптографических библиотек реализовать гибридную криптосистему, включающую:

- 1) генерацию ключевой пары RSA;
- 2) шифрование и расшифрование документа симметричным криптоалгоритмом;
- 3) шифрование и расшифрование сеансового ключа симметричного алгоритма при помощи ключей RSA;
- 4) формирование и проверку цифровой подписи документа.

Полученный шифротекст, открытые ключи должны сохраняться и передаваться через файлы.

2 Описание метода решения

Гибридная криптосистема реализована в виде консольной программы на языке Python. Для использования криптографических алгоритмов использовалась библиотека Crypto[2].

Взаимодействие с программой происходит за счёт ввода в консоль определённых команд:

- 1) **gen keys** - Генерация пары ключей для алгоритма RSA. Пользователь может выбирать значение параметра e и размер ключей. Сгенерированный приватный ключ будет зашифрован с помощью алгоритма AES. В качестве ключа AES используется введённая пользователем фраза. Также пользователь указывает имя каталога, в который будут сохранены сгенерированные ключи.
- 2) **encrypt** - Шифрование данных с помощью алгоритмов RSA, AES и создание цифровой подписи. Пользователь указывает каталог с документом для шифрования и публичным ключом, который получен от второго лица, а также указывает каталог с RSA ключами, которые будут использоваться при создании цифровой подписи. Для использования выбранного приватного ключа RSA потребуется ввести ключевую фразу для его дешифровки с помощью AES. Цифровая подпись будет создана на основе имеющегося приватного ключа RSA и алгоритма хеширования SHA512. С помощью алгоритма AES будет сгенерирован сессионный ключ и зашифрован указанный документ. Сам сессионный ключ будет также зашифрован с помощью публичного ключа, полученного от второго лица. Пользователь указывает название каталога, который будет создан для сохранения полученных данных.
- 3) **decrypt** - Дешифрация данных с помощью алгоритмов RSA, AES и верификация цифровой подписи. Пользователь указывает каталог с данным для дешифрации, а также каталог с RSA ключами. Для использования выбранного приватного ключа RSA потребуется ввести ключевую фразу для его дешифровки с помощью AES. С помощью имеющегося приватного ключа

RSA будет расшифрован сессионный ключ и впоследствии дешифрован документ. Пользователь указывает название каталога, который будет создан для сохранения дешифрованного документа. С помощью полученного от второго лица публичного ключа RSA и используемого алгоритма хеширования SHA512 будет проверена подлинность цифровой подписи документа.

Программа требует определённый формат упаковки данных для их последующего использования. Программа использует следующие каталоги для получения и сохранения данных:

- 1) `./rsa_keys/` - Директория для хранения каталогов с парами ключей RSA. Сами публичный и приватный ключи должны быть сохранены в файлы `public_key` и `private_key` соответственно.
- 2) `./encryption/` - Директория для хранения каталогов с данными для шифрования. В данных каталогах должны находиться следующие файлы:
 - (a) Файл `encrypt` с документом для шифрования.
 - (b) Файл `public_key_recieved` с публичным ключом RSA, полученным от второго лица.
- 3) `./decryption/` - Директория для хранения каталогов с данными для дешифрации. В данных каталогах должны находиться следующие файлы:
 - (a) Файл `decrypt` с зашифрованным документом.
 - (b) Файл `public_key_sended` с публичным ключом RSA.
 - (c) Файл `session_key` с зашифрованным сессионным ключом AES.
 - (d) Файл `signature` с цифровой подписью документа.
- 4) `./results/` - Директория для хранения каталогов с результатами дешифрации документов. В данных каталогах находится файл `decrypted_file` с расшифрованным документом.

2.1 Шифрование приватного ключа RSA

Выбираем ключ `key_phrase`, с помощью которого будет зашифрован приватный ключ `pri_k` на основе алгоритма AES. Ключ `key_phrase` имеет размер 16 байт.

Генерируем инициализирующий вектор *iv* размером 16 байт.

Добавляем ключ *key_phrase* в начало *pri_k*.

С помощью алгоритма AES в режиме CFB[3] с параметрами *key_phrase* и *iv* шифруем полученную композицию *compos_pri_k*. Инициализирующий вектор *iv* добавляем в начало зашифрованной композиции *dec_compos_pri_k*.

2.2 Дешифрация приватного ключа RSA

Выбираем зашифрованный приватный ключ *dec_compos_pri_k*.

Выбираем ключ *key_phrase'*, с помощью которого будет дешифрован *dec_compos_pri_k*.

Достаём из начала *dec_compos_pri_k* инициализирующий вектор *iv*.

С помощью *key_phrase'* и *iv* на основе алгоритма AES дешифруем начало *dec_compos_pri_k*, где храниться *key_phrase*.

Если *key_phrase' = key_phrase*, то дешифруем оставшуюся часть *dec_compos_pri_k* и достаём приватный ключ *pri_k*.

2.3 Шифрование документа

Шаг 1. Выбираем документ для шифрования *doc* и публичный ключ *pub_k* RSA, полученный от второго лица.

Шаг 2. Генерируем сессионный ключ *session_key* размером 32 байта.

Шаг 3. Генерируем инициализирующий вектор *iv* размером 16 байт.

Шаг 4. С помощью алгоритма AES в режиме CFB [3] с параметрами *session_key* и *iv* шифруем документ *doc*. Инициализирующий вектор *iv* добавляем в начало зашифрованного документа *dec_doc*.

Шаг 5. С помощью алгоритма RSA и полученного от второго лица публичного ключа *pub_k* шифруем сессионный ключ *session_key*.

Шаг 6. Зашифрованный сессионный ключ *dec_session_key* и зашифрованный документ *dec_doc* передаём второму лицу.

2.4 Дешифрация документа

- Шаг 1.** Выбираем зашифрованный сессионный ключ $dec_session_key$ и зашифрованный документ dec_doc полученный от второго лица.
- Шаг 2.** Выбираем ключи алгоритма RSA, где pub_k был отправлен второму лицу для шифрации данных.
- Шаг 3.** С помощью приватного ключа pri_k и алгоритма RSA дешифруем сессионный ключ $dec_session_key$.
- Шаг 4.** Достаём из начала документа dec_doc инициализирующий вектор iv .
- Шаг 5.** С помощью расшифрованного сессионного ключа $session_key$ и iv дешифруем документ dec_doc .

2.5 Генерация цифровой подписи документа

- Шаг 1.** Выбираем публичный pub_k и приватный pri_k ключи алгоритма RSA.
- Шаг 2.** Выбираем документ doc , для которого будет создана цифровая подпись.
- Шаг 3.** С помощью алгоритма SHA512 будет получен хеш $hash$ документа doc .
- Шаг 4.** Вычисляем цифровую подпись $signature$ с помощью $hash$ и приватного ключа RSA pri_k с параметрами (d, n) по формуле:

$$signature = hash^d \mod n.$$

- Шаг 5.** Передаём $signature$, pub_k и зашифрованный документ dec_doc второму лицу.

2.6 Верификация цифровой подписи документа

- Шаг 1.** Выбираем цифровую подпись $signature$, публичный ключ RSA pub_k и зашифрованный документ dec_doc с параметрами (e, n) , полученные от второго лица.
- Шаг 2.** Расшифровываем dec_doc .

Шаг 3. С помощью алгоритма SHA512 получаем хеш $hash'$ расшифрованного документа doc .

Шаг 4. Цифровая подпись документа doc является подлинной если выполняется следующее равенство:

$$signature^e \mod n = hash'.$$

3 Примеры кода программы

Весь исходный код программы можно посмотреть и скачать из репозитория [github](#).

3.1 Импортируемые функции библиотеки Crypto

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Hash import SHA512
```

3.2 Генерация ключей RSA

```
# @brief Генерация публичного и приватного ключей
#         для алгоритма RSA.
#
# @param params Параметры для алгоритма RSA.
#
# @return публичный и приватный ключи,
#         если операция завершилась успешно, иначе None.
def gen_rsa(params):
    try:
        private_key = RSA.generate(changable_rsa_params['size'],
                                   e=changable_rsa_params['e'])

        public_key = private_key.publickey()
    except ValueError:
        error_msg(error_msgs["bad_rsa_gen"])
        return None, None
    else:
        return (bytes(public_key.exportKey('PEM')),
                bytes(private_key.exportKey('PEM')))
```

3.3 Шифрование с помощью RSA

```
# @brief Шифрование данных при помощи алгоритма RSA.
#
# @param data Исходные данные.
# @param params Параметры для алгоритма RSA.
#
# @return Зашифрованные данные, если операция завершилась успешно,
#         иначе None
def encrypt_rsa(data, params):
    try:
        cipherrsa = PKCS1_OAEP.new(RSA.importKey(params[0]))
        encrypted_data = cipherrsa.encrypt(data)
    except ValueError:
        error_msg(error_msgs["bad_enc"])
        return None
    else:
        return bytes(encrypted_data)
```

3.4 Дешифрация с помощью RSA

```
# @brief Расшифровка данных при помощи алгоритма RSA.
#
# @param data Зашифрованный набор данных.
# @param params Параметры для работы RSA.
#
# @return Расшифрованные данные, если операция завершилась успешно,
#         иначе None.
def decrypt_rsa(data, params):
    try:
        cipherrsa = PKCS1_OAEP.new(RSA.importKey(params[0]))
        decrypted_data = cipherrsa.decrypt(data)
    except ValueError:
        error_msg(error_msgs["bad_dec"])
        return None
    else:
        return bytes(decrypted_data)
```

3.5 Дешифровка приватного ключа RSA

```
# @brief Проверка пароля и дешифрация приватного ключа RSA.
#
# @param Пароль, введённый пользователем.
# @param Зашифрованный приватный ключ.
#
# @return Расшифрованный приватный ключ, если введённый пароль верен,
#         иначе None.
def decrypt_privat_key(passwd, encrypted_priv_key):
    decrypted_pass = decrypt_aes(
        encrypted_priv_key[: (iv_size + eas_key_size)], [passwd])

    if not decrypted_pass == passwd:
        error_msg(error_msgs["inv_pass"])
        return None
    decrypted_pri_k = decrypt_aes(
        encrypted_priv_key, [passwd])[eas_key_size:]

    return decrypted_pri_k
```

3.6 Генерация ключа AES

```
# @brief Сгенерировать сессионный ключ с помощью алгоритма AES.
#
# @param params Параметры для алгоритма AES.
#
# @return Сессионный ключ, если операция завершилась успешно,
#         иначе None.
def gen_aes(params=[]):
    try:
        session_key = Random.new().read(session_key_size)
    except:
        error_msg(error_msgs["bad_aes_gen"])
        return None
    else:
        return session_key
```

3.7 Шифрование с помощью AES

```
# @brief Зашифровать набор данных при помощи алгоритма AES.
#
# @param data Набор исходных данных.
# @param params Параметры для алгоритма AES.
#
# @return Зашифрованные данные, если операция завершилась успешно,
#         иначе None.
def encrypt_aes(data, params):
    try:
        iv = Random.new().read(iv_size)
        obj = AES.new(params[0], AES.MODE_CFB, iv)
        encrypted_data = iv + obj.encrypt(data)
    except ValueError:
        error_msg(error_msgs["bad_enc"])
        return None
    else:
        return bytes(encrypted_data)
```

3.8 Дешифрация с помощью AES

```
# @brief Расшифровать набор данных при помощи алгоритма AES.
#
# @param data Набор зашифрованных данных.
# @param params Параметры для алгоритма AES.
#
# @return Расшифрованные данные, если операция завершилась успешно,
#         иначе None.
def decrypt_aes(data, params):
    try:
        iv = data[:iv_size]
        obj = AES.new(params[0], AES.MODE_CFB, iv)
        decrypted_data = obj.decrypt(data)
        decrypted_data = decrypted_data[iv_size:]
    except ValueError:
        error_msg(error_msgs["bad_dec"])
        return None
    else:
        return decrypted_data
```


3.9 Создание цифровой подписи

```
# @brief Генерация цифровой подписи набора данных.
#
# @param data Набор данных.
# @param params Параметры для генерации цифровой подписи.
#
# @return цифровая подпись, если операция завершилась успешно,
#         иначе None
def gen_signature(data, params):
    try:
        signer = PKCS1_v1_5.new(RSA.importKey(params[0]))
        hash_value = SHA512.new(data)

    except ValueError:
        error_msg(error_msgs["bad_sign_gen"])
        return None

    else:
        return signer.sign(hash_value)
```

3.10 Верификация цифровой подписи

```
# @brief Проверка цифровой подписи набора данных.
#
# @param signature Цифровая подпись.
# @param params Параметры для проверки цифровой подписи.
# @param data Набор данных.
#
# @return True, если цифровая подпись подлинная,
#         иначе False.
def verify_sifnature(signature, params, data):
    hash_value = SHA512.new(data)
    verifier = PKCS1_v1_5.new(RSA.importKey(params[0]))
    stat = verifier.verify(hash_value, signature)

    if stat:
        print("Цифровая подпись верифицирована.")
    else:
        print("Цифровая подпись недействительна.")

    return stat
```

4 Тестовые данные

4.1 Генерация ключей RSA

```
Введите команду >> gen keys
Проверяем наличие нужной папки.
Warning: Директория уже существует.
/home/dzigen/University/inf4/Crypto/crypto/rsa_keys
Воспользуйтесь командой "help".
Готово.

Для генерации ключей будут использоваться стандартные настройки.
Значение параметра e: 65537
Размер ключей в битах: 2048
Хотите изменить настройки? (y/n) >> n
Выбираем имя каталога для сохранения сгенерированных ключей.
Введите имя каталога для создания >> first
Готово.

Генерация ключей RSA.
Готово.

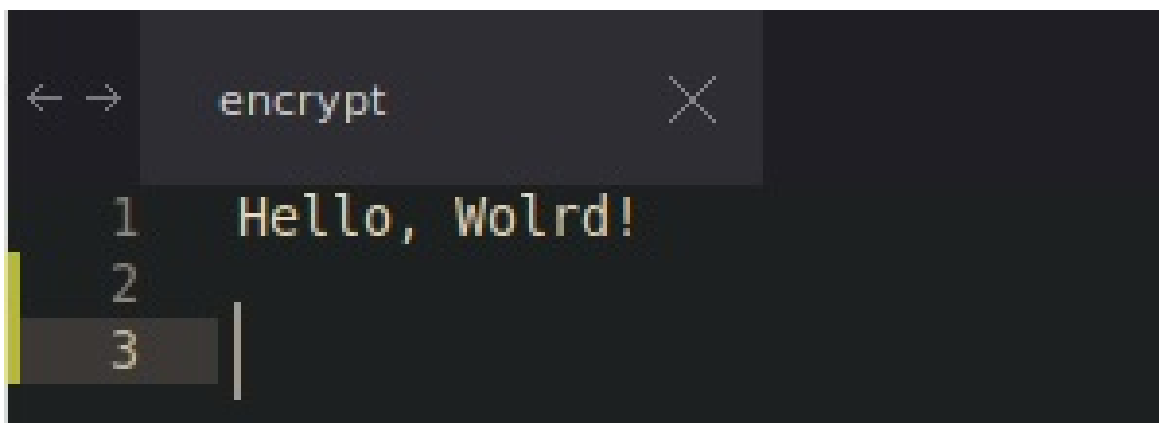
Шифруем приватный ключ RSA.
Введите пароль для шифрования приватного ключа >> first
Готово.
Создаём директорию для сохранения ключей RSA
Готово.

Сохраняем ключи RSA в созданный каталог.
Готово.

Введите команду >> █
```

В каталог `./rsa_keys/first/` будут сохранены сгенерированные публичный и приватный ключи для алгоритма RSA. Аналогичным образом генерируем новую пару ключей алгоритма RSA и сохраняем их в каталог `./rsa_keys/second/`.

4.2 Шифрование данных



Перед началом шифрование в директории `./encryption/` создан каталог `first/` и в него добавлены следующие файлы:

- 1) Файл `encrypt` с документом для шифрования.
- 2) Файл `public_key_recieved` с публичным ключём RSA, взятого из каталога `./rsa_keys/second/`.

```
Введите команду >> encrypt
Проверяем наличие нужной папки.
Warning: Директория уже существует.
/home/dzigen/University/inf4/Crypto/crypto/rsa_keys
Воспользуйтесь командой "help".
Warning: Директория уже существует.
/home/dzigen/University/inf4/Crypto/crypto/encryption
Воспользуйтесь командой "help".
Готово.

Выбираем директорию с RSA ключами.
Список каталогов:
1. second .
2. first .
Выберите нужный каталог >> 2
Готово.

Выбор директории с данными для шифрования
Список каталогов:
1. first .
Выберите нужный каталог >> 1
Готово.

Создание имени новой директории для сохранения зашифрованных данных.
Введите имя каталога для создания >> encrypted_data
Готово.

Чтение публичного и приватного ключей RSA
Готово.

Дешифруем приватный ключ RSA.
Введите пароль для дешифровки приватного ключа >> first
Готово.
```

```
Чтение данных для шифрования
Готово.

Чтение полученного публичного ключа для использования при шифровании
Готово.

Генерация цифровой подписи данных для шифрования
Готово.

Генерация сессионного ключа алгоритмом AES
Готово.

Шифрование исходного набора данных при помощи AES
Готово.

Шифрование сессионного ключа с помощью RSA
Готово.

Создание папки для сохранения зашифрованных данных
Готово.

Сохраняем зашифрованные данные
Готово.

Сохраняем цифровую подпись
Готово.

Сохраняем зашифрованный сессионный ключ
Готово.

Сохраняем личный публичный ключ для передачи второй стороне
Готово.

Введите команду >> █
```

В каталог `./decryption/encrypted_data/` будут сохранены следующие файлы:

- 1) Файл `decrypt` с зашифрованным документом.
- 2) Файл `session_key` с зашифрованным сессионным ключом.
- 3) Файл `signature` с цифровой подписью зашифрованного документа.
- 4) Файл `public_key_sended` с публичным ключом RSA, взятого из каталога `./rsa_keys/first_keys/`.

4.3 Дешифрация данных

```
Введите команду >> decrypt
Проверяем наличие нужной папки.
Warning: Директория уже существует.
/home/dzigen/University/inf4/Crypto/crypto/rsa_keys
Воспользуйтесь командой "help".
Warning: Директория уже существует.
/home/dzigen/University/inf4/Crypto/crypto/decryption
Воспользуйтесь командой "help".
Warning: Директория уже существует.
/home/dzigen/University/inf4/Crypto/crypto/results
Воспользуйтесь командой "help".
Готово.

Выбираем директорию с RSA ключами.
Список каталогов:
1. second .
2. first .
Выберите нужный каталог >> 1
Готово.

Выбор директории с данными для расшифрования.
Список каталогов:
1. encrypted_data .
Выберите нужный каталог >> 1
Готово.

Создание имени новой директории для сохранения расшифрованных данных.
Введите имя каталога для создания >> decrypted_data
Готово.

Чтение публичного и приватного ключей RSA
Готово.

Дешифруем приватный ключ RSA.
Введите пароль для дешифровки приватного ключа >> second
Готово.
```

```
Чтение данных для расшифрования
Готово.

Чтение полученного публичного ключа
Готово.

Чтение зашифрованного сессионного ключа.
Готово.

Чтение цифровой подписи шифрованного документа.
Готово.

Расшифровываем сессионный ключ с помощью RSA.
Готово.

Расшифровываем данные с помощью сессионного ключа AES.
Готово.

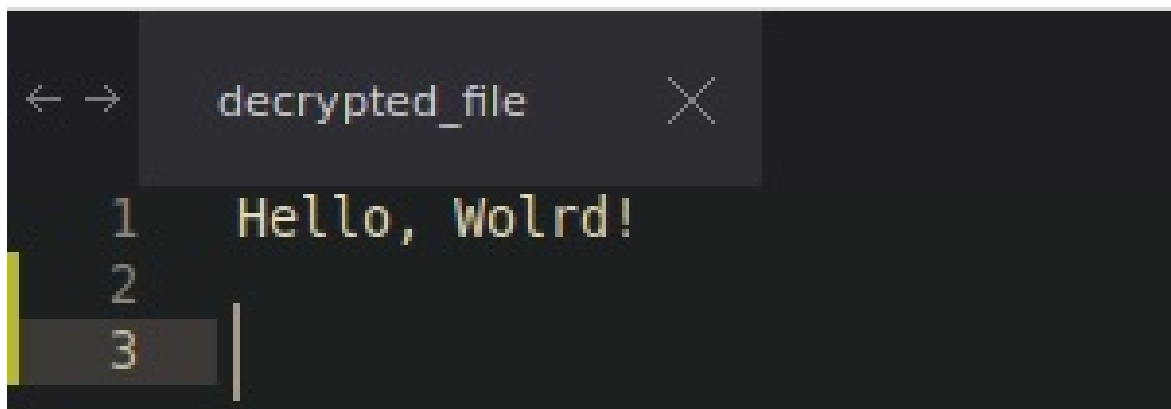
Верифицируем цифровую подпись.
Цифровая подпись верифицирована.
Готово.

Создание папки для сохранения расшифрованных данных
Готово.

Сохраняем расшифрованные данные в созданную папку.
Готово.

Введите команду >> █
```

В директории `./results` будет создан каталог `decrypted_data` и в него сохранен файл `decrypted_file` с расшифрованным документом.



The screenshot shows a file viewer window with the title bar "decrypted_file". The main area displays the text "Hello, Wolrd!". On the left side, there is a list of line numbers: 1, 2, and 3. Line 1 is highlighted in yellow, and line 3 is highlighted in dark gray. The text "Hello, Wolrd!" is on line 1.

Список литературы

1. Воронов Р. В. Основы асимметричных криптосистем : учебное пособие / Петрозаводский Государственный Университет
2. Crypto [Электронный ресурс]. Режим доступа:
<https://pycryptodome.readthedocs.io/en/latest/src/cipher/cipher.html> (дата обращения : 04.04.2022)
3. Режим CFB алгоритма AES [Электронный ресурс]. Режим доступа:
<https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#cfb-mode> (дата обращения : 05.04.2022)