# 1 Криптография простым языком: разбираем симметричное и асимметричное шифрование на примере сюжета Звездных войн (Updated)

Привет всем читателям Хабра! Не так давно решил разобраться с алгоритмами шифрования и принципами работы электронной подписи. Тема, я считаю, интересная и актуальная. В процессе изучения попробовал несколько библиотек, однако самой удобной с моей точки зрения является библиотека PyCrypto. У неё прекрасная документация, сопровождаемая примерами.


**После прочтения материала вы усвоите следующие моменты:**

1. Что такое шифрование;
2. Чем отличается симметричное шифрование от асимметричного;
3. В каком случае эффективнее применять симметричное, а в каких асимметричное шифрование;
4. Что такое хеш данных и для чего он используется в шифровании;

Актуальность рассматриваемой темы постоянно растет. Применение криптографии уже давно не ограничивается шифрованием информации. Алгоритмы шифрования в том или ином виде ежедневно используется нами при посещении сайтов через протокол HTTPS, во время совершения покупок банковской картой, при общении в мессенджерах. Последние несколько лет широкое внимание привлекают блокчейн технологии, основой которых также является криптография.

Целью данной статьи является познакомить читателя с основными алгоритмами шифрования. При написании статьи, я постарался как можно большее внимание уделить вопросу практического применения. Для программирования использовался язык Python 3.6. При написании кода старался делить его на отдельные части и комментировать все ключевые моменты.

В данной статье я не разбирал цифровую подпись, однако после понимания асимметричного шифрования смысл этой технологии станет понятен.

## 1.1 Сюжет

Давайте мысленно перенесемся во вселенную Звездных войн до событий Эпизода 6, когда силам сопротивления становится известно о начале строительства новой Звезды смерти. Командование планирует внедрить разведывательную группу под видом строителей. Операция очень опасна, связь со штабом будет затруднена. В случае экстренной стиуации каждый член группы может отправлять и получать сообщения из штаба на незащищенной частоте. Целью

разведгруппы являются любые данные, которые могут пролить свет на конфигурацию, вооружение и назначение будущей станции. Для хранения данных планируется разработать специальное оборудование и ПО.

**Штаб утвердил два варианта этой операции:**

План А — возвращение агентов с данными повстанческим силам;
План Б — дистанционная передача планов с самой Звезды смерти, используя оборудование станции.
Передача информации при этом будет быстрой, но после передачи агент вероятнее всего будет вычислен и пойман.

Вы являетесь программистом в команде, которая отвечает за разработку ПО.

**При планировании операции рассматриваются несколько возможных негативных сценариев:**

- Противник перехватит сигнал, поймет по его содержимому о планировании атаки и уведет объект ближе к лояльным Империи силам. В этом случае потери среди сопротивления будут выше;
- Один из шпионов будет пойман и на допросе раскроет план операции, что может привести к компрометации ключей шифрования (про них будет сказано ниже);
- Шпион с загруженными данными может быть перехвачен имперскими силами, которые внесут изменения в содержимое, дезинформируя сопротивление о слабых местах станции. В этом случае, при атаке флот повстанцев будет направлен в ложном направлении и постепенно уничтожен;

**Из этих сценариев смормулированы задачи:**

1. Содержимое должно быть надежно зашифровано и защищено от изменений;

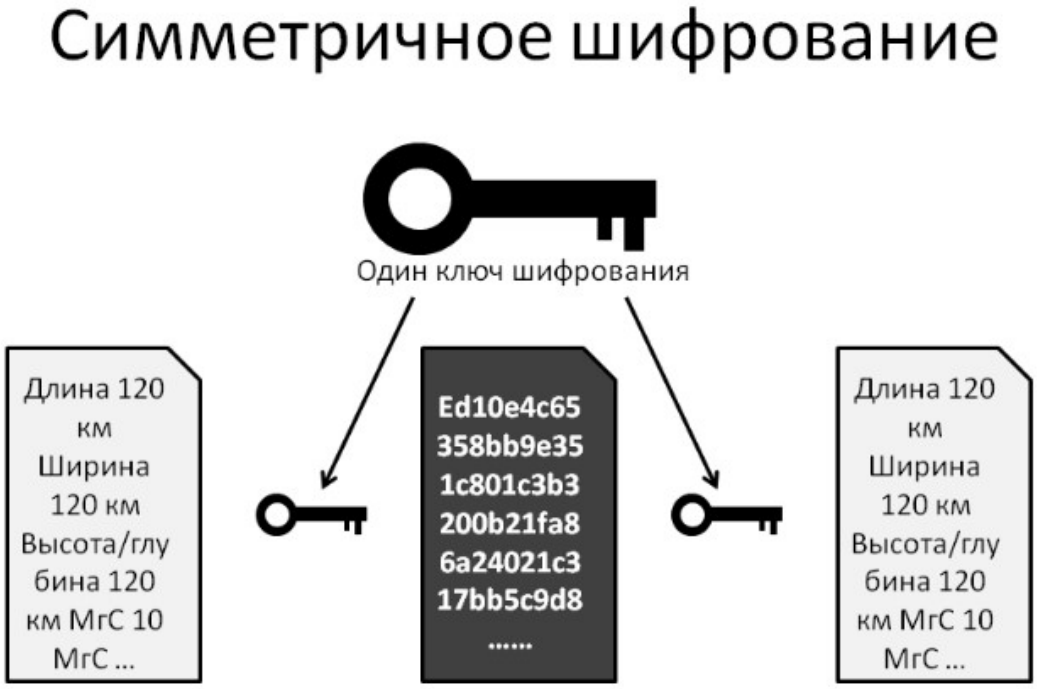
2. В случае утери ключей шифрования или их компрометации, должна быть возможность получения новых ключей шифрования дистанционно на частоте, которая может прослушиваться противником.

## 1.2 Шифрование информации

Давайте решим проблему шифрования информации:

Для шифрования и дешифрования информации используется ключ шифрования. Именно ключ делает шифрование обратимым. Каждый агент будет снабжен ключом шифрования. После загрузки данных агент произведет их шифрацию и отправку в штаб сопротивления.

Метод, при котором шифрование и дешифрация сообщения производится при помощи одного ключа называется симметричное шифрование.




Слабым местом симметричного шифрования является ключ шифрования, точнее его доставка до адресата. Если во время доставки ключ будет скомпрометирован, стороннее лицо легко раскодирует сообщение. Сильной стороной симметричного шифрования является его скорость, что дает возможность кодировать большие объемы данных.

Асимметричное шифрование для кодирования данных использует два связанных друг с другом ключа: открытый и закрытый.

# Асимметричное шифрование


**Механизм действия такой:**

1. адресат отправляет ОТКРЫТЫЙ ключ отправителю;
2. отправитель кодирует сообщение при помощи полученного открытого ключа. При этом, раскодировать сообщение можно теперь только закрытым ключом;
3. при получении зашифрованного сообщения адресат раскодирует его ЗАКРЫТЫМ ключом (который был сгенерирован в паре с открытым).

Для начала разработаем функционал для симметричного шифрования по названием Advanced Encryption Standard (AES). Он является одним из самых распространённых алгоритмов симметричного шифрования.

```python
from Crypto.Cipher import AES # алгоритм шифрования
from Crypto.Hash import SHA256 # Для хеширования данных используем также популярный алгоритм SHA.
from Crypto.Hash import MD5 # Этот алгоритм хеширования будет использован для приведения произвольной строки пароля к 32 битной
from Crypto import Random


def transform_password(password_str):
    """Transform the password string into 32 bit MD5 hash

    :param password_str: <str> password in plain text;
    :return: <str> Transformed password fixed length

    """
    h = MD5.new()
    h.update(key.encode())
    return h.hexdigest()

def symmetric_encrypt(message, key, verbose = True):
    """Encripts the message using symmetric AES algorythm.

    :param message: <str> Message for encryption;
    :param key: <object> symmetric key;
    :return: <object> Message encrypted with key

    """

    key_MD5 = transform_password(key) # Приводим произвольный пароль к длине 32 бита
    message_hash = SHA256.new(message.encode())
    message_with_hash = message.encode() + message_hash.hexdigest().encode() #Добавим в конец сообщения его хеш. он понадобится нам при расшифровки
    iv = Random.new().read(AES.block_size)
    cipher = AES.new(key_MD5, AES.MODE_CFB, iv) # Создаем объект с заданными параметрами. AES.MODE_CFB - надежный режим шифрования, который предполагает наличие вектора инициализации iv. https://www.dlitz.net/software/pycrypto/api/current/Crypto.Cipher.blockalgo-module.html#MODE_CFB
    encrypted_message = iv + cipher.encrypt(message_with_hash) # Включаем случайную последовательность в начало шифруемого сообщения. Это необходимо, чтобы в случае кодирования нескольких блоков текста, аналогичные блоки не давали одинаковые кодированные сообщения.
    if verbose:
        print(f'Message was encrypted into: {encrypted_message.hex()}')
    return encrypted_message
```

```
def symmetric_decrypt(encr_message, key):
    """Decripts the message using private_key and check it's hash

    :param encrypted_message: <object> Encrypted message
    :param key: <object> symmetric key;
    :return: <object> Message decripted with key

    """
    key_MD5 = transform_password(key)

    # Размеры боков нужны, для извлечения их из текста
    bsize = AES.block_size
    dsize = SHA256.digest_size*2

    iv = Random.new().read(bsize)
    cipher = AES.new(key_MD5, AES.MODE_CFB, iv)
    decrypted_message_with_hesh = cipher.decrypt(encr_message)[bsize:] # Извлекаем из блока слу
чайные символу, которые мы добавляли при шифровании
    decrypted_message = decrypted_message_with_hesh[:-dsize] # Извлекаем хеш сообщения, который
мы присоединяли при шифровании
    digest = SHA256.new(decrypted_message).hexdigest() # хеш расшифрованной части сообщения. Он
будет сравниваться с хешем, который мы присоединили при шифровании.

    if digest==decrypted_message_with_hesh[-dsize:].decode(): # Если хеш расшифровааного сообще
ния и хеш, который мы добавили при шифровании равны, расшифровка правильная
        print(f"Success!\nEncrypted hash is {decrypted_message_with_hesh[-dsize:].decode()}\nDe
crypted hash is {digest}")
        return decrypted_message.decode()
    else:
        print(f"Encryption was not correct: the hash of decripted message doesn't match with en
crypted hash\nEncrypted hash is {decrypted_message_with_hesh[-dsize:]}\nDecrypted hash is {dige
st}")
```

Проверим работоспособность кода на примере

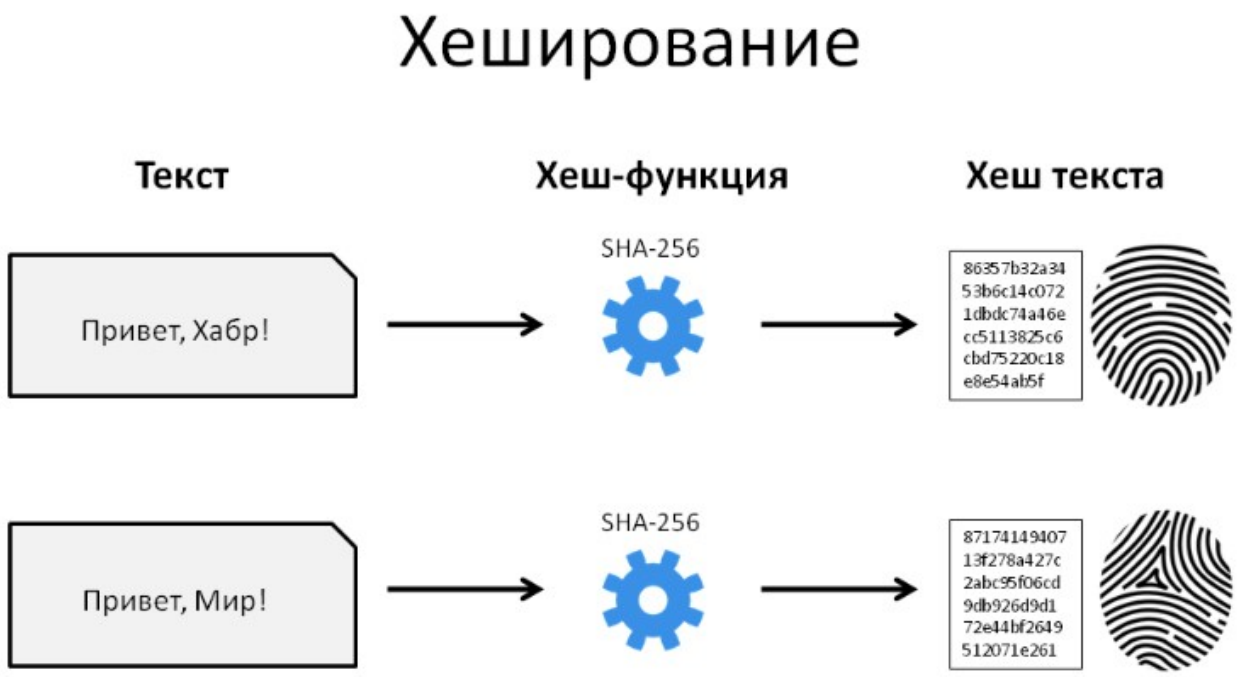
```
message = """
Длина 120 км
Ширина 120 км
Высота/глубина 120 км
МгС 10 МгС
Двигатель Субсветовой двигатель 30-5 (2)
Класс гиперпривода Класс 4.0

"""
key = 'Traveling through hyperspace ain't like dusting crops, farm boy.'
encr_message = symmetric_encrypt(message, key, verbose = True)
print('\n')
print('DECRIPTION')
decr_message = symmetric_decrypt(encr_message, key)
print(decr_message)
```

```
Message was encrypted into: ed10e4c65358bb9e351c801c3b3200b21fa86a24021c317bb5c9d8b3f76bdf9f3
a7d26781a22402f0e4f41ca831b6d2da9e1e6878c34c79ddc7959af3ae9fc2ba0cfff1c0180a7e0f637f1aa5b2450
7d552d5dfe7625e7b81d817b5882b2b19bb95f3988a03c78f850098dfc8e6089863deaa39b887eaea4c1d4ba006ed
aec90205d54b27ed4ac70ed75cdd01732e1176bf04218beb8ae742ff708a201a9d1cb57dd5f2e70dc3239208d2370
5f7a3aae3e315c4df6d73c871b66c4995cce5f19738f731cd58755d21ed92612c44197f875cddf3f7aa1d60e435ce
1492679b9d60c4b8538f52408f321711ac1d2daa6dbbc33dc655abca10e2f5fd3ff27823995b9dcdb62c0bafc1963
ab539ccb466f1c140479df34b0005f578f72fcdd76b17391332037b801f74f733a08


DECRIPTION
Success!
Encrypted hash is b0dbb35b28fbff258350a50c39282b73e31f408c9da937c81d8d48115b491026
Decrypted hash is b0dbb35b28fbff258350a50c39282b73e31f408c9da937c81d8d48115b491026

Длина 120 км
Ширина 120 км
Высота/глубина 120 км
МгС 10 МгС
Двигатель Субсветовой двигатель 30-5 (2)
Класс гиперпривода Класс 4.0
```

Как видим, мы успешно зашифровали сообщение при помощи открытого ключа. Запустите код несколько раз. Вы увидите, что каждый раз зашифрованная часть меняется. Это происходит потому что при шифровании мы применили режим Cipher FeedBack (AES.MODECFB), при котором блоки открытого текста смешиваются с блоками шифротекста. iv — вектор инициализации (подробнее про режим читайте здесь). При расшифровке сообщения мы видим, что хеш расшифрованного сообщения совпадает с хешем, который мы добавляли при шифровании. Это значит, что дешифрация прошла корректно.

## 1.3 Что такое хэш ?

Хеш документа — это просто строка из символов, которая уникальна для какого-либо набора данных. При любом изменении данных хеш очень сильно меняется. Другими словами, хеш — это своеобразный «отпечаток пальца» для какого-либо набора данных.




Но что делать, если ключи шифрования будут по каким-то причинам скомпрометированы? Тогда расшифровать информацию может кто угодно.

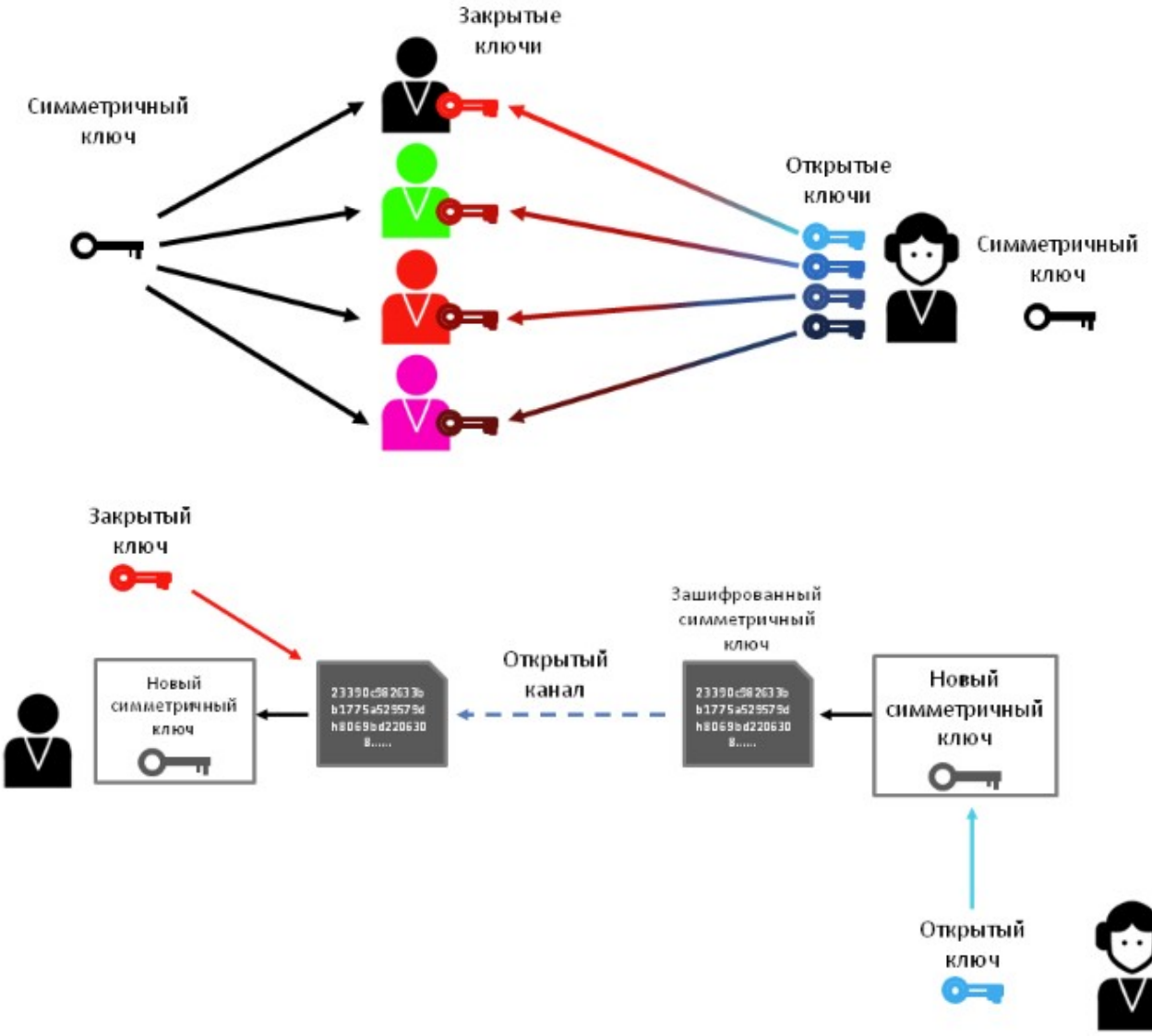
В этом случае нам нужно как-то сменить ключи шифрования дистанционно по частоте, которая может прослушиваться противником. Будем считать, что ее уже слушают. Так каким же образом нам это сделать? Тут на помощь приходит другой метод под названием асимметричное шифрование (или криптографическая система с открытым ключом). В отличие от симметричного шифрования, при ней используется два ключа: открытый и закрытый. Сообщение шифруется открытым ключом, после этого расшифровать его можно только закрытым ключом. Открытый ключ при расшифровке будет бесполезен. Однако есть важный момент: закрытый ключ непременно должен быть из сгенерированной пары с открытым. Наличие открытого ключа одно из нескольких важных и интересных свойство асимметричного шифрования. То есть, мы можем передавать открытый ключ любым каналом и не бояться, что он будет применен для расшифровки сообщения.

Вместе с тем, применительно к нашей задаче есть один ньюанс — асимметричное шифрование подходит для небольших данных, например коротких сообщений. Мы можем только гадать об объеме данных, полученных разведкой. Мы, конечно, можем разбить все полученные данные на небольшие фрагменты и закодировать каждый из них закрытым ключом, но есть более оптимальный вариант решения.

**Реение**

Пользователь Akelawolf справедливо заметил, сгенерировать и отправить открытый ключ может кто угодно. Я внес некоторые коррективы в план.

Будет правильно, если до отправки агентов штаб сгенерирует несколько пар ключей и назначит каждому агенту закрытый ключ. Лучше сгенерировать именно несколько пар, чтобы у каждого агента был индивидуальный ключ. Это необходимо, чтобы точно персонифицировать владельца ключа.
Тогда в случае компрометации ключей центр создаст новый СИММЕТРИЧНЫЙ ключ, закодирует его каждому агенту открытыми ключами и отправит по открытому каналу.




Старое решение

1. Агент сгенерирует пару ключей (открытый и закрытый) на месте, затем отправит открытый ключ повстанческим силам;
2. В штабе сопротивления создадут новый ключ для СИММЕТРИЧНОГО шифрования;
3. Симметричный ключ закодируют при помощи открытого ключа, который прислал агент;
4. Зашифрованный симметричный ключ отправят агенту, который раскодирует его с помощью закрытого ключа.

Обратите внимание на нашу передачу по открытому каналу:

1. Агент отправляет ОТКРЫТЫЙ ключ из пары, ЗАКРЫТЫЙ ключ находится у него;
2. Штаб сопротивления отправляет ключ симметричного шифрования, зашифрованный присланным агентом открытым ключом.

Ни первое ни второе сообщение не представляют какой-либо ценности при перехвате.

Напишем код:

```
# Для генерации ключей мы будем использовать распространенный криптографический алгоритм RSA.
# Для хеширования данных используем также популярный алгоритм SHA256
from Crypto.PublicKey import RSA

# Напишем функцию для генерации ключей
def generate_keys(bits = 2048):

    """Generates the pair of private and public keys.

    :param bits: <int> Key length, or size (in bits)
    of the RSA modulus (default 2048)
    :return: <object> private_key, <object> public_key

    """
    private_key = RSA.generate(bits)
    public_key = private_key.publickey()
    return private_key, public_key

private_key, public_key = generate_keys(bits = 2048)
```

Давайте посмотрим как выглядят ключи:

```
print(private_key.exportKey(format='PEM').decode())
print('\n')
print('#'*65)
print('\n')
print(public_key.exportKey(format='PEM').decode())
```

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA4JDLu7Vtvg2yqbH6Y0eJPfoEsOlKzgmOodqhA1CqkEG4OpKi
sGW7ciGP4v37GE6edHBCEy4UNkVQtnpPBjzTHvKd1pO70B84vD5OSrS7uNw2EYkj
d/ZwhrJMrcQKRwPkkM4OiewaaAaK0vPWJIKwlW61DY9X7LfNz7aOKMTbKnm1vdR0
919AV98FUmNoQBgka6nXFGmNbi7D43MtLwxBZIXfFupEiANSvOs+57hgaCho7OWM
GUOjLkG6HBscPhJ2W1H5DU9GjwL24ynTvKifgo1/2ue61MV1Pzh5CVaicJKNaRtg
Pd99gFhBGINsXV2X6Jh/W5nNsCddU4EI0AlO8wIDAQABAoIBAARM4YnjrIlSK9Sy
EtBp40frjMFyhjsx1ahlzmWI2utORt/gRPtJx3AlEmNPZ8qMXt5t8+X4IOz1INmN
uAuviH90N++O/q66mlSIgOlPUrT0ipiFXseCUZ9StMMzGNtJSMw5FfAwNEU/stLd
VoF2ezkxWIg88XsX/fn3Tfub4XKLvu4raJGcJ+Fo2GI9hYEGKnHhSuHvDHekTLlQ
z46O+cIwtehbFGcKesyK3zDD1uP5YLPIWpiqt1TgKjJzRF0l4ZJLk+RT7kU2pGIQ
mosOnr+06WyMIg724yQyAIwtS9X0czKBGUESrtTTb1HCXLeTwnncOTxh6q2z42LF
tn34+DECgYEA6EEp4oTvjfTQfUQPMByuAjF1hpdFHQqRymygiFgoF+Mg3QmL0w8j
/84H/q7s8FSx+3th8MK87bFq4lrry+h/mYwmvF5zZbhxcnl2uaX+KUPgpT6TgvAo
WOv2wc4BSaoo9DrxrZId86vpO2qbopw6gkBsvw47HSoQ+FSqXtZ0p8kCgYEA94Zj
b1ulctUjybiszO93TAjkzx3lU3yL+B1eZiQXtJa3mgG+ka1R/uMfr0NlT+Jzo0My
wHV30YRJDxziCrDol9OgSSU0sXwEcUxUIBLBwXLCp1EmMsYG9PB/x4OTWve35a8F
O+rMxuvWaZeIOfVCfL8UEcWweYaVdWIonJN+ltsCgYEAjeSZ2UlMLZce9RjqioNL
EA31dlfeoqJ9dYUuAn6RaB6cSk51vWlnnfXazo9CNIYaAsFbkcL3t+QHn+jaXEZc
BowocjbmG4Q20zBAB6XRBJbynSIA7yMYE1N9+uOHx+CMisGkO12krOUfZex4zzzR
RhhkF8ly9htoKL9ZIv20YXkCgYBzH3UF6PkVZJ5lhtgP5Nx2Z7iLwBrV7ppnBrnO
BcFkw6iXH3KT7KmzQ82LxWvMcMVZzLpBGyFkOAOG3OchE9DKNKpa+sv8NHMYguip
li+5mneAPFTozoOTznuPvtl9OLO2RuXHTVh6uFub9tdsJW8L+A8MiQagLwE6fDHp
SQxaewKBgQDIyzL1THpW3+AMNrOZuI/d3Em5wpGJiZbDSBRosvsfGm/sHaz4Ik5E
nWnftgktmsAD60eORTTh9/ww/nm7f3q9kzT8Sv1MmqeRXq9VFIOeP/+8SSE/7LzD
izlb5xEtVD8LuY54jHyiOxiZC++TQswMnOKKi0Gx26MDoO7Tx9akVw==
-----END RSA PRIVATE KEY-----
```

```
######################################################################

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4JDLu7Vtvg2yqbH6Y0eJ
PfoEsOlKzgmOodqhA1CqkEG4OpKisGW7ciGP4v37GE6edHBCEy4UNkVQtnpPBjzT
HvKd1pO70B84vD5OSrS7uNw2EYkjd/ZwhrJMrcQKRwPkkM4OiewaaAaK0vPWJIKw
lW61DY9X7LfNz7aOKMTbKnm1vdR0919AV98FUmNoQBgka6nXFGmNbi7D43MtLwxB
ZIXfFupEiANSvOs+57hgaCho7OWMGUOjLkG6HBscPhJ2W1H5DU9GjwL24ynTvKif
go1/2ue61MV1Pzh5CVaicJKNaRtgPd99gFhBGINsXV2X6Jh/W5nNsCddU4EI0AlO
8wIDAQAB
-----END PUBLIC KEY-----
```

Как видите, ключи асимметричного шифрования представляют из себя длинные математически сгенерированные последовательности символов.

Итак, мы сгенерировали ключи. Теперь давайте напишем функцию для кодирования данных:

```python
from Crypto.PublicKey import RSA # для асимметричного шифрования будем использовать распростран
енный алгоритм RSA.
from Crypto.Hash import SHA256 # Для хеширования данных используем также популярный алгоритм SH
A256. Хеширование в данном
#случае нам нужно для проверки правильности раскодирования
from Crypto.Cipher import PKCS1_OAEP # протокол шифрования

def encrypt_message(message, public_key, verbose = True):
    """Encripts the message using public_key.

    :param message: <str> Message for encryption
    :param public_key: <object> public_key
    :param verbose: <bool> Print description;
    :return: <object> Message encrypted with public_key

    """
    message_hash = SHA256.new(message.encode()) # находим хеш сообщения.
    cipher = PKCS1_OAEP.new(public_key)
    message_with_hash = message.encode() + message_hash.hexdigest().encode() # мы добавляем хеш
сообщения в сообщение, чтобы при раскодировке сравнить его с хешем раскодированного сообщения
    encrypted_message = cipher.encrypt(message_with_hash)
    if verbose:
        print(f'Message: {message} was encrypted to\n{encrypted_message.hex()}')
    return encrypted_message
```

```python
def decrypt_message(encrypted_message, private_key):
    """Decripts the message using private_key and check it's hash

    :param encrypted_message: <object> Encrypted message
    :param private_key: <object> private_key
    :return: <object> Message decripted with private_key

    """
    dsize = SHA256.digest_size*2
    cipher = PKCS1_OAEP.new(private_key)
    decrypted_message_with_hesh = cipher.decrypt(encrypted_message) # все сообщение (вместе с хешем)
    decrypted_message = decrypted_message_with_hesh[:-dsize] # текстовая часть сообщения без хеша
    digest = SHA256.new(decrypted_message).hexdigest() # хеш расшифрованной части сообщения
    if digest==decrypted_message_with_hesh[-dsize:].decode(): # Если хеш расшифрованного сообщения и хеш, который мы добавили при шифровании равны, расшифровка правильная
        print(f"Success!\nEncrypted hash is {decrypted_message_with_hesh[-dsize:].decode()}\nDecrypted hash is {digest}")
        return decrypted_message.decode()
    else:
        print(f"Encryption was not correct: the hash of decripted message doesn't match with encrypted hash\nEncrypted hash is {decrypted_message_with_hesh[-dsize:]}\nDecrypted hash is {digest}")
```

**Схема работы по шагам**

1. Агент генерирует пару ключей:

```python
private_key, public_key = generate_keys()
```

2. Отправляет в штаб ОТКРЫТЫЙ ключ;
3. В штабе с помощью открытого ключа кодируют ключ для симметричного шифрования:

```python
new_symmetric_key = 'SOME_KEY_asdfasdfasdfasdfsdfgrtwhetynt'
encr_msg = encrypt_message(new_symmetric_key, public_key)
```

Вывод Message: SOMEKEYasdfasdfasdfasdfsdfgrtwhetynt was encrypted to 41e940507c96397e3feb4a533

4. Эту длинную последовательность отправляют обратно агенту;
5. Агент дешифрует полученное сообщение при помощи закрытого ключа:

```
recieved_symmetric_key = decrypt_message(encr_msg, private_key)
print('\n')
print(f"New symmetric key is: {recieved_symmetric_key}")
```

Вывод

```
Success!
Encrypted hash is 42ad66445a05ac09e684bb21f9b487d95b9cfa11d02e0b459931321ee02f7c1c
Decrypted hash is 42ad66445a05ac09e684bb21f9b487d95b9cfa11d02e0b459931321ee02f7c1c

New symmetric key is: SOME_KEY_asdfasdfasdfasdfsdfgrtwhetynt
```

6. Затем с помощью нового симметричного ключа агент шифрует полученные данные:

```
message = """
Длина 120 км
Ширина 120 км
Высота/глубина 120 км
МгС 10 МгС
Двигатель Субсветовой двигатель 30-5 (2)
Класс гиперпривода Класс 4.0

"""
encr_message = symmetric_encrypt(message, recieved_symmetric_key, verbose = True)
```

Вывод

```
Message was encrypted into: 665968950814e0edf48bd80f368cda63ee0ac2b06173989b4bc239c02a
064715e433d3e07f638d8e277e9b4bcd0d32c8c819483501066549a64c7ba8a29fa16a289c0283665cfbc2
d1ff8ea8c4c4691b76ca430d9d1a26bfe38eb9d3c6ff398ea8926f44af13cf74a30f870f56bab311237433
1b3a8bdfd5b1ca465604738e83b86481ca8852193c6c27db25690577d3e28c930d357abba6a3bf4c1f77f6
9dd066b933b60b353a7c44439baaabe9a58593efab6b410ee7ff4bbcc72dd07456a4f8c9a68c64e99a9151
bc466f9daa2cdd3b4abcef7f87bc58caaef52a7811ce05cda5e03e9cba4b1537f7e6096b2385491a1c49a4
51373868f9951581643047d8f0c72e789943cc7112e00d06a4848919ee823bb18338e2b157c39b8911e518
fdccdc8847a3e24be4cedad6
```

7. В штабе производят дешифровку:

```
print('DECRIPTION')
decr_message = symmetric_decrypt(encr_message, new_symmetric_key)
print(decr_message)
```

Вывод

```
DECRIPTION
Success!
Encrypted hash is b0dbb35b28fbff258350a50c39282b73e31f408c9da937c81d8d48115b491026
Decrypted hash is b0dbb35b28fbff258350a50c39282b73e31f408c9da937c81d8d48115b491026

Длина 120 км
Ширина 120 км
Высота/глубина 120 км
МгС 10 МгС
Двигатель Субсветовой двигатель 30-5 (2)
Класс гиперпривода Класс 4.0
```

**Вуаля**

На данном абстрактном примере мы увидели работу распространенных алгоритмов шифрования. Симметричное и асимметричное шифрование, а также хеширование применяются в работе веба, электронной подписи, блокчейне и криптовалютах. Надеюсь, материал был полезен для понимания работы этих технологий.

**Послесловие**

В итоге, разведке повстанцев удалось добыть точные сведения об уязвимости станции и пути до нее, присутствии Императора для осмотра, наличии энергощита и его источника на Эндоре. Империя вычислила шпионов, дезинформировала их о боеспособности станции. Станция также была отведена к спутнику Эндора, откуда была защищена щитом.