(https://python-scripts.com/encryption-cryptography)

Обработка Данных (Https://Python-Scripts.Com/Category/Work-With-Data-Python)

Шифрование и криптография в Python

автор Monty Python (https://python-scripts.com/author/monty-python)

Май 23, 2017

В Python не так уж много инструментов стандартной библиотеки, которые работают с шифрованием. Однако, в нашем распоряжении есть библиотеки хешинга. Давайте рассмотрим этот вопрос в данной статье, но более детально сфокусируемся на двух сторонних пакетах: PyCrypto (https://python-scripts.com/encryption-cryptography) и cryptography (https://python-scripts.com/encryption-cryptography). Мы научимся шифровать и расшифровывать строки при помощи двух этих библиотек.

Хеширование

Если вам нужно защитить хэши или алгоритм дайджеста сообщений, то для этого прекрасно подойдет модуль стандартной библиотеки Python hashlib. Он включает в себя безопасные алгоритмы хеширования FIPS, такие как SHA1, SHA224, SHA256, SHA384, а также SHA512 и MD5. Python также поддерживает функции хеширования adler32 и crc32, но они содержатся в модуле zlib. Одно из самых популярны применений хеширования это хранение хеша пароля, вместо самого пароля. Конечно, хеш должен быть хорошим, в противном случае он может быть расшифрован.

Другой популярный случай, в котором применяется хеширование – это хеширование файла, с последующей отправкой файла и его хеша по отдельности. Получатель файла может запустить хеш в файле, чтобы убедиться в том, что файл соответствует отправленному хешу. Если это так, значит никто не менял файл, когда он был отправлен. Давайте попробуем создать хеш md5. Но оказывается, чтобы использовать хеш md5, нужно передать его строке байта, вместо обычной. Так что мы попробовали сделать это, после чего вызвали метод дайджеста, чтобы получить наш хеш. Если вы претпочитаете хешированный дайджест, мы можем сделать и это:

Python

```python
import hashlib
hashlib.md5.update(b'Python rocks!')
result = hashlib.md5.digest()

print(result)
# b'\x14\x82\xec\x1b#d\xf6N}\x16*+[\x16\xf4w'
```

Давайте уделим время на то, чтобы разобраться с увиденным. Сначала мы импортировали модуль hashlib (https://python-scripts.com/encryption-cryptography) и создали экземпляр объекта md5 HASH. Далее, мы вписали небольшой текст в объект хеша и получили трассировку.

Python

```python
print( md5.hexdigest() )
# '1482ec1b2364f64e7d162a2b5b16f477'
```

На самом деле существует метод быстрого создания хеша, мы рассмотрим его, когда создадим наш хеш sha512:

Python

```
import hashlib
sha = hashlib.sha1(b'Hello Python').hexdigest()

print(sha) # '422fbfbc67fe17c86642c5eaaa48f8b670cbed1b'
```

Как вы видите, мы создали наш экземпляр хеша и вызвали его метод дайджеста одновременно. Далее, мы выводим наш хеш, чтобы на него посмотреть. Лично я использую хеш sha1, так как его хеш достаточно короткий и отлично ложится в страницу. Но в то же время он и не очень безопасный, так что вы вольны выбирать то, что вам удобно.

Вывод ключа

У Python весьма ограниченная поддержка вывода ключа, встроенная в стандартную библиотеку. Фактически, единственный метод, предлагаемый hashlib это pbkdf2_hmac, который является основанной на пароле функцией вывода ключа PKCS#5. Он использует HMAC в качестве своей псевдослучайной функцией. Вы можете использовать что-нибудь на подобии для хеширования вашего пароля, так как он поддерживает соль и итерации. Например, если вы собираетесь использовать SHA-256, вам может понадобиться соль минимум в 16 битов и 100.000 итераций. Являясь быстрой отсылкой, соль — это просто случайные данные, которые вы используете в качестве дополнения в вашем хеше, с целью усложнения расшифровки вашего пароля. В целом, она защищает ваш пароль от словарных атак и рассчитанных заранее радужных таблиц. Давайте посмотрим на пример:

Python

```
import binascii

dk = hashlib.pbkdf2_hmac(hash_name='sha256',
    password=b'bad_password34',
    salt=b'bad_salt',
    iterations=100000)

result = binascii.hexlify(dk)

print(result)
# b'6e97bad21f6200f9087036a71e7ca9fa01a59e1d697f7e0284cd7f9b897d7c02'
```

Здесь мы создаем хеш SHA256 в пароле при помощи такой-себе соли со 100,000 итераций. Конечно, SHA в буквальном смысле не рекомендуется для создания ключей паролей. Вместо этого, вам лучше использовать что-то вроде scrypt. Еще одним полезным инструментом может быть сторонний пакет bcrypt. Он разработан специально для хеширования паролей.

PyCrypto

Пакет PyCrypto (https://python-scripts.com/encryption-cryptography), наверное, самый известный сторонний пакет криптографии для Python. К сожалению, его доработка остановилась в 2012 году. Однако продолжается выпускаться под разные версии Python, вы можете получить PyCrypto для версии 3.5 включительно, если вы не брезгуете использовать двоичный код стороннего производства. К примеру, я нашел несколько бинарных колес Python 3.5 для PyCrypto на Github (https://github.com/sfbahr/PyCrypto-Wheels (https://github.com/sfbahr/PyCrypto-Wheels)). К счастью, есть развилка проекта под названием PyCrytodome, которая является неплохой заменой PyCrypto. Для его установки на Linux вы можете использовать следующую команду:

```python
pip install pycryptodome
```

Для Windows немного отличается.

```python
pip install pycryptodomex
```

Если вы столкнетесь со сложностями, это, возможно, связанно с тем, что у вас нет необходимых установленных зависимостей, или необходим компилятор под Windows. Вы можете перейти на официальный сайт PyCryptodome для дополнительной информации о установке, или чтобы связаться с поддержкой. Также стоит отметить, что PyCryptodome имеет ряд преимуществ в сравнении с последней версией PyCrypto. Рекомендую потратить немного времени и посетить их сайт (https://pycryptodome.readthedocs.io/en/latest/), для ознакомления с возможностями PyCryptodome.

Шифрование Строки

Теперь (после того, как вы ознакомились с информацией на сайте, я надеюсь), мы можем перейти к дальнейшим примерам. Для нашего следующего кода мы используем DES для шифровки строки:

```python
```

```python
from Crypto.Cipher import DES

key = b'abcdefgh'

def pad(text):
    while len(text) % 8 != 0:
        text += b' '
    return text

des = DES.new(key, DES.MODE_ECB)
text = b'Python rocks!'
padded_text = pad(text)

encrypted_text = des.encrypt(padded_text)
print(encrypted_text)
# b'>\xfc\x1f\x16x\x87\xb2\x93\x0e\xfcH\x02\xd59VQ'
```

Этот код слегка запутанный, так что давайте уделим немного времени на его анализ. Во первых, обратите внимание на то, что размер ключа под шифровку DES — 8 байт, по этому мы установили нашу переменную ключа в строку размер букв строки. Шифруемая нами строка должна быть кратна 8 в ширину, так что мы создаем функцию (https://python-scripts.com/functions-python) под названием pad, которая может заполнить любую строку пробелами, пока она не станет кратна 8. Далее мы создаем экземпляр DES и текст, который нам нужно зашифровать. Мы также создаем заполненную версию текста. Прикола ради, мы попытаемся зашифровать начальный, незаполненный вариант строки, что приведет нас к ошибке ValueError (https://python-scripts.com/try-except-finally). Таким образом Python ясно дает нам понять, что нам нужно использовать заполненную строку, так что мы передаем вторую версию. Как вы видите, мы получаем зашифрованную строку! Конечно, пример нельзя назвать полным, если мы не выясним, как расшифровать нашу строку:

Python

```python
data = des.decrypt(encrypted_text)
print(data) # Python rocks!
```

К счастью, это очень легко сделать, так как все что нам нужно, это вызвать метод decrypt в нашем объекте des для получения расшифрованной байтовой строки. Наша следующая задача — научиться шифровать файлы и расшифровывать файлы с PyCrypto при помощи RSA. Но для начала, нам нужно создать ключи RSA.

Мы собрали ТОП Книг для Python программиста которые помогут быстро изучить язык программирования Python. Список книг: Книги по Python (https://python-scripts.com/books)

Создание ключей RSA

Если вам нужно зашифровать ваши данные при помощи RSA, тогда вам также нужно получить доступ к паре ключа RSA public / private, или сгенерировать собственную. В данном примере мы генерируем собственную пару ключей. Так как это весьма легко, мы сделаем это в интерпретаторе Python:

Python

```python
from Crypto.PublicKey import RSA

code = 'nooneknows'
key = RSA.generate(2048)

encrypted_key = key.exportKey(
    passphrase=code,
    pkcs=8,
    protection="scryptAndAES128-CBC"
)

with open('my_private_rsa_key.bin', 'wb') as f:
    f.write(encrypted_key)

with open('my_rsa_public.pem', 'wb') as f:
    f.write(key.publickey().exportKey())
```

Сначала мы импортируем RSA из Crypto.PublicKey. Затем, мы создаем примитивный код доступа. Далее, мы генерируем ключ RSA на 2048 битов. Теперь мы подходим к интересной части. Для генерации приватного ключа, нам нужно вызвать метод exportKey нашего ключа RSA, и передать ему наш код доступа, который будет использован стандартом PKCS, чья схема шифровки будет использована для защиты нашего приватного ключа. После этого мы записываем файл на диск. Далее, мы создаем наш приватный ключ через метод publickey нашего ключа RSA. Мы использовали короткий путь в этой части кода, связав вызов метода exportKey с методом publickey для записи файла на диск.

Шифровка файла

Теперь у нас в распоряжении есть и приватный и публичный ключи, так что мы можем зашифровать кое-какие данные и вписать их в файл. Вот достаточно простой пример:

```python
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP


with open('encrypted_data.bin', 'wb') as out_file:
    recipient_key = RSA.import_key(
        open('my_rsa_public.pem').read()
    )

    session_key = get_random_bytes(16)

    cipher_rsa = PKCS1_OAEP.new(recipient_key)
    out_file.write(cipher_rsa.encrypt(session_key))

    cipher_aes = AES.new(session_key, AES.MODE_EAX)
    data = b'blah blah blah Python blah blah'
    ciphertext, tag = cipher_aes.encrypt_and_digest(data)

    out_file.write(cipher_aes.nonce)
    out_file.write(tag)
    out_file.write(ciphertext)
```

Первые три строки покрывают наши импорты из PyCryptodome. Далее мы открываем файл для записи (https://python-scripts.com/work-with-files-python). Далее, мы импортируем наш публичный ключ в переменной и создаем 16-битный ключ сессии. Для этого примера мы будем использовать гибридный метод шифрования, так что мы используем PKCS#1 OAEP (*Optimal asymmetric encryption padding*). Это позволяет нам записывать данные произвольной длинны в файл. Далее, мы создаем наш шифр AES, создаем кое-какие данные и шифруем их. Это дает нам зашифрованный текст и MAC. Наконец, мы выписываем nonce, MAC (или тег), а также зашифрованный текст. К слову, nonce - это произвольное число, которое используется только в

криптографических связях. Обычно это случайные или псевдослучайные числа. Для AES, оно должно быть минимум 16 байтов в ширину. Вы вольны попытаться открыть зашифрованный файл в своем текстовом редакторе. Вы увидите только какое-то безобразие. Теперь попробуем расшифровать наши данные:

```python
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_OAEP


code = 'nooneknows'

with open('encrypted_data.bin', 'rb') as fobj:
    private_key = RSA.import_key(
        open('my_rsa_key.pem').read(),
        passphrase=code
    )

    enc_session_key, nonce, tag, ciphertext = [
        fobj.read(x) for x in (private_key.size_in_bytes(), 16, 16, -1)
    ]

    cipher_rsa = PKCS1_OAEP.new(private_key)
    session_key = cipher_rsa.decrypt(enc_session_key)

    cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
    data = cipher_aes.decrypt_and_verify(ciphertext, tag)

print(data)
```

Если вы разобрались с предыдущим примером, то этот код должен быть весьма простым для разбора. В данном случае, мы открываем наш зашифрованный файл для чтения в бинарном режиме. Далее, мы импортируем наш приватный ключ. Обратите внимание на то, что когда вы импортируете приватный ключ, вы должны передать ему код доступа. В противном случае возникнет ошибка. Далее мы считываем наш файл. Вы заметите, что сначала мы считываем приватный ключ, затем 16 байтов для nonce, за которыми следуют 16 байтов, которые являются тегом, и наконец, остальную часть файла, который и является нашими данными. Далее нам нужно расшифровать наш ключ сессии, пересоздать наш ключ AES и расшифровать данные. Вы можете использовать PyCryptodome в намного более широком ряде случаев. Однако, нам нужно идти дальше и посмотреть, что еще мы можем сделать для наших криптографических нужд в Python.

Мы собрали ТОП Книг для Python программиста которые помогут быстро изучить язык программирования Python. Список книг: Книги по Python (https://python-scripts.com/books)

Пакет cryptography

Пакет cryptography нацелен на то, чтобы быть «*криптографом для людей*», равно как и библиотека requests является «*HTTP для людей*». Суть в том, что вам нужно разработать простые криптографические рецепты которые и безопасны, и простые в использовании. Если нужно, вы можете перейти к низкоуровневым криптографическим примитивам, для которых требуется лишь знать, что вы делаете, в противном случае вы создадите что-то явно бесполезное в контексте защиты. Если вы работаете в Python 3.5 Windows (https://python-scripts.com/install-python-windows), вы можете установить этот пакет при помощи pip (https://python-scripts.com/how-to-install-modules-python) следующим образом:

Python

```
pip install cryptography
```

Вы увидите, что cryptography установится совместно с несколькими зависимостями. Предположим, что с установкой все прошло чисто, и мы можем зашифровать какой-нибудь текст. Давайте используем для этого модуль Fernet.

Модуль Fernet реализует простую в использовании схему аутентификации, которая использует симметричный алгоритм шифрования, который гарантирует, что каждое зашифрованное в нем сообщение не может быть использовано или прочитано без определенного вами ключа. Модуль Fernet также поддерживает ключ ротации через MultiFernet. Давайте взглянем на простой пример:

Python

```
from cryptography.fernet import Fernet

cipher_key = Fernet.generate_key()
print(cipher_key) # APM1JDVgT8WDGOWBgQv6EIhvxl4vDYvUnVdg-Vjdt0o=
```

Python

```
from cryptography.fernet import Fernet

cipher = Fernet(cipher_key)
text = b'My super secret message'
encrypted_text = cipher.encrypt(text)

print(encrypted_text)

# (b'gAAAAABXOnV86aeUGADA6mTe9xEL92y_m0_TlC9vcqaF6NzHqRKkjEqh4d21PInEP3
# b'6bdHsSlRiCNWbSkPuRd_62zfEv3eaZjJvLAm3omnya8=')
```

Python

```
decrypted_text = cipher.decrypt(encrypted_text)
print(decrypted_text) # 'My super secret message'
```

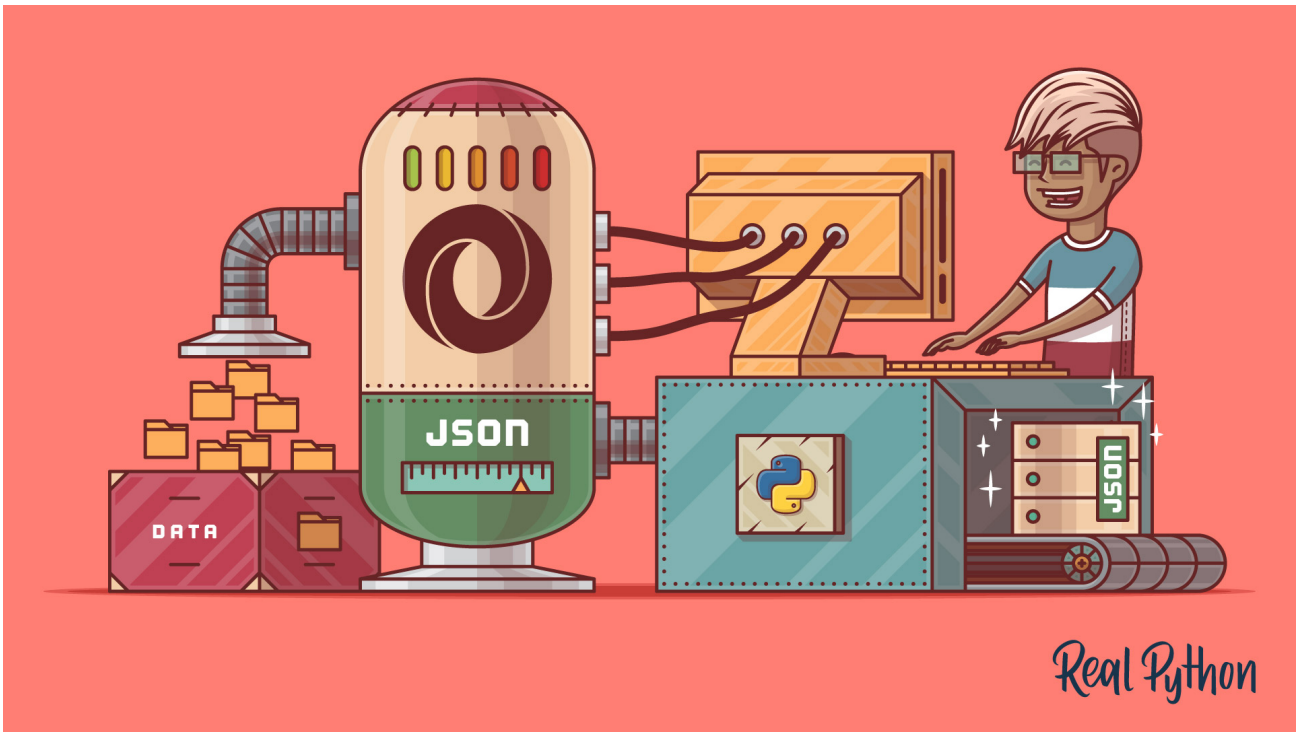
Для начала, нам нужно импортировать Fernet. Затем мы генерируем ключ. Мы выводим ключ, чтобы увидеть, как он выглядит. Как вы видите, это случайна строка байтов. Если хотите, вы можете попробовать запустить метод generate_key несколько раз. Результат каждый раз новый. Далее мы создаем экземпляр нашего шифра Fernet при помощи нашего ключа. Теперь у нас есть шифр, который мы можем использовать для шифрования и расшифровки нашего сообщения. Следующий шаг, это создание сообщения, достойного шифровки, с последующей его шифровкой при помощи метода encrypt. Я пошел вперед и вывел наш зашифрованный текст так, чтобы вы увидели что вы больше не можете его читать. Для расшифровки нашего супер-засекреченного сообщения, мы просто вызовем метод decrypt в нашем шифре и передадим зашифрованный текст. В результате мы получим текстовую байтовую строку нашего сообщения.

Подведем Итоги

В данной статье мы изрядно прошлись по поверхности вопроса : «*Как, и что делать с пакетами PyCryptodome и cryptography?*». Мы рассмотрели изрядное количество вариантов применения данных пакетов в шифровании и расшифровке строк и файлов. Убедитесь в том, что уделите время документации (https://pycryptodome.readthedocs.io/en/latest/), перед тем как начать экспериментировать с изложенной в данной статье информацией.

← Prev (https://python-scripts.com/contextlib) Next → (https://python-scripts.com/importlib)

Вам Может Быть Интересно



(https://python-scripts.com/json)

JSON в Python (https://python-scripts.com/json)

Лучшие примеры форматирования строк в Python (https://python-scripts.com/string-formatting)

Создаем Blockchain с нуля на Python (https://python-scripts.com/blockchain)

Генерация случайных данных в Python (Руководство) (https://python-scripts.com/random-data)

6 правил для разработки эффективного парсера в Python (https://python-scripts.com/requests-rules)

ELForcer • 10 месяцев назад

Пример с шифровкой файла не работает.

```
with open('encrypted_data.bin', 'wb') as out_file:
recipient_key = RSA.import_key(
open('my_rsa_public.pem').read()
)
```

AttributeError: module 'Crypto.PublicKey.RSA' has no attribute 'import_key'

С Расшифровкой тоже самое.
У вас для какой версии Питона примеры? Второй или Третий?

ELForcer • год назад

Спасибо. Поизучаем.

Обучающий Контент

Бесплатный обучающий контент для Python программистов.

Сообщество Python (https://python-scripts.com) Программистов