BOSTON UNIVERSITY METROPOLITAN COLLEGE

MET CS 695S O2 REG Cybersecurity (2024 Summer 2)

Lab 3 Encryption with GPG

Submitted to

Nicklos See, M.S.,

Warren Mansur, M.S.,

Shengzhi Zhang, Ph.D.

MET CS695S O2 REG

Cybersecurity

by

Michael G Nguyen

7/19/2024

Table of Contents

Part 1: Symmetric Encryption

1. Create a folder named “lab3”, and change your current work directory into lab3.

```
┌──(kali㉿kali)-[~]
└─$ mkdir lab3

┌──(kali㉿kali)-[~]
└─$ cd lab3
```

Mkdir lab3 makes the directory of lab3 and cd changes the directory to lab3.

2. Check the gpg version using the “--version” option. Find all supported algorithms, answer Question 1 in the Question section.

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --version
gpg (GnuPG) 2.2.40
libgcrypt 1.10.3
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
```

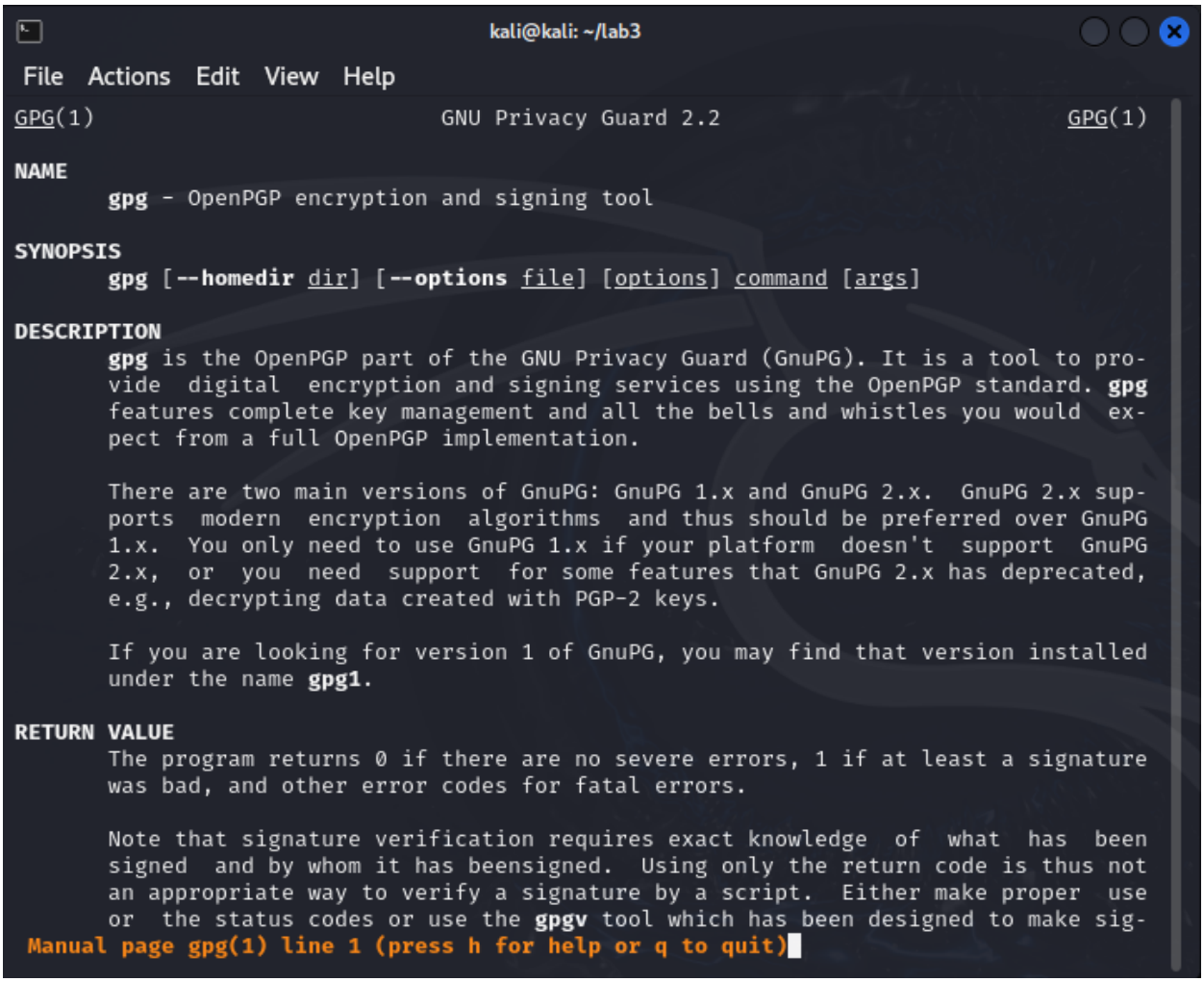
This command checks the Gpg2 version of OpenPGP standard.

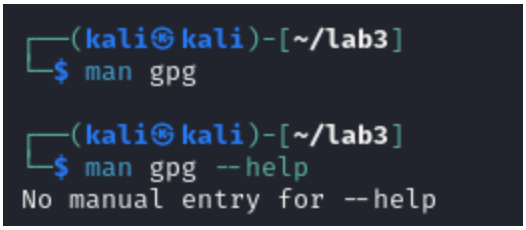
3. Find where the gpg is installed using the “whereis” command:

```
┌──(kali㉿kali)-[~/lab3]
└─$ whereis gpg
gpg: /usr/bin/gpg /usr/share/man/man1/gpg.1.gz
```

This command checks where is gpg being located.

4. Check the manual of the gpg tool using either the “man” command or “--help”option.





Man is used to check for the user manual as well as help.

a. What is the option used to encrypt using symmetric algorithm?

In cryptography, symmetric encryption algorithm uses one key for encryption and decryption and has the key length of 128 or 256 bits. Moreover, symmetric encryption algorithms include Advanced Encryption Standard (AES), Data Encryption Standard (DES), Triple Data Encryption Standard (3DES), and Rivest Cipher 4 (RC4).

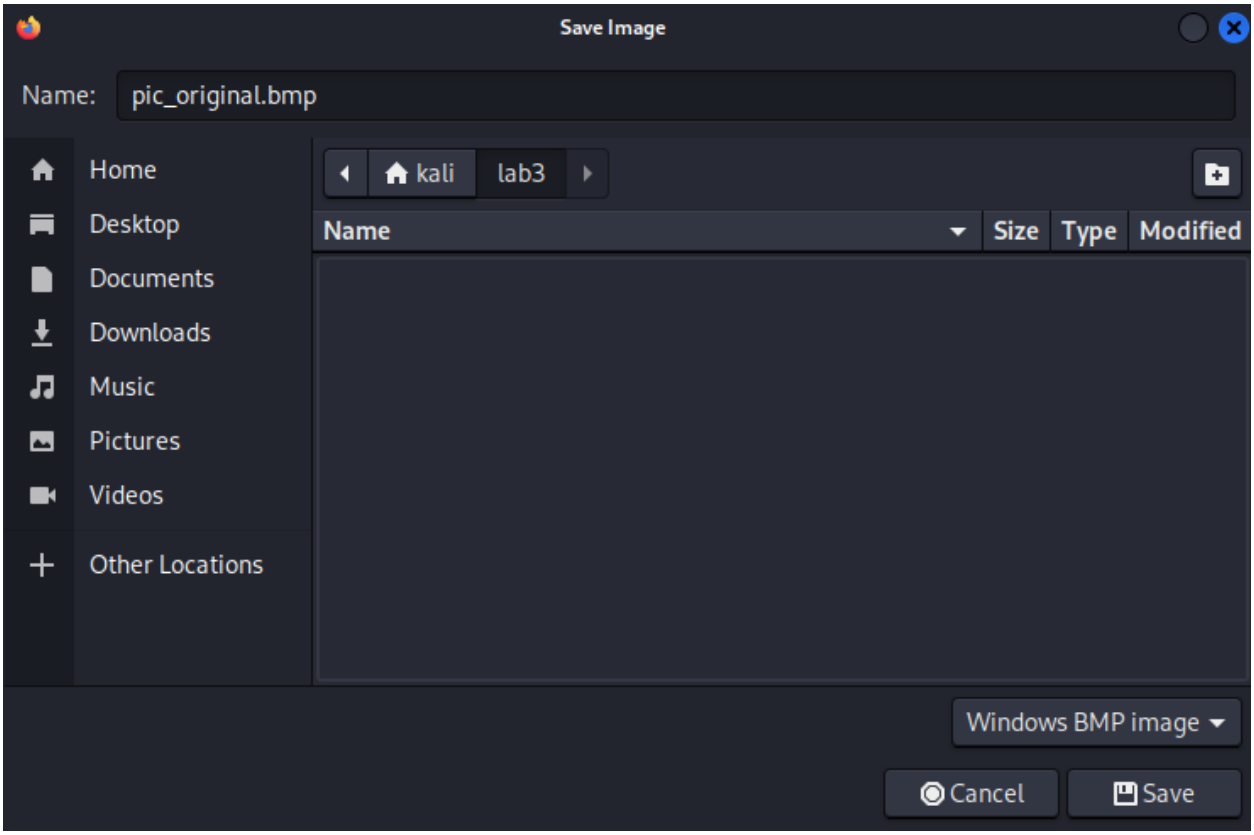
b. What is the option used to specify different encryption algorithms?

In cryptographic libraries and tools, the option to specify different encryption algorithms is typically provided through the API or command-line interface. The specific method depends on the library or tools being used. There are many examples of symmetric encryption algorithms, below are what's included:

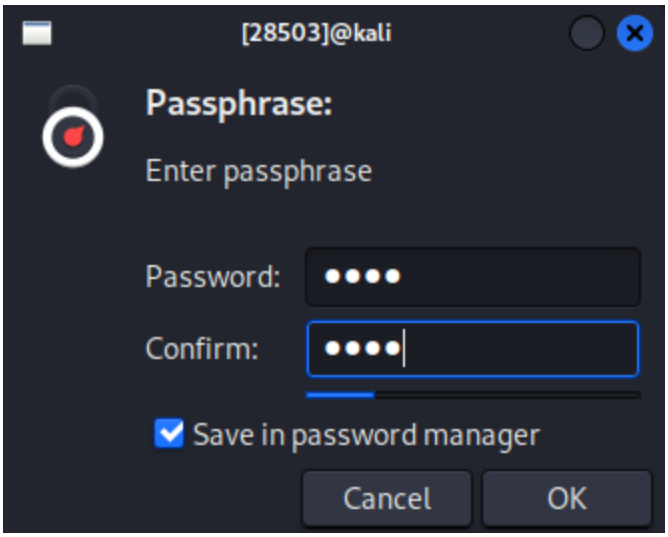
- Blowfish (Drop-in replacement for DES or IDEA) – block cipher
- IDEA (International Data Encryption Algorithm) – block cipher
- AES (Advanced Encryption Standard) – block cipher
- DES (Data Encryption Standard) – stream cipher
- RC6 (Rivest Cipher 6) – block cipher
- RC5 (Rivest Cipher 5) – block cipher
- RC4 (Rivest Cipher 4) – stream cipher

However, AES is the most used symmetric algorithm, AES was known initially by Rijndael, which replaced DES in the 70s. Moreover, AES cipher has block size of 128 bits and can be used with three key lengths, specifically for AES-128, AES-192, and AES-256.

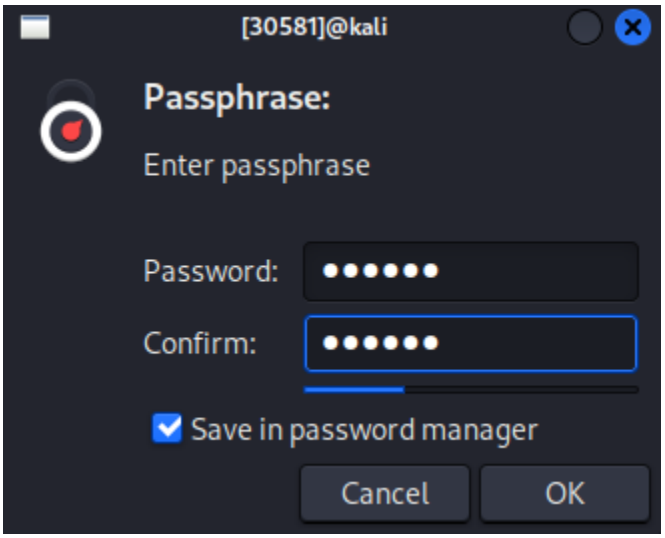
5. Open Firefox web browser (https://www.fileformat.info/format/bmp/sample/index.htm), download a bmp file to your computer and encrypt the file using both AES256 and TWOFISH. Change the name of the file you downloaded to pic_original.bmp and copy it into the lab3 folder. The lab3 folder can be opened using the File Manager tool.



Saving the file as pic_original.bmp.



Saving the password and confirming password as *kali*.



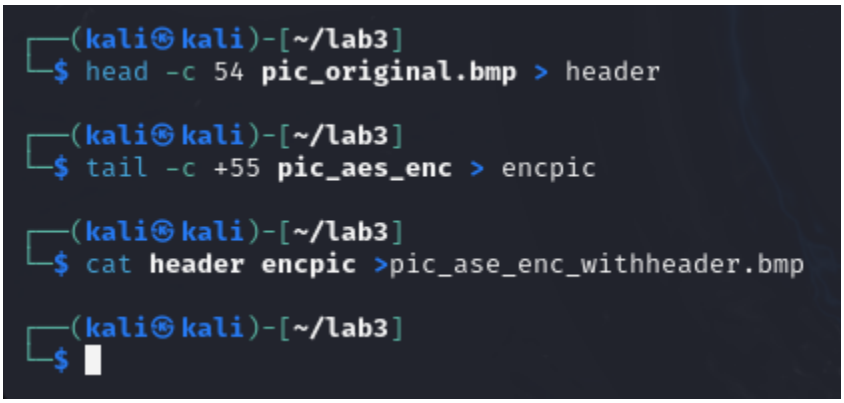
Using TWOFISH saving the password as fornia.

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

┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o pic_aes_enc --symmetric --cipher-algo TWOFISH pic_original.bmp
File 'pic_aes_enc' exists. Overwrite? (y/N)
```

Overwriting the existing file with a yes. Therefore, the password as of now would be *fornia* in TWOFISH encryption.

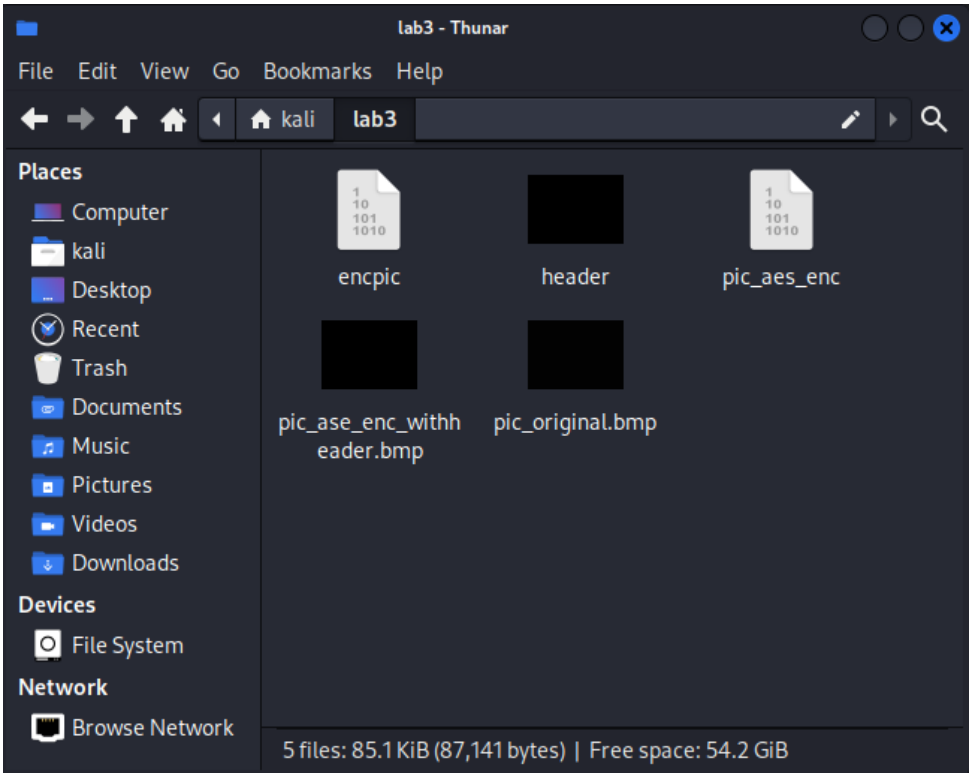
6. Since the file header is also encrypted, we will not be able to open the encrypted file as a bmp file. We can replace the header of the encrypted picture with that of the original picture so that we can treat it as a legitimate .bmp file. For the .bmp file, the first 54 bytes contain the header information about the bmp file. We can get this header from the original file and then the rest of

the data from the encrypted picture to reconstruct a bmp file. The following screenshot shows how to get the header of the original file using the “head” command and the body of the encrypted file using the “tail” command, then concatenate these two to reconstruct a new encrypted bmp file. You will do the same thing with the twofish-encrypted image.



I think these commands reconstruct the head, body and tail of the .bmp file and how it is being displayed. When a .bmp file is encrypted using a symmetric encryption algorithm like TWOFISH, the entire file, including the header is encrypted. As a result, the encrypted file loses its recognizable .bmp structure, making it unreadable by image viewing software.

7. Open the lab3 folder using the File Manager tool as in Step 5, and view files as icons. Also try to view the encrypted picture in an image viewer. What are your findings? Answer Question 2 in the Question section.

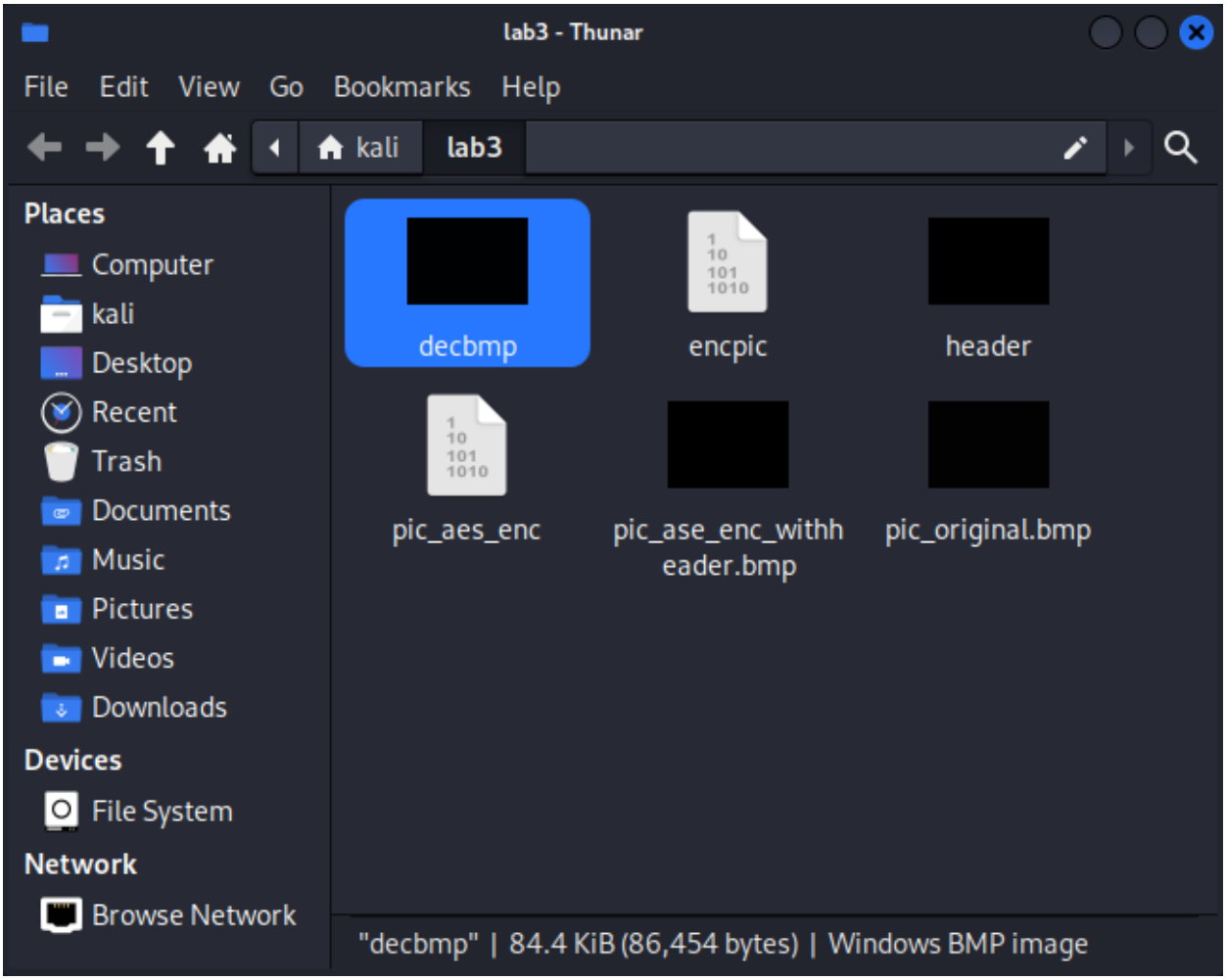


It appears to be black in three files, which seems to be correct, however, two file are in random bytes with unrecognizable file header and randomized file content. As a result of the encryption, the file manager display the encrypted image file with a generic or unknown file type icon. This is because the system cannot determine the file type and therefore falls back to a default icon used for unidentified or binary files.

8. Decrypt your encrypted files (not reconstructed bmp file) and see if it is the same as the original file.

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

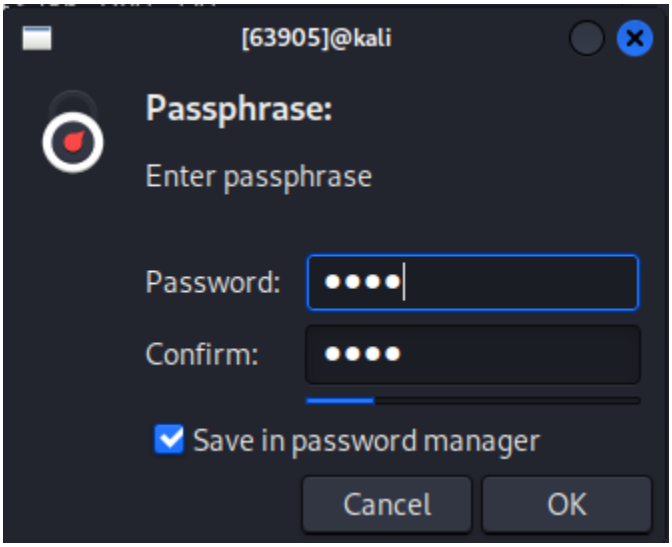
Decrypting the encrypted .bmp file to check the differences.



There is one more image then the previous and it looks correct a black .bmp file.

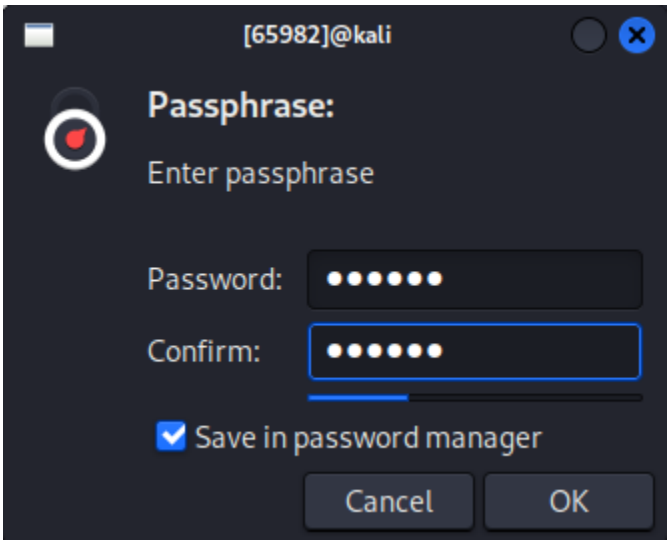
9. You can use the “cat” command or a text editor to create a short text file with less than 20 bytes and a large text file with more than 1000 bytes. You can fill in any data you want to the files. Save

them in the folder of lab3. Encrypt both files using both AES256 and TWOFISH. What are your findings? Answer Question 3 in the Question section.




```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o textAES --symmetric --cipher-algo AES256 text
gpg: WARNING: server 'gpg-agent' is older than us (2.2.40 < 2.2.43)
gpg: Note: Outdated servers may lack important security fixes.
gpg: Note: Use the command "gpgconf --kill all" to restart them.
```

I could not type textAES.txt or text.txt within the command for the file to work, however, without it, it'll work. Also, the password is *kali.*



For TWOFISH the password is *fornia.*

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o textTWOFISH --symmetric --cipher-algo TWOFISH text
gpg: WARNING: server 'gpg-agent' is older than us (2.2.40 < 2.2.43)
gpg: Note: Outdated servers may lack important security fixes.
gpg: Note: Use the command "gpgconf --kill all" to restart them.
```

Encryption using TWOFISH.

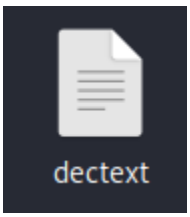
```
┌──(kali㉿kali)-[~/lab3]
└─$ head -c 54 text > header

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

┌──(kali㉿kali)-[~/lab3]
└─$ cat header textAESenc > textAESencwithheader
```

Changing head with header command and body with tail command.

I found that specifically, the text file wasn't able to type in the command with the extension of the file. The original file that I've created was only 80 bytes in *text* format, compare to *textAES*, which is 131 bytes. *TextAESenc* turned less than anticipated, which is 77 bytes, and *textAESencwithheader* is a little more of 131 bytes.

10. Decrypt your encrypted files and see if it is the same as the original file.


```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg -o dectext -d textAES
gpg: AES256.CFB encrypted data
gpg: WARNING: server 'gpg-agent' is older than us (2.2.40 < 2.2.43)
gpg: Note: Outdated servers may lack important security fixes.
gpg: Note: Use the command "gpgconf --kill all" to restart them.
gpg: encrypted with 1 passphrase
File 'dectext' exists. Overwrite? (y/N) y
```



It didn't provide the *textAES* original file of text, but rather, a different file in the same form called *dectext*, which is the same *text* file created.

11. Compare AES256 and TWOFISH and answer Question 4 in the Question section.

The comparison between AES-256 and TWOFISH depends on specific requirements and context.

Standardization and Adoption:

AES-256 is the more widely adopted and standardized encryption algorithm, being the official standard for government and commercial applications. Its extensive use and support in modern hardware make it a go-to choice for most encryption needs.

Performance:

AES-256 generally offers better performance, especially on hardware-accelerated platforms. Its integration into modern processors and encryption hardware provides significant speed advantages.

Security:

Both AES-256 and TWOFISH provide robust security. AES-256's longer key size offers enhanced protection against brute-force attacks. TWOFISH's design incorporates extensive security features to defend against cryptanalytic attacks, but its security is comparable to AES-256 when using a 256-bit key.

Flexibility:

TWOFISH's flexibility in key sizes and its optimization for various hardware architectures make it a versatile choice, particularly in environments where hardware acceleration is not available.

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:

Symmetric encryption uses the same key for both encryption and decryption. It is generally faster and suitable for encrypting large amounts of data. Examples supported by GPG in Kali Linux include:

- AES (Advanced Encryption Standard)
- TWOFISH

Asymmetric Encryption:

Asymmetric encryption uses a pair of keys, one for encryption (public key) and one for decryption (private key). It is commonly used for secure key exchange and digital signatures. Examples supported by GPG in Kali Linux include:

- RSA (Rivest-Shamir-Adleman)
- DSA (Digital Signature Algorithm)
- ElGamal

Cryptographic Hash Algorithms:

Cryptographic hash algorithms generate a fixed-size hash value from input data, which is unique to the original data. They are used for data integrity verification and digital signatures. Examples supported by GPG in Kali Linux include:

- SHA-256 (Secure Hash Algorithm 256-bit)
- SHA-1
- MD5 (Message-Digest Algorithm 5)

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?

When encrypting a BMP file with AES-256 and TWOFISH, the sizes of the encrypted files are typically larger than the original file. This is because of padding and additional metadata used in the encryption process.

- AES-256 and TWOFISH Encrypted BMP File Size:

  The size of the encrypted files is generally the same regardless of the algorithm used (AES-256 or TWOFISH), and both are slightly larger than the original file due to padding added to align the data blocks to the cipher's block size and encryption overhead.

- Reason for Size Increase:

  The increase in size is due to the padding required to make the data fit into the block size of the encryption algorithm (e.g., 16 bytes for AES) and the inclusion of encryption metadata such as the initialization vector (IV) and salt.

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 encrypting a text file with AES-256 and TWOFISH, the sizes of the encrypted files also tend to be larger than the original file.

- AES-256 and TWOFISH Encrypted Text File Size:

  Similar to BMP files, the sizes of encrypted text files using AES-256 and TWOFISH are usually the same and larger than the original file due to padding and encryption metadata.

- Reason for Size Increase:

  The padding ensures that the plaintext fits the block size of the encryption algorithm, and additional metadata like IV and salt contribute to the increased size.

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

Differences:

- Origin:
    - AES-256: Selected as the standard by NIST and designed by Joan Daemen and Vincent Rijmen.
    - TWOFISH: Designed by Bruce Schneier and team, was a finalist in the AES competition but not selected as the standard.
- Algorithm Structure:
    - AES-256: Block cipher with a 128-bit block size, using a substitution-permutation network.
    - TWOFISH: Feistel network with a 128-bit block size, incorporating S-boxes, MDS matrices, and PHT operations.
- Performance:
    - AES-256: Faster in hardware-accelerated environments due to widespread support.
    - TWOFISH: Optimized for 32-bit and 8-bit architectures, performs efficiently in software.

Security Comparison: Based on experiment results, both AES-256 and TWOFISH provide robust security. However, AES-256 benefits from more extensive analysis and standardization, making it a more widely trusted and adopted choice.

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 complexer) and state your findings.

Shorter and Weaker Passphrases:

- Easier to remember but more vulnerable to brute-force attacks.
- Typically less secure due to the limited number of possible combinations.

Longer and Complex Passphrases:

- More secure due to the increased number of possible combinations.
- Harder to remember but significantly increases the time required for brute-force attacks.

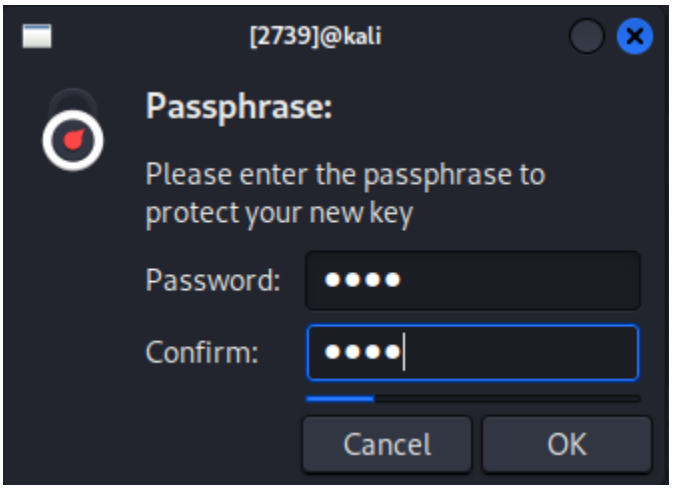
Findings:

- Longer and more complex passphrases offer substantially better security.
- The use of a passphrase manager can help manage complex passphrases without sacrificing usability.

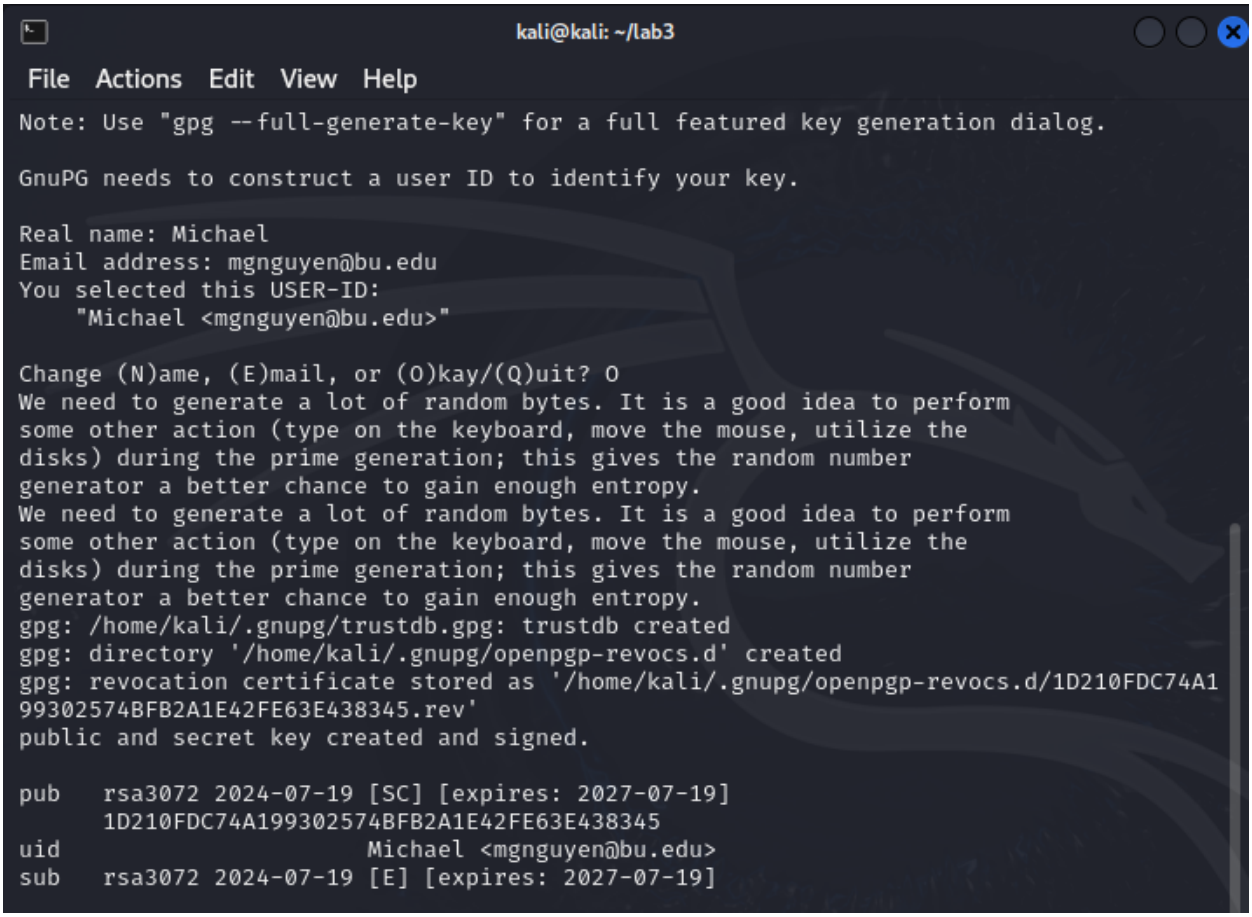
Part 2: Asymmetric Encryption

1. To use asymmetric encryption, we need to first create a key pair. You will be prompted to enter your real name and an email address which identifies you. Then you need to enter a passphrase to

generate the public and private key pair for you. It will also sign the keys. The public key information will be displayed as the output. You can use "-- genkey" option which uses RSA and sets the key length as 3072 bits. You can also use " -- full-generate-key" option to customize the algorithm and the key length if you want.



Passphrase is *kali.*



I used gpg –gen-key to generate a key.

2. After you generate the key pair, you can also use "-- list-keys" to display the public keys you have.

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --list-keys
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 2027-07-19
/home/kali/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072 2024-07-19 [SC] [expires: 2027-07-19]
      1D210FDC74A199302574BFB2A1E42FE63E438345
uid           [ultimate] Michael <mgnguyen@bu.edu>
sub   rsa3072 2024-07-19 [E] [expires: 2027-07-19]
```

3. You can export your public key to a file named xxx_public.key, and provide it to others. (Substitute the "xxx@bu.edu" with your own email address. That is uid of your key. Substitute "xxx" in the "xxx_public.key" with your own name.) Send your public key file to your instructor as early as possible and submit your public key file on blackboard.

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

Exporting the public key with mgnguyen@bu.edu email and uid as mgnguyen.

4. Import the instructor (or the facilitator's) public key (the other file of Assignment 3) into your key store. Then use "-- list-key" option to check if the imported key is there.

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --list-keys
/home/kali/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072 2024-07-19 [SC] [expires: 2027-07-19]
      1D210FDC74A199302574BFB2A1E42FE63E438345
uid           [ultimate] Michael <mgnguyen@bu.edu>
sub   rsa3072 2024-07-19 [E] [expires: 2027-07-19]

pub   rsa3072 2023-01-31 [SC]
      7A7C6E77B0DF00A27E0AB4A3A0979DC5672379B8
uid           [ unknown] Nicklos See <nsee@bu.edu>
sub   rsa3072 2023-01-31 [E]
```

Importing metcs695_lab3_asc.asc keys with gpg –-import and listing the keys with –-list-keys.

5. You can verify if the key is authentic by comparing the fingerprint generated with the provided fingerprint. (substitute "instructoremail@bu.edu with real email of your instructor or facilitator)

```
┌──(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]
```

This command verifies the public key encryption to see who it belongs to, which is Professor See.

6. Decrypt the encrypted message that the instructor (or the facilitator) sent to you. In the example below, message.txt.asc is the encrypted message you got from the instructor.

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --output message_decrypted.txt --decrypt message.txt.asc
gpg: encrypted with 3072-bit RSA key, ID CDB40746B668B4EC, created 2024-07-19
      "Michael <mgnguyen@bu.edu>"
```

This command decrypts Professor See's encrypted file!

```
Open   message_decrypted.txt   Save
       ~/lab3
1 Congratulations, Michael! If you can read this message, you successfully
  decrypted a file using your private key.
2
3  --Nicklos See
4
```

This is what's inside of the file!

7. Compose a new message to reply to the received message using any text editor and save it into a file, for example messagetoinstructor.txt, and then encrypt it and sign it. It will prompt you to input your passphrase that you use to generate your keys. Submit the encrypted message to the blackboard.

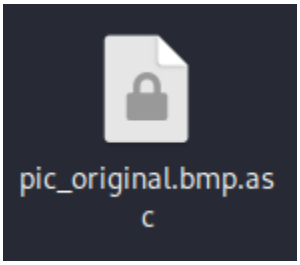
```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --encrypt --sign --armor -r nsee@bu.edu messagetoinstructor.txt
gpg: DF606DF94430D3F2: There is no assurance this key belongs to the named user

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
```

This command encrypts and sign the generated public key of gpg, but also the text file which was messagetoinstructor.txt.asc.

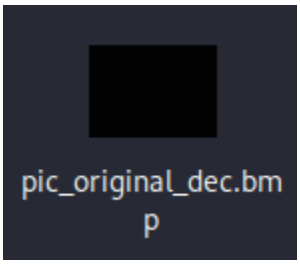
8. Use the “--encrypt” option to encrypt the bmp file in the first part and specify the receiver as yourself, and then decrypt it to see if you can get the same bmp file.

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --encrypt --sign --armor -a -r mgnguyen@bu.edu pic_original.bmp
File 'pic_original.bmp.asc' exists. Overwrite? (y/N) y
```

This command encrypts the pic_original.bmp file. Also, it does not have the same file. Below is the information.



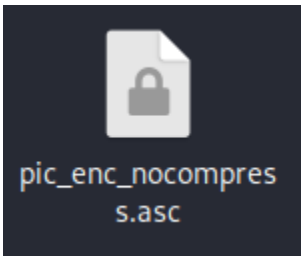

```
┌──(kali㉿kali)-[~/lab3]
└─$ gpg --output pic_original_dec.bmp --decrypt pic_original.bmp.asc
gpg: encrypted with 3072-bit RSA key, ID CDB40746B668B4EC, created 2024-07-19
      "Michael <mgnguyen@bu.edu>"
gpg: Signature made Sat 20 Jul 2024 06:54:43 PM EDT
gpg:                using RSA key 1D210FDC74A199302574BFB2A1E42FE63E438345
gpg: Good signature from "Michael <mgnguyen@bu.edu>" [ultimate]
```



Yes, the file remains the same!

9. Now use “--compress-algo=none” to disable compression in the encryption.

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



Using the command to disable compression in encryption file.

Questions:

1. Check the manual of the gpg command using either the man command or "--help" option and explain the following options: a. --encrypt b. --decrypt c. --sign d. --armor e. --output f. -r

To check the manual of the gpg command, you can use either man gpg or gpg --help. Here are the explanations for the specified options:

a. --encrypt: This option encrypts data to the specified recipients. It uses the recipient's public key to encrypt the data.

b. --decrypt: This option decrypts data. It uses the recipient's private key to decrypt the data that was encrypted with the corresponding public key.

c. --sign: This option signs the data with your private key. The signature can be used by others to verify that the data came from you and has not been altered.

d. --armor: This option creates ASCII armored output, which is suitable for inclusion in text files and emails. The output is base64 encoded and wrapped in a BEGIN/END block.

e. --output: This option specifies the output file. If this option is not used, the output will go to standard output.

f. -r: This is a shorthand for --recipient. It specifies the recipient for whom the data is being encrypted.

2. Gpg uses a hybrid approach to encrypt the message, which uses the public key to encrypt a session key and use 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. You can read the opengpg standard https://tools.ietf.org/html/rfc4880, particularly Section 2.

GPG uses a hybrid encryption approach. Here's a detailed explanation of how a message is encrypted and signed, as well as how it is decrypted:

Encryption and Signing:

- Create a session key: A random session key (symmetric key) is generated.
- Encrypt the message: The actual message is encrypted using this session key with a symmetric encryption algorithm.
- Encrypt the session key: The session key is then encrypted using the recipient's public key with an asymmetric encryption algorithm.
- Sign the message: The sender's private key is used to create a digital signature of the message. This ensures the message integrity and authenticity.
- Combine the encrypted session key and the encrypted message: Both the encrypted session key and the encrypted message (along with the signature) are combined into a single encrypted package.

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

When you encrypt and sign a text message, the size of the message generally increases due to the following reasons:

- Encryption Overhead: The encrypted message includes metadata, such as headers and the encrypted session key, which adds to the size.
- Signature Data: The digital signature appended to the message increases its size.

The encrypted and signed message is significantly larger due to the added headers, encryption, and signature data.

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

BMP files are uncompressed bitmap images, so when they are encrypted with GPG, the file size typically increases due to encryption overhead. However, GPG can compress data before encryption, which might reduce the file size.

Without Compression:

- Original BMP File: 84.4 KiB (pic_original.bmp)
- Encrypted BMP File: 115.6 KiB (pic_enc_nocompression.asc)

With Compression:

- Original BMP File: 84.4 KiB (pic_original.bmp) *without compression*
- Compressed & Encrypted BMP File: 1.6 KiB (pic_original.bmp.asc)

Findings:

- Without Compression: The encrypted file is larger due to the encryption overhead.
- With Compression: The file size can be reduced if the data is compressible. However, some data types (like already compressed files) may not show significant size reduction.

Bibliography

- Daemen, J., & Rijmen, V. *The Design of Rijndael: AES - The Advanced Encryption Standard. Springer.* (2001).
- Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., & Ferguson, N. *Twofish: A 128-Bit Block Cipher.* (1998). AES Candidate Algorithm Submission.
- National Institute of Standards and Technology (NIST). *Announcing the ADVANCED ENCRYPTION STANDARD (AES).* (2001). Federal Information Processing Standards Publication 197.
- Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* (1996). John Wiley & Sons.
- Wikipedia contributors. *Symmetric-key algorithm.* (2024, June 7). In *Wikipedia, The Free Encyclopedia*. Retrieved 22:51, July 19, 2024, from https://en.wikipedia.org/w/index.php?title=Symmetric-key_algorithm&oldid=1227660046
- Advanced Encryption Standard (AES) FIPS 197, Advanced Encryption Standard (AES) (nist.gov)
- "Symmetric Encryption Algorithms." *Bugcrowd*, www.bugcrowd.com/glossary/symmetric-encryption-algorithms/#:~:text=The%20symmetric%20encryption%20algorithms%20include,DES%2C%203DES%2C%20and%20RC4. Accessed 19 July 2024.
- Daemen, J., & Rijmen, V. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer. (2001).
- Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., & Ferguson, N. *Twofish: A 128-Bit Block Cipher*. (1998). AES Candidate Algorithm Submission.
- National Institute of Standards and Technology (NIST). *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. (2001). Federal Information Processing Standards Publication 197.
- Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. (1996). John Wiley & Sons.
- *GNU Privacy Guard (GPG) Documentation*. Retrieved from https://gnupg.org/documentation/
- GPG Manual: Retrieved from man gpg and gpg --help
- OpenPGP Standard (RFC 4880): Retrieved from RFC 4880