mySSL-- TLS的简易模拟实现

mySSL的接口是**模仿** Python标准库的`ssl`模块

1 使用示例

1.1 套接字创建

使用`SSLContext`实例的`SSLContext.wrap_socket()`来将套接字包装为`SSLSocket`对象。

客户端套接字实例:

```
import socket
import mySSL

host = '127.0.0.1'
port = 12000
sock = socket.create_connection((host, port))

# 生成SSL上下文
context = mySSL.SSLContext()
ssock = context.wrap_socket(sock, False)
......
ssock.close()
```

服务器套接字实例:

```
import socket
import mySSL

host = '127.0.0.1'
serverPort = 12000
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((host,serverPort))
sock.listen(1)

context = mySSL.SSLContext()
ssock = context.wrap_socket(sock, True)

while True:
    connectionSocket, addr = ssock.accept()  # 接收到客户连接请求后，建立新的TCP连接套接字
    ......
    ssock.close()
```

1.2 上下文创建

`SSLContext`的设置仅为了形式上与Python标准库的`ssl`一致， `SSLContext`内仅实现了`SSLContext.wrap_socket()`一个方法。

2 SSLSocket

`class mySSL.SSLSocket(socket.socket, server_side)`

重要参数：

- socket　传入套接字
- server_side 布尔类型 用于说明是否是服务端

mySSL 套接字提供了套接字对象的下列方法:

- accept()
- recv()
- send()
- close()

其他的一些方法：

- viewKey()
- server_do_handshake()
- client_do_handshake()
- create()

---

create()

创建`SSLSocket`的方法，创建`SSLSocket`对象时必须调用。

传入参数：

- TCP套接字`socket`
- 服务端客户端布尔类型表示`server_side`

  服务端传入`True`，客户端传`False`

返回值：

- `SSLSocket`本身

accept()

`accept`仅能用于服务端，调用`accept`**默认**会调用`server_do_handshake`与客户端进行握手，若没有客户端连接，`accept`会阻塞。

无传入参数

返回值：

- 连接上的客户端套接字`connectionSocket`
- 客户端地址`addr`

示例：

```
connectionSocket, addr = ssock.accept()
```

recv()

`recv`会首先判断是否连接成功，若未连接，则**先调用**握手代码进行握手流程。

传入参数：

- 返回的字节数`size`

返回值：

- 返回`size`大小的收到内容，若收到的内容大于`size`，超出的部分将会被简单丢弃

send()

`send`会首先判断是否连接成功，若未连接，则**先调用**握手代码进行握手流程。

传入参数：

- 要发送的数据`data`

无返回值

close()

等价于直接调用TCP套接字的`close`，可以不使用

viewKey()

打印连接过程中的所有密钥，仅供测试使用，打印顺序为：

self.pre_master_secret

self.master_secret

self.server_mac_secret

self.client_mac_secret

self.server_write_secret

self.client_write_secret

server_do_handshake()

服务端TLS握手方法，若没有客户端进行连接，`server_do_handshake`调用后会**被阻塞**。

client_do_handshake()

客户端TLS握手方法

3 异常

`exception mySSL.SSLerror`

`mySSL`在握手过程如果出现错误就会引发`SSLerror`异常，`SSLerror`异常继承于`RuntimeError`， 它表示在下层网络连接之上叠加的高层级加密和验证层存在某种问题。

```
class SSLerror(RuntimeError):
    def __init__(self, arg):
        self.args = arg
```

4 辅助文件

4.1 PRF

`PRF`伪随机函数在`mySSL`中用来扩展密钥以得到密钥产生和验证中的各种密钥块。采用`PRF`伪随机函数的目的是使用相对较小的共享密钥值，生成较长的数据块，防止对散列函数和 MAC 的攻击。伪随机函数`PRF`是单独作为一个类编写在`PRF.py`中，使用了Python标准库中的`hmac`库，使用`sha256`散列函数，密钥生成的格式如下：

```
 # secret:    密钥(预备主密钥)
 # label:     标签(目标密钥类型)
 # seed:      种子(随机数)
keyGenerator = PRF.prf(secret, label, seed)
 # size为希望的字节数
key = keyGenerator.ouput(size)
```

4.2 AEScbc

`AEScbc.py`文件中实现了对任意大小的数据进行CBC模式AES加密。

加密：

```
 # data:          要加密的数据
 # secret:        密钥
 # iv:            初始偏移量
ciphertext = AEScbc.encrypt(data, secret, iv)
```

解密：

```
 # ciphertext:    要解密的数据
 # secret:        密钥
 # iv:            初始偏移量
data = AEScbc.decrypt(ciphertext, secret, iv)
```

4.3 packet方法

测试用方法，用于打印clientHello的报文内容，正常使用`mySSL`时无需使用，也不推荐使用，效果如下：

```
[*]TLSContentType:      HANDSHAKE
[*]TLSMessageType:      CLIENT_HELLO
[*]TLSMessage length:   5
------------------------
[*]TLSVersion:  TLSv1_2
[*]Random:
b'\x08\xd2\xb7\xa9<\x0fl\xb0\x08\xdcv\xb9\x92\x87Lu\x91-(\xd4>
[\x86\x83\xcb\xb7z\xea'
[*]session ID:
0
[*]TLSCipherSuites:
['TLS_AES_128_GCM_SHA256', 'TLS_AES_256_GCM_SHA384',
'TLS_CHACHA20_POLY1305_SHA256', 'TLS_AES_128_CCM_SHA256',
'TLS_AES_128_CCM_8_SHA256', 'TLS_RSA_WITH_AES_256_CBC_SHA256']
[*]Compression Methods:
None
```