

**Third Review Document**

**Email System securing and encrypting channel based on PGP**

**Name - Shourya Maheshwari**  
**Reg. No. – 18BCI0179**  
**Mobile No. - 6377308780**  
**Mail Id.- shourya.maheshwari2018@vitstudent.ac.in**

**Name - Laksh Gupta**  
**Reg. No. – 18BCI0190**  
**Mobile No. - 8447200324**  
**Mail Id.-laksh.gupta2018@vitstudent.ac.in**

**Name - Mihir Srivastava**  
**Reg. No. – 18BCI0214**  
**Mobile No. - 9971289345**  
**Mail Id.-mihir.srivastava2018@vitstudent.ac.in**

**Guide Name-SENDHIL KUMAR K.S**  
**Designation-Associate Professor Grade 1**  
**Mobile No.- 9840068152**  
**Mail ID-sendhilkumar.ks@vit.ac.in**

**B.Tech.**

**in**

**Computer Science and Engineering with Specialization in Information Security**

**School of Computer Science & Engineering**

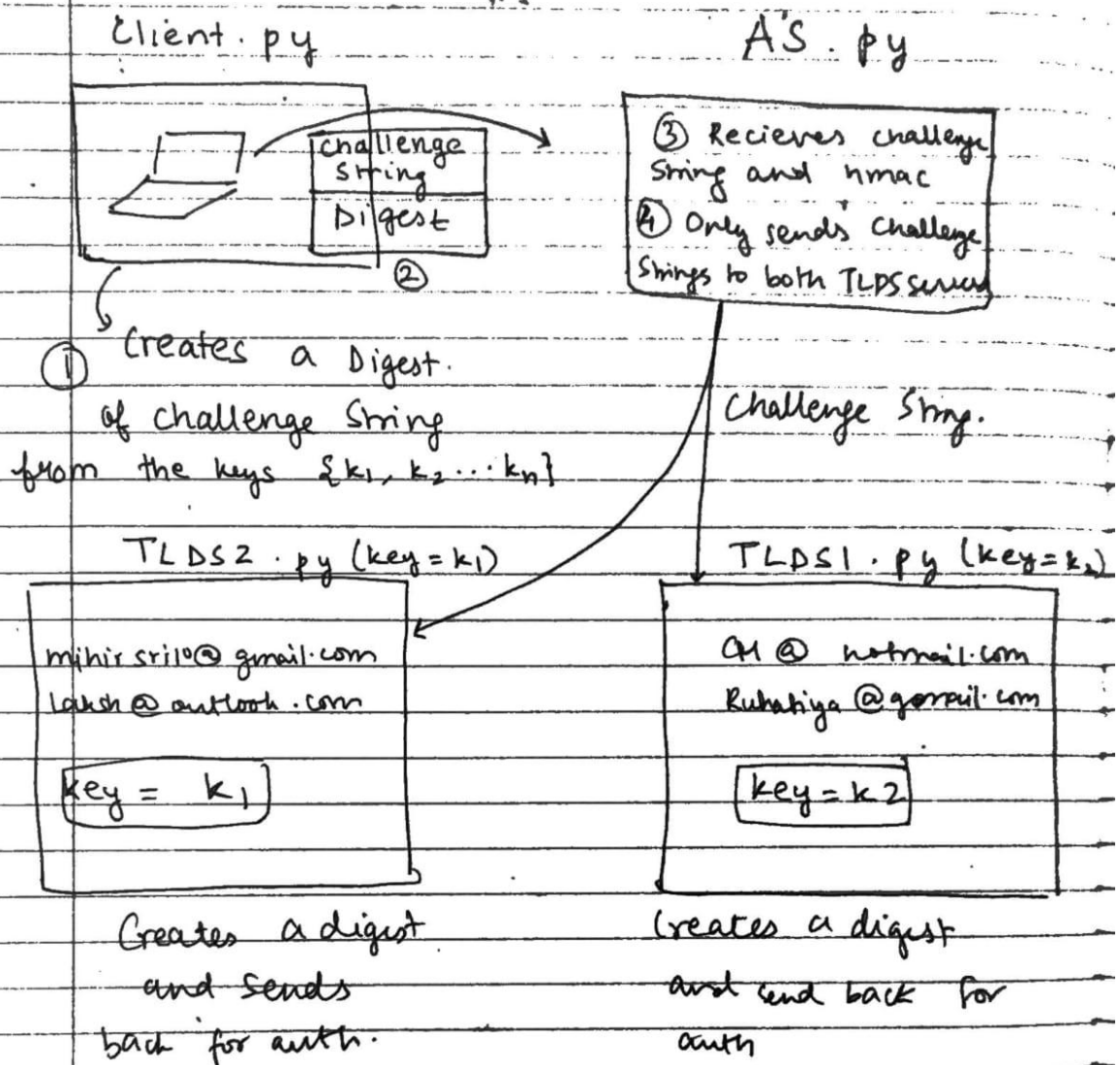


**VIT<sup>®</sup>**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

**Abstract:**

Here we try to implement a custom email application using techniques like ssl and tls, and an openpgp application using pretty good privacy itself and due to this when we try to intercept packets while sending the email, we find the packets to be encrypted. Our email app helps protect the email by converting the content of the email to an unreadable gpg file, to read this gpg file we need a secret key that is known only to the sender of the email and the sender has to send this to the receiver via some other secure channel. We use TLS for securing the line of smtp. When you use a standard POP or IMAP connection to download your email (the most popular method still in use), your username and password is sent in cleartext across the Internet. This means that anyone using the same wireless connection as you, or the same network as you, or watching traffic at your ISP - or anyone in a position to see your Internet traffic can potentially "intercept" your network traffic and clearly read your username and password. With this information, they can easily read all your email and worse, steal confidential information, send out spam or other malicious acts. Also To create an authentication system which will detect any DNS spoofing or DNS cache poisoning and prevent the server from returning an incorrect result record or IP Address.

## System Architecture



Switch on all  
the servers ~~etc~~,  
Authentication,  
TLS1, TLS2,

Create a MNS.txt  
file containing  
wanted emails.

Start Client  
server

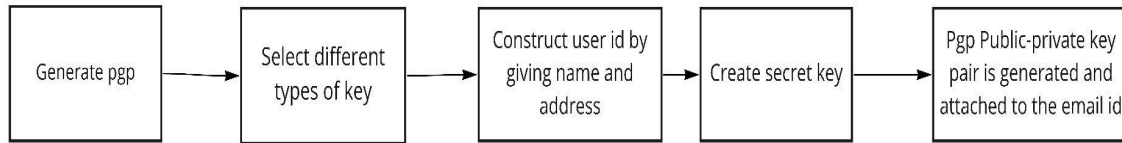
Client server  
gives ~~message~~  
to Auth. Server,  
A Challenge String  
and a digest using  
a key

Client server  
sends both servers  
only the ~~message~~ digest  
challenge string

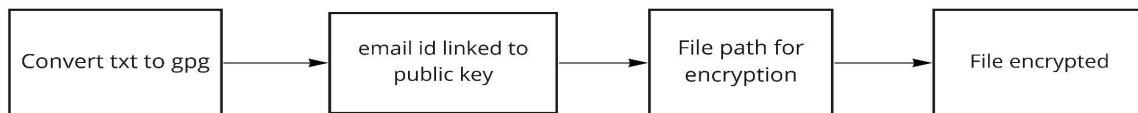
Both TLS's  
create a digest  
& send back  
to AS.py

Then accordingly  
client contacts  
correct TLS  
and fetches  
whether Email  
Exist?

Now AS.py tells  
client whether  
key is from  
(TLS2, TLS1, or Fake)

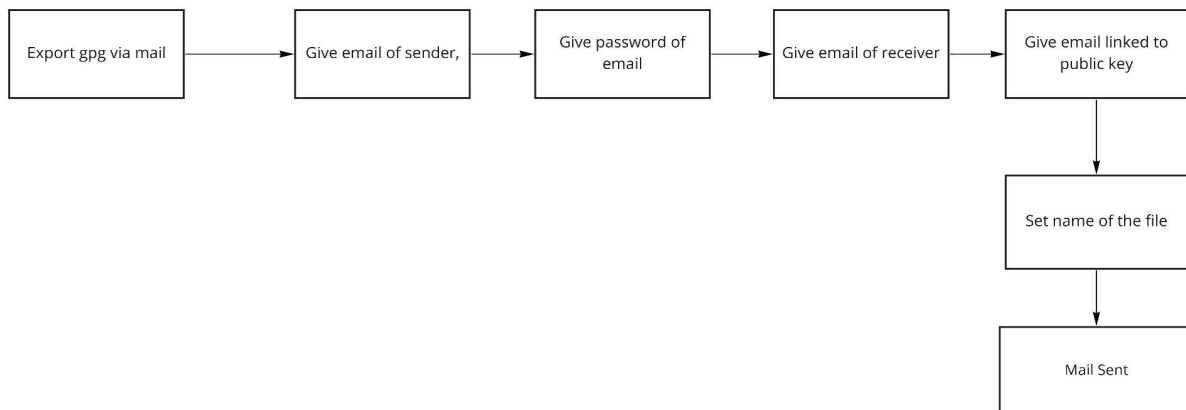


### Creating PGP public private key pair

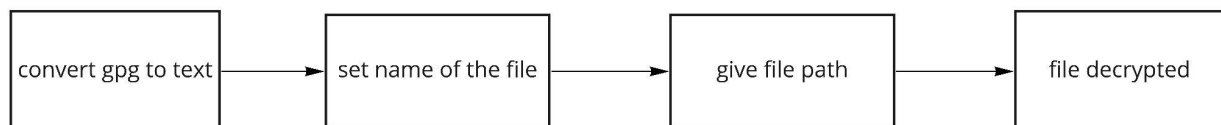


---

### Converting Email to a gpg file



### Exporting the GPG file via Email



### Decoding the received gpg file

## Methodology Adapted

**Pretty Good Privacy (PGP)**-is a program that converts text to a pgp file and provides cryptographic privacy and authentication for data communication. PGP is used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications.

**Custom python application**-Python console emailer is a console (terminal) based python application that allows the user to sign in to their email and generate a PGP pair. The application allows the user to insert a morphed text file in the email via the command line (terminal).

**Email Spoofing prevention python program**-DNS cache poisoning is the act of entering false information into a DNS cache, so that DNS queries return an incorrect response and users are directed to the wrong websites. Assuming an entity needs to spoof the identity of some. Other entity, it's enough to vary the mapping between its low level address and its high level name. It means that an assailant will pretend the name of somebody by modifying the association of his address from his own name to the name he needs to impersonate. Once an assailant has done that, an appraiser will no longer distinguish between truth and fake entity ex- >google.com and gogle.com. Similarly we are trying to apply the same concept for emails.

## Expected Results with discussion

In the open source Openpgp application with the help of a tool first we want to generate our PGP pair for the email ID.

Then we wish to share our public key with people who want to send mail to us.

Then we will try to send this gpg file via email using our custom python app.

After receiving the mail from others we will try to open the gpg file using our private key.

This will help us to protect our mail from any attacker who try to see our mail.

On the other hand the purpose of this custom email project is to gain a greater understanding of how emails are created, secured, and sent. The application can be developed with the smtplib python module to emulate some of the functionality of the email python module.

Here we use TLS for securing the line of smtp.

The project also demonstrated how DNS for the gmail, yahoo and other prominent email service providers, this will prevent the client going to the wrong website in case the attacker has been successful in spoofing DNS.

## **Details of Hardware and Software**

### **Hardware-**

- 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64) processor
- 4 gigabyte (GB) RAM (32-bit) or 2 GB RAM (64-bit)
- 16 GB available hard disk space (32-bit) or 20 GB (64-bit)

### **Software-**

#### **Python**

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation.

#### **OPENPGP**

OpenPGP is an open and free version of the Pretty Good Privacy (PGP) standard that defines encryption formats to enable private messaging abilities for email and other message encryption.

#### **Wireshark-**

Wireshark is an open-source packet analyzer, which is used for education, analysis, software development, communication protocol development, and network troubleshooting.

#### **VSCODE**

Visual Studio Code is a freeware source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

#### **Gmail**

Gmail is a free email service provided by Google. As of 2019, it had 1.5 billion active users worldwide. A user typically accesses Gmail in a web browser or the official mobile app. Google also supports the use of email clients via the POP and IMAP protocols

## Conda environment-

A conda environment is a directory that contains a specific collection of conda packages that you have installed. For example, you may have one environment with NumPy 1.7 and its dependencies, and another environment with NumPy 1.6 for legacy testing. If you change one environment, your other environments are not affected. You can easily activate or deactivate environments, which is how you switch between them. You can also share your environment with someone by giving them a copy of your environment.yaml file. For more information, see Managing environments.

## References : (API format)

### Weblink

1. <https://www.math.utah.edu/~beebe/PGP-notes.html>
2. <https://users.ece.cmu.edu/~adrian/630-f04/PGP-intro.html>
3. [https://www.researchgate.net/publication/332877193\\_Paper\\_on\\_Email\\_Spoofing\\_Analysis](https://www.researchgate.net/publication/332877193_Paper_on_Email_Spoofing_Analysis)
4. <https://searchsecurity.techtarget.com/definition/email-spoofing>

### Journal

1. Dada, E. G., Bassi, J. S., Chiroma, H., Abdulhamid, S. M., Adetunmbi, A. O., & Ajibuwa, O. E. (2019). Machine learning for email spam filtering: review, approaches and open research problems. Heliyon, 5(6), e01802. doi:10.1016/j.heliyon.2019.e01802
2. Wenzheng Zhang, Baodong Qin, Xinfeng Dong, Aikui Tian, Public-key encryption with bidirectional keyword search and its application to encrypted emails, Computer Standards & Interfaces, Volume 78, 2021, 103542, ISSN 0920-5489, <https://doi.org/10.1016/j.csi.2021.103542>.
3. Hongbo Li, Qiong Huang, Jian Shen, Guomin Yang, Willy Susilo, Designated-server identity-based authenticated encryption with keyword search for encrypted emails, Information Sciences, Volume 481, 2019, Pages 330-343, ISSN 0020-0255, <https://doi.org/10.1016/j.ins.2019.01.004>
4. Forensic Analysis of E-mail Date and Time Spoofing Preeti Mishra, Emmanuel S. Pilli and R. C. Joshi Department of Computer Science & Engineering Graphic Era University, Dehradun, India [https://www.researchgate.net/publication/332877193\\_Paper\\_on\\_Email\\_Spoofing\\_Analysis](https://www.researchgate.net/publication/332877193_Paper_on_Email_Spoofing_Analysis)
5. Email Spoofing Analysis Nilay Mistry, Rishiraj Singh Bhati, Heena Jain, Mehul Parmar Institute of Forensic Science Gujarat Forensic Sciences University Gandhinagar, India. [https://www.researchgate.net/publication/286595216\\_Forensic\\_analysis\\_of\\_E-mail\\_address\\_spoofing](https://www.researchgate.net/publication/286595216_Forensic_analysis_of_E-mail_address_spoofing)



## **1. Introduction**

### **1.1. Theoretical Background**

Email encryption is encryption of email messages to protect the content from being read by entities other than the intended recipients. Email encryption may also include authentication. Email encryption can rely on public-key cryptography, in which users can each publish a public key that others can use to encrypt messages to them, while keeping secret a private key they can use to decrypt such messages or to digitally encrypt and sign messages they send.

Email spoofing has been an issue since the earliest days of the SMTP protocol. The root cause of email spoofing is that SMTP does not require authentication between mail relays. An attacker can stand up or find an "Open Relay" (i.e. an SMTP server that can send from arbitrary domains), which is the default configuration for SMTP servers, and use that to send arbitrary emails from arbitrary email addresses.

### **1.2. Motivation**

We made this application to protect the user's data if their email is hacked. Our email application helps protect the email by converting the content of the email to an unreadable gpg file, to read this gpg file we need a secret key that is known only to the sender of the email and the sender has to send this to the receiver via some other secure channel.

### **1.3. Aim of the proposed Work**

To protect the user's data if their email is hacked. any DNS spoofing or DNS cache poisoning and preventing the server to return an incorrect result record or IP Address.

### **1.4. Objectives of the proposed work**

- 1) Open PGP implementation using subprocess
- 2) RSA Based Key Pair Generation using OPENPGP Tool: Private and Public

- 3) Generate a gpg file using subprocess using our app
- 4) GPG file uses RSA and PGP Digital Signature for Authenticity and Integrity
- 5) Packet Capturing and checking whether message is encrypted or not
- 6) Creating a python application which is capable of sending email using SMTPLIB
- 7) Encrypting the channel through which the mail is sent using TLS
- 8) Using Packet Capturing tools to detect whether channel is secure or not
- 9) To create an authentication system which will detect any DNS spoofing or DNS cache poisoning and prevent the server from returning an incorrect result record or IP Address.
- 10) Check if an email can be spoofed or not.

## **2. Literature Survey**

### **2.1. Survey of the Existing Models/Work**

#### **Public-key encryption with bidirectional keyword search and its application to encrypted emails**

A traditional Public-key Encryption scheme with Keyword Search (PEKS) allows multiple senders to encrypt keywords under the public key of a receiver such that the receiver can search on these encrypted keywords using his/her searching secret key. In encrypted email systems, an email user not only needs to search on encrypted emails received from other users, but also needs to search on encrypted emails sent to other users. Motivated by this, the paper proposes a cryptographic method to allow these two types of user (i.e., senders and receivers) to search on encrypted keywords, which is called Public-key Encryption with Bidirectional Keyword Search (PEBKS). We give formal definitions of a PEBKS scheme and its indistinguishable security model to capture the scenario that no adversary can efficiently distinguish two ciphertexts of keywords from each other, even if the adversary can adaptively obtain search trapdoors of many keywords. Specifically, we propose a concrete PEBKS scheme, whose security

relies on a standard hard problem, i.e., bilinear Diffie–Hellman problem, in the random oracle model. Finally, we simulate the proposed PEBKS scheme to assess its practicability and convinces that its feasibility to be applied to encrypted email systems.

### **Designated-server identity-based authenticated encryption with keyword search for encrypted emails**

In an encrypted email system, how to search over encrypted cloud emails without decryption is an important and practical problem. Public key encryption with keyword search (PEKS) is an efficient solution to it. However, PEKS suffers from the complex key management problem in the public key infrastructure. Its variant in the identity-based setting addresses the drawback, however, almost all the schemes does not resist against offline keyword guessing attacks (KGA) by inside adversaries. In this work we introduce the notion of designated-server identity-based authenticated encryption with keyword search (dIBAEKS), in which the email sender authenticates the message while encrypting so that no adversary including the server can launch offline KGA. Furthermore, we strengthen the security requirement so that only the designated server has the capability to search over encrypted emails for receivers. We formally define dIBAEKS and its security models, and propose two dIBAEKS constructions using Type-I and Type-III bilinear pairing, respectively. We compare our schemes with some related IBEKS schemes in the literature, and do experiments to demonstrate its efficiency. Although they are slightly less computationally efficient than but still comparable with the related schemes, our schemes provide stronger security guarantee and better protect users' privacy.

### **Machine learning for email spam filtering: review, approaches and open research problems**

The upsurge in the volume of unwanted emails called spam has created an intense need for the development of more dependable and robust antispam filters. Machine learning methods of recent are being used to successfully detect and filter spam emails. We present a systematic review of some of the popular machine learning based email spam filtering approaches. Our review covers survey of the important concepts, attempts,

efficiency, and the research trend in spam filtering. The preliminary discussion in the study background examines the applications of machine learning techniques to the email spam filtering process of the leading internet service providers (ISPs) like Gmail, Yahoo and Outlook emails spam filters. Discussion on general email spam filtering process, and the various efforts by different researchers in combating spam through the use of machine learning techniques was done. Our review compares the strengths and drawbacks of existing machine learning approaches and the open research problems in spam filtering. We recommended deep learning and deep adversarial learning as the future techniques that can effectively handle the menace of spam emails.

## **2.2. Summary/Gaps identified in the Survey**

### **Designated-server identity-based authenticated encryption with keyword search for encrypted emails**

We introduced the notion of designated-server identity-based authenticated encryption with keyword search, and proposed a concrete dIBAEKS scheme. We proved it to be secure against inside offline KGA and achieve designated testability based on a simple number-theoretic assumption. The scheme could be slightly modified to satisfy the CCA-type designated testability. We also showed how to modify the scheme to work in asymmetric bilinear pairing settings to improve the efficiency. dIBAEKS can be well applied to the encrypted email system to protect users' privacy. However, our dIBAEKS and dIBAEKS-3 schemes have the property that a trapdoor can only be used to search over ciphertexts sent from a specific sender, as both ciphertexts and trapdoors are binded with the identity of the sender (and that of the receiver). We are considering constructing a more flexible dIBAEKS scheme in which a trapdoor can be used to search over multiple users' encrypted data in the future.

There is a rapid increase in the interest being shown by the global research community on email spam filtering. In this section, we present similar reviews that have been presented in the literature in this domain. This method is followed so as to articulate the issues that are yet to be addressed and to highlight the differences with our current review. Lueg presented a brief survey to explore the gaps in whether information

filtering and information retrieval technology can be applied to postulate Email spam detection in a logical, theoretically grounded manner, in order to facilitate the introduction of spam filtering technique.

### **Machine learning for email spam filtering: review, approaches and open research problems**

The scope of the research work is too narrow; almost proffering nothing. The research work, together with its implemented algorithm does not make any significant contribution. Dhanaraj and Palaniswami [105] applied the combination of Firefly algorithm and Naïve Bayes for email classification in a distributed environment using the CSDMC2010 spam corpus dataset. Firefly algorithm was used to optimise and select the feature space with the best fitness. The spam classification was done with the Naïve Bayes classifier. The proposed method was not an improvement over existing methods in terms of specificity. The performance of the proposed method was not evaluated using other performance metrics that can prove the effectiveness of their approach. Choudhary and Dhaka [106] applied the Genetic Algorithm for automatic classification of emails. The algorithm was able to successfully differentiate between spam and ham emails. The proposed method was not evaluated in terms of common email classification metrics like accuracy, precision, recall, and computation time. Hence, its performance in comparison to other methods cannot be rated. Other limitations peculiar to GA algorithm and its variants also applies to the proposed method.

### **Public-key encryption with bidirectional keyword search and its application to encrypted emails**

We defined the concept of a public key encryption with keyword search (PEKS) and gave two constructions. Constructing a PEKS is related to Identity Based Encryption (IBE), though PEKS seems to be harder to construct. We showed that PEKS implies Identity Based Encryption, but the converse is currently an open problem. Our constructions for PEKS are based on recent IBE constructions. We are able to prove security by exploiting extra properties of these schemes.

### **3. Overview of the Proposed System**

#### **3.1. Introduction and Related Concepts**

##### **PGP-**

- PGP is an encryption program that provides cryptographic privacy and authentication for data communication. PGP is used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications.
- For encrypting email, we will use a custom PGP application. With the help of this we can encrypt and decrypt mail.
- With the help of our tool first we generate our key pair-Private Key and
- Public Key. Then we will share our public key with people who want to send mail to us.
- After receiving the mail from others, we will try to decrypt using our private key.
- This will help us to protect our mail from any attacker who try to see our mail

##### **Email Spoofing Prevention**

- DNS is a TCP/IP protocol used on different platforms. The domain name space is divided into three different sections: generic domains, country domains, and inverse domain.
- DNS cache poisoning, also known as DNS spoofing, is a type of attack that exploits vulnerabilities in the domain name system (DNS) to divert Internet traffic away from legitimate servers and towards fake ones.
- DNS cache poisoning is the act of entering false information into a DNS cache, so that DNS queries return an incorrect response and users are directed to the wrong websites.
- Assuming an entity needs to spoof the identity of some. Other entity, it's enough to vary the mapping between its low level address and its high level name. It means that an assailant will pretend the name of somebody by modifying the

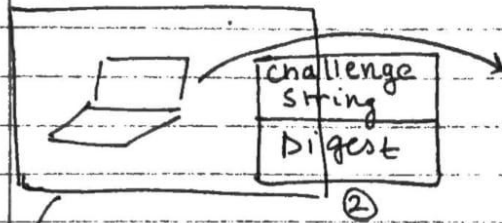
association of his address from his own name to the name he needs to impersonate.

- Once an assailant has done that, an appraiser will no longer distinguish between truth and fake entity ex- >google.com and gogle.com. DNS is a TCP/IP protocol used on different platforms. The domain name space is divided into three different sections: generic domains, country domains, and inverse domain.

### **3.2. Module for the Proposed System**

Client.py

A's.py



③ Recieves challenge string and hmac  
④ Only sends challenge strings to both TLDS servers

① Creates a Digest of challenge string from the keys  $\{k_1, k_2 \dots k_n\}$

Challenge String.

TLDS2.py (key =  $k_1$ )

TLDS1.py (key =  $k_2$ )

mihir.sri10@gmail.com  
laksh@outlook.com

AM@hotmail.com  
Ruhakhiya@gmail.com

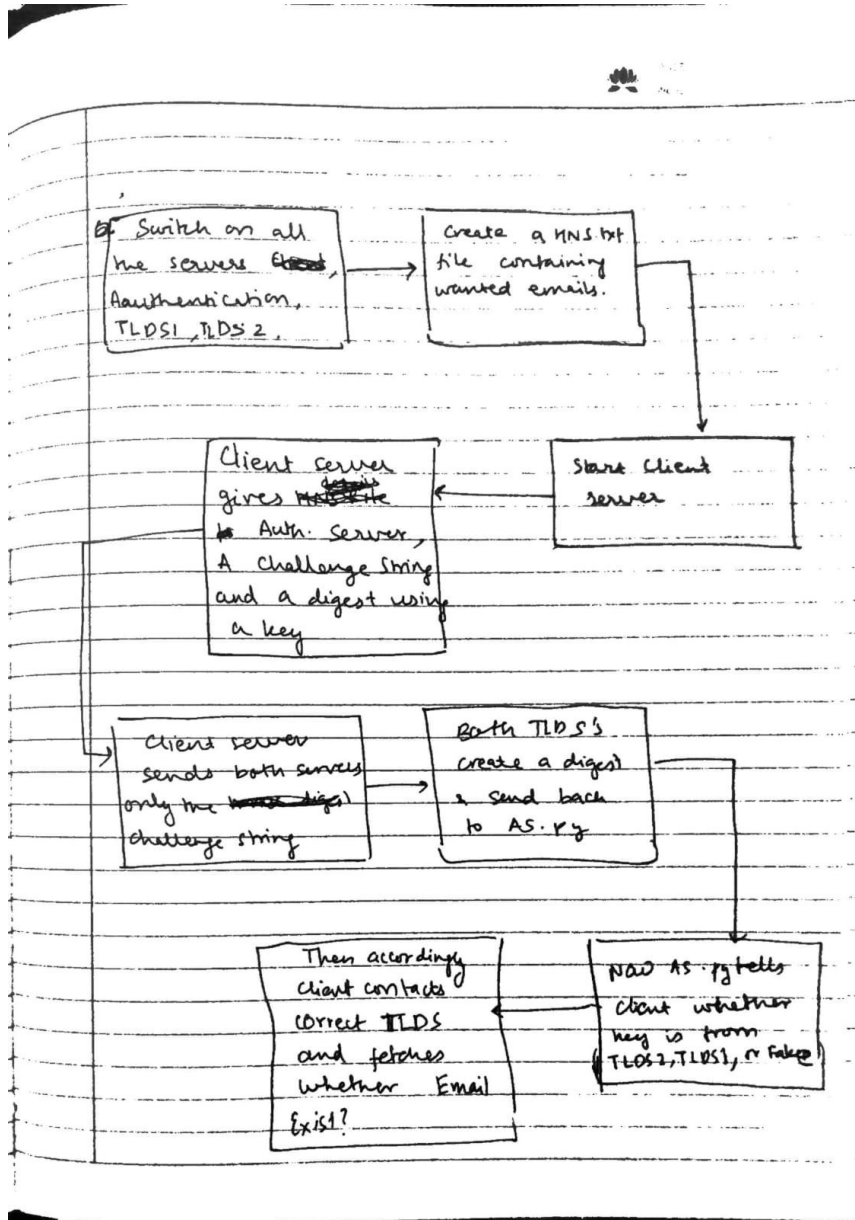
key =  $k_1$

key =  $k_2$

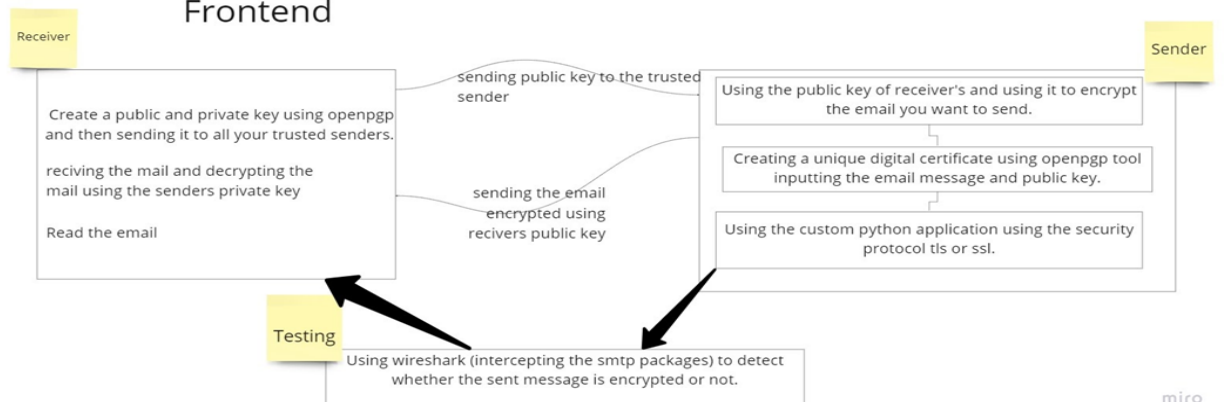
Creates a digest and sends back for auth.

Creates a digest and send back for auth

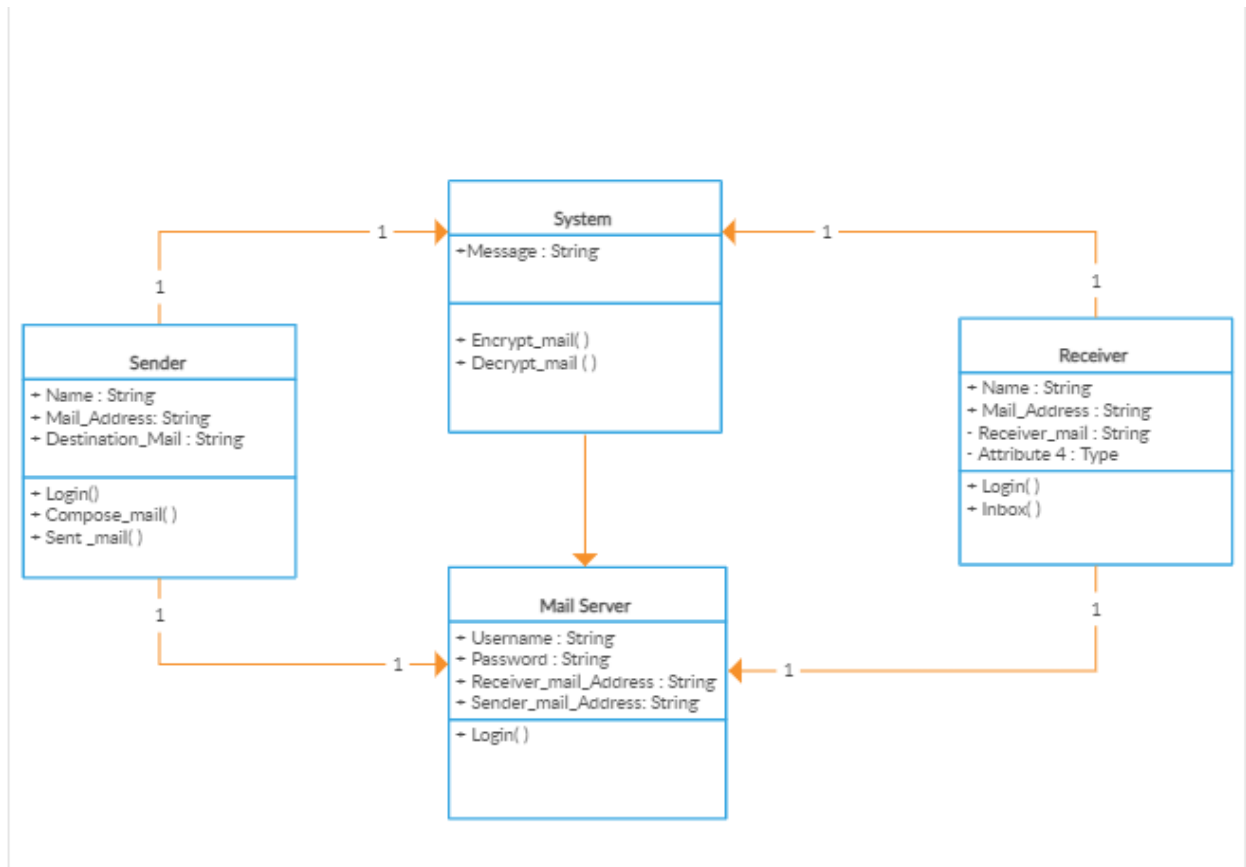




## Frontend



### 3.3. Proposed System Model



Email Security UML

## 4. Proposed System Analysis and Design

### 4.1. Introduction

#### Pretty Good Privacy (PGP)

- is a program that converts text to pgp file provides cryptographic privacy and authentication for data communication. PGP is used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications.
- For exporting pgp files through email we will use our pgp code. It is designed especially for functions like emailing

- With the help of this first we generate our key pair and a GPG File using message = subprocess.check\_output(["gpg", "--armor", "--export", email, ">", file])
- Then we will share our File with people who want to send mail to us.
- After receiving the mail from others we will try to open the GPG File using our custom python application using subprocess using subprocess.call(["gpg", "--output", name\_file, "--decrypt", encrypted\_file]) .
- This will help us to protect our mail from any attacker who tries to see our mail.

### **Custom email python app**

- Python console emailer is a console (terminal) based python application that allows the user to sign in to their email and generate a PGP pair. The application allows the user to insert a morphed text file in the email via the command line (terminal).
- The purpose of this project is to gain a greater understanding of how emails are created, encrypted, and sent. The application can be developed with the smtplib python module to emulate some of the functionality of the email python module.

The python app also allows the user to convert txt file to gpg files and vice versa.

```
server = smtplib.SMTP("smtp.gmail.com", 587)
server.starttls()
server.login(sender, password)
server.sendmail(sender, receiver, msg)
server.quit()
print("Mail sent successfully")
```

### **Email Spoofing prevention-**

#### **Mail Server**

The DNS client program uses a key and a challenge string to create a digest and sends the challenge string as well as the digest to the authentication server. The authentication sever, sends ONLY the challenge string to both the TLDS servers and the TLDS servers return the respective digests. The TLDS servers each store a key and only of them matches the key that the client used to create the digest. Hence, only one of the responses from the TLDS servers match the digest sent by the client. The Authentication server (root server) sends the

hostname of the TLDS server with the correct match to the client. The DNS client then connects to that TLDS server with a query string and obtains the A record (if found) from the authenticated TLDS server. The two TS servers each maintain a DNS\_table consisting of three fields: Hostname, IP address, Flag (A). In addition, the TS servers each maintain a key which is used to create a digest from a challenge. When the DNS client connects to the authenticated TLDS server, it sends the hostname as a string. The TLDS server does a look up in the DNS\_table and if there is a match, sends the DNS table entry as a string ["Hostname IPAddress A"]. If the host name does not exist, then an error string Error: HOST NOT FOUND is returned. Note, that in this Project, the DNS client connects to the TLDS server to get the IP address for a given hostname. The hostname string, the Key and the challenge will be given one per line in a file (PROJ3-HNS.txt) and the keys one per line will be in a file PROJ3-KEY1.txt and PROJ3-KEY2.txt. The DNS tables entries will also be one per line and will be in PROJ3- TLDS1.txt and PROJ3-TLDS2.txt. The TLDS servers, in addition to reading the DNS entries, will each obtain its key by reading the value from the corresponding key files. TLDS1 will read the key from PROJ3-KEY1.txt and TLDS2 will read the key from PROJ3-KEY2.txt Your client program should output the results to a file RESOLVED.txt. As part of your submission, you need to submit four program files as well as the output file.

### **Basic Principles**

Email spoofing has been an issue since the earliest days of the SMTP protocol. The root cause of email spoofing is that SMTP does not require authentication between mail relays. An attacker can stand up or find an "Open Relay" (i.e. an SMTP server that can send from arbitrary domains), which is the default configuration for SMTP servers, and use that to send arbitrary emails from arbitrary email addresses.

### **FQDN Requirements**

In an effort to combat spam, many SMTP servers now block any mail relay that does not have a Fully-qualified Domain Name (FQDN). An FQDN is a DNS A record that points to the relay's IP address. This can be either a domain purchased from a domain registrar, or by using a domain automatically associated with a virtual private server.

## **Email Protections**

As email spoofing is a serious and widespread issue, over the years several protection mechanisms have been added to combat it. However, all of these protections are opt-in and require significant configuration. As such, as much as 98% of the internet is still vulnerable.

### **4.2. Requirement Analysis**

#### **4.2.1. Functional Requirements**

##### **4.2.1.1. Product Perspective**

Our email app helps protect the email by converting the content of the email to an unreadable gpg file, to read this gpg file we need a secret key that is known only to the sender of the email and the sender has to send this to the receiver via some other secure channel.

##### **4.2.1.2. Product features**

Our product can be used to encode an email to a gpg file, send the gpg file via email through command line, it can also get the sent gpg file and decode it to get the plain text back again. The product can also verify domains and IP addresses of popular email providers.

##### **4.2.1.3. User characteristics**

The user would be someone who knows the basics of email security, DNS and common line interface in both linux and windows.

##### **4.2.1.4. Assumption & Dependencies**

The project is done in python and there are multiple libraries and modules required to run it, some of these are: PGP, subprocess, argparse, re, getpass, smtplib, ssl, emailprotectionslib, logging, email etc.

The product also needs a linux environment and a text editor(like VS code to run). Also we need anaconda environment installed.

#### **4.2.1.5. Domain Requirements**

The project can be run on only linux systems with python modules, pip and anaconda navigator installed for Email encryption. As well as a windows environment for DNS part of the project.

#### **4.2.1.6. User Requirements**

The user can expect the product to be able to encode mail, send the encoded file, decode the received encoded file, spoof an email, check whether an email ID can be spoofed.

### **4.2.2. Non Functional Requirements**

#### **4.2.2.1. Product Requirements**

##### **4.2.2.1.1. Efficiency**

The product should be able to encode/decode mails instantly and should be able to send it within 3 seconds. It should also instantaneously verify the DNS provided by matching it with the already saved DNS.

##### **4.2.2.1.2. Reliability**

The product should be able to work for any given email ID and should be and should work on all linux environments without any bugs or crashes.

##### **4.2.2.1.3. Portability**

The product will work in both windows environments and any ubuntu/Debian operating system. I.e ubuntu, linux mint, kali linux etc.

##### **4.2.2.1.4. Usability**

The product doesn't have a Graphical user interface and runs only through the command line, so the user should know the basics of the command line.

#### **4.2.2.2. Organizational Requirements**

##### **4.2.2.2.1. Implementation Requirements (in terms of deployment)**

The product is not deployed, it can be downloaded and used by anyone, if they have the necessary modules and environment available.

##### **4.2.2.2.2. Engineering Standard Requirements**

The code follows standard modular design for python.

#### **4.2.2.3. Operational Requirements**

- **Economic**

The product has no economic operational requirements, as it is free to use.

- **Environmental**

The product requires a linux system to run the email security part and DNS part will work on both windows and linux.

- **Social**

There are no social implications of this project.

- **Political**

There are no political operational requirements for the project.

- **Ethical**

There are no specific ethical requirements for this project, this can however be used to spam emails that can be unethical.

- **Health and Safety**

This is a software only project and does not have any health or safety operational requirement.

- **Sustainability**

The product can stop working if there are some policy changes done by the email service provider( gmail).

- **Legality**

Sending spoofed emails with malicious intent can be considered impersonating and the sender can be prosecuted.

- **Inspectability**

The product is open source and the source code can be inspected by anyone, anytime.

#### **4.2.3. System Requirements**

##### **4.2.3.1. H/W Requirements**

##### **Recommended System Requirements**

Processors: Intel® Core™ i5 processor 4300M or up. 8GB ram, 2-3 GB free disk space.

##### **4.2.3.2. S/W Requirements**

Operating systems: Linux (all debian based for email security part) and windows 10 (DNS part)



Python v3.6 +, *pip* package installer for Python, conda environment, anaconda navigator, multiple python libraries and modules.

Some code editor with inbuilt terminal (VS code, Sublime text editor etc.)

## **5. Results and Discussion**

In the open source Openpgp application with the help of a tool first we generate our PGP pair for the email ID.

Then we will share our puk with people who want to send mail to us.

After receiving the mail from others we will try to open the gpg file using our prk.

This will help us to protect our mail from any attacker who try to see our mail.

On the other hand the purpose of this custom email project is to gain a greater understanding of how emails are created, secured, and sent. The application can be developed with the smtplib python module to emulate some of the functionality of the email python module.

Here we use TLS for securing the line of smtp.

The project also demonstrated how DNS for the gmail, yahoo and other prominent email service providers, this will prevent the client going to the wrong website in case the attacker has been successful in spoofing DNS.

```
lakhsh@lakhsh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$ python user_interface.py
[+] Select your choice!
-----
[1] Generate PGP
[2] Conver txt to GPG
[3] Conver GPG to txt
[4] Export GPG file via email
[5] Quit
-----
1
gpg (GnuPG) 2.2.19; Copyright (C) 2019 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
  (1) RSA and RSA (default)
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 3072
Requested keysize is 3072 bits
Please specify how long the key should be valid.
    0 = key does not expire
    <n> = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 1
Key expires at Thu 20 May 2021 10:20:29 AM +0530
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

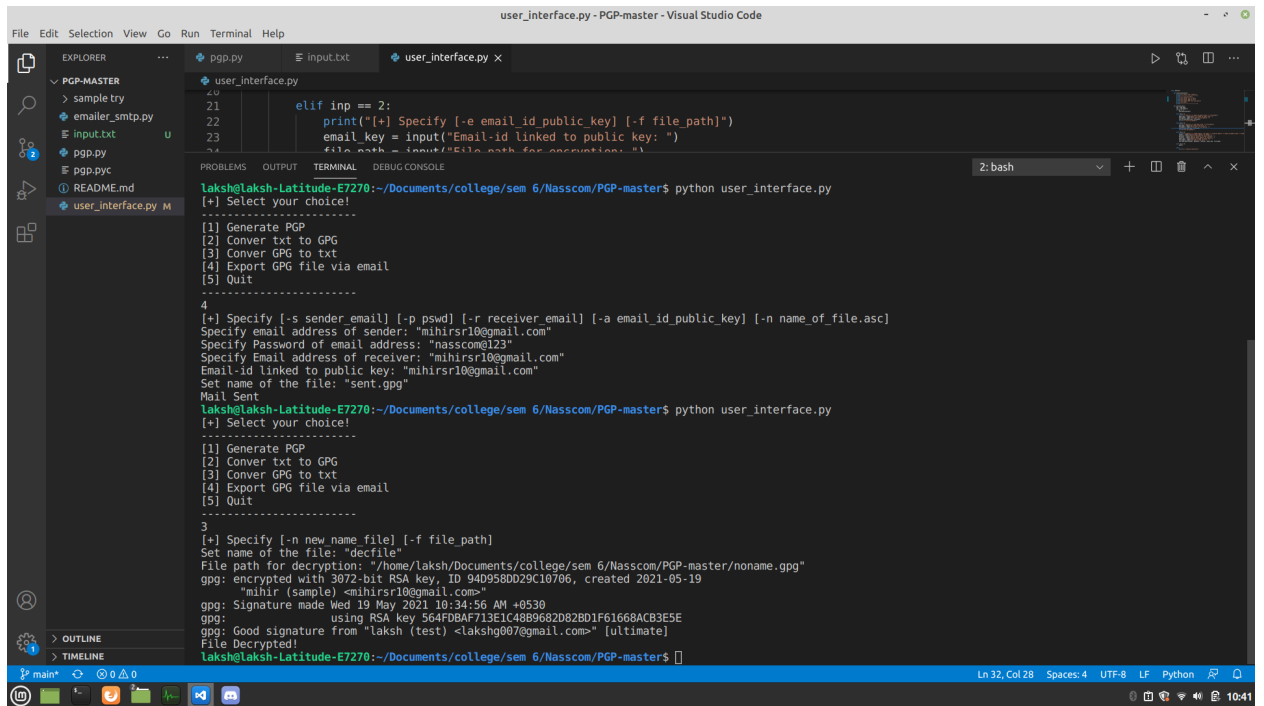
Real name: mihir
Email address: mihirsr10@gmail.com
Comment: test
You selected this USER-ID:
    "mihir (test) <mihirsr10@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? 0
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
```

## Generating PGP public private key pair

```
lakhsh@lakhsh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$ python user_interface.py
[+] Select your choice!
-----
[1] Generate PGP
[2] Conver txt to GPG
[3] Conver GPG to txt
[4] Export GPG file via email
[5] Quit
-----
2
[+] Specify [-e email_id_public_key] [-f file_path]
Email-id linked to public key: "mihirsr10@gmail.com"
File path for encryption: "/home/lakhsh/Documents/college/sem 6/Nasscom/PGP-master/sent"
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 8 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 8u
gpg: next trustdb check due at 2021-05-19
[+] File encrypted:
lakhsh@lakhsh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$ python user_interface.py
[+] Select your choice!
-----
[1] Generate PGP
[2] Conver txt to GPG
[3] Conver GPG to txt
[4] Export GPG file via email
[5] Quit
-----
4
[+] Specify [-s sender_email] [-p pgwd] [-r receiver_email] [-a email_id_public_key] [-n name_of_file.asc]
Specify email address of sender: "mihirsr10@gmail.com"
Specify Password of email address: "nasscom@123"
Specify Email address of receiver: "mihirsr10@gmail.com"
Email-id linked to public key: "mihirsr10@gmail.com"
Set name of the file: "sent.gpg"
Mail Sent
lakhsh@lakhsh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$ python user_interface.py
[+] Select your choice!
```

## Converting txt file to gpg file



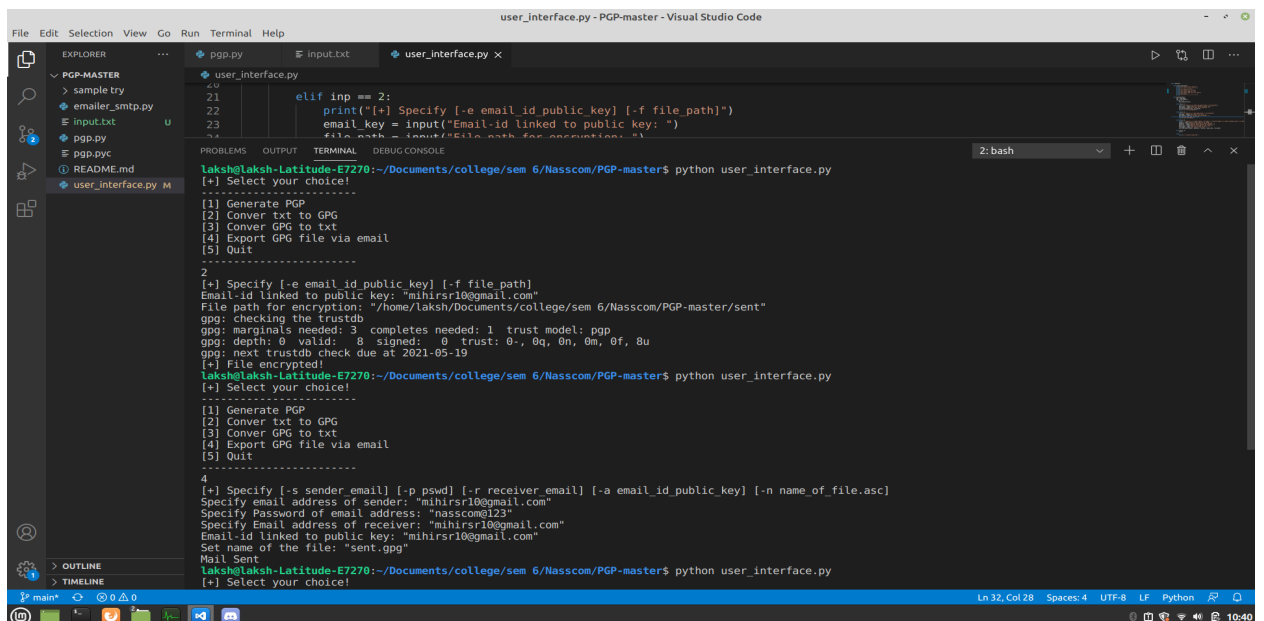
```
user_interface.py - PGP-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
PGP-MASTER
  sample try
  emailer_smtp.py
  input.txt
  pgp.py
  pgp.pyc
  README.md
  user_interface.py M

user_interface.py
21 elif inp == 2:
22     print("[+] Specify [-e email_id_public_key] [-f file_path]")
23     email_key = input("Email-id linked to public key: ")
24     file_path = input("File path for decryption: ")

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
2: bash
[+] Select your choice!
-----
[1] Generate PGP
[2] Conver txt to GPG
[3] Conver GPG to txt
[4] Export GPG file via email
[5] Quit
-----
4
[+] Specify [-s sender_email] [-p pswd] [-r receiver_email] [-a email_id_public_key] [-n name_of_file.asc]
Specify email address of sender: "mihirsr10@gmail.com"
Specify Password of email address: "nasscom@123"
Specify Email address of receiver: "mihirsr10@gmail.com"
Email-id linked to public key: "mihirsr10@gmail.com"
Set name of the file: "sent.gpg"
Mail Sent
laksh@laksh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$ python user_interface.py
[+] Select your choice!
-----
[1] Generate PGP
[2] Conver txt to GPG
[3] Conver GPG to txt
[4] Export GPG file via email
[5] Quit
-----
3
[+] Specify [-n new_name_file] [-f file_path]
Set name of the file: "decfile"
File path for decryption: "/home/laksh/Documents/college/sem 6/Nasscom/PGP-master/noname.gpg"
gpg: encrypted with 3072-bit RSA key, ID 94D958D029C10706, created 2021-05-19
      "mihir (sample) <mihirsr10@gmail.com>"
gpg: Signature made Wed 10 May 2021 10:34:56 AM +0530
gpg:                using RSA key 564FD8AF713E1C48B9682D82B01F61668ACB3E5E
gpg: Good signature from "laksh (test) <lakshg007@gmail.com>" [ultimate]
File Decrypted!
laksh@laksh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$
```

Converting gpg file back to txt file after downloading the file form the received email



```
user_interface.py - PGP-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
PGP-MASTER
  sample try
  emailer_smtp.py
  input.txt
  pgp.py
  pgp.pyc
  README.md
  user_interface.py M

user_interface.py
21 elif inp == 2:
22     print("[+] Specify [-e email_id_public_key] [-f file_path]")
23     email_key = input("Email-id linked to public key: ")
24     file_path = input("File path for decryption: ")

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
2: bash
[+] Select your choice!
-----
[1] Generate PGP
[2] Conver txt to GPG
[3] Conver GPG to txt
[4] Export GPG file via email
[5] Quit
-----
2
[+] Specify [-e email_id_public_key] [-f file_path]
Email-id linked to public key: "mihirsr10@gmail.com"
File path for decryption: "/home/laksh/Documents/college/sem 6/Nasscom/PGP-master/sent"
gpg: checking the trustdb
gpg: marginal: needed: 3, completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 8 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 0u
gpg: next trustdb check due at 2021-05-19
laksh@laksh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$ python user_interface.py
[+] Select your choice!
-----
[1] Generate PGP
[2] Conver txt to GPG
[3] Conver GPG to txt
[4] Export GPG file via email
[5] Quit
-----
4
[+] Specify [-s sender_email] [-p pswd] [-r receiver_email] [-a email_id_public_key] [-n name_of_file.asc]
Specify email address of sender: "mihirsr10@gmail.com"
Specify Password of email address: "nasscom@123"
Specify Email address of receiver: "mihirsr10@gmail.com"
Email-id linked to public key: "mihirsr10@gmail.com"
Set name of the file: "sent.gpg"
Mail Sent
laksh@laksh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$ python user_interface.py
[+] Select your choice!
-----
[1] Generate PGP
[2] Conver txt to GPG
[3] Conver GPG to txt
[4] Export GPG file via email
[5] Quit
-----
2
[+] Specify [-e email_id_public_key] [-f file_path]
Email-id linked to public key: "mihirsr10@gmail.com"
File path for decryption: "/home/laksh/Documents/college/sem 6/Nasscom/PGP-master/sent"
gpg: checking the trustdb
gpg: marginal: needed: 3, completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 8 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 0u
gpg: next trustdb check due at 2021-05-19
laksh@laksh-Latitude-E7270:~/Documents/college/sem 6/Nasscom/PGP-master$
```

Using SMTPLIB to send GPG to receiver

```
C:\Windows\System32\cmd.exe
('[S]: Got a connection request from a client at', ('192.168.56.1', 4153))
('[S]: Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to as.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4156))

C:\Users\mihir\Desktop\RANDO\IT-Phase3-master>python2 tlds2.py
('[S]: Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to as.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4450))
('[S]: Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to as.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4453))

C:\Users\mihir\Desktop\RANDO\IT-Phase3-master>
```

TLDS2 server

```
C:\Windows\System32\cmd.exe
('[S]: Got a connection request from a client at', ('192.168.56.1', 1384))
('[S]: Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to as.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 1387))

C:\Users\mihir\Desktop\RANDO\IT-Phase3-master>python2 tlds1.py
('[S]: Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to as.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4152))
('[S]: Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to as.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4155))

C:\Users\mihir\Desktop\RANDO\IT-Phase3-master>python2 tlds1.py
('[S]: Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to as.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4449))
('[S]: Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to as.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4452))

C:\Users\mihir\Desktop\RANDO\IT-Phase3-master>
```

TLDS1 server

```
C:\Windows\System32\cmd.exe
C:\Users\mihir\Desktop\RANDO\IT-Phase3-master>python2 as.py
('[S]: Authentication Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to client.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4154))

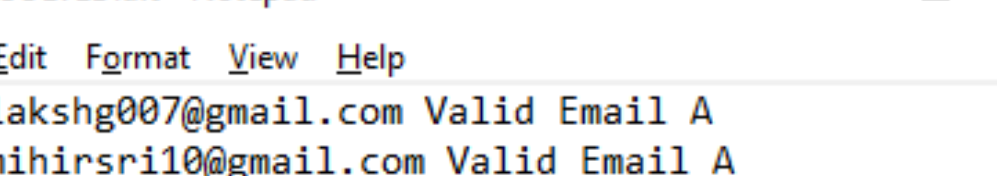
C:\Users\mihir\Desktop\RANDO\IT-Phase3-master>python2 as.py
('[S]: Authentication Server host name is: ', 'DESKTOP-KN9P2UD')
('[S]: Attempting to connect to client.\n[S]: Server IP address is ', '192.168.56.1')
('[S]: Got a connection request from a client at', ('192.168.56.1', 4451))

C:\Users\mihir\Desktop\RANDO\IT-Phase3-master>_
```

AS.py server

```
MIHIR-HNS.txt - Notepad
File Edit Format View Help
k3521 ramirez www.lakshg007@gmail.com
k6854 fedora www.mihirsri10@gmail.com
k6854 wales www.mit@fefooi.com
k3521 roberts www.shouryam@yahoo.co.in
k3 roberts www.shouryam@yahoo.co.in
k3521 arya www.fwmpowjpfw@gmail.co.in
Ln 4, Col 2 100% Unix (LF) UTF-8
```

HNS TEXT FILE



RESOLVED.txt - Notepad

File Edit Format View Help

www.lakshg007@gmail.com Valid Email A  
www.mihirsri10@gmail.com Valid Email A  
www.mit@fefoo.com - Error : NOT FOUND  
www.shouryam@yahoo.co.in Valid Email A  
www.shouryam@yahoo.co.in - No servers matching key  
www.fwmpowjpfw@gmail.co.in - Error:HOST NOT FOUND

Ln 1, Col 1 100% Windows (CRLF) UTF-8

## Result

## Implementation Video

<https://drive.google.com/file/d/1lSaRfUUzrW0ZeHKrn4lqiFPU2pr7TrtA/view?usp=sharing>

## 6. References

1. <https://www.math.utah.edu/~beebe/PGP-notes.html>
2. <https://users.ece.cmu.edu/~adrian/630-f04/PGP-intro.html>
3. [https://www.researchgate.net/publication/332877193\\_Paper\\_on\\_Email\\_Spoofing\\_Analysis](https://www.researchgate.net/publication/332877193_Paper_on_Email_Spoofing_Analysis)
4. <https://searchsecurity.techtarget.com/definition/email-spoofing>
5. Dada, E. G., Bassi, J. S., Chiroma, H., Abdulhamid, S. M., Adetunmbi, A. O., & Ajibuwa, O. E. (2019). Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, 5(6), e01802. doi:10.1016/j.heliyon.2019.e01802
6. Wenzheng Zhang, Baodong Qin, Xinfeng Dong, Aikui Tian, Public-key encryption with bidirectional keyword search and its application to encrypted emails, *Computer Standards*

& Interfaces, Volume 78, 2021, 103542, ISSN 0920-5489,  
<https://doi.org/10.1016/j.csi.2021.103542>.

7. Hongbo Li, Qiong Huang, Jian Shen, Guomin Yang, Willy Susilo, Designated-server identity-based authenticated encryption with keyword search for encrypted emails, Information Sciences, Volume 481, 2019, Pages 330-343, ISSN 0020-0255,  
<https://doi.org/10.1016/j.ins.2019.01.004>
8. Forensic Analysis of E-mail Date and Time Spoofing Preeti Mishra, Emmanuel S. Pilli and R. C. Joshi Department of Computer Science & Engineering Graphic Era University, Dehradun, India  
[https://www.researchgate.net/publication/332877193\\_Paper\\_on\\_Email\\_Spoofing\\_Analysis](https://www.researchgate.net/publication/332877193_Paper_on_Email_Spoofing_Analysis)
9. Email Spoofing Analysis Nilay Mistry, Rishiraj Singh Bhati, Heena Jain, Mehul Parmar Institute of Forensic Science Gujarat Forensic Sciences University Gandhinagar, India  
[https://www.researchgate.net/publication/286595216\\_Forensic\\_analysis\\_of\\_E-mail\\_address\\_spoofing](https://www.researchgate.net/publication/286595216_Forensic_analysis_of_E-mail_address_spoofing)
10. Implementation Video:  
<https://drive.google.com/file/d/1lSaRfUUzrW0ZeHKrn4lqiFPU2pr7TrtA/view?usp=sharing>