CS695 – Lab 3

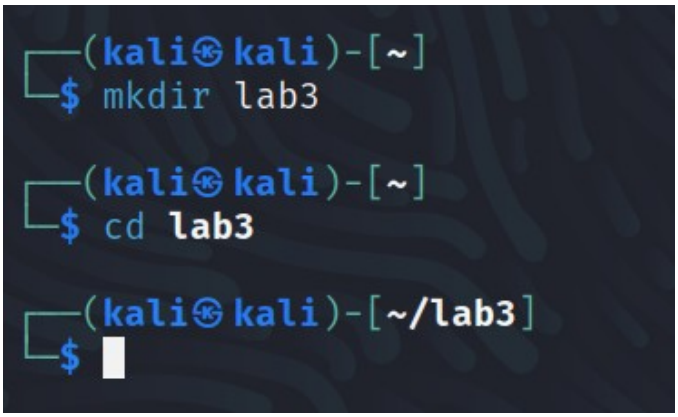
Encryption with GPG

Ryan Christopher

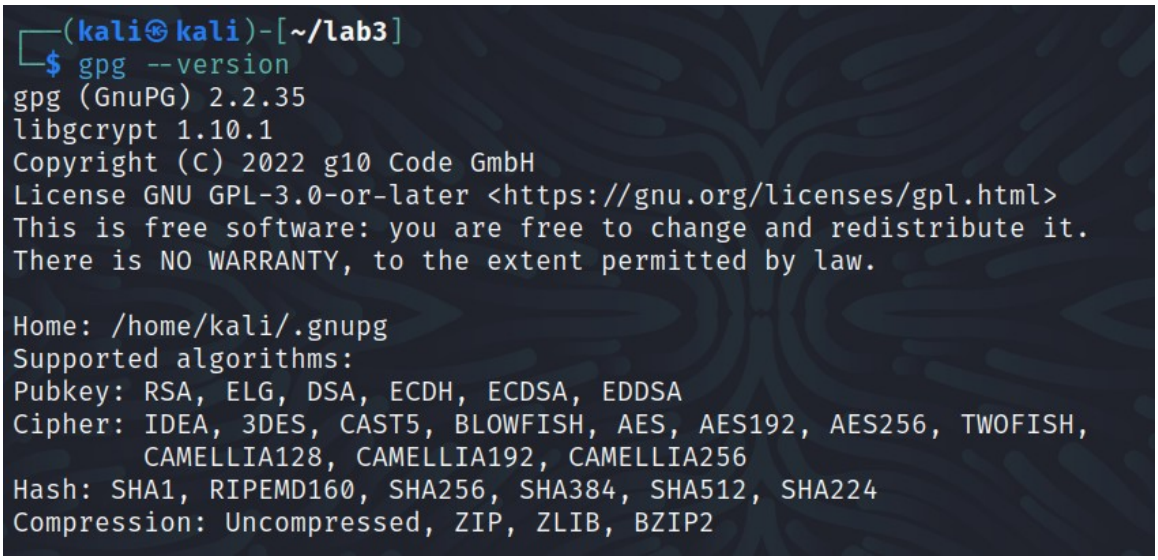
Table of Contents:

Part 1 – Symmetric Encryption

1)



2)



3)

4)

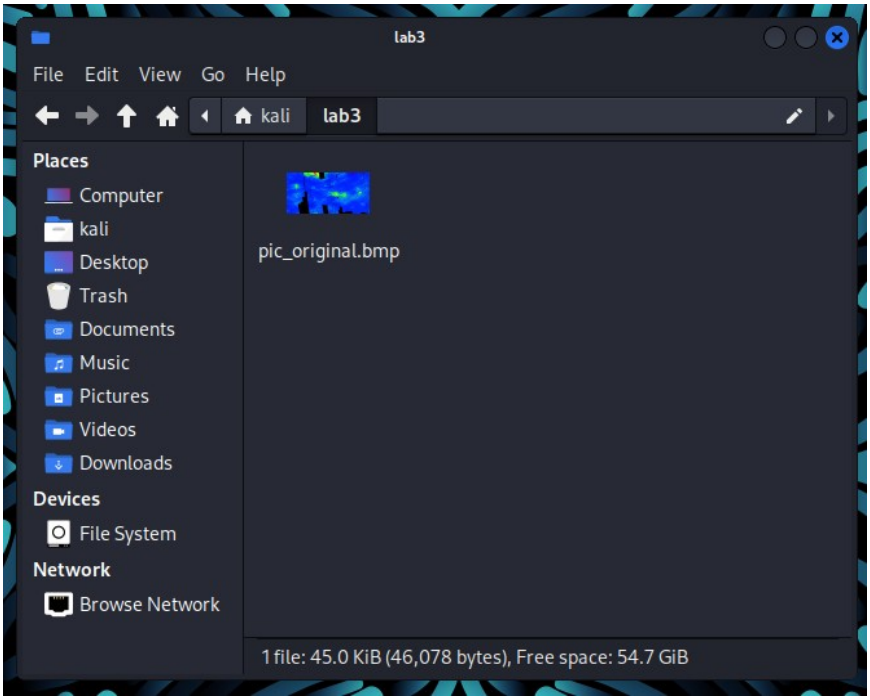
```
┌──(kali㉿kali)-[~/lab3]
└─$ man gpg

┌──(kali㉿kali)-[~/lab3]
└─$ gpg --help
gpg (GnuPG) 2.2.35
libgcrypt 1.10.1
Copyright (C) 2022 g10 Code GmbH
License GNU GPL-3.0-or-later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /home/kali/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2

Syntax: gpg [options] [files]
Sign, check, encrypt or decrypt
Default operation depends on the input data

Commands:

 -s, --sign                          make a signature
     --clear-sign                    make a clear text signature
 -b, --detach-sign                   make a detached signature
```

```
--encrypt
-e      Encrypt  data  to one or more public keys. This command may be combined with --sign (to sign
        and encrypt a message), --symmetric (to encrypt a message that can be decrypted using a  se-
        cret key or a passphrase), or --sign and --symmetric together (for a signed message that can
        be decrypted using a secret key or a passphrase).  --recipient and related  options  specify
        which public keys to use for encryption.

--symmetric
-c      Encrypt  with  a  symmetric  cipher using a passphrase. The default symmetric cipher used is
        AES-128, but may be chosen with the --cipher-algo option. This command may be combined  with
        --sign (for a signed and symmetrically encrypted message), --encrypt (for a message that may
        be decrypted via a secret key or a passphrase), or --sign  and  --encrypt  together  (for  a
        signed  message  that  may  be  decrypted via a secret key or a passphrase).  gpg caches the
        passphrase used for symmetric encryption so that a decrypt operation may  not  require  that
        the user needs to enter the passphrase.  The option --no-symkey-cache can be used to disable
        this feature.
```

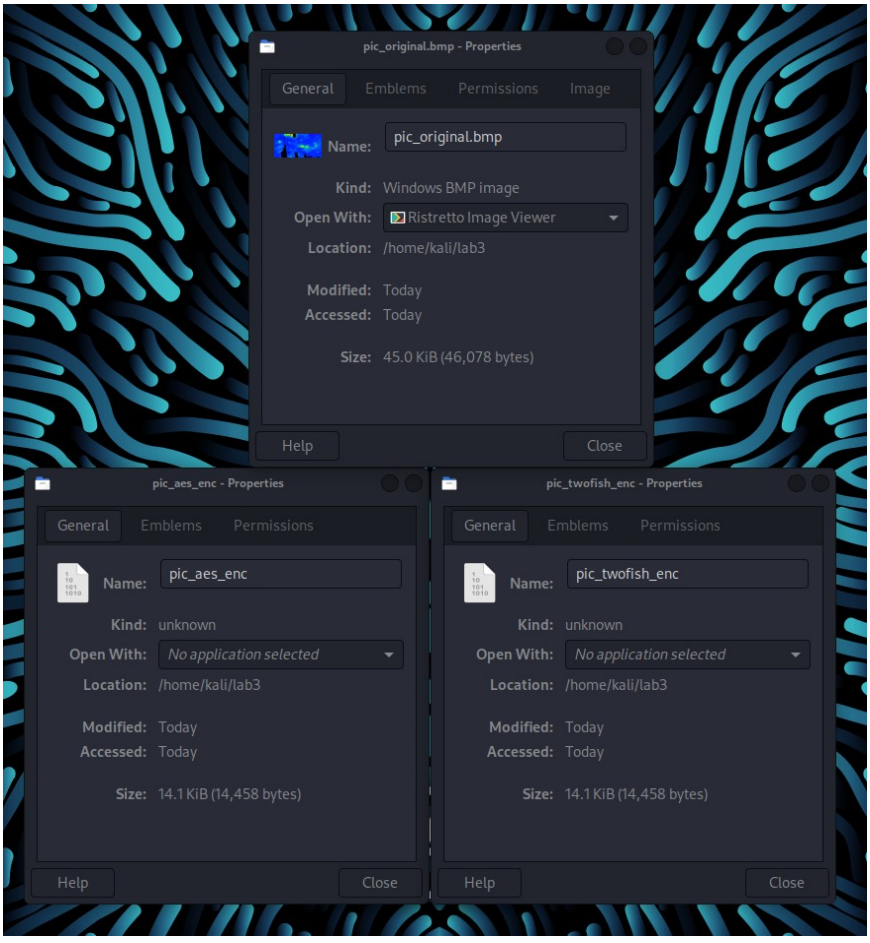
a) The option used to encrypt using a symmetric algorithm is “—symmetric”.

b) The option used to specify different encryption algorithms is “—cipher-algo”.

5)



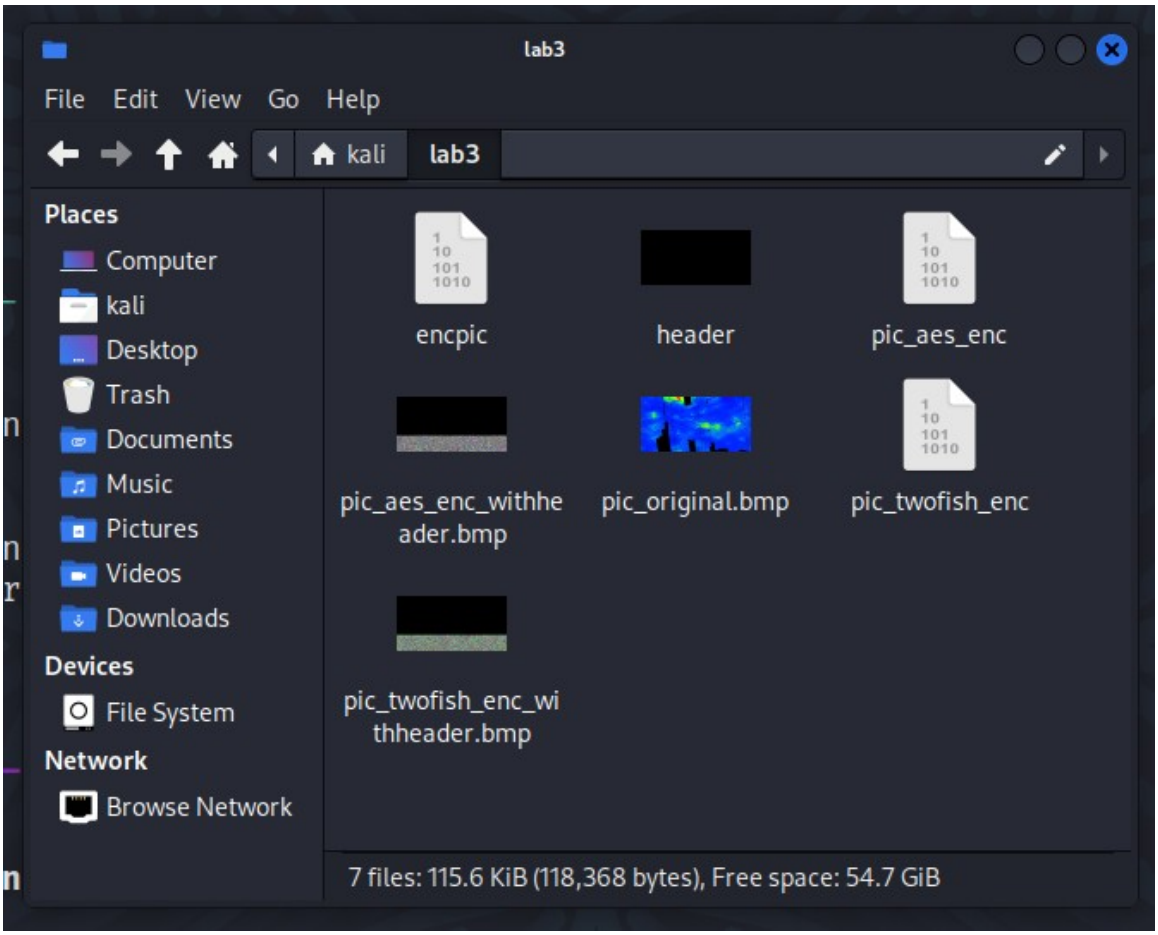

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o pic_aes_enc --symmetric --cipher-algo AES256 pic_original.bmp

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o pic_twofish_enc --symmetric --cipher-algo TWOFISH pic_original.bmp
```

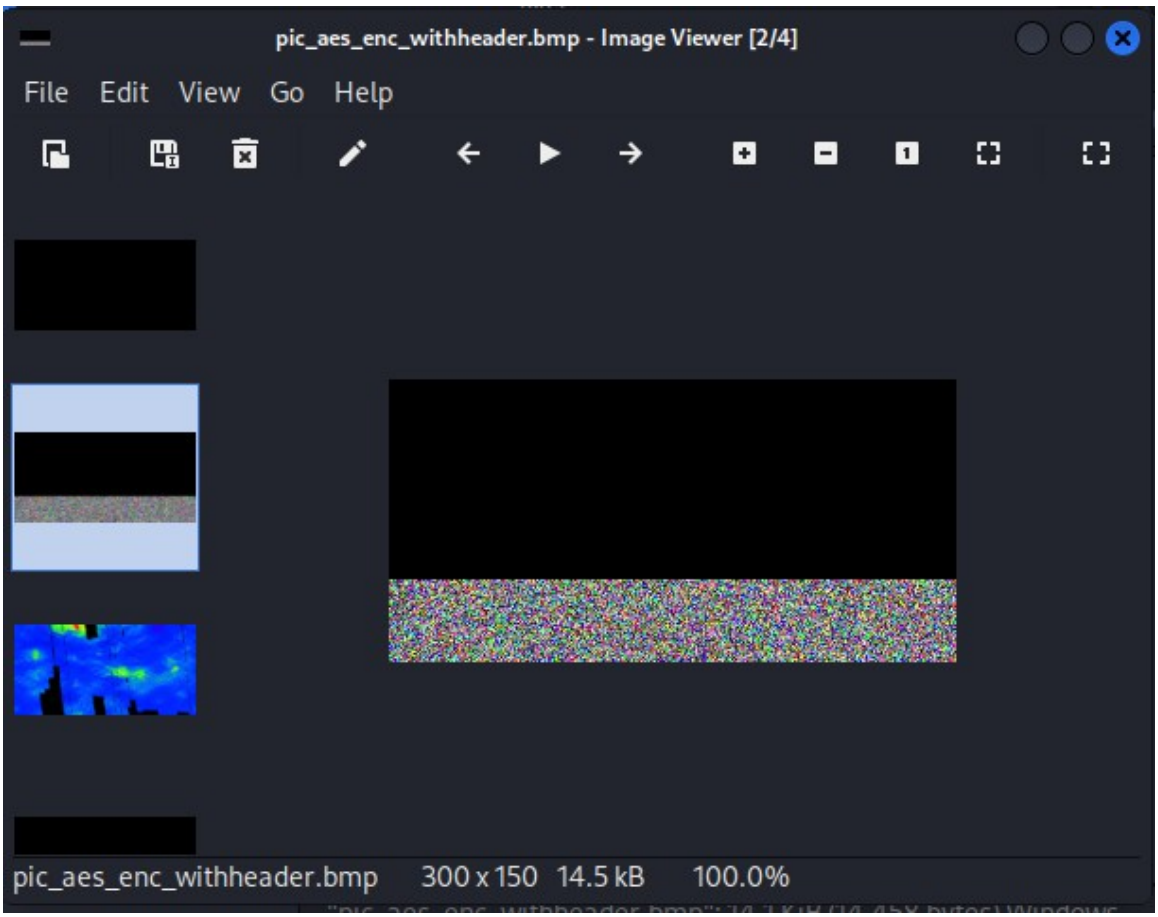


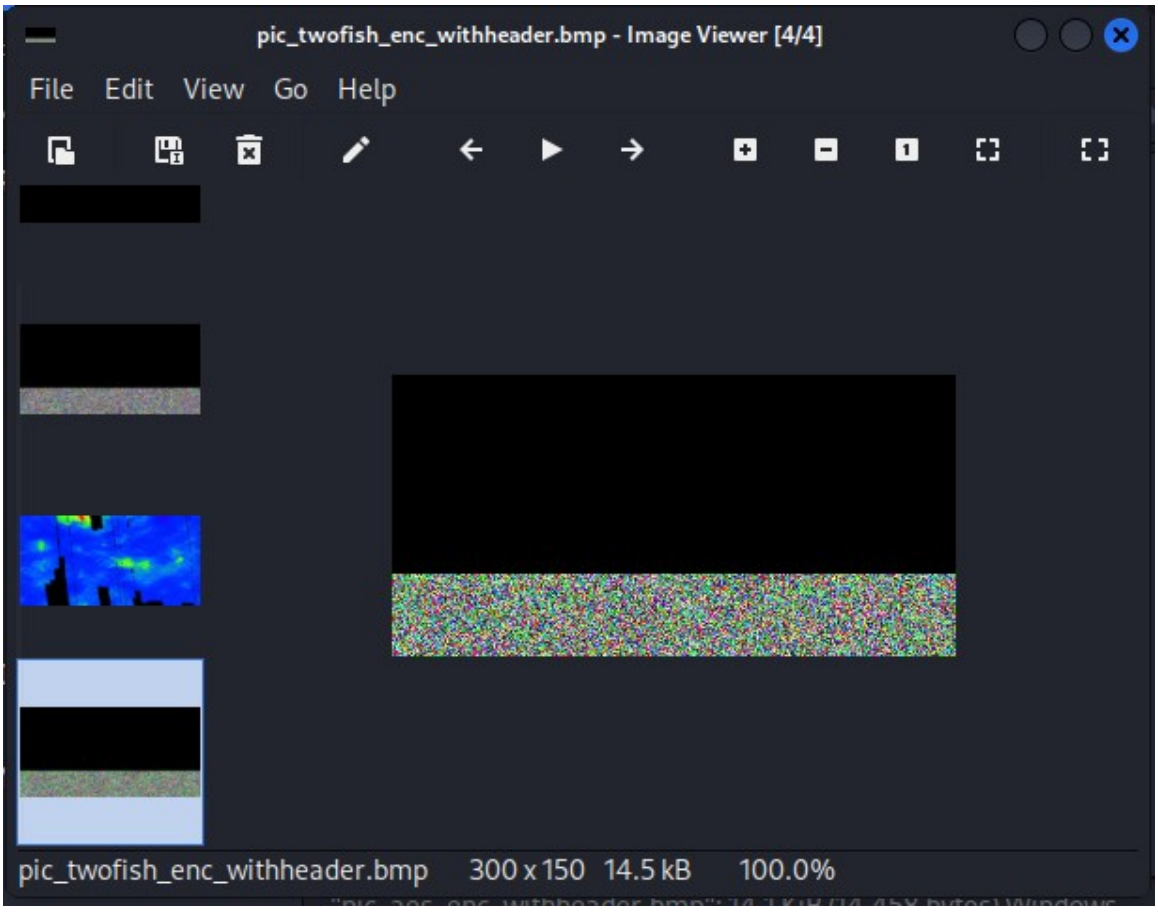
6)

```
┌──(kali㉿kali)-[~/lab3]
└─$ ls
header  pic_aes_enc  pic_original.bmp  pic_twofish_enc

┌──(kali㉿kali)-[~/lab3]
└─$ head -c 54 pic_original.bmp > header

┌──(kali㉿kali)-[~/lab3]
└─$ tail -c +55 pic_aes_enc > encpic

┌──(kali㉿kali)-[~/lab3]
└─$ cat header encpic > pic_aes_enc_withheader.bmp

┌──(kali㉿kali)-[~/lab3]
└─$ tail -c +55 pic_twofish_enc > encpic

┌──(kali㉿kali)-[~/lab3]
└─$ cat header encpic > pic_twofish_enc_withheader.bmp
```

7)

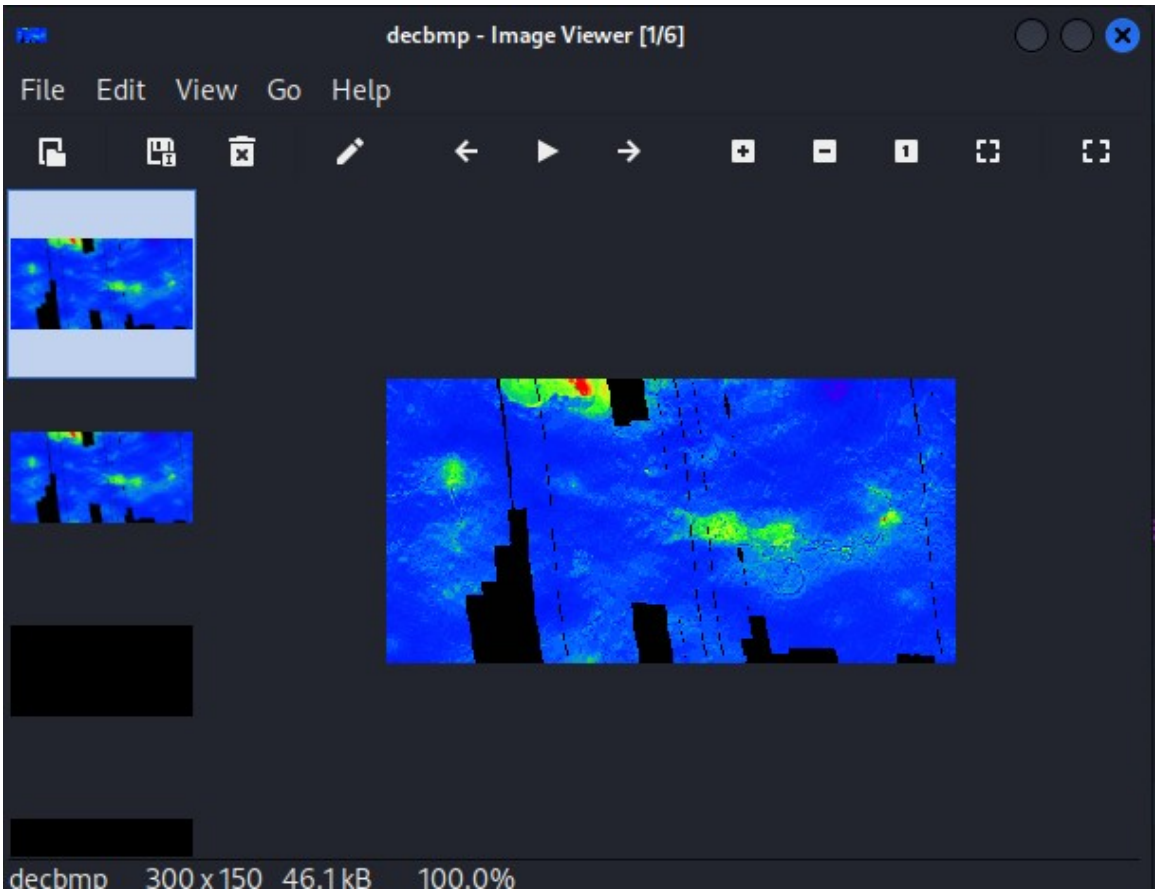


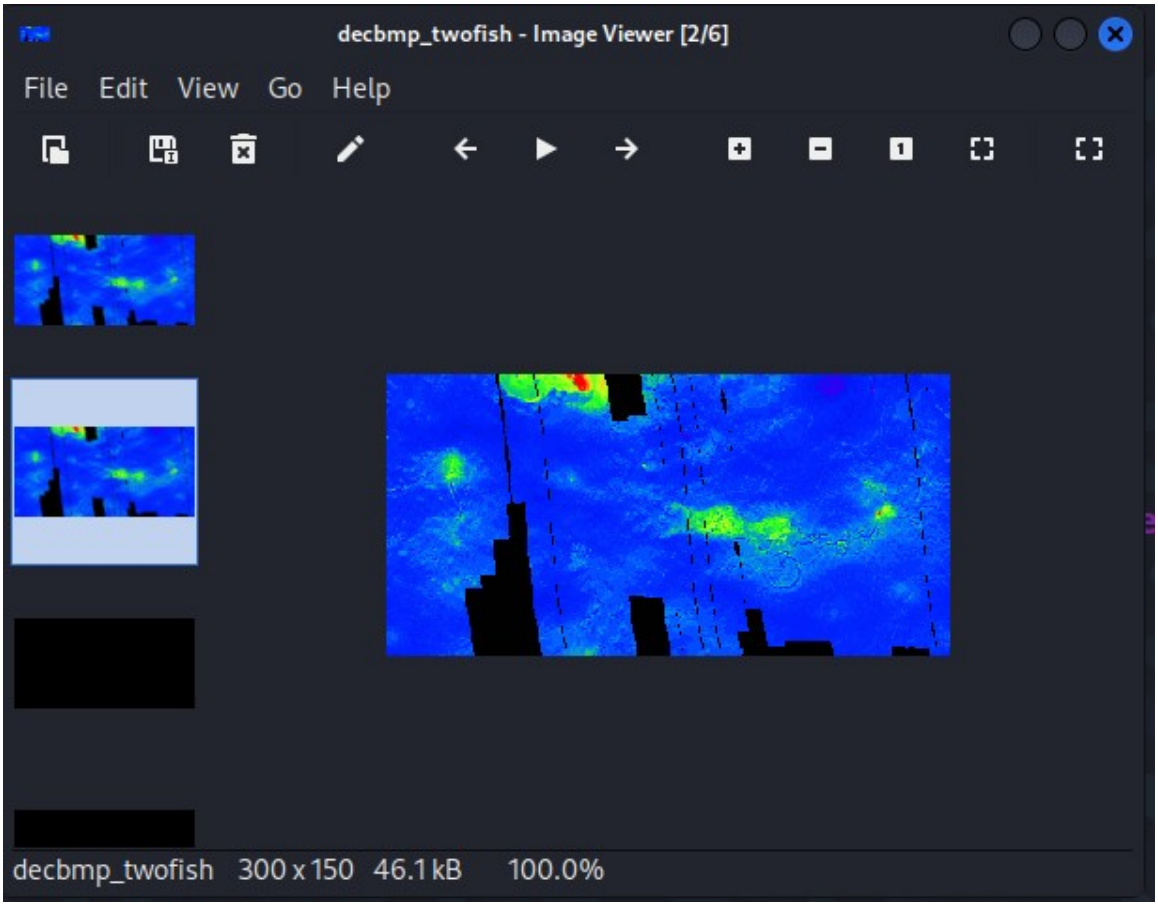

pic_aes_enc_withheader.bmp - Image Viewer [2/4]
File Edit View Go Help
pic_aes_enc_withheader.bmp 300 x 150 14.5 kB 100.0%


pic_twofish_enc_withheader.bmp - Image Viewer [4/4]
File Edit View Go Help
pic_twofish_enc_withheader.bmp 300 x 150 14.5 kB 100.0%

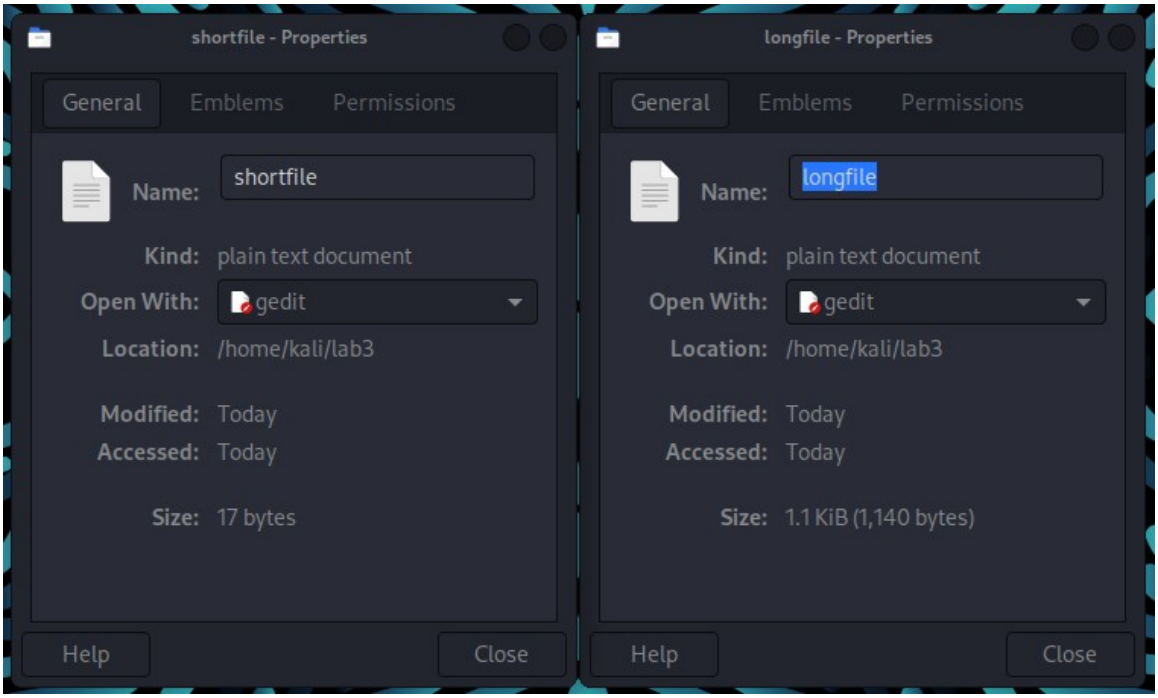
8)

```
┌──(kali㉿kali)-[~/lab3]
└─$ ls
encpic  pic_aes_enc                pic_original.bmp  pic_twofish_enc_withheader.bmp
header  pic_aes_enc_withheader.bmp  pic_twofish_enc

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o decbmp -d pic_aes_enc
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase

┌──(kali㉿kali)-[~/lab3]
└─$ ls
decbmp  header       pic_aes_enc_withheader.bmp  pic_twofish_enc
encpic  pic_aes_enc  pic_original.bmp            pic_twofish_enc_withheader.bmp

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o decbmp_twofish -d pic_twofish_enc
gpg: TWOFISH.CFB encrypted data
gpg: encrypted with 1 passphrase
```

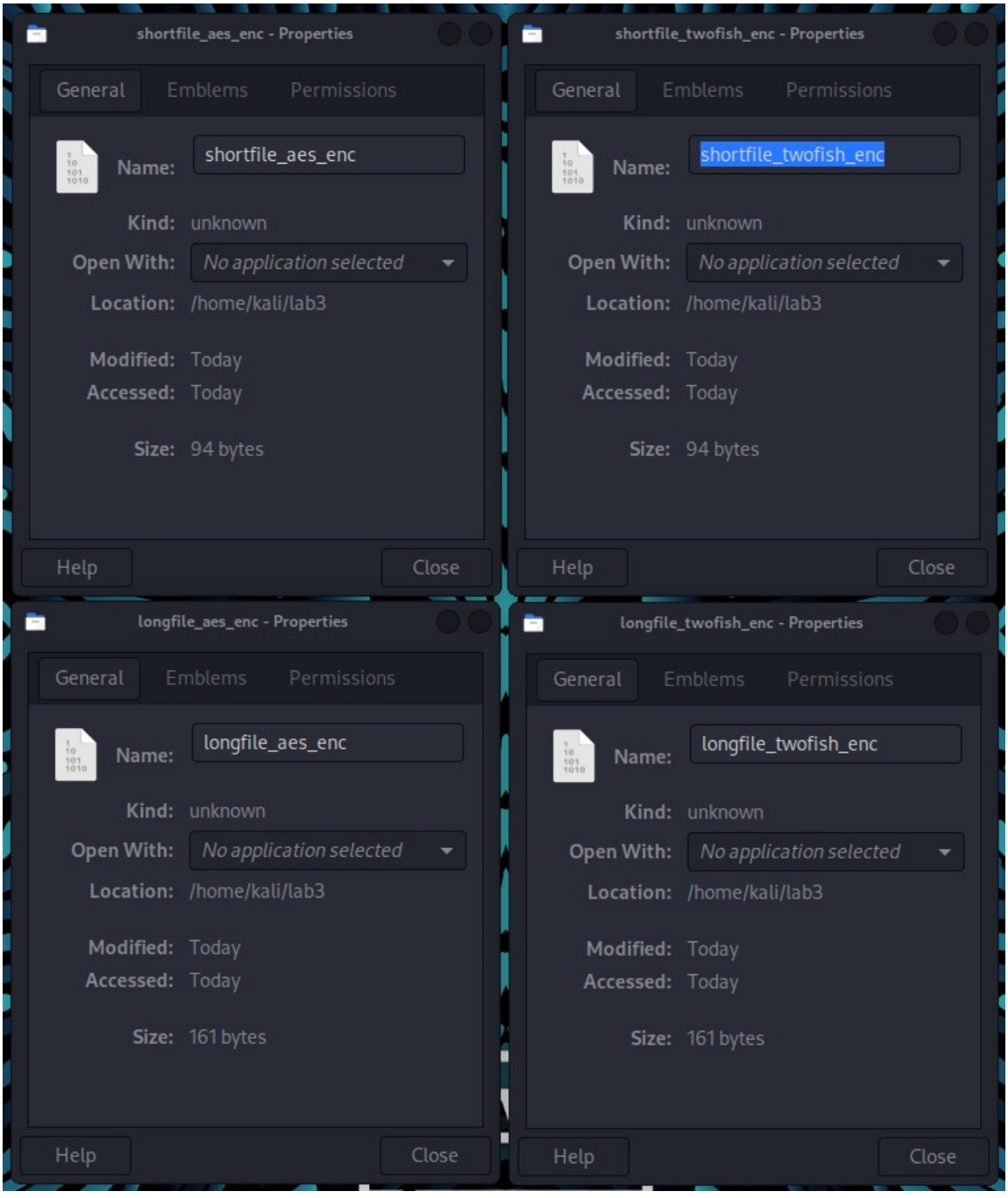



decbmp_twofish - Image Viewer [2/6]
File Edit View Go Help
decbmp_twofish 300 x 150 46.1 kB 100.0%

9)

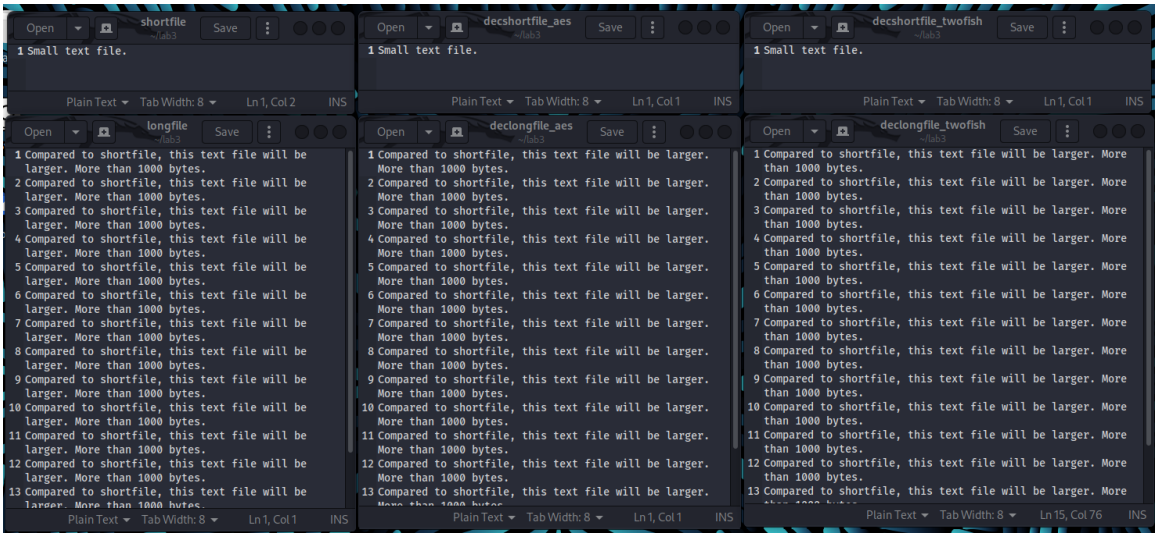

shortfile - Properties
General Emblems Permissions
Name: shortfile
Kind: plain text document
Open With: gedit
Location: /home/kali/lab3
Modified: Today
Accessed: Today
Size: 17 bytes
Help Close
longfile - Properties
General Emblems Permissions
Name: longfile
Kind: plain text document
Open With: gedit
Location: /home/kali/lab3
Modified: Today
Accessed: Today
Size: 1.1 KiB (1,140 bytes)
Help Close

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o shortfile_aes_enc --symmetric --cipher-algo AES256 shortfile

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o longfile_aes_enc --symmetric --cipher-algo AES256 longfile

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o shortfile_twofish_enc --symmetric --cipher-algo TWOFISH shortfile

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o longfile_twofish_enc --symmetric --cipher-algo TWOFISH longfile
```



10)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o decshortfile_aes -d shortfile_aes_enc
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o declongfile_aes -d longfile_aes_enc
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o decshortfile_twofish -d shortfile_twofish_enc
gpg: TWOFISH.CFB encrypted data
gpg: encrypted with 1 passphrase

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o declongfile_twofish -d longfile_twofish_enc
gpg: TWOFISH.CFB encrypted data
gpg: encrypted with 1 passphrase
```



Part 1 Questions:

1) Please describe the differences between symmetric encryption, asymmetric encryption, and cryptographic hash algorithms. List at least three of each supported by the gpg tool in Kali Linux.

Symmetric encryption and asymmetric encryption are means of sending a message between two parties that is unreadable to outside groups, whereas cryptographic hash algorithms are a one-way operation that acts as a signature that proves the integrity of a message. Symmetric encryption uses the same key for both encryption and decryption, however asymmetric encryption uses one key for encrypting the data and a different key for decryption the data. The gpg tool in Kali Linux supports (but is not limited to) 3DES, AES256, and TWOFISH for symmetric encryption, RSA, ELG, and DSA for asymmetric encryption, and SHA1, SHA256, and SHA512 for cryptographic hashing.

2) When you encrypt the bmp file, what is the size of the encrypted files using AES256 and using TWOFISH? Are they the same? Are they larger or smaller than the original file? Why?

AES256 and TWOFISH both are 14.5kB in size after the header is added (14.1kiB before), and significantly smaller than the original 45kiB file. however the images are mostly black, with what looks like static on the lower third of the image. AES256 shows more of a rainbow pattern, whereas TWOFISH has more green in the picture. The files are smaller because they no longer contain the individual pixels of color and are just a string of binary data that can be decrypted into an image file.

When viewed in the File Manager, the encrypted files have an icon showing binary, which I'm guessing means the file is only binary data, however when the header is added then the preview of the mostly black image with static at the bottom is seen.

3) When you encrypt the text file, what is the size of the encrypted file using AES256 and using TWOFISH respectively? Are they the same? Are they larger or smaller than the original file?

When the text files are encrypted, the small file that is originally 17 bytes becomes 94 bytes when encrypted with AES256 and with TWOFISH and the encrypted files are larger than the original. The long file that is originally 1.1kiB, however, becomes 161 bytes when encrypted with AES256 and TWOFISH and is smaller once encrypted.

4) Briefly describe the differences between AES256 and TWOFISH. Which one do you think is more secure based on your experimental results?

While both AES256 and TWOFISH are symmetric key encryption algorithms that operate on fixed-size blocks as they work, they differ in the number of rounds that they perform when encrypting the data. AES256 performs 10 rounds for a 128 bit key, 12 rounds for a 192 bit key, and 14 rounds for a 256 bit key, whereas TWOFISH performs 16 rounds for a 128 bit key, 16 or 20 rounds for a 192 bit key, and 16, 20, or 24 rounds for a 256 bit key. I think that TWOFISH is a more secure algorithm due to the higher number of rounds that it can perform on the data that is being encrypted.

5) You will need to use a passphrase to generate a symmetric key for the encryption and decryption. Compare different passphrases (e.g. shorter and weaker vs longer and more complex) and state your findings.

When testing two passwords, one shorter and weaker and another longer and more complex, the resulted encrypted file did not appear to change the outcome of the encrypted file. On the long text file, both the short password and long password encryptions are 161 bytes in size, despite one of them having a password more than twice as long and with special characters.

Part 2 – Asymmetric Encryption

1)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --gen-key
gpg (GnuPG) 2.2.35; Copyright (C) 2022 g10 Code GmbH
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 generation dialog.

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

Real name: Ryan Christopher
Email address: rechris@bu.edu
You selected this USER-ID:
    "Ryan Christopher <rechris@bu.edu>"
```

```
pub   rsa3072 2025-03-28 [SC] [expires: 2027-03-28]
      5A55934B2D23F1EE9A063F99B76E5CEDE9DAA9DB
uid                      Ryan Christopher <rechris@bu.edu>
sub   rsa3072 2025-03-28 [E] [expires: 2027-03-28]
```

2)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --list-key
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2027-03-28
/home/kali/.gnupg/pubring.kbx
-----------------------------
pub   rsa3072 2025-03-28 [SC] [expires: 2027-03-28]
      5A55934B2D23F1EE9A063F99B76E5CEDE9DAA9DB
uid           [ultimate] Ryan Christopher <rechris@bu.edu>
sub   rsa3072 2025-03-28 [E] [expires: 2027-03-28]
```

3)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --export -a "rechris@bu.edu" > RyanChristopher_public.key

┌──(kali㉿kali)-[~/lab3]
└─$ ls
decbmp                header                    pic_twofish_enc
decbmp_twofish        longfile                  pic_twofish_enc_withheader.bmp
declongfile_aes       longfile_aes_enc          RyanChristopher_public.key
declongfile_twofish   longfile_twofish_enc      shortfile
decshortfile_aes      pic_aes_enc               shortfile_aes_enc
decshortfile_twofish  pic_aes_enc_withheader.bmp  shortfile_twofish_enc
encpic                pic_original.bmp
```

4)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --import metcs695_lab3_asc.asc
gpg: key A0979DC5672379B8: public key "Nicklos See <nsee@bu.edu>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

5)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --fingerprint nsee@bu.edu
pub   rsa3072 2023-01-31 [SC]
      7A7C 6E77 B0DF 00A2 7E0A  B4A3 A097 9DC5 6723 79B8
uid           [ unknown] Nicklos See <nsee@bu.edu>
sub   rsa3072 2023-01-31 [E]
```

6) I did not receive an encrypted message. :(

7)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --encrypt --sign --armor -r "nsee@bu.edu" messagetoinstructor.txt
gpg: DF606DF94430D3F2: There is no assurance this key belongs to the named user

sub  rsa3072/DF606DF94430D3F2 2023-01-31 Nicklos See <nsee@bu.edu>
 Primary key fingerprint: 7A7C 6E77 B0DF 00A2 7E0A  B4A3 A097 9DC5 6723 79B8
      Subkey fingerprint: 98D4 635F CEEA D54A 74F1  7D80 DF60 6DF9 4430 D3F2

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

8)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --encrypt --sign -a -r "rechris@bu.edu" pic_original.bmp
```

9)

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --output pic_enc_nocompress.asc --encrypt --compress-algo=none --sign --armor
 -r "rechris@bu.edu" pic_original.bmp
```

Part 2 Questions

1) Check the manual of the gpg command using either the man command or the "--help" option and explain the following options:

a) --encrypt
encrypts the input data specified

b) --decrypt
decrypts the input data specified with a default method

c) --sign
creates a signature on the input data specified with a cryptographic hash function

d) --armor
creates an ascii armored output (a non-binary encoding)

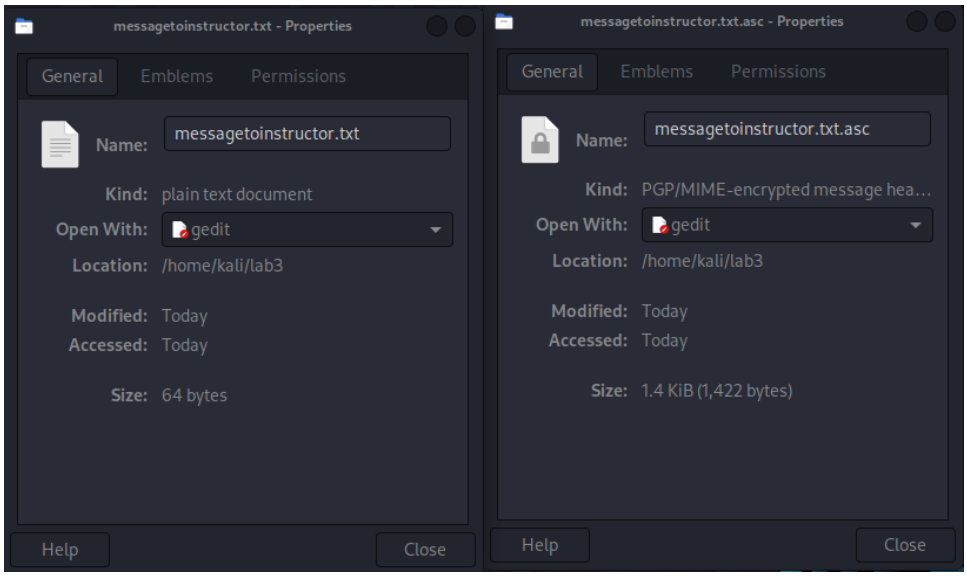
e) --output
write the output to a specified file

f) -r
specifies the recipient USER-ID when encrypting input data

2) Gpg uses a hybrid approach to encrypt the message, which uses the public key to encrypt a session key and uses the session key to encrypt the message. Explain this in more detail about how the message you want to send to the instructor is encrypted and signed as well as the decryption process.

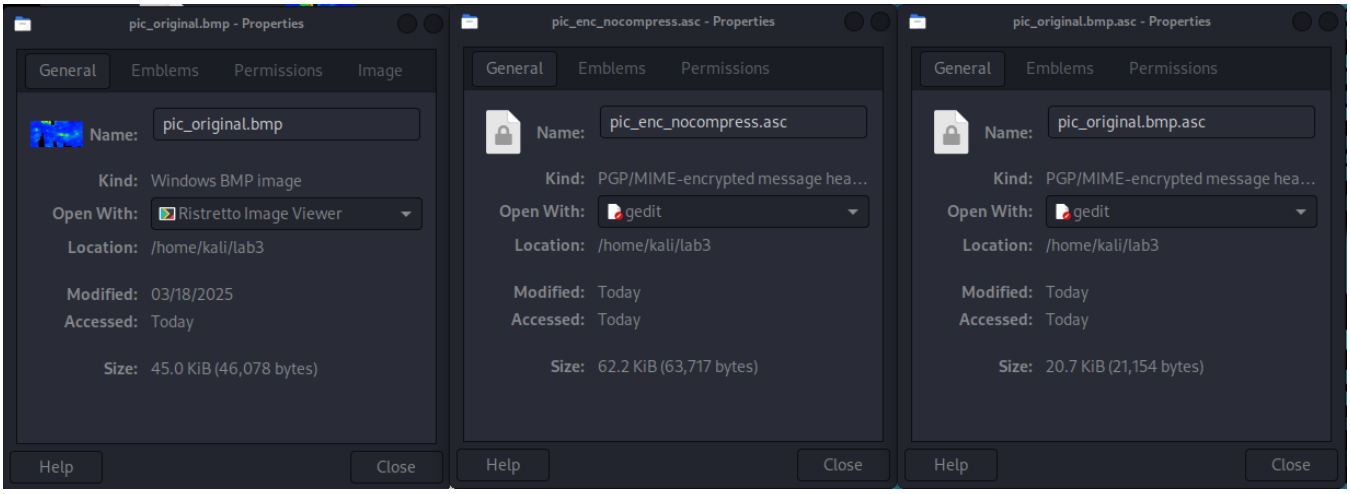
When you send an encrypted message, the random number is generated and used as the session key for this specific message only and encrypted using the recipients public key. Once the session key is encrypted and placed at the beginning of the message, the session key is used to encrypt the message. Then, when the recipient goes to decrypt the message, their private key decrypts the session key and uses the decrypted session key to decrypt the message.

3) Compare the size of the original text message and the encrypted & signed message, and state your findings and reasoning.



The original text message is 64 bytes and the encrypted and signed message is 1.4kiB, the file is large due to the larger size of the message once it is encrypted as well as the signature being included in the file.

4) Compare the file size of the original bmp file and encrypted bmp file with and without compression respectively, state your findings and reasoning.



The original bmp file is 45kiB, whereas the encrypted bmp file with compression is 20.7kiB and the encrypted bmp file without compression is 62.2kiB. The file with compression is logically smaller than the original, however without compression and the signature, the uncompressed file is about 50% larger as a result.

Reflection

a) What is the purpose of the lab in your own words?

The purpose of this lab is to gain experience with encrypting different types of files (image and text) with symmetric and asymmetric encryption methods, as well as decrypting files with symmetric and asymmetric keys to gain the original.

b) What did you learn? Did you achieve the objectives?

I learned how to generate a private/public key, as well as how to use gpg in order to encrypt files with different methods. I would say that I achieved all of the objectives except for decrypting the instructor message as I did not receive one.

c) Is this lab hard or easy? Are the lab instructions clear?

This lab was fairly straightforward, and this is mostly due to the clear instructions in the lab. Without the instructions I would have guessed this lab to be much more difficult, however with the explanations of each step I was able to follow along fairly easily.

d) What do you think about the tools used? What worked? What didn't? Are there other better alternatives?

I am particularly interested in gpg now, as I did not realize that encrypting files and generating private/public keys were performed so simply. I did not encounter any issues, so I would say that everything worked as expected. My guess is that there are more modern alternatives to encryption methods, as the list of "supported" algorithms on gpg did not include ones that I have seen in more recent discussions, however I am guessing that they can accomplish essentially the same result as gpg does.

e) Other feedback

Getting more exposure to the process of encrypting and decrypting data was, in my opinion, a valuable lab as we discuss these actions heavily in classes but I myself have not had much experience doing so. After this lab, I am curious to start experimenting with other methods of encryption, and especially seeing what parts of the processes in this lab have correlation with my project topic.