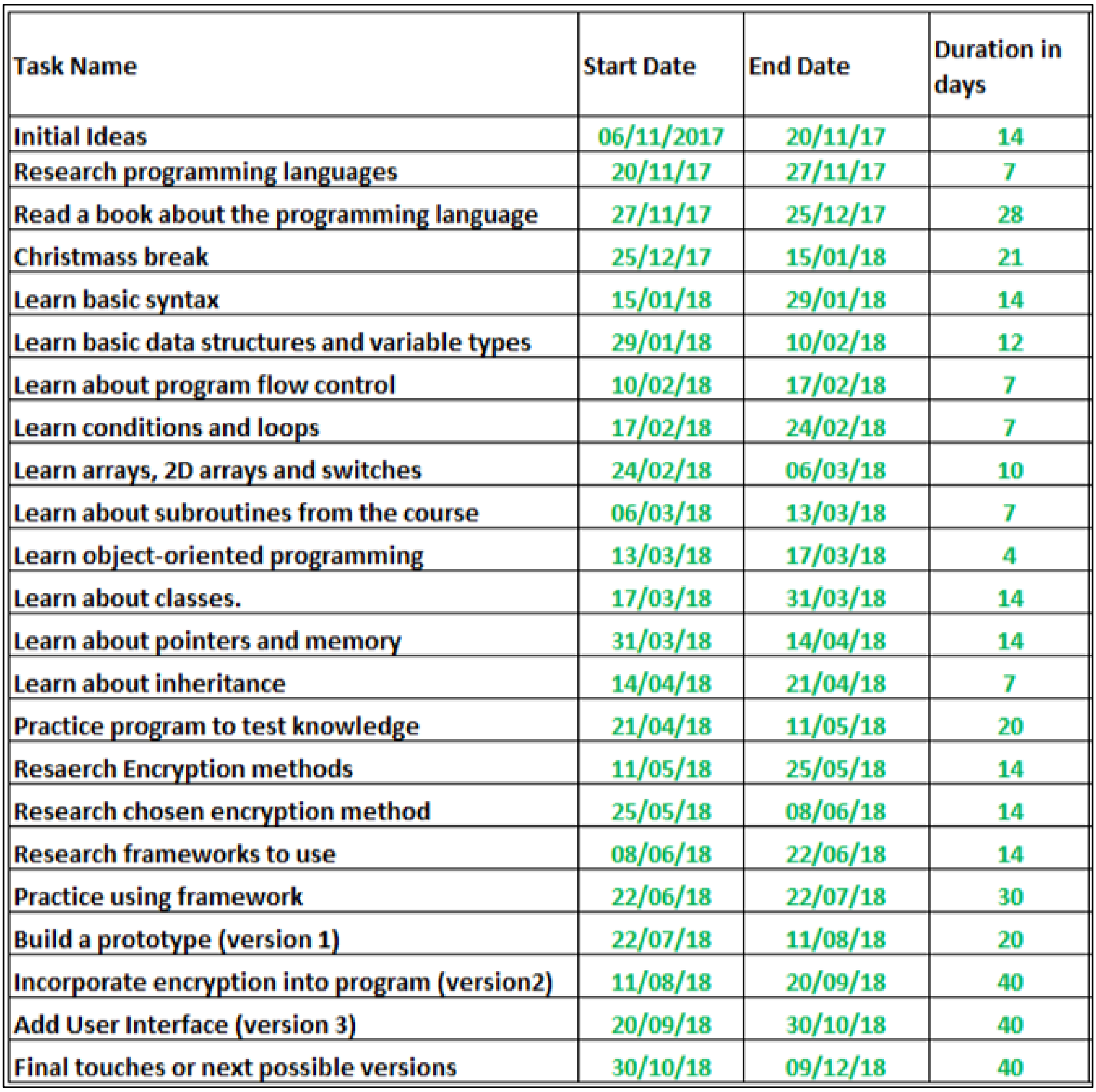
Intro

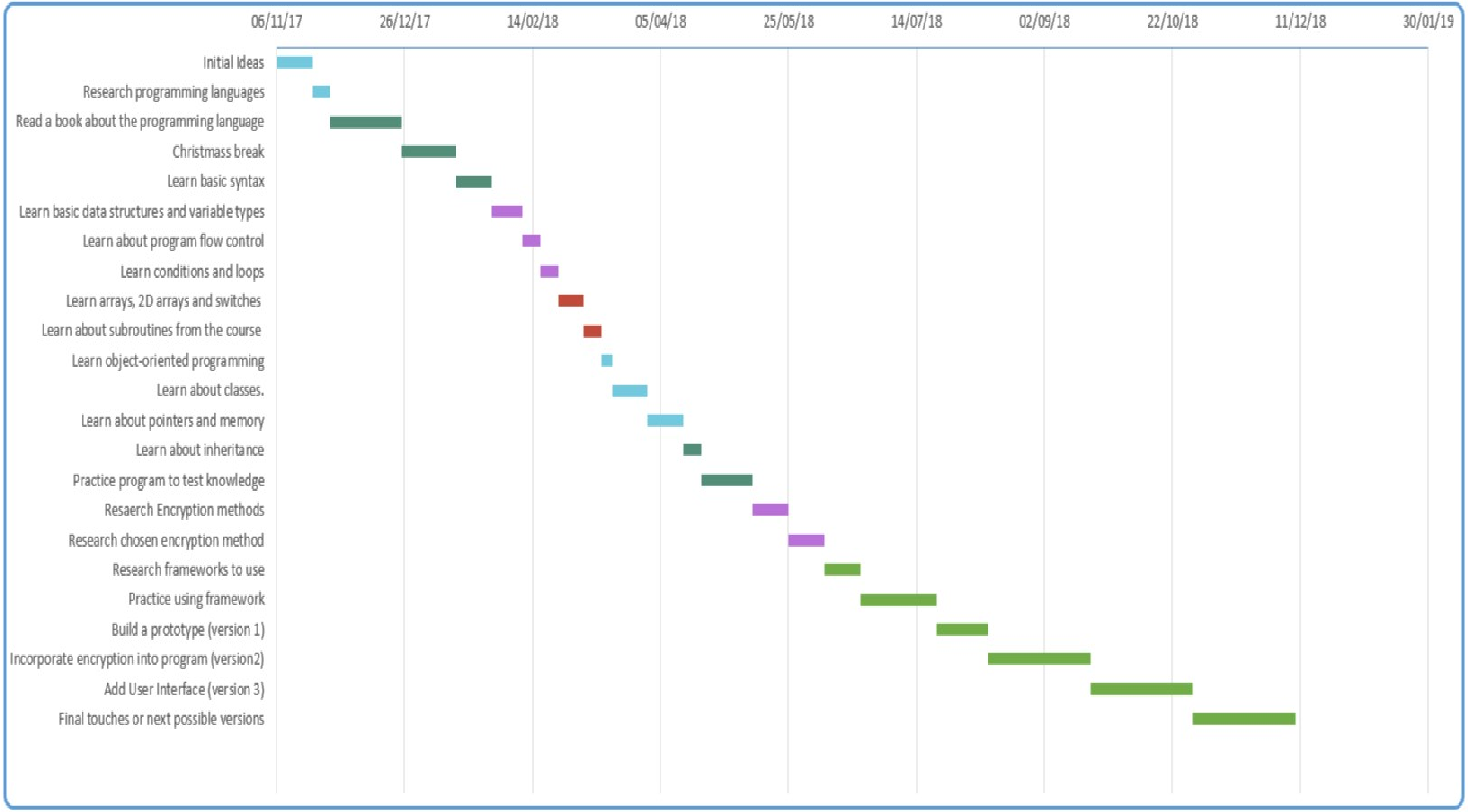
For my Extended Project Qualification, I decided to explore security and encryption and build a program that can encrypt and decrypt text files. The first part of my research was spent researching which programming languages would benefit me the most and any frameworks that were useful to my program. The second part of my research was exploring encryption methods and which method I could use to implement into my program. I managed to run into some issues along the way which I learnt to overcome and work around. I believe this project has enriched my skills and abilities in this field.

Planning:



Task Name	Start Date	End Date	Duration in days
Initial Ideas	06/11/2017	20/11/17	14
Research programming languages	20/11/17	27/11/17	7
Read a book about the programming language	27/11/17	25/12/17	28
Christmass break	25/12/17	15/01/18	21
Learn basic syntax	15/01/18	29/01/18	14
Learn basic data structures and variable types	29/01/18	10/02/18	12
Learn about program flow control	10/02/18	17/02/18	7
Learn conditions and loops	17/02/18	24/02/18	7
Learn arrays, 2D arrays and switches	24/02/18	06/03/18	10
Learn about subroutines from the course	06/03/18	13/03/18	7
Learn object-oriented programming	13/03/18	17/03/18	4
Learn about classes.	17/03/18	31/03/18	14
Learn about pointers and memory	31/03/18	14/04/18	14
Learn about inheritance	14/04/18	21/04/18	7
Practice program to test knowledge	21/04/18	11/05/18	20
Resaerch Encryption methods	11/05/18	25/05/18	14
Research chosen encryption method	25/05/18	08/06/18	14
Research frameworks to use	08/06/18	22/06/18	14
Practice using framework	22/06/18	22/07/18	30
Build a prototype (version 1)	22/07/18	11/08/18	20
Incorporate encryption into program (version2)	11/08/18	20/09/18	40
Add User Interface (version 3)	20/09/18	30/10/18	40
Final touches or next possible versions	30/10/18	09/12/18	40

Gantt Chart


06/11/17
26/12/17
14/02/18
05/04/18
25/05/18
14/07/18
02/09/18
22/10/18
11/12/18
30/01/19
Initial Ideas
Research programming languages
Read a book about the programming language
Christmass break
Learn basic syntax
Learn basic data structures and variable types
Learn about program flow control
Learn conditions and loops
Learn arrays, 2D arrays and switches
Learn about subroutines from the course
Learn object-oriented programming
Learn about classes.
Learn about pointers and memory
Learn about inheritance
Practice program to test knowledge
Resaerch Encryption methods
Research chosen encryption method
Research frameworks to use
Practice using framework
Build a prototype (version 1)
Incorporate encryption into program (version2)
Add User Interface (version 3)
Final touches or next possible versions

Practical research

For the first part of my research, I began researching programming languages and which best suit my needs and are easy to learn. I ultimately broke down my programming language choices down to C++ and Java as these were the two most suitable programming languages. I considered other languages like Python and a web-based program written in PHP. I had to make a decision as to which language to pursue, whether it would be Java or C++ as I had no experience in either of them. Both languages have their benefits and disadvantages and I had to consider them before making a decision.

Java is a good language and is not harder than C++ in terms of the learning curve. Java runs on a virtual machine so the program, if written in Java, would easily be able to run on other platforms. However, since Java runs on a virtual machine, it is slightly slower than C++. During my research on programming languages, I learnt that Java is slower at accessing the memory of a computer than C++ is. Accessing the memory is very important to my program as the program reads a text file from memory and needs to quickly access it. Java is also very hard to make a GUI for, meaning that it does not have many frameworks that provide a graphical user interface which I need for my program.

These facts about Java made C++ a better programming language for me. C++ has some disadvantages such as its cross-platform capabilities as all C++ frameworks are OS based and need to be reconfigured for every other platform. C++ is also faster than Java in terms of accessing the memory of a computer which means that the program can work through a text file faster and encrypt every letter more quickly than Java. C++ also provided many useful frameworks that I could try and experiment with to better help me learn the language. When I was set on the programming language I was going to use for my project, I began researching and learning the language.

I started learning the basic syntax of the language in a console environment from YouTube channels. These were reputable sources for the very basics of the language but after some time I moved to more advanced sources. I picked up two books called “Jumping into C++” by Alex Allain and “Accelerated C++” by Andrew Koenig. I used Jumping into C++ to learn some advanced syntax and Accelerated C++ to practice using what I’ve learnt in challenges from the book. I found myself having a short attention span towards books, so I migrated my learning over to online courses offered from the website Udemy. I found these courses to be engaging and fun to follow.

The course and other informational websites allowed me to build up my knowledge on the syntax and move on to more complicated topics. I learnt about data structure and variable types and fully understood how these variables are stored and how they are accessed which helped me extensively when it came to making my encryption program. I wrote and experimented with many programs along the way to further solidified what I had learnt so far. I learnt a lot about the language and was ready to experiment with frameworks and interfaces.

I initially moved on from a Command line Interface towards a user interface using the Microsoft Visual C++ Forms which allowed me to use objects and forms to create a Graphical User Interface. This was good practice but ultimately, I found that it was lacking a lot of functionality I needed for my program such as handling files. Another big issue that I had with Visual C++ was how it linked all the used libraries. When the program was compiled into an executable file, it would run perfectly fine on my system but if I were to distribute it to a machine which was lacking Microsoft .Net Framework, the program would fail to run. This made me research into Linkers and how to make sure my program can be distributed.

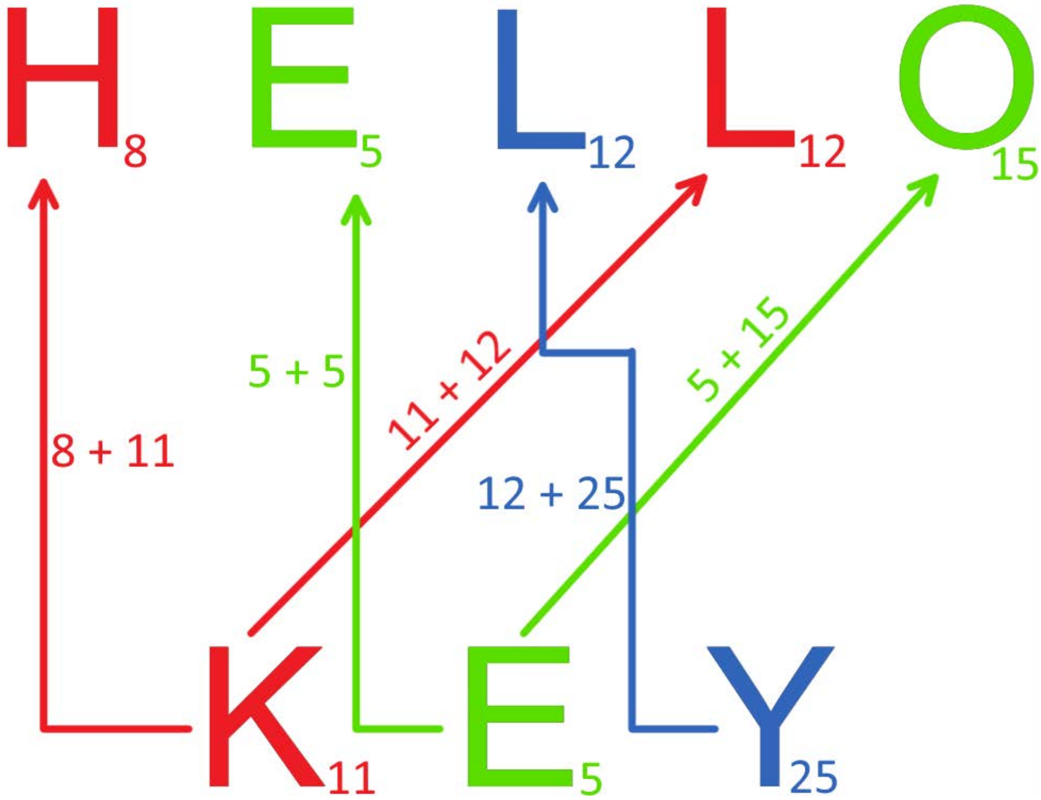
After experimenting with Visual C++, I researched for alternatives. I found a framework that granted me all the functionality I needed and allowed to me create a robust Graphical User Interface. I began learning and experimenting with the QT framework and with the help of another Udemy Course and mostly self-teaching, I was able to make myself comfortable with the framework. At this point, I had learnt everything that I needed from the language and was ready to work on the chosen framework.

Theory research

For my theory research, I set out to research encryption algorithms and how encryption is supposed to work. I researched multiple methods of encryption but the two I mostly focused on were RSA (Rivest–Shamir–Adleman) Cryptosystem and the Caesar Cipher Algorithm. Since my program will be encrypting text files, I decided these two algorithms were suitable options.

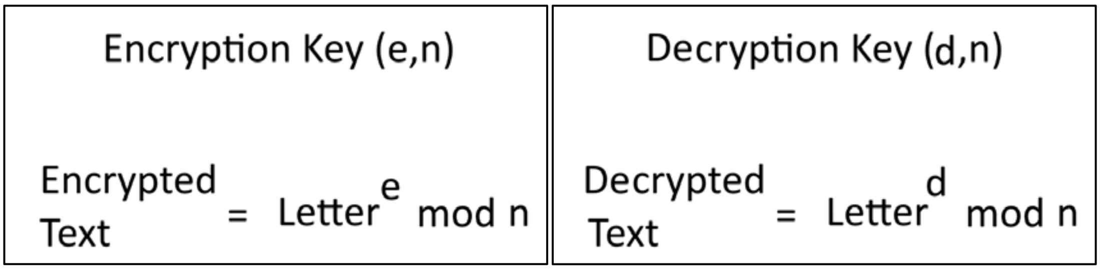
Caesar Cipher

As for Caesar Cipher, I decided not to just shift every letter by a certain number of characters, but to implement a more complicated method that I thought of. Instead of using a number key to shift the letters, I used a word as key and the letters would be shifted using a corresponding value for each letter in the word. To allocate each character on a keyboard a number, I used the ASCII table which is integrated into C++ and is easy to convert to. I decided that I would not have a built-in key in the program and that I would make the user able to generate a random key.



RSA

I also researched RSA encryption. RSA is much more secure and complicated than Caesar Cipher. RSA uses prime numbers which make it very complicated to reverse engineer and crack. To use RSA Cryptosystem, two different keys are needed, one encryption key and one decryption key. The process of encrypting a letter involved converting a letter into a corresponding number, then raising the number to the first number of the encryption key (e) and then applying modulus of (n) into the result. The decryption process is the same but the number, e and n are replaced with the decryption key, d and n.



Encryption Key (e,n)

$$\text{Encrypted Text} = \text{Letter}^{e} \bmod n$$

Decryption Key (d,n)

$$\text{Decrypted Text} = \text{Letter}^{d} \bmod n$$

Through my research, I learnt how to calculate the keys which have mathematical rules they must follow. I calculated my own encryption and decryption keys to use in my program. To calculate the keys, you first begin by choosing two prime number, the larger the prime numbers, the more difficult it is for the encryption to be broken through brute force. I chose the number 131 and 151 which are both prime numbers. The next step is to calculate the value (n) which is the product of the two selected prime numbers. Therefore, n = 131 X 151 = 19781. Another value that is needed to calculate is Phi n (Φn) which is calculated using (131-1) X (151-1) which gives us a value of Φn = 19500. To choose the encryption key, e, it must follow some rules. The value e must be between 1 and Φn and it must be co-prime with n. I wrote a python program which went through every number between 1 and 19500 and tested whether it was a co-prime with n. I acquired a series of numbers that fit these criteria and I chose the number 3451 as e. So far, I have calculated the encryption key (3451, 19781). The decryption key value, d must also follow a rule. The rule is that (d X e) mod Φn = 1. I wrote another python program which tested multiples of 3451 and checked if the modulus of that number and Φn is equal to 1. A series of numbers met these criteria and I chose the smallest value which was 59551. At this stage, I had my two encryption and decryption keys: **Encryption (3451,19781)** and **Decryption (59551,19781)**

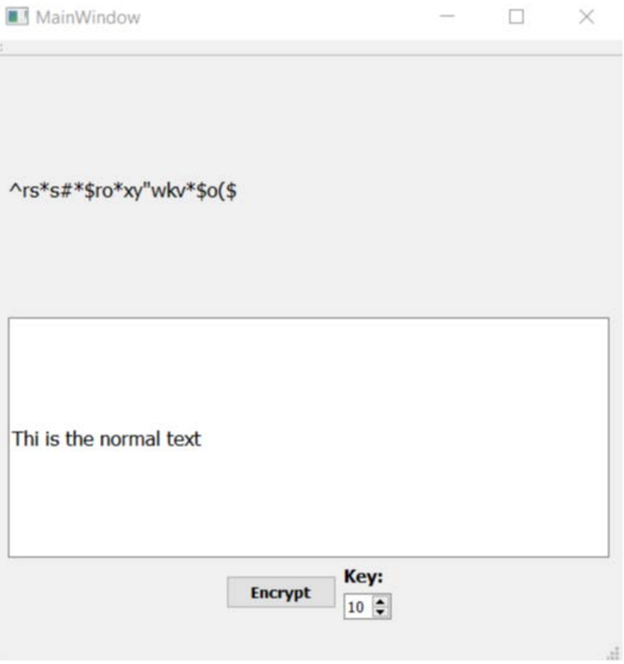
I was complete with my research and happy to start my artefact, having a clear plan as to what I was achieving to do which was create an encryption program in C++ with a friendly Graphical User Interface that was able to encrypt and decrypt text both using RSA and Caesar Cipher.

Making Process

I began coding my program in the QT Development Environment and started programming it step by step. I did not start implementing every feature at once as that would be overwhelming to code and is tedious to test and debug if there are any problems with it.

Version 1:

For my first version, I wanted to have a working program that could encrypt text in real time using Caesar Cipher. For this version I designed a basic Graphical User Interface that would allow the user to enter their text at the bottom box, choose a desired key and then have that plain text encrypted and displayed on the area above the box. This version I only used Caesar Cipher and the Caesar Cipher key was not a word but only a number. It was only a prototype to see a working program that encrypts data and can handle any character on the keyboard. The program also was able to be easily distributed with all the required files being baked into the installer that I created for the program.



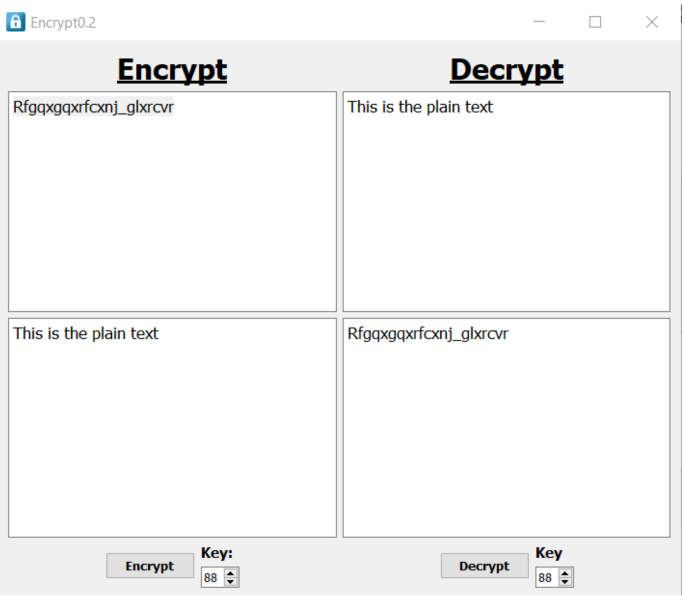
```
^rs*s#*$ro*xy"wkv*$o($
```

```
Thi is the normal text
```

Key: 10

Version 2:

For my second version, I decided to implement decryption of text using Caesar Cipher. The program looked very similar to the first version except it being split into two sections where the user can both encrypt and decrypt text at real-time instantaneously. During my time coding this version, I ran into many errors and bugs which helped me better improve my skills at identifying and diagnosing problems. It also showed me that plans don't always work perfectly, and it is important to adapt such in the case of my encryption function which I had previously written outside of QT that I thought I could easily implement into this program. However, this code needed a lot of adaptation to work with the QT framework as many variables were different classes in QT.



```
Rfgqxgqxrfcxnj_glxrcvr
```

```
This is the plain text
```

```
This is the plain text
```

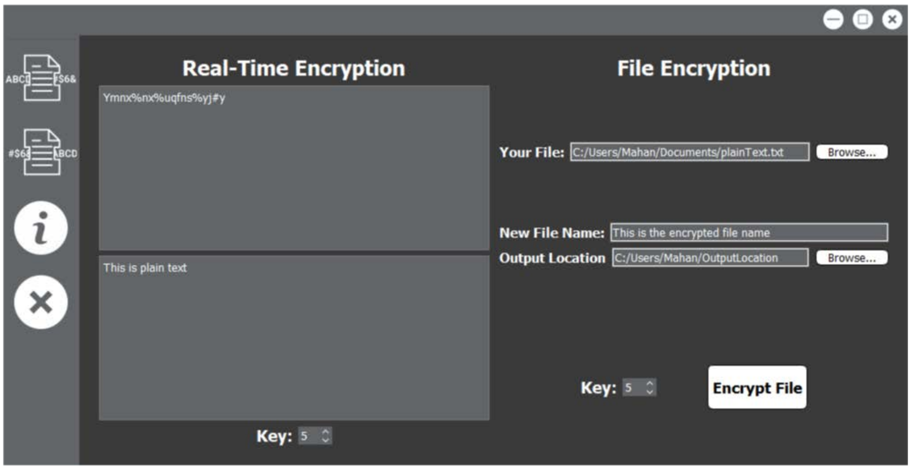
```
Rfgqxgqxrfcxnj_glxrcvr
```

Key: 88

Key 88

Version 3:

Version 3 of my program was a big jump for which I decided to start with the Graphical User Interface, designing and coding a seamless interface that the user would navigate in with ease. I designed my own interface as shown below which adopts a black, grey and white colour palette which in my opinion gives a professional look to the program. All icons used in this program were either designed by me or were from royalty-free sources. The program has two "layers" which are the encryption and decryption page. On each "layer" the user has the choice of Real-Time Encryption or File Encryption. The user can access these two pages by using the buttons on the side which come with self-explanatory icons.



Ymnx%nx%uqfns%yj#y

This is plain text

Key: 5

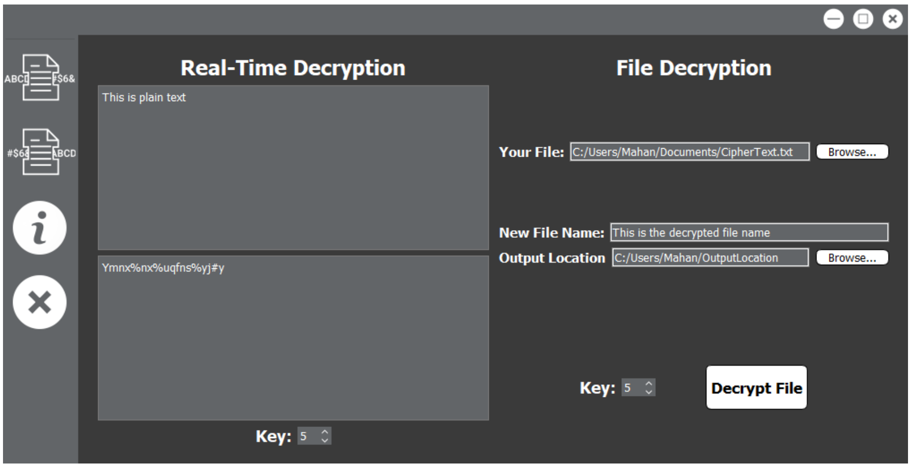
Your File: C:/Users/Mahan/Documents/plainText.txt

New File Name: This is the encrypted file name

Output Location C:/Users/Mahan/OutputLocation

Key: 5

The real-time encryption encrypts the input of the user as the user types it so there is no delay and if the user decides to change the key after they have typed their text, the program will update the cypher text based on the new key. File encryption allows the user to easily browse for a .txt text file from their computer, choose their encrypted file name and location and encrypt it using their desired key. The program has been optimised to minimise any loading while the program encrypts/decrypts and outputs the file.



This is plain text

Ymnx%nx%uqfns%yj#y

Key: 5

Your File: C:/Users/Mahan/Documents/CipherText.txt

New File Name: This is the decrypted file name

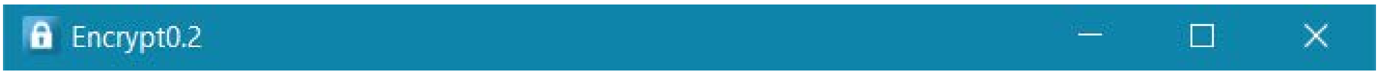
Output Location C:/Users/Mahan/OutputLocation

Key: 5

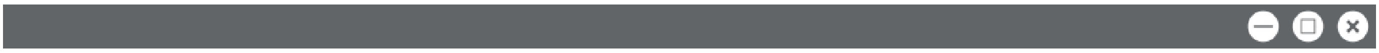
A flaw with this version is that the user must remember the key they had used. I addressed this issue in the next version alongside adding RSA encryption.

I was initially unhappy with the default toolbar that Windows provides. It did not fit the style of my program, therefore, I set out to program in my own toolbar. Adding a toolbar with the buttons was not a hard task however, replicating the ability to move the program window by clicking and dragging the toolbar was challenging. I learnt a new skill of working with vectors and locations to be able to create a function which allowed the window to be moved when clicked and dragged. The function calculates the difference between vector positions of the mouse before the movement and after the movement and applies that difference to the window, replicating movement alongside the mouse.

Before:

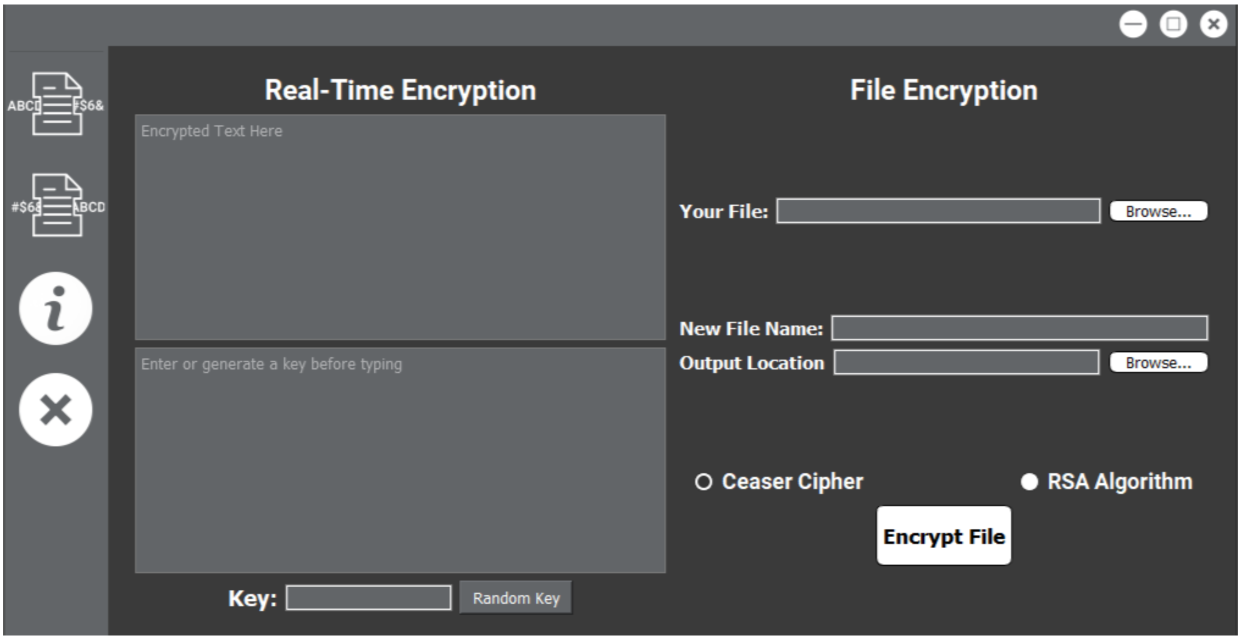


After:



Version 4 (Final Build):

At this point into the project I was happy with the design and I wanted to implement the final encryption algorithms and the final changes that were needed to be made. I decided to change the Caesar Cipher encryption algorithm from using a number key to using my implementation of word key. I also wanted to add RSA encryption and decryption option when encrypting or decrypting a file. The final design of the program was going to be as such:



○ Ceaser Cipher ● RSA Algorithm

It is very similar to the previous version except for some back-end tweaks. I have implemented my implementation of the Caesar Cipher key in form of a word which can be randomly generated by clicking the “Random Key” button or if it is left blank when the user types in their message, the program will automatically generate a random key, only if the field is empty. For decryption, the user is not allowed to type in their cypher text until they have entered a key. An error message shows if the user tries to enter their cypher text before entering the key. They key itself is generated using randomising algorithms and gives back an ASCII value which can be corresponded to a character. This way the key is a word and not just a single number, therefore, the algorithm is more secure.

To fix the issue of the previous version, I implemented a method of hiding the key in the cypher text of the encrypted file. The process works by first adding the length of the key at the beginning of the cypher text, then add the cypher text itself after that and finally add the key at the end. This way, the program will always know where the key is and how long it is to easily extract the key while decrypting. While decrypting, the same process is reversed and only the original plain text is stored in the file and not the key. The length of the key added to the beginning of the cypher text is in the form of a character, so it is not a number being stored, making it difficult to deduce that it is the key length. Neither the key or the key length stored in the cypher text are obvious and easy to find without the knowledge of the algorithm.

After being happy with the Caesar Cipher encryption working properly, I began to implement my RSA algorithm into the program. My code was adopted easily to work with the QT framework and the program ran without a problem however, I noticed that the encryption was incorrect. After investigating and debugging I found no bugs in the code. After conducting my own research on the problem, I was able to find the problem with the algorithm and why the encryption was incorrectly done.

Why RSA did not work:

RSA Algorithm deals with calculation of very large numbers since you must do Letter^{e} mod n. This becomes a problem when the value of e and n are very large. For example, an RSA operation would be encrypting the letter “h” using my RSA keys. “h” = 104 so to encrypt we must do 104^{3451} mod 19781. This entire operation results in a small value close to 104 but the operation produces very large numbers along the way and even more so when decrypting using the key 59551.

The QT framework and C++ have many data types for storing a value. These include int, double and float. These variables also have “unsigned” and “long” variants. Unsigned allows for only positive integers to be stored therefore increasing the range of positive numbers storable. Long allocates more memory to the variable allowing bigger numbers to be stored. The largest storable value in C++ and QT is just under $2x10^{19}$. If we were to encrypt the letter “h” we would have to raise the ASCII value 104 to the power of 3451 which gives a value of $6x10^{6960}$ which is an extremely large amount of times larger than the maximum storable value. This meant that I was unable to perform RSA calculations in C++ without using a special class that allows for very large numbers to be stored.

I began researching custom classes made by others that would allow me to use a data type to store and perform calculations on these very large numbers. I found several classes that allowed me to do this but unfortunately, none of them would allow me to perform both power and modulus on these numbers, therefore, none of these classes were suitable for my application.

I also tried to calculate much smaller encryption and decryption keys so that the calculations would not result in very large numbers but the pattern I encountered was that as I managed to reduce the size of the keys “e” and “n”, the key “d” would get bigger and bigger as “e” got smaller. This meant that I was also unable to change my keys to fit the size constraints of the language.

Unfortunately, all these conclusions meant that without my own custom-made variable class and the custom operations that I needed (power and modulus) available to that class, I would not be unable to incorporate RSA encryption into my program.

This resulted in my program only being able to encrypt using my Caesar Cipher algorithm which is to some extent disappointing but overall, I was able to learn more about how these systems store data and expand my knowledge on the language and encryption.

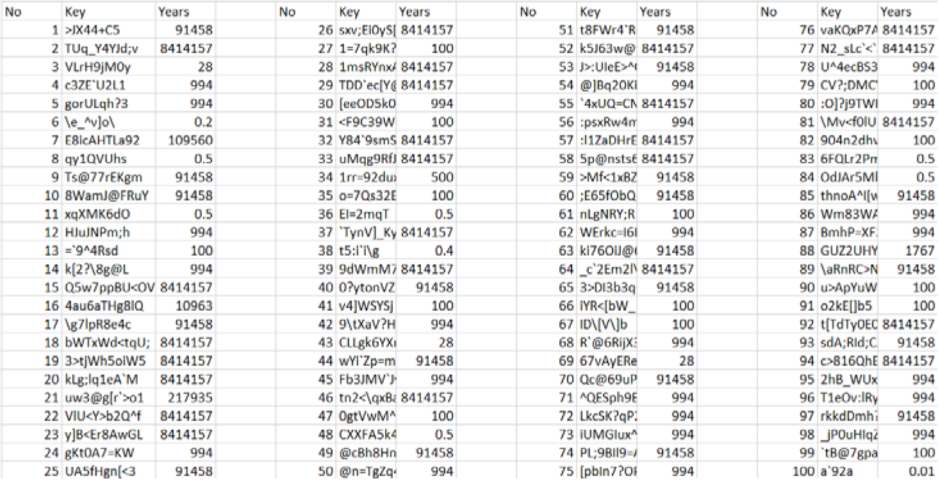
How strong is my implemented encryption algorithm?

To measure how strong an algorithm is, we must see how hard the keys produced are to crack. If we were to imagine an unauthorized person trying to crack the encryption key to gain access to the file, they could use brute force. Brute force is a method in which every single possible combination of a password or a key is tested until the correct combination is found. By calculating how many possible combinations my keys could have and by estimating how many keys can be tested per second, we can calculate an estimate as to how secure the keys are. My keys used in my program are randomly generated between the lengths of 4 to 12 characters. Each character is chosen randomly from 90 possible values from the ASCII table. Therefore, each letter can have 90 combinations. This means that to crack a key of length 12 characters, 90^{12} combinations need to be tested (near to $3x10^{23}$ combinations).

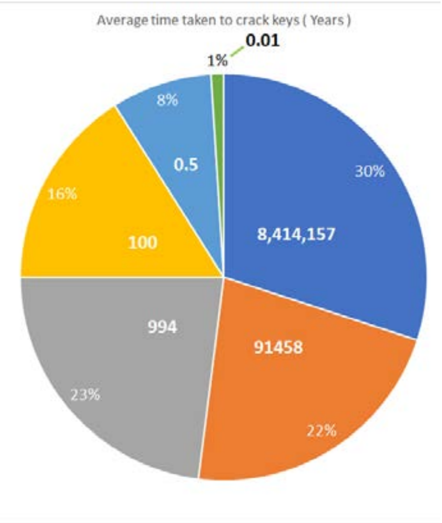
According to research from “BetterBuys.com”*, the estimated number of keys testable in one second in 2018 is around 15 million keys per second. This means that it would take $2x10^{16}$ ($3x10^{23}$/15000000) seconds to test every single possible combination of my keys. This translates to $6x10^{8}$ years to crack one of my keys. This is an astronomically large amount of time, so this means that realistically, this key is uncrackable through brute force.

Real word results?

To test this for myself, I used the estimating cracking time program available on “BetterBuys.com”* to get an estimate of how long randomly generated keys from my program would take to be cracked. I took a sample of 100 keys randomly generated from my program, put into the websites estimator and recorded the given time in years it would take to crack that key. The graph below shows the result of this experiment. It shows the percentage of time a key was generated that had a certain average time to crack. This showed that more than 50% of the time a key is generated takes over 91,458 years to crack. These results show that in practice, these keys are just as reliable and secure as thought to be in theory.



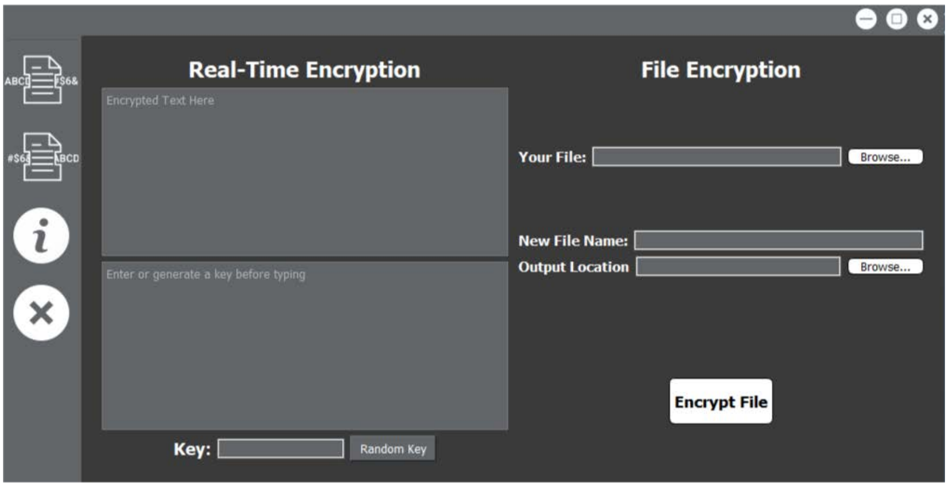
No	Key	Years	No	Key	Years	No	Key	Years	No	Key	Years
1	>JX44+C5	91458	26	sxv;El0yS[	8414157	51	t8FWr4`R	91458	76	vaKQxP7A	8414157
2	TUq_Y4YJd;v	8414157	27	1=7qk9K?	100	52	k5J63w@	8414157	77	N2_sLc`<`	8414157
3	VLrH9jM0y	28	28	1msRYnx/	8414157	53	J>:UleE>^	91458	78	U^4ecBS3	994
4	c3ZE`U2L1	994	29	TDD`ec[Y(	8414157	54	@]Bq20Kl	994	79	CV?;DMC	100
5	gorULqh?3	994	30	[eeOD5k0	994	55	`4xUQ=CN	8414157	80	:O]?j9TWI	994
6	\e_^v]o\	0.2	31	<F9C39W	100	56	:psxRw4n	994	81	\Mv<f0lU	8414157
7	E8icAHTLa92	109560	32	Y84`9smS	8414157	57	:I1ZaDHrE	8414157	82	904n2dhv	100
8	qy1QVUhs	0.5	33	uMqg9RfJ	8414157	58	5p@nsts6	8414157	83	6FQLr2Pn	0.5
9	Ts@77rEKgm	91458	34	1rr=92du	500	59	>Mf<1xBZ	91458	84	OdJAr5Ml	0.5
10	8WamJ@FRuY	91458	35	o=7Qs32E	100	60	;E65fObQ	91458	85	thnoA^l[v	91458
11	xqXMK6dO	0.5	36	El=2mqT	0.5	61	nLgNRY;R	100	86	Wm83WA	994
12	HJuJNPm;h	994	37	`TynV]_Ky	8414157	62	WErkc=l6l	994	87	BmhP=XF.	994
13	=`9^4Rsd	100	38	t5:i`l\g	0.4	63	kl76OiJ@(	91458	88	GUZ2UHY	1767
14	k[2?\8g@L	994	39	9dWmM7	8414157	64	_c`2Em2l\	8414157	89	\aRnRC>N	91458
15	Q5w7ppBU<OV	8414157	40	0?ytonVZ	91458	65	3>DI3b3q	91458	90	u>ApYuW	100
16	4au6aTHg8IQ	10963	41	v4]WSYSj	100	66	iYR<[bW_	100	91	o2kE[]b5	100
17	\g7lpR8e4c	91458	42	9\tXaV?H	994	67	ID\[V\]b	100	92	t[TdTy0EC	8414157
18	bWTxWd<tqU;	8414157	43	CLLgk6YX	28	68	R`@6RijX:	994	93	sdA;RId;C	91458
19	3>tjWh5olW5	8414157	44	wYl`Zp=m	91458	69	67vAyERe	28	94	c>816QhE	8414157
20	kLg;lq1eA`M	8414157	45	Fb3JMV`J	994	70	Qc@69uP	91458	95	2hB_WUx	994
21	uw3@g[r`>o1	217935	46	tn2<\qxB	8414157	71	^QESph9E	994	96	T1eOv:lRy	994
22	VIU<Y>b2Q^f	8414157	47	0gtVwM^	100	72	LkcSK?qP:	994	97	rkkdDmh;	91458
23	y]B<Er8AwGL	8414157	48	CXXFA5k4	0.5	73	iUMGlux^	994	98	_jP0uHlqZ	994
24	gKt0A7=KW	994	49	@cBh8Hn	91458	74	PL;9BII9=/	91458	99	`tB@7gpa	100
25	UA5fHgn[<3	91458	50	@n=TgZq	994	75	[pbIn7?OI	994	100	a`92a	0.01

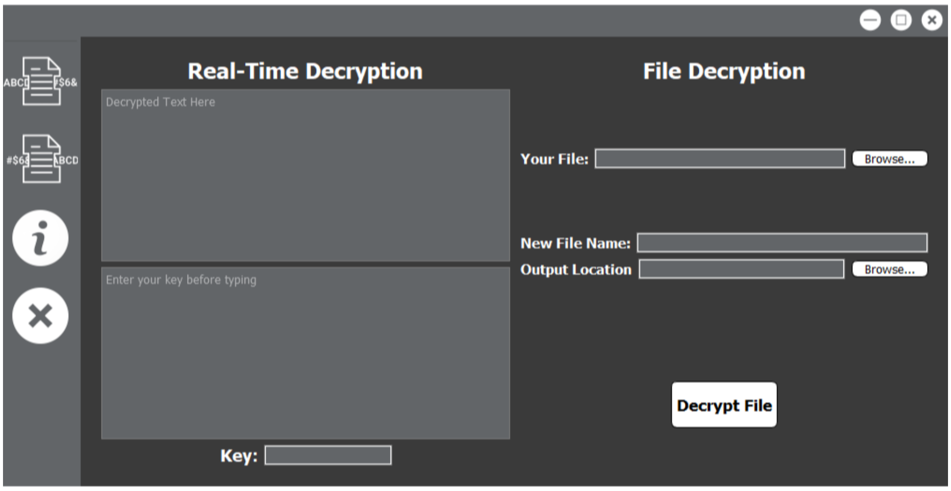


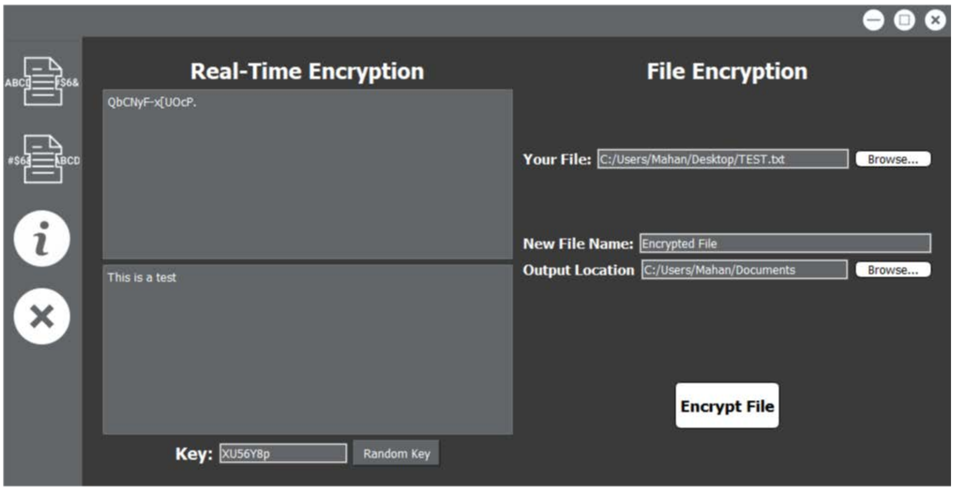
Conclusion:

In conclusion, this project allowed me to learn a brand-new language and research on a field that matters to me. Security is very important to our day to day life so getting to explore and dive into this field by making a product was very enjoyable. I learnt many skills ranging from time management to practical coding skills I picked up. I am very happy with the outcome of my program. It is a successful product that fulfils its purpose of encrypting and decrypting text files. It also looks professional and works efficiently.

*https://www.betterbuys.com/estimating-password-cracking-times/







QbCNyF-x[UOcP.

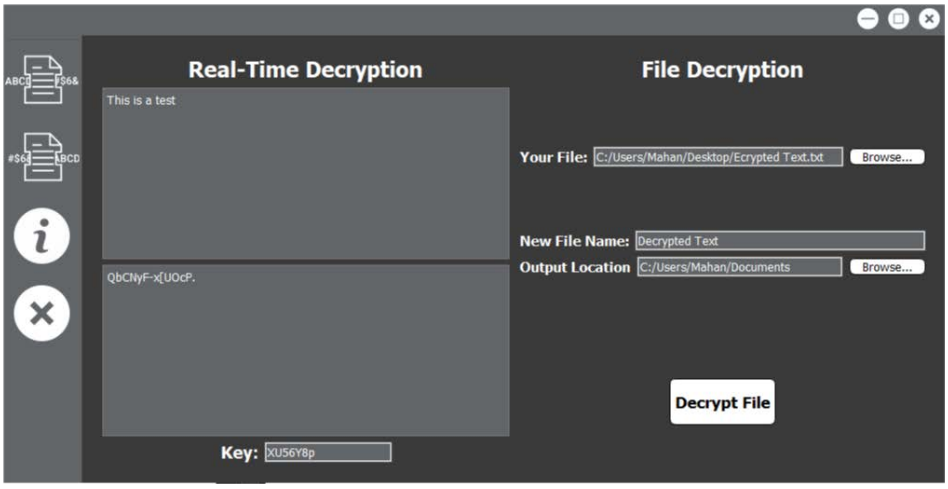
This is a test

Key: XU56Y8p

Your File: C:/Users/Mahan/Desktop/TEST.txt

New File Name: Encrypted File

Output Location C:/Users/Mahan/Documents



This is a test

QbCNyF-x[UOcP.

Key: XU56Y8p

Your File: C:/Users/Mahan/Desktop/Ecrypted Text.txt

New File Name: Decrypted Text

Output Location C:/Users/Mahan/Documents