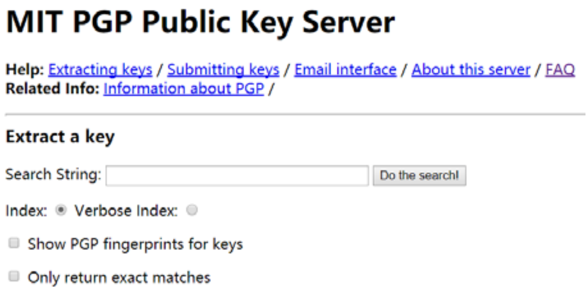
# Project I: E-mail and GPG based Secured Instant Message Application

1. Description
   In this project, you are required to use the existing e-mail services (Gmail, Outlook, etc.) via IMAP to implement a desktop IM application.
2. Instruction
   Users of your application only need to set the e-mail services to start. **You are NOT allowed to design a user system (username, password for login) by your own**, except for the locking password* for your client.
   As we have learnt before, the content of e-mail isn't secure
   a) asymmetrical cryptography and digital signature
      i. asymmetrical cryptography: RSA, for example.
      ii. digital signature: using asymmetrical cryptography and crypto hash function to ensure the authentication of digital messages or documents.
      You should have learnt it in the *Discrete Math*. Ask Prof. Wang if you forget it.
   b) GPG
      Here's a brief introduction of GPG. Read it if you don't know GPG. **We highly recommend you go through GPG to ensure you fully understand how GPG works.**We also recommend you usepgp.mit.edu as your GPG key server.



**MIT PGP Public Key Server**

**Help:** Extracting keys / Submitting keys / Email interface / About this server / FAQ
**Related Info:** Information about PGP /

**Extract a key**

Search String: [ ] Do the search!

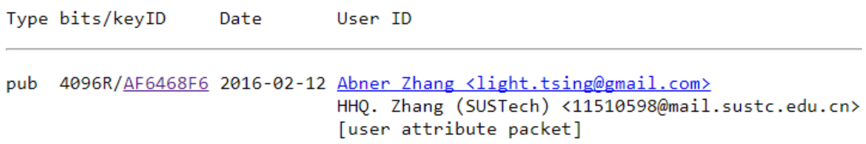
Index: ● Verbose Index: ○

□ Show PGP fingerprints for keys

□ Only return exact matches

**You should use GPG to secure our chatting.** While two clients talk with each other, any of them should encrypt the message/text using the other's GPG public key. Also, any received message can be decrypted using their private key**.

GPG User ID usually is an email address with name and comment.
Organized as: Name (comment) <email address>
Example: Bob(Crypto Example) <bob@gmail.com>
GPG keyID is a hash of the public key which is unique for each user.



```
Type bits/keyID     Date       User ID
---------------------------------------------------------------------------
pub  4096R/AF6468F6 2016-02-12 Abner Zhang <light.tsing@gmail.com>
                               HHQ. Zhang (SUSTech) <11510598@mail.sustc.edu.cn>
                               [user attribute packet]
```

*locking password: using to lock the app when user isn't using the app. This can prevent other people from explore your chat history without permission. This can also be the encrypting password of the local data storage.

****Be careful** when you are dealing with the private key!!!! Private key **SHOULD NEVER** store without encryption. Using the locking password or design another mechanism to protect the private key. An example workflow of your app: unlock the app when it starts with the password, app using the password to decrypt the main database which contains the private key. Then load the private key into memory for further usage.

Then, our app's contacts system should base on email address. When you add a contact, you should ask user for the email address or the keyID. Then, trying to retrieve the key from key server, display it to user to confirm add contact.

Trust contact is an optional feature. When you totally confirm a GPG key is belongs to a user, you can generate a sign for user and add it to the public key of the user to tell other people that you had already checked the key.

Group chat is another optional feature. GPG encryption allows you to choose multiple user who can decrypt the message.

3. Requirements

   You need to implement all basic features which means that you need to design the chatting protocol.

   a) GUI
   b) Contacts system based on GPG
      i. Add contact
      ii. Delete contact
      iii. Block contact
      iv. Trust contact (optional)
   c) Chat secured by GPG
      i. Text chatting
      ii. Photo (optional)
      iii. File (optional)
   d) Group (optional)
      i. Group creating/deleting
      ii. Group admin
      iii. Group user management
      iv. Group chatting

4. Protocol Definition

   a) Payload Definition

      Payload is the GPG encrypted content in the email message part.

      i. Header

         There can be multiple areas in the header part. In the standard protocol, there's two

         The header organized as key-value strings with "\r\n" as separator. The header part end with a new line "\r\n".

         Here's an Example:

         ```
         UserAgent: AliceChat/2.7.3 (Windows NT 10.0; Win64; x64) likes BobChat/1.3.2\r\n
         Sequence: 1012\r\n
         \r\n
         ```

         1. UserAgent

            User Agent stands for the chat client of this protocol. With the user agent string in the header part, your client can add some features that's not supported in the standard protocol.

            UA organized as <client name>/<version> (environment) <opt client name>...

            a) Test of User Agent: Test if the user agent string contains or higher than your client’s. If so, then you should treat it as a compatible client.

b) Example: Your client’s UA is “BobChat/1.0.1”. Another client’s UA is “AliceChat/2.7.3 (Windows NT 10.0; Win64; x64) likesBobChat/1.3.2”. Then you should treat AliceChat/2.7.3 as a compatible client which means Alice’s Chat has all extra features of BobChat/1.0.1

c) Version Number: Your version number should obey the rules of Semantic Versioning. In the above example, the 1.3.2 version is higher than 1.0.1 while is compatible with it.

2. Sequence

Sequence Number is unique to each message of each user in each chat and start from 1.

For example, Bob and Alice are chatting. The first message from Bob has seq. of 1. The second message from Bob to Alice is 2. Then Alice’s first message to Bob is 1.

ii. Body

Body is the content of the chat message. Define it with your client’s features.

b) One to One Chat

One to one chat means chatting between two users (a.k.a. email address).

The subject of the email should be the chat id. The chat id is a uuid version 4 string.

Here are some references.

https://docs.python.org/3/library/uuid.html

https://www.uuidgenerator.net/version4

On creating the chat, sending an email with a new uuid as the subject.

c) Group Chat

Group chatting is not included in the standard protocol.

5. Tips

You can use any programming language you love (except for Python2).

a) GUI

Java FX for Java

TKinter, PyQt for Python

Qt for C++

or Electron with Node.js

b) How can you do symmetrical crypto based on password?

There’s some function call key derivation function (kdf). For example, you can use PBKDF2 or scrypt as your key derivation function which can map the password to a fixed length bytes. Then you can use these bytes as your key for symmetrical ciphers (for example, AES-256). (**DO NOT use single hash function as your kdf.**)

c) We might need to register new e-mail account for this project

6. Submission

a) A technical report including all the detailed protocol design and system implementation.

b) A presentation slides

c) Source codes including necessary comments and readme files.