Anirudh Parappil Menon
UPI: amen076
UID: 611800390

# Pretty Good Privacy

## Introduction

Nowadays emails are important in our day to day life, normally we use emails for our official purposes. In the current scenario, emails are not secured because there is no end to end encryption used. For instance, Google is one of the top email service providers they use encryption between the client and their server. However, the emails can be viewed by Google in its server end. In my opinion, our email data are really important and it needs to be secured. Edward Snowden who was a computer programmer in National Security agency, he was someone wants to share sensitive information on the internet. So, he used PGP for private email communication with journalists ("Edward Snowden", n.d.).

In 1991, Phil Zimmerman and Associates developed Pretty Good Privacy and was used to secure the data posted in the BBS(Bulletin Board Systems). PGP was now acquired by Symantec Corporation. In 1997, Phil Zimmerman was worked on Open-source PGP version that had no licensing issues and it was accepted by Internet Engineering Task Force(IETF). OpenPGP is a standard which can be used by any program that aids OpenPGP software. GNU Privacy guard was developed by Werner Koch in 1999, which was another option for Symantec version of PGP. It is freely available software based on OpenPGP standards and it is interoperable with Symantec PGP ("OpenPGP, PGP, and GPG: What is the difference?", n.d.).

## PGP overview and growth factors

PGP is a software used for email security and file storage applications which allow users to achieve privacy and authentication. It uses a mixture of symmetric encryption, public key cryptography, and data compression. There number of reasons why PGP is now widely used. Firstly, It is accessible openly worldwide in versions that run on several platforms like Windows, UNIX, Macintosh etc. Moreover, the enterprise version provides vendor support. Secondly, PGP uses best available cryptographic algorithms which are considered as secure. Especially, the package includes DSS, Diffie-Hellman, and RSA for public-key encryption; IDEA, 3DES and CAST-128 for symmetric encryption; and SHA-1 for hash coding. Thirdly, it has a wide range of applicability, encrypting emails contents to individuals who wish to communicate securely with others worldwide over the Internet. Finally, PGP was not developed by, nor is it controlled by any governmental or standard organization (Stallings, 2012).

## PGP Operations

### Authentication

The authentication methodology used by PGP is Digital Signature. Generally, the sender will attach a signature with the message, the signature is produced by hashing the message and encrypting that hash using sender’s private key.

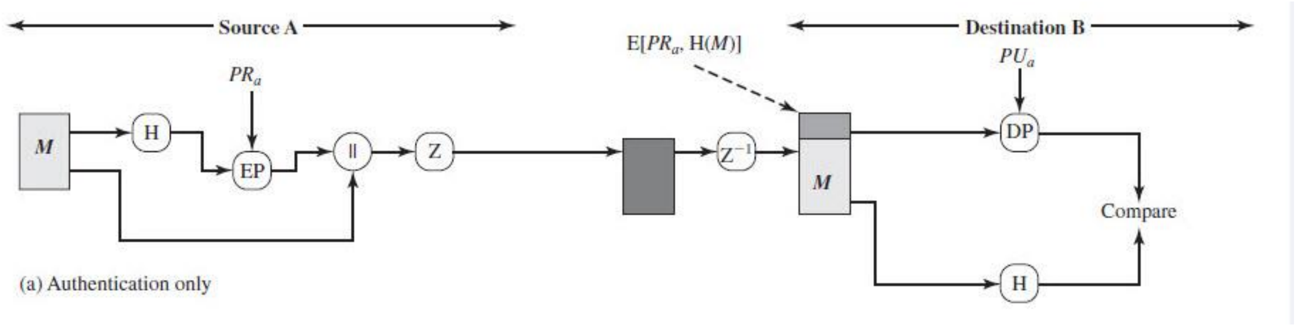


Figure 1 Authentication (Stallings, 2012)

$K_s$ = session key used in symmetric encryption scheme
$PR_a$ = private key of user A, used in public-key encryption scheme
$PU_a$ = public key of user A, used in public-key encryption scheme
EP = public-key encryption
DP = public-key decryption
H = hash function
|| = concatenation
Z = compression function
$Z^{-1}$ = decompression function

Initially, User A creates a message M, Message is passed through the hash function(SHA-1) and generates 160-bit hash code of the corresponding message. Hash code is encrypted using the private key of user A, which is concatenated with a message and passed through the compression function. The receiver passes the data through $Z^{-1}$ function, finds the hash by decrypting using the public key of user A and the message again hashed using a hash function. Finally, user B compares the computed hash and received hash. If they are equal, user B can confirm that the message was from user A and authentic.

The user can use one of the two methodologies DSS/SHA or RSA/SHA. Hashcode can be generated using SHA-1. The obtained hash is encrypted using RSA or DSS with the sender's private key.

**Confidentiality**

Confidentiality is another feature involved in PGP operations, which encrypt the entire messages to be transmitted. The message can be encrypted using CAST-128 or IDEA or 3DES (algorithms) with one-time session key created by the sender. The produced session key can encrypt using RSA or Diffie-Hellman algorithms.

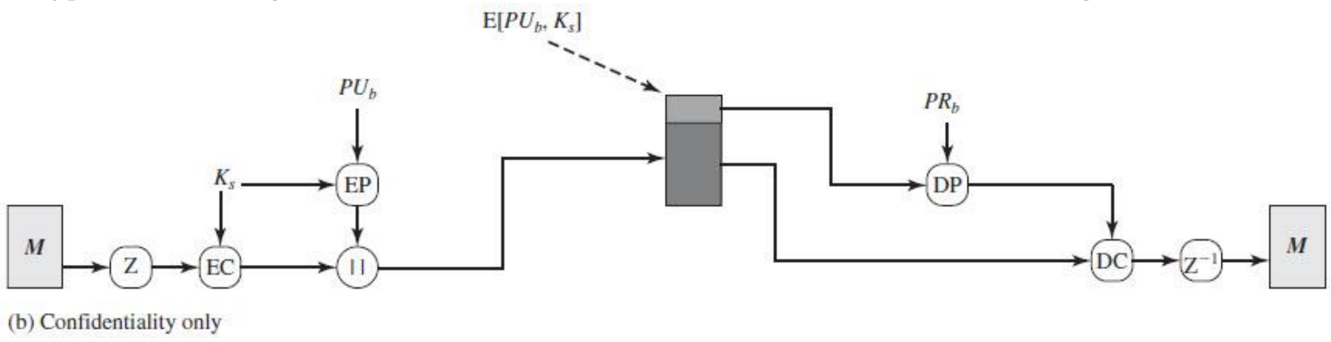


Figure 2 Confidentiality (Stallings, 2012)

Firstly, user A created message and a 128 bit random number used as session key for message which was passed through the compression function, encrypting the compressed data using with CAST-128 (or 3DES or IDEA) session key generated by user A. Session key is again encrypted using public key of user B and this is attached to encrypted message. The recipient retrieves the session key using user B's private key and user B will procure compressed data when data was decrypted using session key. Finally, user B acquire message when data is passed through inverse compression function.

**Confidentiality and Authentication**

This explains the usage of both services simultaneously in the message. Initially, the signature was generated for the corresponding message and appended to the message. Entire data was compressed with using ZIP algorithm, which is encrypted with the session key generated by user A, session key was encrypted using RSA or Diffie Helman algorithm and include the encrypted data. User b recovers session key and decrypt the ciphertext using session key. Finally, verify the signature by comparing the computed hash and received hash.

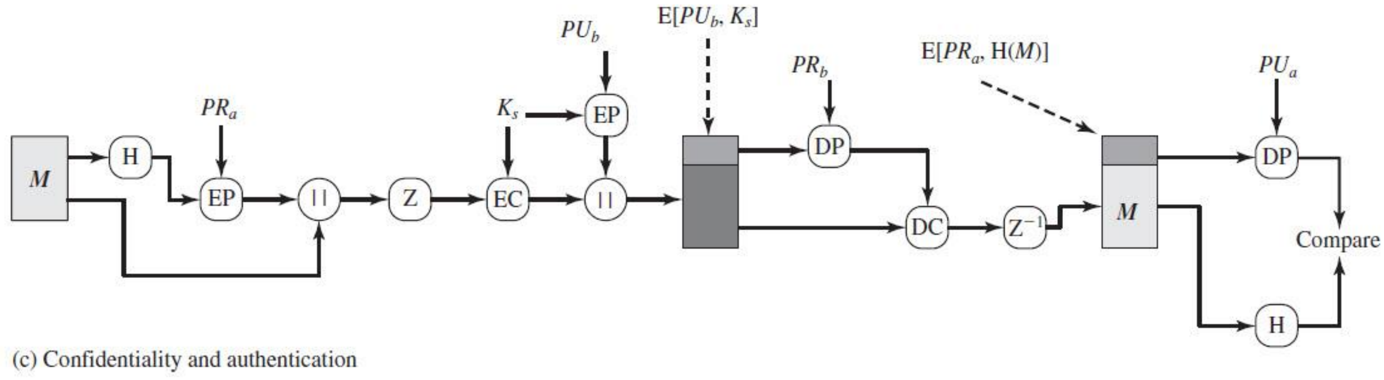


Figure 3 Confidentiality and Authentication (Stallings, 2012)

**Compression**

As mentioned in the figures (fig 1, fig 2 and fig 3), Z is used for compression and $Z^{-1}$ is used for inverse compression. The algorithm used for compression and decompression is ZIP algorithm. By default, PGP compresses the message after applying the signature but before encryption. This enables the benefit of saving space during transmission.

**Email Compatibility**

When PGP is used, at least portion of the blocks is encrypted via transmission and these blocks consist of a stream of arbitrary 8-bit octets. But most of the electronic mail systems allow the use of blocks to be formed in ASCII characters. PGP overcomes this problem by using radix-64 conversion, each group of 3 octets of binary data is mapped into corresponding four ASCII characters and this format also includes a cyclic redundancy check to detect transmission errors. Apart from that, the use of radix-64 expands a message by 33%.

For example: (Ding, n.d.)

| | |
|---|---|
| Email Message: | new |
| ASCII Format: | 01101110 01100101 01110111 |

After encryption: 10010001 10011010 10001000

Radix-64 conversion:

| | | | | |
|---|---|---|---|---|
| 3 octets of binary data: | 10010001 10011010 10001000 | | | |
| Four 6-bit blocks: | 100100 | 011001 | 101010 | 001000 |
| Integer version: | 36 | 25 | 38 | 8 |
| Printable version: | k | Z | m | I |

### Segmentation and Reassembly

Email features are limited to the maximum length of the message, for instance, some of the email facilities which imposes a maximum length of 50000 octets. If the message length exceeds, the message will be divided into smaller segments and each of them will be sent separately. To avoid this issue, PGP automatically broke the message into smaller segments and sent via email if the message is too large. The segmentation process is performed in the end even after radix-64 conversion. Therefore, the signature section and session key section appear only once, at the starting of the first segment. Receiver end needs to be reassembled before verifying signature and decryption ("Pretty Good Privacy", n.d).

## PGP Trust Model

The PGP trust is not required for a small number of users unless PGP users are scaled to globally. For example: Consider a real-life scenario, Anirudh knows a person named Ray. They were working together for more than one year and Anirudh trust Ray. While Anirudh doesn't know another person named Francis, but Ray knows Francis very well and Ray trusts him. So Ray introduces Francis to Anirudh and now Anirudh believes Francis. Similarly, PGP trust model works based on a chain of introducers.

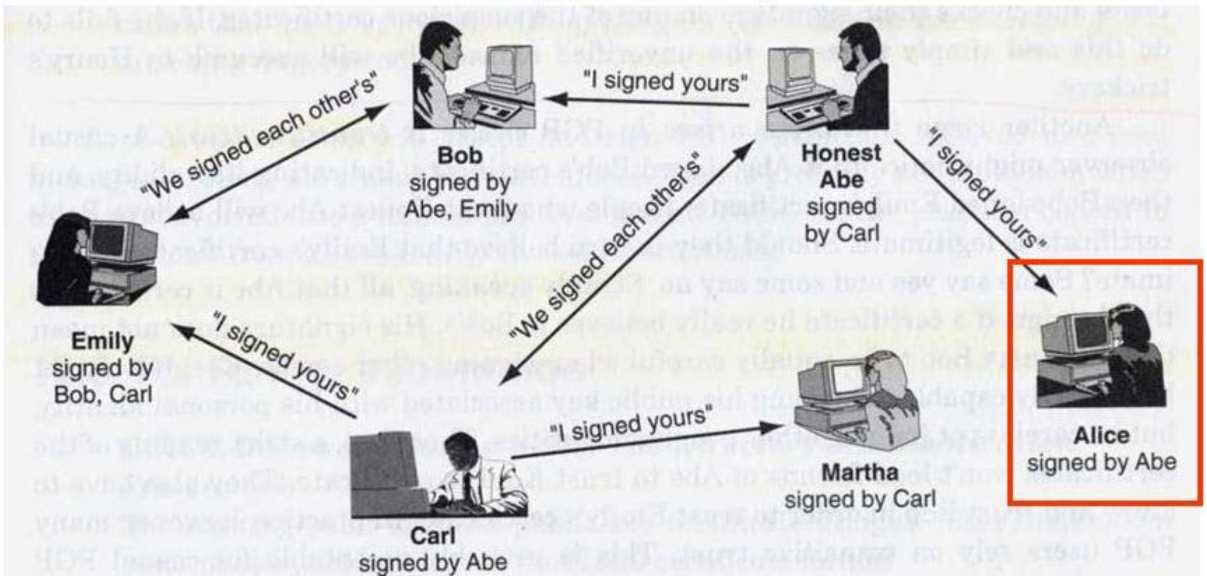


Figure 4 Web of Trust ("Electronic mail security PGP & S/MIME - ppt video online download", n.d.)

In the above figure, Alice trusts only Honest Abe to sign certificates and she won't believe the certificates from Emily and Martha are genuine. If Alice trusts Bob and Carl, she can trust everyone's certificate.

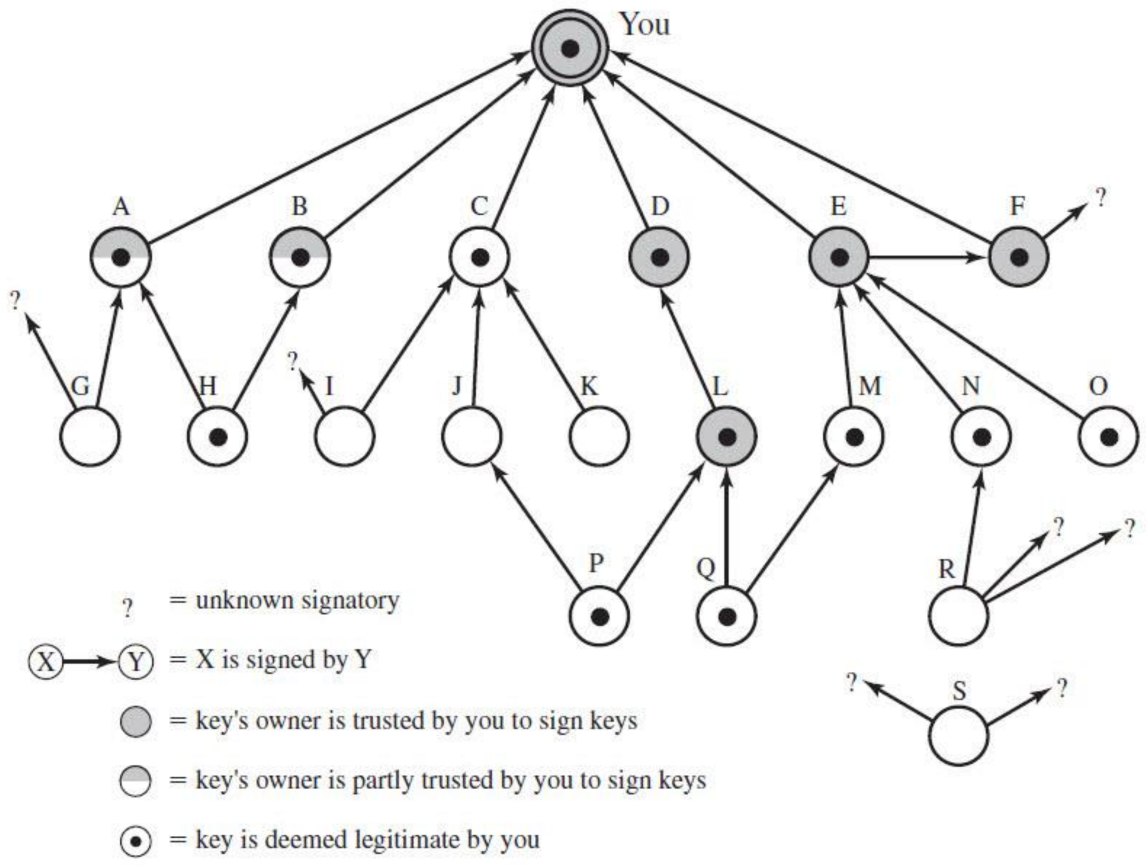

You
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
?
? = unknown signatory
X → Y = X is signed by Y
● = key's owner is trusted by you to sign keys
◓ = key's owner is partly trusted by you to sign keys
⊙ = key is deemed legitimate by you

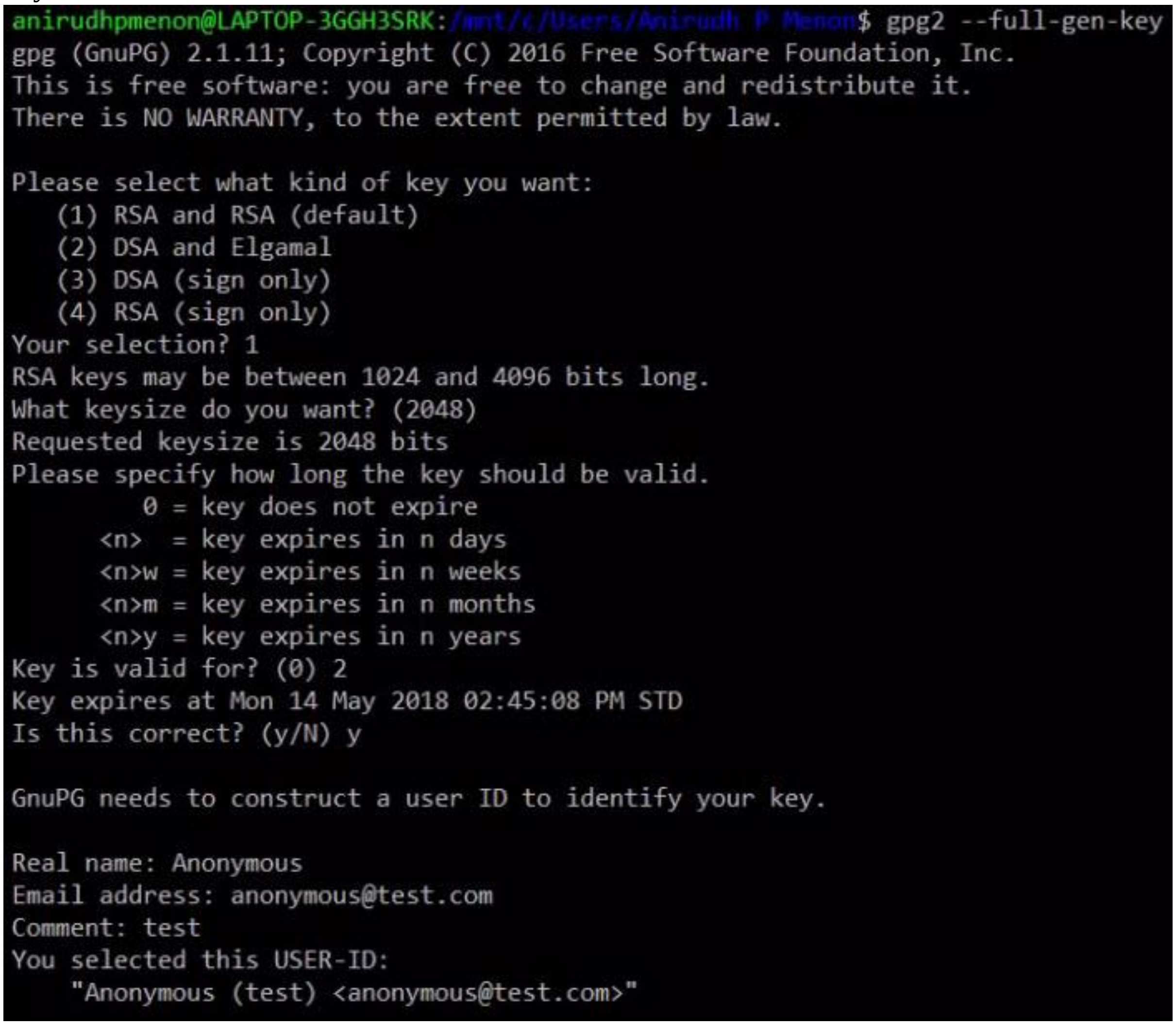
Figure 5 PGP Trust model (Stallings, 2012)

The node named “you” refers to the entry in the public key ring corresponding to this user. This key is legitimate and owner trust value is ultimate trust. In this example, You node always trusts some entities they are D, L, E and F whereas you node partially trusts user A and user B. The shading shows the level of trust assigned by you node. The tree structure illustrates which keys are signed by other users. If a key is signed by a node whose key is also in the tree, the arrow connects the signed key to the signatory. If a key is not signed by a node whose key is not present in the key ring, the arrows connect the signed key to an unknown signatory(?) which means signatory is unknown to this user.

## GnuPG Tool

Gnu Privacy Guard is a PGP based tool for secure communication, it includes key pair generation, exchanging and verifying keys, encrypting and decrypting messages, and authenticating using digital signatures ("The GNU Privacy Handbook", n.d.).

1. **Generating a Key**
   The command-line option --full-gen-key or –gen-key is used to generate a new pair of keys.



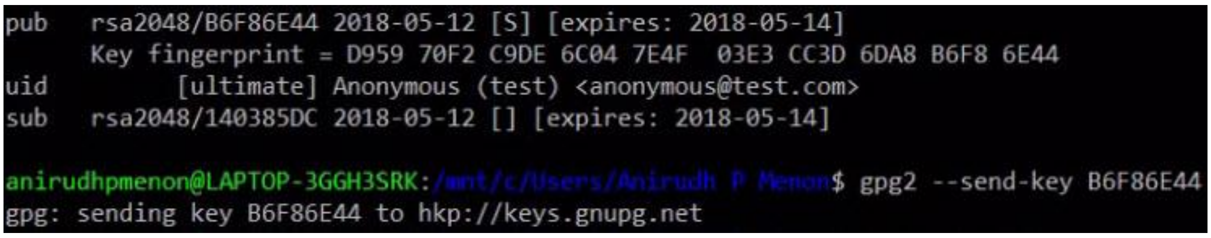
```
anirudhpmenon@LAPTOP-3GGH3SRK:/mnt/c/Users/Anirudh P Menon$ gpg2 --full-gen-key
gpg (GnuPG) 2.1.11; Copyright (C) 2016 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

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)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 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) 2
Key expires at Mon 14 May 2018 02:45:08 PM STD
Is this correct? (y/N) y

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

Real name: Anonymous
Email address: anonymous@test.com
Comment: test
You selected this USER-ID:
    "Anonymous (test) <anonymous@test.com>"
```

Figure 6 Generating key

GnuPG has several options for creating keys, but primary key must be capable of making signatures. There are mainly four options, option 1 and option 2 creates 2 pairs of keys, one for signature and the other for encryption. In the case of option 3 and option 4, there is only one key generated and which is used for signing purpose.

You need to choose the size of the RSA keys between 1024 and 4096 bits. The longer the key more secure against brute-force attacks, however for almost all purposes the default key size is sufficient. Apart from that, larger key length will slower the decryption and encryption process, and a larger key size may affect the signature length. You must choose an expiration period normally most users use a key that does not expire, it varies upon the usage of users. Finally, the user needs to provide user ID to the generate the key. GnuPG requires a passphrase to protect the primary and subordinate private keys that the user keep in possession.

After that, you may need to perform some actions (type on the keyboard, move the mouse, utilize the disk) on your computer for a while for GPG to generate a random number. This gives the random number generator a better chance to gain entropy.

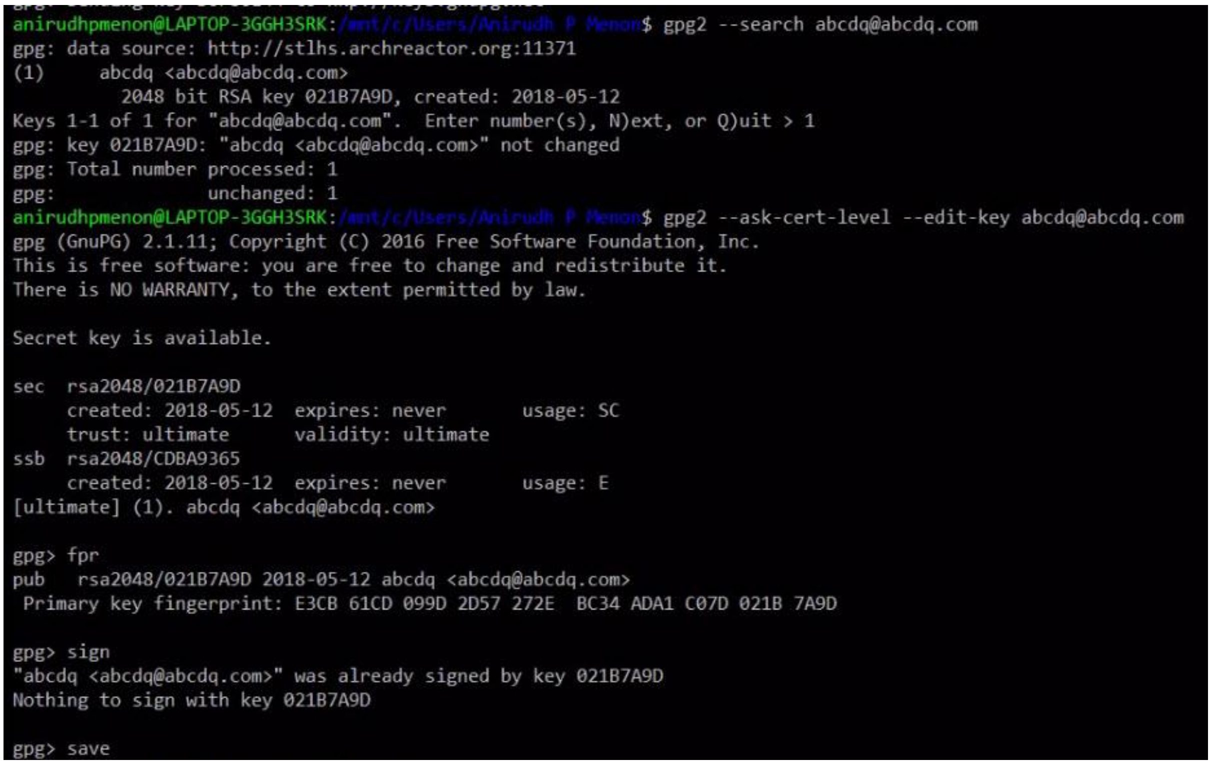


```
pub   rsa2048/B6F86E44 2018-05-12 [S] [expires: 2018-05-14]
      Key fingerprint = D959 70F2 C9DE 6C04 7E4F  03E3 CC3D 6DA8 B6F8 6E44
uid         [ultimate] Anonymous (test) <anonymous@test.com>
sub   rsa2048/140385DC 2018-05-12 [] [expires: 2018-05-14]

anirudhpmenon@LAPTOP-3GGH3SRK:/mnt/c/Users/Anirudh P Menon$ gpg2 --send-key B6F86E44
gpg: sending key B6F86E44 to hkp://keys.gnupg.net
```

Figure 7 Key details

In this case, the Key ID is B6F86E44. To publish the newly created public key to the public key server via command line is –send –key “key id”.

2. **Finding and signing a key**
When a user signing someone’s key, the user should always verify the identity as the best. In this case, I had created another user named “abcdq” before. Initially, we need to search the user abcdq@abcdq.com.



```
anirudhpmenon@LAPTOP-3GGH3SRK:/mnt/c/Users/Anirudh P Menon$ gpg2 --search abcdq@abcdq.com
gpg: data source: http://stlhs.archreactor.org:11371
(1)     abcdq <abcdq@abcdq.com>
          2048 bit RSA key 021B7A9D, created: 2018-05-12
Keys 1-1 of 1 for "abcdq@abcdq.com".  Enter number(s), N)ext, or Q)uit > 1
gpg: key 021B7A9D: "abcdq <abcdq@abcdq.com>" not changed
gpg: Total number processed: 1
gpg:              unchanged: 1
anirudhpmenon@LAPTOP-3GGH3SRK:/mnt/c/Users/Anirudh P Menon$ gpg2 --ask-cert-level --edit-key abcdq@abcdq.com
gpg (GnuPG) 2.1.11; Copyright (C) 2016 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.

Secret key is available.

sec  rsa2048/021B7A9D
     created: 2018-05-12  expires: never       usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/CDBA9365
     created: 2018-05-12  expires: never       usage: E
[ultimate] (1). abcdq <abcdq@abcdq.com>

gpg> fpr
pub   rsa2048/021B7A9D 2018-05-12 abcdq <abcdq@abcdq.com>
 Primary key fingerprint: E3CB 61CD 099D 2D57 272E  BC34 ADA1 C07D 021B 7A9D

gpg> sign
"abcdq <abcdq@abcdq.com>" was already signed by key 021B7A9D
Nothing to sign with key 021B7A9D

gpg> save
```

Figure 8 Finding and signing a key

Using –ask-cert-level flag specifies the level of certainty about this key and using “fpr” command shows the fingerprint (i.e public key), by this way user verify another user. “sign” command will let the user sign the keys(In this case “abcdq” was already signed). Finally, finish the process by saving the signature, and publish the key to the key server using “save” command.

## PGP Issues

- PGP uses end to end encryption, so end users may receive malicious emails.
- PGP Trust model issues: User needs to find a chain of introducers. Apart from that suppose a user X is compromised by an adversary, other users who trust user X would be at risk.
- No forward secrecy: Forward secrecy means that the encrypted communications and sessions recorded in the past cannot be retrieved and decrypted, suppose secret key or password compromised in future attacker can retrieve all data.
- Both sender and receiver should agree with using email encryption.
- Most of the configurations in PGP are done manually. For instance, creating a pair of keys, creating revoke a certificate, adding new user’s public key, verifying fingerprint.

## Possible Solutions

- Certificate authority: Most of the PGP trust model issues can be eliminated by using a reliable certificate authority. A user can present his or his public key to the authority in a secure manner and obtain a certificate and the user can publish the certificate. Suppose anyone needing user A's public key can obtain the certificate and verify that it is valid by the way of attached trusted signature.

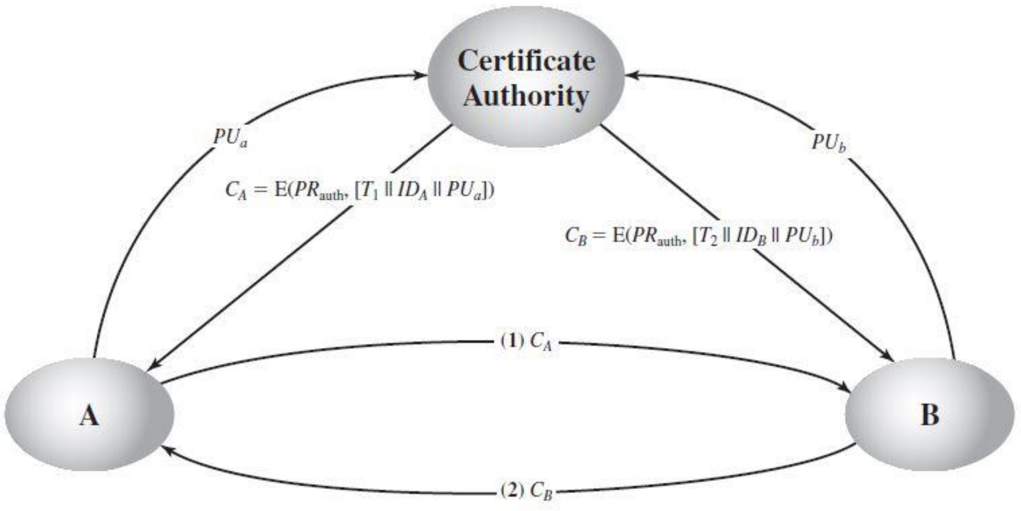

Certificate Authority
$PU_a$
$PU_b$
$C_A = E(PR_{auth}, [T_1 \| ID_A \| PU_a])$
$C_B = E(PR_{auth}, [T_2 \| ID_B \| PU_b])$
(1) $C_A$
(2) $C_B$
A
B

Figure 9 Certificate Authority (Stallings, 2012)

For instance: Google, Yahoo, Mircosoft are currently top email providers. If any of these organization becomes a trusted third party(CA), most of the problems can be avoided.

- End users need to use some anti-spam filters or some software to filter out malicious emails.

## Alternatives

The signal is an encrypted communication app for Android and iOS, they provide with desktop versions for Windows, Linux, and macOS. Signal uses the internet to send one-to-one and group messages. It uses standard cellular mobile numbers as identifiers and uses end-to-end encryption to secure all communications to other signal users. The application provides a methodology by which users can independently verify the identity of their messaging correspondents and the integrity of the data channel. However, the signal is not providing email services ("Signal (software)", n.d.).

## Conclusion

In my opinion, most of the people are not aware of PGP service. Moreover, some users or organizations feel very difficult to use PGP due to manual configurations. Finally, it depends on your usage of email. If you or your company needs a secure communication, I think PGP is the good option.

## References

- Stallings, W. (2012) Cryptography and network security (5th ed.).
- A pretty good introduction to PGP. Retrieved from https://georgebrock.github.io/talks/pretty-good-introduction/
- OpenPGP, PGP, and GPG: What is the difference?. Retrieved from https://www.goanywhere.com/blog/2013/07/18/openpgp-pgp-gpg-difference
- The GNU Privacy Handbook. (2018). Retrieved from https://www.gnupg.org/gph/en/manual.html
- Ding, C. Radix-64 Conversion in PGP.
- Electronic mail security PGP & S/MIME - ppt video online download. Retrieved from http://slideplayer.com/slide/9050742/
- Signal (software). Retrieved from https://en.wikipedia.org/wiki/Signal_(software)
- Edward Snowden. Retrieved from https://en.wikipedia.org/wiki/Edward_Snowden
- Pretty Good Privacy. (2018). Retrieved from http://jbiet.edu.in/coursefiles/cse/HO/cse3/NS4.pdf