Sockets And Message Encryption/Decryption Between Client and Server Python 2.x 2.7

Cryptography is used for security purposes. There are not so many examples of Encryption/Decryption in Python using IDEA encryption MODE CTR. **Aim of this documentation :**

Extend and implement of the RSA Digital Signature scheme in station-to-station communication. Using Hashing for integrity of message, that is SHA-1. Produce simple Key Transport protocol. Encrypt Key with IDEA encryption. Mode of Block Cipher is Counter Mode

Examples

Client side Implementation

```
import time
import socket
import threading
import hashlib
import itertools
import sys
from Crypto import Random
from Crypto.PublicKey import RSA
from CryptoPlus.Cipher import IDEA

#animating loading
done = False
def animate():
    for c in itertools.cycle(['....','.......','..........','............']):
        if done:
            break
        sys.stdout.write('\rCONFIRMING CONNECTION TO SERVER '+c)
        sys.stdout.flush()
        time.sleep(0.1)

#public key and private key
random_generator = Random.new().read
key = RSA.generate(1024,random_generator)
public = key.publickey().exportKey()
private = key.exportKey()

#hashing the public key
hash_object = hashlib.sha1(public)
hex_digest = hash_object.hexdigest()

#Setting up socket
server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

#host and port input user
host = raw_input("Server Address To Be Connected -> ")
port = int(input("Port of The Server -> "))
```

Server side Implementation

```
import socket
import hashlib
import os
import time
import itertools
import threading
import sys
import Crypto.Cipher.AES as AES
from Crypto.PublicKey import RSA
from CryptoPlus.Cipher import IDEA

#server address and port number input from admin
host= raw_input("Server Address - > ")
port = int(input("Port - > "))
#boolean for checking server and port
check = False
done = False

def animate():
    for c in itertools.cycle(['....','.......','..........','............']):
        if done:
            break
        sys.stdout.write('\rCHECKING IP ADDRESS AND NOT USED PORT '+c)
        sys.stdout.flush()
        time.sleep(0.1)
```

```
    sys.stdout.write('\r -----SERVER STARTED. WAITING FOR CLIENT-----\n')
try:
    #setting up socket
    server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    server.bind((host,port))
    server.listen(5)
    check = True
except BaseException:
    print "-----Check Server Address or Port-----"
    check = False
```

Syntax

Parameters

Remarks

Language Used: Python 2.7 (Download Link: https://www.python.org/downloads/)

Library Used:

* **PyCrypto** (Download Link: https://pypi.python.org/pypi/pycrypto)

* **PyCryptoPlus** (Download Link: https://github.com/doegox/python-cryptoplus)

Library Installation:

PyCrypto: Unzip the file. Go to the directory and open terminal for linux(alt+ctrl+t) and CMD(shift+right click+select command prompt open here) for windows. After that write python setup.py install (Make Sure Python Environment is set properly in Windows OS)

PyCryptoPlus: Same as the last library.

Tasks Implementation: The task is separated into two parts. One is handshake process and another one is communication process. Socket Setup:

- As the creating public and private keys as well as hashing the public key, we need to setup the socket now. For setting up the socket, we need to import another module with “import socket” and connect(for client) or bind(for server) the IP address and the port with the socket getting from the user.

  ----------Client Side----------

  ```
  server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
  host = raw_input("Server Address To Be Connected -> ")
  port = int(input("Port of The Server -> "))
  server.connect((host, port))
  ```

  ----------Server Side---------

  ```
  try:
  #setting up socket
  server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
  server.bind((host,port))
  server.listen(5)
  except BaseException: print "-----Check Server Address or Port-----"
  ```

  “ socket.AF_INET,socket.SOCK_STREAM” will allow us to use accept() function and messaging fundamentals. Instead of it, we can use “ socket.AF_INET,socket.SOCK_DGRAM” also but that time we will have to use setblocking(value) .

Handshake Process:

- (CLIENT)The first task is to create public and private key. To create the private and public key, we have to import some modules. They are : from Crypto import Random and from Crypto.PublicKey import RSA. To create the keys, we have to write few simple lines of codes:

```
random_generator = Random.new().read
        key = RSA.generate(1024,random_generator)
        public = key.publickey().exportKey()
```

random_generator is derived from “ **from Crypto import Random** ” module. Key is derived from “ **from Crypto.PublicKey import RSA** ” which will create a private key, size of 1024 by generating random characters. Public is exporting public key from previously generated private key.

- (CLIENT)After creating the public and private key, we have to hash the public key to send over to the server

using SHA-1 hash. To use the SHA-1 hash we need to import another module by writing “import hashlib” .To hash the public key we have write two lines of code:

```
hash_object = hashlib.sha1(public)
hex_digest = hash_object.hexdigest()
```

Here hash_object and hex_digest is our variable. After this, client will send hex_digest and public to the server and Server will verify them by comparing the hash got from client and new hash of the public key. If the new hash and the hash from the client matches, it will move to next procedure. As the public sent from the client is in form of string, it will not be able to be used as key in the server side. To prevent this and converting string public key to rsa public key, we need to write server_public_key = RSA.importKey(getpbk) ,here getpbk is the public key from the client.

- (SERVER)The next step is to create a session key. Here, I have used “os” module to create a random key “key = os.urandom(16)” which will give us a 16bit long key and after that I have encrypted that key in “AES.MODE_CTR” and hash it again with SHA-1:

```
#encrypt CTR MODE session key
en = AES.new(key_128,AES.MODE_CTR,counter = lambda:key_128) encrypto = en.encrypt(key_128)
#hashing sha1
en_object = hashlib.sha1(encrypto)
en_digest = en_object.hexdigest()
```

So the en_digest will be our session key.

- (SERVER) For the final part of the handshake process is to encrypt the public key got from the client and the session key created in server side.

```
#encrypting session key and public key
E = server_public_key.encrypt(encrypto,16)
```

After encrypting, server will send the key to the client as string.

- (CLIENT) After getting the encrypted string of (public and session key) from the server, client will decrypt them using Private Key which was created earlier along with the public key. As the encrypted (public and session key) was in form of string, now we have to get it back as a key by using eval() . If the decryption is done, the handshake process is completed also as both sides confirms that they are using same keys. To decrypt:

```
en = eval(msg)
decrypt = key.decrypt(en)
# hashing sha1
en_object = hashlib.sha1(decrypt) en_digest = en_object.hexdigest()
```

I have used the SHA-1 here so that it will be readable in the output.

Communication Process:

For communication process, we have to use the session key from both side as the KEY for IDEA encryption MODE_CTR. Both side will encrypt and decrypt messages with IDEA.MODE_CTR using the session key.

- (Encryption) For IDEA encryption, we need key of 16bit in size and counter as must callable. Counter is mandatory in MODE_CTR. The session key that we encrypted and hashed is now size of 40 which will exceed the limit key of the IDEA encryption. Hence, we need to reduce the size of the session key. For reducing, we can use normal python built in function string[value:value]. Where the value can be any value according to the choice of the user. In our case, I have done “key[:16]” where it will take from 0 to 16 values from the key. This conversion could be done in many ways like key[1:17] or key[16:]. Next part is to create new IDEA encryption function by writing IDEA.new() which will take 3 arguments for processing. The first argument will be KEY,second argument will be the mode of the IDEA encryption (in our case, IDEA.MODE_CTR) and the third argument will be the counter= which is a must callable function. The counter= will hold a size of of string which will be returned by the function. To define the counter= , we must have to use a reasonable values. In this case, I have used the size of the KEY by defining lambda. Instead of using lambda, we could use Counter.Util which generates random value for counter= . To use Counter.Util, we need to import counter module from crypto. Hence, the code will be:

```
ideaEncrypt = IDEA.new(key, IDEA.MODE_CTR, counter=lambda : key)
```

Once defining the “ideaEncrypt” as our IDEA encryption variable, we can use the built in encrypt function to encrypt any message.

```
eMsg = ideaEncrypt.encrypt(whole)
#converting the encrypted message to HEXADECIMAL to readable eMsg =
eMsg.encode("hex").upper()
```

In this code segment, whole is the message to be encrypted and eMsg is the encrypted message. After encrypting the message, I have converted it into HEXADECIMAL to make readable and upper() is the built in function to make the characters uppercase. After that, this encrypted message will be sent to the opposite station for decryption.

- **(Decryption)**

To decrypt the encrypted messages, we will need to create another encryption variable by using the same arguments and same key but this time the variable will decrypt the encrypted messages. The code for this same as the last time. However, before decrypting the messages, we need to decode the message from hexadecimal because in our encryption part, we encoded the encrypted message in hexadecimal to make readable. Hence, the whole code will be:

```
decoded = newmess.decode("hex")
ideaDecrypt = IDEA.new(key, IDEA.MODE_CTR, counter=lambda: key)
dMsg = ideaDecrypt.decrypt(decoded)
```

These processes will be done in both server and client side for encrypting and decrypting.