Student Declaration of Authorship

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

Course code and name:	F21CN Computer Network Security
Type of assessment:	**Group**
Coursework Title:	Trusted Documents Broadcasting
Student Name:	Sasha Fathima Suhel (50%)
Student ID Number:	H00410394

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Anyuses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penaltiesthat I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarismexplained in the University guidance on Academic Integrity and Plagiarism

Student Signature: *Sasha Fathima Suhel*

Date: *05/12/2022*

Student Declaration of Authorship

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

Course code and name:	F21CN Computer Network Security
Type of assessment:	**Group**
Coursework Title:	Trusted Documents Broadcasting
Student Name:	Salman Ansari (50%)
Student ID Number:	H00410360

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Anyuses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penaltiesthat I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarismexplained in the University guidance on Academic Integrity and Plagiarism

Student Signature: *Salman Ansari*

Date: *05/12/2022*

Computer Network Security

F21CN

Coursework 2

Trusted Documents Broadcasting

Index

S. no	Title
1	Introduction
2	Requirement Checklist
3	Implementation
	3.1. Certifications and Signatures
	3.2. Application Development
4	Conclusion
5	Reflections on programming language and implementation
6	References

1. Introduction

This coursework consists of creating two certificate files, x509 and PGP. There are two parts to this coursework in which the initial component is achieved on CentOS 9 (Linux). All the tasks relating to their certificates and signatures were created on the terminal using Openssl toolkit commands. From this PGP keys and CA (Certification Authority) certificates were issued and shared among peers. All the tasks from this component listed in the coursework are achieved which will be explained in the report.

The second component of the application is creating an application that verifies the authenticity of the broadcast documents with the help of the files created in the first component. The application is utilized to read a document and its signature and check the source from the trusted authors with the help of using PGP certificates. This list of trusted authors is certified by the CA. The application (Doc Verifier) has two parts being the Server (Server_App) and the Receiver (Receiver_App) which must be opened on the Windows terminal simultaneously. A readme file guides the initial setup process to run the application successfully.

We expected to learn advanced Openssl commands and encryption on both windows Operating System and CentoOS. We also expected to learn and comprehend how and why this specific type of certificates would be implemented. We anticipated that the entire application and tasks would be only plausible on CentOS. But to our benefit deployed the initial task on CentOS and the later application on Windows OS with the help of Visual Studio.

System Environment: Windows OS Version 10.0.22621, CentOS 9, Python 3.10.7, OpenSSL 3.0.7, Visual Studio Community 2022 17.3.6

2. Requirement Checklist

The following are achieved in the coursework:

Certificates and Signatures

1) Created Self signed PGP certificates and public and private keys individually
2) Signed multiple PGP certificates of other peers
3) Signed a document using our individual private key which was shared among peers
4) Created a X509 certificate and private key known as Client
5) Created a local CA and Client as an intermediate certificate
6) Signed an intermediate certificate by another CA

Application Development

1) Created a separate form for Server GUI and Receiver GUI
2) Contains a network interface
3) Be able to send and receive documents through the network
4) Verify the authenticity of the signature by checking whether the document was signed by a trusted author or not. Positive and negative testing accomplished

3. Design Considerations

3.1 Certificates and Signatures

All the functionalities of these tasks were created on CentOS for ease of implementation with the use of Openssl package.

I. First, we created self-signed PGP keys for our team Sasha and Salman. We ran "gpg --full-generate-key" and created signed certificates and created public and private individually for both of us. We also created a CRL for our certificates if at any time we decided to revoke our certificates.

```
pub   rsa4096 2022-11-15 [SC] [expires: 2023-11-15]
      5F8A0D598B89EF26E10FB50D66171B4CA3AF50D5
uid           [  full  ] salman <ssa2006@hw.ac.uk>
sub   rsa4096 2022-11-15 [E] [expires: 2023-11-15]
```

Figure 2 Generating PGP certificate for Salman

```
pub   rsa4096 2022-11-15 [SC] [expires: 2023-11-15]
      6BA77A5D5744AE7D6A898A9B88B41B7C51F0F193
uid           [ultimate] Sasha <sfs2001@hw.ac.uk>
sub   rsa4096 2022-11-15 [E] [expires: 2023-11-15]
```

Figure 1 Generating PGP certificate for Sasha

II. The next step was to sign multiple PGP certificates of our peers around us who have created PGP certificates as well. This process can only be achieved if our keys are upload to a server which can then be accessed by other peers to sign and send their keys. We used the server "keyserver.ubuntu.com" on Linux and uploaded our certificates first which can then be accessed by others. Once our keys can be accessed either through our key numbers or email ID's, our keys would then be signed by others by running "gpg --sign-key AXFX9XX2XXF7XXX6E7X1X". This process is repeated until it is done vice versa

```
pub rsa4096/5f8a0d598b89ef26e10fb50d66171b4ca3af50d5 2022-11-15T05:27:01Z
        Hash=48b32d61ef4f93a48085099f0143d2e5

uid salman <ssa2006@hw.ac.uk>
sig  sig  66171b4ca3af50d5 2022-11-15T05:27:01Z 2023-11-15T05:27:01Z _____ [selfsig]
sig  sig  00f42aba135b022e 2022-11-16T01:37:02Z _____ _____ 00f42aba135b02
sig  sig  a22c206116062eda 2022-11-16T17:09:44Z _____ _____ a22c206116062e
sig  sig  88b41b7c51f0f193 2022-11-16T17:19:02Z _____ _____ 88b41b7c51f0f1
sig  sig  63fa4cf0c4fd062e 2022-11-16T17:36:20Z _____ _____ 63fa4cf0c4fd06
sig  sig  314d4bac3f6cbc4e 2022-11-16T17:37:22Z _____ _____ 314d4bac3f6cbc
sig  sig  9be848f7fc762fee 2022-11-16T17:46:47Z _____ _____ 9be848f7fc762f
sig  sig  f20904bf4a7eb48e 2022-11-16T20:32:01Z _____ _____ f20904bf4a7eb4
sig  sig  cfbf5b9031d57c1a 2022-11-20T13:07:41Z _____ _____ cfbf5b9031d57c
sig  sig  b87121ed8b833869 2022-11-21T08:32:28Z _____ _____ b87121ed8b8338
sig  sig  9a836d034d897766 2022-11-22T18:34:03Z _____ _____ 9a836d034d8977
sig  sig  189ed2056efd86cb 2022-11-23T10:06:34Z _____ _____ 189ed2056efd86

sub rsa4096/5343bc5c985c3727e66d3b1402dc232e3b1b18eb 2022-11-15T05:27:01Z
sig sbind 66171b4ca3af50d5 2022-11-15T05:27:01Z _____ 2023-11-15T05:27:01Z []
```

Figure 3 PGP Received Signatures

where we sign many other PGP certificates of our peers. These signatures can then be viewed on the website showing the entire log of the time and data of the signer and the signee.

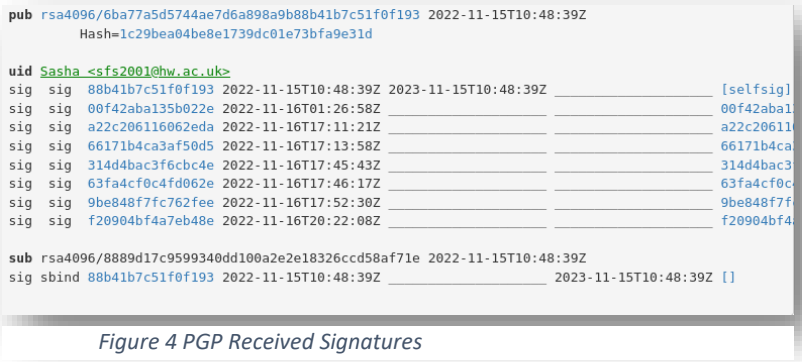

Figure 4 PGP Received Signatures

III. Once we have all the basic files, we then create a document which is going to be signed by our PGP private keys which will be sent to other students and will be used in the application development later in this report. This signed document will be utilized for verification by the server. This text file (doc.txt) was used by both the team members to create a signed document (.asc) file. We ran the command "gpg –encrypt –sign –armor -r doc.txt > doc.sig" on Linux.

IV. In this task our aim was to create a local CA and an intermediate CA certificate. We referred the website https://www.golinuxcloud.com/openssl-create-certificate-chain-linux/ which helped us to create CA and intermediate certificates for our pair. A separate document that was signed by our CA was shared with another pair and vice versa. This process was simple to do on Openssl but required some editing prior to creation. We obtained the "Openssl.cnf" file from the root directory (/etc/pki/tls) and made changes to create local CA. This reference also let us create private and public keys for our CA certificate.

V. Once the local CA was created from the above reference, we made Client certificate as a certificate signed by our CA. This certificate is what's been utilized in the application to verify the and obtain the public key and used the private key to encrypt the document from above signed by our PGP private key.

VI. The last task from this component is to have a certificate signed by another CA from another pair which was completed and located in a separate folder. Simultaneously we signed the certificate for another pair completing the last component of this task

3.2 Application Development

We have created the entire application using python tkinter. The application consists of 2 different independent applications: Server and Receiver. The server application includes the main part of the application, and the receiver part handles receiving files from the network sent via the server side of the application.

The server app consists of the following functionalities:

- Load all the files, creating CSR and sign document
- Sending files via network

- Verify other pair document and signature
- Printing the logs of the operation
- Certificate revocation
- PGP verification for documents received

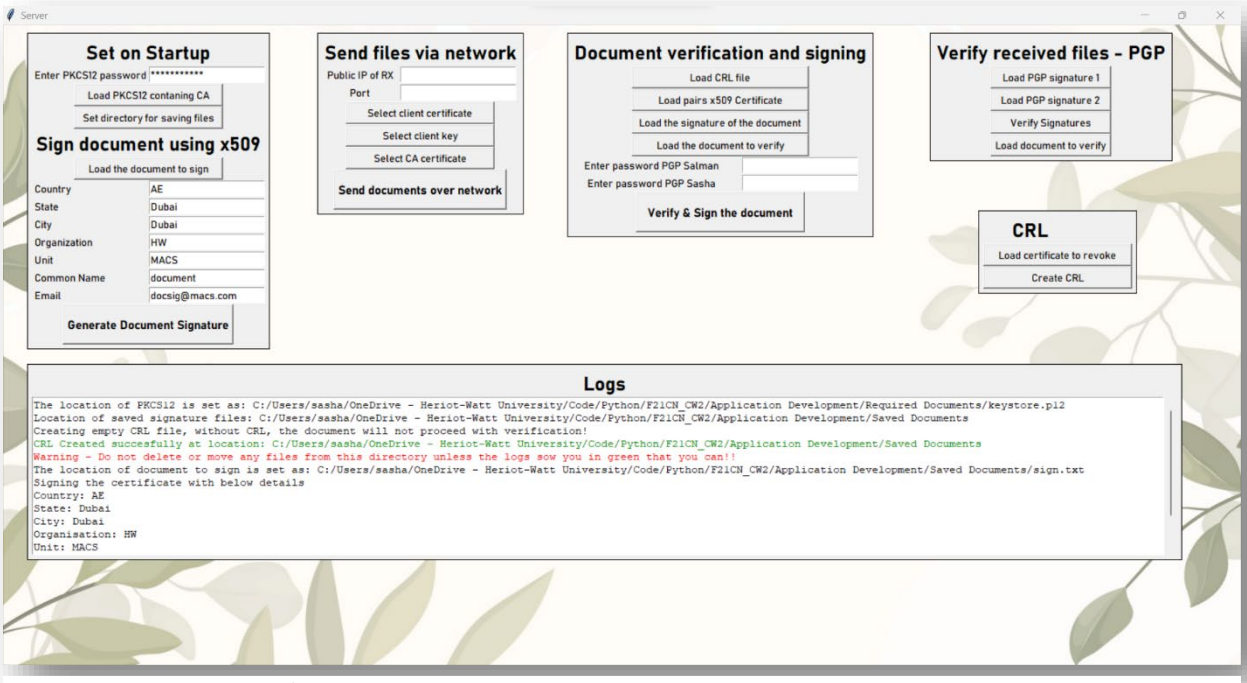

Figure 5 Server Application GUI

The receiver app consists of the following functionalities:

- Receiving the documents
- Printing the logs of the operation

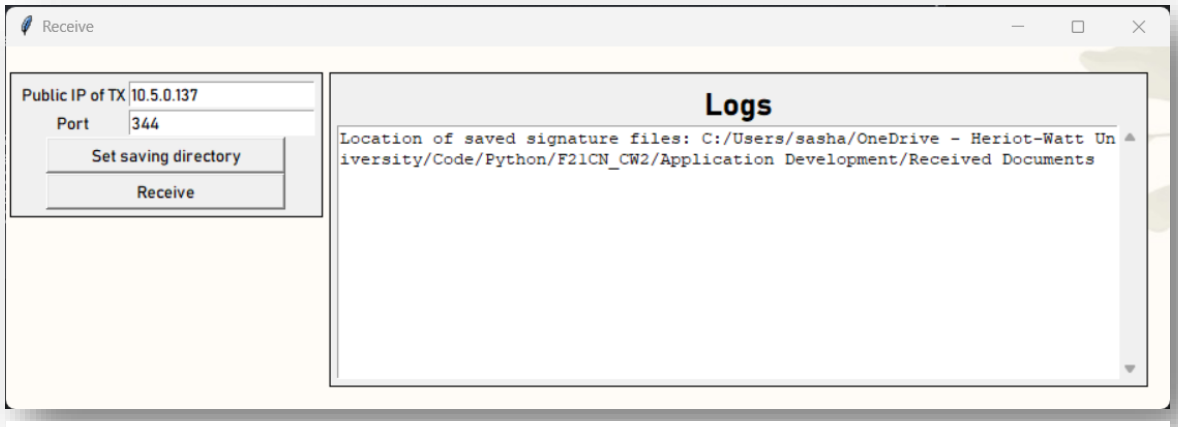

Figure 6 Receiver Application GUI

On application startup, the user must enter the PKCS12 password which contains all the list of certificates stored in it. Next step is to load the PKCS12 containing CA and select the directory for saving the files.

We can sign the document using X509 certificate. We load the document to sign, enter the details and

click on the generate document signature button.

The next part is to send files via network. The user must enter the IP address of the receiver and the port number. The user must select Client certificate, Client key and CA certificate to send over the network. Once selected the user can send the documents over the network.

The next part involves Document verification and signing. We load the CRL file along with the pair X509 certificate, signature, and the document to verify. Then enter the password for both the pair and click on the 'Verify and Sign Document' button.

The next part includes verifying the received files. The user can load both the PGP signatures and the document to verify. The user can check the log for the status of verification.

As part of adding extra features, we have added the functionality of CRL. The User can revoke the certificate by loading the appropriate certificate to revoke.

4. Conclusion

Even though both PGP and X509 have different implementations in real life for different scenarios, in this application we were able to understand the usage of both under one context. By verifying the documents with our PGP keys while using x509 as a mechanism through it. Most of the development went through understanding how to use and incorporate the network interface into this application compared to the actual encryption. This application may not yet be used by a wide public but still gave us a great understanding behind implementing this application. This application was created on a Windows OS system with the help of Visual Studio but also can be imported with some changes into any Linux OS.

5. Reflections on programming language and implementation

From this application one of the main difficulties, we faced was during the network interface. As it progressed to be a lot harder than we initially expected. We initially tested this application on CentOS since the initial certificates and signatures component was completed on that OS. But later we faced many issues and technical syntax and path errors since we lacked advanced knowledge in Linux.

We felt that creating this on a Windows OS and using python as the programming knowledge made this process easier to compute rather than if we tried other languages. Due to the slow lag time most linux systems have when opened in a virtual box. Windows OS systems showed to have a much faster computation compared to otherwise.

6. References

[1] https://docs.python.org/3/library/tkinter.html

[2] https://sectigo.com/resource-library/what-is-x509-certificate

[3] https://pypi.org/project/py-pgp/