denis_kiber 24 августа 2015 в 10:52

Криптография на Python: шифрование информации и создание электронных цифровых подписей с помощью пакета PyCrypto

Python, Информационная безопасность, Криптография, Программирование

Из песочницы Tutorial

Долго мучился с PyCrypto, в итоге получилась эта статья и полная реализация следующего протокола:

Этап отправки:

1. Алиса подписывает сообщение своей цифровой подписью и шифрует ее открытым ключом Боба (асимметричным алгоритмом).
2. Алиса генерирует случайный сеансовый ключ и шифрует этим ключом сообщение (с помощью симметричного алгоритма).
3. Сеансовый ключ шифруется открытым ключом Боба (асимметричным алгоритмом).
Алиса посылает Бобу зашифрованное сообщение, подпись и зашифрованный сеансовый ключ.

Этап приёма:

Боб получает зашифрованное сообщение Алисы, подпись и зашифрованный сеансовый ключ.
4. Боб расшифровывает сеансовый ключ своим закрытым ключом.
5. При помощи полученного, таким образом, сеансового ключа Боб расшифровывает зашифрованное сообщение Алисы.
6. Боб расшифровывает и проверяет подпись Алисы.

Вышеописанный протокол является гибридной системой шифрования, которая работают следующим образом. Для симметричного алгорит AES (или любого другого) генерируется случайный сеансовый ключ.

Такой ключ как правило имеет размер от 128 до 512 бит (в зависимости от алгоритма). Затем используется симметричный алгоритм для шифрования сообщения. В случае блочного шифрования необходимо использовать режим шифрования (например CBC), что позволит шифровать сообщение с длиной, превышающей длину блока. Что касается самого случайного ключа, он должен быть зашифрован с помо открытого ключа получателя сообщения, и именно на этом этапе применяется криптосистема с открытым ключом RSA.

Поскольку сеансовый ключ короткий, его шифрование занимает немного времени. Шифрование набора сообщений с помощью асимметри алгоритма — это задача вычислительно более сложная, поэтому здесь предпочтительнее использовать симметричное шифрование. Затем

достаточно отправить сообщение, зашифрованное симметричным алгоритмом, а также соответствующий ключ в зашифрованном виде. Получатель сначала расшифровывает ключ с помощью своего секретного ключа, а затем с помощью полученного ключа получает и всё сообщение.

Начнем с генерации пары ключей для Алисы и Боба.

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

# key generation Alisa
privatekey = RSA.generate(2048)
f = open('c:\cipher\\alisaprivatekey.txt','wb')
f.write(bytes(privatekey.exportKey('PEM'))); f.close()
publickey = privatekey.publickey()
f = open('c:\cipher\\alisapublickey.txt','wb')
f.write(bytes(publickey.exportKey('PEM'))); f.close()
# key generation Bob
privatekey = RSA.generate(2048)
f = open('c:\cipher\\bobprivatekey.txt','wb')
f.write(bytes(privatekey.exportKey('PEM'))); f.close()
publickey = privatekey.publickey()
f = open('c:\cipher\\bobpublickey.txt','wb')
f.write(bytes(publickey.exportKey('PEM'))); f.close()
```

На данный момент система шифрования на основе RSA считается надёжной, начиная с размера ключа в 2048 бит.

В системе RSA можно создавать сообщения, которые будут и зашифрованы, и содержать цифровую подпись. Для этого автор сначала дол добавить к сообщению свою цифровую подпись, а затем — зашифровать получившуюся в результате пару (состоящую из самого сообщен подписи к нему) с помощью открытого ключа, принадлежащего получателю. Получатель расшифровывает полученное сообщение с помощ своего секретного ключа и проверяет подпись автора с помощью его открытого ключа.

Использование односторонней криптографической хеш-функции позволяет оптимизировать вышеописанный алгоритм цифровой подписи. Производится шифрование не самого сообщения, а значения хеш-функции, взятой от сообщения. Данный метод обеспечивает следующие преимущества:

1. Понижение вычислительной сложности. Как правило, документ значительно больше его хеша.
2. Повышение криптостойкости. Криптоаналитик не может, используя открытый ключ, подобрать подпись под сообщение, а только под его х
3. Обеспечение совместимости. Большинство алгоритмов оперирует со строками бит данных, но некоторые используют другие представле Хеш-функцию можно использовать для преобразования произвольного входного текста в подходящий формат.

Хэш-функции, представляют собой функции, математические или иные, которые получают на вход строку переменной длинны (называему прообразом) и преобразуют ее в строку фиксированной, обычно меньшей, длинны (называемую значением хэш-функции). Смысл хэш-фун состоит в получении характерного признака прообраза — значения, по которому анализируются различные прообразы при решении обрат задачи. Однонаправленная хэш-функция — это хэш-функция, которая работает только в одном направлении: легко вычислить значение хэ функции по прообразу, но трудно создать прообраз, значение хэш-функции которого равно заданной величине. Хэш-функция является откр тайны ее расчета не существует. Безопасность однонаправленной хэш-функции заключается именно в ее однонаправленности.

Переходим к написанию первого пункта протокола:

1. Алиса подписывает сообщение своей цифровой подписью и шифрует ее открытым ключом Боба (асимметричным алгоритмом RSA).

```
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

# creation of signature
f = open('c:\cipher\plaintext.txt','rb')
plaintext = f.read(); f.close()
privatekey = RSA.importKey(open('c:\cipher\\alisaprivatekey.txt','rb').read())
myhash = SHA.new(plaintext)
signature = PKCS1_v1_5.new(privatekey)
signature = signature.sign(myhash)
# signature encrypt
publickey = RSA.importKey(open('c:\cipher\\bobpublickey.txt','rb').read())
```

```
cipherrsa = PKCS1_OAEP.new(publickey)
sig = cipherrsa.encrypt(signature[:128])
sig = sig + cipherrsa.encrypt(signature[128:])
f = open('c:\cipher\signature.txt','wb')
f.write(bytes(sig)); f.close()
```

В следующем листинге код для следующих двух пунктов протокола:

2. Алиса генерирует случайный сеансовый ключ и шифрует этим ключом сообщение (с помощью симметричного алгоритма AES).
3. Сеансовый ключ шифруется открытым ключом Боба (асимметричным алгоритмом RSA).

```
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

# creation 256 bit session key
sessionkey = Random.new().read(32) # 256 bit
# encryption AES of the message
f = open('c:\cipher\plaintext.txt','rb')
plaintext = f.read(); f.close()
iv = Random.new().read(16) # 128 bit
obj = AES.new(sessionkey, AES.MODE_CFB, iv)
ciphertext = iv + obj.encrypt(plaintext)
f = open('c:\cipher\plaintext.txt','wb')
f.write(bytes(ciphertext)); f.close()
# encryption RSA of the session key
publickey = RSA.importKey(open('c:\cipher\\bobpublickey.txt','rb').read())
cipherrsa = PKCS1_OAEP.new(publickey)
sessionkey = cipherrsa.encrypt(sessionkey)
f = open('c:\cipher\sessionkey.txt','wb')
f.write(bytes(sessionkey)); f.close()
```

Для режима шифрования CFB требуется вектор инициализации (IV) (переменная iv).

В криптографии, вектор инициализации (IV) представляет собой некоторое число, как правило оно должно быть случайным или псевдослучайным. Случайность имеет решающее значение для достижения семантической безопасности, которая при повторном использо схемы под тем же ключом не позволит злоумышленнику вывести отношения между сегментами зашифрованных сообщений. Вектор инициализации не шифруется и сохраняется перед зашифрованным сообщением.

Далее Алиса посылает Бобу зашифрованное сообщение, подпись и зашифрованный сеансовый ключ.

Боб получает зашифрованное сообщение Алисы, подпись и зашифрованный сеансовый ключ.

4. Боб расшифровывает сеансовый ключ своим закрытым ключом.
5. При помощи полученного, таким образом, сеансового ключа Боб расшифровывает зашифрованное сообщение Алисы.

```
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

# decryption session key
privatekey = RSA.importKey(open('c:\cipher\\bobprivatekey.txt','rb').read())
cipherrsa = PKCS1_OAEP.new(privatekey)
f = open('c:\cipher\sessionkey.txt','rb')
sessionkey = f.read(); f.close()
sessionkey = cipherrsa.decrypt(sessionkey)
# decryption message
f = open('c:\cipher\plaintext.txt','rb')
ciphertext = f.read(); f.close()
iv = ciphertext[:16]
obj = AES.new(sessionkey, AES.MODE_CFB, iv)
plaintext = obj.decrypt(ciphertext)
plaintext = plaintext[16:]
```

```
f = open('c:\cipher\plaintext.txt','wb')
f.write(bytes(plaintext)); f.close()
```

Последний этап:

6. Боб расшифровывает и проверяет подпись Алисы.

```
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

# decryption signature
f = open('c:\cipher\signature.txt','rb')
signature = f.read(); f.close()
privatekey = RSA.importKey(open('c:\cipher\\bobprivatekey.txt','rb').read())
cipherrsa = PKCS1_OAEP.new(privatekey)
sig = cipherrsa.decrypt(signature[:256])
sig = sig + cipherrsa.decrypt(signature[256:])
# signature verification
f = open('c:\cipher\plaintext.txt','rb')
plaintext = f.read(); f.close()
publickey = RSA.importKey(open('c:\cipher\\alisapublickey.txt','rb').read())
myhash = SHA.new(plaintext)
signature = PKCS1_v1_5.new(publickey)
test = signature.verify(myhash, sig)
```

Безопасность RSA основана на сложности задачи факторизации произведения двух больших простых чисел. Факторизация целых чисел дл
больших чисел является задачей большой сложности. Не существует никакого известного способа, чтобы решить эту задачу быстро.
Факторизация кандидат в односторонние функции, которые относительно легко вычисляются, но инвертируются с большим трудом. То есть
x просто рассчитать f(x), но по известному f(x) нелегко вычислить x. Здесь, «нелегко» означает, что для вычисления x по f(x) могут потребов
миллионы лет, даже если над этой проблемой будут биться все компьютеры мира.

Литература:
Брюс Шнайер — Прикладная криптография

Теги: Криптография, PyCrypto, Электронная подпись, Шифрование, AES, RSA

15 августа 2013 в 10:18

Безопасность электронной почты: шифрование писем

+26 93,8k 307 51

Комментарии 6

lair 24 августа 2015 в 13:23

Скажите, пожалуйста, какую задачу вы пытались решить и зачем?

stargrave2 24 августа 2015 в 14:20

Мелочь: вы подписываете .hexdigest(), а стоило бы просто .digest() — hexdigest это в два раза больше информации не несущей полезной нагрузки. К того размер подписи станет кратным 16-байтам.

Серьёзное: вы делаете подпись чистым RSA. Шнайер в своих книгах неоднократно подчёркивает что ни в коем случае никогда простой RSA нельзя использовать. Обязательно использование padding. PyCrypto предоставляет PKCS1_PSS и PKCS1_v1.5 RSA padding функции которые и надо использовать. И вам не понадобится думать о кратности какому-то размеру (кода станет меньше).

Если использовать CFB режим (в PyCrypto он есть) симметричного шифрования, то plaintext может быть в итоге любой длины, не требовать дополне (опять же код станет проще).

Длины сообщений вы записываете в виде ASCII десятичных символов. Ведь в Python есть удобный ord(), chr() подходящих для небольших сообщен чтобы сохранить в виде одного байта длину. Код станет проще. Либо использовать struct.pack/unpack для хранения чисел большой длины.

Почему бы не зашифровать и подпись и plaintext один раз симметричным ключом сразу и только этот симметричный ключ зашифровать публичным? Здесь вы делаете дорогие асимметричные операции с публичным ключом и для шифрования подписи и для сессионного ключа. Подпись без дешифрованных данных бесполезна. AES(Key, plaintext + RSASignature) + RSAPubEnc(Public, Key), вместо AES(plaintext) + RSAPubEnc(Public, RSASignature) + RSAPubEnc(Public, KEy).

Вообще криптостойкость не повышается с использованием хэш-функций вместо полных документов. В хэш-функциях возможны коллизии и из-за «парадокса дней рождения» вероятность очень высока: 128/2 = 2**64 если бы MD5 был коллизий-устойчивым. Но из-за ресурсоёмкости и невозмож подписать длинные сообщения — без хэша никуда. Но именно из-за коллизий сейчас вовсю стремятся использовать 512-1024 бит хэш знаения.

stargrave2 24 августа 2015 в 14:27

Кстати, раз сессионный ключ у вас при отправке каждого сообщения разный, то получается что ключ используется только для одного сообщения. Вполне безопасно в данном случае будет иметь нулевой вектор инициализации (одни нулевые байты) и не добавлять IV в отправляемые сообще что уменьшит их размеры и никак не повлияет на безопасность.

denis_kiber 27 августа 2015 в 00:57

без вектора IV не хочет записывать

denis_kiber 27 августа 2015 в 01:00

спасибо. исправил.

mird 24 августа 2015 в 14:47

Мне не очень понятно, какой смысл сейчас использовать md5 hash? В нем брутфорсом можно сейчас коллизию найти. Его имеет смысл использова только если у вас legacy система, которая ничего более современного не умеет.

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки | Неделя | Месяц

Если в глаз попал суперклей

+134 55,3k 73 114

Митинг против изоляции Рунета

+116 38,2k 22 459

кВт, кВт*ч и кВт/ч

+132 39,7k 148 270

Как Protonmail блокируется в России

+197 26k 76 156

Снижает ли скорость снижение скорости?

+49 23,6k 47 238