Lab 2

CSC154

Daniel Hammon

TTH 5:30pm

Jordan Penaloza

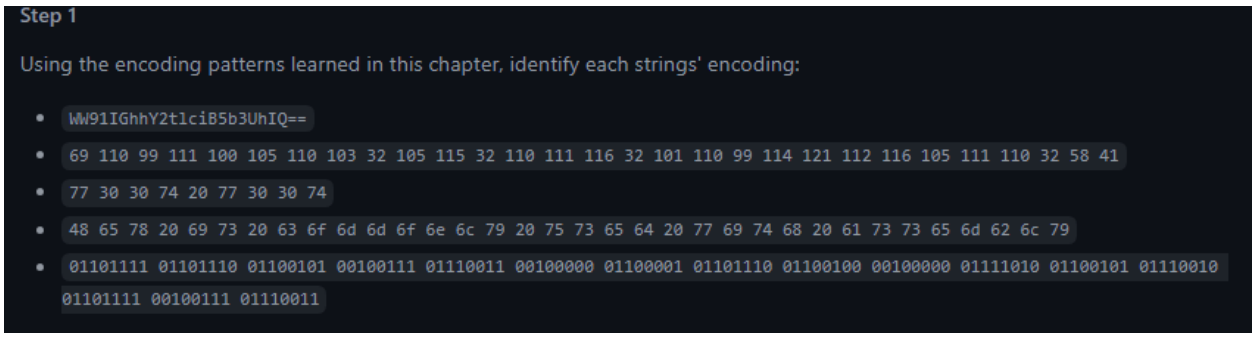
Lab 2 Overview:

In this lab we will be covering various cryptology topics found in the chapter 2 exercises of the textbook.

2.1 - Encoding and Decoding

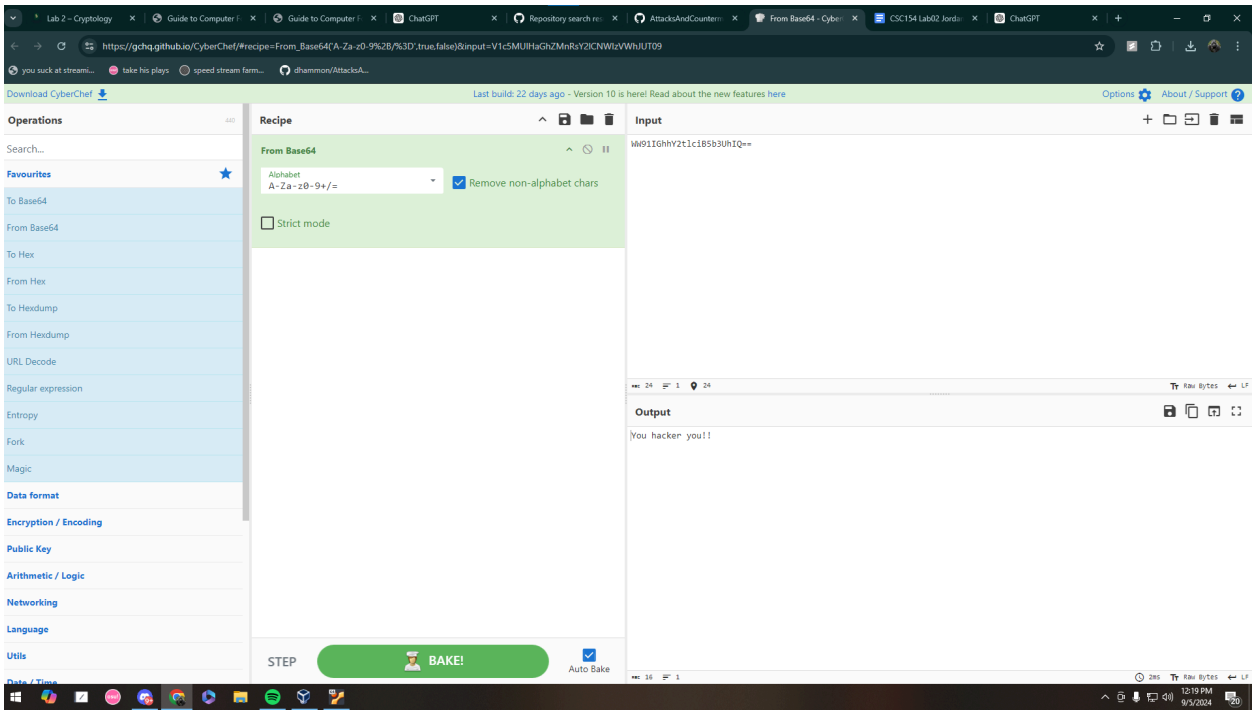
Type	Hallmarks	Example
Binary	ones and zeros	01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010 01101100 01100100 00100001
Decimal	numbers only	72 101 108 108 111 32 87 111 114 108 100 33
Hexadecimal	numbers and letters a-f	48 65 6c 6c 6f 20 57 6f 72 6c 64 21
Base64	numbers, upper and lowercase letters, and special characters +, /, and =	SGVsbG8gV29ybGQh

Here are the encoding examples in the textbook.

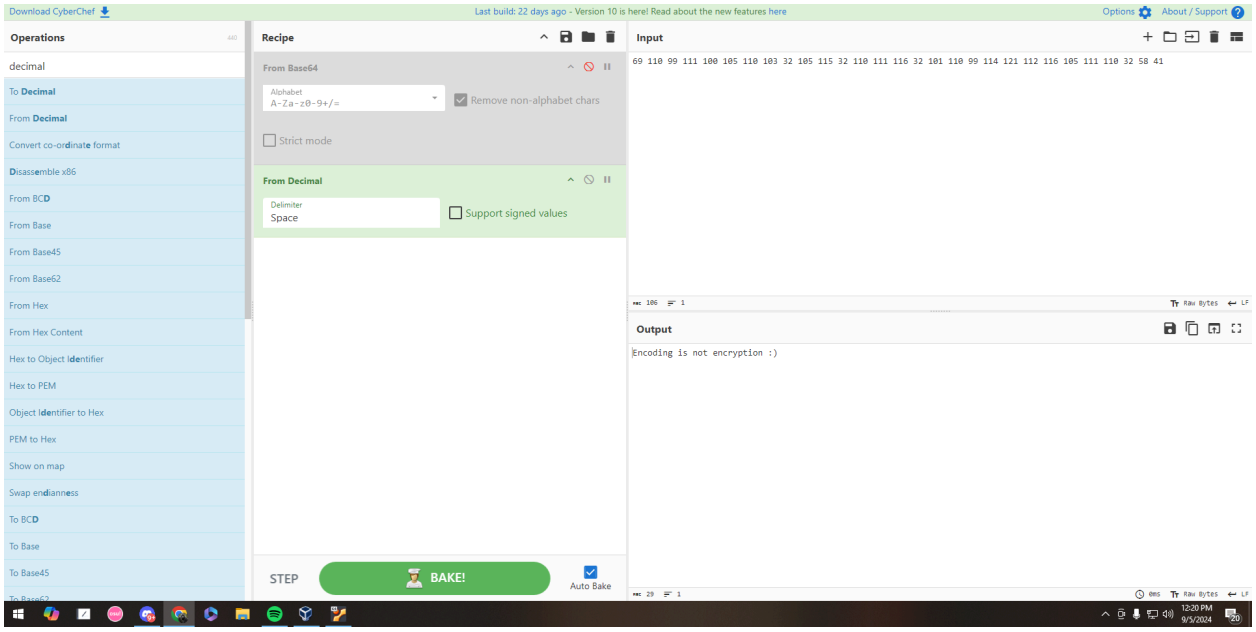


1. Base64
2. Decimal
3. Hexadecimal
4. Hexadecimal
5. Binary

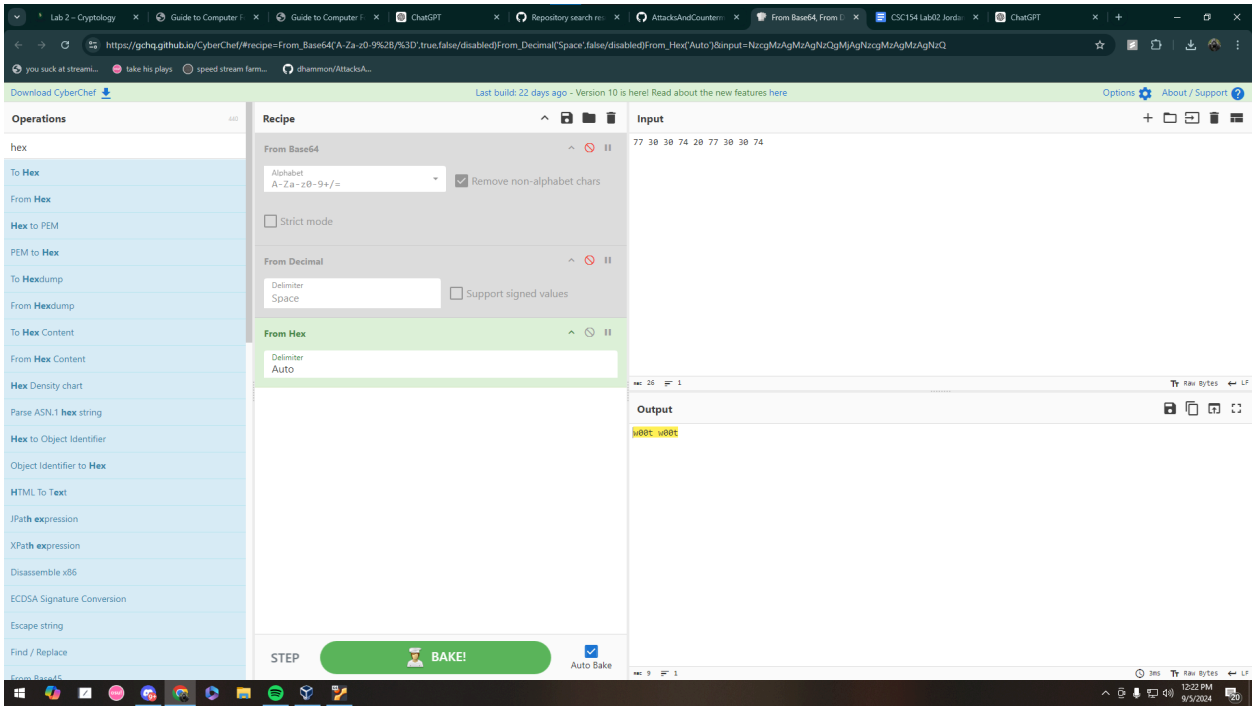
Step 2 was to decode each of these using cyberchef



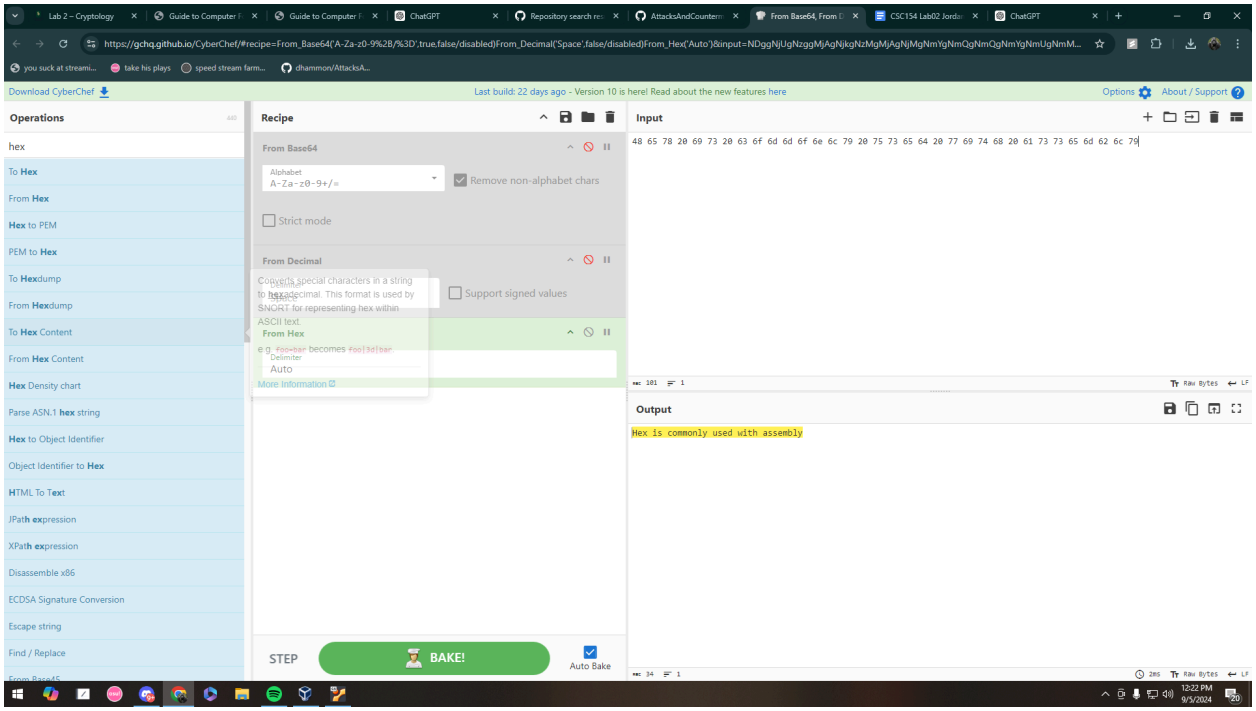
You hacker you!!



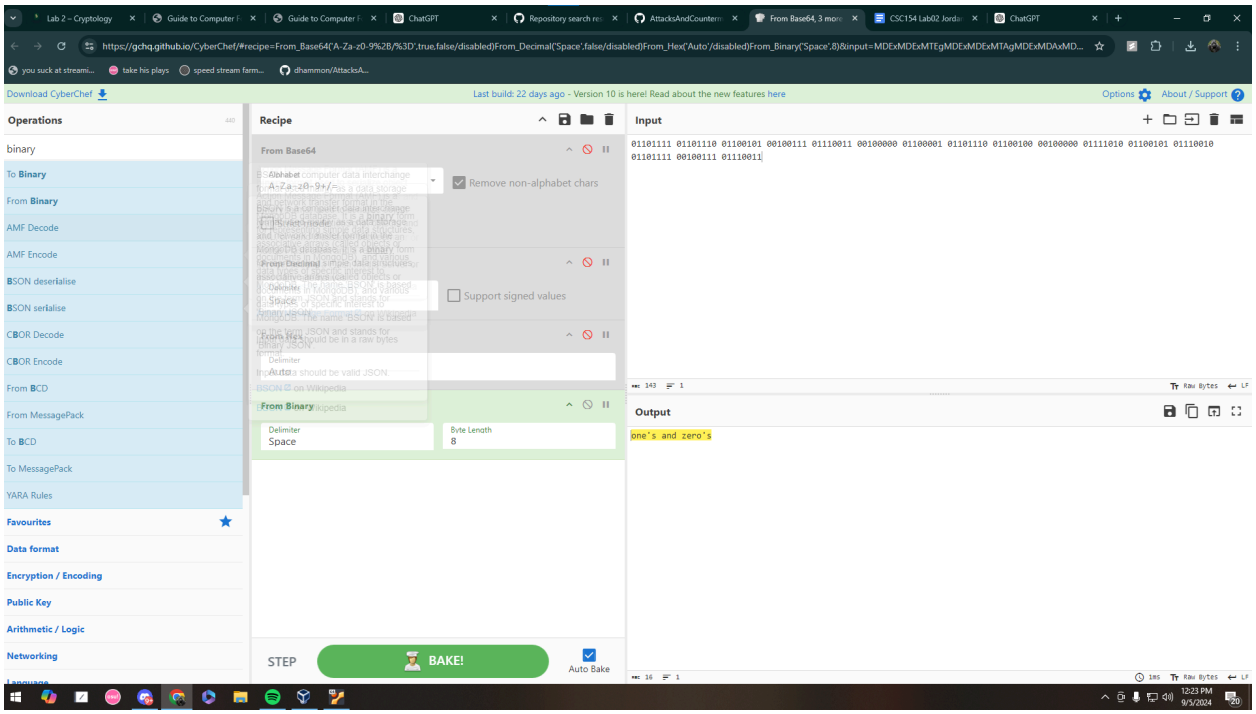
Encoding is not encryption :)



w00t w00t



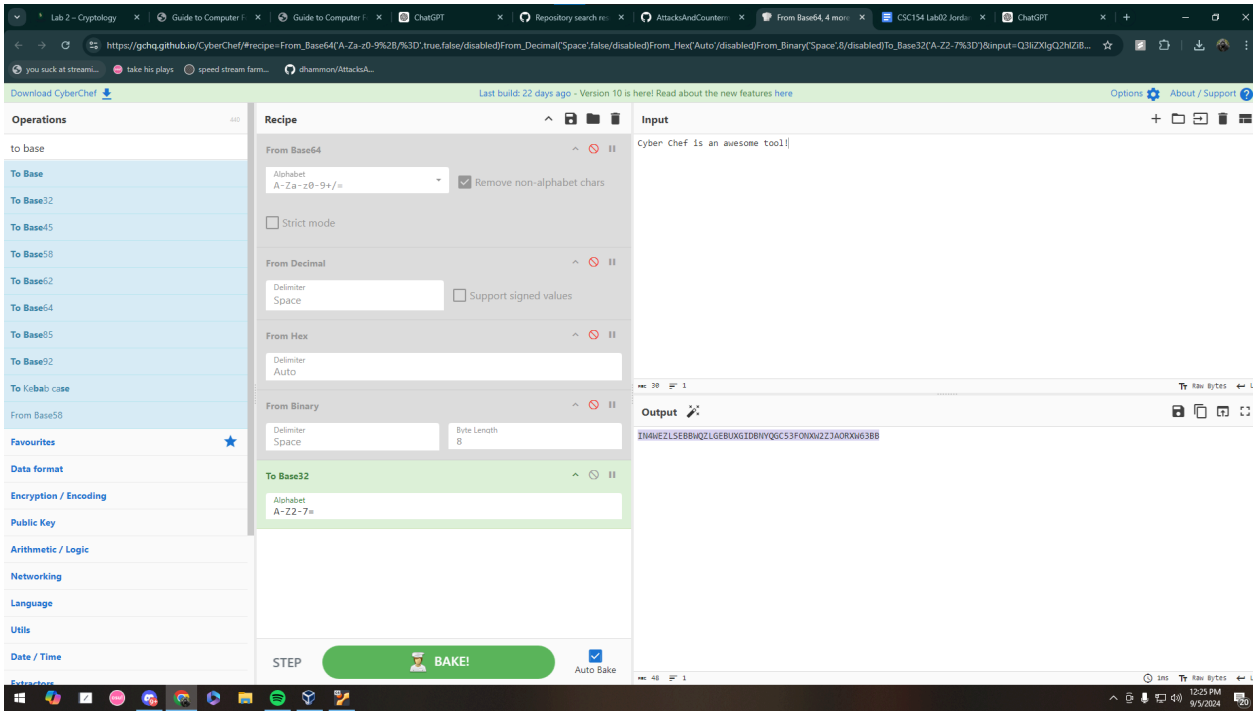
Hex is commonly used with assembly



one's and zero's

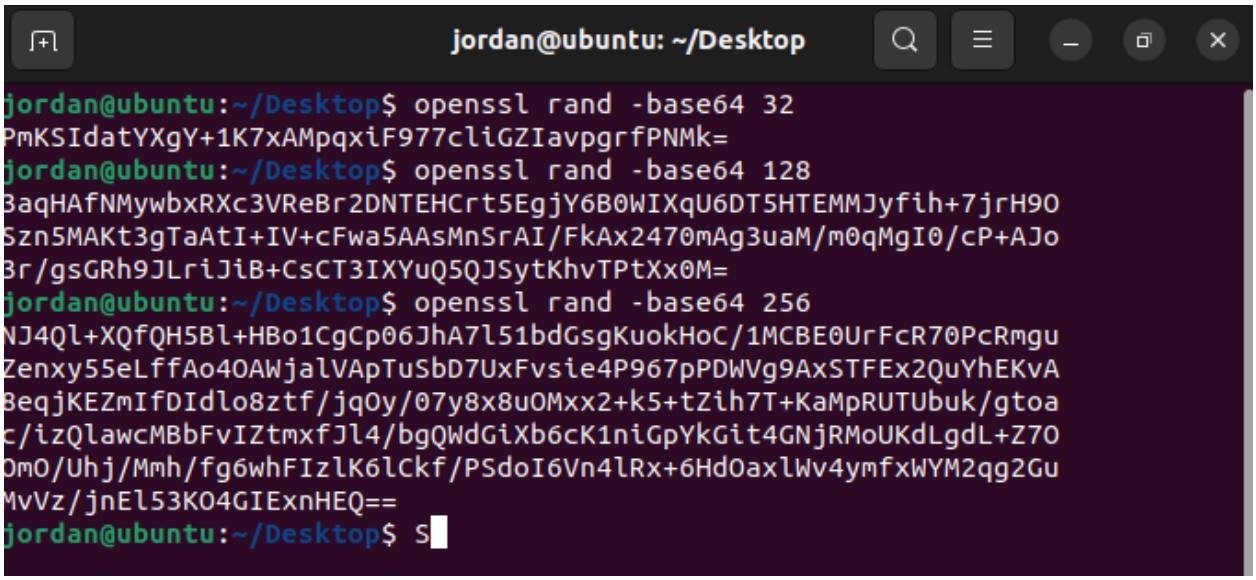
In step 3 we will encode the following string “Cyber Chef is an awesome tool!” into base32 using cyberchef.

IN4WEZLSEBBWQZLGEBUXGIDBNYQGC53FONXW2ZJAORXW63BB



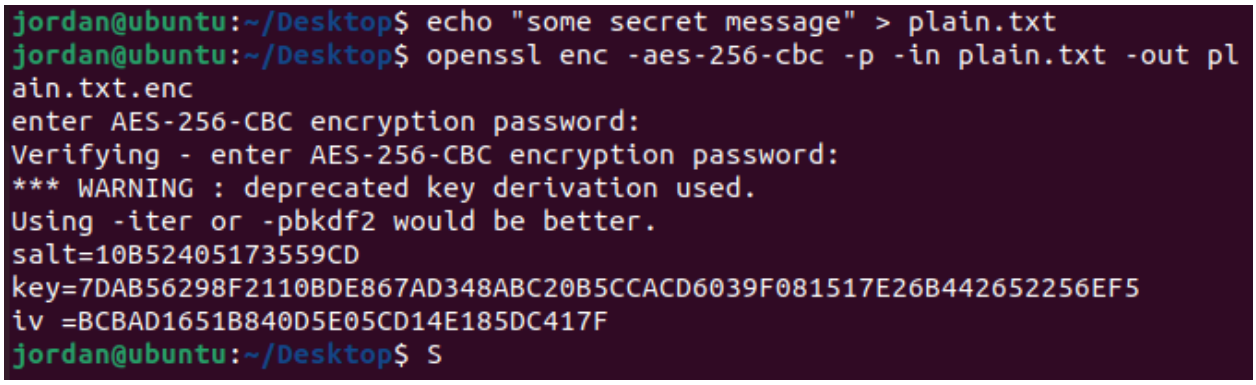
2.2 - Keyspace

We will now be using keyspace in openSSL on the ubuntu machine to generate random keys.



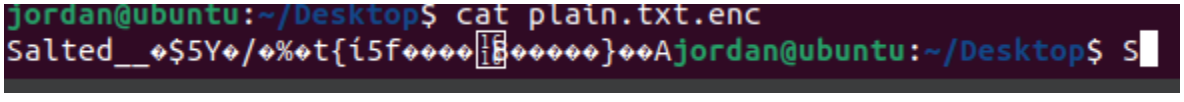
Here are the keys we generated

2.3 - Symmetric Encryption



Here is me encrypting a secret message in plain.txt using aes-256 code block cipher encryption algorithm.

I can decrypt it using my key that I made. But first, here is the contents of the encrypted file



```
Files @ubuntu:~/Desktop$ openssl enc -aes-256-cbc -d -A -in plain.txt.enc
enter AES-256-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
some secret message
jordan@ubuntu:~/Desktop$ S
```

Here is me decrypting my message using the same key I created to encrypt it.

2.4 Hash Generation:

The next task is to create hash digests using ubuntu's native md5sum and sha256sum tools

I will first need to create a file to be used with hashing utilities.

```
jordan@ubuntu:~/Desktop/Lab2$ echo "Tamperproof Message: crypto is the coo
lest!" > message.txt
jordan@ubuntu:~/Desktop/Lab2$ md5sum message.txt
586275d781bbf077bc9b2def6848de9b  message.txt
jordan@ubuntu:~/Desktop/Lab2$ sha256sum message.txt
8adc78f1bb49c8a86bbff34d9cdbeb433217820acbfa186e4556dd69e29872ce  message.
txt
jordan@ubuntu:~/Desktop/Lab2$
```

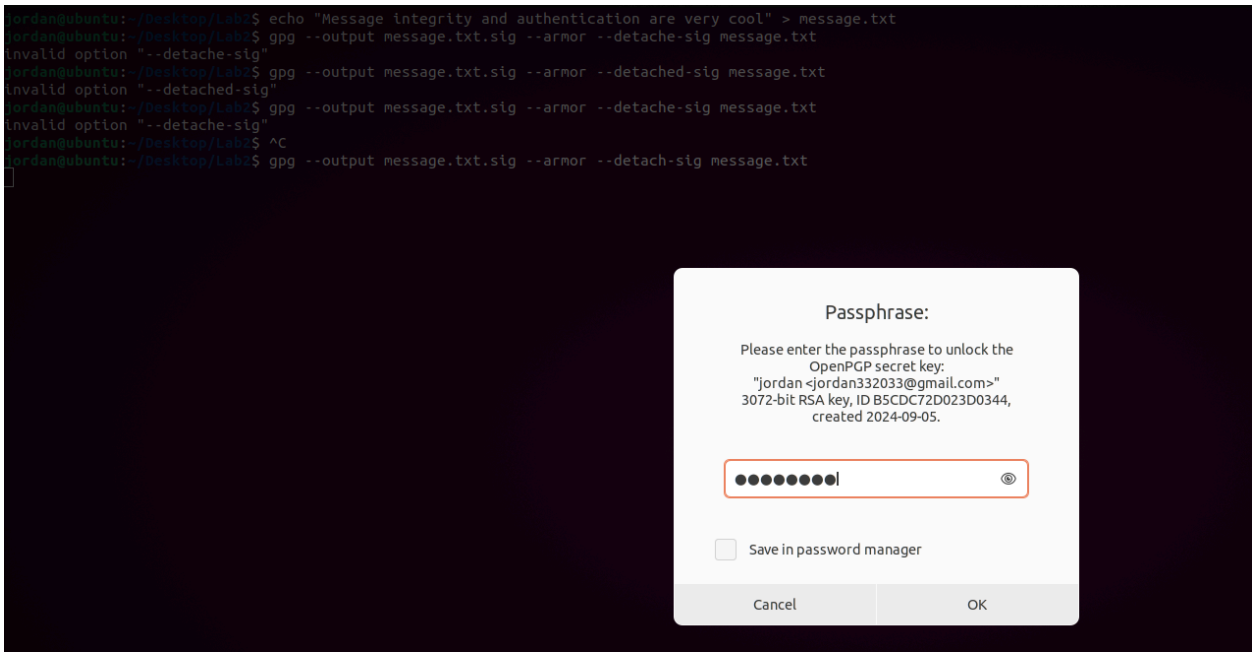
Here I have created a message.txt file and made md5 and sha256 hash digests for this file. Sha256 is much longer than md5.

2.5 Detached Digital Signature

We will now use our ubuntu vm to create a detached DS and verify it

```
jordan@ubuntu:~/Desktop/Lab2$ gpg --gen-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 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.

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: jordan
Email address: jordan332033@gmail.com
You selected this USER-ID:
    "jordan <jordan332033@gmail.com>"

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 perform
some other action (type on the keyboard, move the mouse, utilize the
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 perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key B5CDC72D023D0344 marked as ultimately trusted
gpg: directory '/home/jordan/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/jordan/.gnupg/openpgp-revocs.d/A8B0B5D1C7DB0560A4E3173CB5CDC72D023D0344.rev'
public and secret key created and signed.

pub   rsa3072 2024-09-05 [SC] [expires: 2026-09-05]
      A8B0B5D1C7DB0560A4E3173CB5CDC72D023D0344
uid                      jordan <jordan332033@gmail.com>
sub   rsa3072 2024-09-05 [E] [expires: 2026-09-05]
```

Here I have generated a key and will act as the message sender. The next step is to create a message to sign.



After creating the message, I digitally signed it using GPG.

```
jordan@ubuntu:~/Desktop/Lab2$ ls
message.txt  message.txt.sig  plain.txt  plain.txt.enc
```

As you can see, message.txt.sig was generated.

```
jordan@ubuntu:~/Desktop/Lab2$ cat message.txt.sig
-----BEGIN PGP SIGNATURE-----

iQGzBAABCgAdFiEEqLC10cfbBWCk4xc8tc3HLQI9A0QFAmbaCqsACgkQtc3HLQI9
A0RJiQv/X0seRE2gCHcqUsPBMPs/3Hsgbp2MtUMvYsc1QYh21mwTheIxScmZI6aB
aDKMyFiaKNEaDfAAV8xwdezQF0v6ljPdb/KgpYke3XrIXSGZmus5EaSFOVXgI23s
RMvDWuRDfVRo08NdLnkVM4/33p45y5tGOFzbrW8Uldvt53yzkxucays6M+1aNllq
FdlCmbpPTcqPbEExgS59n+/whPhzvQG+JfIPaACl77HB5S/im46uxGb6aJqOBvc/
Fh+jlwlxEvk+G/LPrPkcMZR8R+XfQJuLIRuU/+uD8YjnI4iIeceYfrzTeVD68VKv
tJbnn3jOuJoS/BwrvEzeVtfBjHtdXwLWgo37lIae1ozaAK8+BDLgpSS4B3NQ+IP+
9wXzGF+GzVzbuqLqyzvkjTQzfIuO2MOTsQMioKq1Nc1TlcKM9bdwjTA/XGBf5SeE
br2Paz3FJ9AJlw45XwgjaFtRaRLYgwNPoc5MMRbTlBc4sik97UB0ftEAm1AdIhBx
ISKMRN7E
=IVMW
-----END PGP SIGNATURE-----
```

Here is the digital signature that was generated.

```
jordan@ubuntu:~/Desktop/Lab2$ gpg --verify message.txt.sig message.txt
gpg: Signature made Thu 05 Sep 2024 12:46:51 PM PDT
gpg:                using RSA key A8B0B5D1C7DB0560A4E3173CB5CDC72D023D0344
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 2026-09-05
gpg: Good signature from "jordan <jordan332033@gmail.com>" [ultimate]
```

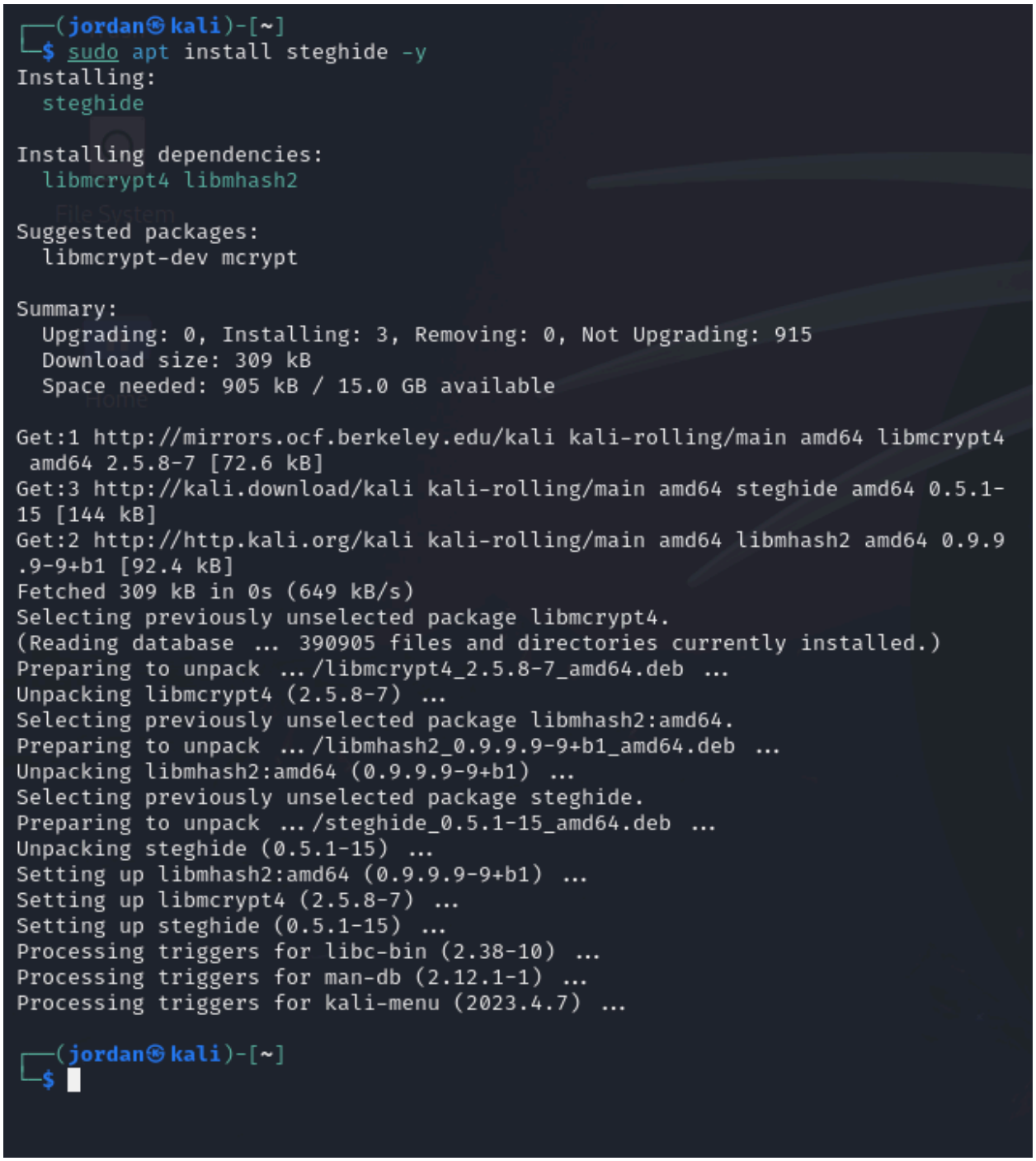
Here is the checking of the signature to make sure integrity is kept.

```
jordan@ubuntu:~/Desktop/Lab2$ nano message.txt
jordan@ubuntu:~/Desktop/Lab2$ gpg --verify message.txt.sig message.txt
gpg: Signature made Thu 05 Sep 2024 12:46:51 PM PDT
gpg:                using RSA key A8B0B5D1C7DB0560A4E3173CB5CDC72D023D0344
gpg: BAD signature from "jordan <jordan332033@gmail.com>" [ultimate]
jordan@ubuntu:~/Desktop/Lab2$
```

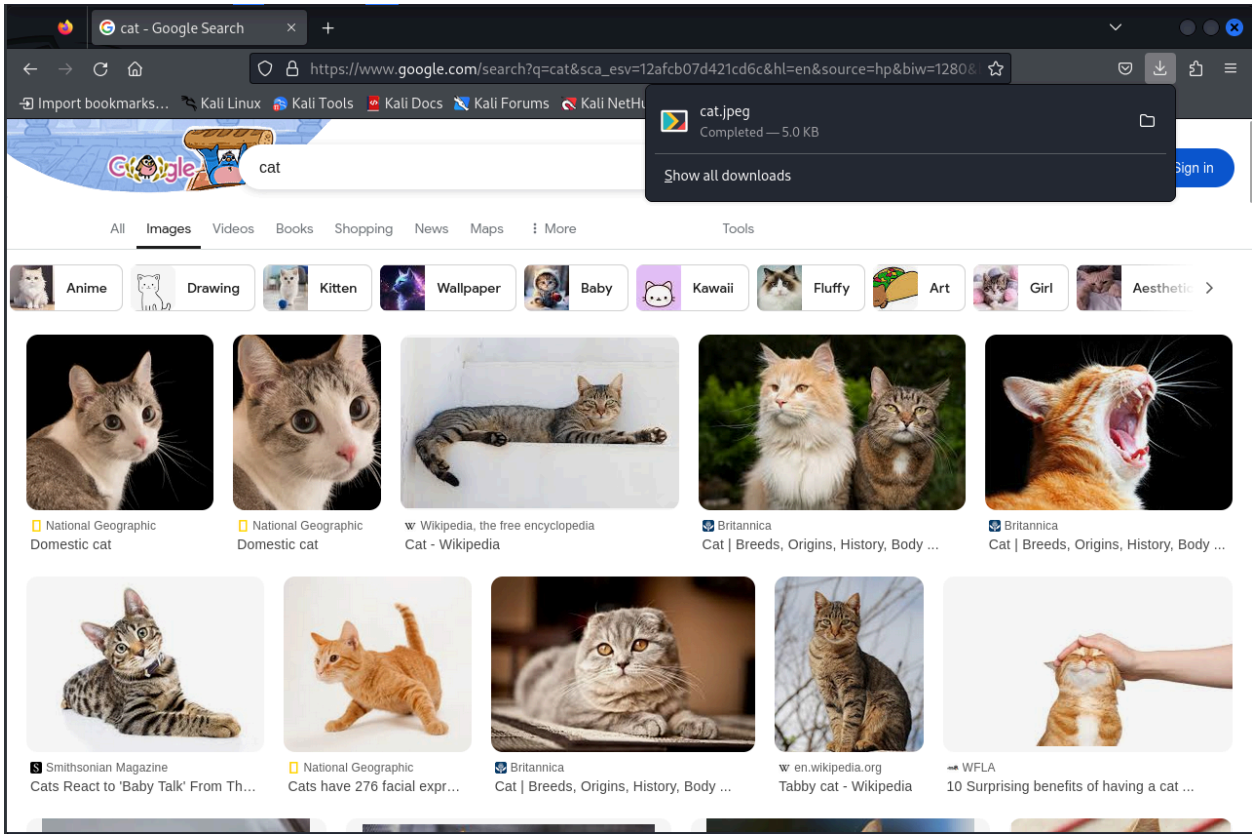
After slightly changing the message.txt file we now have a bad signature showing that data integrity is not upheld. In a real world situation this would mean that the file was found by some third party and changed during transport.

2.6 - Steghide

Steghide is a steganography tool which hides data in plain site like image files. We will create a message and conceal it in an image file, then extract the secret from the image.



After updating and installing steghide, I proceeded to create a message and file



I downloaded a jpeg of a cat and created a message

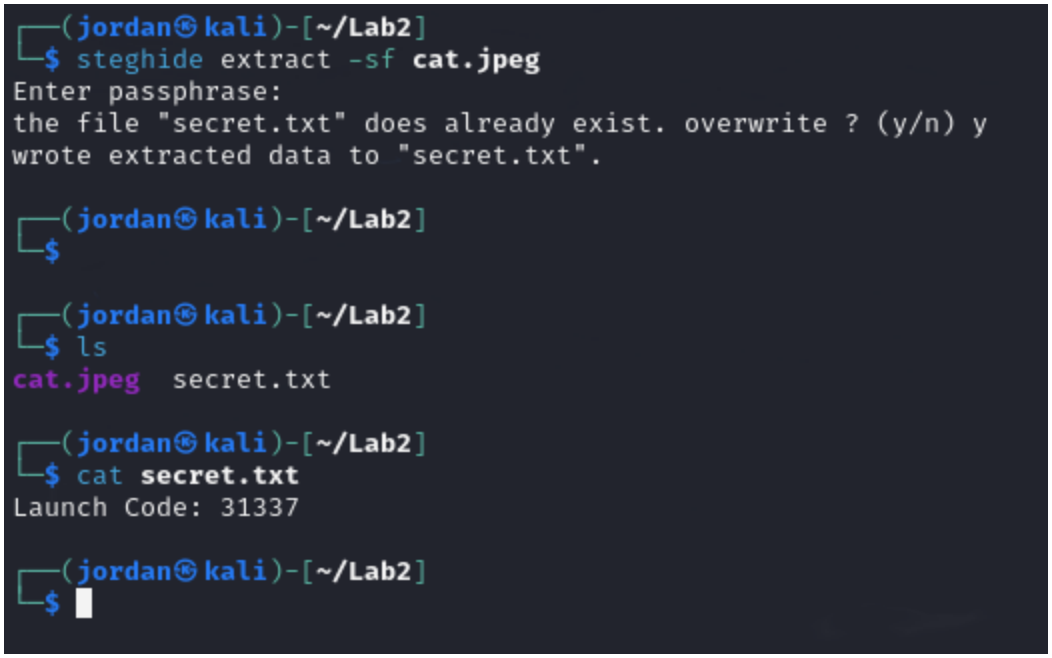
```
┌──(jordan㉿kali)-[~/Lab2]
└─$ echo "Launch Code: 31337" > secret.txt
```

We will now hide our data in this image using steghide. -ef flag is used to embed file and -cf flag is used to insert the secret message into the image.

```
┌──(jordan㉿kali)-[~/Lab2]
└─$ steghide embed -ef secret.txt -cf cat.jpeg
Enter passphrase:
Re-Enter passphrase:
embedding "secret.txt" in "cat.jpeg"... done
```

Our message is now embedded

Pretending we sent this to someone else, we will now extract the secret data



We used the -sf clag to extract the message

2.7 Known Plaintext Attack

In this exercise we were given plain and cipher text and asked to decrypt the cipher text and explain the algorithm
Plaintext: Break my simple encryption
Ciphertext: rOe nx zlzfvrcya rlpevcgba

Just by looking at these side by side I can see one major flaw. Capital letters are still capitalized after encryption. This gives away that B = O, from there I can see that the text is scrambled in some way. My next intuition tells me that I should run this in a shift cipher brute force to see what looks similar to the plain text.

Running this into a brute forcer I found

```
→13 (←13) eBr ak mymsiepln eycripton
```

This looks just like the plain text but scrambled.
eBr,_ak,my,msi,epl,n_e,ycr,ipt,on

I have uncovered the algorithm used. It is a 13 shift cipher using a uniform scrambling technique. For every 3 chars abc, they are transformed to cab. I showed

this in my above example where I replaced spaces with an underscore and segmented each 3 chars with a comma.