GnuPG/GPG- An Implementation of the OpenPGP standard

Bharani MOORTHY

Master of Engineering (System,Networks and Security), EPITA, Paris, France

bharani.moorthy3495@gmail.com

***Abstract*—The acceleration in the advance of web communication has leads to extension growth of the data privacy risks and increase in the vulnerability of the data. Cryptography is an indispensable in computer communication, due to the online privacy concerns our data is not safe without encryption. Zimmerman invented the PGP encryption standard, where it is then implemented as GPG - an Opensource. It gives the effective solution for the secure email communication Here, I have discussed about its function, its RSA computation by taking Libgcrypyt's, and its Historical changes by fixing the vulnerabilities.**

Keywords— public key, random number generation, cryptography, Symmetricity,signature, Error Function,Hash Functions, RSA algorithm

I. PGP AND OPENPGP OVERVIEW

In 1991, Phil Zimmerman- the man behind the PGP Program, in order to provide privacy, security, and authentication for the online communication systems. In 1997, Phil proposed to the Internet Engineering Task Force (IETF) to create an open source PGP. It was accepted and created as an OpenPGP, defines the standard format to encrypt and decrypt over the internet, authenticate messages with digital signatures and encrypt stored files-Full-disk Encryption and network protection.

Chronological Releases:

PGP (1991) → OpenPGP (1997) → GnuPG (1999)

*PGP is currently owned by Symantec Corporation, it is a proprietary version

II. PGP FUNCTION:

It uses public and private key pairs-but it performs encryption and decryption in more difficult forms as it combines both the symmetric and asymmetric cryptography.Let's look on the

A. PGP Encryption:

The sender of the message has the original plain messages, then generates a random symmetric encryption key, sender encrypts the message using the random key and encrypts the random key with the recipient's public key. Sender then transfers the encrypted message which is the combination of the encrypted data and the encrypted random key.

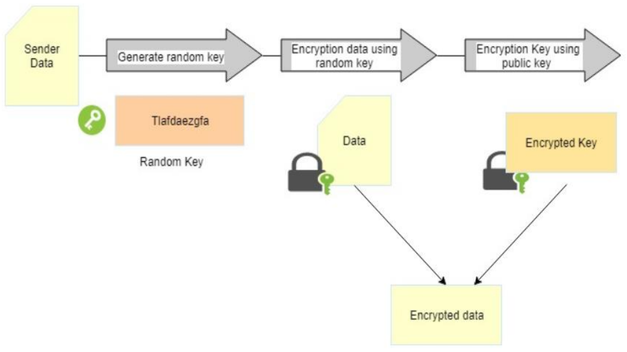


Fig. 1. Example of a figure caption. (*PGPEncryption)drawn in draw.io*

B. PGP Decryption:

When the recipient receives the encryption messages, the user performs the following decryption process

- Firstly, the recipient decrypts the encrypted random key using the recipient's private key.
- It produces the random key created by the sender.
- Next, the recipient uses that random key to decrypt the encrypted message and retrieve the original message.

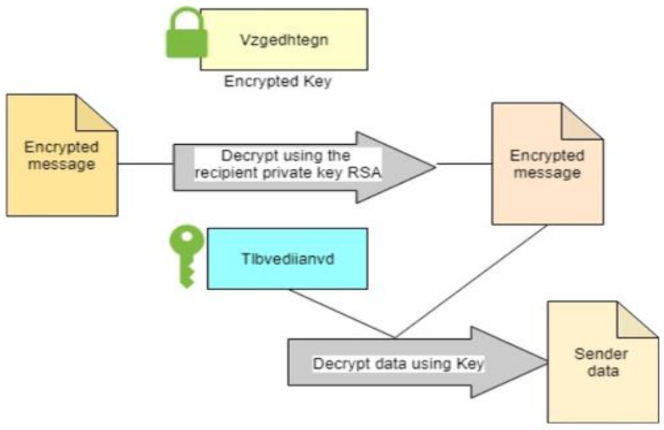


Fig. 2. Example of a figure caption. (PGP Decryption) drawn in draw.io

C. Into GnuPG (GPG):

GnuPG is the implementation of the OpenPGP and its free. As it has been derived from the OpenPGP standard, it uses the Hybrid-encryption- where *Symmetric-key cryptography for speed* + public-key cryptography for flexibility in secure key *exchange* and it uses access modules for all kinds of public key directories. This protocol is widely used for email communication encryption.

Moreover, GPG is the command line tool also called as universal crypto engine does not have any graphic tools.

GnuPG has 2 prominent versions: GnuPG 1.x and GnuPG 2.x. GnuPG 2.x this supports the modern encryption algorithms that should be preferred over GnuPG 1.x[1].

D. *GPG Trust model using the Web of trust:*

Trust and authenticity networks are possible solutions for the key authenticity problem in this decentralized public-key infrastructure. A trust model, the so-called Web of Trust, has been proposed for and is implemented in this popular e-mail GnuPG. Yes, instead of the digital certificate in traditional key exchange, here we are using the Web of trust (WoT) [2].

1. Firstly, there are three levels of trust supported: complete trust, marginal trust, and no trust.
2. The owner of the key ring, who needs to manually assign these trust values for all other users, automatically receives full trust (also called implicit or ultimate trust).
3. When a user places trust in an introducer, implicitly it means that the user possesses a certain amount of confidence in the introducer's capability to issue valid certificates, i.e. correct bindings between users and public keys.

E. *Key Validation in GPG*

Based on such trust values, the GPG trust model suggests accepting a given public key in the key ring as completely valid, if anyone of the following

(a) the public key belongs to the owner of the key ring,

(b) the key ring contains at least C certificates from completely trusted introducers with valid which public keys,

(c) the key ring should contain at least M certificates from marginally trusted introducers with the proper valid public keys.

In order to compensate for the above-mentioned mess-up of the trust levels, the PGP trust model allows the users to individually adjust the two scepticism parameters C (also called COMPLETES NEEDED) and M (also called MARGINALS NEEDED).

In general, higher numbers for these parameters imply that more people would be needed to conspire against you. The default values in GPG are C = 1 and M = 2, and C = 1 and M = 3 in GnuPG. If a given key is not completely valid according to the above rules, but if at least one certificate of a marginally or completely trusted introducer with a valid public key is present, then the key attains the status marginally valid.[6]

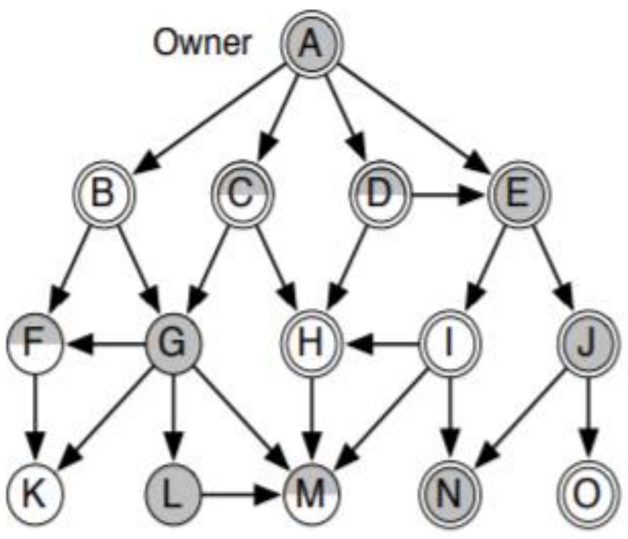


Fig. 3. Example of a figure caption. (*web of trust tree*) adapted from[2]

Example: Considering the certificate graph which shown in above figure which illustrates the PGP trust model. An arrow from X to Y representing a certificate issued by X for Y. Grey circles indicating the **complete trust** (A, E, G, J,L, N), grey semicircles indicate **marginal trust** (C, D, F, M), and white circles indicate **no trust** (B, H, I,K, O). The results of the key validation are shown for C = 1 and M = 2. Completely valid public keys are represented by nested circles (A, B, C, D, E, H, I, J, N, O). It is good to noting that all public keys with a certificate issued by A, the owner of the key ring, which are completely valid.

We are here to discuss about one of the GPG cryptographic libraries – Libgcrypt which is written in C program.

Libgcrypt Library (Gcrypt):

Libgcrypyt is a general-purpose cryptographic library and it is most widely used Library on GPG. It is C language-based code and it facilitates functions for all cryptographic building blocks such as symmetric ciphers, hash algorithms, MACs, large integer functions, public key algorithms, random numbers and a lot of supporting functions. It has two advantages such as

- it is Opensource-free to use
- It encapsulates the low-level cryptography-using extendable and flexible API compared to others

Importantly, Library is fully dependable on the library 'libgpg-error', it contains the common error handling for GPG.

All the interfaces of the Libcrypt libraries are defined in this header file – **#include <gcrypt.h>** thus the function and the type name are **gcry_*** and **GCRY*** for the other symbol and as same applies for the libgpg-error where it uses **gpg_err_*** and **GPG_ERR_*.**

Additionally, gcrypt.h file is attached along with this following zip file

Many functions in Libgcrypt could be return an error if they fail. It handles **gcry_error_t** datatype and the multiple function which extracts the error code and handle it.

Multi-Threading, this library called as thread-safe that supports the multithreading when it call-back its command GCRYCTL_SET_THREAD_CBS.

F. *Architecture of the Libgcrypt:*

The two most paramount things about any encryption technique are the algorithm and the key. Conventional protection of sensitive data where it requires a strong algorithm, which means that it's harder to break mathematically. Additionally, It requires that the key which is used to encrypt and decrypt the data is protected properly. Where the Libgcrypyt's cryptographic blocks description has been explained in below

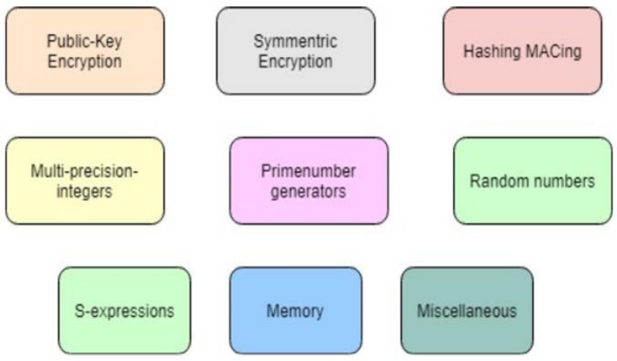


Fig. 4. Example of a figure caption. (*Libgcrypt architechture) draw.io*

G. *Symmetric cryptography:*

As seen in the introduction, the user initially generates the random symmetric encryption, to generate those it uses the cipher functions-where it utilizes the shared key.GPG has numerous available cipher function, each function starts with GCRY_CIPHER_* , in order to use the cipher algorithm ,user allocates an according handle using following function

gcry_error_t gcry_cipher_open (gcry cipher hd t *hd, int algo, int mode, unsigned int flags) [Function]

where the ID of algorithm to use must be specified via algo it follows chosen algorithm AES, AES128, RIJNDAEL, BLOWFISH, 3DES, GOST28147 etc

H. *Public Key cryptography:*

This is also known as the asymmetric cryptography used for key management and to provide digital signatures, Libgcrypt allows two completely versatile interface to public key cryptography- RSA (Rivest-Shamir-Adleman) algorithms as well as DSA (Digital Signature Algorithm) and Elgamal.

1. This feature allows to add more algorithms in future
2. It uses s expression libraries API in C thus the S-expressions are a data structure for representing complex data. It is used to read the complex integer codes by parsing where the following information is stored
   - keys
   - plain text data
   - encrypted data
   - Signatures
3. To create S-expressions is by using **gcry_sexp_build** where it allows rp pass a string to insert MPI (multi-precision-integers)

I. *How the RSA algorithm is really computed in Libgcrypt library:*

An RSA private key and public key syntax is described by this S-expression:

```
(private key                    /* Private key*/
(rsa
(n n-mpi)
(e-mpi)
(d d-mpi)
(p p-mpi)
(q q-mpi)
(u u-mpi)))
(public key                   /* Public key*/
(rsa
(n n-mpi)
(e e-mpi)))
n-mpi    RSA public modulus n.
e-mpi    RSA public exponent e.
d-mpi    RSA secret exponent d = e-1 mod (p −
1) (q − 1).   /* rsa-use-e value*/
p-mpi    RSA secret prime p.
q-mpi    RSA secret prime q with p < q.
u-mpi    Multiplicative inverse u = p-1 mod q.  /*
```

* Where rsa-use-e value can be only used with RSA to give a hint for the public exponent. The value will be used as a base to test for a usable exponent. Some values are special:

'0' - Use a secure and fast value. This is currently the number 41.

'1' - Use a value as required by some crypto policies. This is

currently the number 65537.

'2' - Reserved

'> 2' - Use the given value.

For signing and decryption the parameters (p, q, u) are optional but using that it greatly improves the performance[4].

J. *Cryptographic Functions-About #PKCS:*

Based on the RSA, Libgcrypt uses PKCS#1 block type 2 padding for encryption, block type 1 padding for signing, It is described as (*data*

(flags pkcs1)

(value block))

The above function checks that this data can be used with the given key, does the padding and encrypts it. If function is successful, it will return value will be 0 and the format will be as

(enc-val

(rsa

(a a-mpi))

Here it defines the mathematical properties and definitions that above said RSA public key and private keys must have the traditional key pair is as calculated as the following functions:

*gcry_error_t gcry_pk_sign (gcry sexp t *r_sig, gcry sexp t data, gcry sexp t skey)*

This function creates a digital signature for data using the private key skey and place

it into the variable at the address of r sig data may either be the simple. Mostly, this will be "**sha256" or "sha1**"

(data

(flags pkcs1)

(hash hash-algo block)) *[5]*

Where **Modulus n,** product of two distinct large prime numbers **p and q**, such that .

When it is multi-prime key n = ri * r2 * …….ri , for i >=2 ,thus normally p = r1 and q = r2.,

- whereas RSA public key is represented as the tuple (n,e), thus e is the public exponent
- RSA private key is representes as (n,d) thus d is the private exponent

Basically, Gcrpty uses-*x931* Force the use of the ANSI X9.31 key generation algorithm instead of the default algorithm. This flag is only meaningful for RSA key generation and usually not required. Note that this algorithm is implicitly used if other than derive-parameters is given.

Where S/MIME integration used PKCS#7 secure message format.

Example of generation of the RSA parameters P and Q:

```
(genkey
(rsa
    (nbits 4:1024)
    (rsa-use-e 1:3)
(derive-parms
    (Xp1 #1A1916DDB29B4EB7EB6732E128#)
    (Xp2 #192E8AAC41C576C822D93EA433#)
(Xp #D8CD81F035EC57EFE822955149D3BFF70C53520D
    769D6D76646C7A792E16EBD89FE6FC5B605A6493
    39DFC925A86A4C6D150B71B9EEA02D68885F5009
    B98BD984#)
(Xq1 #1A5CF72EE770DE50CB09ACCEA9#)
(Xq2 #134E4CAA16D2350A21D775C404#)
(Xq #CC1092495D867E64065DEE3E7955F2EBC7D47A2D
    7C9953388F97DDDC3E1CA19C35CA659EDC2FC325
    6D29C2627479C086A699A49C4C9CEE7EF7BD1B34
    321DE34A#))))
```

referred from[4]

These intended definitions of the cryptography also include the X.509 thus the string is processed by the RSA algorithm is formatted by the concatenation of a header, padding, the hash and a trailer, it operates only on the bit steams as the following parameters

- Where the data is converted between byte and bit string formats by taking the most-significant bit of the foremost byte of the byte string as the leftmost bit of the bit string.
- A signature is transformed from a bit string to a byte string by padding the bit string on the left with 0 to 7 zero bits so that the resultant length in bits is a multiple of 8,

It uses 2 functions to verify and Signing process as follows

Function	**Key Type**	**Input Length**	**Output Length**
C_Sign	RSA private key	<= *k* -2	*k*
C_verify	RSA public key	<= *k* - 2, *k* ^{2}	N/A

Additionally, I have a look on the RSA in Libgcrypt algorithm which taught by professor Erra as below it follows the ANS X9.31.

RSA in libgcrypt 1.4.4: e >= 65537

Algorithm: RSA key generation (it follows ANS X9.31)

Input: — an integer k = 1024 + 256s > 0;

Output: — (N, e, d) with N a k bit number

```
Begin:
e = 65537;
Compute randomly a prime p of k/2 bits;
Compute randomly a prime q of k/2 bits;
Compute N = p q and φ (N) = (p - 1)(q - 1);
Compute (N) = lcm (p - 1; q - 1) = φ (N)/gcd(p - 1; q - 1)
While GCD (e, λ (N)) ≠1  e = e + 2;
/* So Again: if e > 65537, we gain information about (N) */
```

```
    Compute d = e^-1 mod f ;
    End.                                        referred [7]
```

K. Hashing Functions and MACing:

Libgcrypt provides the Hashing individually as well as Hashing with MAC (HMAC) using the below mentioned routines. Creating a message digest object for algorithm algo. flags may be given as a bitwise OR of constants described below

*gcry_error_t gcry_md_open (gcry md hd t *hd, , int algo, unsigned int flags* *[Hash Function]*

gcry_error_t gcry_mac_open (gcry mac hd t *hd, int algo, unsigned int flags, gcry ctx t ctx) [MAC Function]

of the hash functions from the GCRY_MD_*, * - SHA1 , SHA512, MD5 etc. where for the message authentication code would be like the GCRY_MAC_HMAC_* , * may be SHA256 , SHA224 , SHA3_256 etc.

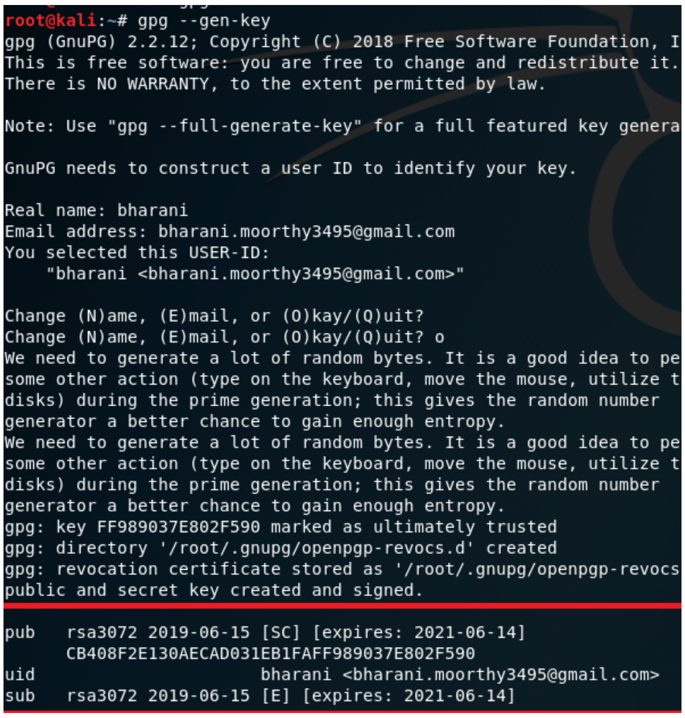
III. Generating the Keys Using the GPG Tool

Here, I am generating the keys using the GPG 2.2 version as seen below

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
GPG(1)                    GNU Privacy Guard 2.2                    GPG(1)

NAME
       gpg - OpenPGP encryption and signing tool
```

Fig. 5. Example of a figure caption. (*GPG 2.2 version*)

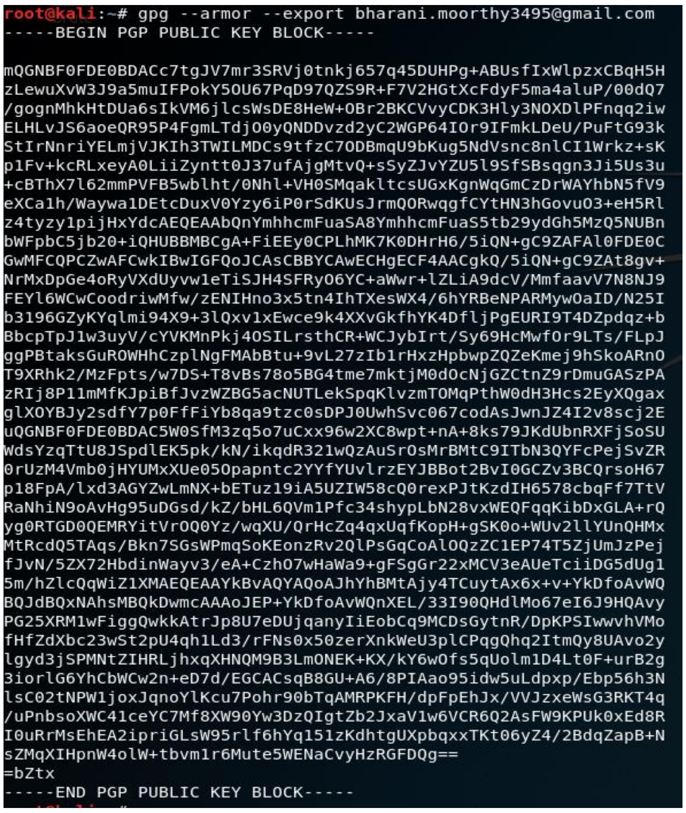
Secondly, generating the own key which prompts to asking information about the mail ID, key length whether 2048 or 4096 bits and the passphrase.



```
root@kali:~# gpg --gen-key
gpg (GnuPG) 2.2.12; Copyright (C) 2018 Free Software Foundation, I
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key genera

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

Real name: bharani
Email address: bharani.moorthy3495@gmail.com
You selected this USER-ID:
    "bharani <bharani.moorthy3495@gmail.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit?
Change (N)ame, (E)mail, or (O)kay/(Q)uit? o
We need to generate a lot of random bytes. It is a good idea to pe
some other action (type on the keyboard, move the mouse, utilize t
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to pe
some other action (type on the keyboard, move the mouse, utilize t
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key FF989037E802F590 marked as ultimately trusted
gpg: directory '/root/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/root/.gnupg/openpgp-revocs
public and secret key created and signed.

pub   rsa3072 2019-06-15 [SC] [expires: 2021-06-14]
      CB408F2E130AECAD031EB1FAFF989037E802F590
uid                      bharani <bharani.moorthy3495@gmail.com>
sub   rsa3072 2019-06-15 [E] [expires: 2021-06-14]
```

Fig. 6. Example of a figure caption. (*generating the GPG Keys*)

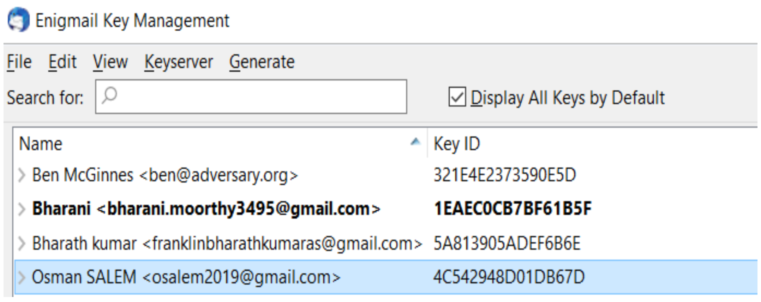
I am exporting the GPG key -Base 64 format as it begins and ends the RSA key as shown



```
root@kali:~# gpg --armor --export bharani.moorthy3495@gmail.com
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBF0FDE0BDACc7tgJV7mr3SRVj0tnkj657q45DUHPg+ABUsfIxWlpzxCBqH5H
zLewuXvW3J9a5muIFPokY5OU67PqD97QZS9R+F7V2HGtXcFdyF5ma4aluP/00dQ7
/gognMhkHtDUa6sIkVM6jlcsWsDE8HeW+OBr2BKCVvyCDK3Hly3NOXDlPFnqq2iw
ELHLvJS6aoeQR95P4FgmLTdjO0yQNDDvzd2yC2WGP64IOr9IFmkLDeU/PuFtG93k
StIrNnriYELmjVJKIh3TWILMDCs9tfzC7ODBmqU9bKug5NdVsnc8nlCI1Wrkz+sK
p1Fv+kcRLxeyA0LiiZyntt0J37ufAjgMtvQ+sSyZJvYZU5l9SfSBsqgn3Ji5Us3u
+cBThX7l62mmPVFB5wblht/0Nhl+VH0SMqakltcsUGxKgnWqGmCzDrWAYhbN5fV9
eXCa1h/Waywa1DEtcDuxV0Yzy6iP0rSdKUsJrmQORwqgfCYtHN3hGovuO3+eH5Rl
z4tyzy1pijHxYdcAEQEAAbQnYmhhcmFuaSA8YmhhcmFuaS5tb29ydGh5MzQ5NUBn
bWFpbC5jb20+iQHUBBMBCgA+FiEEy0CPLhMK7K0DHrH6/5iQN+gC9ZAFAl0FDE0C
GwMFCQPCZwAFCwkIBwIGFQoJCAsCBBYCAwECHgECF4AACgkQ/5iQN+gC9ZAt8gv+
NrMxDpGe4oRyVXdUyvw1eTiSJH4SFRyO6YC+aWwr+lZLiA9dcV/MmfaavV7N8NJ9
FEYl6WCwCoodriwMfw/zENIHno3x5tn4IhTXesWX4/6hYRBeNPARMywOaID/N25I
b3196GZyKYqlmi94X9+3lQxv1xEwce9k4XXvGkfhYK4DfljPgEURI9T4DZpdqz+b
BbcpTpJ1w3uyV/cYVKMnPkj4OSILrsthCR+WCJybIrt/Sy69HcMwfOr9LTs/FLpJ
ggPBtaksGuROWHhCzplNgFMAbBtu+9vL27zIb1rHxzHpbwpZQZeKmej9hSkoARnO
T9XRhk2/MzFpts/w7DS+T8vBs78o5BG4tme7mktjM0dOcNjGZCtnZ9rDmuGASzPA
zRIj8P11mMfKJpiBfJvzWZBG5acNUTLekSpqKlvzmTOMqPthW0dH3Hcs2EyXQgax
glXOYBJy2sdfY7p0FfFiYb8qa9tzc0sDPJ0UwhSvc067codAsJwnJZ4I2v8scj2E
uQGNBF0FDE0BDAC5W0SfM3zq5o7uCxx96w2XC8wpt+nA+8ks79JKdUbnRXFjSoSU
WdsYzqTtU8JSpdlEK5pk/kN/ikqdR321wQzAuSrOsMrBMtC9ITbN3QYFcPejSvZR
0rUzM4Vmb0jHYUMxXUe05Opapntc2YYfYUvlrzEYJBBot2BvI0GCZv3BCQrsoH67
p18FpA/lxd3AGYZwLmNX+bETuz19iA5UZIW58cQ0rexPJtKzdIH6578cbqFf7TtV
RaNhiN9oAvHg95uDGsd/kZ/bHL6QVm1Pfc34shypLbN28vxWEQFqqKibDxGLA+rQ
yg0RTGD0QEMRYitVrOQ0Yz/wqXU/QrHcZq4qxUqfKopH+gSK0o+WUv2llYUnQHMx
MtRcdQ5TAqs/Bkn7SGsWPmqSoKEonzRv2QlPsGqCoAlOQzZC1EP74T5ZjUmJzPej
fJvN/5ZX72HbdinWayv3/eA+CzhO7wHaWa9+gFSgGr22xMCV3eAUeTciiDG5dUg1
5m/hZlcQqWiZ1XMAEQEAAYkBvAQYAQoAJhYhBMtAjy4TCuytAx6x+v+YkDfoAvWQ
BQJdBQxNAhsMBQkDwmcAAAoJEP+YkDfoAvWQnXEL/33I90QHdlMo67eI6J9HQAvy
PG25XRM1wFiggQwkkAtrJp8U7eDUjqanyIiEobCq9MCDsGytnR/DpKPSIwwvhVMo
fHfZdXbc23wSt2pU4qh1Ld3/rFNs0x50zerXnkWeU3plCPqgQhq2ItmQy8UAvo2y
lgyd3jSPMNtZIHRLjhxqXHNQM9B3LmONEK+KX/kY6wOfs5qUolm1D4Lt0F+urB2g
3iorlG6YhCbWCw2n+eD7d/EGCACsqB8GU+A6/8PIAao95idw5uLdpxp/Ebp56h3N
lsC02tNPW1joxJqnoYlKcu7Pohr90bTqAMRPKFH/dpFpEhJx/VVJzxeWsG3RKT4q
/uPnbsoXWC41ceYC7Mf8XW90Yw3DzQIgtZb2JxaV1w6VCR6Q2AsFW9KPUk0xEd8R
I0uRrMsEhEA2ipriGLsW95rlf6hYq151zKdhtgUXpbqxxTKt06yZ4/2BdqZapB+N
sZMqXIHpnW4olW+tbvm1r6Mute5WENaCvyHzRGFDQg==
=bZtx
-----END PGP PUBLIC KEY BLOCK-----
```

Fig. 7. Example of a figure caption. (exporting the PGP key Base64 format)

Additionally, I have used thunderbird email communication to get the concrete real time example, as I created, and I have added my friend and sent the mail using the signing and the encryption of the mail. Below screen shot is the keyring of my account,each key we can set the expiration date.[8]



Enigmail Key Management

File Edit View Keyserver Generate

Search for: ☑ Display All Keys by Default

Name	Key ID
Ben McGinnes <ben@adversary.org>	321E4E2373590E5D
Bharani <bharani.moorthy3495@gmail.com>	**1EAEC0CB7BF61B5F**
Bharath kumar <franklinbharathkumaras@gmail.com>	5A813905ADEF6B6E
Osman SALEM <osalem2019@gmail.com>	4C542948D01DB67D

Fig. 8. Fig. 8. Example of a figure caption. (keyring of Enigmail)

IV. Historical changes in GPG by fixing the vulnerabilities

As we have seen that GPG has predominantly two version as 1.X and 2.X, but the initial version starts from the 0.X. The first release version of the GPG 0.0.0 was 1997.This 0.X and 1.X version will be called as the “classic version” and 2.X version series is called as “ Modern “.

– “Classic” GPG 0.X to 1.X changes overview

GnuPG 0.2.3 (1998)-Initially the set of the crew found that a bug in the calculation of ELG fingerprints. This is now fixed, but all existing fingerprints and keyids for ELG keys are not any more valid. They have implemented the logic to handle the web of trust. It also has some bugs. As the string "(INSECURE!)" which appended to a new user-id if there is need to generate on a system even without a good random number generator.

GnuPG 0.3.3(1998): where it uses the Iterated+Salted passphrases worked. Making sure that the PGP is able to handle them you may want to use the options k-mode 3 –s2k-cipher-algo cast5 –s2k-digest-algo sha1" when changing a passphrase.

GnuPG 0.4.3(1998): The experimental support for keyrings stored in a GDBM database. This is much faster than a standard keyring. It will notice that the import gets slower with time, so the reason is that all new keys are used to verify signatures of previous inserted keys.It used *"–keyring gnupg-gdbm:<name-of-gdbm-file>"*.This was not (yet) supported for secret keys.

GnuPG 1.0.5 (2001) The large File Support (LFS) has been working from this version. Secret keys were not allowed to be imported unless user use the new option –allow-secret-key-import.

– “Modern” -2.x

GnuPG 2.0.0 (2006) – First stable version which integrates openPGP and the S/MIME,where it uses PKCS#7 secure message format.

GnuPG 2.2.8 (2018)-The Vulnerabilities exploits the original filename all over decryption and verification actions, which permits remote attackers to spoof the output that GnuPG where it sends on file descriptor to other programs uses"--status-fd 2" option.[9]

About the Libcrypt Vulnerabilities:

Thus the Libgcrypt 1.7.10 and 1.8.x - It permits a memory-cache side-channel attack on ECDSA signatures that can be lessened over the use of blinding during the signing process in the _gcry_ecc_ecdsa_sign function in cipher/ecc-ecdsa.c, it is called as the Return Of the Hidden Number Problem. To determine an ECDSA key, the attacker needs access to either the local machine or a different virtual machine on the same physical host.

V. Applications:

Using the GPG correctly, we can secure our communications. This is extremely helpful, especially when dealing with sensitive information, but also when dealing with regular, everyday messaging as email services as enigmail, Protonmail, mailfence etc.I have studied about whistle blowing application named as “SecureDrop”,here they are using the GPG protocol to securely encrypting the file where the user needs to drop to the magazine .

REFERENCES

[1] https://en.wikipedia.org/wiki/Pretty_Good_Privacy
[2] https://en.wikipedia.org/wiki/Web_of_trust
[3] https://gnupg.org/
[4] https://raw.githubusercontent.com/Chronic-Dev/libgcrypt/master/src/gcrypt.h
[5] https://gnupg.org/documentation/manuals/gcrypt.pdf
[6]https://pdfs.semanticscholar.org/b026/34c5c134a68d30952ed497f30c1dc166f806.pdf
[7] How to compute RSA keys? Robert Erra & Christophe Grenier aka the EG Group
[8] https://www.securityweek.com/gnupg-vulnerability-allows-spoofing-message-signatures
[9] https://docs.securedrop.org/en/release-0.9/journalist.html