Information Systems Security
Final project report

Minas Katsiokalis (2011030054)

1 INTRODUCTION

The final project is composed by 3 parts. The first part was about encryption, decryption (symmetric/ asymmetric), hashing, signing and validation of signing and certificate. The second part was an introduction to SSL (secure socket layer) implementation. A connection should be established between a server and a client. The third part was more advance techniques about SSL and how encrypted connection can be achieved.

2 PART 1

2.1 AES-128 encryption/decryption (aes.py)

For this implementation AES for Crypto.Cipher used.
Encryption method, takes an in_file, and out_file, an IV, and a key. In_file and key by user. The data that's saved in the in_file are encrypted with AES-128, CBC mode, PKCS#5 padding using the key the user has inserted. IV generated(see Menu). The result is stored in the out_file.
Decryption method, takes in_file, and out_file, an IV, and a key. In_file and key by user. The data that's saved in the in_file are decrypted with AES-128, CBC mode, PKCS#5 padding using the key the user has inserted, IV generated(see Menu). The result is stored in the out_file.

2.2 SHA-256 hashing (sha.py)

For this implementation Hashlib used.
A simple hashing method, takes in_file and out_file as attributes. Reads the data which is saved in the in_file, hash them with SHA2-256, and stores the result in the out_file.

2.3 Signing and verification (sign.py)

For this implementation RSA from Crypto.PublicKey used.
Contains two functions one for signing and one for signature verification.
Signature methods need as attributes a file_in with data to be signed, and a file with the private key that is going to used in order to sign the data. Before we sign the data we hash them with SHA-256 for more security. The signature (the result) is stored to a file. Verification method, needs as attributes an file_in as the signed filed, a file with the public key of the key that is used for signing, and the file with the signature. The method decrypts the signatures with the public key, Hash the data with SHA-256 of the signed filed and check if these two are the same. If they are, then the verification message is shown in the console.

2.4 Certificate Verification (cert_verification.py)

Crypto from OpenSSL used.
A method that requires a file of certificate, and the name of the public key owner. The method extract the data from the encrypted certificate. Checks the expiration data and the subject's name.
If the certificate has not expired and the subject's name is the same with public key owner, then the certificate is verified.

2.5 Menu (part1.py)

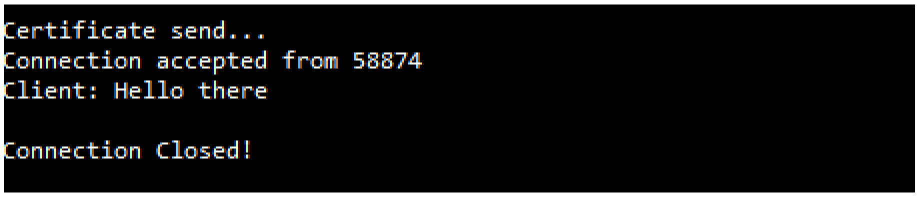
All the above can be tested using the main program code part1.py. Program offers a menu of option and guide the user through them. For better experience see 'README.txt' file for further information about the usage of the program.

---

3 PART 2

3.1 Server side (server.py)

In this module a simple Server is implemented. The server creates a python socket, sets the hostname (localhost at this point) and the port that is used. Bind them and starts to listen for client to be connected. The server stays at "listening-mode" for 10 seconds. Once a client asks for connection, then the server send his certificate, and key through a wrap socket in order to be verified by the client (one way authentication). If the verification is successful then server waits for the client to send data. When the client the sends the data, server get them and print them on the console. When the timeout has passed the connection is terminated.

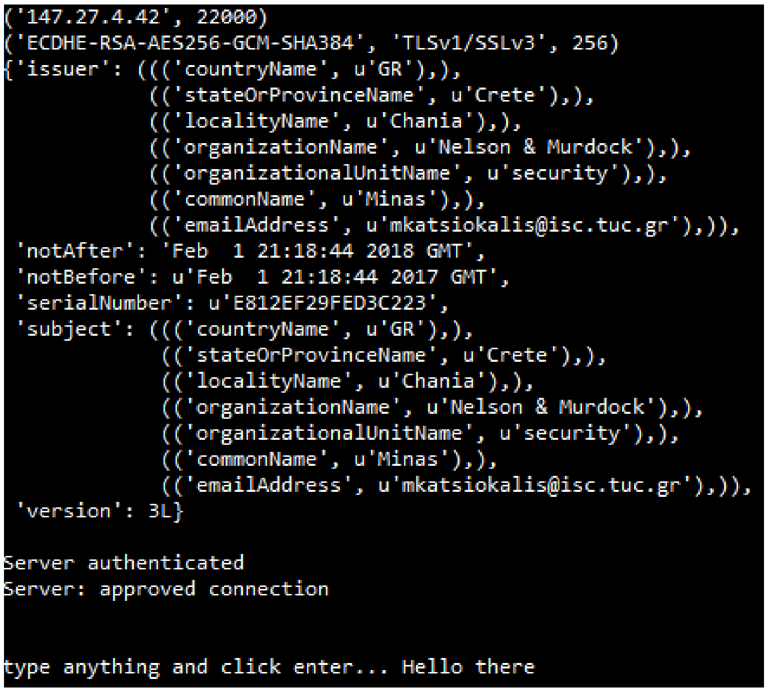


```
Certificate send...
Connection accepted from 58874
Client: Hello there

Connection Closed!
```

Figure 1 Example of server in part 2

3.2 Client side (client.py)

In this module a simple Client is implemented. The client creates a socket and make it to require certificate in order to proceed in connection. As ca file we use the same file with certificate because it is self-signed. Client tries to connect to the port and hostname the server has define (hostname is again local host-same machine). If the connection and the authentication were successful, information about the server shown in client's side. Then asks the user to write some data to send to the server. Client sends the data and terminates the connection.



```
('147.27.4.42', 22000)
('ECDHE-RSA-AES256-GCM-SHA384', 'TLSv1/SSLv3', 256)
{'issuer': ((('countryName', u'GR'),),
            (('stateOrProvinceName', u'Crete'),),
            (('localityName', u'Chania'),),
            (('organizationName', u'Nelson & Murdock'),),
            (('organizationalUnitName', u'security'),),
            (('commonName', u'Minas'),),
            (('emailAddress', u'mkatsiokalis@isc.tuc.gr'),)),
 'notAfter': 'Feb  1 21:18:44 2018 GMT',
 'notBefore': u'Feb  1 21:18:44 2017 GMT',
 'serialNumber': u'E812EF29FED3C223',
 'subject': ((('countryName', u'GR'),),
             (('stateOrProvinceName', u'Crete'),),
             (('localityName', u'Chania'),),
             (('organizationName', u'Nelson & Murdock'),),
             (('organizationalUnitName', u'security'),),
             (('commonName', u'Minas'),),
             (('emailAddress', u'mkatsiokalis@isc.tuc.gr'),)),
 'version': 3L}

Server authenticated
Server: approved connection


type anything and click enter... Hello there
```

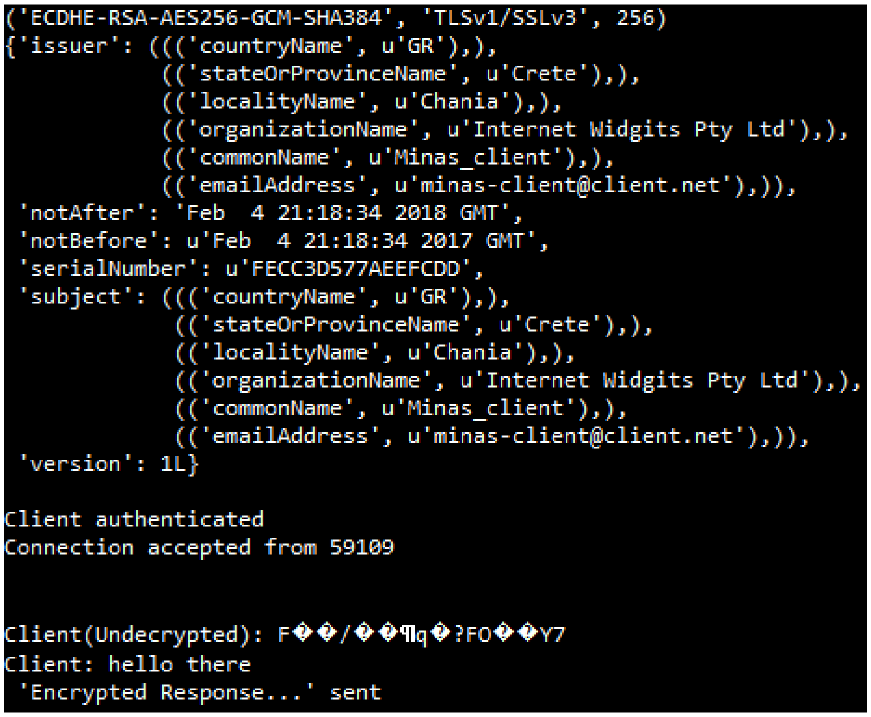
Figure 2 Client example from part 2

More information on execution can be found in file 'README.txt'.

4 PART 3

4.1 Server side (server_app.py)

In this module a simple Server is implemented. The server creates a python socket, sets the hostname (localhost at this point) and the port that is used. Bind them and starts to listen for client to be connected. The server stays at "listening-mode" for 10 seconds. Once a client asks for connection, then the server send his certificate, and key through a wrap socket in order to be verified by the client. He also wait for the client to send his certificate in order to authenticate him. If the verification is successful information about client is shown to the server side. Then server extracts the public key of the client from his certificate. Generates a random AES-128 key (16 bytes) and encrypts it with client's public key. Then he sends the encrypted AES-key to the client and a random generated IV (for AES-128 CBC-mode). Server wait for client to respond. Server waits encrypted data from client with AES-128, using the key that he created before. When the data arrives, server gets them decrypts them and show them on the console. If the data is the same with the uncrypted data that client has sent, that means the key has been exchanged with success. Server responds with encrypted data too, so client can understand that he gets the right key.

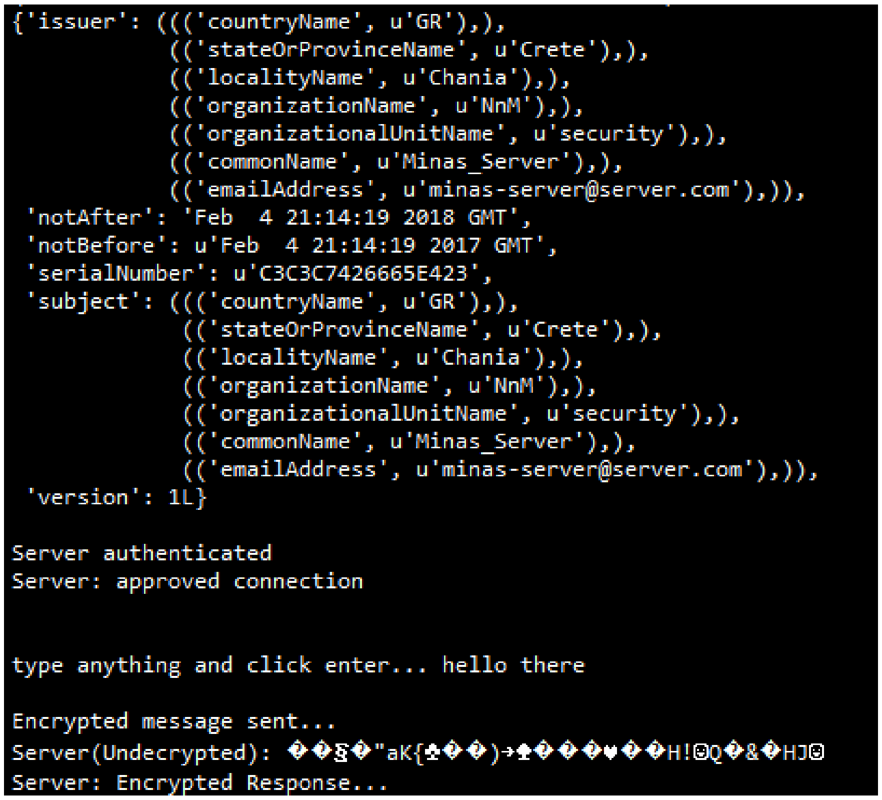


```
('ECDHE-RSA-AES256-GCM-SHA384', 'TLSv1/SSLv3', 256)
{'issuer': ((('countryName', u'GR'),),
            (('stateOrProvinceName', u'Crete'),),
            (('localityName', u'Chania'),),
            (('organizationName', u'Internet Widgits Pty Ltd'),),
            (('commonName', u'Minas_client'),),
            (('emailAddress', u'minas-client@client.net'),)),
 'notAfter': 'Feb  4 21:18:34 2018 GMT',
 'notBefore': u'Feb  4 21:18:34 2017 GMT',
 'serialNumber': u'FECC3D577AEEFCDD',
 'subject': ((('countryName', u'GR'),),
             (('stateOrProvinceName', u'Crete'),),
             (('localityName', u'Chania'),),
             (('organizationName', u'Internet Widgits Pty Ltd'),),
             (('commonName', u'Minas_client'),),
             (('emailAddress', u'minas-client@client.net'),)),
 'version': 1L}

Client authenticated
Connection accepted from 59109


Client(Undecrypted): F��/��¶q�?FO��Y7
Client: hello there
 'Encrypted Response...' sent
```

Figure 3 server example of part 3

4.2 Client side (client_app.py)

In this module a simple Client is implemented. The client creates a socket and make it to require certificate in order to proceed in connection. As ca file we use the same file with certificate because it is self-signed. Client tries to connect to the port and hostname the server has define (hostname is again local host-same machine). Sends his certificate and key in order to be authenticated by the server too. If the connection and the authentication were successful, information about the server shown in client's side. Then he waits for the server to send him the encrypted AES-128 key. When he gets the key, he used his private key to decrypt it and use it for the communication. App ask from user to enter some data via console. The data are encrypted with AES-128, CBC mode and they are send to the server. Client waits for a server response encrypted with the AES-key. If the response can be decrypted that means the key has been exchanged with success.



```
{'issuer': ((('countryName', u'GR'),),
            (('stateOrProvinceName', u'Crete'),),
            (('localityName', u'Chania'),),
            (('organizationName', u'NnM'),),
            (('organizationalUnitName', u'security'),),
            (('commonName', u'Minas_Server'),),
            (('emailAddress', u'minas-server@server.com'),)),
 'notAfter': 'Feb  4 21:14:19 2018 GMT',
 'notBefore': u'Feb  4 21:14:19 2017 GMT',
 'serialNumber': u'C3C3C7426665E423',
 'subject': ((('countryName', u'GR'),),
             (('stateOrProvinceName', u'Crete'),),
             (('localityName', u'Chania'),),
             (('organizationName', u'NnM'),),
             (('organizationalUnitName', u'security'),),
             (('commonName', u'Minas_Server'),),
             (('emailAddress', u'minas-server@server.com'),)),
 'version': 1L}

Server authenticated
Server: approved connection


type anything and click enter... hello there

Encrypted message sent...
Server(Undecrypted): ��§�"aK{♣��)→♠���♥��H!☺Q�&�HJ☺
Server: Encrypted Response...
```

Figure 4 client example of part 3

More information on execution can be found in file 'README.txt'

ACKNOWLEDGMENTS

My co-students helped me a lot via courses forum. Implementations and examples found on GitHub.com and StackOverflow.com were useful too.

REFERENCES

[1] stackoverflow.com
[2] docs.python.org
[3] www.sslshopper.com
[4] stackexchange.com
[5] github.com
[6] pyopenssl.sourceforge.net
[7] www.laurentluce.com