

Analysis

Project Idea:

The idea for my project, is to have a program that acts as a vault for important files. It will encrypt files given, and store them in a specified location. Once they are encrypted, they will only be accessible from within the program, and will only be accessible within the program if you know the encryption key (passphrase) that you set when creating the “vault”.

Within the vault, you should be able to easily organise your files, add more to the vault, and remove (decrypt) files from the vault to any location (if possible).

My program would be useful for teachers, as they have to keep documents on student's grades, and any other student details secure. Since this is my use case, I will have to thoroughly test the security and practicality of my program to make sure teachers want to use it, and trust the program with these files. Also, I will add an optional mobile app that the user can download, which lets them connect to the program via Bluetooth to unlock the vault. This is would be useful if you are a teacher, as if you leave the room with your phone in your pocket, and it is connected to the vault, if you have forgotten to lock the vault then a student might try to browse through it while you are gone, but with the app, as soon as you disconnect the Bluetooth connection it locks the vault, so if you forgot to close it then it closes itself.

The Bluetooth app should also be able to receive files from the PC app, so that the user can download files that are in the vault onto their mobile device. This would be useful for teachers that do not take their PC home (e.g not a laptop), so they can upload the files from the computer, to their phone so that they can edit the file at home or on the move (with another mobile app).

The program needs to work on both Windows, Linux and MacOS, as then teachers/users have more flexibility with what operating systems they can use it on, so they can easily go from machine to machine and carry their vault with them (on a USB stick for example), and they know that they can reliably use the program on most machines.

The user experience has to be pretty good. Good design practice will have to be used when making the GUI (e.g not putting the delete button next to the decrypt button), as I want my program to be easy to use by a wide range of people, so that even people who are not so good with computers can easily use the program. The way the user is directed around the program has to be logical as to not confuse the user, and adding a panic button to take you back to the main screen may be a good idea.

Client:

An example client for my project could be a teacher/school, as they have to keep files about students secure. For example, pupil details, exam results and other important student details. My program aims to help the teacher/school keep the pupil's files safe, and prevent the files from being accessed if their device is stolen. It will encrypt files given to the program, and be secured by a pin code that is transferred over Bluetooth to the computer from a mobile device. Once the mobile device is unpaired from the computer, the app will lock again. This will prevent someone from having access to the files if the computer is unlocked and is stolen, as the mobile device will go out of range of the computer, so the computer will lock.

I sent a questionnaire to a member of the IT office at my school to ask what regulations there were about keeping a teacher's files safe, and what encryption they would suggest for keeping the files secure.

Hi Josh,

What encryption should I use when encrypting the user's files? [The bare minimum would be 128 bit AES, though 256 bit is recommended.](#)

Are there any standards or laws about what encryption method I should be using for files such as a teacher's student files (one of the clients for this program)? [Data protection laws. The current UK Law is the Data Protection Act 1998. Though as of 25th May, the law will be General Data Protection Regulations \(EU Law regarding all EU Citizens\). This is a very complicated law, that is causing headaches for businesses worldwide. I've attached some links you might find useful regarding GDPR towards the end of this email.](#)

Hope this helps!

Many thanks Mr ____

<https://www.eugdpr.org/>

<https://itpeernetwork.intel.com/gdpr-opportunity-rethink-security/>

<https://ico.org.uk/for-organisations/guide-to-the-general-data-protection-regulation-gdpr/>

https://media.datalocker.com/marketing/GDPR_infographic_2017.pdf

<https://www.kingston.com/en/usb/resources/eu-gdpr>

I will be using this information as guidance for what I have to take into consideration. I will keep in mind the data protection laws when I am storing the user's files, and make sure I am within the regulations.

The EU General Data Protection Regulations consist of (As of 25/05/18):

Breach Notification:

If a data breach has been found and it might "result in a risk for the rights and freedoms of individuals", then the person that the data belongs to has to be notified within 72 hours.

Right to Access:

The person who's data it is can at any point ask for confirmation as to whether or not data concerning them is being processed, where it is being processed if it is and for what purpose.

Right to be Forgotten:

The data subject can ask for their data to be erased, and stop the processing of their data. This will be done depending on whether there is public interest in their data (e.g if a politician says something stupid then they can't ask Google to delete it just because it makes them look bad), and if the data is no longer relevant (e.g your cookies from last week that were used for targeted ads).

Data Portability:

The data subject should be allowed to ask to receive the data, and they should also be able to change which company is controlling their data.

Privacy by Design:

Tells the controllers of the data to only use the data absolutely necessary for the purposes they need it for. For example, an advertisement company might use your cookies to target ads to you, however they can't then use your location unless they are also using that to target ads. Basically don't take more than you need.

For my project, as the user is the data controller, then they already have the right to access, the right to be forgotten and data portability. For the breach notification, they will probably know it has happened as someone needs to have physical access to where the data is stored to breach it. However, with privacy by design, I will not be using any of the user's data for advertising, or any other agenda. I will make this clear to the user when they first use the program. Also the security will be

Another issue could be that if a file is deleted, the contents of the file might still remain. To fully remove the file I may have to use a one way function that ruins the data before deletion so that it cannot be accessed after it is deleted.

Objectives:

1. GUI should:
 - a. Be easy to use:
 - i. Logically laid out.
 - ii. Have simple options.
 - b. Display the files currently stored in the vault, along with the file extension and the size of the file.
 - c. Display the storage space remaining on the storage device the program is running on.
 - d. The user should be able to easily encrypt and decrypt files:
 - i. Using easy to access buttons in the UI.
 - ii. Using drag and drop.
 - e. Have an options menu, including the options to:
 - i. Change security level (from 128 bit AES to 256 bit AES).
 - ii. Change the location of the vault.
 - iii. Set the default login method (Bluetooth or no Bluetooth).
 - iiii. Change if the search in the file browser is recursive or not.
 - f. Make it easy to manage the files in the vault (move to other folders in the vault, rename, etc).
 - g. Have a secure login screen.
 - i. Ask the user to either input the key via their keyboard (no Bluetooth for that session), or connect via the app.
 - ii. Tell the user if the key is invalid or not, and smoothly transition into the main program.
 - h. Look relatively good without being bloated.
 - i. Allow the user to easily read file names, and easily tell folders and files apart.

j. Let the user preview images without opening them (using thumbnails or an information screen).

2. App should:

- a. Be easy to use.
- b. Connect via Bluetooth to the PC.
- c. Allow the user to input their pin code easily.
- d. Tell the user if the pin code is invalid or not.
- e. Make it easy to recover from mistakes (e.g invalid pin code, or if they make a typo).
- f. Allow the user to see a list of files currently in the vault, and let the user download those files onto their mobile device.

3. File handling:

- a. Store the encrypted contents in the location specified by the user.
 - b. Encrypt and decrypt relatively quickly, while still being secure.
 - c. When the Bluetooth device goes out of range (if using Bluetooth), encrypt all decrypted files and lock the program until the pin code is input correctly again.
 - e. Have a recycling bin so that the user can recover their files.
 - f. When a file is opened, check for changes once it is closed.
 - g. Files stored in the vault should not be accessible from outside of the app.
 - h. Names of the files stored in the vault should also not be view-able from outside of the app (encrypt the name).
-

Design

Bluetooth:

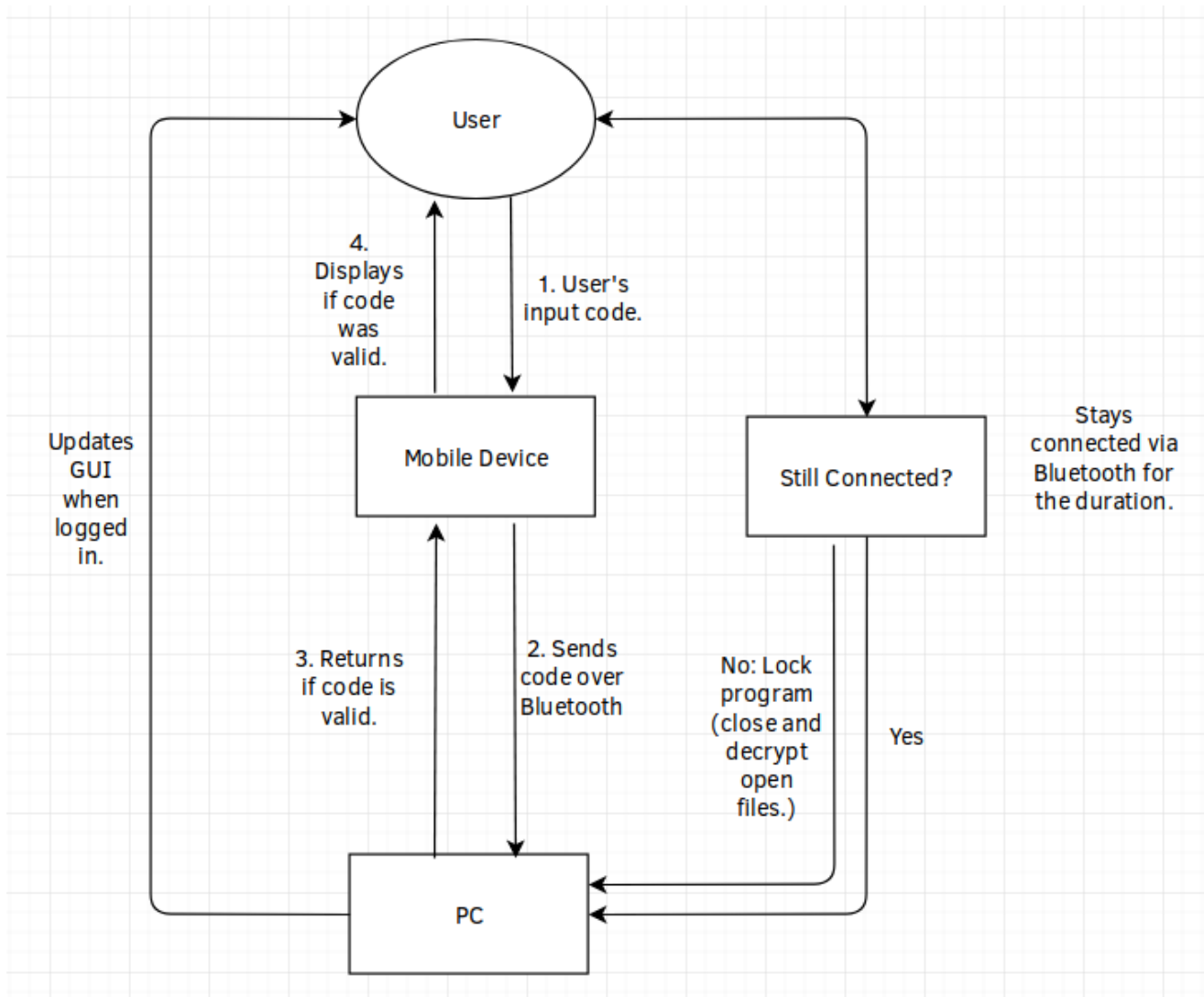
For the file store to be unlocked, I need to send the passcode to the computer via a Bluetooth connection.

For the computer and android device to connect to each other, one device has to be assigned as the server, so it makes sense to me to use the computer as the server, as it will be running for the entire duration that the user wants to use the program.

For the mobile app, I will be using Kivy to program the app. I am using Kivy so that the design is consistent with the design of the PC app. I will be using the android.bluetooth library that is included in the android SDK to transmit the data via Bluetooth.

For the Bluetooth server (on the pc), I will be using Python to receive the pin from the mobile device using PyBluez, check the sent pin, and send a message back saying if the code was valid or not. If the code is not valid, a message will be displayed on the computer that the code is invalid, and the code on the screen of the phone will be erased.

Here is a flow diagram for what Bluetooth will be like:



To send the files, I will need a protocol. A protocol is a set of rules for communicating over a network. A protocol will allow the program to distinguish data that is being sent is a key, file list or a file itself.

Protocol

The protocol rules all have to be strings of bytes that are not likely to appear in a key, file list or a file. This is a necessity because otherwise mid way through sending a key, file list or file, if the program encounters a protocol rule within the key, list or file, then it may cause the program to get confused as to what is being sent, or if the current key, list or file has finished being sent.

For each of the possible items that are going to be sent, each item needs a start header, and an end header. Start header:

```
1 | !<operation>!
```

End header:

```
1 | ~!END!
```

For operations that do not have any extra data (arguments), then only the start header is sent.

For sending more complex operations, I will use objects that hold the data, pickle them (object sterilisation), and send the object data sandwiched between the `!<operation>!` header (start header) and the `~!END!` header. For more complex operations that have multiple arguments, a separator is used to separate those arguments:

```
1 | ~!~
```

Here is an example with multiple arguments:

```
1 | !<operation>!<argument1>~~!~~<argument2>~!END!
```

This is especially useful for files, as this way I can send the metadata in one big lump, then send the file bit by bit. Here is what a file would look like when it is sent:

```
1 | !FILE!<metadata_object>~~!~~<data>~!END!
```

For the key however, since it will always be small (< 16 bytes), I will just send it with a `#` at the start, and a `~` to finish the message. This is acceptable because when the PC program starts, it doesn't expect any requests from the client, so it is just waiting for the key. The key should also only be made up of numbers.

```
1 | #<key>~
```

For items such as file metadata, I will use Python pickle to send an object (more of a structure) containing the metadata, rather than using separators, as then it is much easier for me to add information I want to send.

Sending files over Bluetooth:

To send a file from the vault, first it has to be decrypted to a temporary location. I could instead send the data from within AES, so that when a block is decrypted it is sent, however I don't plan on writing AES in Python since speed is essential for AES (and a new Bluetooth socket would have to be set up if using a different language).

Metadata will be sent as an object before sending the file contents, as talked about in the above section.

An example class for file metadata may look like this:

File Metadata
+ name: string
+ size: int
+ isFolder: boolean

```
1 | class fileMetadata:
2 |     def __init__(self, name, size, isFolder):
3 |         self.name = name          # The name of the file being sent.
4 |         self.size = size          # The size of the file being sent.
5 |         self.isFolder = isFolder # Boolean for if the file is actually a folder.
```

This is more of a structure than an object, as it has no methods, and is just a collection of data.

After the metadata is sent, a separator will have to be sent to separate the metadata from the file data itself. I discuss this in the above section.

For the file itself, I will send the file in chunks, so that

1. I don't use too much memory (since mobile devices usually have a small amount of memory compared to regular computers).
2. The Bluetooth adapter can keep up with the amount being sent.

This reduces the stress on both the mobile device and the PC.

Once the full file is sent, an end header is sent to tell the program that the full file has been transmitted.

File Storage:

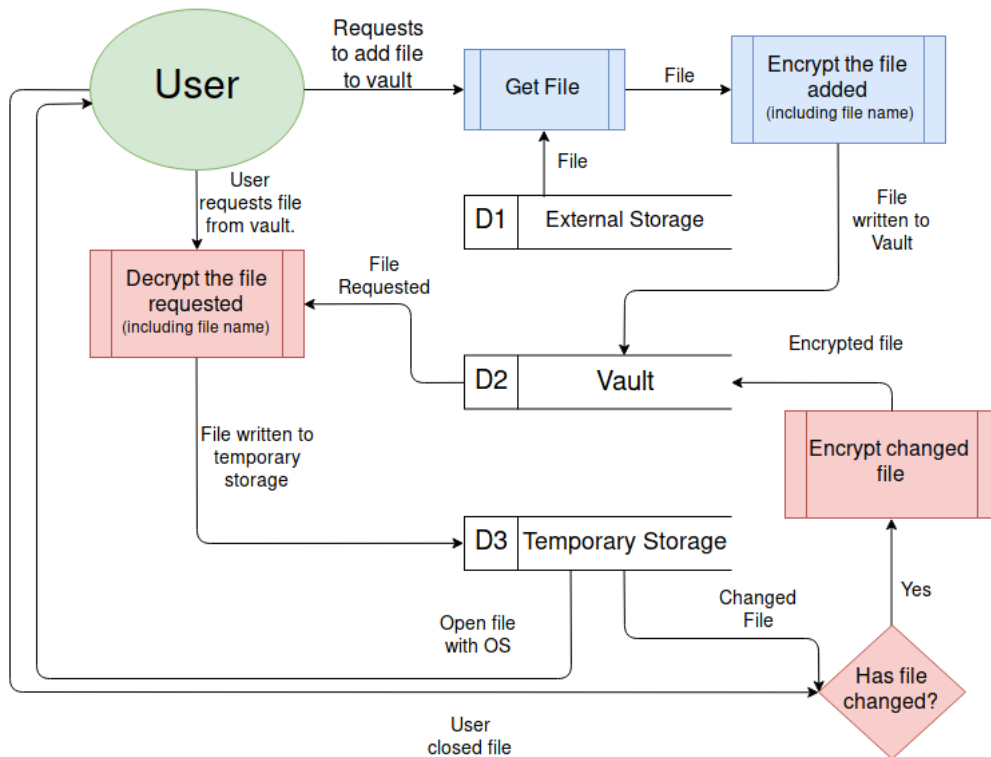
For storing the files, I will store the encrypted files in a directory set by the user. The directory will be managed using a tree structure, where the root folder contains folders for each file, with the name of every folder and file being encrypted, as otherwise anyone can see the name of your file.

The encryption method I will use AES 128 bit, as it will slightly compromise security over using 256 bit, however it will be faster to decrypt files for use, giving the user a better experience, however I might add an option to use 256 in the settings if the user needs more security over performance. For the encryption key, the key will be set up every time a new vault is created (this includes first starting the program). It will tell the user to enter the new key, and then from that moment forwards in that vault, that key will remain the same, and will be used every time a file is encrypted/decrypted in the vault.

When a file is encrypted, the key is appended to the start of the data, and is then encrypted. This is so that when the data is decrypted, only the first block has to be decrypted and compared with the key entered to check if the key entered was correct, rather than decrypting the whole file just to find out that the key was incorrect. This will also be used to check the key entered at login, where the login will try to find the first file it can within the vault, decrypt the first block of that file and compare it with the input.

The key will have to be hashed if I send it over Bluetooth, as it may get intercepted, and it is also a good idea to hash it on the computer program as well, as if someone somehow manages to get the key, it will not be the user's original input, so if the user uses it for something else, their other accounts will be fine.

Here is a data flow diagram showing how the data is handled once logged into the program:



The key is also passed to any stages that encrypt or decrypt, as at this point the user should already be logged in.

When a file is edited, the file should be checked to see if any changes have been made, and if there has been changes, remove the version of the file currently in the vault, and encrypt the latest version into the vault. Also, if there are any new files in the temporary folder (for example if the user renames the file), then encrypt them to the vault as well.

To do this, I need a way of getting a checksum of the file before and after it has been opened. I need a fast algorithm so that the user is not waiting too long for the file to open and close, but it also needs to be unlikely that there will be a collision (where if they change the file and the checksum gives an answer that is the same as before the file was changed, that would be a collision). I will discuss which checksum I will be using in the next section.

When viewing the files in my program, I will use an object that holds all of the information I need about the file, and any methods that I need to get that information.

Here is what I expect the class to be like:

File
+ rawSize: int
+ displaySize: string
+ isDir: bool
+ path: string
+ name: string
+ hexPath: string
+ hexName: string
+ getChecksum(self): string
+ getSize(self): int

Where `getChecksum` will get the BLAKE2b checksum of the file. The `hexPath` and the `hexName` will hold the encrypted path and encrypted name of the file, so that I don't keep encrypting and decrypting the file name.

Choosing the right algorithms:

When encrypting, decrypting and hashing data in my program, I want it to be as fast as possible without compromising too much on security.

Hashing:

When hashing the key when it is input, the algorithm has to be very secure, and speed does not matter as much. A member of the SHA2 family of algorithms would be a good algorithm to do this, as it is quite slow, but it is very secure (SHA1 was found to have a lot of hash collisions). Speed does not matter as much for the key, as the input data will only ever be less than 16 bytes. A faster algorithm will only provide a few milliseconds over SHA, so there is no point compromising on security for a negligible time decrease.

For getting the checksum of files, the algorithm has to be very fast, as it will be done on the data in the file before and after the file is opened to check for changes. If this algorithm is slow, then the overall user experience will be much worse if the algorithm takes ages to open and close files. I will test each algorithm I am thinking of using for hashing and compare them using this algorithm (Python):

```

1  import hashlib          # Library of hashing algorithms.
2  from random import randint # Used to generate the data.
3  from time import time    # Used to measure how long the operation takes.
4
5  def generate(times, size): # Generates data, each block of length "size", and "times" number of blocks.
6      data = []
7      for i in range(times):
8          for j in range(size):
9              data.append(randint(0, 255)) # Randomly generate a byte.
10     return bytearray(data)
11
12 def test(times, size):
13     data = generate(times, size) # Generate the data

```

```

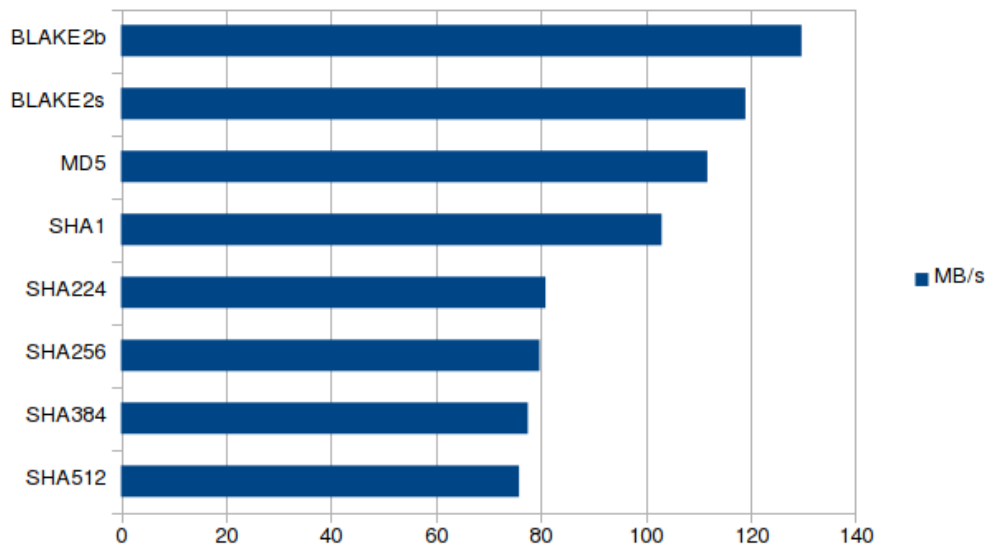
14     start = time()                # Get the start time
15     for i in range(times):
16         hashlib.sha256(data[i*size:(i+1)*size]).hexdigest() # Do the hash (in this case SHA256)
17
18     return (times*size)/(time()-start)    # Return the bytes per second.
19
20 print(test(1000, 128)) # Run the program.

```

I will run this algorithm on the same computer and make sure background tasks are closed, so that the results are not affected by other programs.

Here are the results:

Megabytes per second for each hash function (using 1000 blocks of 128 bytes (128 kilobytes)):



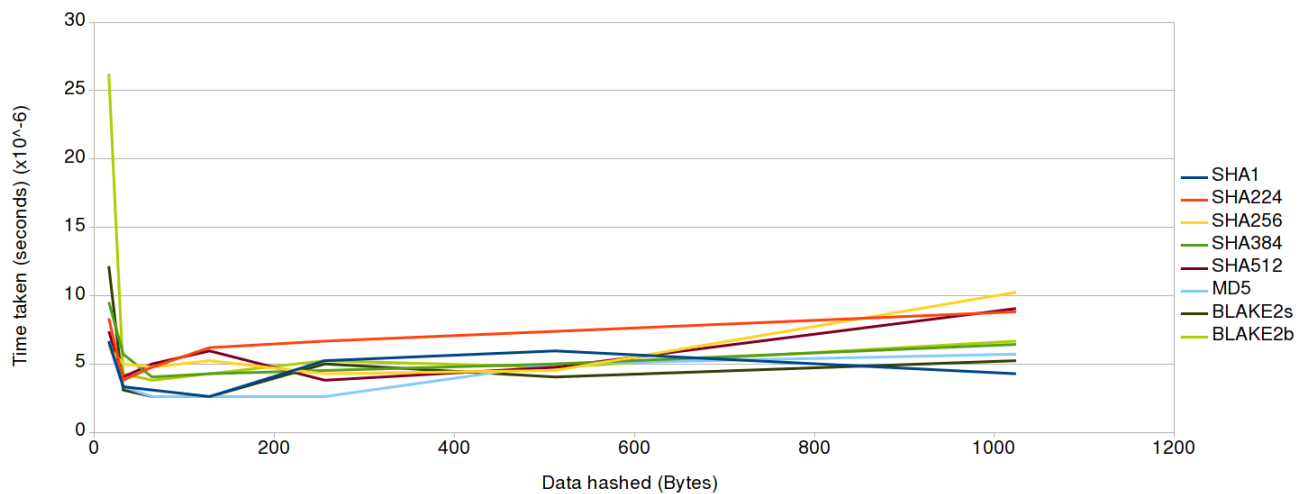
For my next tests, I will do data hashed against time. For this I will be using different sized files that I will make using this function:

```

1 def generateFile(name, totalSize):
2     fo = open(name, "wb")
3     a = bytearray()
4     for i in range(totalSize):
5         a.append(randint(0, 255))
6     fo.write(a)
7     fo.close()

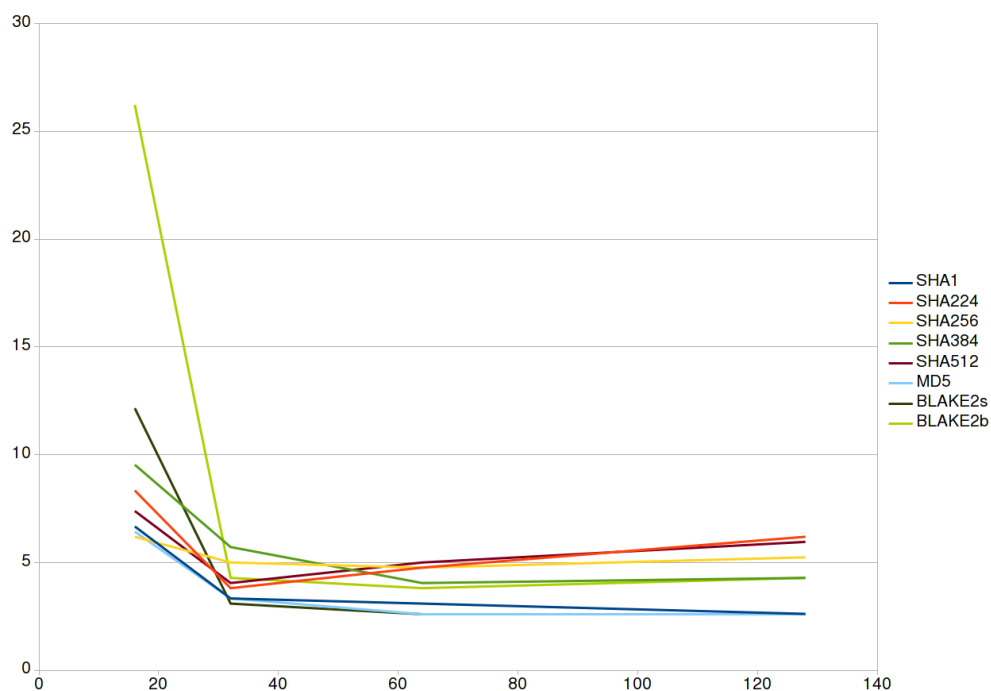
```

First I will test each hash function with encrypting very small data (≤ 1 KiB). These were the results:



This image can be found larger in the **Large Images** section as **Figure 3**.

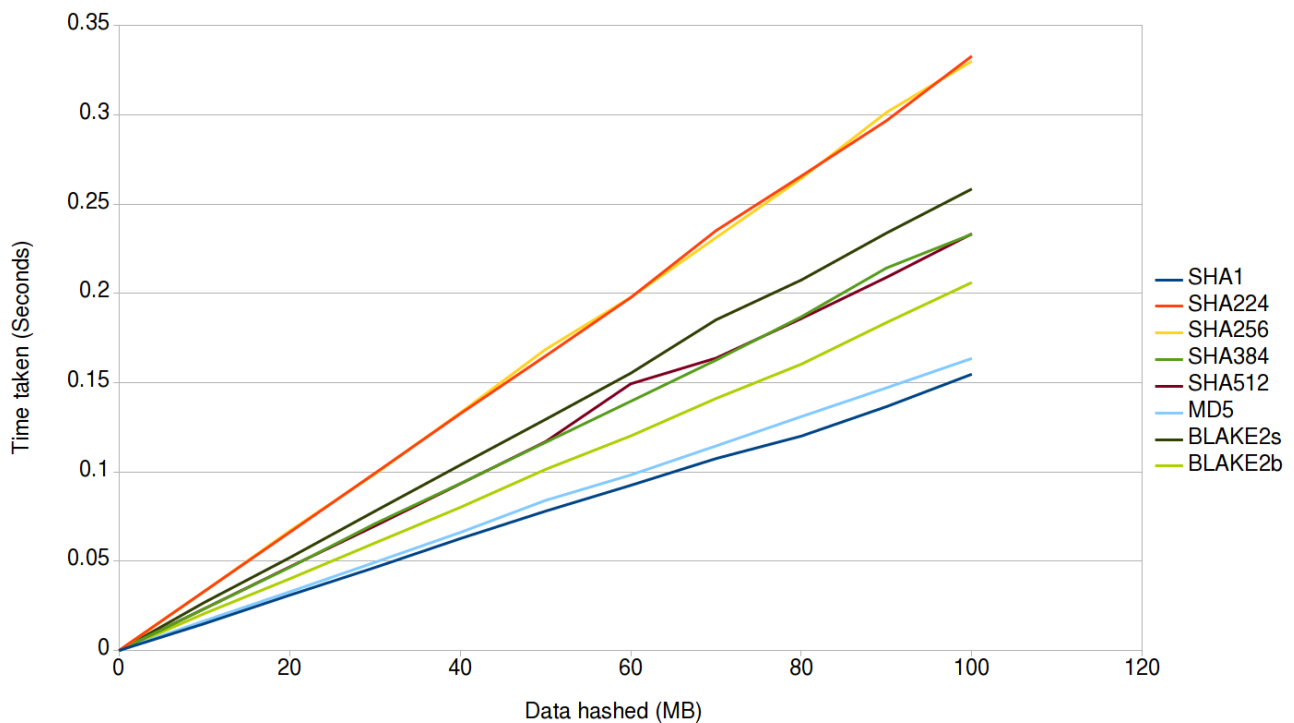
Here is the start of the graph, as that is the most interesting bit:



The axis on this graph are the same as the one before it.

Here we can see that SHA256 is the fastest at hashing 16 bytes, but is quickly surpassed by most of the algorithms. Both BLAKE algorithms had a bad performance at the start, but after 64 bytes both were doing alright. MD5 is the quickest overall out of the group. From these results I think I will use SHA256 for hashing the key, since the key is 16 bytes in length, and also because SHA is more aimed at security than BLAKE, and MD5 and SHA1 are obsolete in terms of security.

The BLAKE algorithms were designed for big data, which is what I am going to look at next:



In this graph, the gradient (rate of increase) of each line is the ratio of seconds to megabytes of each function (so $\frac{x}{y} = \text{megabytes/second}$). So the less steep the line is, the faster the operation.

SHA256 and SHA224 have taken the longest, at almost identical rates. BLAKE2s is quite slow, and this is because BLAKE2s is designed for 32-bit CPU architectures, and my CPU is 64-bit. MD5 and SHA1 are both the fastest, and have similar performance, but have security problems. BLAKE2b was the fastest out of the secure functions, so I will be using BLAKE2b for checksums in the program, as checksums need to be calculated quickly, as discussed before.

Encryption:

For encryption, I will definitely be using AES, because it is the standard and has been tested extremely thoroughly by the public. I do not want to compromise on security, and AES is still pretty fast anyway.

I will use 128 bit AES mainly, as it is still proven to be secure from attacks, and may include the option to use 256 bit if desired by the user. The majority of users will not need AES 256 level security, but I will include it for people that may need it.

AES:

History:

In 1997, the encryption standard at the time, DES, was becoming obsolete due to the advancements in the computer industry. This resulted in the National Institute of Standards and Technology in the United States to call for a new advanced encryption standard (AES).

They held a competition that consisted of 15 different algorithms that had been submitted by different teams. The algorithm that won was an algorithm called Rijndael, an algorithm created by two Belgian cryptographers – Vincent Rijmen and Joan Daemen.

One of the reasons AES has been more successful than DES so far is that AES was thoroughly tested by members of the public during the competition, analysing every aspect of the algorithms to find a way to break them. On the other hand, DES was created in secrecy by IBM in the 70s, and the algorithm was only released a few years later.

This open-source approach ended up helping the new Advanced Encryption Standard, as the program could be heavily analysed by people all across the globe.

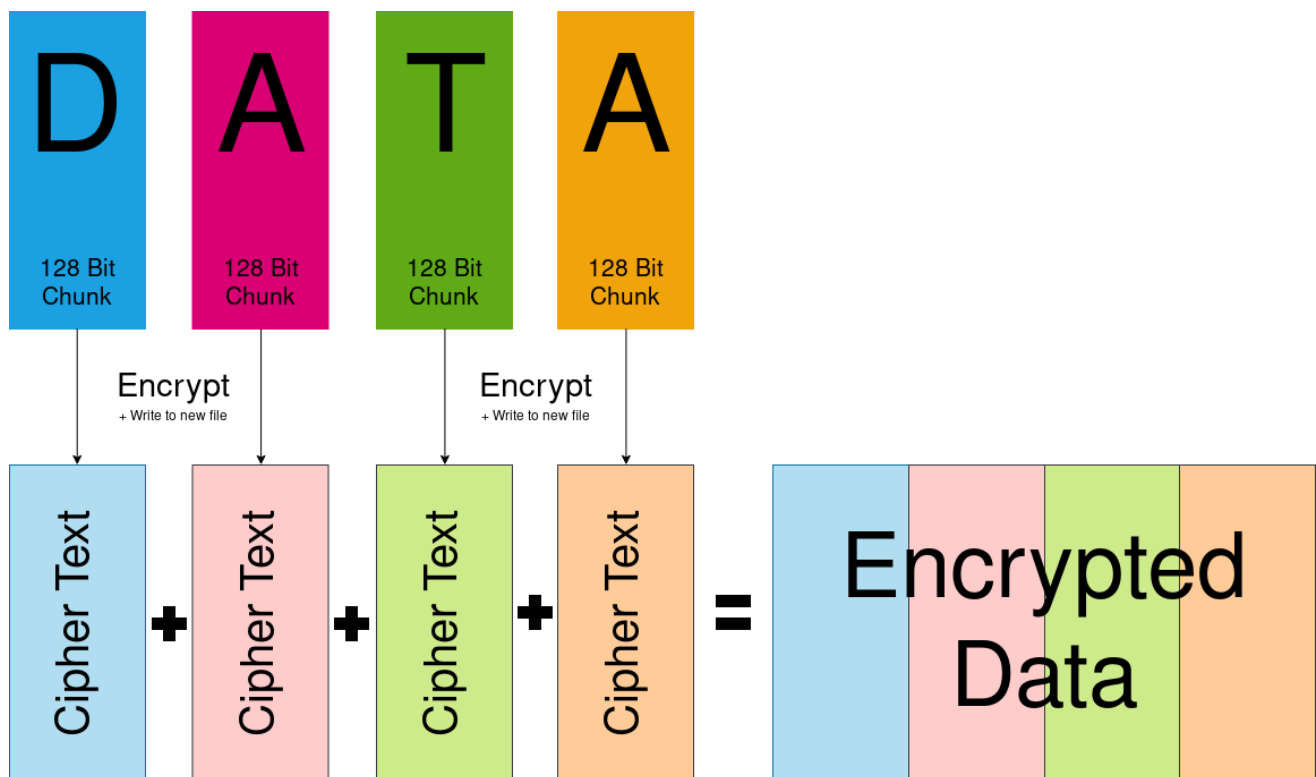
The Algorithm (128 bit AES):

How the data is handled:

AES works by using a block cipher, so it splits the data given into 128 bit, 192 bit or 256 bit chunks depending on what AES you choose (128, 192 or 256). You then use the algorithm on each block to get the cipher text, then you write it to the new file, and move onto the next block.

AES is a symmetric cipher, so only one key is needed to both encrypt and decrypt the data.

Here is an example for 128 bit AES encryption:



Decryption works exactly the same, however the cipher text is split up and decrypted.

Each 128 bit "block" of data can also be called a "state".

Before the operation starts:

First, the data has to be a multiple of 16 in length. If it isn't then more bytes need to be added to the end such that the data is 16 bytes in length (padding).

However, the padding cannot just be 0's at the end, as when we decrypt the block, we have no way of distinguishing these 0's from the rest of the data, or know if they are supposed to be there. To get around this, when we add the padding, we give each byte the value of how many more bytes we need to add to get the length of the block to 16 bytes. This sounds confusing, but here is an example:

Say we had a block that was = **[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]**

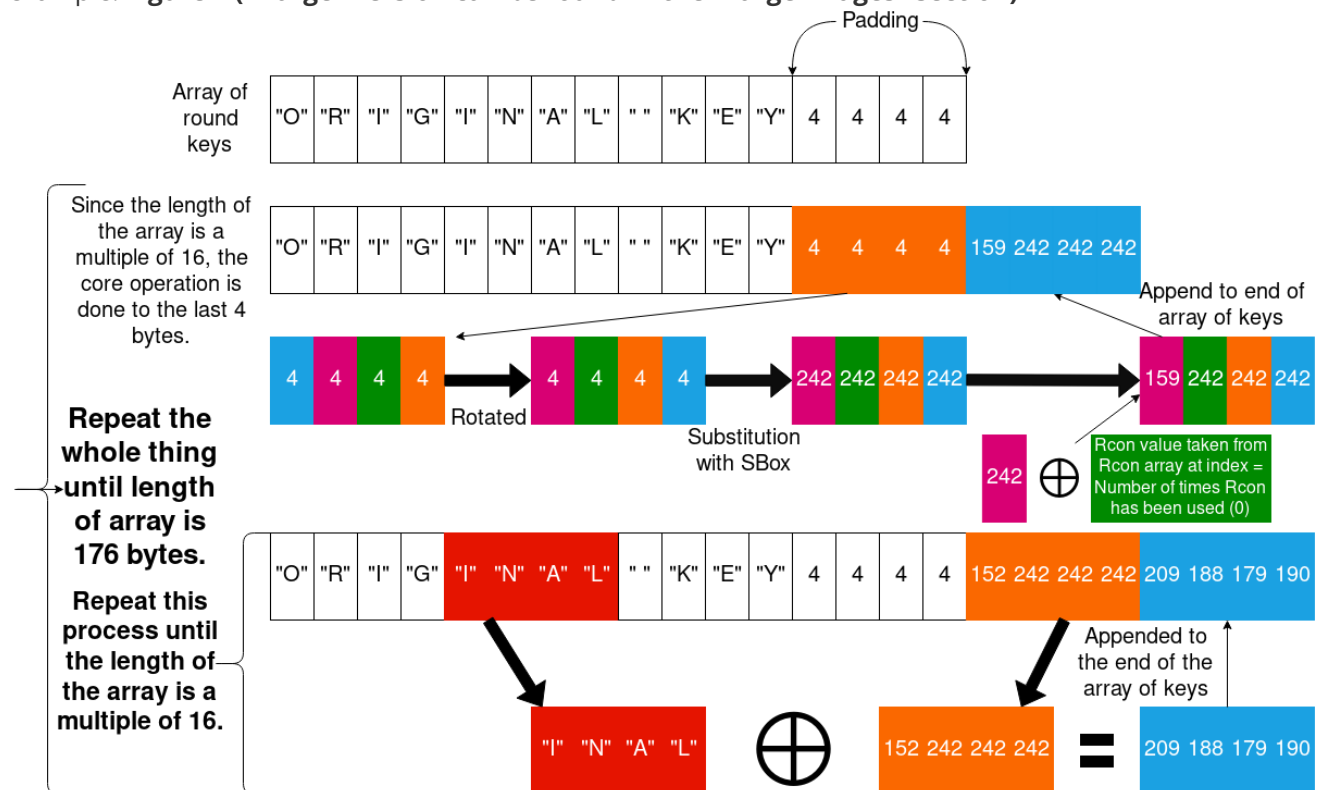
This block is not 16 bytes in length. To pad this block, we need to add 3 lots of the number 3 to the end (since $16 - \text{length of the block} = 3$). The new block would look like this:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 3, 3, 3]

When we go to decrypt this block, we check to see if the value of the last byte in the block is lower than 16, and that if the number occurs the same number of times as the value, then we remove these bytes.

For each round of the encryption, a different key has to be used. To make the cipher decipherable, these keys have to be derived from the original key given. For 128 bit AES (the main one I will be using in the program), the 16 byte key has to be transformed into a 176 byte list of 16 byte keys (11 keys in total, one for every round).

The first 16 bytes are the key, and then from there, the algorithm is started. Here is the algorithm with example: **Figure 1 (A larger version can be found in the "Large Images" section)**



The algorithm in pseudocode:

```

1  function expandKey(inputKey)
2      expanded := inputKey
3      bytesGenerated := 16
4      rconIteration := 1
5      temp := uint8[4]
6

```

```

7   while bytesGenerated < 176
8       temp = expanded[bytesGenerated - 4:bytesGenerated]
9
10      if bytesGenerated MOD 16 == 0 then
11          temp[0], temp[1], temp[2], temp[3] = temp[1], temp[2], temp[3], temp[0]
12          temp[0], temp[1], temp[2], temp[3] = sBox[temp[0]], sBox[temp[1]], sBox[temp[2]],
sBox[temp[3]]
13
14          temp[0] = temp[0] XOR rcon[rconIteration]
15          rconIteration = rconIteration + 1
16      end if
17
18      for i := 0 to 4
19          expanded[bytesGenerated] = expanded[bytesGenerated - 16] XOR temp[i]
20          bytesGenerated = bytesGenerated + 1
21      end
22      return expanded
23 end

```

The array of round keys starts off the exact same as the original key. Then if the length of the round key array is a multiple of 16 (which it is), the last 4 bytes of the previous round key (in this case the last 4 bytes of the original key) is:

1. Rotated (The first element of the 4 bytes is put at the end).
2. Substituted (Using the Rijndael Substitution-Box found at: https://en.wikipedia.org/wiki/Rijndael_S-box).
3. First byte of the 4 is XOR-ed with it's corresponding Round Constant (depending on the round number the key will be used in).
4. The result is appended to the array of round keys.

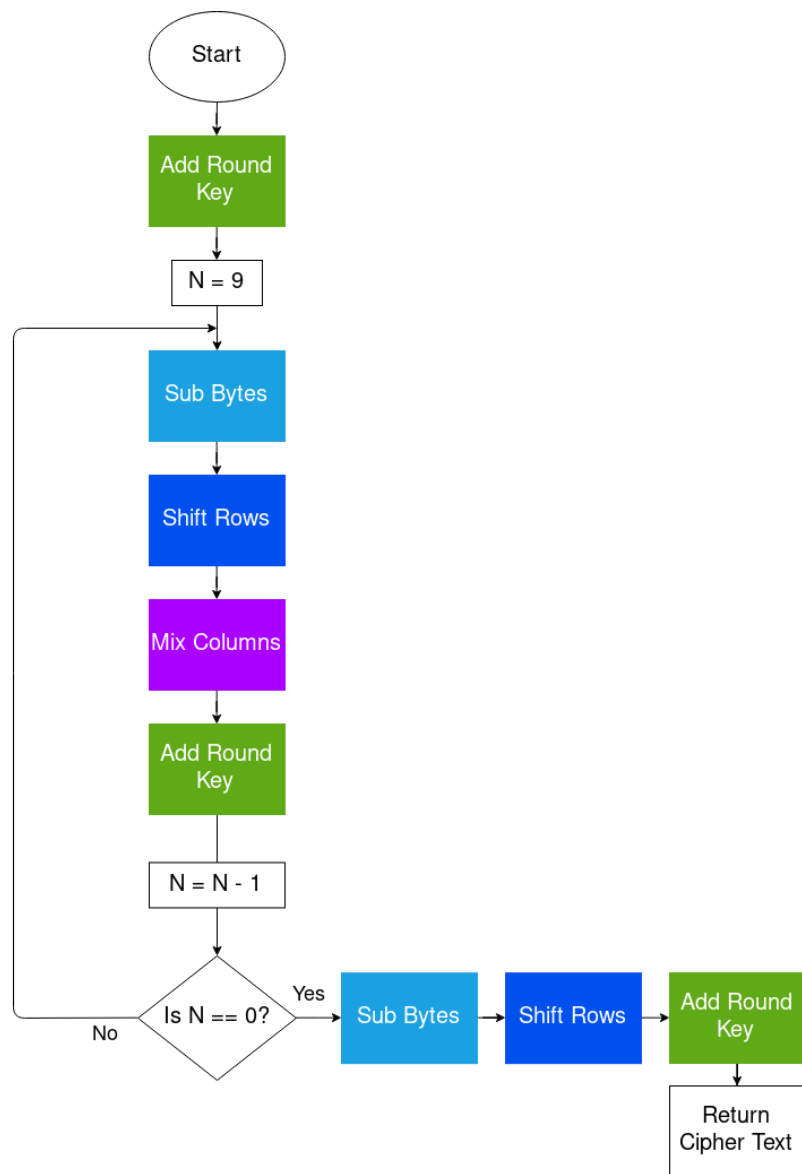
If the length of the round key array is not a multiple of 16, then the last 4 bytes in the array are XOR-ed with 4 bytes of the array that are 16 bytes before hand (shown in **Figure 1**).

This process is repeated until the length of the round key array is 176 bytes, then we will have one 16 byte key for each of the 11 rounds.

And that's all of the preparations done.

The operation:

Here is a diagram of the operation (I will explain each step in detail below):



In total there are 11 rounds (9 regular rounds). For each round, the corresponding round key (that we calculated beforehand) is used in the operation.

The 16 bytes in the state can be represented in a 4x4 grid, to make it easier to visualise what is happening at each stage:

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Add Round Key:

The Add Round Key step is literally just XOR-ing each byte in the current block of 16 bytes, with each byte in the 16 byte round key, and returning the state.

Here is pseudocode for the **Add Round Key** step:

```
1 function addRoundKey(state, roundKey)
2   for i := 0 to 15
3     state[i] = state[i] XOR roundKey[i]
4   return state
```

Sub Bytes:

Sub bytes substitutes each byte in the state with it's corresponding value in the Rijndael substitution box:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

When using the sub-box, you have to think of each byte as hexadecimal (0xYZ). Each row of the sub box is the value of the Y value (16s) in the hexadecimal representation of the byte. Each column of the sub box is the value of the Z value (1s) in the hexadecimal representation of the byte.

For example, if I had the hex `0x1A`, it would be substituted by the value: `0xA2`, as it is row "1", column "A".

Here is the pseudocode for the **Sub Bytes** step:

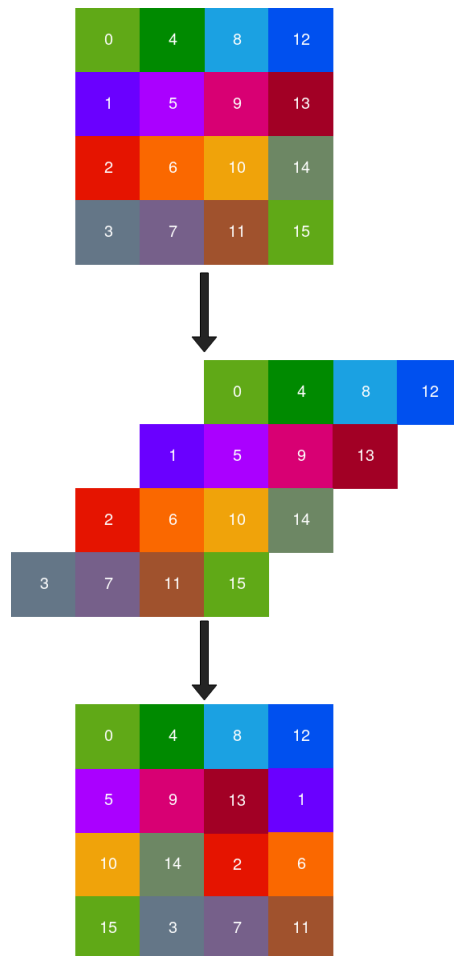
```
1 function subBytes(state)
2   for i := 0 to 15
3     state[i] = sBox[state[i]]
4   return state
```

It is pretty much the same as **Add Round Key** but instead of XORing you substitute each byte of the state with the corresponding byte in the sub-box (sBox).

Shift Rows:

Shift Rows shifts the rows (really?) left depending on the row number.

For example, the first row is shifted left by 0, second row shifted by 1 and so on:



Here is the algorithm for **Shift Rows**:

```

1  function shiftRows(state)
2      temp := []
3
4      temp[ 0] = state[ 0]
5      temp[ 1] = state[ 5]
6      temp[ 2] = state[10]
7      temp[ 3] = state[15]
8
9      temp[ 4] = state[ 4]
10     temp[ 5] = state[ 9]
11     temp[ 6] = state[14]
12     temp[ 7] = state[ 3]
13
14     temp[ 8] = state[ 8]
15     temp[ 9] = state[13]
16     temp[10] = state[ 2]
17     temp[11] = state[ 7]
18
19     temp[12] = state[12]
20     temp[13] = state[ 1]
21     temp[14] = state[ 6]
22     temp[15] = state[11]
23
24     return temp

```

The array is indexed to correspond to the images above.

Mix Columns:

Mix columns is the most confusing step of AES, so I will try to break it down into small pieces.

The mix columns calculation is this:

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Where r_0 to r_3 is the result of the operation, and a_0 to a_3 is the 4 bytes that make up the input column.

This is matrix multiplication, but we need to do dot product multiplication. This is where we multiply each corresponding element in each row of the pre-defined matrix (the one with numbers already in it), with the corresponding element in a_0 to a_3 , and then adds them up MOD2, also known as XOR (so that it is still 1 byte).

One way to represent this is like this:

$$\begin{aligned} r_0 &= (2 \times a_0) \oplus (3 \times a_1) \oplus (1 \times a_2) \oplus (1 \times a_3) \\ r_1 &= (1 \times a_0) \oplus (2 \times a_1) \oplus (3 \times a_2) \oplus (1 \times a_3) \\ r_2 &= (1 \times a_0) \oplus (1 \times a_1) \oplus (2 \times a_2) \oplus (3 \times a_3) \\ r_3 &= (3 \times a_0) \oplus (1 \times a_1) \oplus (1 \times a_2) \oplus (2 \times a_3) \end{aligned}$$

To dot product two binary numbers, they need to be represented using a Galois field.

A number can be represented by using a Galois field. A Galois field is just a way to represent a number as a polynomial, e.g $5x^2 + 2x + 3$, where x^2 is 10^2 , so the number of 100s in the number (for decimal), while x is the number of tens. In this case, this Galois field would represent the number 523, as there are 5 hundreds, 2 tens and 3 ones.

For example, if we wanted to represent the decimal number: 25301 as a Galois field, it would be:

$$2x^4 + 5x^3 + 3x^2 + 1$$

Note that the 0 in 25301 is not included, as $0x = 0$.

To represent a binary number, the same logic applies. For example, to represent the binary number 10011011 as a Galois field, it would be:

$$x^7 + x^4 + x^3 + x^1 + 1$$

To get back to decimal, we can replace the x with the number 2, as binary is base 2:

$$2^7 + 2^4 + 2^3 + 2^1 + 1 = 155 = 10011011$$

The dot product of two Galois fields is like expanding brackets: $(x + 2)(x + 3) = x^2 + 5x + 6$, which is $(x \times x) + (2 \times x) + (x \times 3) + (3 \times 2)$, so we just multiply each item in each bracket together.

Now I will do an example of doing one result (r_0) of mix columns.

Lets use these values of a_0 to a_3 for the example:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} d4 \\ d4 \\ d4 \\ d5 \end{bmatrix}$$

To get r_0 I have to do:

$$r_0 = (2 \times a_0) \oplus (3 \times a_1) \oplus (1 \times a_2) \oplus (1 \times a_3)$$

which is:

$$r_0 = (2 * d4) \oplus (3 * d4) \oplus (1 * d4) \oplus (1 * d5)$$

in this example.

I am using $d4, d4, d4, d5$ as test values as they are test vectors used on this page: https://en.wikipedia.org/wiki/Rijndael_MixColumns, to check that we get the right answer.

Now I need to convert the hex values $d4$ and $d5$ to binary:

$d4$ in binary is 11010100

$d5$ in binary is 11010101

Now i need to convert both of these into Galois fields:

$$\begin{aligned} 11010100 &= x^7 + x^6 + x^4 + x^2 \\ 11010101 &= x^7 + x^6 + x^4 + x^2 + 1 \end{aligned}$$

Then I need to multiply them all by their corresponding value in the pre-defined table expressed as a Galois field (e.g. $2 \equiv x$):

$$\begin{aligned} (x^7 + x^6 + x^4 + x^2)(x) &= x^8 + x^7 + x^5 + x^3 \\ (x^7 + x^6 + x^4 + x^2)(x + 1) &= x^8 + x^7 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 \\ &= x^8 + 2x^7 + x^6 + x^5 + x^4 + x^3 + x^2 \\ (x^7 + x^6 + x^4 + x^2)(1) &= x^7 + x^6 + x^4 + x^2 \\ (x^7 + x^6 + x^4 + x^2 + 1)(1) &= x^7 + x^6 + x^4 + x^2 + 1 \end{aligned}$$

But hang on a second, the answer to $d4 * 3$ and $d4 * 2$ both have a x^8 term, which means it's bigger than 255 (since $2^8 = 256$), so it is no longer a byte, which means that it no longer fits in with 128 bit AES.

To fix this, we replace all of the x^8 terms with this pre-determined polynomial (Rijndael's finite field), reducing by MOD2 as we go along:

$$x^8 \equiv x^4 + x^3 + x + 1$$

Let's try this with $d4*3$:

$$\begin{aligned} 3d4 &= x^8 + 2x^7 + x^6 + x^5 + x^4 + x^3 + x^2 \\ &= (x^4 + x^3 + x + 1) + 2x^7 + x^6 + x^5 + x^4 + x^3 + x^2 \\ &= 2x^7 + x^6 + x^5 + 2x^4 + 2x^3 + x^2 + x + 1 \\ &= 0x^7 + x^6 + x^5 + 0x^4 + 0x^3 + x^2 + x + 1 \quad \text{Here is where I did MOD2} \\ &= x^6 + x^5 + x^2 + x + 1 \end{aligned}$$

Again with $d4*2$:

$$\begin{aligned}
2d4 &= x^8 + x^7 + x^5 + x^3 \\
&= (x^4 + x^3 + x + 1) + x^7 + x^5 + x^3 \\
&= x^7 + x^5 + x^4 + 2x^3 + x + 1 \\
&= x^7 + x^5 + x^4 + x + 1
\end{aligned}$$

Now, with our new values for a_0 to a_3, we can finally do the equation:

$$\begin{aligned}
r_0 &= (2 \times d4) \oplus (3 \times d4) \oplus (1 \times d4) \oplus (1 \times d5) \\
r_0 &= (x^7 + x^5 + x^4 + x + 1) \oplus (x^6 + x^5 + x^2 + x + 1) \oplus (x^7 + x^6 + x^4 + x^2) \oplus (x^7 + x^6 + x^4 + x^2 + 1) \\
r_0 &= (2^7 + 2^5 + 2^4 + 2 + 1) \oplus (2^6 + 2^5 + 2^2 + 2 + 1) \oplus (2^7 + 2^6 + 2^4 + 2^2) \oplus (2^7 + 2^6 + 2^4 + 2^2 + 1)
\end{aligned}$$

$$\begin{array}{r}
r_0 = 10110011 \\
 01100111 \\
 11010100 \\
\oplus 11010101 \\
\hline
= 11010101
\end{array}$$

$$r_0 = 213(decimal)$$

And, thank god, that is the correct answer for the test vector on this page: https://en.wikipedia.org/wiki/Rijndael_MixColumns.

To get r_1 , r_2 and r_3 , you repeat the process using the equations for each defined at the top of this section.

This whole process has to be done on each column.

On a computer, this would be very demanding on the processor, however since the range of the inputs is 0-255 (since the number has to be represented by 1 byte), you can make a lookup table with all of the 256 possible outputs, for each of the multiplications, for each of the 256 possible inputs. This drastically increases speed, and also makes it easier to program. You would have a table for multiplication by 2 and 3, and for the inverse function of Mix Columns you would need multiplication by 9, 11 and 13.

This trades a few kilobytes of memory for a drastic improvement in speed.

This makes the pseudocode for **Mix Columns** very simple:

```

1 // mul2 and mul3 are the pre-defined tables talked about above.
2 function mixColumns(state)
3     temp := []
4
5     temp[ 0] = mul2[state[0]] XOR mul3[state[1]] XOR state[2] XOR state[3]
6     temp[ 1] = state[0] XOR mul2[state[1]] XOR mul3[state[2]] XOR state[3]
7     temp[ 2] = state[0] XOR state[1] XOR mul2[state[2]] XOR mul3[state[3]]
8     temp[ 3] = mul3[state[0]] XOR state[1] XOR state[2] XOR mul2[state[3]]
9
10    temp[ 4] = mul2[state[4]] XOR mul3[state[5]] XOR state[6] XOR state[7]
11    temp[ 5] = state[4] XOR mul2[state[5]] XOR mul3[state[6]] XOR state[7]
12    temp[ 6] = state[4] XOR state[5] XOR mul2[state[6]] XOR mul3[state[7]]
13    temp[ 7] = mul3[state[4]] XOR state[5] XOR state[6] XOR mul2[state[7]]
14
15    temp[ 8] = mul2[state[8]] XOR mul3[state[9]] XOR state[10] XOR state[11]
16    temp[ 9] = state[8] XOR mul2[state[9]] XOR mul3[state[10]] XOR state[11]
17    temp[10] = state[8] XOR state[9] XOR mul2[state[10]] XOR mul3[state[11]]
18    temp[11] = mul3[state[8]] XOR state[9] XOR state[10] XOR mul2[state[11]]
19
20    temp[12] = mul2[state[12]] XOR mul3[state[13]] XOR state[14] XOR state[15]
21    temp[13] = state[12] XOR mul2[state[13]] XOR mul3[state[14]] XOR state[15]
22    temp[14] = state[12] XOR state[13] XOR mul2[state[14]] XOR mul3[state[15]]
23    temp[15] = mul3[state[12]] XOR state[13] XOR state[14] XOR mul2[state[15]]
24

```

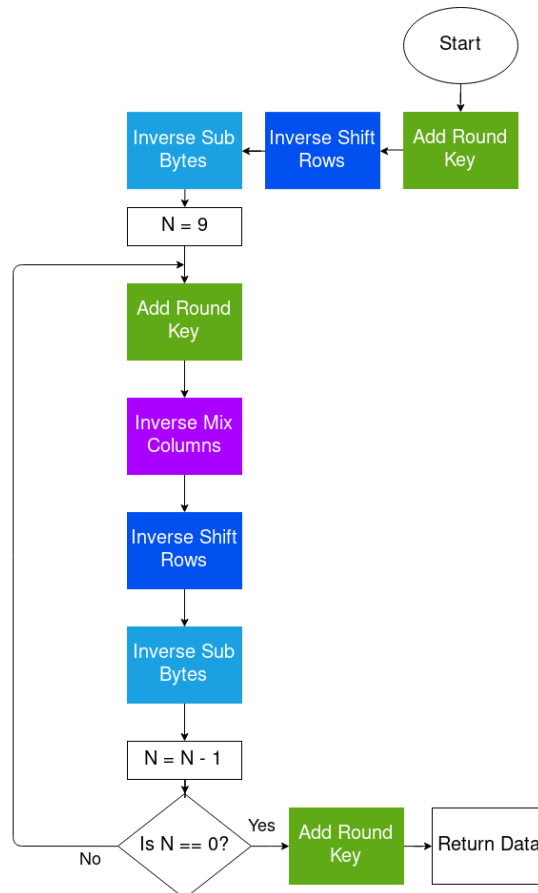
```

25 |     return temp
26 | }
27 |

```

Decryption

Decryption is just encryption, but in reverse. This uses the inverse functions of each function used to encrypt the data. Here is the algorithm:



It is literally just the encryption algorithm in reverse.

Before decryption, the exact same steps need to be taken as in encryption, apart from the padding because all the blocks should have already been encrypted, so each block should be 16 in length.

Inverse Add Round Key:

Add round key is its own inverse, as XOR is the same forwards as it is backwards.

Inverse Sub Bytes:

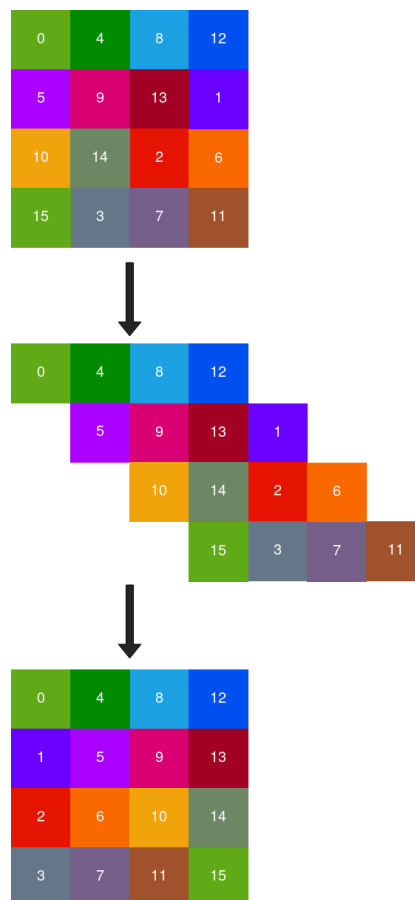
Inverse sub bytes is the same as sub bytes, it just has an inverse of the S-Box.

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Inverse Shift Rows:

Inverse shift rows does what shift rows does, but shifts each row right instead of left.

In the diagram below it takes the shifted data and orders it again.



Inverse Mix Columns:

Inverse mix columns works the same as normal mix columns, but with a different matrix to multiply each element with:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix}$$

The a's are the original data, the r's are the encrypted data.

Just like with normal mix columns, you can just use lookup tables for each possible answer to each possible input.

And that's all for AES.

SHA256:

SHA256 (in the Secure Hash Algorithm 2 family) takes an input of 32 bytes (256 bits), and gives a 32 byte output based on the input, but is meaningless. This is useful for passwords, or pin codes like in my program, where you don't want the original password to be known, but for the password to still be unique.

A small difference in the input gives you a drastic change in the output. For example, if I put in:

```
1 | "test string"
```

I get:

```
1 | d5579c46dfcc7f18207013e65b44e4cb4e2c2298f4ac457ba8f82743f31e930b
```

But when I put in:

```
1 | "test strinh"
```

I get:

```
1 | 4e4d20e9fc77e913bf56cc69a2b4685d761a9e44d833198612e80a72dcd563f1
```

A vastly different output to the one above. This is important, as there should be no pattern to the output, otherwise the original password could be guessed based off of similar inputs.

Now you might be asking "Why are you using 256 bit SHA, when size key you need for AES is 128 bits?". It is because the more bits you have, the less likely you are to have collisions with other inputs. The security of SHA-1 (128 bit SHA) (measured in bits) is less than 63 bits due to collisions (if it was fully secure it would be the full 128 bits).

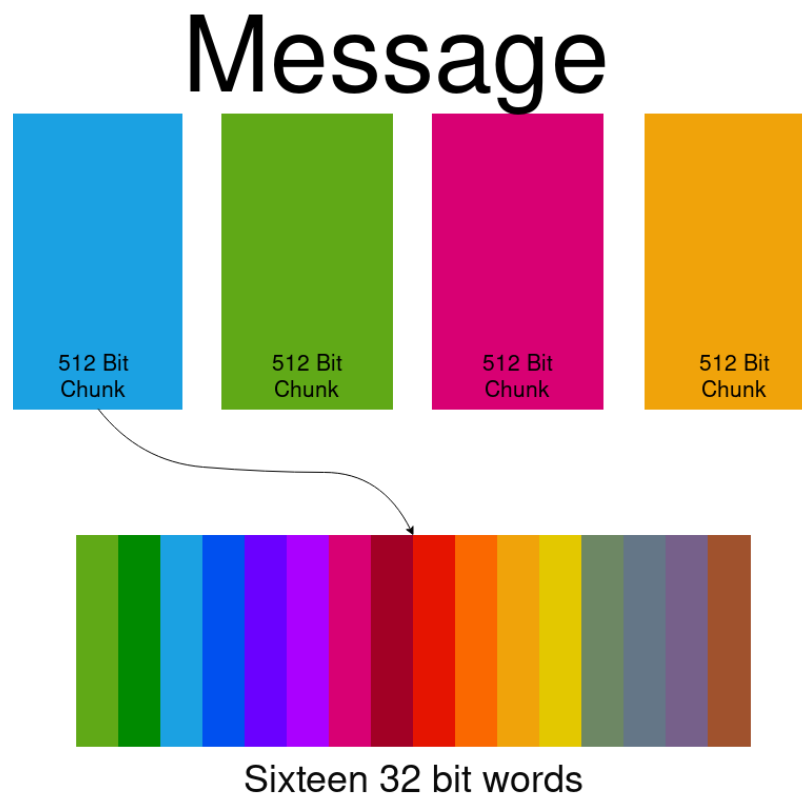
What I am doing instead, is taking the output of SHA256, splitting it in half, and XORing each half with each other to get a 128 bit output. This doesn't affect how secure it is, as you still have the extra step of XOR, making it still more secure than SHA-1.

The Algorithm:

Bear in mind that SHA works on a bitwise level, so while I will be explaining it, I will be talking in terms of bits.

How the message is handled:

When doing operations on the data, it will be done in 32 bit words. The message is split into 512 bit blocks, containing sixteen 32 bit words.



SHA is operates on every 32 bit word.

Since the maximum key size for my AES will be 16 bytes (128 bits), I don't need to worry about splitting the message into 512 bit chunks, as the input will only ever be 128 bits as SHA will only ever be used for the AES key. So, for the examples below I won't go into detail on how a message bigger than 512 bits will be handled.

Before the operation starts:

Before we start, we need to **pad the message** M so that it is 512 bits in length.

Let l = the length of the message M .

First, we need to append the bit 1 to the end of the message, followed by k 0 bits, where k is the smallest positive solution to the equation:

$$l + 1 + k \equiv 448 \pmod{512}$$

To get k , the algorithm would look something like this (I wrote this in Python 3):

```

1 | k = 0
2 | while ((l+1+k)-448) % 512 != 0:
3 |     k += 1

```

Then, you append the binary representation of the length of the message l as a 64 bit binary number. This makes the message 512 bits in length.

Let's do an example: $M = \text{"i don't know"}$.

$$l = 12 \times 8 = 96$$

Append a "1":

$$M = \text{"i don't know"} + 1$$

$$448 - (96 + 1) = 351 \text{ Zero Bits}$$

$$M = \text{"i don't know"} + 1 + 351(0s)$$

$$l = 96 = 01100000$$

Final Padded Message:

$$M = \text{"i don't know"} + 1 + 351(0s) + 56(0s) + 01100000$$

The message has to be 512 bits in length so that it works with the calculations later.

Then, we also need to **set the initial hash values** for each word in the current block. The initial hash values set by the creators of SHA:

"These words were obtained by taking the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers. "

$$H_0 = 6a09e667$$

$$H_1 = bb67ae85$$

$$H_2 = 3c6ef372$$

$$H_3 = a54ff53a$$

$$H_4 = 510e527f$$

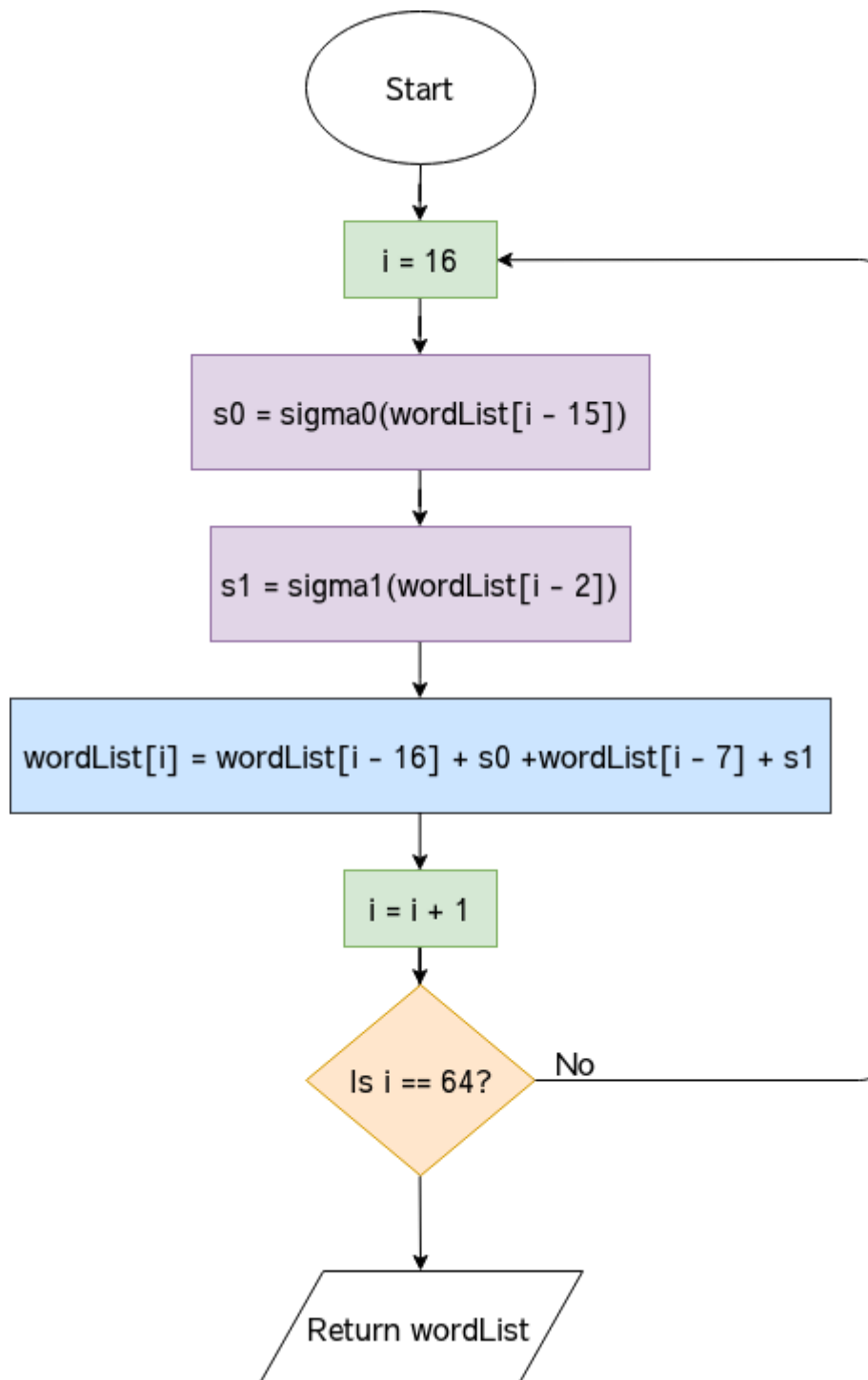
$$H_5 = 9b05688c$$

$$H_6 = 1f83d9ab$$

$$H_7 = 5be0cd19$$

Next, each 32 bit word in the message has to be expanded from 32 bits to 64 bits.

Here is the algorithm:



To do this, we need two functions, sigma 0 σ_0 and sigma 1 σ_1 .

Sigma 0 (Expansion) (σ_0):

Sigma 0 (Expansion) looks like this:

$$\sigma_0(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3)$$

\ggg means that we rotate the 32 bit word x right by the number given (y). What this does is shift the bytes along y places to the right, and wraps them around to the start of x .

I will do an example of \ggg with a 4 bit nibble:

$$\begin{aligned}
 f(x) &= x \ggg 1 \\
 f(1011) &= 1011 \ggg 1 \\
 f(1011) &= 1101
 \end{aligned}$$

As you can see, the 1 bit at the end gets moved to the front, as I shifted it right by 1.

\gg Means shift the 32 bit word x right by the number given (y). This is different from \ggg , because instead of wrapping the bits around to the beginning of the word again, we just shove a 0 bit at the front instead.

\oplus is just XOR.

For example:

$$\begin{aligned}
 f(x) &= x \gg 1 \\
 f(1011) &= 1011 \gg 1 \\
 f(1011) &= 0101
 \end{aligned}$$

$$\begin{aligned}
 g(x) &= x \gg 2 \\
 g(0101) &= 0101 \gg 2 \\
 g(0101) &= 0001
 \end{aligned}$$

Here the byte is shifted right, and the bytes are removed as they are shifted.

Sigma 1 (Expansion)(σ_1):

Sigma 1(Expansion)(σ_1) is the same as Sigma 0 (Expansion)(σ_0), apart from how much you rotate and shift the word:

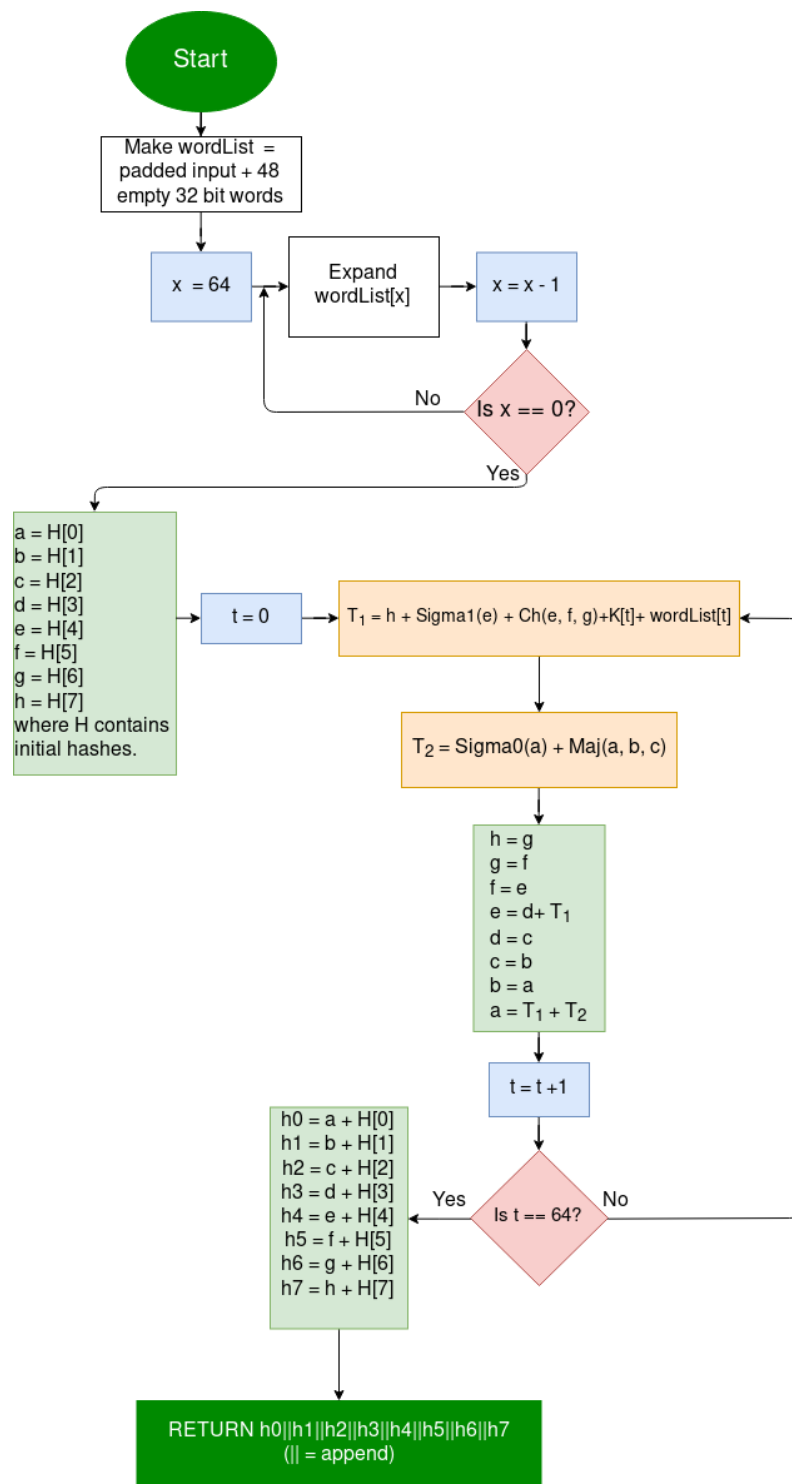
$$\sigma_0(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10)$$

The operation:

All addition is MOD(2^{32}).

Here is the full algorithm:

Figure 2 (Found larger on "Large Images" section)



In the diagram above, H is the array of initial hash values discussed earlier, $wordList$ is a 2D array containing the 32 bit words. $||$ means append, so $h0||h1||h2||\dots$ just appends the items together. K is the array with the round constants in (see <https://csrc.nist.gov/csrc/media/publications/fips/180/4/archive/2012-03-06/documents/fips180-4.pdf> section 4.2.2).

The step "Expand $wordList[x]$ " is covered in the section above.

All of the SHA functions operate on 32 bit words, and return a new 32 bit word. I will now explain what the functions Σ_0 (Σ_0), Σ_1 (Σ_1), Ch and Maj .

Sigma 0 (Σ_0):

Σ_0 is this equation:

$$\Sigma_0(x) = (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22)$$

This looks confusing, but let me break it down.

\ggg means that we rotate (shift and move displaced numbers to the begining/end of the number) the number right by the number specified.

\oplus means that we XOR the items either side with each other.

Here is an example of the rotate function:

$$\begin{aligned} A &= 1001110 \\ A \ggg 2 &= 1010011 \quad \text{The last two bits are moved to the end.} \end{aligned}$$

Let me do an example with a 32 bit word:

$$\begin{aligned} A &= 10010111011011111000110111011101 \\ \Sigma_0 &= (10010111011011111000110111011101 \ggg 2) \oplus (10010111011011111000110111011101 \ggg 13) \oplus \\ &\dots (10010111011011111000110111011101 \ggg 22) \\ (10010111011011111000110111011101 \ggg 2) &= 01100101110110111110001101110111 \\ \text{The two bits at the end have been moved to the front one by one.} \\ (10010111011011111000110111011101 \ggg 13) &= 11110001101110111011001011101101 \\ (10010111011011111000110111011101 \ggg 22) &= 01110111011001011101101111100011 \\ 01100101110110111110001101110111 &\oplus 11110001101110111011001011101101 \oplus 01110111011001011101101111100011 \\ &= 11100011000001011000101001111001 \end{aligned}$$

Sorry if that is a bit small.

It isn't too difficult it's just understanding what the \ggg does.

Sigma 1 (Σ_1):

Sigma 1 (Σ_1) is pretty much the same as Σ_0 , the only difference being the amount you rotate by:

$$\Sigma_1(x) = (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25)$$

Ch:

The Ch function looks like this:

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$

This also looks a bit confusing, but it really isn't too bad.

The \wedge symbol is the bitwise operator AND.

The \oplus symbol is the bitwise operator XOR.

The \neg symbol is the bitwise operator NOT.

I will do one example run with Ch with three 4 bit nibbles to keep it simple:

$$Ch(1011, 1001, 0011) = (1011 \wedge 1001) \oplus (\neg 1011 \wedge 0011)$$

$$\begin{array}{r} 1011 \\ \wedge 1001 \\ \hline = 1001 \end{array}$$

$$Ch(1011, 1001, 0011) = 1001 \oplus (\neg 1011 \wedge 0011)$$

$$\neg 1001 = 0110$$

$$\begin{array}{r} 0110 \\ \wedge 0011 \\ \hline = 0010 \end{array}$$

$$Ch(1011, 1001, 0011) = 1001 \oplus 0010$$

$$\begin{array}{r} 1001 \\ \oplus 0010 \\ \hline = 1011 \end{array}$$

$$Ch(1011, 1001, 0011) = 1011$$

Maj:

the Maj function looks like this:

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

You should recognise the symbols in this one, since they appear in the other ones used in SHA that we have covered. Here is an example with three 4 bit nibbles:

$$Maj(1011, 1001, 0011) = (1011 \wedge 1001) \oplus (1011 \wedge 0011) \oplus (1001 \wedge 0011)$$

$$\begin{array}{r} 1011 \\ \wedge 1001 \\ \hline = 1001 \end{array}$$

$$\begin{array}{r} 1011 \\ \wedge 0011 \\ \hline = 0011 \end{array}$$

$$\begin{array}{r} 1001 \\ \wedge 0011 \\ \hline = 0001 \end{array}$$

$$Maj(1011, 1001, 0011) = 1001 \oplus 0011 \oplus 0001$$

$$\begin{array}{r} 1001 \\ 0011 \\ \oplus 0001 \\ \hline = 0101 \end{array}$$

$$Maj(1011, 1001, 0011) = 0101$$

BLAKE 2b:

BLAKE was a finalist in the SHA 3 contest. The SHA 3 contest was announced on November 2nd 2007, as a new hash function was needed, that was very different from the SHA 2 family of hash functions in case a huge issue was found with the SHA 2 family.

BLAKE did not win, as it was too similar to SHA2:

"desire for SHA-3 to complement the existing SHA-2 algorithms ... BLAKE is rather similar to SHA-2."

<https://blake2.net/acns/slides.html>

However, BLAKE was the fastest out of all of the competitors (at 8.4 cycles per byte, cycles being the fetch decode execute cycle of a processor), and was tested to be secure. This meant that even though BLAKE did not win the competition, it is still used in numerous programs. Due to BLAKE's speed, it is ideal for getting the checksum of large data.

No preparations have to be done so lets just jump right into the algorithm.

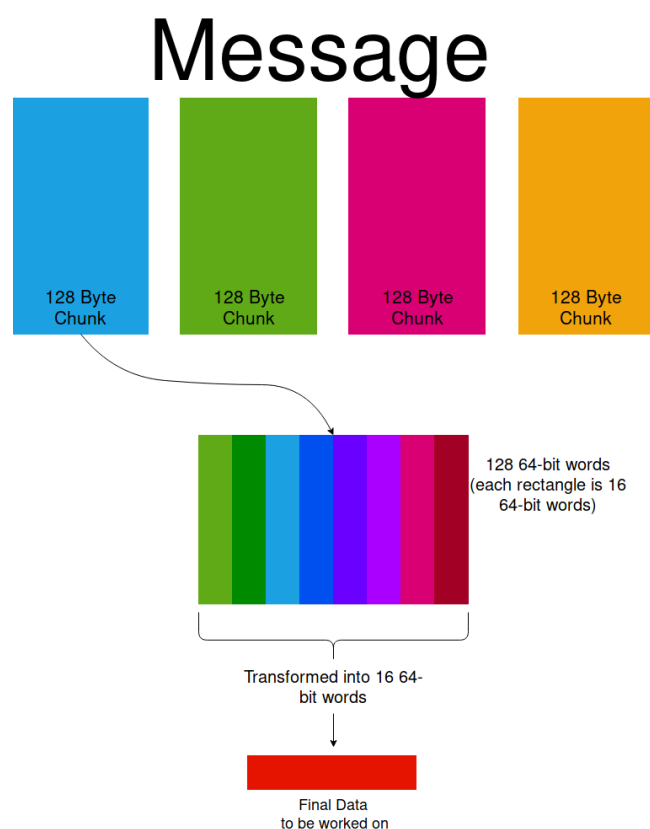
The Algorithm:

How the data is read:

8 initial hash values of size 64-bits are initialised at the start (using pre-defined values), and these are worked on throughout the program.

The data is read in 128 bytes, where each byte is then converted into a 64-bit word (just shove some 0s on the front). Each chunk is operated on using the 8 hash values, creating 8 new hash values. These new hash values are used in computation using the next block and so on.

Here is a diagram showing how the data is converted into data that can be processed:

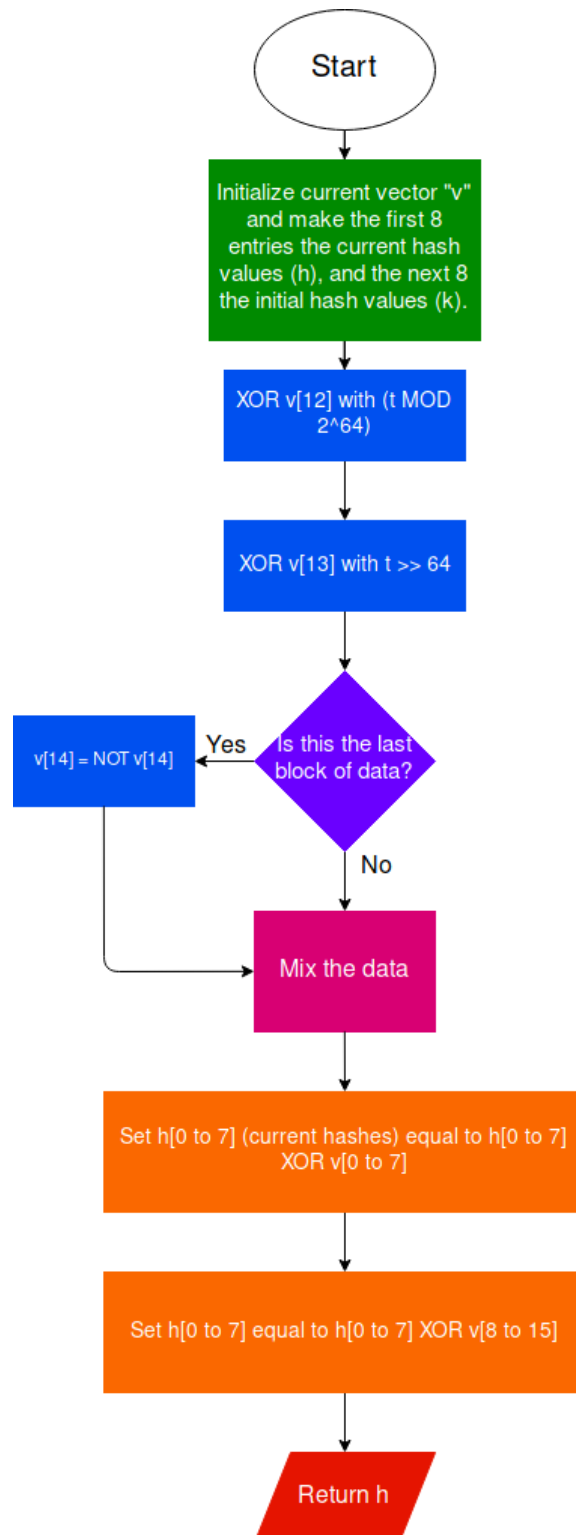


To transform a list of 16 64-bit words into 1 64-bit word, you do this algorithm (where a is the list of words):

$$new = a[0] \oplus (a[1] \ll 8) \oplus (a[2] \ll 16) \oplus (a[3] \ll 24) \oplus (a[4] \ll 32) \oplus (a[5] \ll 40) \oplus (a[6] \ll 48) \oplus (a[7] \ll 56)$$

What this does is XOR's the bytes in the array with each other in a way that produces a single word at the end.

The operation:



Each block has to be compressed and returned as 8 hash values. Above is the compression function. t is the number of bytes in total that have been compressed so far, h is a list of the 8 current hashes, and k is the list of 8 initial hash values set here <https://tools.ietf.org/pdf/rfc7693.pdf> section 2.6, the same initial hash values of SHA512.

The operation is quite simple compared to other hash functions like SHA512, as it was built for speed.

The **Mix the data** step looks like this:

```

1  for i := 0 to 12
2      v = mix(v, 0, 4, 8, 12, m[sigma[i][0]], m[sigma[i][1]])
3      v = mix(v, 1, 5, 9, 13, m[sigma[i][2]], m[sigma[i][3]])
4      v = mix(v, 2, 6, 10, 14, m[sigma[i][4]], m[sigma[i][5]])
5      v = mix(v, 3, 7, 11, 15, m[sigma[i][6]], m[sigma[i][7]])
6
7      v = mix(v, 0, 5, 10, 15, m[sigma[i][ 8]], m[sigma[i][ 9]])
8      v = mix(v, 1, 6, 11, 12, m[sigma[i][10]], m[sigma[i][11]])
9      v = mix(v, 2, 7, 8, 13, m[sigma[i][12]], m[sigma[i][13]])
10     v = mix(v, 3, 4, 9, 14, m[sigma[i][14]], m[sigma[i][15]])

```

Sigma (σ) is a 2-dimensional array containing some constant values, that determine what index of the current working vector v (a 16 length array of 64-bit words) will be mixed with what other index of v . Sigma is defined here: <https://tools.ietf.org/pdf/rfc7693.pdf> section 2.7 as:

$$\begin{aligned}
 \sigma[0] &= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] \\
 \sigma[1] &= [14, 10, 4, 8, 9, 15, 13, 6, 1, 12, 0, 2, 11, 7, 5, 3] \\
 \sigma[2] &= [11, 8, 12, 0, 5, 2, 15, 13, 10, 14, 3, 6, 7, 1, 9, 4] \\
 \sigma[3] &= [7, 9, 3, 1, 13, 12, 11, 14, 2, 6, 5, 10, 4, 0, 15, 8] \\
 \sigma[4] &= [9, 0, 5, 7, 2, 4, 10, 15, 14, 1, 11, 12, 6, 8, 3, 13] \\
 \sigma[5] &= [2, 12, 6, 10, 0, 11, 8, 3, 4, 13, 7, 5, 15, 14, 1, 9] \\
 \sigma[6] &= [12, 5, 1, 15, 14, 13, 4, 10, 0, 7, 6, 3, 9, 2, 8, 11] \\
 \sigma[7] &= [13, 11, 7, 14, 12, 1, 3, 9, 5, 0, 15, 4, 8, 6, 2, 10] \\
 \sigma[8] &= [6, 15, 14, 9, 11, 3, 0, 8, 12, 2, 13, 7, 1, 4, 10, 5] \\
 \sigma[9] &= [10, 2, 8, 4, 7, 6, 1, 5, 15, 11, 9, 14, 3, 12, 13, 0]
 \end{aligned}$$

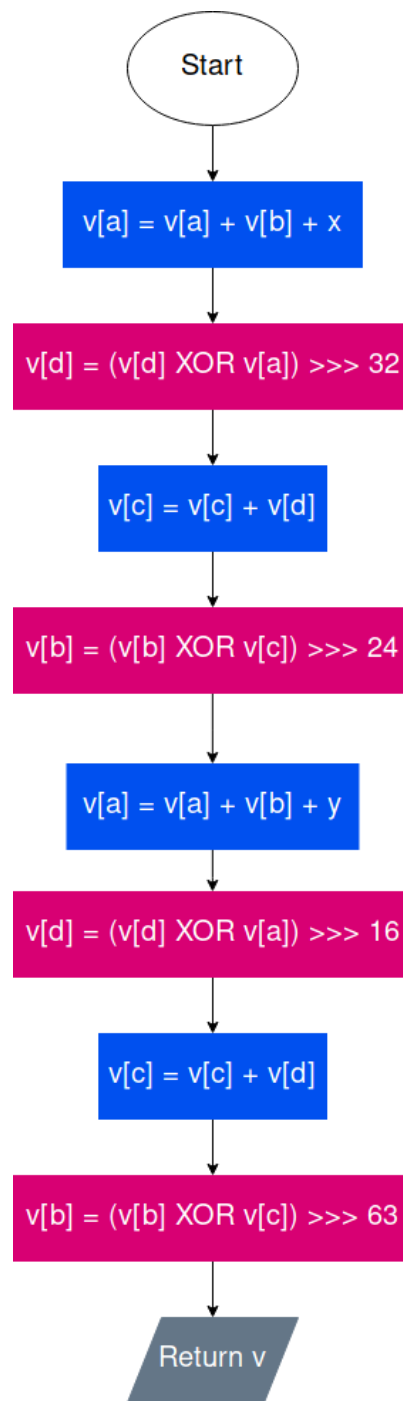
σ is defined for BLAKE2s, and BLAKE2s only has 10 rounds, while BLAKE2b has 12, so σ_0 and σ_1 are repeated again to make the array 12 in length.

Notice that in the first lot of mixing, the vector is mixed row by row normally (with the same indexing as AES), but in the second lot of mixing, the indices change. They shift each column up depending on the column. Column 0 is shifted 0 places, column 1 is shifted 1 place up, column 2 is shifted 2 places up, and column 3 is shifted 3 places up. This is a much better way of shifting each column than doing it before hand.

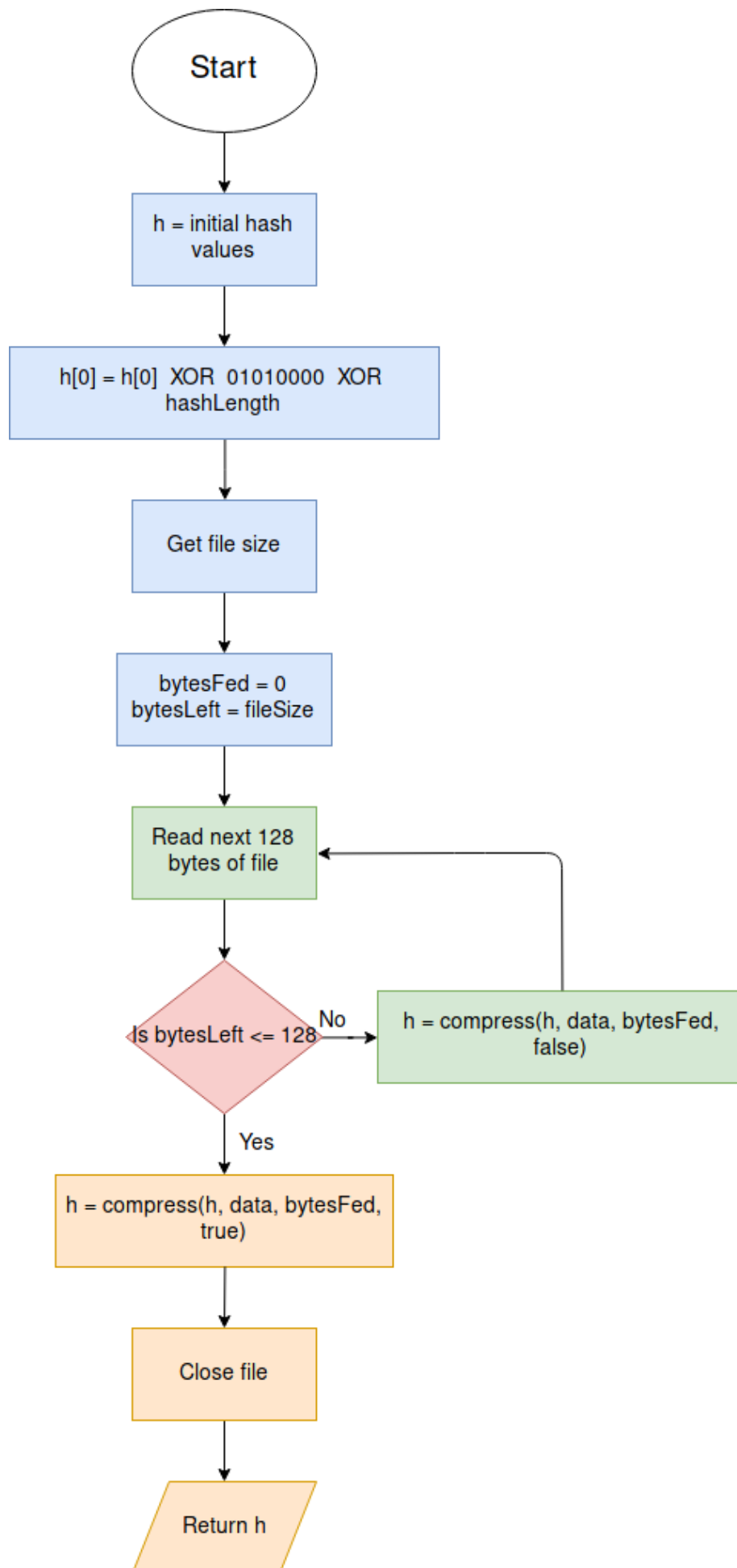
The main mixing function takes the inputs:

$$mix(v, a, b, c, d, x, y)$$

Where v is the current vector (16 64-bit words), a, b, c, d, x , and y are the indices of the working vector you want to work with. Here is the main mixing algorithm:



So all together, this is the BLAKE2b checksum algorithm:



The second step ($h[0] = h[0] \oplus 01010000 \oplus hL$) XORs $h[0]$ with $0101kknn$, where kk is the length of the key (which is optional, so I probably will never use it), and nn is the hash length desired.

Quick Sort

My program will need a quick sort for sorting the files by:

- Size
- Name

I have chosen quick sort because it is quicker than most sorts (it's in the name!) with a big-O notation of $O(n \log n)$ on average, with the worst case being $O(n^2)$. Merge sort has a big-O notation of $O(n \log n)$, and worst case of $O(n \log n)$, so why am I not using merge sort? Merge sort is supposed to be quicker mathematically, however merge sort has to access the array of items more often, usually resulting in putting more strain on the hardware, and also slows the overall process down because getting items from memory takes a fair amount of time. Here is a good video comparing merge sort and quick sort (along with a few other algorithms): <https://youtu.be/ZZuD6iUe3Pc>

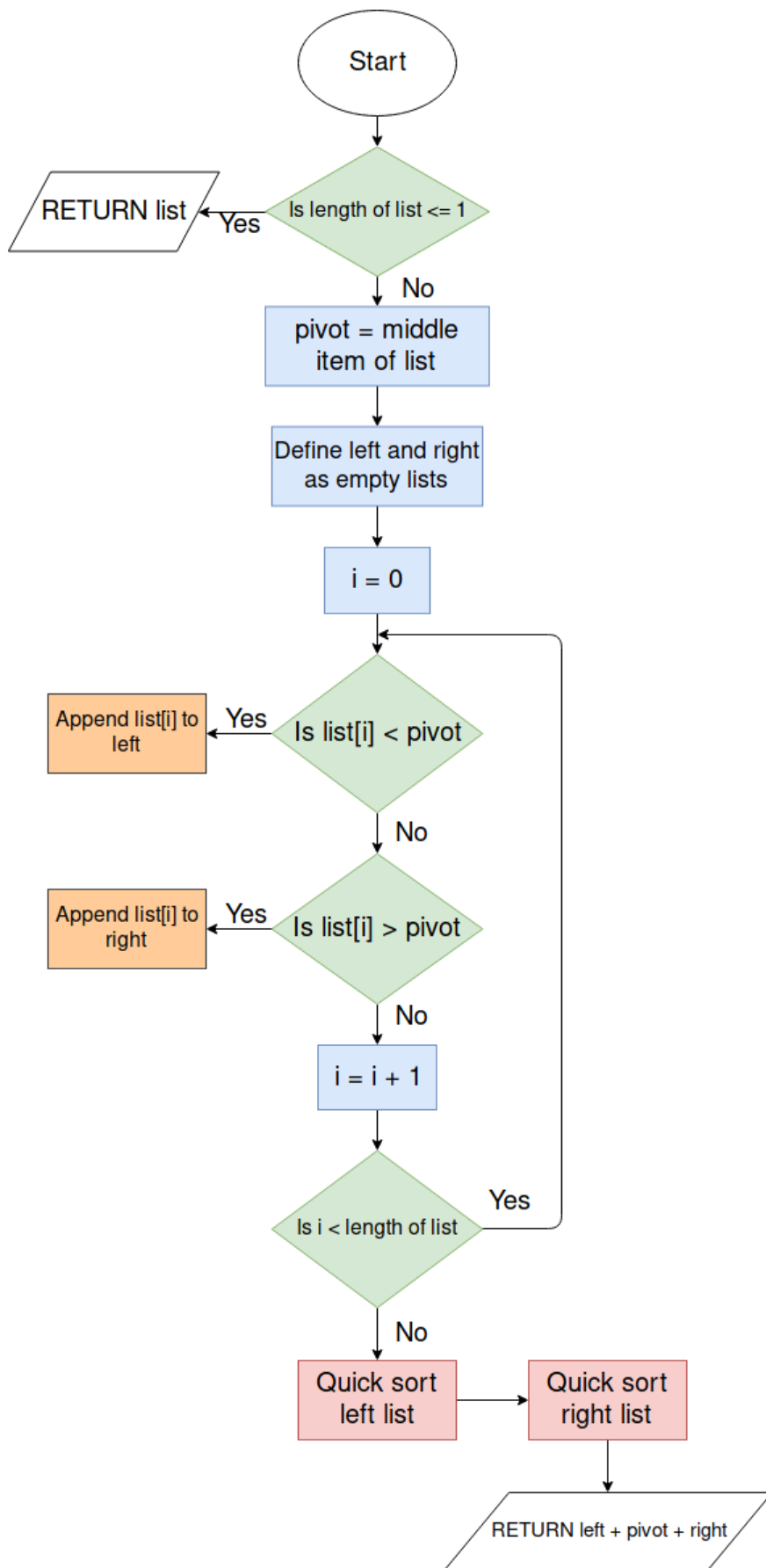
The algorithm goes like this (using a list of items to be sorted):

1. Take the item in the middle of the list. Call this the "pivot".
2. Compare each item either side of the pivot. If the item is bigger than the pivot, add it to a new list called "right", if the item is smaller than the pivot, add the item to a new list called "left".
3. Then repeat this process with the left and right lists (making this algorithm recursive).
4. Once the current left and right lists have been sorted, append the left list and right list with the pivot in the middle.

Here is the pseudocode of the algorithm:

```
1  function quickSort(list)
2      if length(list) <= 1 then
3          return list
4      else
5          left  = []
6          middle = []
7          right = []
8          pivot = list[int(length(list)/2)]
9          for i = 0 to length(list) do
10             if list[i] < pivot then
11                 left.append(list[i])
12             else if list[i] > pivot then
13                 right.append(list[i])
14             else
15                 middle.append(list[i])
16             end
17         end
18         return quickSort(left)+middle+quickSort(right)
19     end
20 end
```

Here is a flow diagram to represent this:

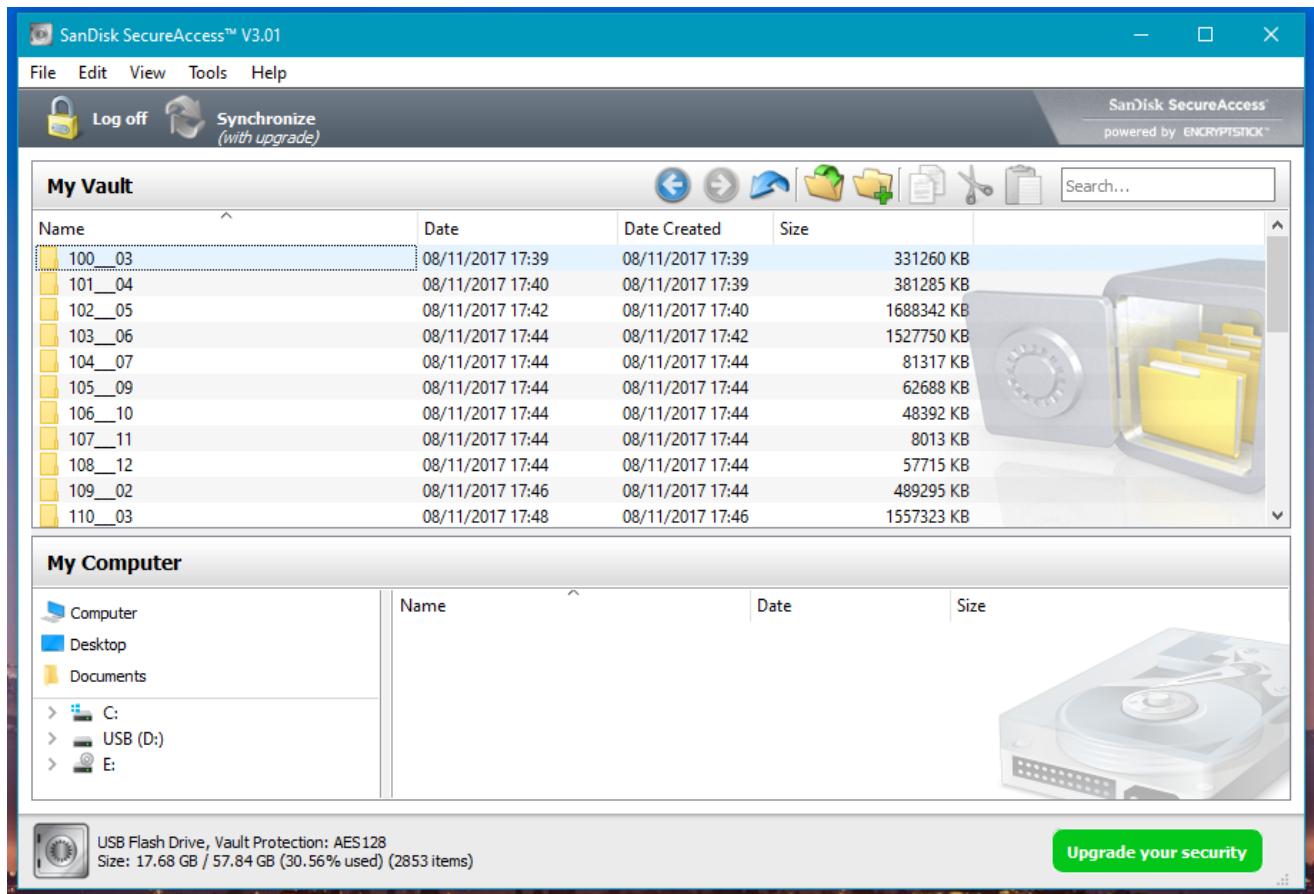


UI Research:

For the UI of both apps, I will use Kivy (a Python module) to make both the mobile app and the PC program. I have chosen Kivy because using it on both the app and the main program means that the design will stay consistent, and Kivy does look quite nice "out of the box".

Main Program (on PC):

The main program has to be designed to be easy to use, and actions that are used a lot should be easily accessible. I think I will go for a similar layout to a program that already exists, SanDisk Secure Access:



SanDisk Secure Access did inspire this project, however I do not want to make a carbon copy of it. I will take what SanDisk have done right, and improve the areas they lacked on.

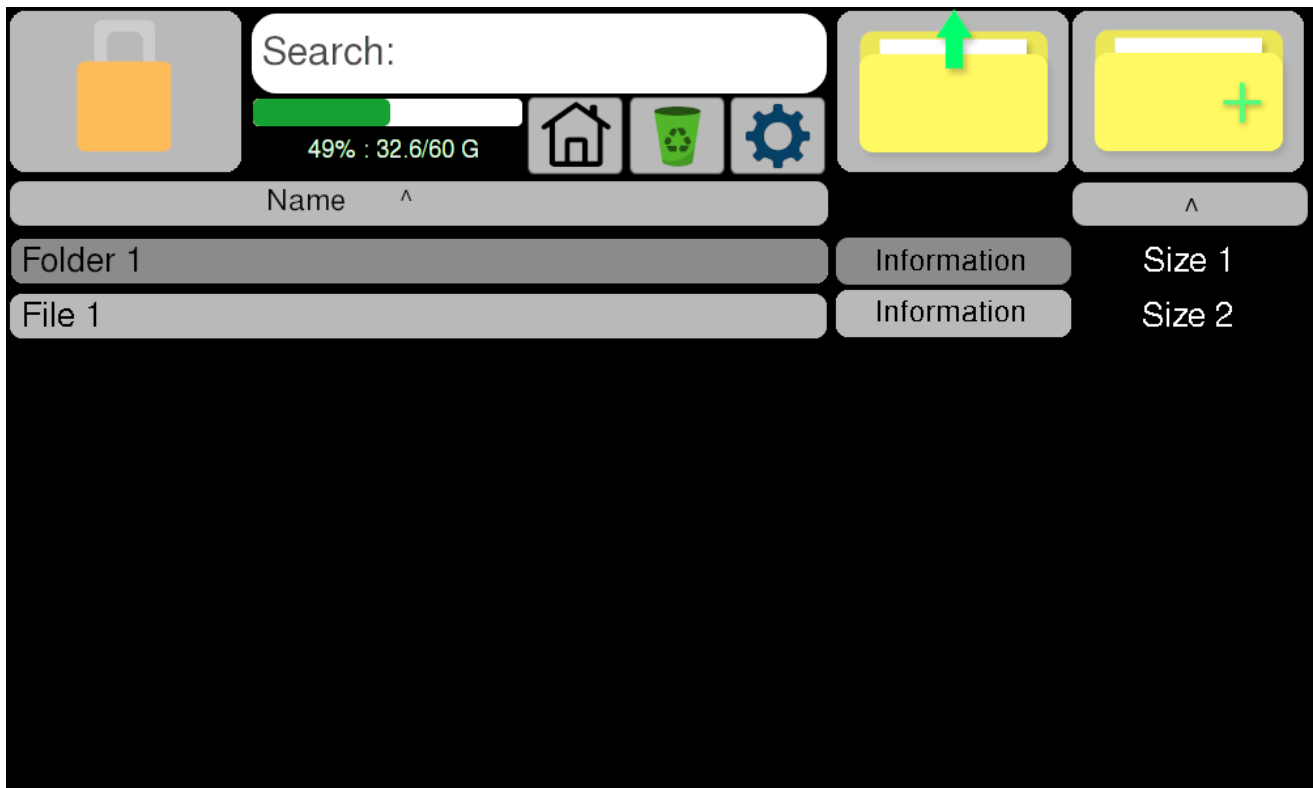
SanDisk did these things right:

- The layout is pretty good because all buttons you would need regularly are available, and it doesn't differ too much in design from the Windows file explorer, so it feels familiar to it's users.
- Shows the user how much space is left on their device.
- Shows useful information about each file.
- The user can easily sort the list of files however they want.
- More options are hidden unless needed regularly.
- Allows the user to search the vault for a file.
- I can easily drag files in and out of the program.

What I think SanDisk did not do too well:

- Looks a bit cluttered with all the extra stuff at the bottom. If I wanted to see other files on my computer I would open my file manager, and if I wanted to add files to the vault I can just drag it in easily.
- Faded pictures in the background are distracting.
- Some buttons are quite small, so may be hard for some users to click.
- Aesthetically alright but could be better.
- Some icons are confusing when first using the program (like the folder with the green arrow inside of it; too much going on).
- Size is displayed in kilobytes, which is alright but is kind of hard to read for files larger than 1 megabyte.

Taking all of these points into consideration, here is a possible design for the UI of my program:



Everything grey is a clickable button. This helps the user distinguish between buttons and information. The most important buttons are large, as they will be used the most. The user can sort by name or size, and can search the entire vault for a search term.

The information button displays more information, such as:

- The time the file (if it is a file) was added to the vault.
- The full directory path from the vault.
- The size of the file/folder.
- The option to delete the file/folder.

The button with the home picture on it takes the user back to the root directory of the vault. The recycling bin button is for the recycling folder, where the files that have been deleted can be either restored or deleted. The cog wheel button is settings, where all the settings are kept. I gave the settings it's separate section to avoid clutter, as most users will probably not need to use it very often.

The user can sort the files by name alphabetically, or they can sort by size. Space remaining on the current device is shown underneath the search bar.

While searching through large folders, the search results should update every so often since it may take a while to search the full file tree.

When using the recycling bin, the program will look exactly the same, but warn the user that they are in the recycling bin "mode", so when they click files, instead of decrypting the file and opening it the file is instead moved back into the vault, recovering it to where it originally came from.

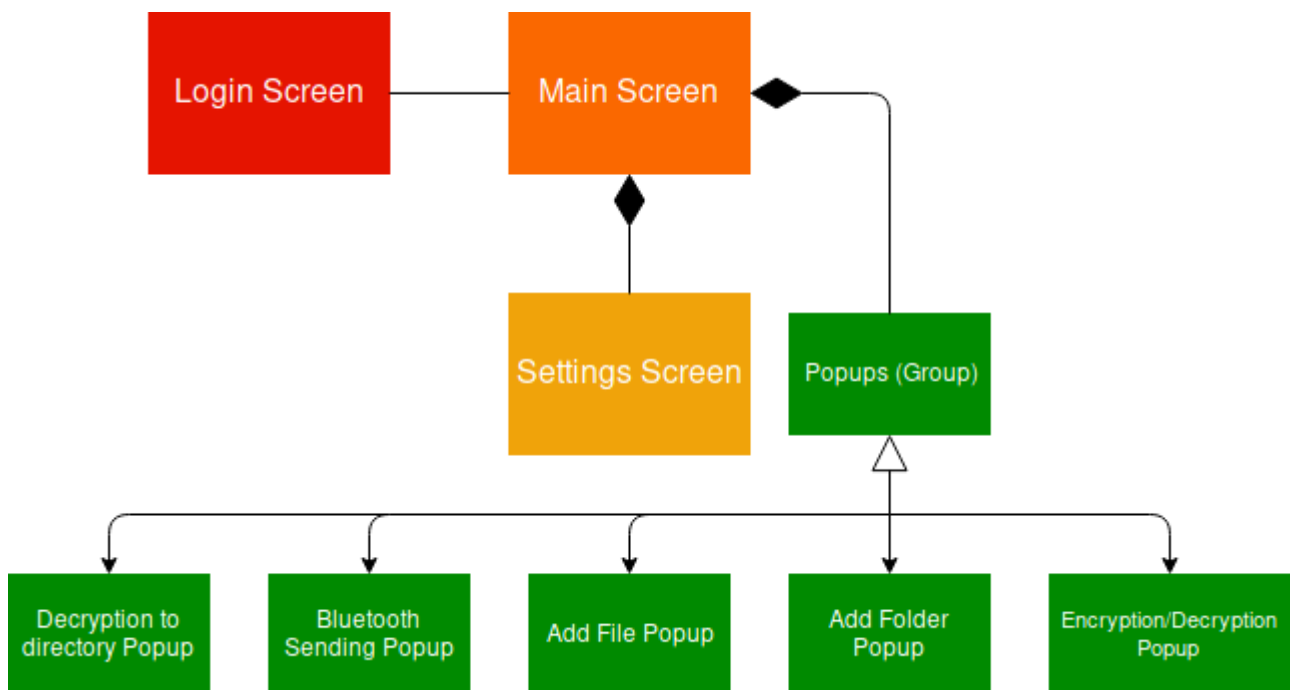
The login screen will have 2 modes:

1. Login without Bluetooth (can't use any Bluetooth functions while logged in).
2. Login with Bluetooth.

I will also make it so that you can easily switch between Bluetooth and non-Bluetooth login, whether that be a button on the login screen, or in the configuration file. Also, when in non-Bluetooth mode, the user will not need to have PyBluez installed, neither will they need Bluetooth on their PC.

When navigating the app, the navigation should be easy and simple so that the user does not get lost. I will have 2 main screens, a login screen and a main screen (to view files and open other functions once logged in), and within the main screen I will have a screen for settings, and a few other popups.

Here is a class diagram to show the relationship between screens and popups:



These are only the custom classes, so regular buttons and labels and such will be left out of this diagram.

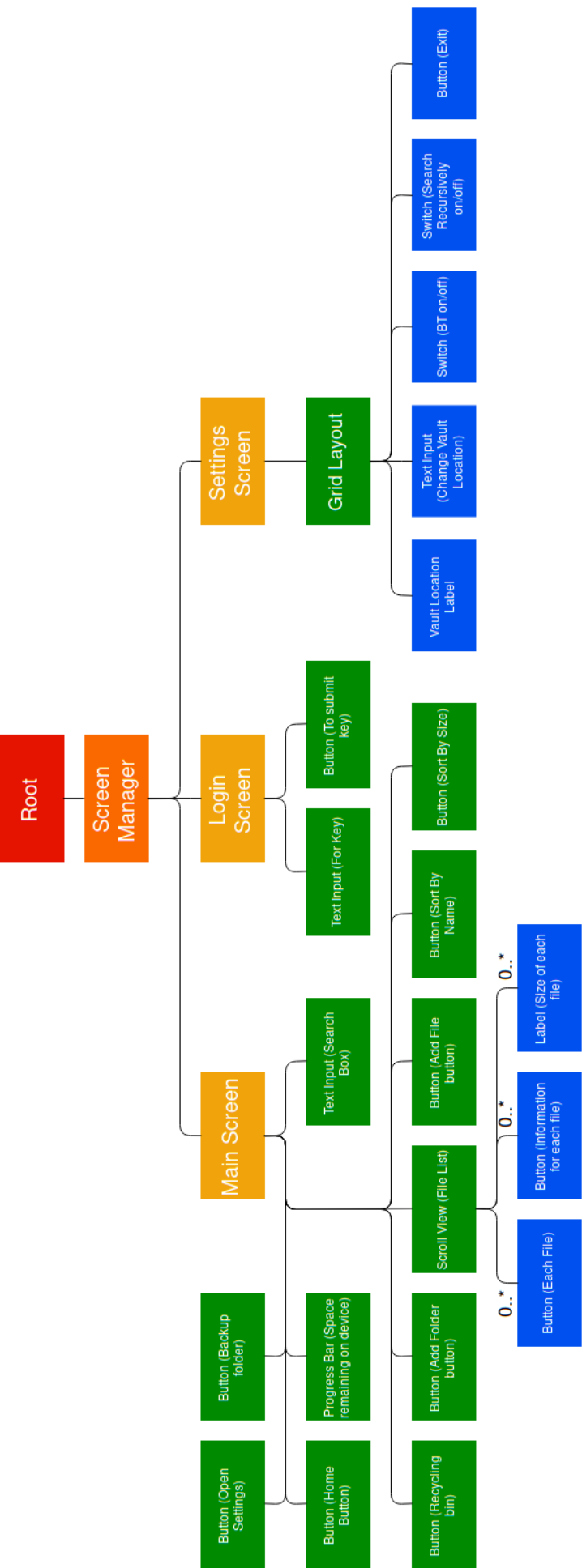
The Encryption/Decryption popup should be opened when the user encrypts/decrypts a file, and should display information including how fast the file is being enc/decrypted (in kb or mb per second), the percentage of the file that has been enc/decrypted so far, and how many items have been done out of the total files to be enc/decrypted. There should also be a progress bar at the bottom, showing the percentage visually.

The Bluetooth sending popup should show the exact same information, but for the current status of the file being sent over Bluetooth.

The add file and add folder popups should both be similar in design, however the add file popup will let the user encrypt a file or folder to the vault, while the add folder popup will allow the user to create a new folder within the vault.

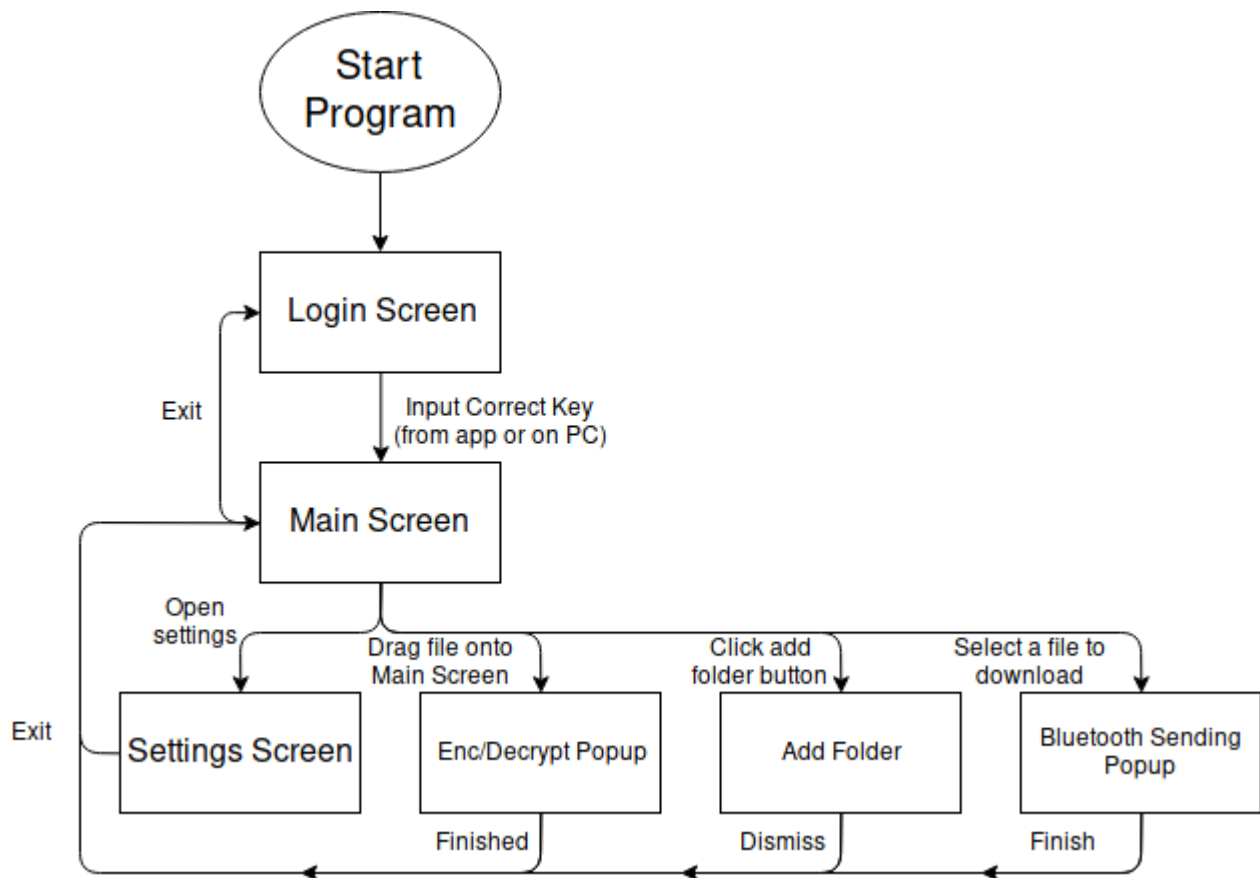
Popups are designed for one purpose only, and are usually used briefly before they are closed again. Screens will be used throughout the program, acting as the base of the GUI, where child widgets can be added to the screen, such as buttons, text inputs and views (such as scroll views). The screens will inherit from Kivy's Screen class, and the popups will inherit from Kivy's Popup class. The screens get managed by a ScreenManager, also a Kivy widget. The ScreenManager is then added to the app's root widget (the base widget of an app).

A hierarchy diagram for the entire GUI would look something like this (since Popups can be added and removed to any widget when needed, I will not include them in this diagram):



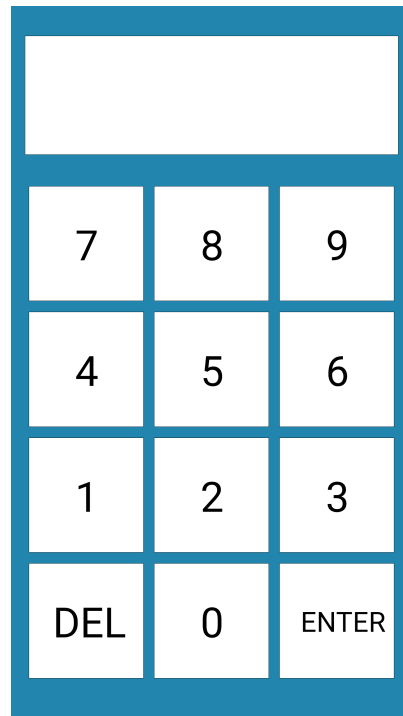
Each layer has it's own colour, since I couldn't think of a better way of making this clear without making the image extremely wide. "0..*" means 0 to many of this widget can exist at any time. This shows all of the widgets that will be on each screen at all times (unless obstructed by a popup) as default.

Here is a top-down view of how the GUI will flow while the user is using the program:



The App:

The app's UI design should be very simple, as I do not need to add much. All it needs to be is a number pad with a display, an enter button and a screen to have open while you are connected to the PC, and a file browser similar to the one on the PC app. Here is a prototype I made in Processing (A java based "software sketchbook):



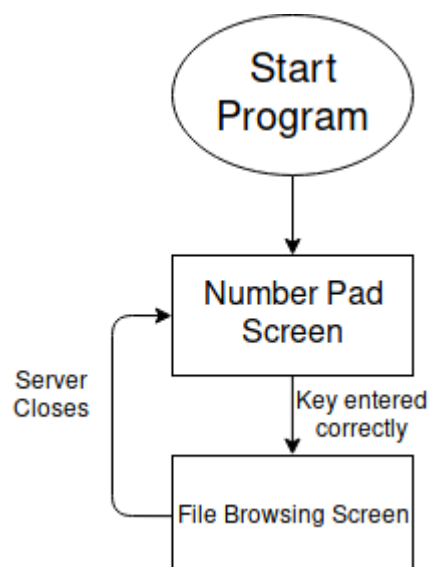
It is very minimal, as I decided to keep it as minimal as possible so that the user doesn't get confused, and to keep clutter at a minimum.

Once the vault is unlocked, the user should be given the option to browse files in the vault from their phone, and select files to download, or instead just minimise the app and continue using their phone. The vault should only close once the user has exited the app, rather than when they minimise the app.

The user should be able to browse the folders independently from the computer program (so both programs can be looking at different folders), browsing the files should be a seamless experience, and when searching for files, the searching work should be done on the computer so that precious phone battery is not wasted, and also because it is quicker in general to just send the search results to the mobile once they are generated.

The app should have a pin-code screen and a file browsing screen. The pin-code screen should only be used when the PC program is logged out.

Here is a top-down diagram of how the GUI will flow while the user is using the program:



The program as a whole:

My program will handle a fair amount of data, so here is a IPSO (Input, Processing, Storage, Output) chart to simplify it a little:

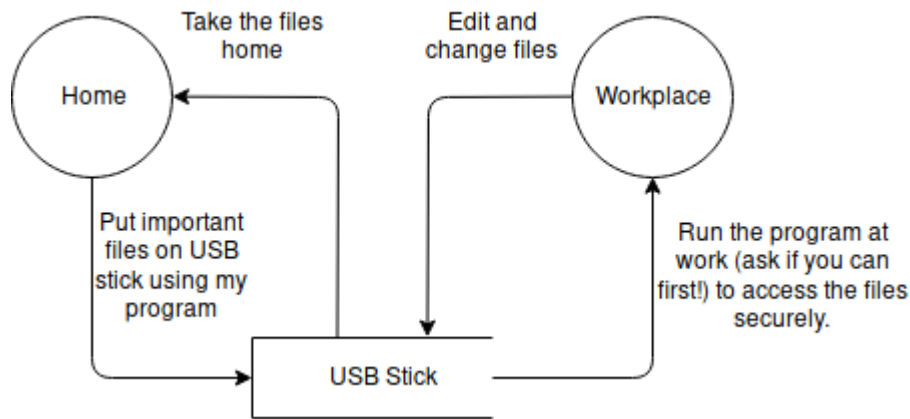
IPSO	Program Section	Item
Input	Login	Key (From user input). Vault directory path. Files in the Vault (for checking the key).
	File Browser (Main Screen)	Vault directory path. Recycling folder path (for when files are deleted). The key (for displaying encrypted file names).
	Search Bar (Main Screen)	Search item entered by user.
	Settings	Configuration file path. Current settings.
	Encryption/Decryption	The file path of the file that is desired to be enc/decrypted. The path to write the new data to. The key.
	Add Folder Popup	The name of the new folder to be created.
	Add File Popup	The path of the file to be encrypted to the vault.
	Recycling bin folder	Recycling folder path. Where each file came from originally.
Processing	Login	Decrypt the first block of the first file you find in the Vault, and check that it is equal to the key entered.
	File Browser (Main Screen)	Getting the sizes of each file. Sorting the files by name or size. Decrypting files when files are clicked. Encrypting files when files/folders are dragged into the window, or if a file/folder is added via the add file popup. Changing directory when a folder is clicked.
	Search Bar (Main Screen)	Search recursively for the file/folder in the Vault, or if recursive search is disabled in settings just search the current directory that the file browser is in.

	Settings	Change settings in the configuration file when changed in the program.
	Encryption/Decryption	Encrypt/Decrypt the file given using the key given.
	Add Folder Popup	Create the new folder in the current directory of the file browser.
	Recycling Bin Folder	Move files selected to original position.
Storage	File Browser (Main Screen)	Read the current files in the current directory that the file path is in.
	Settings	Read from the configuration file, and write to the configuration file when settings are changed.
	Add Folder Popup	Make new directory in the current directory the file browser is in.
	Encryption/Decryption	Read data from the file to be enc/decrypted, and write the enc/decrypted data to the location specified.
	Recycling Bin Folder	Read the file names of the files in the recycling bin.
Output	Login	Change the screen to Main Screen if the key is correct, otherwise create a Popup telling the user that the key is incorrect.
	File Browser (Main Screen)	Display the files in the Vault sorted how the user has specified, along with the size of each file, and a more information button.
	Search Bar	Return the list of closest matches to the search item given.
	Settings	Edit changes to file, and return values of each setting to the main program.
	Add File Popup	Pass the file path of the file to be added to the encryption function, with the path in the Vault where the new data should be written to.

There are many different use cases for my program. Some people may want to travel with the data, some people may just want to use it on one computer. In this section I will outline different ways I intend my program to be used.

Using a USB stick:

People who want to take the data with them to other places, a USB stick is a good idea. All the user has to do is download my program, put it on the USB stick and set the vault directory as a directory on the USB stick. No more setup should be needed. The program should be able to run on Windows, MacOS and Linux so using the USB on most devices should not be an issue. Here is a data flow diagram showing how the user may handle the data:

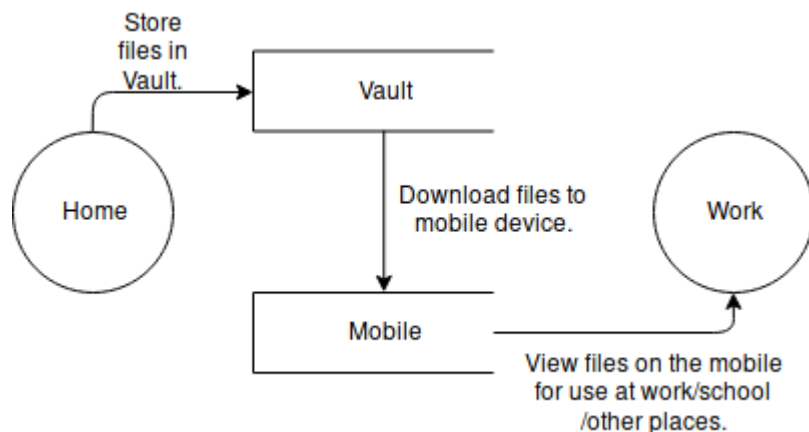


Storing the files at home:

People who may not need to travel as much with their data may just want to store their files at home, however if they want to take files to work/other places I will try to make it as easy as possible to do so.

The user should be able to decrypt the files they need to a folder (possibly on a USB stick), or download files from the Vault to their mobile device. This is worse than just using the whole app on the USB stick as mentioned in the last section, as the files will not be encrypted once they are in the folder or downloaded onto the mobile device. It is not recommended to do this if you want to edit the files while away from home, unless you can edit it on your device, however if not you may as well just put the files onto a USB stick.

A data flow diagram for this use case would look something like this:



If you wanted to edit the files at work without putting the entire program on a USB, you could instead decrypt the file and put it on a USB, take it to work, edit the file, go home and then encrypt it back into the vault, however the file is not encrypted.

Technical Solution

All intensive AES and BLAKE are written in Go, while everything else is written in Python, however the sorts are Cythonized (Python that has been compiled to a C shared object, using a mix of static variables and dynamic variables). Python communicates with Go using stdin and stdout pipes. SHA is written in Python because it is only needed a couple of times during the program, and only ever has to hash small data, so it does not need to be as fast as possible.

The File Structure of the code:

```
1  code
2  |  └─ mobile
3  |  |  └─ btShared.py
4  |  |  └─ bulldozer.spec
5  |  |  └─ fileSelectionScreen.py
6  |  |  └─ main.py
7  |  |  └─ mainScreen.py
8  |  |  └─ pad.kv
9  |  |  └─ padScreen.py
10 |  |  └─ SHA.py
11 |  |  └─ sortsCy.c
12 |  |  └─ sortsCy.cpython-37m-x86_64-linux-gnu.so
13 |  └─ python-go
14 |  |  └─ AES
15 |  |  └─ aesFName.py
16 |  |  └─ aes.go
17 |  |  └─ aes_test.go
18 |  |  └─ AESWin.exe
19 |  |  └─ BLAKE
20 |  |  └─ blake.go
21 |  |  └─ config.cfg
22 |  |  └─ configOperations.py
23 |  |  └─ fileClass.py
24 |  |  └─ kivyStuff
25 |  |  |  └─ kvFiles
26 |  |  |  |  └─ loginScBT.kv
27 |  |  |  |  └─ loginSc.kv
28 |  |  |  |  └─ mainScClasses.kv
29 |  |  |  |  └─ mainSc.kv
30 |  |  |  |  └─ settingsSc.kv
31 |  |  |  └─ loginClass.py
32 |  |  |  └─ mainBtns.py
33 |  |  |  └─ mainFileStatusPops.py
34 |  |  |  └─ mainScClass.py
35 |  |  |  └─ mainSmallPops.py
36 |  |  |  └─ ui.py
37 |  |  └─ settingsScreen.py
38 |  |  └─ SHA.py
39 |  |  └─ sortsCy.c
40 |  |  └─ sortsCy.cpython-37m-x86_64-linux-gnu.so
41 |  |  └─ sortsCythonSource
42 |  |  |  └─ build
43 |  |  |  |  └─ temp.linux-x86_64-3.7
44 |  |  |  |  |  └─ sortsCy.o
45 |  |  |  └─ setup.py
46 |  |  |  └─ sortsCy.pyx
47 |  |  └─ start.py
48
49 7 directories, 39 files
```

I have taken out all of the `__pycache__` folders that Python generates.

This is the output of `tree code` in my projects' `code` directory. You can find my project at <https://github.com/Lytchett-Minster/nea-12Colclough>.

The `code` directory, surprisingly, holds the code for my project. Inside is one folder for the mobile app (`mobile`), and one folder for the PC app (`python-go`). The PC app is started by running `start.py . start.py` imports `kivyStuff/ui.py` and runs it. This means that any Python files in `kivyStuff` can import any of the files that are in the same directory as `start.py` (`python-go`), and any Python files in `kivyStuff`. It also makes it easier to find the start script, as it isn't as buried.

The `assets` directory holds all the images needed for the GUI of the PC program (the images on the buttons). Here is a `tree` of the `assets` folder:

```
1  assets/
2  ├── exports
3  │   ├── addFile.png
4  │   ├── backUpFolder.png
5  │   ├── folder.png
6  │   ├── home.png
7  │   ├── info.png
8  │   ├── padlock.png
9  │   ├── recycling.png
10 │   ├── refresh-icon.png
11 │   ├── remove file.png
12 │   ├── search.png
13 │   └── settings.png
14  └── psd
15     ├── add file.psd
16     ├── back up folder.psd
17     ├── folder.psd
18     ├── info.svg
19     ├── padlock.psd
20     └── remove file.psd
21
22  2 directories, 17 files
```

Some images are taken from the internet, so they do not have `.psd` files (photoshop files).

Configuration of the program:

The program can be configured via the configuration file `config.cfg` located at `code/python-go/config.cfg`, or instead if the user is on Linux, then they can copy the configuration file into `~/.config/FileMate/config`, which is the standard area in Linux where configuration files are stored (and by the way, I called the program "File Mate" because it was the first thing that popped into my head).

The configuration file is edited by the settings menu in the main screen of the app, however if something goes horribly wrong, the user can edit it themselves easily.

The layout of the configuration file looks something like this:

```
1  vaultDir--<file path here>
2  searchRecursively--<True / False>
3  bluetooth--<True / False>
```

`vaultDir` is the path to the Vault that you would like to use to store all encrypted files and folders.

`searchRecursively` determines if the program should search for items recursively, as this may take a long time if you have a lot of files, and some people may just want to search within the current folder. `bluetooth` determines the default Login Screen to start when the program starts.

I have used `--` to separate the setting name from it's set value, as it does not appear at the start of file paths, and should not be needed much in any settings that could possibly be added in the future.

To change the configuration of the program from within the program, `configOperations.py` located at `code/python-go/configOperations.py` has a few functions that can get the configured settings, and write new ones.

Here is the content of `configOperations.py`:

```
1  from os import path as osPath
2  from os import listdir, makedirs
3  from sys import platform
4  from tempfile import gettempdir
5
6
7  def findConfigFile(startDir, fileSep):
8      config = None
9      if fileSep == "/":
10         try:
11             home = listdir(osPath.expanduser("~/config/FileMate/"))
12         except:
13             print("No config file in .config")
14         else:
15             if "config" in home:
16                 config = osPath.expanduser("~/config/FileMate/config")
17
18     if config == None:
19         try:
20             configFile = open(startDir+"config.cfg", "r")
21         except Exception as e:
22             raise FileNotFoundError("No config file found. Refer to the README if you need help.")
23         else:
24             configFile.close()
25             config = startDir+"config.cfg"
26
27     return config
28
29
30 def readConfigFile(configLocation=None, lineNumToRead=None):
31     if configLocation == None:
32         fSep = getFileSep()
33         configLocation = findConfigFile(getStartDir(fSep)[0], fSep)
34
35     configFile = open(configLocation, "r")
36     if lineNumToRead == None:
37         for line in configFile:
38             lineSplit = line.split("--")
39             lineSplit[1] = lineSplit[1].replace("\n", "")
40             if lineSplit[0] == "vaultDir":
41                 path = lineSplit[1]
42             elif lineSplit[0] == "searchRecursively":
43                 if lineSplit[1] == "True":
44                     recurse = True
45                 elif lineSplit[1] == "False":
46                     recurse = False
47             else:
48                 raise ValueError("Recursive search settings not set correctly in config file: Not
49 True or False.")
50             elif lineSplit[0] == "bluetooth":
51                 if lineSplit[1] == "True":
52                     bt = True
53                 elif lineSplit[1] == "False":
54                     bt = False
55             else:
56                 raise ValueError("Bluetooth not configured correctly in config file: Not True or
57 False.")
```

```

56         configFile.close()
57
58     return path, recurse, bt
59
60 else:
61     lineSplit = configFile.readlines()[lineNumToRead].split("--")
62     lineSplit[1] = lineSplit[1].replace("\n", "")
63     return lineSplit[1]
64
65 def getFileSep():
66     if platform.startswith("win32"): # Find out what operating system is running.
67         return "\\"
68     else: #windows bad
69         return "/"
70
71 def getStartDir(fileSep=None):
72     if fileSep == None:
73         fileSep = getFileSep()
74     startDir = osPath.dirname(osPath.realpath(__file__))+fileSep
75     tempDir = startDir.split(fileSep)
76     for i in range(2):
77         del tempDir[len(tempDir)-2]
78     return startDir, fileSep.join(tempDir)+fileSep+"assets"+fileSep+"exports"+fileSep
79
80 def editConfTerm(term, newContent, config): # Edits a given term in the config.cfg file.
81     with open(config, "r") as conf:
82         confContent = conf.readlines()
83
84     for i in range(len(confContent)):
85         a = confContent[i].split("--")
86         if term == a[0]:
87             a[1] = newContent+"\n"
88             confContent[i] = "--".join(a)
89
90     with open(config, "w") as confW:
91         confW.writelines(confContent)
92
93 def dirInputValid(inp, fileSep):
94     valid = bool((inp[0] == fileSep) and ("\n" not in inp)) #If it starts with the file separator
95     and doesn't contain any new lines, then it is valid for now.
96     inp = inp.split(fileSep)
97     focusIsSlash = False
98     for item in inp: #Checks for multiple file separators next to each other, as that would
99         be an invalid folder name.
100             if item == "":
101                 if focusIsSlash:
102                     valid = False
103                     focusIsSlash = True
104             else:
105                 focusIsSlash = False
106     return valid
107
108 def changeVaultLoc(inp, fileSep, config): #Sorts out the UI while the vault location is changed.
109     if inp != "":
110         if dirInputValid(inp, fileSep):
111             if osPath.exists(inp) and osPath.isdir(inp):
112                 editConfTerm("vaultDir", inp, config)
113             else:
114                 makedirs(inp)
115                 if inp[len(inp)-1] != fileSep:
116                     inp += fileSep
117                 editConfTerm("vaultDir", inp, config)
118
119 def runConfigOperations():
120     fileSep = getFileSep()
121     osTemp = gettempdir()+fileSep #From tempfile module

```

```

123     # Get config settings.
124     startDir, sharedAssets = getStartDir(fileSep)
125
126     configLoc = findConfigFile(startDir, fileSep)
127     path, recurse, bt = readConfigFile(configLoc)
128     return fileSep, osTemp, startDir, sharedAssets, path, recurse, bt, configLoc # 8 Outputs in total.
129

```

`findConfigFile` checks for the configuration file in `~/ .config/FileMate/`, and if it does not exist, checks for it in `code/python-go/config.cfg`. Once the configuration file has been found, it returns the path to the file.

`readConfigFile` reads the configuration file, and gets each configured option and returns their value. It can also return the value of a specific line in the config file.

`getFileSep` just gets the file separator of the current system. For Windows this is `\\`, but for MacOS and Linux this is `/`.

`getStartDir` gets the path of the current file (located in `code/python-go/`), and the path to the `assets` directory, which is used for the images on the buttons.

`editConfTerm` edits a term in the configuration file. If the term was "bluetooth" then it would find the line that starts with "bluetooth", and change the data after the `--` with the new data specified.

`dirInputValid` checks that a given input is a valid file path (e.g no "/" in a row). This is in here because it is used all over the program, and is used for changing the directory of the Vault.

`changeVaultLoc` changes the location of the Vault using `dirInputValid` to check the input, and `editConfTerm` to update the configuration file.

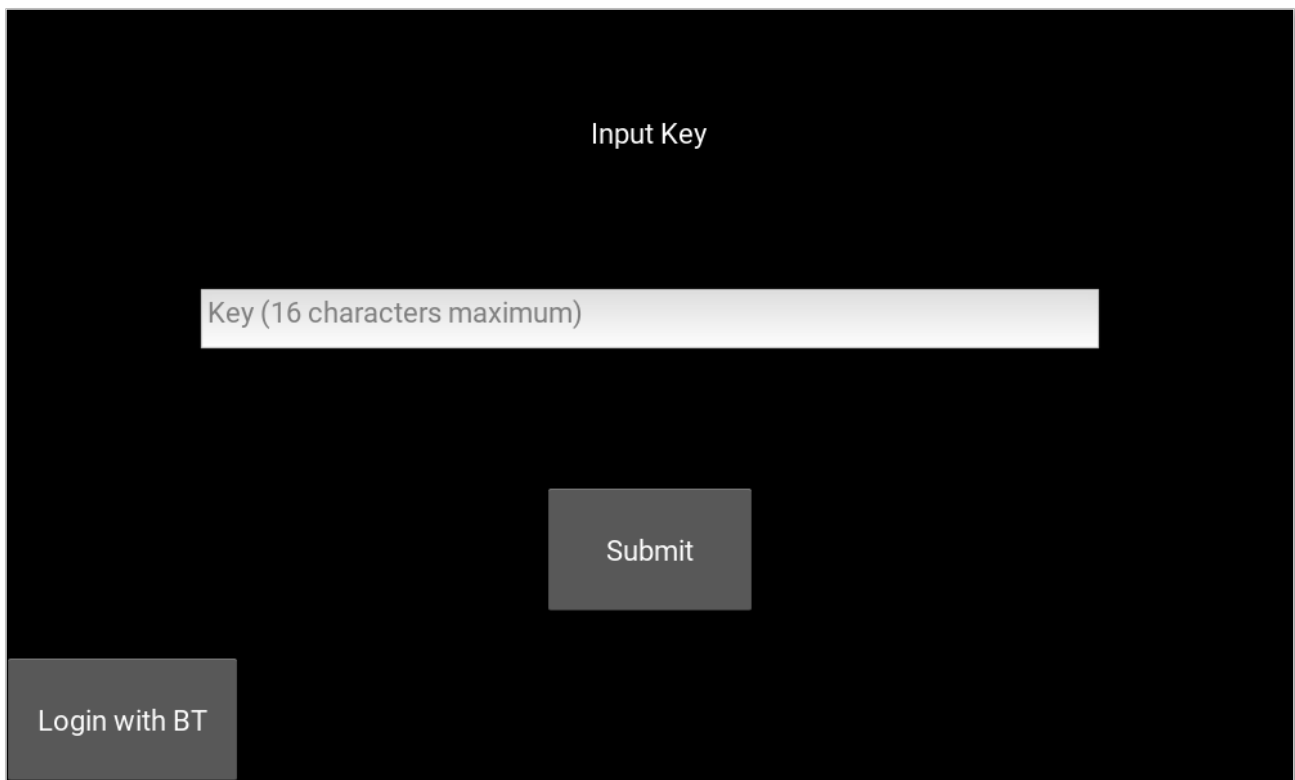
`runConfigOperations` runs all of the operations required for when the program is started, and returns the variables needed by the rest of the program. This is done in `ui.py`, which loads the configuration file, and starts the program.

The GUI:

I will go through the visuals first, and then move onto the code.

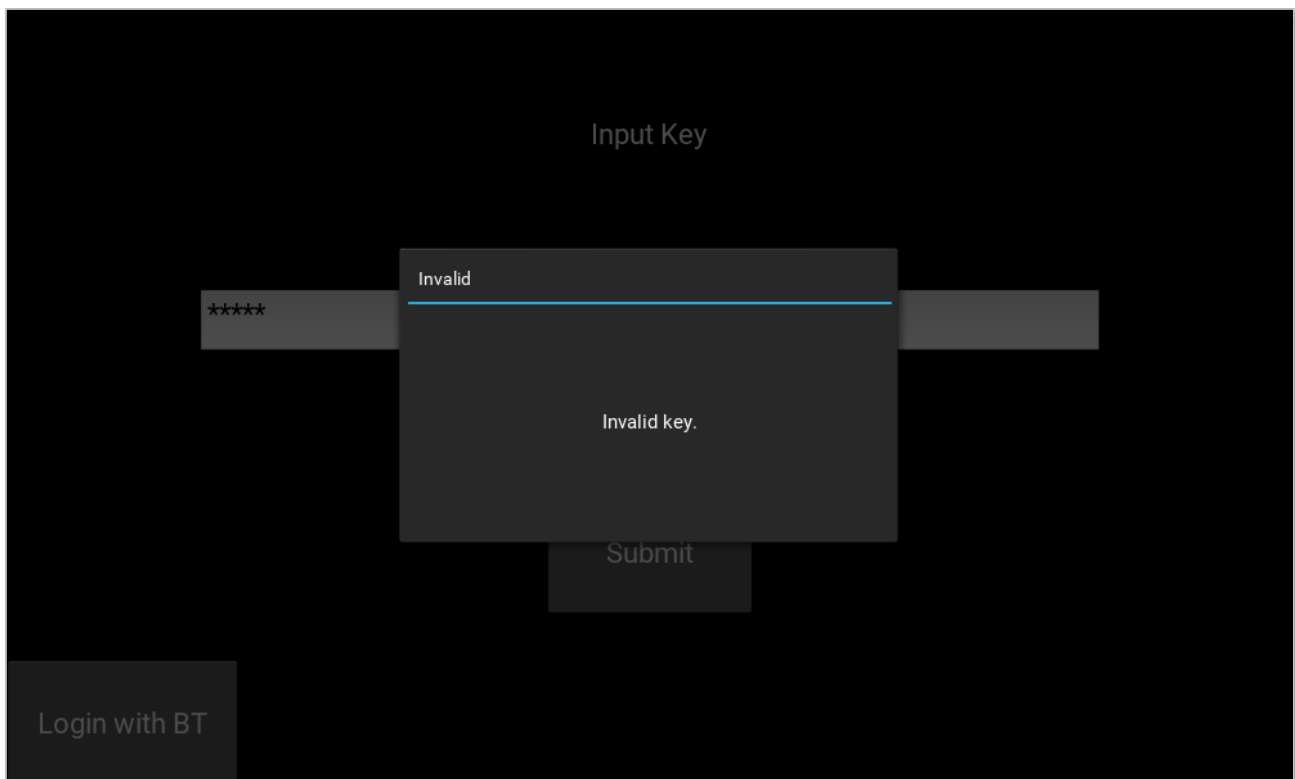
Login Screen

Here is an image of the Login Screen:

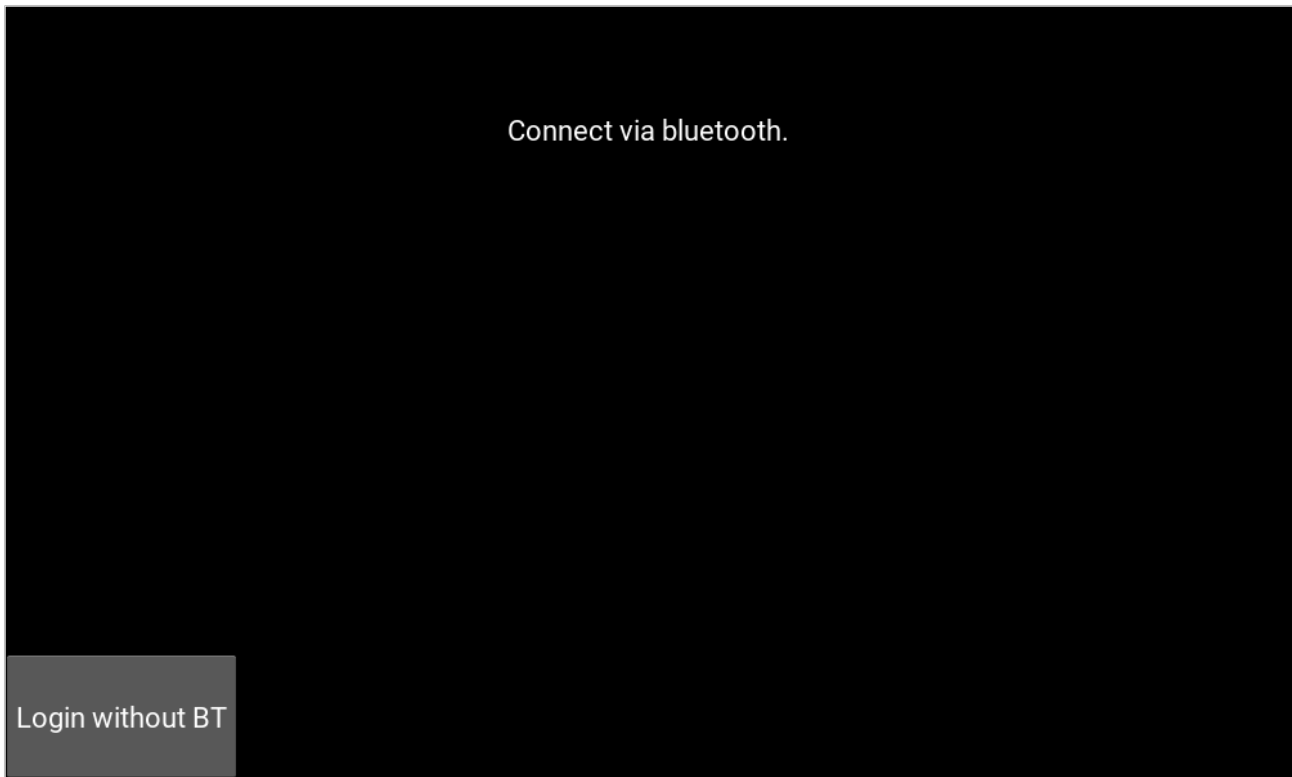


It consists of a key entry text input, a "Submit" button and a button to switch between logging in with Bluetooth and without Bluetooth.

When you enter an incorrect key, a popup tells the user the key is invalid:

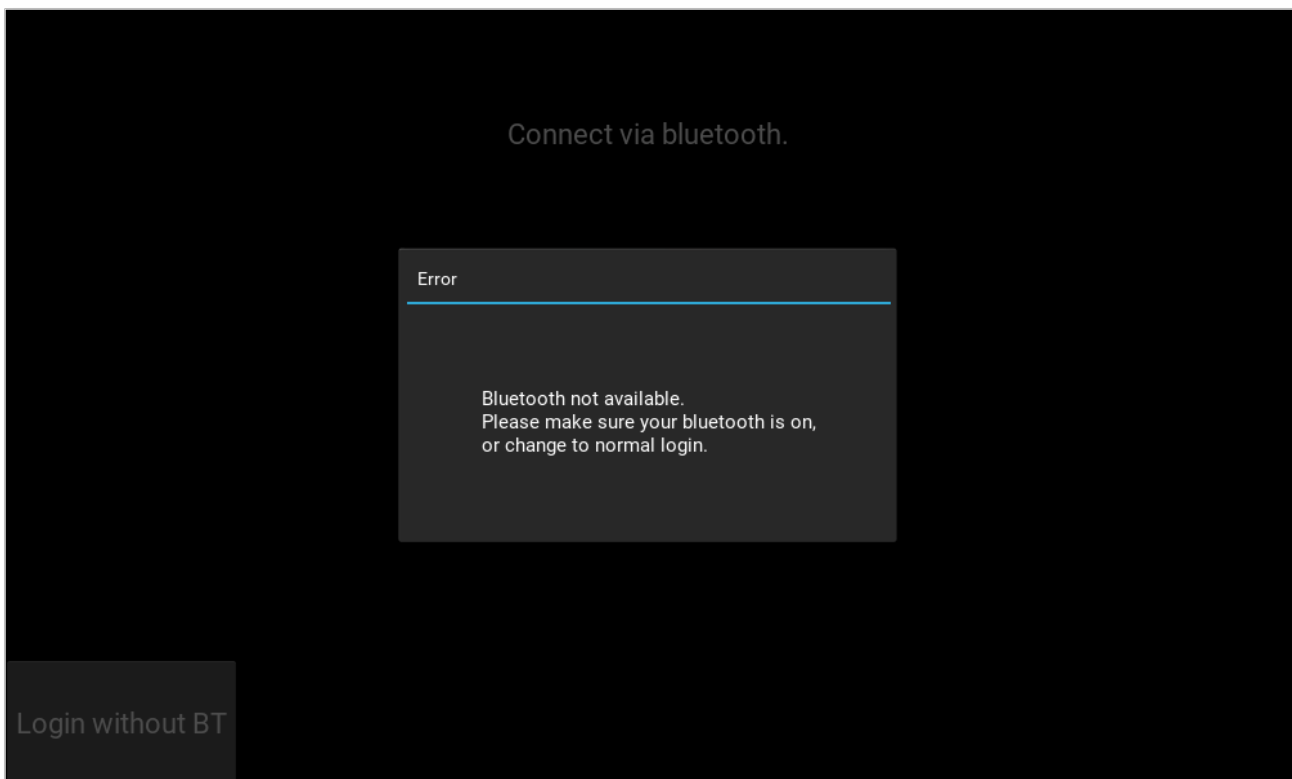


The Bluetooth login screen can be accessed by clicking the "Login with BT" button, changing the configuration file, or changing the settings once logged in. Here is an image of the Login Screen with Bluetooth:



I have tried to keep it as simple and as clutter-free as possible. When a user connects to the BT server, the address of the device connected appears in the middle of the screen, to let the user know that they have connected. The user then proceeds to enter the pin code on the app.

If Bluetooth is not available, or can't start, then a popup appears warning the user that they cannot use the Bluetooth login until Bluetooth becomes available, or they can instead login with regular login:

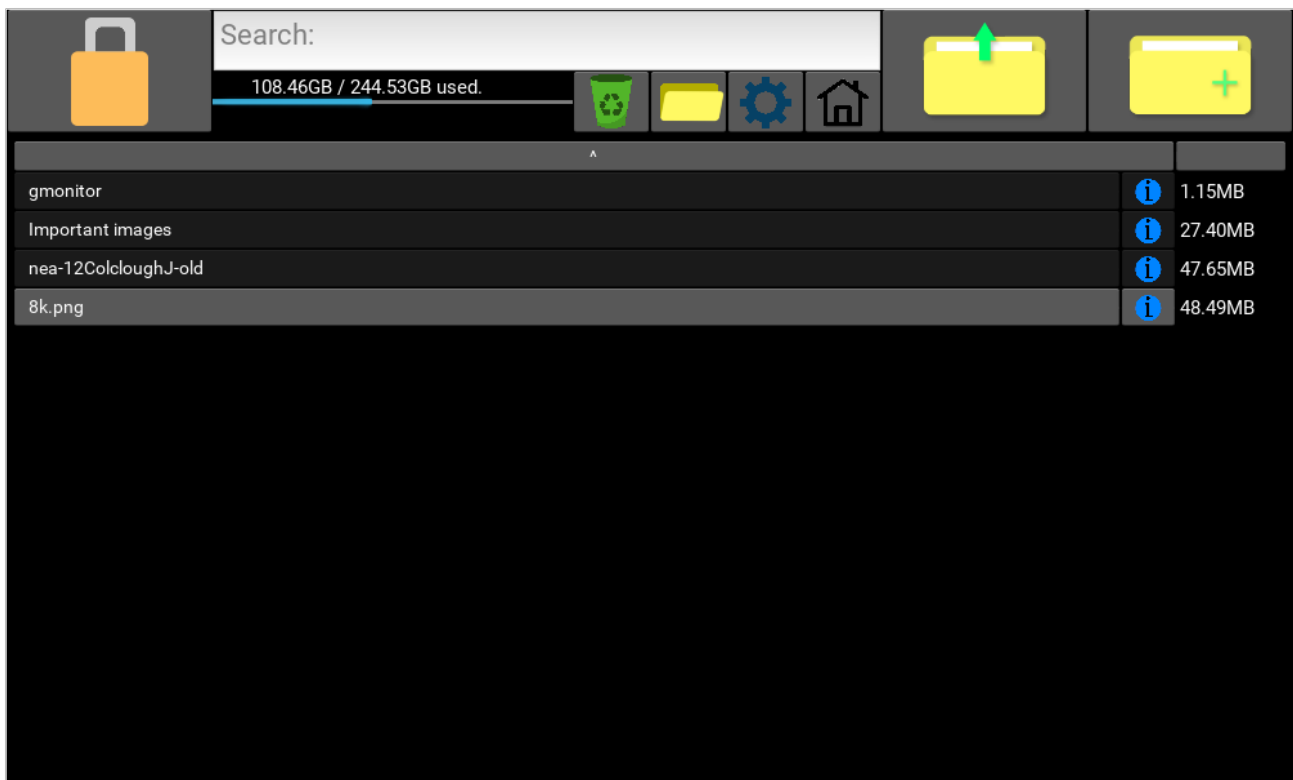


They can then click away from the popup to dismiss it, and do what they want from there.

Once the key has been entered correctly, the screen changes to the Main Screen.

Main Screen

Here is an image of the Main Screen:



I tried to keep it as similar as possible to the design in the **Design** section, however I had to add a button to add a new folder, as I realised that was a necessity. Also, instead of writing "Information" on the information button, I made an image to be put over the button instead, as it looks a bit better. All the buttons in the GUI are darker than in the design, but that is fine.

It is easy to distinguish between files and folders, and doesn't feel cluttered. The progress bar showing the amount of space used on the current storage device, in my opinion looks better than in the design. It is easy to sort by name (click the button above all the files) or to sort by size (click the button above all of the sizes).

When you click a folder, you change directory to that folder, and the contents of that folder are displayed on the screen. If it is a file, it is decrypted to `<systems_temp_folder>/FileMate/<fileName>`, where it is then opened with the system's default application and can be renamed and edited.

When you click to add a new folder, you get the exact same popup as in the **Information Tab** when you decrypt an item to a location.

The Information Tab

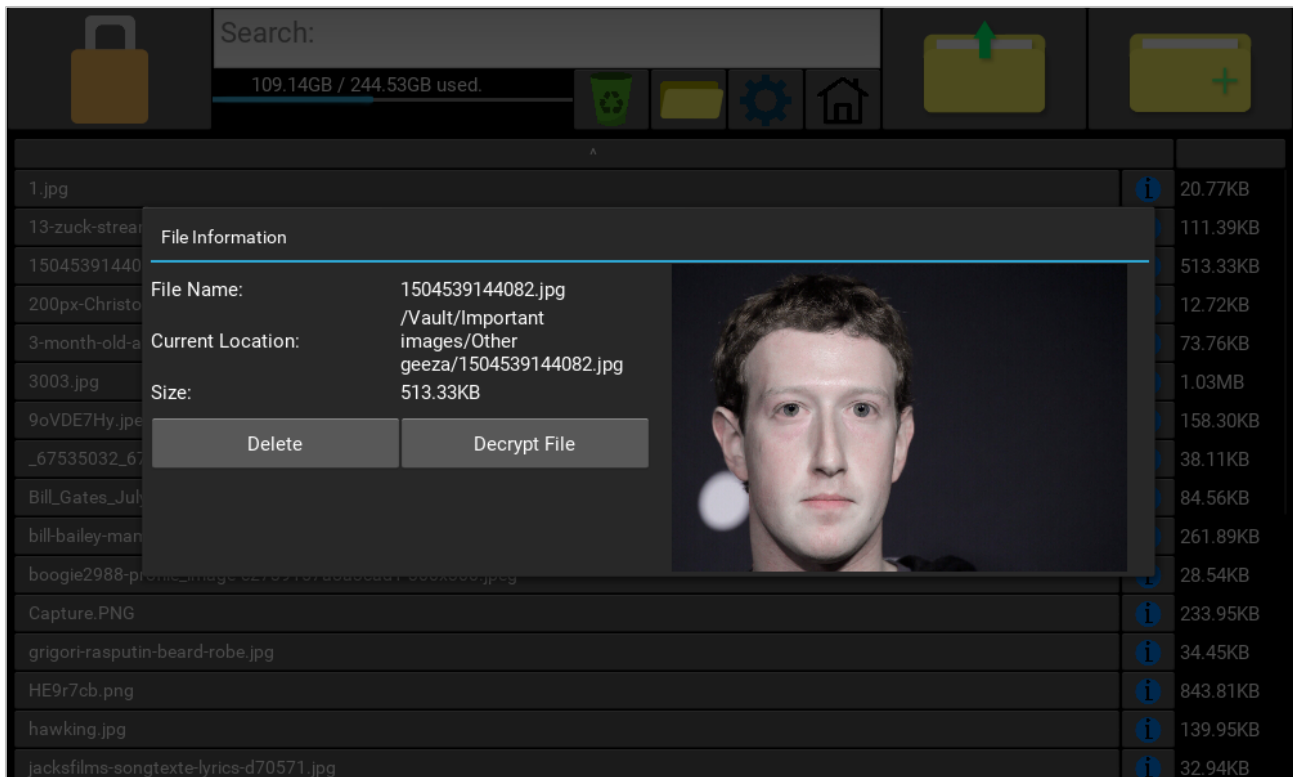
The information tab shows you information about the file:

- The location the file/folder is relative to the Vault.
- The size of the file/folder.
- A thumbnail of the image, if it is a file not a folder, and if the file is an image.

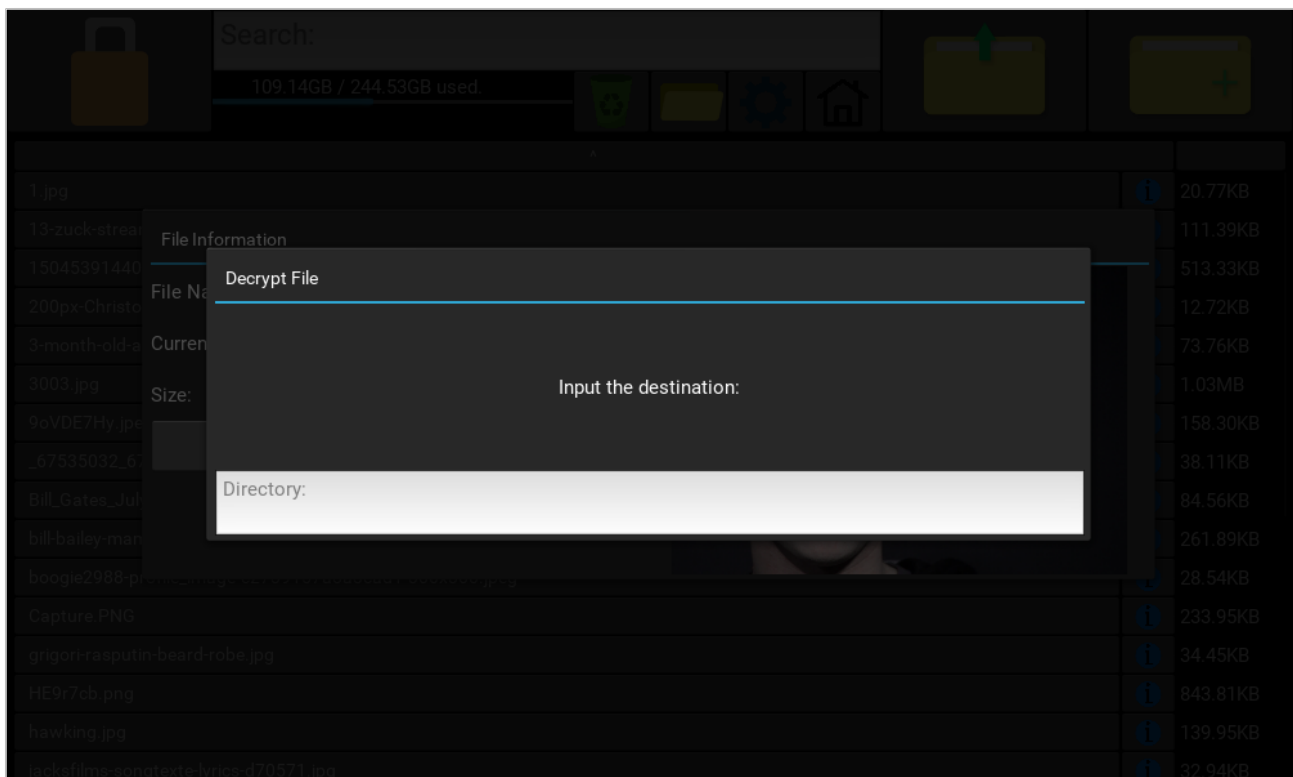
You also have a few options within the information tab to chose from:

- Delete the file/folder.
- Decrypt the file/folder to a specified location.

Here is a screenshot of the information popup:



When you click decrypt file, you are greeted with another popup asking where you would like the file to go:



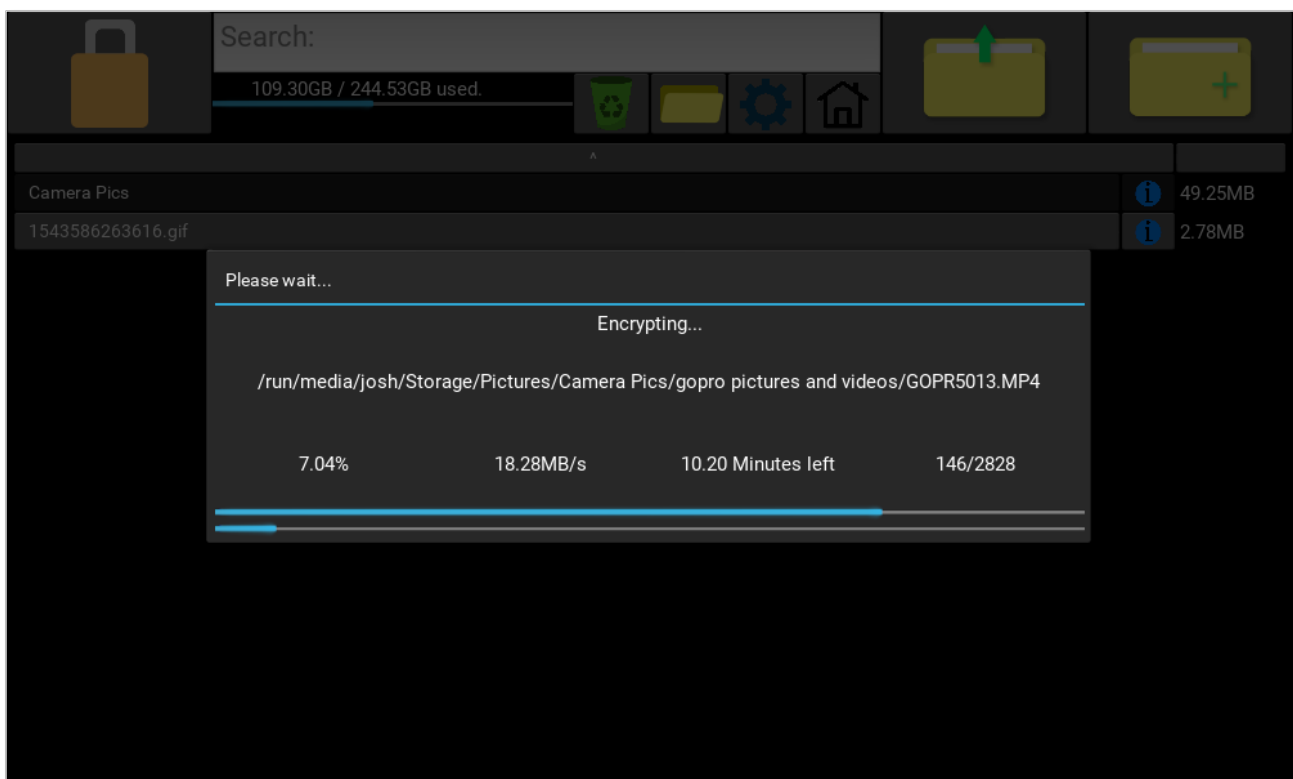
Once you input a correct directory name, it decrypts the file to that directory. If the path ends with the file separator (e.g. "/"), then it will be decrypted to that folder with it's original name. If the path does not, then it is saved to that exact location, with that new name. For example, if I wanted to decrypt a file called `Zuckerburg.png` , then I put in `/home/josh/zucc.png` , then it would decrypt the file and would be saved as `zucc.png` . If I instead put in `/home/josh/` , then it would be saved to `/home/josh/Zuckerburg.png` .

When you delete the file, the file moves to the recycling folder located in `Vault/.$recycling` (relative to the vault). To recover the file, you click the recycling bin button, and you get put in the recycling folder (with a popup warning the user they are in the recycling bin). Now when you click items in the recycling folder, instead of opening them and decrypting it, it moves the file back into the vault. You can still view information about the file like usuall, and search for items. To leave the recycling bin, you click the folder up button on the top right, or the home key below the search bar.

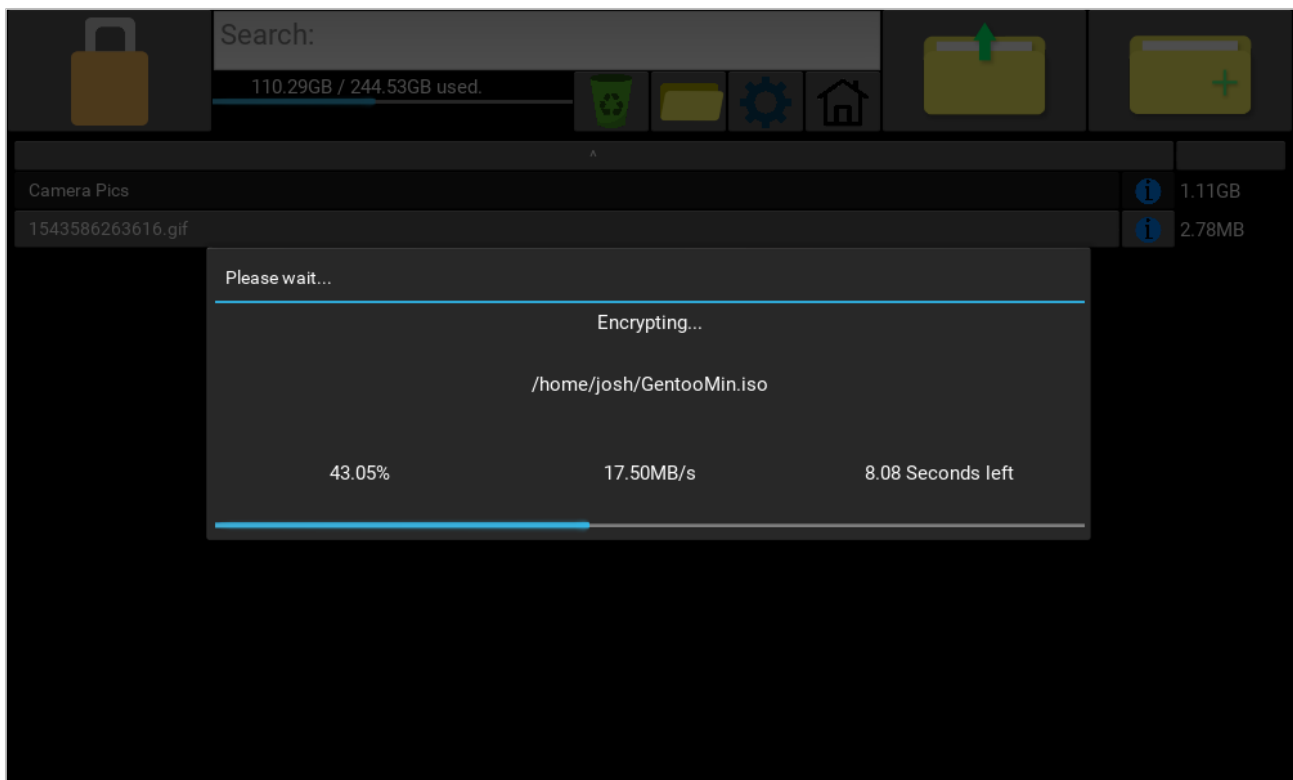
Encryption and Decryption Status

When decrypting a file, a popup opens showing you the percentage of the way through you are in the file and the current speed of decryption. When decrypting a folder, the same information is shown, however there are two progress bars. One for the current file being decrypted, and one for the total progress of decrypting the folder. Also, you are shown how many files in the folder have been encrypted out of the total. This is the exact same for encryption too by the way.

Here is an image of a folder being encrypted (12GB):



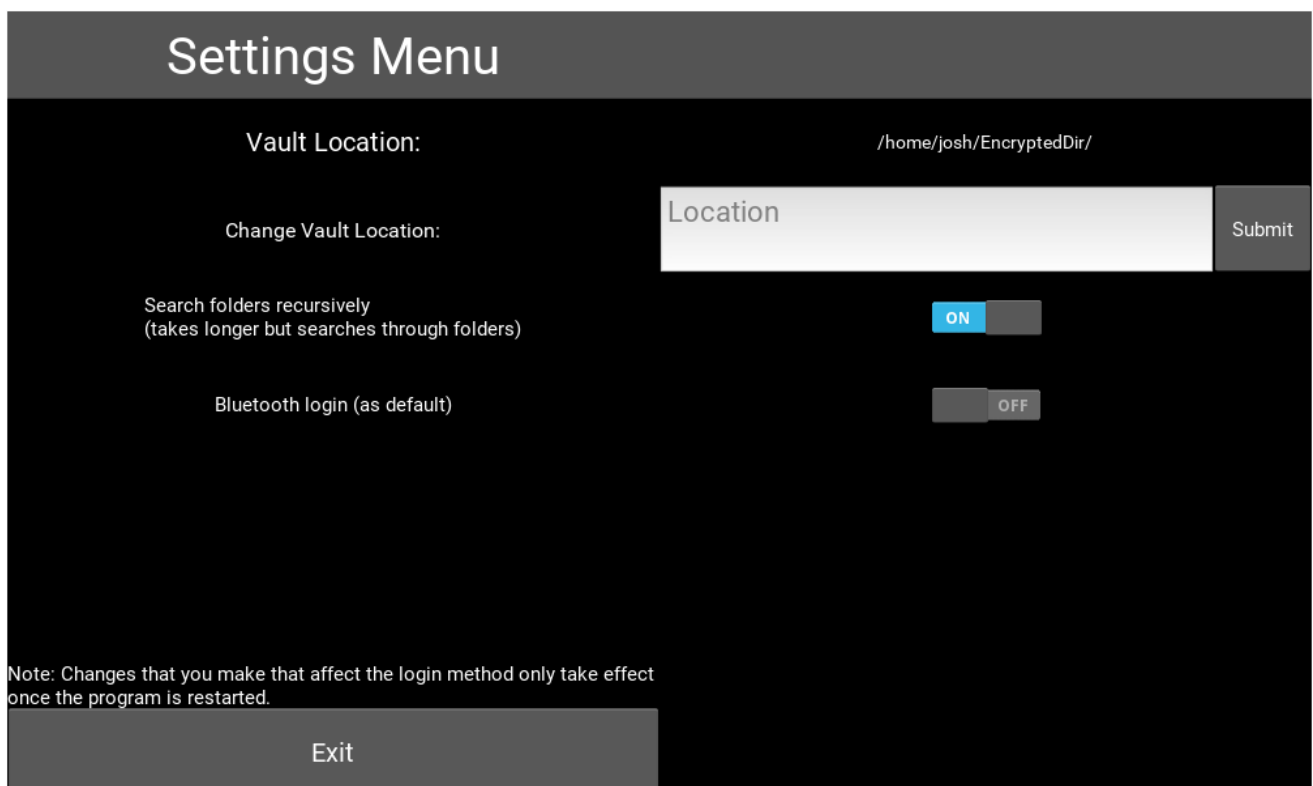
Here is an image of a file being encrypted (244 MB):



As I have said before, when a file is decrypted, it is decrypted to `<systems_temp_folder>/FileMate/<fileName>`, and is then opened using the system's default program for that file type. A checksum is calculated before the file is opened, and after the file is closed using BLAKE2b. The checksum is then compared, and if the checksum is different, then the file is encrypted back into the vault.

Settings Screen

Here is an image of the settings screen:



The current Vault location is displayed at the top of the screen, followed by an input to change the Vault location, followed by a pair of switches to change whether the search is recursive, and which login screen to use as default. When done, the user can click "Exit" to exit to the main screen again.

The Search

The search does a linear search through the unsorted directory, checking if the search term is in the file name, and at what position the item appears in the file name. This data is appended in tuples like this: `(pos, fileName)`. These tuples are then sorted by their `pos` value using a quick sort, and if the search term matched a file in the list exactly, letter for letter, then it will be added to the start of the list.

The list of results is then displayed:



AES:

Here is the code for AES (`code/python-go/aes.go`):

```
1 package main
2
3 import (
4     "fmt"          // For sending output on stout
5     "os"           // For opening files
6     "io"           // For reading files
7     "io/ioutil"    // ^
8     "strings"      // For converting string key to an array of bytes
9     "strconv"      // ^
10 )
11
12 // Global lookup tables.
```

```

13 var sBox = [256]byte {0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76,
14     0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0,
15     0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15,
16     0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75,
17     0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84,
18     0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF,
19     0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02,0x7F,0x50,0x3C,0x9F,0xA8,
20     0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2,
21     0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73,
22     0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB,
23     0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79,
24     0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08,
25     0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A,
26     0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E,
27     0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF,
28     0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16}
29
30 var invSBox = [256]byte
31     {0x52,0x09,0x6A,0xD5,0x30,0x36,0xA5,0x38,0xBF,0x40,0xA3,0x9E,0x81,0xF3,0xD7,0xFB,
32
33     0x7C,0xE3,0x39,0x82,0x9B,0x2F,0xFF,0x87,0x34,0x8E,0x43,0x44,0xC4,0xDE,0xE9,0xCB,
34
35     0x54,0x7B,0x94,0x32,0xA6,0xC2,0x23,0x3D,0xEE,0x4C,0x95,0x0B,0x42,0xFA,0xC3,0x4E,
36
37     0x08,0x2E,0xA1,0x66,0x28,0xD9,0x24,0xB2,0x76,0x5B,0xA2,0x49,0x6D,0x8B,0xD1,0x25,
38
39     0x72,0xF8,0xF6,0x64,0x86,0x68,0x98,0x16,0xD4,0xA4,0x5C,0xCC,0x5D,0x65,0xB6,0x92,
40
41     0x6C,0x70,0x48,0x50,0xFD,0xED,0xB9,0xDA,0x5E,0x15,0x46,0x57,0xA7,0x8D,0x9D,0x84,
42
43     0x90,0xD8,0xAB,0x00,0x8C,0xBC,0xD3,0x0A,0xF7,0xE4,0x58,0x05,0xB8,0xB3,0x45,0x06,
44
45     0xD0,0x2C,0x1E,0x8F,0xCA,0x3F,0x0F,0x02,0xC1,0xAF,0xBD,0x03,0x01,0x13,0x8A,0x6B,
46
47     0x3A,0x91,0x11,0x41,0x4F,0x67,0xDC,0xEA,0x97,0xF2,0xCF,0xCE,0xF0,0xB4,0xE6,0x73,
48
49     0x96,0xAC,0x74,0x22,0xE7,0xAD,0x35,0x85,0xE2,0xF9,0x37,0xE8,0x1C,0x75,0xDF,0x6E,
50
51     0x47,0xF1,0x1A,0x71,0x1D,0x29,0xC5,0x89,0x6F,0xB7,0x62,0x0E,0xAA,0x18,0xBE,0x1B,
52
53     0xFC,0x56,0x3E,0x4B,0xC6,0xD2,0x79,0x20,0x9A,0xDB,0xC0,0xFE,0x78,0xCD,0x5A,0xF4,
54
55     0x1F,0xDD,0xA8,0x33,0x88,0x07,0xC7,0x31,0xB1,0x12,0x10,0x59,0x27,0x80,0xEC,0x5F,
56
57     0x60,0x51,0x7F,0xA9,0x19,0xB5,0x4A,0x0D,0x2D,0xE5,0x7A,0x9F,0x93,0xC9,0x9C,0xEF,
58
59     0xA0,0xE0,0x3B,0x4D,0xAE,0x2A,0xF5,0xB0,0xC8,0xEB,0xBB,0x3C,0x83,0x53,0x99,0x61,
60
61     0x17,0x2B,0x04,0x7E,0xBA,0x77,0xD6,0x26,0xE1,0x69,0x14,0x63,0x55,0x21,0x0C,0x7D}
62
63
64 var rcon = [256]byte {0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,
    // https:// en.wikipedia.org/wiki/Rijndael\_key\_schedule
    0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,
    0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,
    0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,
    0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,
    0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,
    0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,
    0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,0x6a,0xd4,0xb3,
    0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,0x25,0x4a,0x94,
    0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,0x08,0x10,0x20,
    0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,0xc6,0x97,0x35,
    0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,0x61,0xc2,0x9f,
    0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d,0x01,0x02,0x04,
    0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a,0x2f,0x5e,0xbc,0x63,
    0xc6,0x97,0x35,0x6a,0xd4,0xb3,0x7d,0xfa,0xef,0xc5,0x91,0x39,0x72,0xe4,0xd3,0xbd,
    0x61,0xc2,0x9f,0x25,0x4a,0x94,0x33,0x66,0xcc,0x83,0x1d,0x3a,0x74,0xe8,0xcb,0x8d}
65
66 var mul2 = [256]byte {0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,

```

```

65      0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,
66      0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,
67      0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,
68      0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,
69      0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,
70      0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,
71      0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,
72      0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,
73      0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,
74      0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,
75      0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,
76      0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,
77      0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,
78      0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,
79      0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5}
80
81  var mul3 = [256]byte {0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,
82      0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,
83      0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,
84      0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,
85      0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,
86      0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,
87      0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,
88      0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,
89      0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,
90      0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,
91      0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xea,0xea,
92      0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,
93      0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,
94      0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,
95      0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,
96      0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a}
97
98  var mul9 = [256]byte {0x00,0x09,0x12,0x1b,0x24,0x2d,0x36,0x3f,0x48,0x41,0x5a,0x53,0x6c,0x65,0x7e,0x77,
99      0x90,0x99,0x82,0x8b,0xb4,0xbd,0xa6,0xaf,0xd8,0xd1,0xca,0xc3,0xfc,0xf5,0xee,0xe7,
100     0x3b,0x32,0x29,0x20,0x1f,0x16,0x0d,0x04,0x73,0x7a,0x61,0x68,0x57,0x5e,0x45,0x4c,
101     0xab,0xa2,0xb9,0xb0,0x8f,0x86,0x9d,0x94,0xe3,0xea,0xf1,0xf8,0xc7,0xce,0xd5,0xdc,
102     0x76,0x7f,0x64,0x6d,0x52,0x5b,0x40,0x49,0x3e,0x37,0x2c,0x25,0x1a,0x13,0x08,0x01,
103     0xe6,0xef,0xf4,0xfd,0xc2,0xcb,0xd0,0xd9,0xae,0xa7,0xbc,0xb5,0x8a,0x83,0x98,0x91,
104     0x4d,0x44,0x5f,0x56,0x69,0x60,0x7b,0x72,0x05,0x0c,0x17,0x1e,0x21,0x28,0x33,0x3a,
105     0xdd,0xd4,0xcf,0xc6,0xf9,0xf0,0xeb,0xe2,0x95,0x9c,0x87,0x8e,0xb1,0xb8,0xa3,0xaa,
106     0xec,0xe5,0xfe,0xf7,0xc8,0xc1,0xda,0xd3,0xa4,0xad,0xb6,0xbf,0x80,0x89,0x92,0x9b,
107     0x7c,0x75,0x6e,0x67,0x58,0x51,0x4a,0x43,0x34,0x3d,0x26,0x2f,0x10,0x19,0x02,0x0b,
108     0xd7,0xde,0xc5,0xcc,0xf3,0xfa,0xe1,0xe8,0x9f,0x96,0x8d,0x84,0xbb,0xb2,0xa9,0xa0,
109     0x47,0x4e,0x55,0x5c,0x63,0x6a,0x71,0x78,0x0f,0x06,0x1d,0x14,0x2b,0x22,0x39,0x30,
110     0x9a,0x93,0x88,0x81,0xbe,0xb7,0xac,0xa5,0xd2,0xdb,0xc0,0xc9,0xf6,0xff,0xe4,0xed,
111     0x0a,0x03,0x18,0x11,0x2e,0x27,0x3c,0x35,0x42,0x4b,0x50,0x59,0x66,0x6f,0x74,0x7d,
112     0xa1,0xa8,0xb3,0xba,0x85,0x8c,0x97,0x9e,0xe9,0xe0,0xfb,0xf2,0xcd,0xc4,0xdf,0xd6,
113     0x31,0x38,0x23,0x2a,0x15,0x1c,0x07,0x0e,0x79,0x70,0x6b,0x62,0x5d,0x54,0x4f,0x46}
114
115  var mul11 = [256]byte {0x00,0x0b,0x16,0x1d,0x2c,0x27,0x3a,0x31,0x58,0x53,0x4e,0x45,0x74,0x7f,0x62,0x69,
116     0xb0,0xbb,0xa6,0xad,0x9c,0x97,0x8a,0x81,0xe8,0xe3,0xfe,0xf5,0xc4,0xcf,0xd2,0xd9,
117     0x7b,0x70,0x6d,0x66,0x57,0x5c,0x41,0x4a,0x23,0x28,0x35,0x3e,0xf0,0x04,0x19,0x12,
118     0xcb,0xc0,0xdd,0xd6,0xe7,0xec,0xf1,0xfa,0x93,0x98,0x85,0x8e,0xbf,0xb4,0xa9,0xa2,
119     0xf6,0xfd,0xe0,0xeb,0xda,0xd1,0xcc,0xc7,0xae,0xa5,0xb8,0xb3,0x82,0x89,0x94,0x9f,
120     0x46,0x4d,0x50,0x5b,0x6a,0x61,0x7c,0x77,0x1e,0x15,0x08,0x03,0x32,0x39,0x24,0x2f,
121     0x8d,0x86,0x9b,0x90,0xa1,0xaa,0xb7,0xbc,0xd5,0xde,0xc3,0xc8,0xf9,0xf2,0xef,0xe4,
122     0x3d,0x36,0x2b,0x20,0x11,0x1a,0x07,0x0c,0x65,0x6e,0x73,0x78,0x49,0x42,0x5f,0x54,
123     0xf7,0xfc,0xe1,0xea,0xdb,0xd0,0xcd,0xc6,0xaf,0xa4,0xb9,0xb2,0x83,0x88,0x95,0x9e,
124     0x47,0x4c,0x51,0x5a,0x6b,0x60,0x7d,0x76,0x1f,0x14,0x09,0x02,0x33,0x38,0x25,0x2e,
125     0x8c,0x87,0x9a,0x91,0xa0,0xab,0xb6,0xbd,0xd4,0xdf,0xc2,0xc9,0xf8,0xf3,0xee,0xe5,
126     0x3c,0x37,0x2a,0x21,0x10,0x1b,0x06,0x0d,0x64,0x6f,0x72,0x79,0x48,0x43,0x5e,0x55,
127     0x01,0x0a,0x17,0x1c,0x2d,0x26,0x3b,0x30,0x59,0x52,0x4f,0x44,0x75,0x7e,0x63,0x68,
128     0xb1,0xba,0xa7,0xac,0x9d,0x96,0x8b,0x80,0xe9,0xe2,0xff,0xf4,0xc5,0xce,0xd3,0xd8,
129     0x7a,0x71,0x6c,0x67,0x56,0x5d,0x40,0x4b,0x22,0x29,0x34,0x3f,0x0e,0x05,0x18,0x13,
130     0xca,0xc1,0xdc,0xd7,0xe6,0xed,0xf0,0xfb,0x92,0x99,0x84,0x8f,0xbe,0xb5,0xa8,0xa3}
131
132  var mul13 = [256]byte {0x00,0x0d,0x1a,0x17,0x34,0x39,0x2e,0x23,0x68,0x65,0x72,0x7f,0x5c,0x51,0x46,0x4b,
133     0xd0,0xdd,0xca,0xc7,0xe4,0xe9,0xfe,0xf3,0xb8,0xb5,0xa2,0xaf,0x8c,0x81,0x96,0x9b,

```

```

134         0xbb,0xb6,0xa1,0xac,0x8f,0x82,0x95,0x98,0xd3,0xde,0xc9,0xc4,0xe7,0xea,0xfd,0xf0,
135         0x6b,0x66,0x71,0x7c,0x5f,0x52,0x45,0x48,0x03,0x0e,0x19,0x14,0x37,0x3a,0x2d,0x20,
136         0x6d,0x60,0x77,0x7a,0x59,0x54,0x43,0x4e,0x05,0x08,0x1f,0x12,0x31,0x3c,0x2b,0x26,
137         0xbd,0xb0,0xa7,0xaa,0x89,0x84,0x93,0x9e,0xd5,0xd8,0xcf,0xc2,0xe1,0xec,0xfb,0xf6,
138         0xd6,0xdb,0xcc,0xc1,0xe2,0xef,0xf8,0xf5,0xbe,0xb3,0xa4,0xa9,0x8a,0x87,0x90,0x9d,
139         0x06,0x0b,0x1c,0x11,0x32,0x3f,0x28,0x25,0x6e,0x63,0x74,0x79,0x5a,0x57,0x40,0x4d,
140         0xda,0xd7,0xc0,0xcd,0xee,0xe3,0xf4,0xf9,0xb2,0xbf,0xa8,0xa5,0x86,0x8b,0x9c,0x91,
141         0x0a,0x07,0x10,0x1d,0x3e,0x33,0x24,0x29,0x62,0x6f,0x78,0x75,0x56,0x5b,0x4c,0x41,
142         0x61,0x6c,0x7b,0x76,0x55,0x58,0x4f,0x42,0x09,0x04,0x13,0x1e,0x3d,0x30,0x27,0x2a,
143         0xb1,0xbc,0xab,0xa6,0x85,0x88,0x9f,0x92,0xd9,0xd4,0xc3,0xce,0xed,0xe0,0xf7,0xfa,
144         0xb7,0xba,0xad,0xa0,0x83,0x8e,0x99,0x94,0xdf,0xd2,0xc5,0xc8,0xeb,0xe6,0xf1,0xfc,
145         0x67,0x6a,0x7d,0x70,0x53,0x5e,0x49,0x44,0x0f,0x02,0x15,0x18,0x3b,0x36,0x21,0x2c,
146         0x0c,0x01,0x16,0x1b,0x38,0x35,0x22,0x2f,0x64,0x69,0x7e,0x73,0x50,0x5d,0x4a,0x47,
147         0xdc,0xd1,0xc6,0xcb,0xe8,0xe5,0xf2,0xff,0xb4,0xb9,0xae,0xa3,0x80,0x8d,0x9a,0x97}
148
149 var mul14 = [256]byte {0x00,0x0e,0x1c,0x12,0x38,0x36,0x24,0x2a,0x70,0x7e,0x6c,0x62,0x48,0x46,0x54,0x5a,
150         0xe0,0xee,0xfc,0xf2,0xd8,0xd6,0xc4,0xca,0x90,0x9e,0x8c,0x82,0xa8,0xa6,0xb4,0xba,
151         0xdb,0xd5,0xc7,0xc9,0xe3,0xed,0xff,0xf1,0xab,0xa5,0xb7,0xb9,0x93,0x9d,0x8f,0x81,
152         0x3b,0x35,0x27,0x29,0x03,0x0d,0x1f,0x11,0x4b,0x45,0x57,0x59,0x73,0x7d,0x6f,0x61,
153         0xad,0xa3,0xb1,0xbf,0x95,0x9b,0x89,0x87,0xdd,0xd3,0xc1,0xcf,0xe5,0xeb,0xf9,0xf7,
154         0x4d,0x43,0x51,0x5f,0x75,0x7b,0x69,0x67,0x3d,0x33,0x21,0x2f,0x05,0x0b,0x19,0x17,
155         0x76,0x78,0x6a,0x64,0x4e,0x40,0x52,0x5c,0x06,0x08,0x1a,0x14,0x3e,0x30,0x22,0x2c,
156         0x96,0x98,0x8a,0x84,0xae,0xa0,0xb2,0xbc,0xe6,0xe8,0xfa,0xf4,0xde,0xd0,0xc2,0xcc,
157         0x41,0x4f,0x5d,0x53,0x79,0x77,0x65,0x6b,0x31,0x3f,0x2d,0x23,0x09,0x07,0x15,0x1b,
158         0xa1,0xaf,0xbd,0xb3,0x99,0x97,0x85,0x8b,0xd1,0xdf,0xcd,0xc3,0xe9,0xe7,0xf5,0xfb,
159         0x9a,0x94,0x86,0x88,0xa2,0xac,0xbe,0xb0,0xea,0xe4,0xf6,0xf8,0xd2,0xdc,0xce,0xc0,
160         0x7a,0x74,0x66,0x68,0x42,0x4c,0x5e,0x50,0x0a,0x04,0x16,0x18,0x32,0x3c,0x2e,0x20,
161         0xec,0xe2,0xf0,0xfe,0xd4,0xda,0xc8,0xc6,0x9c,0x92,0x80,0x8e,0xa4,0xaa,0xb8,0xb6,
162         0x0c,0x02,0x10,0x1e,0x34,0x3a,0x28,0x26,0x7c,0x72,0x60,0x6e,0x44,0x4a,0x58,0x56,
163         0x37,0x39,0x2b,0x25,0x0f,0x01,0x13,0x1d,0x47,0x49,0x5b,0x55,0x7f,0x71,0x63,0x6d,
164         0xd7,0xd9,0xcb,0xc5,0xef,0xe1,0xf3,0xfd,0xa7,0xa9,0xbb,0xb5,0x9f,0x91,0x83,0x8d}
165
166
167 func keyExpansionCore(inp [4]byte, i int) ([4]byte) {
168     // Shift the inp left by moving the first byte to the end (rotate).
169     inp[0], inp[1], inp[2], inp[3] = inp[1], inp[2], inp[3], inp[0]
170
171     // S-Box the bytes
172     inp[0], inp[1], inp[2], inp[3] = sBox[inp[0]], sBox[inp[1]], sBox[inp[2]], sBox[inp[3]]
173
174     // rcon, the round constant
175     inp[0] ^= rcon[i]
176
177     return inp
178 }
179
180 func expandKey(inputKey []byte) ([176]byte) {
181     var expandedKeys [176]byte
182     // first 16 bytes of the expandedKeys should be the same 16 as the original key
183     for i := 0; i < 16; i++ {
184         expandedKeys[i] = inputKey[i]
185     }
186     var bytesGenerated int = 16 // needs to get to 176 to fill expandedKeys with 11 keys, one for every
    round.
187     var rconIteration int = 1
188     var temp [4]byte
189
190     for bytesGenerated < 176{
191         // Read 4 bytes for use in keyExpansionCore
192         copy(temp[:], expandedKeys[bytesGenerated-4:bytesGenerated])
193
194         if bytesGenerated % 16 == 0 { // Keys are length 16 bytes so every 16 bytes generated, expand.
195             temp = keyExpansionCore(temp, rconIteration)
196             rconIteration += 1
197         }
198
199         for y := 0; y < 4; y++ {
200             expandedKeys[bytesGenerated] = expandedKeys[bytesGenerated - 16] ^ temp[y] // XOR first 4 bytes
    of previous key with the temporary list.

```



```

201     bytesGenerated += 1
202 }
203 }
204
205 return expandedKeys
206 }
207
208 func addRoundKey(state []byte, roundKey []byte) ([]byte) { // Add round key is also it's own
inverse
209     for i := 0; i < 16; i++ {
210         state[i] ^= roundKey[i] // XOR each byte of the key with each byte of the state.
211     }
212     return state
213 }
214
215 func subBytes(state []byte) []byte {
216     for i := 0; i < 16; i++ {
217         state[i] = sBox[state[i]]
218     }
219     return state
220 }
221
222 func invSubBytes(state []byte) []byte {
223     for i := 0; i < 16; i++ {
224         state[i] = invSBox[state[i]]
225     }
226     return state
227 }
228
229 func shiftRows(state []byte) ([]byte) {
230     return []byte{state[0], state[5], state[10], state[15],
231                 state[4], state[9], state[14], state[3],
232                 state[8], state[13], state[2], state[7],
233                 state[12], state[1], state[6], state[11]}
234     // Shifts it like this:
235     //
236     // 0 4 8 12      0 4 8 12   Shifted left by 0
237     // 1 5 9 13 ----> 5 9 13 1   Shifted left by 1
238     // 2 6 10 14 ----> 10 14 2 6   Shifted left by 2
239     // 3 7 11 15      15 3 7 11   Shifted left by 3
240 }
241
242 func invShiftRows(state []byte) ([]byte) {
243     return []byte{state[0], state[13], state[10], state[7],
244                 state[4], state[1], state[14], state[11],
245                 state[8], state[5], state[2], state[15],
246                 state[12], state[9], state[6], state[3]}
247
248     // 0 4 8 12      0 4 8 12   Shifted right by 0
249     // 5 9 13 1 ----> 1 5 9 13   Shifted right by 1
250     // 10 14 2 6 ----> 2 6 10 14   Shifted right by 2
251     // 15 3 7 11      3 7 11 15   Shifted right by 3
252 }
253
254 func mixColumns(state []byte) ([]byte) {
255     return []byte{mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3],
256                 state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3],
257                 state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]],
258                 mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]],
259
260                 mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7],
261                 state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7],
262                 state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]],
263                 mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]],
264
265                 mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11],
266                 state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11],
267                 state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]],
268                 mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]],

```

```

269         mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15],
270         state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15],
271         state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]],
272         mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]}
273     }
274 }
275
276 func invMixColumns(state []byte) ([]byte) {
277     return []byte{mul14[state[0]] ^ mul11[state[1]] ^ mul13[state[2]] ^ mul9[state[3]],
278                 mul9[state[0]] ^ mul14[state[1]] ^ mul11[state[2]] ^ mul13[state[3]],
279                 mul13[state[0]] ^ mul9[state[1]] ^ mul14[state[2]] ^ mul11[state[3]],
280                 mul11[state[0]] ^ mul13[state[1]] ^ mul9[state[2]] ^ mul14[state[3]],
281
282                 mul14[state[4]] ^ mul11[state[5]] ^ mul13[state[6]] ^ mul9[state[7]],
283                 mul9[state[4]] ^ mul14[state[5]] ^ mul11[state[6]] ^ mul13[state[7]],
284                 mul13[state[4]] ^ mul9[state[5]] ^ mul14[state[6]] ^ mul11[state[7]],
285                 mul11[state[4]] ^ mul13[state[5]] ^ mul9[state[6]] ^ mul14[state[7]],
286
287                 mul14[state[8]] ^ mul11[state[9]] ^ mul13[state[10]] ^ mul9[state[11]],
288                 mul9[state[8]] ^ mul14[state[9]] ^ mul11[state[10]] ^ mul13[state[11]],
289                 mul13[state[8]] ^ mul9[state[9]] ^ mul14[state[10]] ^ mul11[state[11]],
290                 mul11[state[8]] ^ mul13[state[9]] ^ mul9[state[10]] ^ mul14[state[11]],
291
292                 mul14[state[12]] ^ mul11[state[13]] ^ mul13[state[14]] ^ mul9[state[15]],
293                 mul9[state[12]] ^ mul14[state[13]] ^ mul11[state[14]] ^ mul13[state[15]],
294                 mul13[state[12]] ^ mul9[state[13]] ^ mul14[state[14]] ^ mul11[state[15]],
295                 mul11[state[12]] ^ mul13[state[13]] ^ mul9[state[14]] ^ mul14[state[15]]}
296 }
297
298
299 func encrypt(state []byte, expandedKeys [176]byte, regularRounds int) ([]byte) {
300     state = addRoundKey(state, expandedKeys[:16])
301
302     for i := 0; i < regularRounds; i++ {
303         state = subBytes(state)
304         state = shiftRows(state)
305         state = mixColumns(state)
306         state = addRoundKey(state, expandedKeys[(16 * (i+1)):(16 * (i+2))])
307     }
308     // Last round
309     state = subBytes(state)
310     state = shiftRows(state)
311     state = addRoundKey(state, expandedKeys[160:])
312
313     return state
314 }
315
316 func decrypt(state []byte, expandedKeys [176]byte, regularRounds int) ([]byte) {
317     state = addRoundKey(state, expandedKeys[160:])
318     state = invShiftRows(state)
319     state = invSubBytes(state)
320
321     for i := regularRounds; i != 0; i-- {
322         state = addRoundKey(state, expandedKeys[(16 * (i)):(16 * (i+1))])
323         state = invMixColumns(state)
324         state = invShiftRows(state)
325         state = invSubBytes(state)
326     }
327     // Last round
328     state = addRoundKey(state, expandedKeys[:16])
329
330     return state
331 }
332
333
334 func check(e error) { // Used for checking errors when reading/writing to files.
335     if e != nil {
336         panic(e)
337     }

```

```

338 }
339
340 func compareSlices(slice1, slice2 []byte) bool { // Function used for checking first block of a file
with the key when decrypting.
341     if len(slice1) != len(slice2) {
342         return false
343     } else {
344         for i := 0; i < len(slice1); i++ {
345             if slice1[i] != slice2[i] {
346                 return false
347             }
348         }
349     }
350     return true
351 }
352
353 func encryptFile(key []byte, f, w string) {
354     a, err := os.Open(f) // Open original file to get statistics
355     check(err)
356     aInfo, err := a.Stat() // Get statistics
357     check(err)
358
359     fileSize := int(aInfo.Size()) // Get size of original file
360
361     var expandedKeys [176]byte
362     expandedKeys = expandKey(key) // Expand the key for each round
363
364     if _, err := os.Stat(w); err == nil { // If file already exists, delete it
365         os.Remove(w)
366     }
367
368     var bufferSize int = 32768 // The buffer size is 2^15 (I went up powers of 2 to find best
performance)
369
370     if fileSize < bufferSize { // If the buffer size is larger than the file size, just read the whole
file.
371         bufferSize = fileSize
372     }
373
374     var buffCount int = 0 // Keeps track of how far through the file we are
375
376     e, err := os.OpenFile(w, os.O_CREATE|os.O_WRONLY|os.O_APPEND, 0644) // Open file for appending.
377     check(err) // Check it opened correctly
378
379     // Append key so that when decrypting, the key can be checked before decrypting the whole file.
380     e.Write(encrypt(key, expandedKeys, 9))
381     e.Seek(16, 0) // Move where we are writing to past the key.
382
383     for buffCount < fileSize { // Same as a while buffCount < fileSize: in python3
384         if bufferSize > (fileSize - buffCount) {
385             bufferSize = fileSize - buffCount // If this is the last block, read the amount of data left in
the file.
386         }
387
388         buff := make([]byte, bufferSize) // Make a slice the size of the buffer
389         _, err := io.ReadFull(a, buff) // Read the contents of the original file, but only enough to fill
the buff array.
390
391         // The "_" tells go to ignore the value returned by io.ReadFull,
which in this case is the number of bytes read.
392         check(err)
393
394         if len(buff) % 16 != 0 { // If the buffer is not divisible by 16 (usually the end of a file), then
padding needs to be added.
395             var extraNeeded int
396             var l int = len(buff)
397             for l % 16 != 0 { // extraNeeded holds the value for how much padding the block needs.
398                 l++
399                 extraNeeded++
400             }

```

```

400
401     for i := 0; i < extraNeeded; i++){           // Add the number of extra bytes needed to the
end of the block, if the block is not long enough.
402         buff = append(buff, byte(extraNeeded)) // For example, the array [1, 1, 1, 1, 1, 1, 1, 1] would
have the number 8 appended to then end 8 times to make the array 16 in length.
403     } // This is so that when the block is decrypted, the pattern can be recognised, and the correct
amount of padding can be removed.
404     }
405
406     var encBuff []byte    // Make a buffer to hold encrypted data.
407     for i := 0; i < bufferSize; i += 16 {
408         encBuff = append(encBuff, encrypt(buff[i:i+16], expandedKeys, 9)...)
409     }
410     e.Write(encBuff) // Buffer is used because accessing the file every 16 bytes slows down the process
a lot.
411
412     buffCount += bufferSize
413 }
414 a.Close() // Close the files used.
415 e.Close()
416 }
417
418
419 func decryptFile(key []byte, f, w string) {
420     a, err := os.Open(f)
421     check(err)
422     aInfo, err := a.Stat()
423     check(err)
424
425     fileSize := int(aInfo.Size())-16 // Take away length of added key for checksum
426
427     var expandedKeys [176]byte
428
429     expandedKeys = expandKey(key)
430
431     if _, err := os.Stat(w); err == nil { // If file exists, delete it
432         os.Remove(w)
433     }
434
435     var bufferSize int = 32768
436
437     if fileSize < bufferSize {
438         bufferSize = fileSize
439     }
440
441     var buffCount int = 0
442
443     e, err := os.OpenFile(w, os.O_CREATE|os.O_WRONLY|os.O_APPEND, 0644) // Open file for appending.
444     check(err)
445
446     // Check first block is key
447     firstBlock := make([]byte, 16)
448     _, er := io.ReadFull(a, firstBlock)
449     check(er)
450     decFirst := decrypt(firstBlock, expandedKeys, 9)
451
452     if compareSlices(key, decFirst) {
453         a.Seek(16, 0)           // Move read head 16 bytes into the file
454         for buffCount < fileSize{ // While the data done is less than the fileSize
455             if bufferSize > (fileSize - buffCount) {
456                 bufferSize = fileSize - buffCount
457             }
458
459             buff := make([]byte, bufferSize)
460             _, err := io.ReadFull(a, buff) // Ignore the number of bytes read (_)
461             check(err)
462
463             var decBuff []byte
464             for i := 0; i < bufferSize; i += 16 {

```

```

465         if fileSize - i == 16 { // If on the last block of whole file
466             var decrypted []byte = decrypt(buff[i:i+16], expandedKeys, 9) // Decrypt 128 bit chunk of
buffer
467             // Store in variable as we are going to change it.
468             var focus int = int(decrypted[len(decrypted)-1])
469             var focusCount int = 0
470
471             if focus < 16 { // If the last number is less than 16 (the maximum amount of padding to
add is 15)
472                 for j := 15; int(decrypted[j]) == focus; j-- {
473                     if int(decrypted[j]) == focus {focusCount++}
474                 }
475                 if focus == focusCount {
476                     decrypted = decrypted[:16-focus] // If the number of bytes at the end is equal to the
value of each byte, then remove them, as it is padding.
477                 }
478             }
479             decBuff = append(decBuff, decrypted...) // ... is to say that I want to append the items in
the array to the decBuff, rather than append the array itself.
480         } else {
481             decBuff = append(decBuff, decrypt(buff[i:i+16], expandedKeys, 9)...)
482         }
483     }
484     e.Write(decBuff)
485
486     buffCount += bufferSize
487 }
488 } else {
489     panic("Invalid Key") // If first block is not equal to the key, then do not bother trying to
decrypt the file.
490 }
491 a.Close()
492 e.Close()
493 }
494
495
496 func checkKey(key []byte, f string) bool{
497     a, err := os.Open(f) // Open an encrypted file to check first block against key
498     check(err)
499
500     var expandedKeys [176]byte
501
502     expandedKeys = expandKey(key) // Expand the key
503
504     // Check first block is key
505     firstBlock := make([]byte, 16)
506     _, er := io.ReadFull(a, firstBlock) // Fill a slice of length 16 with the first block of 16 bytes in
the file.
507     check(er)
508     firstDecrypted := decrypt(firstBlock, expandedKeys, 9) // Decrypt first block
509
510     a.Close()
511     return compareSlices(key, firstDecrypted) // Compare decrypted first block with the key.
512 }
513
514 func strToInt(str string) (int, error) { // Used for converting string to integer, as go doesn't have
that built in for some reason
515     n := strings.Split(str, ".") // Splits by decimal point
516     return strconv.Atoi(n[0]) // Returns integer of whole number
517 }
518
519
520 func main() {
521     bytes, err := ioutil.ReadAll(os.Stdin)
522     check(err)
523     fields := strings.Split(string(bytes), ", ") // Splits input by ", "
524     keyString := strings.Split(string(fields[3]), " ") // Splits the key by " "
525
526     var key []byte

```

```

527 for i := 0; i < len(keyString); i++ {
528     a, err := strconv.Atoi(keyString[i])
529     check(err)
530     key = append(key, byte(a))
531 }
532
533 if string(fields[0]) == "y" {
534     encryptFile(key, string(fields[1]), string(fields[2]))
535 } else if string(fields[0]) == "n" {
536     decryptFile(key, string(fields[1]), string(fields[2]))
537 } else if string(fields[0]) == "test" {
538     valid := checkKey(key, string(fields[1]))
539     if valid {
540         fmt.Println("-Valid-")
541     } else {
542         fmt.Println("-NotValid-")
543     }
544 } else {
545     panic("Invalid options.")
546 }
547 }

```

The program accepts the fields `<encryptionType>`, `<field1>`, `<field2>`, `<key>`, where `<field1>` is the first argument of the function you want to execute, and `<field2>` is the second argument. If there is no second argument, then this can just be set to `0`. The key is input like this: `1 2 3 4 5 6 7 8 9 10 11 12 13 14 15` (it is hashed first though), where it is then split by the space in-between each number, and each number is converted to a byte, where it can be used in the functions that need it.

`aes.go` is compiled to `AES` for Linux/macOS, and `AESWin.exe` for Windows.

AES for file names:

AES for file names is written in Python, as it needs to be accessed a lot, and does work on small volumes of data. Here is the code (`code/python-go/aesFName.py`):

```

1  # For more information on each function, look at aes.go since most of this is the same.
2
3  #Lookup tables
4  #      0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f      - first
   digit of input
5  sBox = [0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76, #00
6         0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0xAF,0x9C,0xA4,0x72,0xC0, #10
7         0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15, #20
8         0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,0x9A,0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75, #30
9         0x09,0x83,0x2C,0x1A,0x1B,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84, #40
10        0x53,0xD1,0x00,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF, #50
11        0xD0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02,0x7F,0x50,0x3C,0x9F,0xA8, #60
12        0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,0x10,0xFF,0xF3,0xD2, #70
13        0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7,0x7E,0x3D,0x64,0x5D,0x19,0x73, #80
14        0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x88,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB, #90
15        0xE0,0x32,0x3A,0x0A,0x49,0x06,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79, #a0
16        0xE7,0xC8,0x37,0x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08, #b0
17        0xBA,0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B,0x8A, #c0
18        0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x86,0xC1,0x1D,0x9E, #d0
19        0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x87,0xE9,0xCE,0x55,0x28,0xDF, #e0
20        0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16] #f0
21
22  invSBox = [0x52,0x09,0x6A,0xD5,0x30,0x36,0xA5,0x38,0xBF,0x40,0xA3,0x9E,0x81,0xF3,0xD7,0xFB,
23            0x7C,0xE3,0x39,0x82,0x9B,0x2F,0xFF,0x87,0x34,0x8E,0x43,0x44,0xC4,0xDE,0xE9,0xCB,
24            0x54,0x7B,0x94,0x32,0xA6,0xC2,0x23,0x3D,0xEE,0x4C,0x95,0x0B,0x42,0xFA,0xC3,0x4E,
25            0x08,0x2E,0xA1,0x66,0x28,0xD9,0x24,0xB2,0x76,0x5B,0xA2,0x49,0x6D,0x8B,0xD1,0x25,

```

```

26      0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
27      0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
28      0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
29      0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
30      0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
31      0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
32      0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
33      0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
34      0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
35      0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
36      0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
37      0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D]
38
39  rcon = [0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
#https://en.wikipedia.org/wiki/Rijndael_key_schedule
40      0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
41      0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
42      0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
43      0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
44      0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
45      0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
46      0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
47      0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
48      0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
49      0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
50      0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
51      0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
52      0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
53      0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
54      0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d]
55
56  mul2 = [0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1a, 0x1c, 0x1e,
57      0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3a, 0x3c, 0x3e,
58      0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54, 0x56, 0x58, 0x5a, 0x5c, 0x5e,
59      0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74, 0x76, 0x78, 0x7a, 0x7c, 0x7e,
60      0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94, 0x96, 0x98, 0x9a, 0x9c, 0x9e,
61      0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4, 0xb6, 0xb8, 0xba, 0xbc, 0xbe,
62      0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde,
63      0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe,
64      0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x07, 0x05,
65      0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f, 0x2d, 0x23, 0x21, 0x27, 0x25,
66      0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f, 0x4d, 0x43, 0x41, 0x47, 0x45,
67      0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f, 0x6d, 0x63, 0x61, 0x67, 0x65,
68      0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f, 0x8d, 0x83, 0x81, 0x87, 0x85,
69      0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf, 0xad, 0xa3, 0xa1, 0xa7, 0xa5,
70      0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf, 0xcd, 0xc3, 0xc1, 0xc7, 0xc5,
71      0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed, 0xe3, 0xe1, 0xe7, 0xe5]
72
73  mul3 = [0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e, 0x1d, 0x14, 0x17, 0x12, 0x11,
74      0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28, 0x2b, 0x2e, 0x2d, 0x24, 0x27, 0x22, 0x21,
75      0x60, 0x63, 0x66, 0x65, 0x6c, 0x6f, 0x6a, 0x69, 0x78, 0x7b, 0x7e, 0x7d, 0x74, 0x77, 0x72, 0x71,
76      0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e, 0x4d, 0x44, 0x47, 0x42, 0x41,
77      0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8, 0xdb, 0xde, 0xdd, 0xd4, 0xd7, 0xd2, 0xd1,
78      0xf0, 0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed, 0xe4, 0xe7, 0xe2, 0xe1,
79      0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe, 0xbd, 0xb4, 0xb7, 0xb2, 0xb1,
80      0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88, 0x8b, 0x8e, 0x8d, 0x84, 0x87, 0x82, 0x81,
81      0x9b, 0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85, 0x86, 0x8f, 0x8c, 0x89, 0x8a,
82      0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5, 0xb6, 0xbf, 0xbc, 0xb9, 0xba,
83      0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xe3, 0xe0, 0xe5, 0xe6, 0xef, 0xec, 0xea, 0xeb,
84      0xcb, 0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5, 0xd6, 0xdf, 0xdc, 0xd9, 0xda,
85      0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45, 0x46, 0x4f, 0x4c, 0x49, 0x4a,
86      0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x73, 0x70, 0x75, 0x76, 0x7f, 0x7c, 0x79, 0x7a,
87      0x3b, 0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25, 0x26, 0x2f, 0x2c, 0x29, 0x2a,
88      0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15, 0x16, 0x1f, 0x1c, 0x19, 0x1a]
89
90  mul9 = [0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53, 0x6c, 0x65, 0x7e, 0x77,
91      0x90, 0x99, 0x82, 0x8b, 0xb4, 0xbd, 0xae, 0xaf, 0xd8, 0xd1, 0xca, 0xc3, 0xfc, 0xf5, 0xee, 0xe7,
92      0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x73, 0x7a, 0x61, 0x68, 0x57, 0x5e, 0x45, 0x4c,
93      0xab, 0xa2, 0xb9, 0xb0, 0x8f, 0x86, 0x9d, 0x94, 0xe3, 0xea, 0xf1, 0xf8, 0xc7, 0xce, 0xd5, 0xdc,

```

```

94     0x76, 0x7f, 0x64, 0x6d, 0x52, 0x5b, 0x40, 0x49, 0x3e, 0x37, 0x2c, 0x25, 0x1a, 0x13, 0x08, 0x01,
95     0xe6, 0xef, 0xf4, 0xfd, 0xc2, 0xcb, 0xd0, 0xd9, 0xae, 0xa7, 0xbc, 0xb5, 0x8a, 0x83, 0x98, 0x91,
96     0x4d, 0x44, 0x5f, 0x56, 0x69, 0x60, 0x7b, 0x72, 0x05, 0x0c, 0x17, 0x1e, 0x21, 0x28, 0x33, 0x3a,
97     0xdd, 0xd4, 0xcf, 0xc6, 0xf9, 0xf0, 0xeb, 0xe2, 0x95, 0x9c, 0x87, 0x8e, 0xb1, 0xb8, 0xa3, 0xaa,
98     0xec, 0xe5, 0xfe, 0xf7, 0xc8, 0xc1, 0xda, 0xd3, 0xa4, 0xad, 0xb6, 0xbf, 0x80, 0x89, 0x92, 0x9b,
99     0x7c, 0x75, 0x6e, 0x67, 0x58, 0x51, 0x4a, 0x43, 0x34, 0x3d, 0x26, 0x2f, 0x10, 0x19, 0x02, 0x0b,
100    0xd7, 0xde, 0xc5, 0xcc, 0xf3, 0xfa, 0xe1, 0xe8, 0x9f, 0x96, 0x8d, 0x84, 0xbb, 0xb2, 0xa9, 0xa0,
101    0x47, 0x4e, 0x55, 0x5c, 0x63, 0x6a, 0x71, 0x78, 0x0f, 0x06, 0x1d, 0x14, 0x2b, 0x22, 0x39, 0x30,
102    0x9a, 0x93, 0x88, 0x81, 0xbe, 0xb7, 0xac, 0xa5, 0xd2, 0xdb, 0xc0, 0xc9, 0xf6, 0xff, 0xe4, 0xed,
103    0x0a, 0x03, 0x18, 0x11, 0x2e, 0x27, 0x3c, 0x35, 0x42, 0x4b, 0x50, 0x59, 0x66, 0x6f, 0x74, 0x7d,
104    0xa1, 0xa8, 0xb3, 0xba, 0x85, 0x8c, 0x97, 0x9e, 0xe9, 0xe0, 0xfb, 0xf2, 0xcd, 0xc4, 0xdf, 0xd6,
105    0x31, 0x38, 0x23, 0x2a, 0x15, 0x1c, 0x07, 0x0e, 0x79, 0x70, 0x6b, 0x62, 0x5d, 0x54, 0x4f, 0x46]
106
107    mul11 = [0x00, 0x0b, 0x16, 0x1d, 0x2c, 0x27, 0x3a, 0x31, 0x58, 0x53, 0x4e, 0x45, 0x74, 0x7f, 0x62, 0x69,
108            0xb0, 0xbb, 0xa6, 0xad, 0x9c, 0x97, 0x8a, 0x81, 0xe8, 0xe3, 0xfe, 0xf5, 0xc4, 0xcf, 0xd2, 0xd9,
109            0x7b, 0x70, 0x6d, 0x66, 0x57, 0x5c, 0x41, 0x4a, 0x23, 0x28, 0x35, 0x3e, 0x0f, 0x04, 0x19, 0x12,
110            0xcb, 0xc0, 0xdd, 0xd6, 0xe7, 0xec, 0xf1, 0xfa, 0x93, 0x98, 0x85, 0x8e, 0xbf, 0xb4, 0xa9, 0xa2,
111            0xf6, 0xfd, 0xe0, 0xeb, 0xda, 0xd1, 0xcc, 0xc7, 0xae, 0xa5, 0xb8, 0xb3, 0x82, 0x89, 0x94, 0x9f,
112            0x46, 0x4d, 0x50, 0x5b, 0x6a, 0x61, 0x7c, 0x77, 0x1e, 0x15, 0x08, 0x03, 0x32, 0x39, 0x2f,
113            0x8d, 0x86, 0x9b, 0x90, 0xa1, 0xaa, 0xb7, 0xbc, 0xd5, 0xde, 0xc3, 0xc8, 0xf9, 0xf2, 0xef, 0xe4,
114            0x3d, 0x36, 0x2b, 0x20, 0x11, 0x1a, 0x07, 0x0c, 0x65, 0x6e, 0x73, 0x78, 0x49, 0x42, 0x5f, 0x54,
115            0xf7, 0xfc, 0xe1, 0xea, 0xdb, 0xd0, 0xcd, 0xc6, 0xaf, 0xa4, 0xb9, 0xb2, 0x83, 0x88, 0x95, 0x9e,
116            0x47, 0x4c, 0x51, 0x5a, 0x6b, 0x60, 0x7d, 0x76, 0x1f, 0x14, 0x09, 0x02, 0x33, 0x38, 0x25, 0x2e,
117            0x8c, 0x87, 0x9a, 0x91, 0xa0, 0xab, 0xb6, 0xbd, 0xd4, 0xdf, 0xc2, 0xc9, 0xf8, 0xf3, 0xee, 0xe5,
118            0x3c, 0x37, 0x2a, 0x21, 0x10, 0x1b, 0x06, 0x0d, 0x64, 0x6f, 0x72, 0x79, 0x48, 0x43, 0x5e, 0x55,
119            0x01, 0x0a, 0x17, 0x1c, 0x2d, 0x26, 0x3b, 0x30, 0x59, 0x52, 0x4f, 0x44, 0x75, 0x7e, 0x63, 0x68,
120            0xb1, 0xba, 0xa7, 0xac, 0x9d, 0x96, 0x8b, 0x80, 0xe9, 0xe2, 0xff, 0xf4, 0xc5, 0xce, 0xd3, 0xd8,
121            0x7a, 0x71, 0x6c, 0x67, 0x56, 0x5d, 0x40, 0x4b, 0x22, 0x29, 0x34, 0x3f, 0x0e, 0x05, 0x18, 0x13,
122            0xca, 0xc1, 0xdc, 0xd7, 0xe6, 0xed, 0xf0, 0xfb, 0x92, 0x99, 0x84, 0x8f, 0xbe, 0xb5, 0xa8, 0xa3]
123
124    mul13 = [0x00, 0x0d, 0x1a, 0x17, 0x34, 0x39, 0x2e, 0x23, 0x68, 0x65, 0x72, 0x7f, 0x5c, 0x51, 0x46, 0x4b,
125            0xd0, 0xdd, 0xca, 0xc7, 0xe4, 0xe9, 0xfe, 0xf3, 0xb8, 0xb5, 0xa2, 0xaf, 0x8c, 0x81, 0x96, 0x9b,
126            0xbb, 0xb6, 0xa1, 0xac, 0x8f, 0x82, 0x95, 0x98, 0xd3, 0xde, 0xc9, 0xc4, 0xe7, 0xea, 0xfd, 0xf0,
127            0x6b, 0x66, 0x71, 0x7c, 0x5f, 0x52, 0x45, 0x48, 0x03, 0x0e, 0x19, 0x14, 0x37, 0x3a, 0x2d, 0x20,
128            0x6d, 0x60, 0x77, 0x7a, 0x59, 0x54, 0x43, 0x4e, 0x05, 0x08, 0x1f, 0x12, 0x31, 0x3c, 0x2b, 0x26,
129            0xbd, 0xb0, 0xa7, 0xaa, 0x89, 0x84, 0x93, 0x9e, 0xd5, 0xd8, 0xcf, 0xc2, 0xe1, 0xec, 0xfb, 0xf6,
130            0xd6, 0xdb, 0xcc, 0xc1, 0xe2, 0xef, 0xf8, 0xf5, 0xbe, 0xb3, 0xa4, 0xa9, 0x8a, 0x87, 0x90, 0x9d,
131            0x06, 0x0b, 0x1c, 0x11, 0x32, 0x3f, 0x28, 0x25, 0x6e, 0x63, 0x74, 0x79, 0x5a, 0x57, 0x40, 0x4d,
132            0xda, 0xd7, 0xc0, 0xcd, 0xee, 0xe3, 0xf4, 0xf9, 0xb2, 0xbf, 0xa8, 0xa5, 0x86, 0x8b, 0x9c, 0x91,
133            0x0a, 0x07, 0x10, 0x1d, 0x3e, 0x33, 0x24, 0x29, 0x62, 0x6f, 0x78, 0x75, 0x56, 0x5b, 0x4c, 0x41,
134            0x61, 0x6c, 0x7b, 0x76, 0x55, 0x58, 0x4f, 0x42, 0x09, 0x04, 0x13, 0x1e, 0x3d, 0x30, 0x27, 0x2a,
135            0xb1, 0xbc, 0xab, 0xa6, 0x85, 0x88, 0x9f, 0x92, 0xd9, 0xd4, 0xc3, 0xce, 0xed, 0xe0, 0xf7, 0xfa,
136            0xb7, 0xba, 0xad, 0xa0, 0x83, 0x8e, 0x99, 0x94, 0xdf, 0xd2, 0xc5, 0xc8, 0xeb, 0xe6, 0xf1, 0xfc,
137            0x67, 0x6a, 0x7d, 0x70, 0x53, 0x5e, 0x49, 0x44, 0x0f, 0x02, 0x15, 0x18, 0x3b, 0x36, 0x21, 0x2c,
138            0x0c, 0x01, 0x16, 0x1b, 0x38, 0x35, 0x22, 0x2f, 0x64, 0x69, 0x7e, 0x73, 0x50, 0x5d, 0x4a, 0x47,
139            0xdc, 0xd1, 0xc6, 0xcb, 0xe8, 0xe5, 0xf2, 0xff, 0xb4, 0xb9, 0xae, 0xa3, 0x80, 0x8d, 0x9a, 0x97]
140
141    mul14 = [0x00, 0x0e, 0x1c, 0x12, 0x38, 0x36, 0x24, 0x2a, 0x70, 0x7e, 0x6c, 0x62, 0x48, 0x46, 0x54, 0x5a,
142            0xe0, 0xee, 0xfc, 0xf2, 0xd8, 0xd6, 0xc4, 0xca, 0x90, 0x9e, 0x8c, 0x82, 0xa8, 0xa6, 0xb4, 0xba,
143            0xdb, 0xd5, 0xc7, 0xc9, 0xe3, 0xed, 0xff, 0xf1, 0xab, 0xa5, 0xb7, 0xb9, 0x93, 0x9d, 0x8f, 0x81,
144            0x3b, 0x35, 0x27, 0x29, 0x03, 0x0d, 0x1f, 0x11, 0x4b, 0x45, 0x57, 0x59, 0x73, 0x7d, 0x6f, 0x61,
145            0xad, 0xa3, 0xb1, 0xbf, 0x95, 0x9b, 0x89, 0x87, 0xdd, 0xd3, 0xc1, 0xcf, 0xe5, 0xeb, 0xf9, 0xf7,
146            0x4d, 0x43, 0x51, 0x5f, 0x75, 0x7b, 0x69, 0x67, 0x3d, 0x33, 0x21, 0x2f, 0x05, 0x0b, 0x19, 0x17,
147            0x76, 0x78, 0x6a, 0x64, 0x4e, 0x40, 0x52, 0x5c, 0x06, 0x08, 0x1a, 0x14, 0x3e, 0x30, 0x22, 0x2c,
148            0x96, 0x98, 0x8a, 0x84, 0xae, 0xa0, 0xb2, 0xbc, 0xe6, 0xe8, 0xfa, 0xf4, 0xde, 0xd0, 0xc2, 0xcc,
149            0x41, 0x4f, 0x5d, 0x53, 0x79, 0x77, 0x65, 0x6b, 0x31, 0x3f, 0x2d, 0x23, 0x09, 0x07, 0x15, 0x1b,
150            0xa1, 0xaf, 0xbd, 0xb3, 0x99, 0x97, 0x85, 0x8b, 0xd1, 0xdf, 0xcd, 0xc3, 0xe9, 0xe7, 0xf5, 0xfb,
151            0x9a, 0x94, 0x86, 0x88, 0xa2, 0xac, 0xbe, 0xb0, 0xea, 0xe4, 0xf6, 0xf8, 0xd2, 0xdc, 0xce, 0xc0,
152            0x7a, 0x74, 0x66, 0x68, 0x42, 0x4c, 0x5e, 0x50, 0x0a, 0x04, 0x16, 0x18, 0x32, 0x3c, 0x2e, 0x20,
153            0xec, 0xe2, 0xf0, 0xfe, 0xd4, 0xda, 0xc8, 0xc6, 0x9c, 0x92, 0x80, 0x8e, 0xa4, 0xaa, 0xb8, 0xb6,
154            0x0c, 0x02, 0x10, 0x1e, 0x34, 0x3a, 0x28, 0x26, 0x7c, 0x72, 0x60, 0x6e, 0x44, 0x4a, 0x58, 0x56,
155            0x37, 0x39, 0x2b, 0x25, 0x0f, 0x01, 0x13, 0x1d, 0x47, 0x49, 0x5b, 0x55, 0x7f, 0x71, 0x63, 0x6d,
156            0xd7, 0xd9, 0xcb, 0xc5, 0xef, 0xe1, 0xf3, 0xfd, 0xa7, 0xa9, 0xbb, 0xb5, 0x9f, 0x91, 0x83, 0x8d]
157
158
159    def keyExpansionCore(inp, i):
160        #Shift the inp left by moving the first byte to the end (rotate).
161        inp[0], inp[1], inp[2], inp[3] = inp[1], inp[2], inp[3], inp[0]
162        #S-Box the bytes

```



```

163 inp[0], inp[1], inp[2], inp[3] = sBox[inp[0]], sBox[inp[1]], sBox[inp[2]], sBox[inp[3]]
164 #rcon, more galois feilds that lead to lookup tables.
165 inp[0] ^= rcon[i]
166
167 return inp
168
169 def expandKey(inputKey, expandedKeys):
170     #first 16 bytes of the expandedKeys should be the same 16 as the original key
171     for i in range(16):
172         expandedKeys[i] = inputKey[i]
173
174     bytesGenerated = 16 #needs to get to 176
175     rconIteration = 1
176     temp = [0, 0, 0, 0]
177
178     while bytesGenerated < 176:
179         #Read 4 bytes for use in keyExpansionCore
180         temp = expandedKeys[bytesGenerated-4:bytesGenerated]
181
182         if bytesGenerated % 16 == 0: #keys are length 16 bytes so every 16 bytes generated, expand.
183             temp = keyExpansionCore(temp, rconIteration)
184             rconIteration += 1
185
186         for y in range(4):
187             expandedKeys[bytesGenerated] = expandedKeys[bytesGenerated - 16] ^ temp[y]
188             bytesGenerated += 1
189
190     return expandedKeys
191
192
193 def addRoundKey(state, roundKey): #is also it's own inverse
194     return [state[0]^roundKey[0], state[1]^roundKey[1], state[2]^roundKey[2], state[3]^roundKey[
195 3],
196             state[4]^roundKey[4], state[5]^roundKey[5], state[6]^roundKey[6], state[7]^roundKey[
197 7],
198             state[8]^roundKey[8], state[9]^roundKey[9], state[10]^roundKey[10],
199 state[11]^roundKey[11],
200             state[12]^roundKey[12], state[13]^roundKey[13], state[14]^roundKey[14],
201 state[15]^roundKey[15]]
202
203 def subBytes(state): # For loops in python are quite bad, so this gives you about 6 KB/s more speed. In
204 my Go implementation a for loop is faster.
205     return [sBox[state[0]], sBox[state[1]], sBox[state[2]], sBox[state[3]],
206             sBox[state[4]], sBox[state[5]], sBox[state[6]], sBox[state[7]],
207             sBox[state[8]], sBox[state[9]], sBox[state[10]], sBox[state[11]],
208             sBox[state[12]], sBox[state[13]], sBox[state[14]], sBox[state[15]]]
209
210 def invSubBytes(state):
211     for i in range(16):
212         state[i] = invSBox[state[i]]
213
214     return state
215
216 def shiftRows(state):
217     return [state[0], state[5], state[10], state[15],
218             state[4], state[9], state[14], state[3],
219             state[8], state[13], state[2], state[7],
220             state[12], state[1], state[6], state[11]]
221
222 def invShiftRows(state):
223     return [state[0], state[13], state[10], state[7],
224             state[4], state[1], state[14], state[11],
225             state[8], state[5], state[2], state[15],
226             state[12], state[9], state[6], state[3]]
227
228 def mixColumns(state):
229     return [mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3], # Col 1
230             state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3],
231             state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]],

```

```

227     mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]],
228
229     mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7], # Col 2
230     state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7],
231     state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]],
232     mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]],
233
234     mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11], # Col 3
235     state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11],
236     state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]],
237     mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]],
238
239     mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15], # Col 4
240     state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15],
241     state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]],
242     mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]]
243
244 def invMixColumns(state):
245     return [mul14[state[0]] ^ mul11[state[1]] ^ mul13[state[2]] ^ mul9[state[3]],
246             mul9[state[0]] ^ mul14[state[1]] ^ mul11[state[2]] ^ mul13[state[3]],
247             mul13[state[0]] ^ mul9[state[1]] ^ mul14[state[2]] ^ mul11[state[3]],
248             mul11[state[0]] ^ mul13[state[1]] ^ mul9[state[2]] ^ mul14[state[3]],
249
250             mul14[state[4]] ^ mul11[state[5]] ^ mul13[state[6]] ^ mul9[state[7]],
251             mul9[state[4]] ^ mul14[state[5]] ^ mul11[state[6]] ^ mul13[state[7]],
252             mul13[state[4]] ^ mul9[state[5]] ^ mul14[state[6]] ^ mul11[state[7]],
253             mul11[state[4]] ^ mul13[state[5]] ^ mul9[state[6]] ^ mul14[state[7]],
254
255             mul14[state[8]] ^ mul11[state[9]] ^ mul13[state[10]] ^ mul9[state[11]],
256             mul9[state[8]] ^ mul14[state[9]] ^ mul11[state[10]] ^ mul13[state[11]],
257             mul13[state[8]] ^ mul9[state[9]] ^ mul14[state[10]] ^ mul11[state[11]],
258             mul11[state[8]] ^ mul13[state[9]] ^ mul9[state[10]] ^ mul14[state[11]],
259
260             mul14[state[12]] ^ mul11[state[13]] ^ mul13[state[14]] ^ mul9[state[15]],
261             mul9[state[12]] ^ mul14[state[13]] ^ mul11[state[14]] ^ mul13[state[15]],
262             mul13[state[12]] ^ mul9[state[13]] ^ mul14[state[14]] ^ mul11[state[15]],
263             mul11[state[12]] ^ mul13[state[13]] ^ mul9[state[14]] ^ mul14[state[15]]]
264
265 def padKey(key):
266     while len(key) != 16:
267         key.append(0)
268
269     return key
270
271 def checkForPadding(inp): # Padding for file names can be any number that is not in the ascii
272     while inp[len(inp)-1] == 0: # 0 is not a letter and is not punctuation.
273         inp = inp[:len(inp)-2]
274
275     return inp
276
277 def encrypt(state, expandedKeys, regularRounds):
278     state = addRoundKey(state, expandedKeys[:16])
279
280     for i in range(regularRounds):
281         state = subBytes(state)
282         state = shiftRows(state)
283         state = mixColumns(state)
284         state = addRoundKey(state, expandedKeys[(16 * (i+1)):(16 * (i+2))])
285
286     #Last round
287     state = subBytes(state)
288     state = shiftRows(state)
289     state = addRoundKey(state, expandedKeys[160:])
290
291     return state
292
293 def decrypt(state, expandedKeys, regularRounds):
294     state = addRoundKey(state, expandedKeys[160:])

```

[illegible]

```

353     for i in range(len(byteName)//16):
354         encBlock = decrypt(byteName[(i*16):((i+1)*16)], expandedKeys, 9)
355         # print(encBlock, "encBlock")
356         for element in encBlock:
357             if (element < 127) and (element > 31): #Removes padding by only adding if character.
358                 encName.append(chr(element))
359
360     return "".join(encName)
361
362 def convHexDigestToBytes(hexIn):          # Used when decrypting the file name
363     hexList = []
364     for i in range(0, len(hexIn), 2):
365         hexList.append(hexIn[i]+hexIn[i+1]) # Append each 2 digits to the list
366     for j in range(len(hexList)):
367         hexList[j] = int("0x"+hexList[j], 16) # Convert each 2 digits back to hex.
368
369     return bytearray(hexList)
370
371

```

When a file name is encrypted, the original name is encoded into bytes, it is encrypted, then each byte is converted to its hex representation. The `0x` is removed, and if the hex doesn't have 2 digits, then a `0` is added at the front, as Python's `hex` function returns `0x9` if the number was `9`, for example. This means that when you go to change this hex string back into numbers, you can't tell the numbers apart, so you have to make each 2 digits in length.

SHA256:

Here is the code for SHA256 (`code/python-go/SHA.py` , `code/mobile/SHA.py`):

```

1  k = [0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,    #Round constants
2      0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
3      0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
4      0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
5      0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
6      0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
7      0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
8      0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
9      0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
10     0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
11     0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
12     0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
13     0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
14     0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
15     0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
16     0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2]
17
18 def makeBitArray(inp):
19     bitArray = []
20     for element in inp:
21         tempByte = intToBits(element)
22         for bit in tempByte:
23             bitArray.append(bit)
24     return bitArray
25
26 def intToBits(inp, bitLength=8):
27     tempByte = []
28     for x in range(bitLength):
29         tempByte.append(0) #Initialize

```

[illegible]

```

95     for i in range(32):
96         temp[i] = word[checkLessThan32(i-amount)]
97     return temp
98
99 def addMod2W(array1, array2, W=32):    # Adds  $2^W$  two arrays, so that the word does not overflow it's
word length
100     if len(array1) != len(array2):
101         raise IndexError("Arrays not same size - ", array1, array2)
102     return intToBits((bitsToInt(array1) + bitsToInt(array2)) % 2**W, 32)
103
104 def Shr(x, n):
105     temp = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
106
107     for i in range(32):
108         temp[i] = checkShiftInBounds(x, i-n)
109     return temp
110
111 def SigExpansion0(x):
112     return xorArrays(xorArrays(RotR(x, 7), RotR(x, 18)), Shr(x, 3))
113
114 def SigExpansion1(x):
115     return xorArrays(xorArrays(RotR(x, 17), RotR(x, 19)), Shr(x, 10))
116
117 def Sig0(x):
118     return xorArrays(xorArrays(RotR(x, 2), RotR(x, 13)), RotR(x, 22))
119
120 def Sig1(x):
121     return xorArrays(xorArrays(RotR(x, 6), RotR(x, 11)), RotR(x, 25))
122
123 def Ch(x, y, z):
124     return xorArrays(andBitArrays(x, y), andBitArrays(notArray(x), z))
125
126 def Maj(x, y, z):
127     return xorArrays(xorArrays(andBitArrays(x, y), andBitArrays(x, z)), andBitArrays(y, z))
128
129 def sha256(inp):
130     #Initial hash values - https://csrc.nist.gov/csrc/media/publications/fips/180/4/archive/2012-03-06/documents/fips180-4.pdf section 5.3.3
131     hList = [0x6a09e667,    # H0
132              0xbb67ae85,    # H1
133              0x3c6ef372,    # H2
134              0xa54ff53a,    # H3
135              0x510e527f,    # H4
136              0x9b05688c,    # H5
137              0x1f83d9ab,    # H6
138              0x5be0cd19]    # H7
139
140     bits = makeBitArray(inp)
141     bits = pad(bits)
142     bits = [bits[x:x+32] for x in range(0, len(bits), 32)]    #Split padded message into 32 bit words
143     bits = bits+[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
0, 0, 0] for y in range(48)]
144     #Main part
145
146     for x in range(16, 64): #Expand current bits to be 64 words
147         bits[x] = addMod2W(addMod2W(addMod2W(bits[x-16], SigExpansion0(bits[x-15])), bits[x-7]),
SigExpansion1(bits[x-2]))
148
149     a = intToBits(hList[0], 32)
150     b = intToBits(hList[1], 32)
151     c = intToBits(hList[2], 32)
152     d = intToBits(hList[3], 32)
153     e = intToBits(hList[4], 32)
154     f = intToBits(hList[5], 32)
155     g = intToBits(hList[6], 32)
156     h = intToBits(hList[7], 32)
157
158     for i in range(64):

```

```

159     temp1 = addMod2W(addMod2W(addMod2W(addMod2W(h, Sig1(e)), Ch(e, f, g)), intToBits(k[i], 32)),
bits[i])
160     S0 = Sig0(a)
161     maj = Maj(a, b, c)
162
163     h = g
164     g = f
165     f = e
166     e = addMod2W(d, temp1)
167     d = c
168     c = b
169     b = a
170     a = addMod2W(temp1, addMod2W(S0, maj))
171
172     resultBits = addMod2W(intToBits(hList[0], 32), a)+addMod2W(intToBits(hList[1], 32),
b)+addMod2W(intToBits(hList[2], 32), c)+addMod2W(intToBits(hList[3], 32),
d)+addMod2W(intToBits(hList[4], 32), e)+addMod2W(intToBits(hList[5], 32),
f)+addMod2W(intToBits(hList[6], 32), g)+addMod2W(intToBits(hList[7], 32), h)
    # Looks really ugly but works better (otherwise I would have to store each in variables)
173
174
175     resultBytes = [resultBits[x:x+8] for x in range(0, len(resultBits), 8)] # Makes 2D array of bytes
176     result = []
177     for byte in resultBytes:
178         result.append(bitsToInt(byte)) # Converts each byte into an integer
179     return result
180
181 def getSHA128of16(data):
182     out = sha256(data)
183     return [out[i]^out[i+16] for i in range(16)]

```

Each byte is made into an array of bits. Doing it this way made it easier to debug, however probably made the algorithm much slower than it needed to be. However, I don't really care too much about how fast SHA is, as it is only used a few times in the program, and only ever works on very small amounts of data, so it will probably be unnoticeable for the user.

The file is called SHA.py, and is imported by LoginScreen (default login without Bluetooth), which is in `code/kivyStuff/loginClass.py`, for use when the key is entered.

BLAKE2b:

Here is the code for BLAKE2b (`code/python-go/blake.go`):

```

1  package main
2
3  import (
4      "fmt"
5      "math"
6      "os"
7      "io"
8      "io/ioutil"
9  )
10
11
12  // Inital constants.
13  var k = [8]uint64 {0x6A09E667F3BCC908,
14                    0xBB67AE8584CAA73B,
15                    0x3C6EF372FE94F82B,
16                    0xA54FF53A5F1D36F1,
17                    0x510E527FADE682D1,
18                    0x9B05688C2B3E6C1F,
19                    0x1F83D9ABFB41BD6B,

```

```

20         0x5BE0CD19137E2179}
21
22     var sigma = [12][16]uint64 {
23         {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15},
24         {14, 10, 4, 8, 9, 15, 13, 6, 1, 12, 0, 2, 11, 7, 5, 3},
25         {11, 8, 12, 0, 5, 2, 15, 13, 10, 14, 3, 6, 7, 1, 9, 4},
26         {7, 9, 3, 1, 13, 12, 11, 14, 2, 6, 5, 10, 4, 0, 15, 8},
27         {9, 0, 5, 7, 2, 4, 10, 15, 14, 1, 11, 12, 6, 8, 3, 13},
28         {2, 12, 6, 10, 0, 11, 8, 3, 4, 13, 7, 5, 15, 14, 1, 9},
29         {12, 5, 1, 15, 14, 13, 4, 10, 0, 7, 6, 3, 9, 2, 8, 11},
30         {13, 11, 7, 14, 12, 1, 3, 9, 5, 0, 15, 4, 8, 6, 2, 10},
31         {6, 15, 14, 9, 11, 3, 0, 8, 12, 2, 13, 7, 1, 4, 10, 5},
32         {10, 2, 8, 4, 7, 6, 1, 5, 15, 11, 9, 14, 3, 12, 13, 0},
33         {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15},
34         {14, 10, 4, 8, 9, 15, 13, 6, 1, 12, 0, 2, 11, 7, 5, 3}}
35
36     // Research: https://tools.ietf.org/pdf/rfc7693.pdf
37
38     func check(e error) { //Used for checking errors when reading/writing to files.
39         if e != nil {
40             panic(e)
41         }
42     }
43
44     func rotR64(in uint64, n int) uint64 { // Rotates 64 bit words right by n amount.
45         return (in >> uint(n)) ^ (in << (64 - uint(n)))
46     }
47
48     func mix(v [16]uint64, a, b, c, d int, x, y uint64) [16]uint64 {
49         v[a] = v[a] + v[b] + x
50         v[d] = rotR64((v[d] ^ v[a]), 32)
51
52         v[c] = v[c] + v[d]
53         v[b] = rotR64((v[b] ^ v[c]), 24)
54
55         v[a] = v[a] + v[b] + y
56         v[d] = rotR64((v[d] ^ v[a]), 16)
57
58         v[c] = v[c] + v[d]
59         v[b] = rotR64((v[b] ^ v[c]), 63)
60
61         return v
62     }
63
64     func get64(in []uint64) uint64 { // Gets a full 64-bit word from a list of 8 64-bit bytes.
65         return uint64(in[0] ^ (in[1] << 8) ^ (in[2] << 16) ^ (in[3] << 24) ^ (in[4] << 32) ^ (in[5] << 40) ^
66             (in[6] << 48) ^ (in[7] << 56))
67     }
68
69     func blakeCompress(h [8]uint64, block []uint64, t int, lastBlock bool) [8]uint64 { // Compressing
70         function
71         var v = [16]uint64{} // Current vector
72         for i := 0; i < 8; i++ {
73             v[i] = h[i]
74             v[i+8] = k[i]
75         }
76         v[12] = v[12] ^ uint64(math.Mod(float64(t), 18446744073709552000)) // 2 ^ 64 = 18446744073709552000
77         v[13] = v[13] ^ (uint64(t) >> 64)
78
79         if lastBlock {
80             v[14] = ^v[14] // NOT v[14]
81         }
82
83         var m [16] uint64
84         for i := 0; i < 16; i++ {
85             m[i] = get64(block[i*8:(i*8)+8])
86         }
87         for i := 0; i < 12; i++ {
88             // Mix

```



```

87     v = mix(v, 0, 4, 8, 12, m[sigma[i][0]], m[sigma[i][1]])
88     v = mix(v, 1, 5, 9, 13, m[sigma[i][2]], m[sigma[i][3]])
89     v = mix(v, 2, 6, 10, 14, m[sigma[i][4]], m[sigma[i][5]])
90     v = mix(v, 3, 7, 11, 15, m[sigma[i][6]], m[sigma[i][7]])
91
92     v = mix(v, 0, 5, 10, 15, m[sigma[i][8]], m[sigma[i][9]]) // Rows have been shifted
93     v = mix(v, 1, 6, 11, 12, m[sigma[i][10]], m[sigma[i][11]])
94     v = mix(v, 2, 7, 8, 13, m[sigma[i][12]], m[sigma[i][13]])
95     v = mix(v, 3, 4, 9, 14, m[sigma[i][14]], m[sigma[i][15]])
96 }
97
98 for i := 0; i < 8; i++ {
99     h[i] ^= v[i]
100    h[i] ^= v[i+8]
101 }
102
103 return h
104 }
105
106 func getNiceOutput(h [8]uint64) [64]byte {
107     var out [64]byte
108     for i := 0; i < 8; i++ {
109         for j := 8; j != 0; j-- {
110             out[i*8+(j-1)] = byte(((h[i] << uint64(64 - uint64((j)*8))) & 0xFFFFFFFFFFFFFFFF) >> 56)
111         }
112     }
113     return out
114 }
115
116 func BLAKEChecksum(f string, hashL int) [64]byte {
117     h := k // Initialize h0-7 with initial values.
118     h[0] = h[0] ^ (0x01010000 ^ uint64(hashL)) // Not using a key
119
120     a, err := os.Open(f) // Open file
121     check(err)
122     aInfo, err := a.Stat() // Get statistics of file
123     check(err)
124
125     fileSize := int(aInfo.Size()) // Get size of original file
126
127     var bufferSize int = 65536
128
129     if fileSize < bufferSize { // If the buffer size is larger than the file size, just read the whole
130         // file.
131         bufferSize = fileSize
132     }
133
134     var buffCount int = 0 // Keeps track of how far through the file we are
135     var bytesFed int = 0
136     var bytesLeft int = fileSize
137
138     for buffCount < fileSize {
139         if bufferSize > (fileSize - buffCount) {
140             bufferSize = fileSize - buffCount
141         }
142         buff := make([]uint64, bufferSize)
143         tempBuff := make([]byte, bufferSize) // Make a slice the size of the buffer
144         _, err := io.ReadFull(a, tempBuff) // Read the contents of the original file, but only enough to
145         // fill the buff array.
146         // The "_" tells go to ignore the value returned by io.ReadFull,
147         // which in this case is the number of bytes read.
148         check(err)
149         for i := range tempBuff {
150             buff[i] = uint64(tempBuff[i])
151         }
152         tempBuff = nil // Delete array
153
154         for len(buff) % 128 != 0 {
155             buff = append(buff, 0) // Append 0s when buffer is not long enough

```

```

153     }
154
155     for i := 0; i < bufferSize; i += 128 {
156         if bytesLeft <= 128 {
157             h = blakeCompress(h, buff[i:i+128], bytesFed+bytesLeft, true)
158         } else {
159             bytesFed += 128
160             h = blakeCompress(h, buff[i:i+128], bytesFed, false)
161         }
162         bytesLeft -= 128
163     }
164
165     buffCount += bufferSize
166 }
167 a.Close()
168
169 return getNiceOutput(h)
170 }
171
172 func main() {
173     bytes, err := ioutil.ReadAll(os.Stdin) // Read file to hash from stdin
174     check(err)
175     f := string(bytes)
176
177     fmt.Printf("%x", BLAKEchecksum(f, 64)) // Returns the hex digest in hex form over stdout
178 }

```

The way that BLAKE2b goes through the file is very similar to AES, so I stole some of the code from my AES and adapted it slightly.

The `main()` function is much simpler than AES's `main()` function, as I am only receiving one input: the path of the file that needs to be hashed.

`getNiceOutput` turns the array `h`, which contains 8 64-bit words, into an array of 64 bytes, that can then be turned into a hex output that is a bit more readable. The way it works is it generates a little-endian interpretation of the 64-bit words as bytes. So if I had the word `0D4D1C983FA580BA`, the output of the function would return `BA80A53F981C4D0D`. The function `getNiceOutput` uses bit masking (shifting the bits in the word around to leave the bits you want to change exposed) to get each byte of the 64-bit word, then appends the byte to the list in reverse order (since it is little-endian). Little-endian is just a way to store a number larger than a byte. Little-endian and big-endian are needed in computer systems because in memory, each address can only store a single byte, so if a number is bigger than that then the number needs to be split into separate bytes. For example, if I had the number `354`, then I would first convert that into binary: $2 + 32 + 64 + 256 = 354$, $= 101100010$, however this is larger than 8 bits, so split it into two:

00000001 and 01100010, where the first byte starts at 2^8 . Little-endian arranges these bytes in memory like this:

Address1: 01100010, Address2: 00000001

It is called little-endian because the smaller (little) number is stored in the first address (the end). Big-endian is just the opposite way around.

The Sorts:

Here is the code for the sorts (`code/python-go/sortsCythonSource/sortsCy.pyx`):

```


```

```

1  cpdef int compareStrings(fileObj, string2, fileObjects=True): # Returns 0 if str1 < str2, 1 if str1 >
    str2, and 2 if str1 == str2
2      cdef int count = 0
3
4      if fileObjects:
5          string1 = fileObj.name
6      else:
7          string1 = fileObj
8
9      while not (count >= len(string1) or count >= len(string2)):
10         if ord(string2[count].lower()) < ord(string1[count].lower()):
11             return 1
12         elif ord(string2[count].lower()) > ord(string1[count].lower()):
13             return 0
14         else:
15             if ord(string2[count]) < ord(string1[count]): #if the same name but with capitals - e.g
(Usb Backup) and (usb backup)
16                 return 1
17             elif ord(string2[count]) > ord(string1[count]):
18                 return 0
19             else:
20                 if string2 == string1:
21                     return 2
22                 else:
23                     count += 1
24         if len(string1) > len(string2):
25             return 1
26         elif len(string1) < len(string2):
27             return 0
28         else:
29             raise ValueError("Two strings are the same in compareStrings.")
30
31
32
33 cpdef list quickSortAlph(list myList, fileObjects=True): #Quick sorts alphabetically
34     cdef list left = []
35     cdef list right = [] #Make seperate l+r lists, and add on at the end.
36     cdef list middle = []
37     if len(myList) > 1:
38         pivot = myList[int(len(myList)/2)]
39         for item in myList:
40             if fileObjects:
41                 leftSide = compareStrings(pivot, item.name)
42             else:
43                 leftSide = compareStrings(pivot, item, False)
44             if leftSide == 2:
45                 middle.append(item)
46             elif leftSide == 1:
47                 left.append(item)
48             elif leftSide == 0:
49                 right.append(item)
50
51         return quickSortAlph(left, fileObjects)+middle+quickSortAlph(right, fileObjects)
52     else:
53         return myList
54
55
56 cpdef list quickSortSize(list fileObjects):
57     cdef list left = []
58     cdef list right = [] #Make seperate l+r lists, and add on at the end.
59     cdef list middle = []
60     cdef int pivotSize
61     if len(fileObjects) > 1:
62         pivot = fileObjects[int(len(fileObjects)/2)]
63         if pivot.rawSize == "-":
64             pivotSize = 0
65         else:
66             pivotSize = pivot.rawSize
67

```

```

68         for i in fileObjects:
69             if i.rawSize == "-":
70                 left.append(i)
71             elif i.rawSize < pivotSize:
72                 left.append(i)
73             elif i.rawSize > pivotSize:
74                 right.append(i)
75             else:
76                 middle.append(i)
77         return quickSortSize(left)+middle+quickSortSize(right)
78     else:
79         return fileObjects
80
81 cpdef list quickSortTuples(list tuples): #Quick sorts tuples (for search results).
82     cdef list left = []
83     cdef list right = [] #Make seperate l+r lists, and add on at the end.
84     cdef list middle = []
85     cdef int pivot
86     if len(tuples) > 1:
87         pivot = tuples[int(len(tuples)/2)][0]
88         for i in tuples:
89             if i[0] < pivot:
90                 left.append(i)
91             elif i[0] > pivot:
92                 right.append(i)
93             else:
94                 middle.append(i)
95         return quickSortTuples(left)+middle+quickSortTuples(right)
96     else:
97         return tuples

```

The way Cython works, is that functions defined using `cpdef` are accessible by both Cython and Python, while variables can be defined using `cdef` internally, as they only need to be accessible via Cython.

Cython speeds Python code up significantly, depending on how many variables have a declared variable type. If variables / functions are used a lot, then it is a good idea to declare their type. Variables that are not used so often do not need to be defined with their data type, as they may only be used a couple of times during the program.

When you build the Cython program, you get a shared object file (.so), and a C file. You can import the name of the .c file in Python to use the module.

`quickSortTuples` is used for sorting search results, as search results are collected along with the position that the search item was found in the word. For example, if I searched for "b" in a folder, and there was a file called "brian.png", then the search result would be (0, "brian.png"). `quickSortTuples` then sorts these results by the number. I need to use a tuple so that I know what string belongs to which number.

The File class:

Here is the code for the File class (`code/python-go/fileClass.py`):

```

1  from os import path as osPath
2  from os import listdir
3  from subprocess import Popen, PIPE
4
5  import aesFName
6
7  class File:
8

```

```

9      def __init__(self, screen, hexPath, hexName, fileSep, extension=None, isDir=False, name=None,
10      path=None):
11          self.outerScreen = screen
12          self._totalSize = 0
13          self.hexPath, self.hexName, self.isDir, self.fileSep, self.extension = hexPath, hexName, isDir,
14      fileSep, extension
15          self.thumbDir = ""
16          self.checkSum = None
17          self.rawSize = self._getFileSize()
18          self.size = self.outerScreen.getGoodUnit(self.rawSize)
19          self.isDir = isDir
20          if path == None:
21              self.path = self._getNormDir(self.hexPath)
22          else:
23              self.path = path
24          if name == None:
25              self.name = aesFName.decryptFileName(self.outerScreen.key, self.hexName)
26          else:
27              self.name = name
28
29          if extension == None:
30              extension = self.path.split(".")
31              self.extension = extension[len(extension)-1].lower()
32
33          if self.isDir:
34              self.hexPath += self.fileSep
35              self.path += self.fileSep
36
37      def _getNormDir(self, hexDir):          # Private functions as they are usually only needed once and
38      should only be callable from within the class
39          hexDir = (hexDir.replace(self.outerScreen.path, "")).split(self.fileSep)
40          for i in range(len(hexDir)):
41              hexDir[i] = aesFName.decryptFileName(self.outerScreen.key, hexDir[i])
42
43          return self.fileSep.join(hexDir)
44
45      def _getFileSize(self, recurse=True):
46          if self.isDir:
47              if recurse:
48                  self._totalSize = 0
49                  self._recursiveSize(self.hexPath)
50                  size = self._totalSize
51                  return size
52              else:
53                  return " -"
54          else:
55              try:
56                  size = osPath.getsize(self.hexPath) # Imported from os module
57                  return size
58              except Exception as e:
59                  print(e, "couldn't get size.")
60                  return " -"
61
62      def _recursiveSize(self, f, encrypt=False): #Get size of folders.
63          fs = listdir(f)
64          for item in fs:
65              if encrypt:
66                  item = aesFName.encryptFileName(self.key, item)
67              if osPath.isdir(f+self.fileSep+item):
68                  try:
69                      self._recursiveSize(f+self.fileSep+item)
70                  except OSError:
71                      pass
72              else:
73                  try:
74                      self._totalSize += osPath.getsize(f+self.fileSep+item)
75                  except PermissionError: #Thrown when the file is owned by another user/administrator.
76                      pass

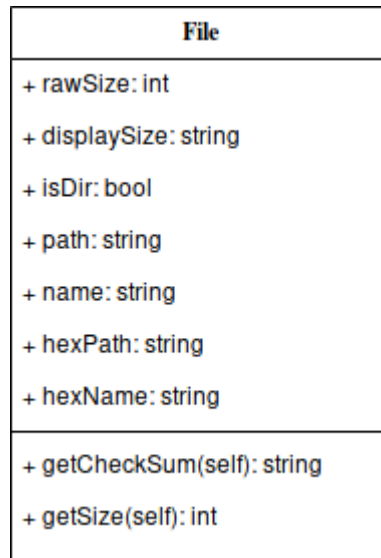
```

```

75
76     def getChecksum(self, new=True):
77         if self.checkSum == None or new:
78             goproc = Popen(self.outerScreen.startDir+"BLAKE", stdin=PIPE, stdout=PIPE)
79             out, err = goproc.communicate((self.hexPath).encode())
80             if err != None:
81                 raise ValueError(err)
82
83             self.checkSum = out.decode()
84
85         return self.checkSum

```

This is the File class talked about in the **File Storage** section of the design. As a recap, here is the class diagram I made for this class:



Most of the variables have been kept the same, however `extension` was added for when I get the thumbnail of the file, as if the file is not a png or a jpg, then a thumbnail cannot be shown (since it isn't an image). I also have the variable `outerScreen` that holds a reference to the Kivy Screen object that created it, so it can access functions and variables from the Screen if it needs to.

There are a few new functions too. `_getNormDir` gets the normal file path if path is `None` (it is also a private function (`_`), as it is only needed once by the object, and shouldn't be used again by anything else). `_getFileSize` gets the total size of the File object. If it is a folder (`isDir`) then `_recursiveSize` is called to handle it. If the size can not be read, then the function returns "-", which will display nicely in the GUI.