Kavian Shirkoohi

South Essex College of Further and Higher Education

Abstract

A simplified block cipher simulation, demonstrated through fully documented code and interrupts, where the user can step through the algorithm to observe the individual processes required to successfully encrypt data.

h446 programming project

An introduction to block ciphers

Synopsis

This simulation demonstrates the processes of a typical block cipher, utilizing simplifications to substitute topics beyond A-Level Computer Science, Maths and Further Pure Maths specifications. The simulation will be developed into an executable program. At the beginning of the program, a GUI will provide a background introduction to modern ciphers. The relevance of block ciphers will be illustrated (i.e. the need to encrypt information using data matrices), along with their strengths and vulnerabilities. Once the user has been introduced to block ciphers, they will be able to run an adaptation of a simplification of the Rijndael cipher in a 128-bit, 192-bit or 256-bit key AES configuration (albeit, with a few abstractions for the A-Level context). They will be prompted to enter a fixed-length string message (simulating the plaintext), run the cipher algorithm to encrypt it, and view the encrypted data. They can then decrypt the message to reverse the process, where they will be able to appreciate the speed at which block ciphers can encrypt and decrypt data. Through each step of the cipher's algorithm, the user can view the code executed and can step through the execution using predefined breakpoints to inspect the variables and current execution process, learning step-by-step the process of the cipher.

Contents

[Synopsis 1](#_Toc480802210)

[Definition 4](#_Toc480802211)

[Analysis 5](#_Toc480802212)

[Analysis 7](#_Toc480802213)

[Conclusion 8](#_Toc480802214)

[Identifying limitations 8](#_Toc480802215)

[Software requirements 9](#_Toc480802216)

[Hardware requirements 9](#_Toc480802217)

[Design 12](#_Toc480802218)

[Problem decomposition 12](#_Toc480802219)

[Block ciphers explained 13](#_Toc480802220)

[The solution 14](#_Toc480802221)

[Primary algorithms and processes 17](#_Toc480802222)

[Conceptual Block Cipher no. 1 17](#_Toc480802223)

[Summary of the cipher algorithm 20](#_Toc480802224)

[Usability features 22](#_Toc480802225)

[Sketches of the user interface 22](#_Toc480802226)

[Key variables and data structures 26](#_Toc480802227)

[Testing and development approaches 28](#_Toc480802228)

[Development 30](#_Toc480802229)

[Primary testing methods used during development 30](#_Toc480802230)

[A brief description of the cipher routine 30](#_Toc480802231)

[The form system 31](#_Toc480802232)

[Development of the working cipher (Stage 1) 32](#_Toc480802233)

[Source code of the cipher 32](#_Toc480802234)

[Testing the cipher 45](#_Toc480802235)

[Development of the forms (Stage 2) 57](#_Toc480802236)

[Implementation of the cipher with the forms (Stage 3) 62](#_Toc480802237)

[Data validation and testing 73](#_Toc480802238)

[Evaluation 82](#_Toc480802239)

[Evaluation testing 82](#_Toc480802240)

[Final development-side testing 82](#_Toc480802241)

[Client testing and evaluation 88](#_Toc480802242)

[Outcome 90](#_Toc480802243)

[The final program 91](#_Toc480802244)

[Maintainability 91](#_Toc480802245)

[Taking the program further 91](#_Toc480802246)

[Citations 92](#_Toc480802247)

Definition

This project will focus on the development of an educational application introducing the user to block ciphers, within the context of the A-Level Computer Science and Mathematics syllabus. Block ciphers are an essential topic of cryptography; they are the most conventional form of encrypting enormous amounts of data efficiently and quickly within the computer systems we use today, whilst in some configurations remaining virtually unbreakable with the current technology available. Since block ciphers focus largely on data represented in matrix form, this subject area can provide a unique insight into how Mathematics and Computer Science topics can combine. Highlighting the interlinking between these subjects can demonstrate the importance of learning them in class, which could provide a source of motivation for students to conduct further research into the topic of data encryption. The importance of fluency in data security is essential for prospective Computer Science students in a world which is becoming increasingly reliant on data systems and information technology. This requirement has been expressed by the government’s decision to form the upcoming National College of Cyber Security in Bletchley Park [1].

After investigating the web for more information about block ciphers, it was discovered that most websites and learning resources are not well suited for the A-Level student. They contain a lot of content which is well above the scope of the subject syllabus. This presents an inconvenience the student would need to overcome to appreciate the intriguing topic of data encryption. The student would have to cross reference multiple sources, learning new topics from links to web pages that assume the reader has a background knowledge of Computer Science and Computational Mathematics at postgraduate level. Most of the information provided from these sources is well above the relevance of being used in the subject of block ciphers. Outside the internet, few publications exist which are of a narrative or abstraction level relevant to the A-Level student which would be of any use. (There are several lengthy scientific papers published about block ciphers, such as the Rijndael document who's intended audience would be U.S. Government judges for the Advanced Encryption Standard competition, none of which would contain simplifications of the topic for synoptic comprehension.)

My project solution to this problem is simple – to provide an executable application with the preliminary information required to grasp the concept of this essential topic of modern cryptography and their relevance to the fields of the A-Level subjects studied by the client student, whilst presenting a unique opportunity to observe an interactive simulation of a real block cipher application operating on data that the user has input.

The program will consist of an introduction to block ciphers containing the preliminary knowledge of how they work and the concepts needed to understand them, along with a running simulation of a block cipher which will contain components based on the AES. Within the simulation, the user can pick a key size which will determine the length of the key, and how many rounds the cipher will use to encrypt the data. The user will be prompted to enter their plaintext data, a maximum of 128-bits in data (in other words, a maximum of 16 characters) into the system. They will then enter the key for the data (could be a password, for example) which will be used to encrypt it. The plaintext will be converted to ciphertext using the algorithm and the key, and can be stored to a spreadsheet. After the message is encrypted, the user can view it. They can then either encrypt a new message, view the spreadsheet containing the encrypted data or decrypt a message they have entered or one from the spreadsheet.

Analysis

Introduction

The topic of block ciphers can be complex. It demands preliminary knowledge and a thorough understanding of its processes. The approach to this demonstrating this topic can be improved by creating an interactive and easy-to-use program. Simplicity and relevance will be the key objectives when deciding which content to include for the user. The program will be an executable application that will consist of several key components. The user will be introduced to block ciphers at the beginning of the program, where they will then be able to run a simulation of a block cipher, based on the Advanced Encryption Standard (AES) algorithms. This simulation will be able to encrypt a message that the user inputs and save it to a spreadsheet. The user can also decrypt messages that they input, or from the spreadsheet. The emphasis is on illustrating the importance of encryption and how data can be ‘securely’ stored once it is encrypted. An explanation on how the simulation works will be explained in more detail later.

The components of the program identified so far consist of three main elements, noted below:

* User interface
* Simulation of the block cipher
* Spreadsheet

The *user interface* will provide the user with the ability to read preliminary information, possibly a body of text with supporting illustrations and animations. This could be in the form of presentation slides, where the user is presented with a large diagram and supporting text. The nature of the interface suggests that it can be form-based. In most situations, the nature of understanding this field of cryptography may best be suited by supporting illustrations and captions – with the primary advantage of a picture being able to convey the same message several words might not be able to as effectively. The information presented in the slides may be sourced from online articles from academic websites and journals or online encyclopaedias such as Wikipedia.

Once the user has completed the background reading of block ciphers and their relevance to cryptography, they may then run the *simulation*. The aim of the simulation is to provide a working block cipher which the user can observe, noting each step covered of the cipher before moving on to the next. The interactivity of this feature should help the student grasp the multiple processes the algorithm works with to successfully encrypt data. The simulation will start by prompting the user to enter the plaintext message to encrypt as a string, along with a suitable master key (a password). A key schedule will expand this small master key into a larger expanded key which can be used in encryption rounds later. Once the key and plaintext message have been input, the encryption will start. An interrupt will be placed at each crucial step of the encryption algorithm, so that the user can view a snapshot of the code being executed and a description of what is happening, as well as any significant variables that may need their values inspected. The user will then be prompted if they want to progress to the next algorithm step. As the user gains a unique step-by-step view of the code, they can benefit from a precise view of the encryption components, as if they were debugging the program themselves. Once the encryption process is completed, the user can view all steps that they observed for the program, to gain a synoptic understanding of the algorithm. They will be able to view the ciphertext message, which was the message they input encrypted using the key and the algorithm.

A *spreadsheet* can be used to store the encrypted messages that the simulation would have processed, simulating a ‘secure’ information storage system; if the spreadsheet was accessed by an unauthorised user, the information would be rendered useless to them as each message would be encrypted. The key for each encrypted message would be stored within it, and the decryption algorithm would contain the necessary processes for retrieving it.

At the end of the program, the user will have the options to encrypt a new message, view the spreadsheet containing all encrypted messages, or decrypt a message. Encrypting a new message will run the simulation again, albeit with new input for the key and plaintext, and each new message will be stored successively in the next row below the previous in the spreadsheet. Viewing the spreadsheet will allow the user to observe the overall messages encrypted, presenting them with the ability to appreciate the incomprehensibility of the resultant ciphertext. The final option, to decrypt a message, will allow the user to decrypt either a message that they may enter, or decrypt a message that was saved to the spreadsheet. The former of these options serves as a backup option for decryption should the latter cause unforeseen issues, or the spreadsheet is corrupted/inaccessible.

An interactive application provides a good opportunity for the user to gain ‘hands-on’ experience with learning about the cipher. This is a major advantage over more traditional approaches such as written textbooks, guides, and YouTube videos, where the user will only be passively learning the content. An interactive program can widely supplement conventional learning from textbooks by demonstrating the processes in real-time. YouTube videos have the advantage over textbooks in that they can use the advantage of sound and moving pictures to convey ideas much better through visualisation, however, videos are limited in that they run in a fixed amount of time, and the user simply watches the video without the opportunity of interacting with it. The program that will be developed will improve on this approach greatly. Observing how the cipher’s algorithm processes the plaintext input by the user in a systematic format gives a clear insight into how the encryption works; the simulation will greatly benefit the user’s understanding of the numerous procedures the algorithm undertakes. The user interface system will be designed with simplicity in mind, so the user will be able to easily navigate between introductory slides, simulation steps and input/output forms.

The client

The client who would greatly benefit from this program would be the typical prospective Computer Science/Engineering degree student, who may currently be studying A-Levels in Mathematics and Computer Science. The basic preliminary knowledge of topics such as matrices, content from the first year of Computer Science and core mathematical skills, is essential for effective understanding of the concepts demonstrated within the program. The user will greatly benefit from preliminary knowledge, as the program will be written primarily for the A-Level student’s level of ability.

The client whom would represent this category of student is Daniel Knott, my fellow classmate. He is currently studying A-Level Mathematics and Further Mathematics, alongside Computer Science, and has the required level of understanding to effectively benefit from using the program. He is curious to know how the subjects he is learning interlink, and more about applications of mathematics in Computer Science. As he is considering an Electrical Engineering degree for his future study, he will be interested in the applications of computer systems that his field of study will be interacting with. Block ciphers such as the AES cipher are running on some microprocessors which contain their own AES instruction set, to optimise applications performing encryption and decryption using the cipher [2]. Machine code at this level may be relevant to the Electrical Engineer, who may be tasked with designing or refining processor chips.

Throughout this project, I will be developing the program within cooperation from Daniel, and the feedback he will give me throughout various prototypes during development and testing will further generate relevant points to bring forward to later iterations of the program. The supervision and guidance of the Computer Science teacher within the college will also be sought after throughout the iterative development of this project, to ensure that the information included in the program is relevant and not above the A-Level student’s understanding. Both the teacher and Daniel will serve as key stakeholders throughout the project during its development and evaluation, and they will both be asked to review the project at key stages and to input their feedback to assist in improving the program.

Preliminary research

To further investigate similar approaches to demonstrating the process and relevance of block ciphers, internet research was conducted to obtain similar programs to the one discussed in this project. The information sources sought after were concise explanations of the block cipher, and AES algorithm demonstrations, through visualisations, simulations, or outlines notes with diagrams. This research was performed as a Google search, which would be the method of investigation most typical to the student demographic. Offline resources e.g. books from the library on block ciphers and encryption were rendered irrelevant to the project requirements; they lacked the multimedia features in which the user would benefit from.

From the research, the following sources 1-4 performed an excellent job in ‘visualising’ a block cipher encryption algorithm, with sources 5-7 providing a good outline with supporting diagrams:

1. <http://www.formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng.swf>
2. <http://artfordorks.com/2010/04/aes/>
3. <https://youtu.be/mlzxpkdXP58> - AES Rijndael Cipher - Visualisation
4. <https://youtu.be/-Gk9kaFoBxU> - Cryptography Lesson #1 – Block Ciphers
5. <https://en.m.wikipedia.org/wiki/Advanced_Encryption_Standard>
6. <https://en.m.wikipedia.org/wiki/Block_cipher>
7. <https://www.tutorialspoint.com/cryptography/block_cipher.htm>

Analysis

The animation in source 1 provided an insight into the algorithm of AES through the form of an animation. The animation required flash, however, so users would not be able to view it on a browser without flash or mobile devices. The program we shall develop will not require flash, so it can be run on a PC without a browser. It can be ported to mobile operating systems such as iOS and Android if there is a demand for it.

Source 2 was an impressive video hosted on the Vimeo site which displayed crucial steps of the AES algorithm in progress through the representation of matrices as shapes similar to Rubik’s cube blocks, where the transformations of matrices was efficiently demonstrated, notably the mix-columns and shift-rows stages. The draw back was that the video provided a limited introduction to the context of the stages, and descriptions of each step. An improvement over this in our program will be sufficient text descriptions of each stage, conveying the context of the operation relative to the main algorithm. A diagram of the whole encryption algorithm could be given for each stage, with the current process highlighted down the flowchart tree.

Source 3 was a concise illustration of the AES block cipher as a YouTube video, however lacked audio commentary and little description to the context in which the displayed procedures would be relevant to, save for the title expressing it was related to AES. This source would only be useful as a mnemonic for a viewer who was already familiar with AES and block ciphers. Our program will of course contain correct annotation, and a narration of the annotations utilising text to speech software may be implemented if there is a demand.

Source 4, another YouTube video, did a slightly better job in explaining what a block cipher was especially with the informative commentary. This may well have been the best animated example introducing the viewer to the topic. Inspiration can be taken from this and implemented into our program, where animations could be implemented alongside annotations and code explaining the process, further assisting the user in understanding the topic. This may be decided at a later iterative development stage, where the client will provide feedback relating to if the user interface needs to be improved.

Sources 5-6 were Wikipedia articles, sometimes a student’s first port of call when looking up a new topic to independently learn if they weren't given links to use by a supervising teacher. As with all Wikipedia articles, there is a degree of risk that the article could be sabotaged by other users to provide misinformation, as well as the majority of general information relying on endless hyperlinks – the student would have no way to determine how long they would need to learn the said topic. While being informative, it can be said that the amount of information from these types of sites could be reduced to the details that are relevant only to the topic to be studied. Learning from this example, a certain level of abstraction will be considered when deciding which preliminary topics to include into the educational content of the program, and which topics should be implied that the user already knows.

Source 7 was an interesting learning resource which provided a concise explanation of block ciphers, as well as surrounding topics for a contextual introduction to block ciphers within the subject of cryptography. It even proceeded to explain each of the common block ciphers, such as the Feistal cipher model, Data Encryption Standard (DES) and Advanced Encryption Standard (AES). This resource, however, lacked animations or explanations on how to implement the ciphers in code. Content to be included in our program could be sourced from this website, with adequate citation references provided in the credits section if need be. Again like with sources 5 and 6, information to be included in the program’s educational content can be selected relative to the user’s assumed preliminary knowledge.

As with all the sources examined, most did effectively inform the user about block ciphers. However, they lacked a demonstration of the process from a programmer’s point of view. In the context of programming, pseudocode could have been provided to show the different steps of the algorithms being processed.

The program to be developed will display the code being executed at each step, possibly by screenshots of the procedure’s code that was being executed at the time.

Conclusion

From this research, we can conclude that the key features of the program will need to include, but will not be limited to:

* A graphical user interface designed with simplicity in mind
* Relevant educational content at a level where a reasonable background of preliminary knowledge is implied from the user (A-Level maths knowledge, understanding of symmetric keys and matrices etc.)
* A simulation consisting of a running cipher algorithm, along with interrupts at each major step, where screenshots of the procedure being executed can be displayed on the GUI along with annotations
* A spreadsheet to store the encrypted results to decrypt later – this will be helpful in giving the user a ‘hands-on’ experience of viewing the results of their encryption and how the encrypted message would be virtually incomprehensible to the unfamiliar unauthorised user
* Possible features that could be added at a later iterative stage could include text-to-speech annotation and animations for the user’s preference

Identifying limitations

As with any application, there will be known and unforeseen limitations towards its functionality and use. The limitations known relate directly to the components the program will be developed from.

The simulation is an abstraction of a block cipher, and will never be able to accurately represent a real block cipher due to some concepts being utilised in the creation of the algorithms requiring postgraduate knowledge of mathematical principles, e.g. Galois fields used in the key scheduling algorithms of the AES cipher. The concepts can be simplified, but only to a certain extent until they are no longer a faithful abstraction of the original topic. Some steps will be reduced in the encryption algorithm to simplify the content, so that the program communicates a concise description of the conceptual encryption process, rather than an accurate simulation of a real block cipher, such as AES or DES. The encryption algorithm itself will be loosely based on AES, but the steps will be simple. The main objective in this program will be to ensure the user understands the content, not the user needing to memorise every single step of a real encryption algorithm. The simulation serves purely for connecting the programming theory and computational concepts together, into an application package that the user will be able utilise effectively for understanding.

The program will be developed initially for a desktop-based operating system, so will not be able to run on mobile devices. However, the functions and procedures will be formulated in such that they can be ported over to a more compact operating system with ease; through efficient annotations and comments within code, and appropriate identifiers for procedures, functions, objects, and variables. The user interface itself will limited to the constraints of the display of the running computer, so the readability of the forms will be restricted to the resolution of the monitor.

System requirements

This section deals with specifying the minimum system requirements needed to run the program. It is not recommended to run the program on lower specifications than the ones mentioned in this section. The operating system, application dependencies and the hardware specifications of the computer running the program will be mentioned.

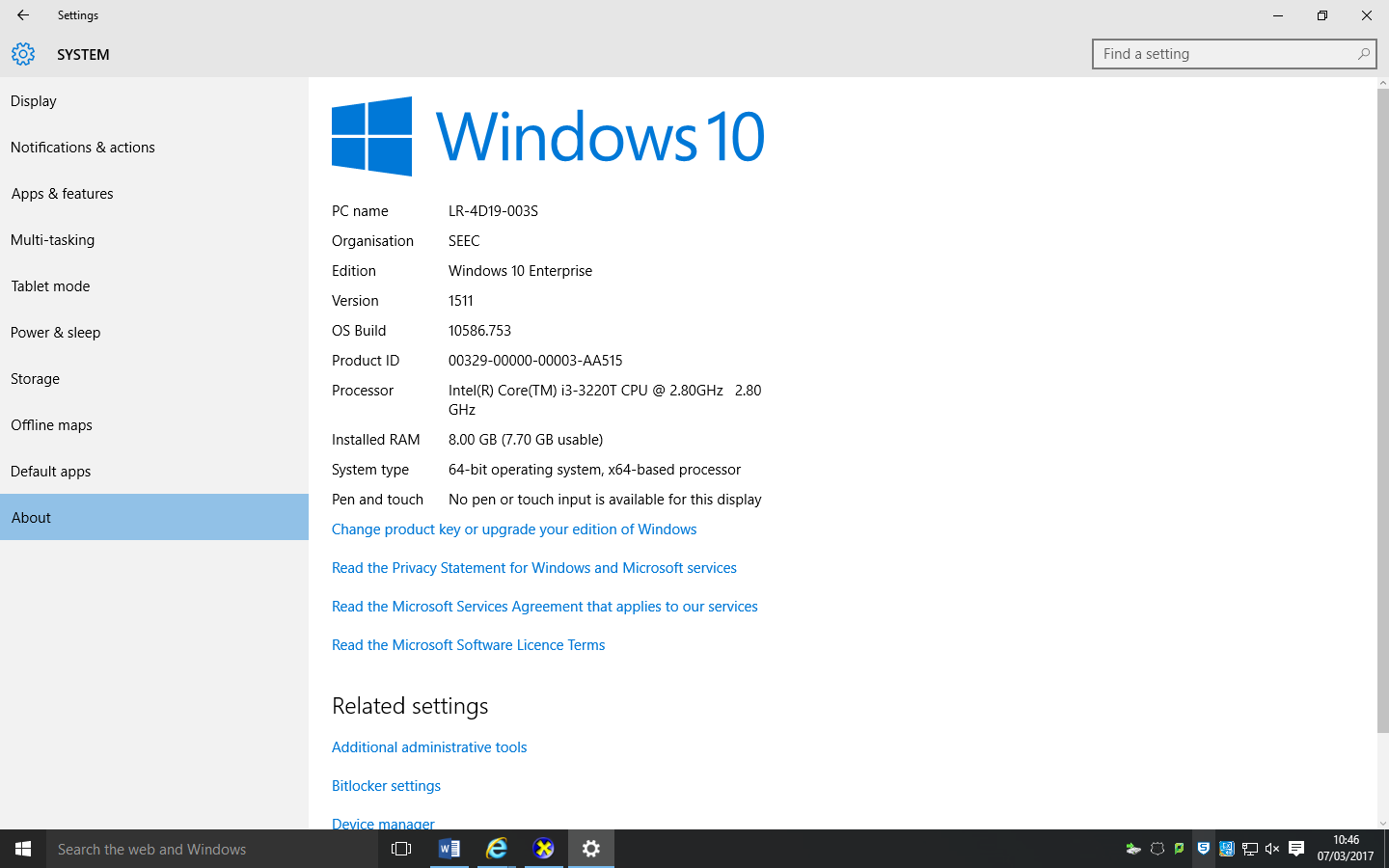
Software requirements

The program’s code will be written in Visual Basic 6.0 and complied using the Microsoft Visual Studio 2015 IDE. The program is designed to run on the Windows operating system (10 and above), and will not run on other operating systems.

Hardware requirements

As a sensible measure, the minimum hardware requirements will be the hardware specifications of the computer it was tested on. The computer that will be used to create, compile, and test the program will be the college issued-desktop PCs which are used within the Computing & IT wing of the campus, as well as the Learning Resource Centre (a computer-based library which most students utilise to complete their coursework assignments). These computers demonstrate a sub-par performance during performing regular duties such as booting-up, opening internet browsers and accessing files. This deficit in performance will be used as an advantage to represent the minimal specification environment that the program can be run on.

A snapshot of the computer requirements of the college issued-desktop PCs is shown below. This snapshot was taken from the system information page of the Windows 10 OS, which displayed the relevant information for the specifications of the computer.



From the screenshot, the essential specifications of the computer can be observed. Below is a list of the minimum requirements of the program; the relevant specifications of the college’s PCs:

* Windows 10 operating system (64-bit)
* Keyboard
* Mouse
* A suitable Visual Display Unit
* A processor clocked at 2.80GHz or above
* 8GB of RAM

The program may require up to 200MB of storage. It is a simple application, so not much storage will be needed but it is advisable that the user is aware for good measure.

Success criteria

For the project to be a success, the program must be able to run on the desired platform within the system requirements. The client will be able to fully utilise the program to their learning abilities and will be satisfied with the final product. Other factors also affect the decision on whether the program is successful or not, such as whether the program effectively educates the user about the topic of block ciphers. To organise this further, the multifaceted success criterial will consist of three core elements, designated “Requirement Objectives”, which can be measurable throughout different stages of the project as the development progresses. These Requirement Objectives (RO) are essential to the completion of the finished program, and will be numbered for clarification during quick reference. They are as follows:

* RO 1. – functionality (the program must be complete, minimising redundancy)
* RO 2. – user friendly (the client must find the program easy to use)
* RO 3. – educational relevance (the content to include must suit the client’s degree of understanding)

To elaborate on these Requirement Objectives (ROs), each will be explained as follows. RO. 1 covers the overall finish of the program, such as the completion of key elements such as a running simulation and comprehensive user interface. RO. 2 supervises the usability of the program, such as well-thought forms and images which will stimulate the user's interest in the program’s educational content, as well as the user interface being easy to navigate through with little or no documentation being required to use it. RO. 3 will be considered throughout the entire project, and is responsible for ensuring the educational content included will suit the typical user’s level of understanding (such as whether a certain topic requires a diagram to support its body of text, or whether a diagram is not necessary).

The description of these ROs for this specific program will be explicitly defined as guidelines for the design, iterative development, and evaluation of the program. Contextual definition will be given as follows:

* **RO. 1:** during the design stage, every form and process must be defined. The user interface will be visualised, and considered as a key component of the overall program. *There is an overlap with the design stage description in the OCR project specification – this RO overlaps with many of the points made and expands upon them by linking them to context of this project.* During development and testing, programming methods in place such as annotations, suitable identifiers for functions and variables, and appropriate libraries will be considered in efforts to reduce redundancy in the elements of the application, ensuring wholeness of the project and readability for the programmer should improvements need be made during the evaluation. Thorough testing will be conducted during the development and testing stage for the simulation’s input values against output, to effectively simulate the block cipher system. More on testing will be explained during this stage.
* **RO. 2:** throughout the design stage, the user interface and relevance of the learning method will be considered. The client may be prompted to suggest whether they prefer improvements to be made upon the proposed solution, and modifications to the original design will be made prior to development so that less time is wasted during the programming stage on backtracking procedures and rewriting them for the sake of a non-technical requirement. Forms will be designed with simplicity in mind, and minimal elaborate graphics. The simulation will be considered to run efficiently and suitable annotation will be provided in each procedure as if it was a step-by-step guide to the procedure’s workings. This will be essential for the screenshots of the code that will be displayed to the user in each step of the cipher process, so that minimal supporting text will be required.
* **RO. 3:** as the typical user (and client) will be learning at a Level 3 qualification stage, it will be assumed that they have completed GCSEs to a high standard, so many English and Maths skills will be assumed without a prior brief. Since the user will have taken A-Level subjects in Maths and Computer Science, and preferably Further Maths (though not required), only a brief introduction to the concept of matrices and encryption will be required before moving on to the main topic area. Content will be cited from educational resources on the internet, and there will be a credits section in the user interface system which shall display the citations should the user need to verify the program’s content. This RO will be assessed throughout the design and evaluation stages. RO. 2 will cover RO. 3’s requirements within the development stage, so RO. 3’s explicit implementation will not be necessary.

At the end of each stage during the iterative development process, and for relevant parts of the design description, a table will be devised with each of the ROs and an explanation on how the work organised in the stage was able to satisfy *each* of their requirements towards creating the complete project. For each stage to be marked as successful, it must satisfy *at least one* RO, before the next stage may be conducted. The entire project will be successful when the ROs are satisfied during evaluation and the end user, the client, is fully content with the final program. If there is a lack of time towards development of the project at the end, a statement must be made describing what will be necessary to satisfy all three ROs if there were more time available.

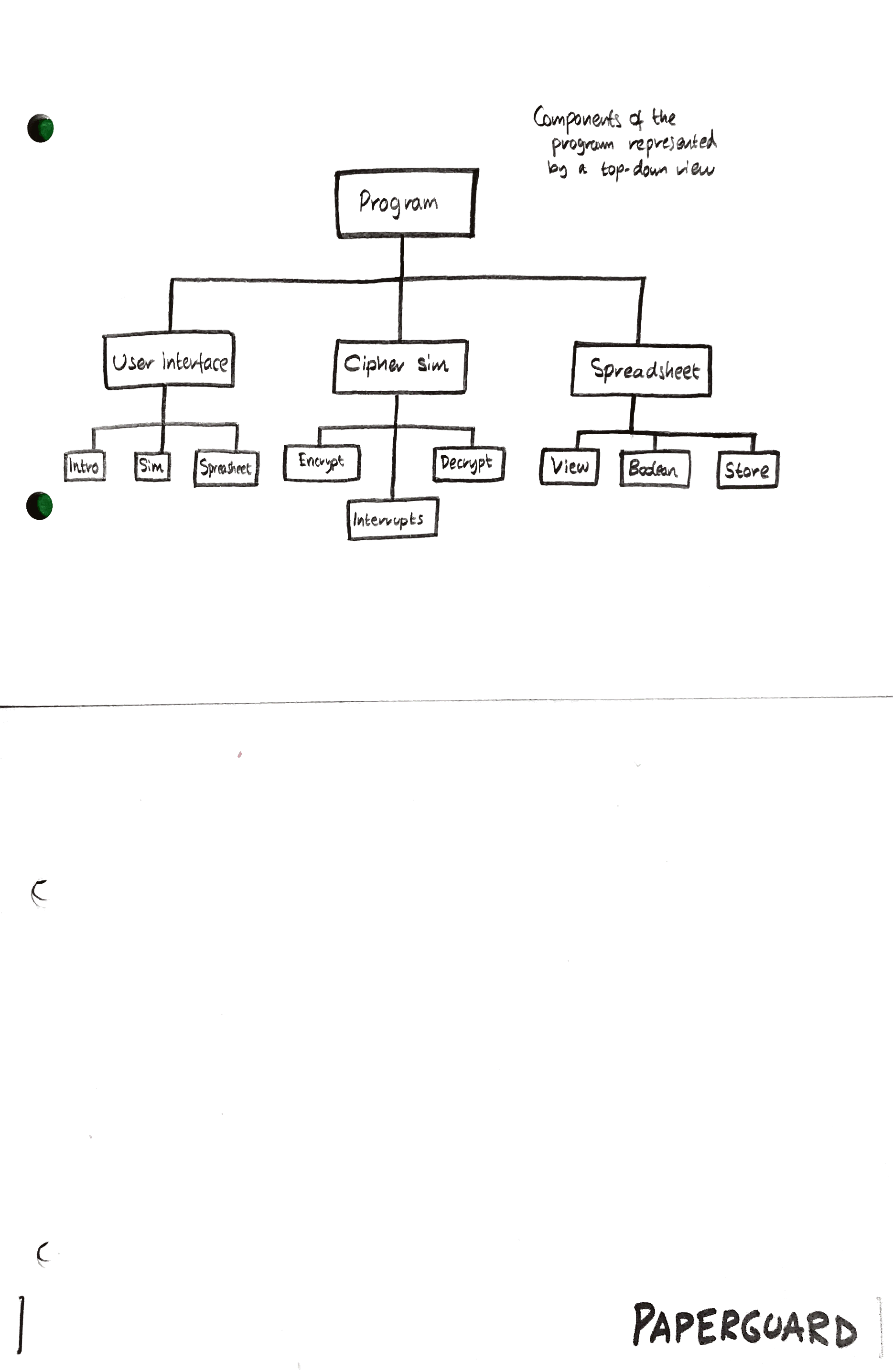
Design

# Problem decomposition

The program will consist of three main components: the user interface, the simulation which was explained above, and a spreadsheet. The user interface will provide the user with a medium to enter input and view output from the simulation, and the spreadsheet will store the results of the encryption for later retrieval (should the user want to decrypt the messages they stored). The user interface and spreadsheet implementation will be described in more detail now that the cipher simulation has been explained.

The design of the program will be focused around the simulation. The user interface, introduction, and spreadsheet implementation serve simply as an aid towards the interpretation of the simulation from the user’s prospective. So in this stage, the components of the simulation will be explained and demonstrated first before discussing the user interface and other features in a similar manner to how a book is usually written first before it is given a title. To build an effective simulation, the components of the process in which it is modelling must first be identified and examined carefully before an abstraction can be made.

Within the user interface, there will be sections to introduce the user to the cipher topic and to integrate with the simulation and spreadsheet. The cipher simulation will provide encryption and decryption functions of a user-input message with suitable interrupts placed at key steps, so the user will be able to observe what is going on. Within the spreadsheet, storage of encrypted messages and viewing of these messages must be implemented through a link between the program and the spreadsheet file. The spreadsheet is also responsible for storing a Boolean value letting the program know if the user has run it for the first time.



Above is a top-down chart illustrating the components of each major section that are needed to be developed in order to create the program. These components will be considered during the development process in order to clarify which work must be attributed to which skill set i.e. when working on the user interface, consider the introduction pages, the simulation interrupts and integration with the spreadsheet.

Block ciphers explained

*The type of cipher that will be used in the simulation, along with the preparatory information required to interpret its operation will be shown in this section for clarification purposes and reference for when it is being written in code during development.*

A block [cipher](http://searchsecurity.techtarget.com/definition/cipher) is a method of encrypting [text](http://whatis.techtarget.com/definition/text) (to produce [ciphertext](http://searchcio-midmarket.techtarget.com/definition/ciphertext)) in which a cryptographic key and [algorithm](http://whatis.techtarget.com/definition/algorithm) are applied to a block of data (for example, 64 contiguous bits) at once as a group rather than to one bit at a time [3]. Block ciphers work by organising the plaintext data into a matrix, called the ‘state’, on which several mathematical operations will be transformed on its elements in order to make the data secure. If there is less plaintext than expected (e.g. 12 characters input to an 8x8 matrix), the state is padded with trivial values decided by the cipher algorithm (the padding must be consistent so it is possible to decrypt later).

These operations can include:

* Transpositions – rows can be swapped with columns or other rows, and so on.
* Substitutions – elements can be substituted with other elements from a lookup-table (an extra reference matrix) and operations can combine the elements on the matrix with the elements on the lookup table.
* Permutations – elements, rows and columns can be arranged in a manner which can completely confuse the reader should they attempt to decipher the matrix containing the information

With these three operations performed throughout the encryption algorithm, block ciphers can perform a powerful encryption of data. Computers can perform these simple operations relatively fast compared to some more difficult operations, such as evaluating prime numbers at very high values. (The latter used in some encryption methods which can prove very secure, but also very taxing on the CPU’s ALU so not suitable for conventional systems). Due to the relative speed in encryption and decryption of these matrices, along with the high security of their ciphertext (in some configurations of the AES cipher, the complete plaintext is practically inaccessible by brute force methods [4]), block ciphers are widely used in many forms of encryption on computer software and hardware. Another advantage of block ciphers is that by being a symmetric type of cipher, the precise process of the encryption algorithms can be known and is publicly accessible, without jeopardising the security of the encryption process. This is known as Kerckhoffs’s principle [7].

Symmetric key cryptography is useful if you want to encrypt files on your computer, and you intend to decrypt them yourself. It is less useful if you intend to send them to someone else to be decrypted, because in that case you have a "key distribution problem": securely communicating the encryption key to your correspondent may not be much easier than securely communicating the original text [5]. For these purposes, the key is usually sent through a secure network or is itself, encrypted using a longer cipher

Typically, a block cipher will accept two inputs and produce a single output. The plaintext and the key will be input to the cipher. The encryption process will use the key to define the number of ‘rounds’ (iterations of the main cipher process) that will be performed on the data, as well as substitution values and permutation instructions. The key is usually 128 bit, 192 bit or 256 bit, and will be used to generate sub keys that can be used for each round later by using a key schedule algorithm [6]. In the main process, the algorithm will run a set of instructions which perform an iteration round of the encryption. These instructions could consist of the main three operations performed on the state: transposition, substitution, and permutation. Once these following operations have been applied, the state enters a final integrity check before it is considered ciphertext. To decrypt the ciphertext, the reverse steps are applied. The algorithm simply requires the key that the cipher was encrypted using, and the ciphertext as input. A check will be performed to determine if there are any padding values present, and once they are detected, will be excluded from the decryption algorithm. (For padding values, they may fit a mathematical rule that is true for certain values or bounds. Fool proofing will be implemented in the encryption algorithm to avoid padding false positives for non-trivial values.)

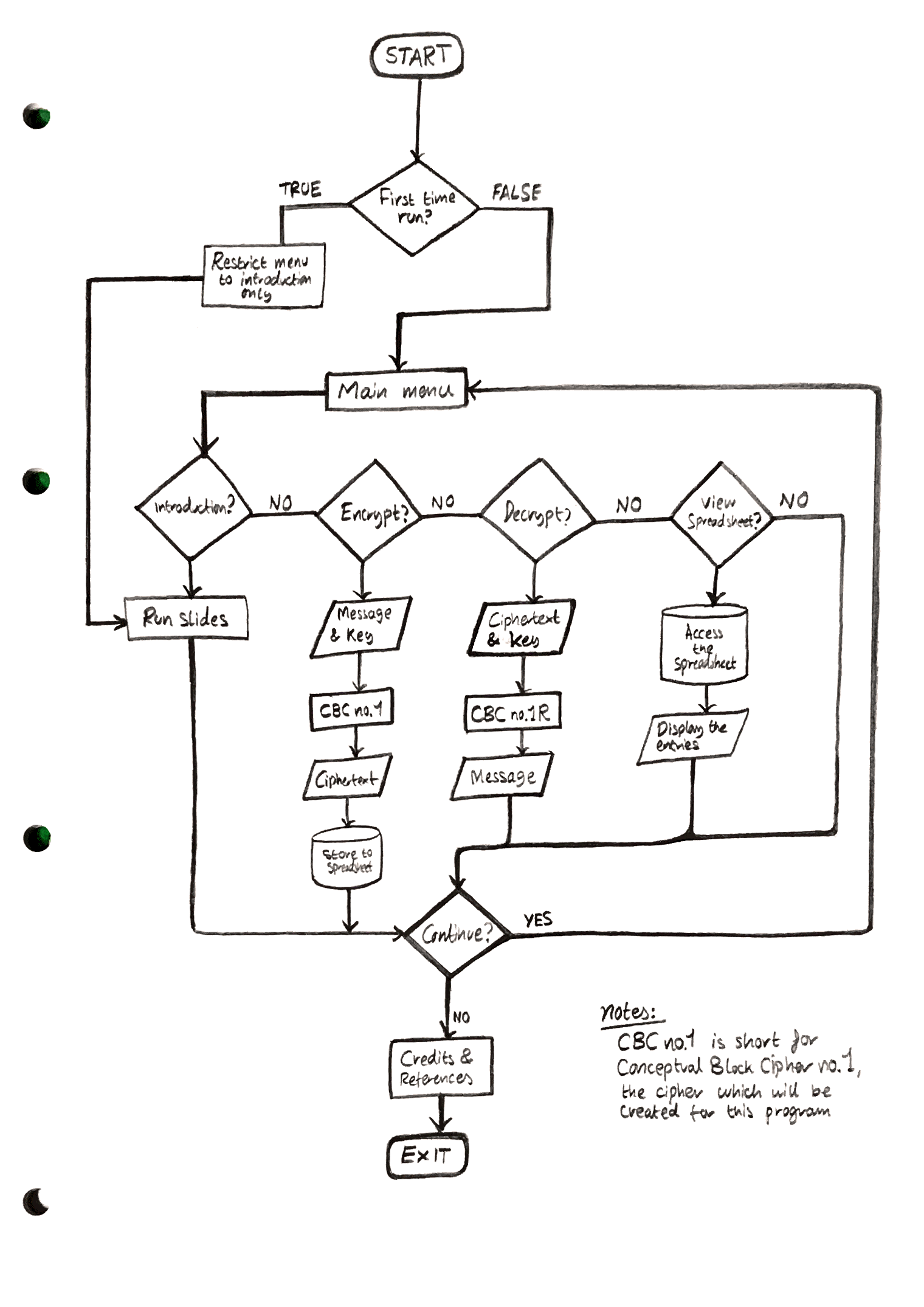
The precise description of the block cipher that will be written in the program will explained in the algorithms section, along with the rest of the main algorithms utilised in the main program.

# The solution

The program will be written in the Visual Basic programming language in version 6.0, and compiled into an executable application using the Visual Studio 2015 IDE. This choice is made due to the fact that Visual Studio comes pre-installed on most the faculty computers within the college. Visual Basic (VB) is a popular and versatile programming language which provides support for object-oriented programming and with the abundance of libraries available, includes the ability to implement matrices and bitwise operations on data – which are essential to the workings of the program. Another purpose that VB is chosen is due to it being the most familiar programming language in the Computer Science classroom. It was the teacher’s language of choice when the CS class started, and all students have a good understanding of its structure and syntax. This means it will be easy for the client to read the code which will be important later in the program.

The combination of the simulation, a user interface with educational slides and an informative glossary will provide the user with an interactive application which will greatly help their understanding of the topic and further satisfy their interest with the subject, as mentioned in the Analysis section. The spreadsheet used will be created as a Microsoft Excel document. This allows for easy integration with code written by Microsoft’s Visual Studio IDE and is a practical format for the program’s simple processes.

Below is a flowchart of the whole program’s operation:



As shown above, much of the application is centred around the simulation. The program will consist of a Windows Forms Application, providing a familiar native form to the menus and user interface elements of the program (this can easily be created in Visual Studio). At the beginning, the user enters the main menu. At the main menu, they are presented with the following choices:

* Visit the cipher introduction
* Encrypt a new message
* Decrypt a ciphertext message
* View the encrypted message spreadsheet

If it is the first time that the user has run the program, only the first option will be available, visiting the introduction page. This page is accessible to experienced users (users who have read the introduction and encrypted a message etc.) if they wish to revise the content in that section. Tracking if the user has run the page first time round is performed by a check at the program start up, dependent on a Boolean value stored in the spreadsheet.

In the cipher introduction page, the user is introduced to the topic of block ciphers through the form of presentation slides, which they must read before proceeding to the simulation if it is their first time running the program. When the user has reached the simulation stage, they enter their input for the message to encrypt and the key they wish to use to encrypt it with. The simulation process runs the cipher algorithm which encrypts the message using the key. It is advised that the user thoroughly knows their key and it is a memorable one, or that they have it written down on paper/a secure document as the key is the only means of decrypting the ciphertext message later. The user may be prompted to do this at the beginning.

During the simulation, the algorithm is paused at major stages to show the user what is happening. Interrupts are placed at each stage of the encryption process to achieve this. At each stage, the form page will show three parts of information: a snippet of the code that is executed, a description of the code being executed including its stage in the algorithm and its function, and the inspection of key variables such as the round key and the state. This will provide the user with an immersive experience of the encryption algorithm. Once the message is encrypted, the encryption algorithm will be complete and the encrypted message will automatically be saved to a spreadsheet. The user will have the option of three choices:

* Copy the ciphertext to clipboard
* View the message’s entry in the spreadsheet
* Return to the main menu

Decryption of a ciphertext message works in an equivalent manner to the encryption process. The user must enter the ciphertext message and the key that was used to encrypt it as two inputs to the decryption algorithm, and the algorithm decrypts the message using the reverse process of the encryption cipher (more explained in the algorithm section coming next). At end of the decryption, the user can view and copy the decrypted message to clipboard.

Viewing the spreadsheet is purely for observational purposes and will display the current list of saved messages that the user has encrypted using the program previously. If it is the first time the user has run the program/it is freshly installed on a computer, the spreadsheet will contain no data (except the Boolean value used in the first-time user check). In theory, the user could decrypt a message from the spreadsheet by copying it to clipboard, then pasting it as input to the decryption simulation. Of course, they would need to remember the key they encrypted it with – they could use the same key for every message if they wanted to avoid forgetting it in the future, though this would be less secure in the long term. Throughout the whole program the user can quit the simulation and return to the introduction page, or quit the entire program.

With the components of the program described, the main algorithm processes will now be explained in detail.

# Primary algorithms and processes

Most the user interface algorithms are trivial and simply navigate the user throughout the program, so the will be shown in the development section, rather than being described here.

The main algorithm being used in the program is the in-built block cipher, which is designed to encrypt a 16-character message using a 16/24/32-character key. The algorithm is also capable of decrypting a message provided the key is input along with the ciphertext.

The purpose of this algorithm is to provide an effective simulation that the user can actively observe through each major step, serving the breakthrough element in educating the user how this type of cipher works.

Conceptual Block Cipher no. 1

*This section will describe the precise algorithm we will use in the simulation process of the program. It will be written as a guide to the reader as to what most the code will be focused on, during the program development. It will also be used as a reference guide during program development.*

This block cipher algorithm is a simplified abstraction of a real block cipher, with a few components modified for simplicity. The encryption rounds are loosely based on the rounds used in the AES cipher. This abstraction also takes inspiration from AES with the number of rounds increasing as the key size gets larger. The basic summary is that the algorithm takes two input values, plaintext data and a master key, generates sub keys from the master key and uses them in each round to encrypt the plaintext data. The ciphertext produced is the same size and data type as the plaintext. With the key, the algorithm can perform reverse operations to decrypt the ciphertext and obtain the original message.

***Input***

Many block ciphers operate with data at a low-level in the form of bytes. A state is created that contains each byte of the original message. If the number of bytes exceeds the maximum that the predefined state can carry, the overflow of bytes is marked and placed in another state that the algorithm will encrypt separately (using the same key, of course). If the number of bytes is less than the amount expected by the state, the matrix is padded with trivial values.

The difference in the algorithm that we are going to develop is that the cipher will operate on characters. In Visual Basic, an ASCII character is 1 byte. This provides the substitution of characters over bytes as an effective approximation of the input data a block cipher operates on – the data element sizes are the same. Instead of decomposing the plaintext message into individual bytes, the message will be split up into an array of characters, which will be input to the state. This simplifies the program without any dependence on low-level libraries and will speed up the development process. Implementation for bytes can be added later.

The two parameters that the algorithm will take are as follows:

* The plaintext message (maximum length of 16 characters – equivalent to 128 bits)
* The encryption key (can be either 16, 24 or 32 characters long – equivalent to 128, 192 or 256 bits respectively)

The user will be able to select the length of their key (16, 24 or 32 etc.), and then type their key of maximum characters of that length. The user will be able to verify the key and copy it to clipboard to save for later use. (A prompt will instruct the user to write the key and key size down or save it to a secure storage, since these two values will be their only means of decrypting the ciphertext in the future.)

With the necessary inputs provided, the algorithm will now be able to prepare them for encryption.

***Preparation***

The 16-character plaintext will be input into an array of characters that will evaluated to see if it contains any empty elements. If there are empty elements present, they will be padded with trivial values, which will be able to be identified later for decryption purposes. These trivial values take the binary value of zero, as it represents a NULL character in ASCII. The user interface will not let the user enter NULL values for their message, so that this distinctive character will be reserved for empty space in the message. The elements from the array (including the padded values) are then transferred to a 128-bit 4x4 column-major matrix, the *state*. (Columns dominate the matrix, with the elements of the columns stored closely in memory.)

The encryption key which was input to the algorithm will be utilised to generate sub keys to be used in each encryption round. The amount of sub keys generated depends on the number of rounds, which is dependent on the key size. Keys of 16, 24 or 32 characters will generate 10, 12 or 14 sub keys respectively. A key schedule will generate the sub keys. The key schedule is essential towards preparing the key to be used within the encryption process. This process is irreversible as there is no reason any user would need to view the expanded key or obtain the original key from it – this stage is purely technical and will be performed within the procedure. The details of the key schedule procedure are below.

***Key schedule***

The term “master key” will be used to refer to the key that the user entered through the user interface or that was randomly generated to encrypt the message with. This key schedule is responsible for taking the 16/24/32-character master key string, and expanding it to provide as many 16-character sub-keys as required for the 10/12/14 rounds the encryption process will perform on the state. The sub-keys must be unique so that there is an added layer of security applied to the state with each round.

The fundamental concepts are the same for each size key i.e. 16, 24 and 32 character keys. For example, a 16-character master key will be expanded into an array that contains 10 distinct 16-character sub keys, which can be used during the encryption rounds to be XORed with the state, for example. If a 24 or 32-character master key was used instead, it would be expanded into an array that contains 12/14 16-character sub keys for the respective 12/14 rounds that would be performed. *No matter how large the master key or number of rounds that are performed may be, a singular sub key will always be 16-characters to match the dimensions of the state.*

Expansion procedure

*Before explaining this procedure, it is important to note that the details of reversing the expanding key is not important. The key expansion process is irreversible as the original key need not be obtained. The only process that is required to be reversed is the actual encryption procedure which will be performed on the state. The expanded key is simply required to reverse those steps, not the steps of creating the expanded key itself. The expansion procedure will be run during the encryption process in the same order, because the key must be expanded to get the same values that were used to encrypt the state with. The expansion process must be repeatable, so that the same round keys will be generated from the key during encryption and decryption.*

Creating the expanded key

The basic instructions designed to expand the master key in binary will be as follows:

1. The 16/24/32-character key string is input
2. The key string is iterated through and each character is converted to its ASCII value in bytes
3. The ASCII values are stored to an array of bytes with the same dimensions as the key size
4. The key size determines the number of duplicates, with the aim of creating enough round keys
   * For a 128-bit key, the expanded key must contain (10\*16)/16 = 10 duplicates
   * For a 192-bit key, the expanded key must contain (8\*24)/16 = 8 duplicates
   * For a 256-bit key, the expanded key must contain (8\*32)/16 = 8 duplicates
5. Each value in the byte array is iterated through and duplicated to a new array with above dimensions, dependent on the key size
6. The expanded key is then created by dividing the array into 16-character sections, each representing a round key. The result is stored to a 2D array. This ensures the expanded key contains 10/12/14 round keys for the respective 128/192/256-bit key sizes
7. To make each round key unique so that it can be used effectively within encryption rounds, this expanded key array is subjected to the key-expansion procedure which is as follows:
   * Perform a circular\* left shift of rotation 2 bytes in the block
   * XOR each byte with the previous byte

\*Circular shifts are achieved by converting the byte values to their respective 8-bit binary values as a string of 1’s and 0’s. This string is then converted to an array of characters and the array is iterated through, setting each element equal to its succeeding element, and the last bit is set to equal the first.

***Encryption process***

We now have a 4x4 matrix containing 16 characters that can be operated on, and an expanded key containing enough round keys for the rounds specified. Thus, the encryption process will begin. During this stage, the following operations will be performed successively upon the state:

1. Byte substitution
2. Row-shifting – a circular shift
3. Column-transposition
4. Implementing the round key

To effectively encrypt the data in the state, these operations need to be performed multiple times. They will be repeated in each round, each time making the data more secure.

*Byte substitution*

A look-up table, stored as a predefined matrix of bytes at the start of the algorithm, will be mapped onto the corresponding bytes in the state matrix. The mapping will be performed as follows:

* Each binary value in the state is XORed with its corresponding element on the look up table
* The look up table is defined as the ‘S-Box’ at the beginning of the program, with the same dimensions as the state

*This is like the SubBytes process in the AES cipher but the operations performed are less complex and more straightforward, with security being sacrificed for the purposes of demonstration.*

*Row-shifting*

This is a simple permutation process which is composed of circular shifts towards the bytes in the row [10]. The same techniques with the circular shifting in the key expansion procedure are used the steps on shifting the elements in the rows are as follows:

* Each row is shifted 2 bytes to the left

Therefore, there is a permutation of bytes between the columns.

*Column-transposition*

In the AES algorithm, for each column, each byte is mapped into a corresponding new value which is an output of a function using all bytes in the column [11]. In this simplified cipher algorithm, the columns will just be swapped instead i.e. a transposition of the matrix. Reversing a function in which a byte is mapped by a function using all bytes in the column is not practical for the purposes of this demonstration.

*Implementing the round key*

Each byte of the state is combined with the round key by performing an XOR operation per byte. For this to happen, the nth element in the expanded key array is used for round n and each byte per round key becomes XORed with each byte in the state.

***Output***

The result is a 128-bit encrypted ciphertext message, represented as an array of 16 hexadecimal byte values. The advantage of representing the output as a hexadecimal value as opposed to decimal ASCII values or converted ASCII characters is that the numerical values are preserved if they are out-of-range of the ASCII value bounds (i.e. over 127) but they can be easily converted to whatever they were before through reversing the operations and adapting the decryption algorithm to take the byte array as input.

The property of the hexadecimal values is that they keep their number of digits for a high-range i.e. most hex values in this context will be two-digit, making it easier to store them in an array/more practical to read and write down.

The hexadecimal output can be converted to a string and can be exported to a Microsoft Excel spreadsheet and stored as a ‘secure’ message for demonstration purposes. The user can view the encrypted message and copy it to clipboard if they wish.

It is possible to decrypt the hexadecimal output and turn it into its original character string, through methods of decryption and using the same key.

***Decryption process***

The steps taken to encrypt the message in the encryption process are designed to be reversible so that the encrypted message can be decrypted. The key is needed to decrypt the ciphertext – without the key, it is not possible to decrypt the message.

To decrypt the ciphertext, the following steps must be considered in the following order:

1. Implementing the round key – XOR with the round key again, this time work backwards from the round key list e.g. for a 128-bit key size, the first decryption round is 10, the second is 9, and so on.
2. Column-transposition – again, swap columns with rows
3. Row-shifting – a circular shift in the opposite direction
4. Byte substitution – XOR with the S-Box like in encryption, but like round key implementation, work backwards from the array in relation to the selected columns in the state

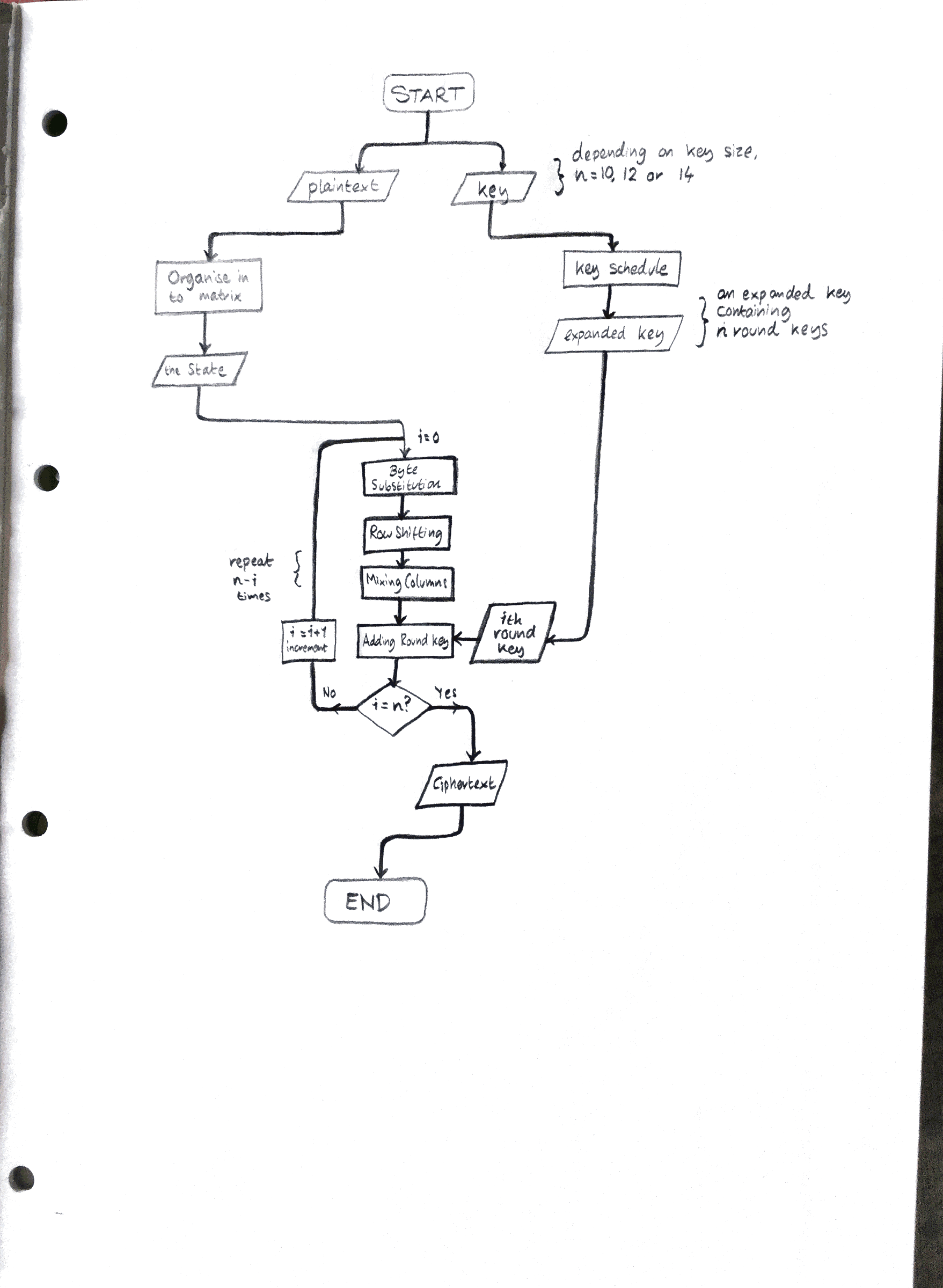
This stage must be performed carefully. Each step must be reversed completely to obtain the original value, including the order of the rows and the selection of the round key from the expanded key. This will be discussed in more detail during the development stage.

Summary of the cipher algorithm

Albeit a loosely based on some elements from the AES cipher, this block cipher contains the necessary steps involved in simulating an encryption of a message, and successful decryption of it. The steps are organised as such so that they can be easily followed by the user when they are being presented with code snippets and a contextual description of each process.

Within each vital step of the encryption process, an interrupt will be placed in the subroutine providing the user to view the code snippet of that subroutine and a description of its operation, as well as the ability to inspect variables such as the round key or current modification of the state. This is crucial to the relevance of the learning process in the program altogether, and will be the deciding element which the client will draw upon when they are satisfied with the final application.

A flowchart summarises this cipher simulation, so that the whole operation and the order in which the subroutines are executed is shown in perspective.

**

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement objectives met: | ***RO.1*** | ***RO.2*** | ***RO.3*** |
| *Contextual description of the objectives met in this section* | The cipher algorithm has been fully defined in operation and parameter.  The use of flow-charts and drop down menus have described the structure of the solution in detail. | - | A description of the topic of block ciphers has been given, which can be placed in the introduction section of the program at a later stage. |

# Usability features

The interface will consist of two main parts: the introduction and the simulation. The introduction will consist of preliminary information to brief the user about the context of the cipher and how it works in general (like the *Block ciphers explained* section). This information will be organised into slides like a classroom BoardWorks or Microsoft PowerPoint presentation. This format will present the content to the user in a clear and easy-to-understand manner, with simplicity as a priority focus so that the information is not over complicated.

The pages will be organised into an encyclopaedia topic-style form factor which resembles the traditional textbook article layout which the user will be familiar with. The sketches of the introduction diagrams are shown below.

Once the user has completed the introduction slides, they can move on to the simulation process. At the beginning of the simulation, they will be briefed on what they will expect. This is so they will be able to take full advantage of the process and understand how each component of the simulation’s algorithm links to the next. Once the user has been briefed on the overview of the cipher and the simulation process, they may continue. The next page will prompt them to enter a message to encrypt which can only be a maximum of 16 characters, which can be entered in a text-box which only accepts ASCII characters. A key will be input in the same manner, and the user will be prompted to enter a memorable key or write theirs down before it is used in the encryption.

Once the message and key are input, the encryption begins. The user is taken to a new page, which will display all the relevant information for the current encryption stage to be dissected and studied. The page should contain four key items:

* A text box containing a snippet of the code being read, containing relevant and helpful comments
* Another text box describing what the code is doing, as well as its role in the larger process
* Two labels which contain brief text describing the values of variables, the name of the stage etc. that can change dynamically, depending on the code being observed
  + Label 1 will contain the contextual information of each step in the process, such as the name of the stage e.g. “Column Mixing”, “Byte Substitution”, as well as the value of any local variables of key importance in which the changing values of these variables will clearly indicate a crucial step being made
  + Label 2 will contain the information linking the process to the overall program, as well as the inspection of major global values such as the state, the expanded key, the current round key and round number etc.

With the above information, the user should be able to follow the encryption process effectively. The same layout of the page will be applied for the decryption process, where the user can appreciate the importance of reversing the steps and using even functions to obtain the original message.

This format will be the standard throughout the whole simulation process, and will be closely linked to the interrupts in the encryption/decryption code by passing key values to the pages from the functions.

Sketches of the user interface

Sketches of the diagrams which will be created for the program are shown below. Supporting information in the margins will show clearly what each element on the pages will look like/contain.

Throughout each form, the user can return to the main menu and quit their current process. Global and public variables may have their values refreshed, should the simulation be running again with new data. As is standard with most Windows forms, the three main window-manipulation options are present on the top-right corner of the form – close window and exit program, maximise the window, and minimise the window.

Simulation menus

*These three menu forms will be used to present the user with an interactive experience during the cipher simulation. The data capture form will be displayed first, followed by the simulation process tracker and the simulation results form.*

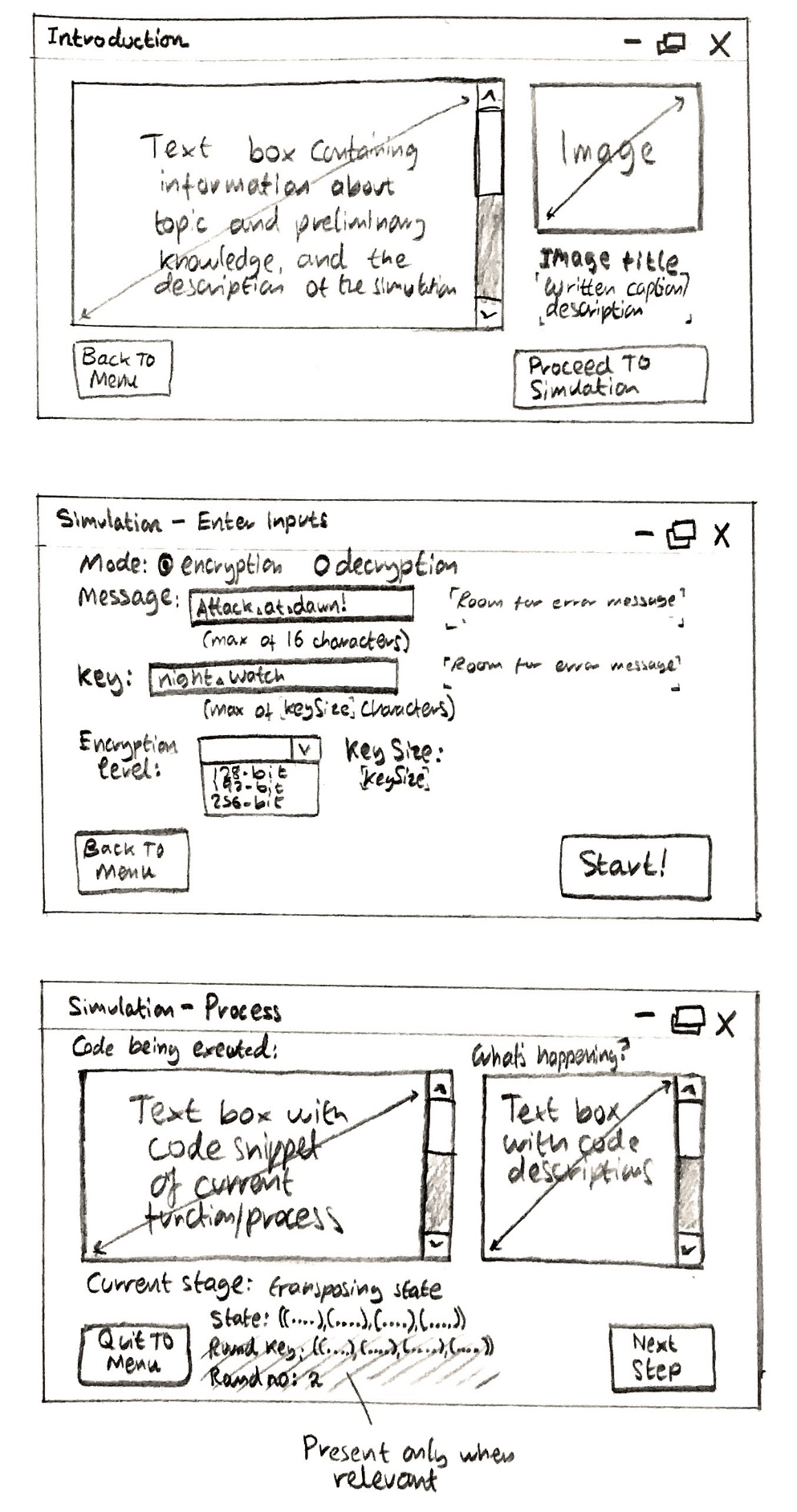
***Data capture form for the block cipher simulation***

*Captures input for the encryption/decryption mode, message string, key string, and encryption level selection. A radio selection button of two values, encryption, or decryption, is used to decide which mode the simulation will be run in. The restrictive nature of the radio selection button is suitable for this data input as only one value should be selected. The message and key are entered into a text box which returns an error if the characters entered are over the limit. A drop-down list is used to select the encryption key – this ensures the user doesn’t make any typing mistakes with entering the wrong value for the encryption level.*



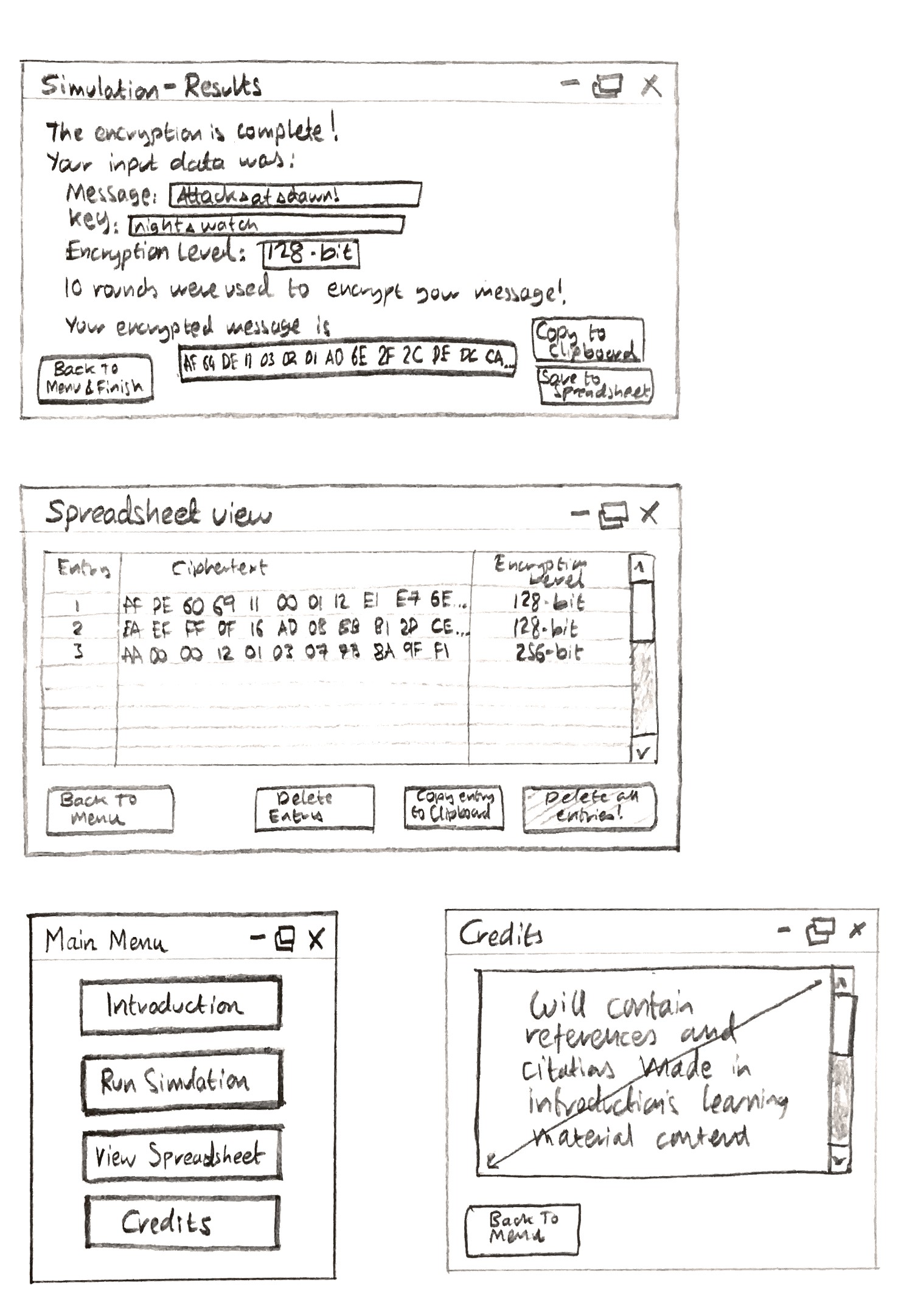
***Simulation process tracker***

*Displays the current step during encryption/key expansion procedures, showing snippets of the code being executed at the step and a description of the code within scrollable text boxes. Also displays the relevant key variables that will change during the procedures as changeable labels, e.g. the state array, the round key, and the round number.*



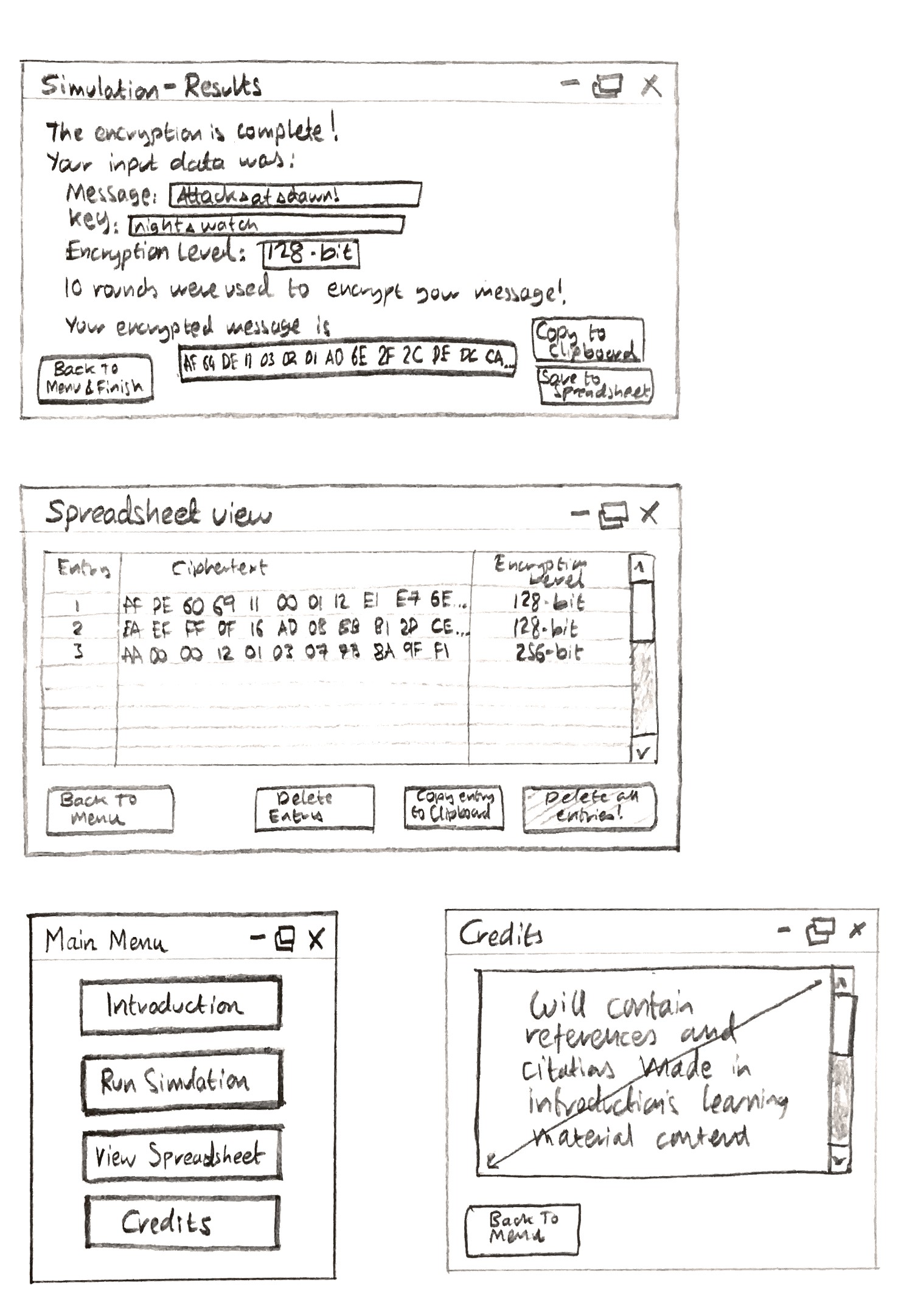
***Simulation results form***

*Displays the results of the encryption/decryption and presents the user with the opportunity to copy their encrypted/decrypted message to clipboard, and if encryption mode was enabled, save their hexadecimal string ciphertext to the spreadsheet. Original input used to encrypt/decrypt is also displayed for clarity purposes. The message, key, encryption level and the encrypted message are all displayed in immutable text boxes, with the aid of labels to describe their contents.*



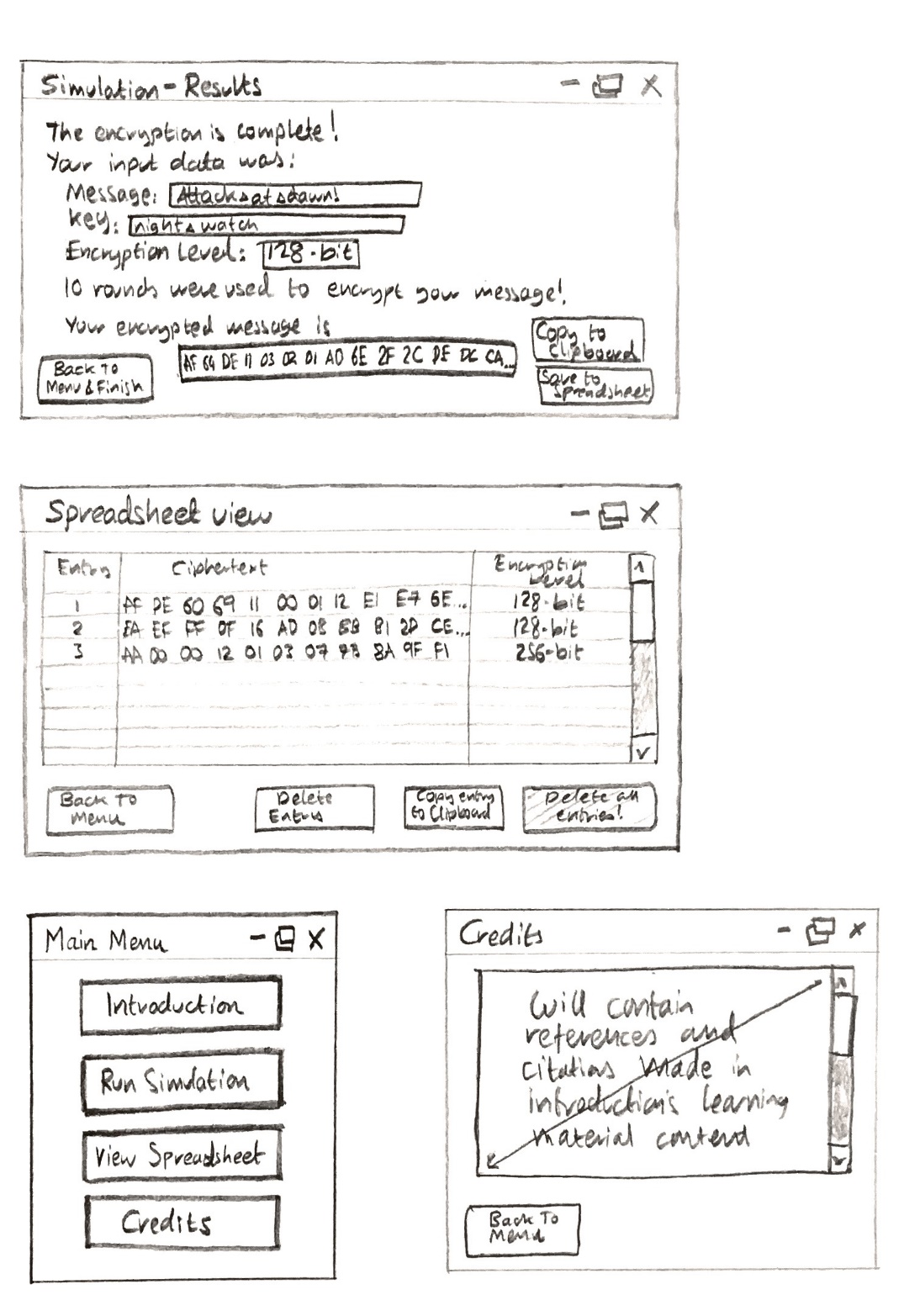
Spreadsheet viewer

*Displays the elements present within the spreadsheet i.e. the encrypted messages, their encryption level, and the entry number in the spreadsheet in a scrollable table. The encrypted message and the encryption level are the only two pieces of information required to successfully decrypt a ciphertext message. User has the options to delete a highlighted entry, copy the entry to clipboard, or refresh the list and delete all entries.*



The main menu & credits

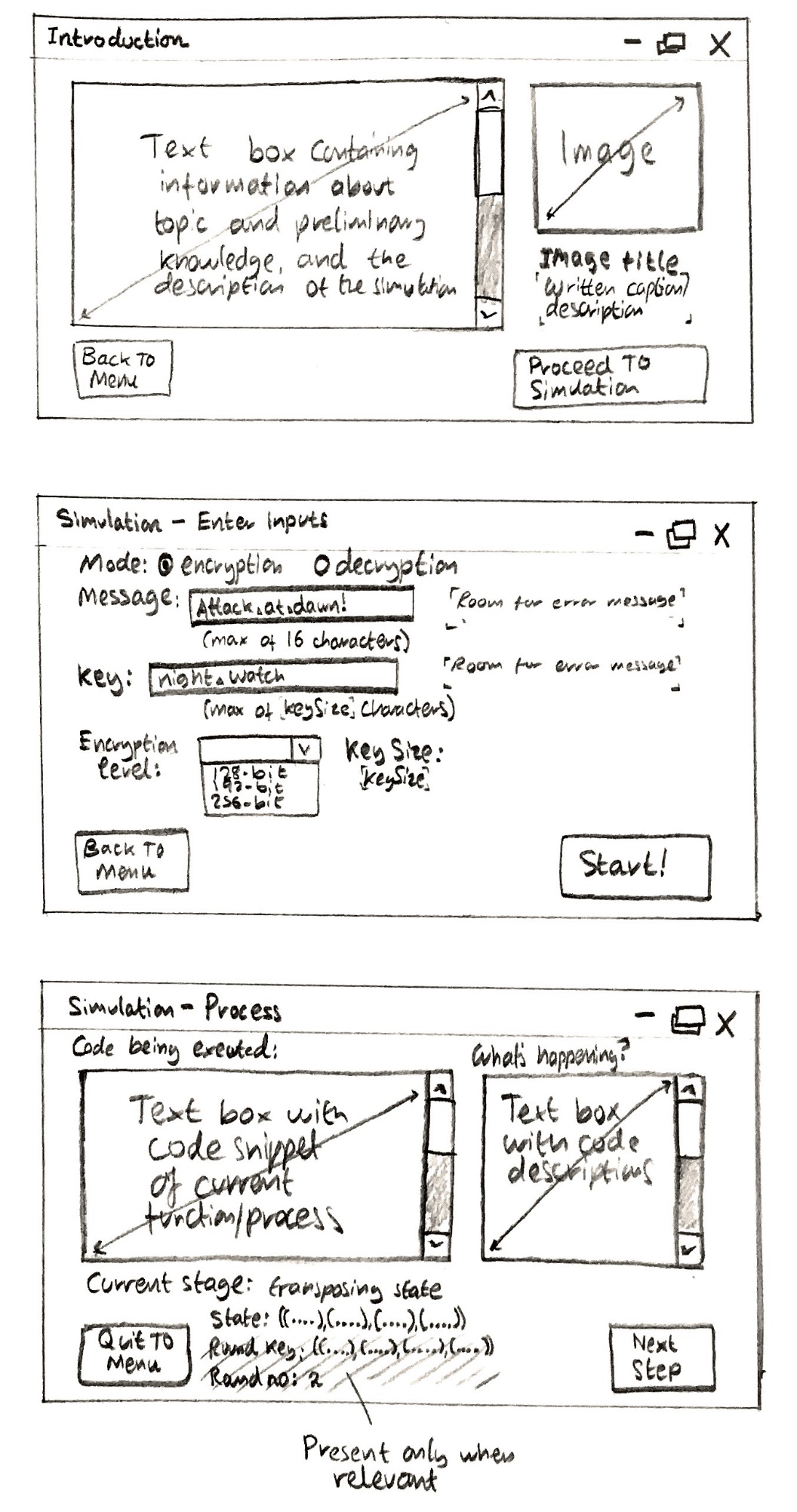
*The main menu presents the user with four options: to run the introduction page, to step right into the cipher simulation, view the spreadsheet of encrypted messages or view the credits. This main menu can be accessible throughout any part of the application using a permanent button in each other form, which resides in the bottom-left of the window. The credits window displays the credits (if necessary) in a scrollable text box for the code used and the references/citations made during the description of block ciphers in the introduction page.*



The introduction page

*The user is presented with a scrollable text box containing the preliminary knowledge and information about encryption and block ciphers before they are ready to proceed with the simulation. An image box with room for a caption label is placed as a neighbour to the text box, and can display images that can help with the user’s comprehension of the text-box material, such as displaying a state matrix during the definition of the state.*

*The user will be briefed shortly about what the simulation will consist of, as well as important instructions for data-entry, data interpretation and saving the hexadecimal output during encryption.*



|  |  |  |  |
| --- | --- | --- | --- |
| Requirement objectives met: | ***RO.1*** | ***RO.2*** | ***RO.3*** |
| *Contextual description of the objectives met in this section* | The structure of the forms has been fully described. | The forms are the components of the user interface, and by being sketched have set out the blueprints for the development in the IDE. | The forms have been designed to aid the user in understanding the cipher, as was explained in the problem definition section at the start of this document. |

# Key variables and data structures

In this section, the precise description of the variables that make up the program will be explained.

The conventional notation of capitalising the identifiers of functions and using camelCase for variables will be used to differentiate functions from variables in the program. This will make the code easier to read and trace through later during development. Library functions and variables will be left as they are – they usually have their own prefixes for identifiers and it is best left that way to avoid confusion for experienced programmers that may be familiar with them.

Regarding libraries for the cipher code, no external libraries may be utilised or required as they exist natively within the Visual Studio 2015 IDE. Many native methods that will be used belong to the System class, and consist of the following:

* Console – simple input and output for debugging
* Int32 – for parsing a decimal byte number and converting it to its binary form as a string

The following main variables throughout the cipher program will now be shown.

‘state’

A 4x4 two-dimensional array of type byte which the bytes of the plaintext/ciphertext are placed onto, ready for encryption or decryption. The second dimensions represent the columns of the state matrix, and the position of the elements within the columns represent the rows.

‘sbox’

Another 4x4 two-dimensional array of type byte, whose elements will be XORed with the corresponding elements of the state, during encryption and decryption to make those elements unique to the others within the state during each round

‘encLvl’

The level of encryption, in bytes, as an integer. It can either be 128, 192 or 256, and determines the key size for the encryption.

‘keySize’

The amount of characters, or bytes, that the key will contain. An integer determined by encLvl. The keySize is determined by the following rules:

* If encLvl = 128, keySize = 16 (128 bits/8 = 16 bytes)
* If encLvl = 192, keySize = 24 (192 bits /8 = 24 bytes)
* If encLvl = 256, keySize = 32 (256 bits /8 = 32 bytes)

It is much more convenient to work in terms of bytes required than having to divide the total number of bits all the time to figure out how many bits are in the key. This will be useful later on, when the keySize is used to determine the number of rounds required.

‘rounds’

The amount of iterations the main encryption/decryption processes repeat upon the state, as well as the number of round keys that are generated within the key expansion process. The number of rounds are dependent on the value of the keySize:

* If keySize = 16, rounds = 10
* If keySize = 24, rounds = 12
* If keySize = 32, rounds = 14

The more rounds performed, the more secure the encrypted message is.

‘plaintext’

An array of 16 bytes which represent the ASCII values of the characters in the message. This identifier is sometimes overloaded by the output of the decryption function, but no real effect will be observed since this variable has no relevant purpose/is undefined during the decryption process. This array will be iterated through during initial encryption stages to obtain the character bytes to populate the state array with.

‘ciphertext’

An array of 16 bytes which represent the ASCII values of the characters in the encrypted message. This identifier can sometimes be overloaded in the same manner that the plaintext identifier can be. During decryption, this array is used for the same purpose as the plaintext array is.

‘key’

A string containing the original key which is typed by the user as a password to encrypt the message with, and can be used during decryption alongside the keySize to obtain the original message. This string is iterated through during the key expansion process and used to create duplicates which can then be modified to generate round keys.

*Each of the above variables is declared publicly at the beginning of the block cipher module, so that they can be accessed and modified throughout the program.*

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement objectives met: | ***RO.1*** | ***RO.2*** | ***RO.3*** |
| *Contextual description of the objectives met in this section* | The key variables have been described in full. | The key variables may be output to the forms system, so it is important they are defined correctly. | The key variables can let the user know the changing values of essential features, such as the state matrix during encryption rounds. |

# Testing and development approaches

To effectively test the program and make sure that the encryption and decryption ciphers are fully functional, the test data and techniques must carefully be considered at each development stage. During iterative development, the program will be developed in three stages:

* Stage 1 – A console application containing a fully functional Conceptual Block Cipher no. 1 algorithm, including key expansion, encryption and decryption procedures
* Stage 2 – A working windows-form application which contains navigable menus and suitable data capture entries
* Stage 3 – An implementation of the algorithms from the console application with the windows-form application to create the final program

Due to the nature of these three components, separate testing procedures will be carried out separately to ensure that each component is developed effectively. Testing for each respective stage will be carried out as follows.

Stage 1

Various values of valid data will be input to the console application for evaluation. It is important to note that at this stage, data validation testing is not important as the key focus is developing a working cipher. The developer will be entering correct data into the encryption and decryption algorithms as an implementation of black box testing. If a certain feature doesn’t work or a control structure isn’t reached, white box testing can be implemented by placing markers within the code which output a message to the console to let the developer know if the program has reached that point.

Different combinations of upper case, lower case and alphanumeric values will be input to the cipher with varying key lengths to check that the message can be encrypted correctly. The output will be input to the decryption cipher, and the program will be evaluated to discover whether the encrypted message can be fully decrypted, or whether the algorithm needs more adjustment.

Encrypted messages under a certain encryption level will be decrypted with the wrong encryption level to prove that the key expansion process has a direct effect on the calculation of the state during several rounds. Messages will also be decrypted using the wrong key to demonstrate the importance of remembering the password.

Stage 2

For this stage, the focus is creating a fully working windows-form system, so buttons will be tested to see whether they navigate to the correct page, text boxes will be evaluated to see if the maximum character limit works (e.g. maximum of 16 characters for the message input text box) and lists will be evaluated to see if they can be populated.

A spreadsheet implementation will also be tested, with functions for retrieving, updating and deleting data on the spreadsheet from the spreadsheet window in the program will be evaluated. The spreadsheet file will be checked to see whether it was effectively updated by the program.

Stage 3

This stage is the final development process. With the working cipher and windows-form elements combined, the whole program will now be tested. Rigorous data validation tests will be performed on data capture sheets, such as forbidden characters being entered for the message and key strings. Hexadecimal strings will be partially pasted into the decryption text box, observing how an error message is generated through the array index of the parsing feature is going out of bounds. Other data to be used for testing this stage will be the spamming of buttons (quickly pressing buttons in succession) to discover whether the program may process the wrong response at the wrong time. Lastly, tests used in Stage 1 will be implemented into the new data capture forms, and the results will be evaluated to judge whether the windows-form application needs to be amended.

For clarification purposes, the following data will be tested in every data capture element where possible:

* Valid data which is expected for the element to receive as input.
* Valid extreme data such as a maximum or minimum value, e.g. one character for the message input string, or a key consisting of 24 characters for an encryption level of 192-bit.
* Invalid data such as a 200-character message string input to the message text box.
* Invalid extreme data i.e. a 17-character message string, or a 0-character key string.
* Erroneous data of the wrong data type. This type of data will be difficult to test, because most data capture elements permit the input of strings, which may contain any character input from the keyboard. Instead, this stage may be bypassed and a test of using the wrong keys and key lengths for encryption and decryption may be carried out instead.

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement objectives met: | ***RO.1*** | ***RO.2*** | ***RO.3*** |
| *Contextual description of the objectives met in this section* | Specifying data testing ensures the program is working with valid correct values for input and output. | - | - |

Development

The development cycle method that will be chosen for this program is the Rapid Application Development (RAD) process. Due to the time constraints and the relatively small nature of the program (there is only one standalone feature of the program, the encryption and decryption block cipher) this application can be developed quickly in a short time-frame using prototypes. The program will be modular in nature, so small issues in code or slight design changes can be accounted for and adjusted within each prototype to ensure minimal time is wasted that there is clarity within the development stages and features of the project.

For the planned development cycle, the following stages will be considered:

* Stage 1 - Development of the working cipher
* Stage 2 - Development of the forms
* Stage 3 - Implementation of the cipher with the forms

The cipher will be developed as a standalone program so that it can be effectively debugged and adjusted with minimal editing to the user interface. The native console feature in Visual Studio is an excellent form of debugging.

Primary testing methods used during development

Black-box testing will be used throughout the development and testing of the cipher to ensure that it can effectively encrypt and decrypt data. Important use of black box testing will be when observing elements of arrays such as hexadecimal to ASCII character output and monitoring key expansion elements.

White box testing will be used by placing text output marking a specific point in the program has been reached – i.e. to minimise logic errors, markers in code will be placed to let the developer know that a specific stage has been reached, so that they can move on to adjust the rest of the program.

A brief description of the cipher routine

The cipher can take three parameters: the 16-character message to encrypt, the level of encryption which determines the key size, and the key used to encrypt the message which has a maximum length of the key size chosen.

The encryption cipher expands the key using the key expansion procedure, and fills the state with the ASCII representation of the characters input, as type byte. The output of the encryption cipher will be a 16-number hexadecimal string composed of 47 characters (32 hex digits and the spaces between them), which is composed of 16 two-digit hexadecimal values. These values represent the ASCII codes of the characters in the ciphertext, some of which are out-of-range of the readable ASCII values so their values are retained through not being converted to ASCII.

The input of the decryption will be these very same hexadecimal digits, and a function parses the hexadecimal string, converting every element in pairs that is not an empty space back into its relevant ASCII value. These ASCII values are then input to the state and evaluated through reverse functions of the encryption routine. The same expanded key is used for XORing the state during encryption and decryption, so it is important that the user remembers the exact key and key size they used to encrypt their message with before attempting to decrypt it. The output of decryption is an ASCII character string, much like the input that the encryption routine took.

This method is an effective way of encrypting a 16-character (or below) message and decrypting it, without losing the ASCII values upon storage. The user must make sure they perform the following to correctly encrypt and decrypt a message:

* Message must not exceed 16-characters
* Size of the key must not exceed the amount of characters the encryption level holds (128-bit has a maximum of 16 characters, 192-bit has a maximum of 24, and 256-bit has a maximum of 32)
* The user must remember/make note of their key and the encryption level
* To decrypt, the user must carefully copy the hexadecimal string that was generated by encryption up to the last hexadecimal digit, excluding any spaces at the end of the string. This will make sure the string is 47 characters long and valid data for the decryption input
* The user must enter their key, case-sensitive, and the encryption level they used to encrypt it

*Appropriate data-validation measures will be implemented within the form-system to ensure that the user enters their input correctly, and that the encryption/decryption ciphers do not receive the wrong data-type or value of data.*

The form system

As noted above, the form system will be responsible for ensuring the user has entered the correct data to the cipher. The form system is also responsible for containing the introduction to the topic of block ciphers and providing the background information to the topic for the user to fully comprehend the cipher process. During encryption or decryption, the form system will display the code executed in a text-box window, as well as a description of the code and its purpose in the context of the cipher process. Key variables such as the state, the round key, and the round number will be displayed so the user can trace the crucial variables at each stage of the encryption. They will be able to trace the code themselves to work out how the variables are generated, if necessary.

Interrupts will be placed at each major part of the encryption process, so that the relevant key variables, code snippets and description of the process can clearly be explained. The user can read this information and then click a button to proceed to the next stage. The form content will be refreshed and populated with the new information from the next stage of encryption.

The following processes will be subject to interrupts being placed for observation within the form:

* The inputs the user has chosen
* The conversion of the input to their relevant data types, e.g. hexadecimal to ASCII byte array for decryption, and the message string to an array of ASCII byte values for encryption input
* The key expansion procedure, with each round key being displayed and being explained which round it will be used in during encryption/decryption (mentioning that decryption uses the (rounds – 1)th key, as the operation is performed in reverse)
* The main encryption/decryption cycle, with the modified state during each round, the round key, the current round, and the values of iteration for columns and elements clearly displayed and highlighted, so that the user knows each step that is made to ensure the key is secure
* The final output, showing how the state ASCII value elements are converted to a hexadecimal string for encryption output, and how the state’s ASCII value elements are converted to their relevant ASCII characters for decryption output

With each process, the user gets a chance to observe the snippet of the code that was executed, a description of the code within the context of the cipher process and the relevant key variables needed, before they can click a button to proceed to the next process. At the end of the encryption or decryption, the user will then be able to view the results of their encryption, describing a summary of the process which will include information on:

* The original message input, i.e. the plaintext message for encryption or the hexadecimal string for decryption
* The final message, either a hexadecimal string for an encrypted plaintext message or a string of characters which contain the original message for the decrypted ciphertext
* The key that was used to encrypt/decrypt the message
* The encryption level, in number of bits, which was used to encrypt/decrypt the message
* The key size in characters, determined by the encryption level, which the key was adjusted to according to the encryption level
* The number of rounds used to encrypt/decrypt the message, determined by the encryption level
* The expanded key that was used, including the round keys which were generated for each round

The user will then be presented with an option to save their final message, should they want to check if it can be encrypted/decrypted again.

*Current limitations of the cipher: Because the maximum number of characters the encryption input can consist of is 16, a message cannot be encrypted multiple times as the encryption output dimensions are greater than the input (an encryption generates a 47-character string output). This could be amended later if the client wishes the feature to be included, by using the same algorithm that the decryption cipher uses to interpret the hexadecimal input into an ASCII array for the state input. This arbitrary feature will not be implemented at the moment, as the main purpose is to get a working cipher integrated into the forms system with the focus on the user observing each step of a single encryption/decryption.*

# Development of the working cipher (Stage 1)

The Conceptual Block Cipher no. 1 will be developed separately to the main program, using a console application for debugging. This is part of an iterative Rapid Application Development process, in which the cipher along with output for debugging will be developed as a console application for the first prototype.

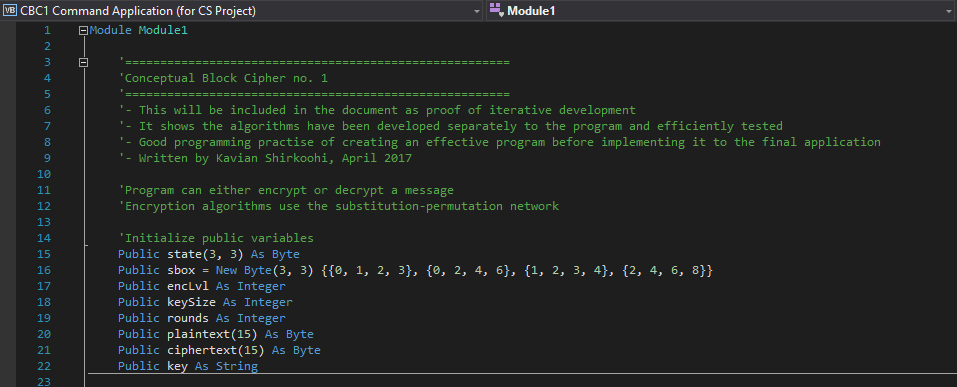
The initial creation of the cipher is documented in the section below, where screenshots of the source code for the whole console application written in Visual Basic within the Visual Studio 2015 ID is displayed and documented by comments within the code.

Source code of the cipher

This code deals with the console application of the block cipher. The user is presented with a choice at the start, typing ‘1’ for encryption and ‘2’ for decryption. For encryption, the console takes user input for the message, the key, and the key size. For decryption, the console takes the user input for the hexadecimal ciphertext string, the key used and the key size. These values are then passed to the essential control structure subroutines which navigate the control of the program towards utilising key expansion and encryption/decryption functions. The console screen displays the state with every iteration of the round so the values of each column can be observed. The final output of the console screen will be either the hexadecimal ciphertext string (for encryption) or the original key string as ASCII characters (for decryption). The user can copy the output, restart the console application, and input the string as the input for the inverse operation (e.g. output from decryption can be pasted into the input for encryption) to observe the functionality of the encryption/decryption features.

In order of the source code within the IDE, the following features from the console application will be shown and their features described within each step.

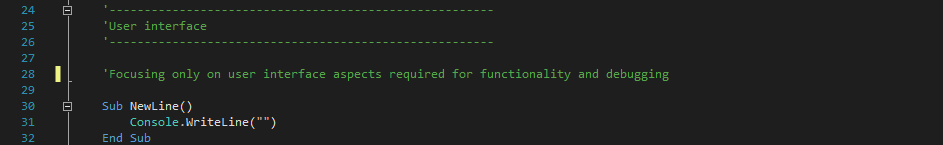
**Initializing public variables**



The public variables *state, sbox, encLvl, keySize, rounds, plaintext, ciphertext,* and *key* are initialised and prepared for usage throughout the console application. The public declaration will allow them to be modified and accessed throughout the code, so that subroutines and functions can effectively update them.

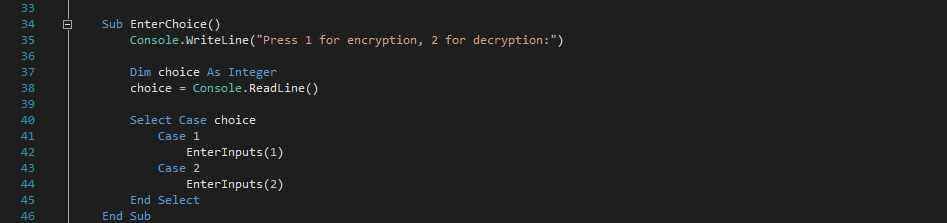
**Console user-interface subroutines**

*Sub NewLine()*



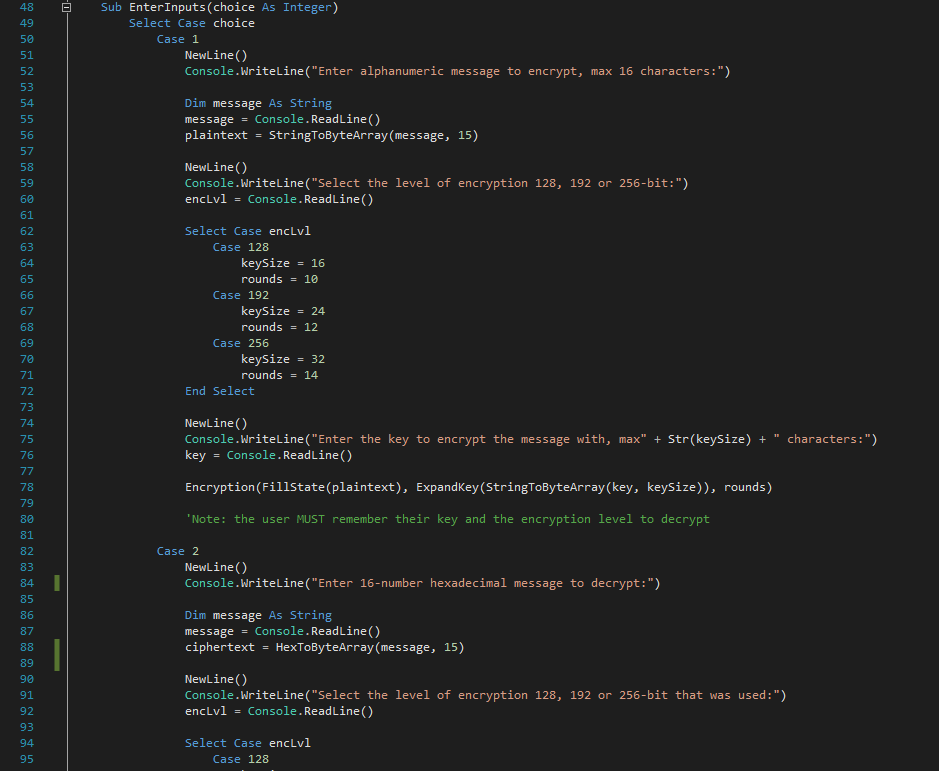
This is a simple subroutine which saves time and space in code when writing a new line to the console.

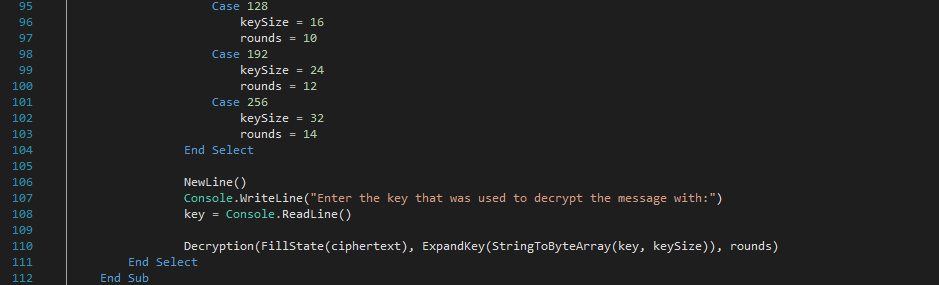
*Sub EnterChoice()*



This subroutine prompts the user to enter an integer value *choice* which is used in a select control statement to determine which parameter to pass to *EnterInputs*, which will decide whether encryption or decryption will be performed.

*Sub EnterInputs()*



This subroutine is split into two mutually exclusive parts – encryption and decryption. Through a parameter received from its call, the integer *choice* is evaluated and determines whether the console will be set to encryption mode (choice = 1) or decryption mode (choice = 2). This subroutine is responsible for receiving user input for the message to encrypt/decrypt, the key, and the encryption level *encLvl* to encrypt the message with. The total rounds variable *rounds* and key size *keySize* are determined by a select statement which evaluates the encryption level variable. Most variables are self-descriptive within their identifiers.

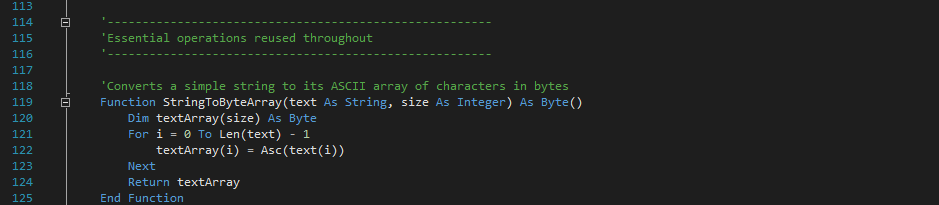
During encryption mode, the message string entered is converted to a 16-element array of ASCII bytes by the function *StringToByteArray* and assigned to the variable plaintext. The plaintext array is sent to the function *FillState* as a parameter and the key’s ASCII byte array is sent to the key expansion function, *ExpandKey*, as a parameter along with *keySize* and *rounds*.

During the decryption mode, the hexadecimal string of the message entered is parsed and converted to array of the hexadecimal digits’ corresponding ASCII decimal values through a return of the function *HexToByteArray* when it is sent as a parameter, and assigned to the 16-element array *ciphertext*. The ciphertext array is sent to the function *FillState* as a parameter. The key’s ASCII byte array is sent to the key expansion function, *ExpandKey*, along with *keySize* and *rounds* as parameters.

*Data validation methods are not present within this stage of the development as this console application is created purely for alpha testing. The user at this stage is the developer, and during entering the inputs and observing the outputs knows how the code is structured and what to expect.*

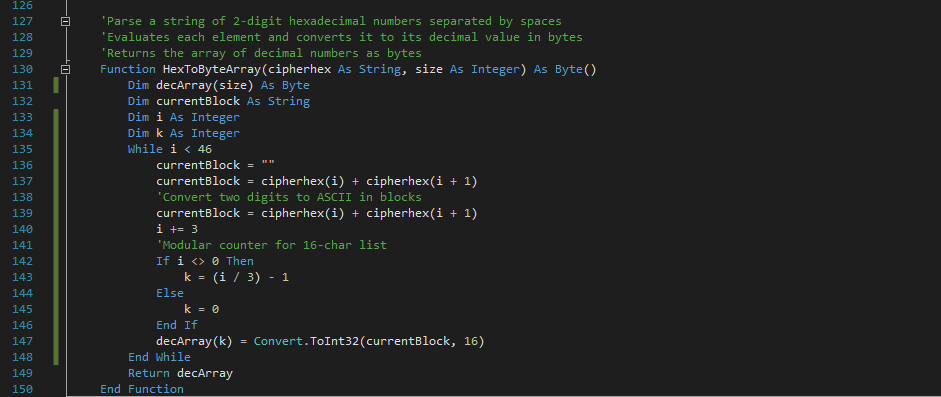
**Essential operations reused throughout the code**

*Function StringToByteArray()*



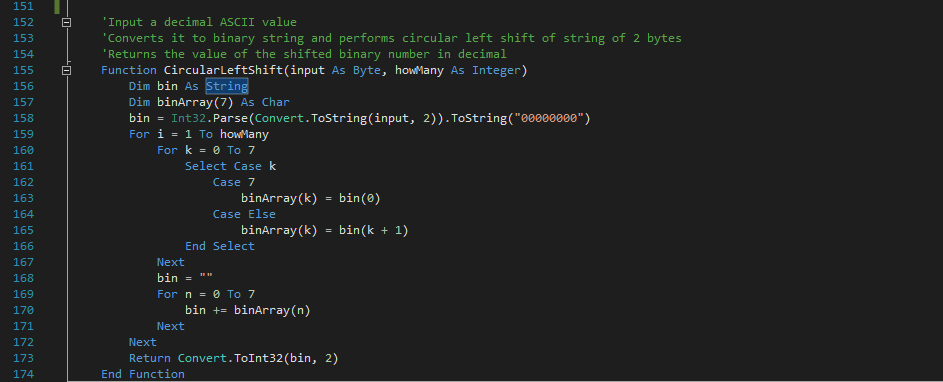
Takes a string *text* and iterates through the length of the string. Converts the individual characters to their relevant ASCII values as bytes and stores in an array *textArray* of length *size*. Returns *textArray* as a value.

*Function HexToByteArray()*



Takes a 47-character string *cipherhex* as input, which contains 16 two-digit hexadecimal numbers separated by spaces between them. Iterates through and parses the *cipherhex* string, comparing the two-digit hexadecimal numbers in blocks. Converts the hexadecimal numbers to their relevant decimal values. Adds the converted numbers to an array *decArray* of dimensions *size*, using the counter variable *i* to navigate through the parsed-string and *k* through *decArray*. Returns the array of converted decimal numbers, *decArray*.

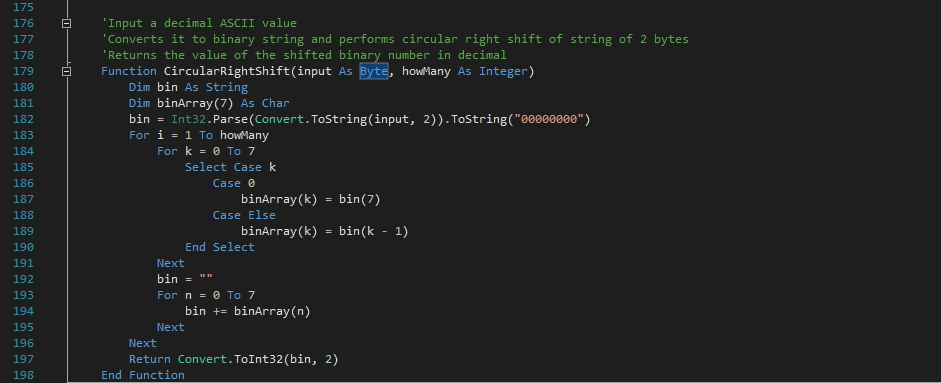
*Function CircularLeftShift()*



Takes a decimal number *input* of type Byte, along with the integer value *howMany* to specify how many places to shift the binary number. Converts the decimal number to an 8-character bit string using the function *Int32.Parse(Convert.ToString(input, 2)).ToString(“00000000”),* which is iterated through and set as an array of characters. The last element of the character array is always set to the first element in the array, with every other element being set to its right-hand-side neighbour. This effectively shifts the bits in the array by one, and the process can be repeated *howMany* times to achieve circular bit-shifting by a number larger than one.

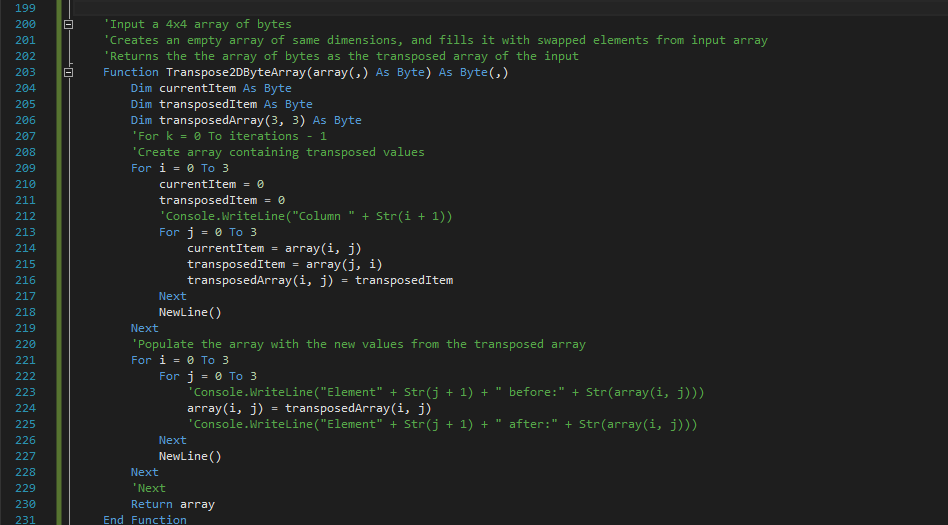
The bit characters from the array are added to the string *bin* which is then parsed and converted back from binary to its relevant decimal value through the function *Convert.ToInt32(bin, 2).* This decimal value is returned as function output.

*Function CircularRightShift()*



Does the same as the *CircularLeftShift* function, but in the reverse direction. *Parameters and return values are the same.*

*Function Transpose2DByteArray()*

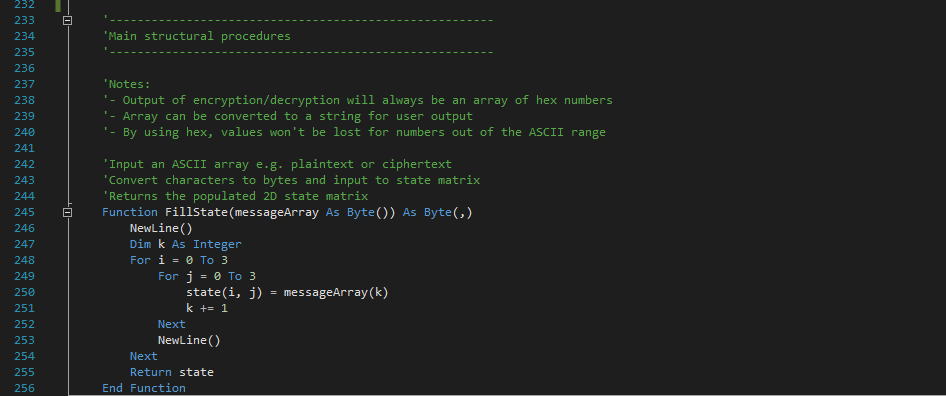


Takes a two-dimensional *array* as input, and transposes it, swapping the elements between columns. This is the same technique as transposing a 4x4 matrix, and this function was written with transpositions of the state in mind. An array *transposedArray* of matching dimensions to the *state* is created, and the elements of the input array are evaluated and the transposed equivalent (with the iteration values swapped) are stored into the *transposedArray*. The return value is the *array* which is assigned the value of the *transposedArray*.

*Important note: it is important that a separate array is created during this process in which to store the transposed elements on to, as the input array needs to remain intact so it can be used for valid reference. If the input array is modified during transposition procedures, it will provide incorrect data for the transposed array.*

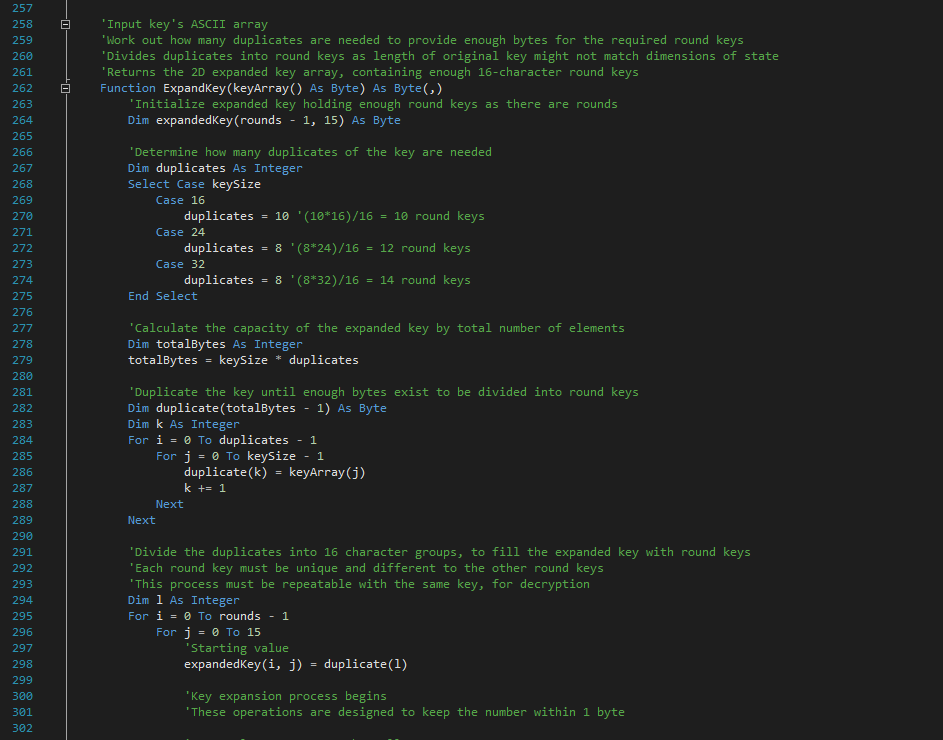
**Main procedures for encryption and decryption**

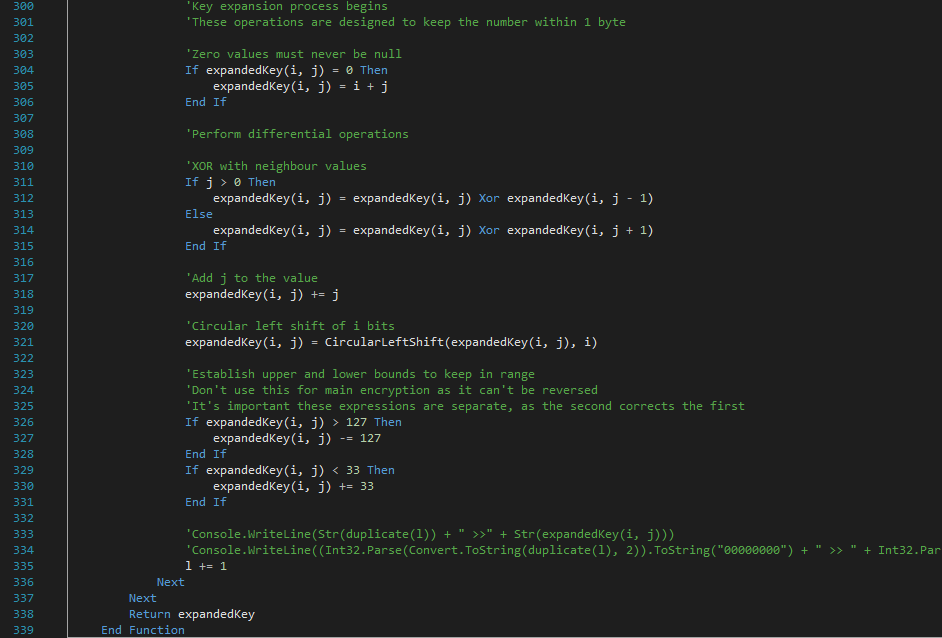
*Function FillState()*



Takes a 16-character array of ASCII bytes as a parameter, *messageArray*, iterates through this array and stores the corresponding values onto the two-dimensional array called the state. The *state* will be the main form of representing the original data that was input as the message in which it will be modified with during the encryption/decryption process. The return value is the 2D array *state*.

*Function ExpandKey()*





This function takes the array of characters of the key, *keyArray*, as a parameter. It creates a two-dimensional expanded key array of dimensions (rounds, 16) and determines the number of duplicates required to provide enough bytes to be separated into the number of round keys needed. It then creates a one-dimensional array of duplicate keys, iterates through this array and stores blocks of 16 characters into the nested arrays within the expanded key array. This effectively fills the expanded key with duplicates of the original key, ready for modification to turn these duplicates into unique round keys.

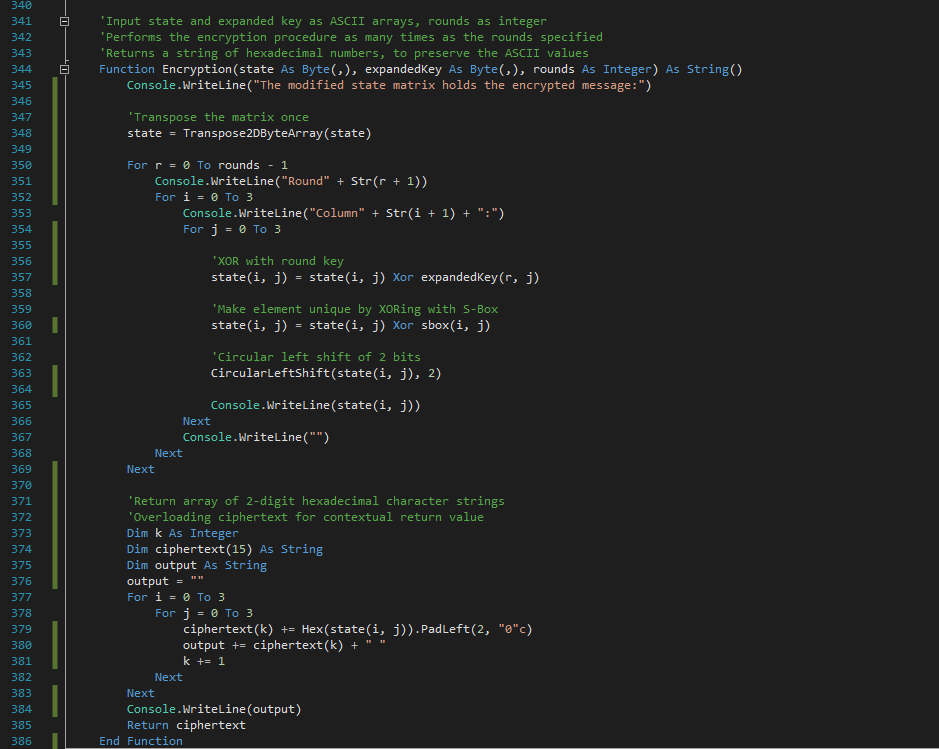
A key expansion procedure is performed to make each round key unique and different to the others, and the following operations are performed on the elements within each nested array (i, j):

* A conversion of zero values to the value of the iteration counters *i + j,* so the expanded key will never contain similar elements of value ‘0’
* An XOR with the previous element, except for the first element being XORed with the next to avoid the index reaching out of bounds
* An addition of *j* to the value of the element – this makes the elements in each round key unique
* A circular left shift of the binary value of the element by *i* times, adding another layer of unique property to the value
* An evaluation of the value to ensure it never falls out-of-range of the visible ASCII characters 0-9, A-Z etc. by adding 33 to ASCII values less than 33, and subtracting 127 from ASCII values over 127

*The above steps are designed to be repeatable with the same key, so that the exact same expanded key is obtained containing the same round keys when the same key is used for decryption.*

The expanded key array is returned, which can be used in the encryption or decryption functions as a parameter.

*Function Encryption()*



This function is the main encryption routine. The state, expanded key and number of rounds run are input as parameters. The state is transposed once, through the function Transpose2DByteArray with the state passed as a parameter. Then a for-loop up to the number of rounds specified is run to perform the main encryption procedures upon the state, in which each element within the two-dimensional state array *(i, j)* is iterated through:

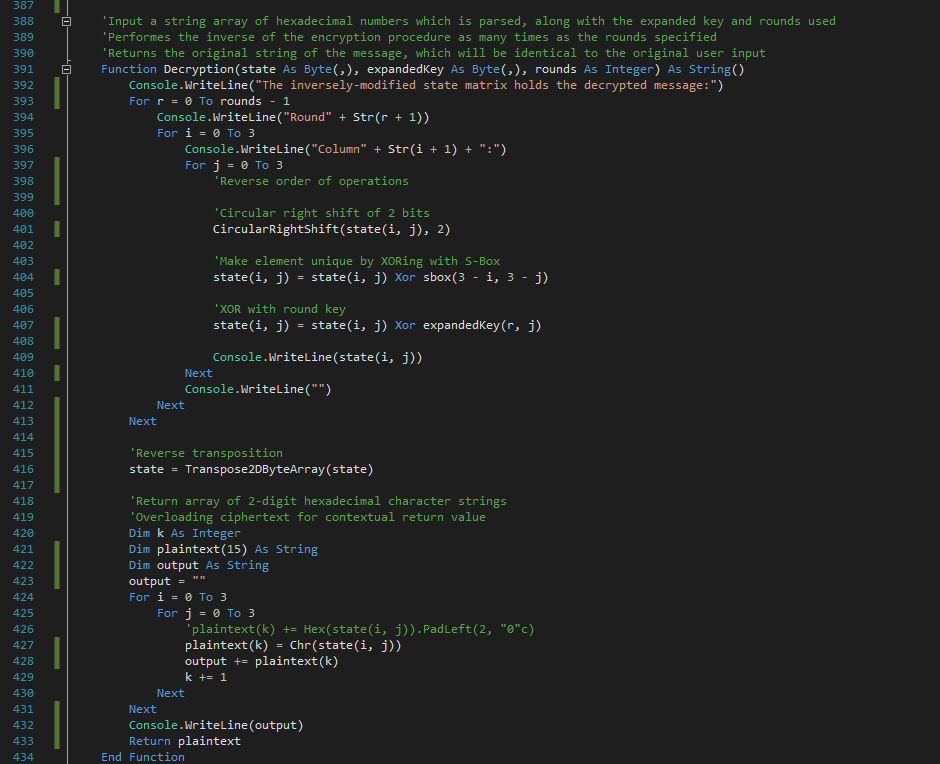
* An XOR of the current state element with the corresponding round key element, the expanded key array element of the same number as the round
* An XOR of the state element with the corresponding S-box value
* A circular left shift of the state element by 2 bits

After the state has been fully modified for the number of rounds specified, it can now be considered to hold ‘encrypted’ bytes of the original message. These bytes must be output as hexadecimal digits in a certain format, to retain their ASCII values and to be easily parsed for decryption. The main purpose of using hexadecimal for the output is that the numbers are guaranteed to be represented by two-digits since the maximum ASCII decimal value the state element would hold is 255 = FF in hexadecimal, making it possible to parse the string later to retrieve the ASCII values. This string can be copied to clipboard by the user or stored as a string on a spreadsheet, for example, without the risk of the values of the ASCII bytes being truncated or lost.

To achieve the conversion to hexadecimal, the whole state is iterated through for each element, and converted to its hexadecimal equivalent using the function *Hex().PadLeft(2, “0”c)* where the *PadLeft* call ensures that numbers less than 10 are two digits and of the form 01, 02, 03 etc.

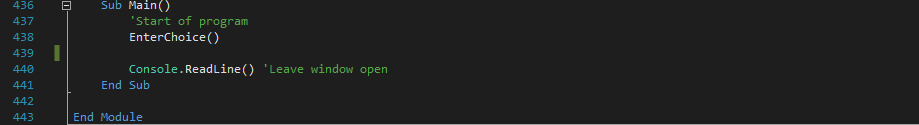
The return value is the string *ciphertext*, which is a concatenation of each element of the two-digit hexadecimal value array of ASCII numbers.

*Function Decryption()*



This function works in the exact same manner that the encryption function does, except the order of operations is carefully changed to ensure the exact inverse is performed. The return value of this function is the string *plaintext*, which is a concatenation of the string of the ASCII characters of the byte values within the state.

*Sub Main()*



The main execution space of the console application. A simple call to the subroutine *EnterChoice* is required to start the whole encryption/decryption program.

Testing the cipher

The console application will be used to ensure that several messages ranging up to 16-characters can be successfully encrypted and decrypted by the block cipher algorithm, utilising black-box testing methods to observe the output produced by the input. A variety of key lengths will be used for the same message to ensure that each encryption level is successful in encryption and decrypting a message. A few slight modifications to the code within the console application will be made to omit debugging output commands for clarity of testing, such as the Console.WriteLine() functions displaying the output of the state at each round.

The input and output for the console application will now be demonstrated for several permutations of the message and key chosen:

* Case 1 – lower case message, lower case key, 128-bit
* Case 2 – upper case message, lower case key, 128-bit
* Case 3 – upper case message, upper case key, 128-bit
* Case 4 – lower case message, upper case key, 128-bit
* Case 5 – alphanumeric message, alphanumeric key, 128-bit
* Case 6 – a message containing special characters, with a key containing special characters, 128-bit
* Case 7 – an alphanumeric message, containing an alphanumeric key, for 128-bit, 192-bit and 256-bit configurations. This case demonstrates that all key configurations are capable of encrypting and decrypting the same message, regardless of the different decrypted message hex string that the encryption generates
* Case 8 – an alphanumeric message, alphanumeric key, 192-bit encryption but entering the wrong key on decryption. This case demonstrates how entering the wrong password for decryption makes the message inaccessible
* Case 9 – an alphanumeric message, alphanumeric key, 256-bit encryption but upon decryption an encryption level of 128 is entered instead of 256. This case demonstrates how important the key expansion procedure is in producing suitable round keys for encryption and decryption, as if the wrong key size is chosen, insufficient round keys are generated

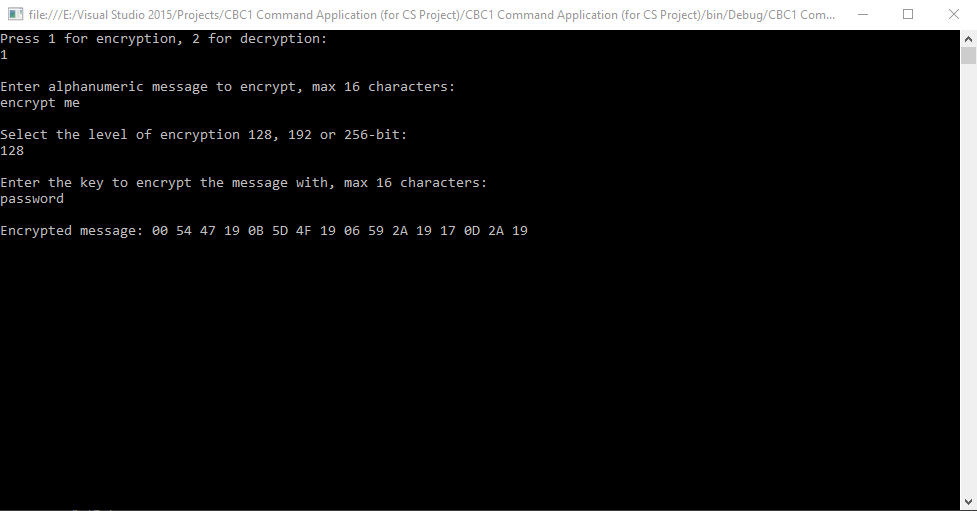
**Case 1**

Message: “encrypt me”

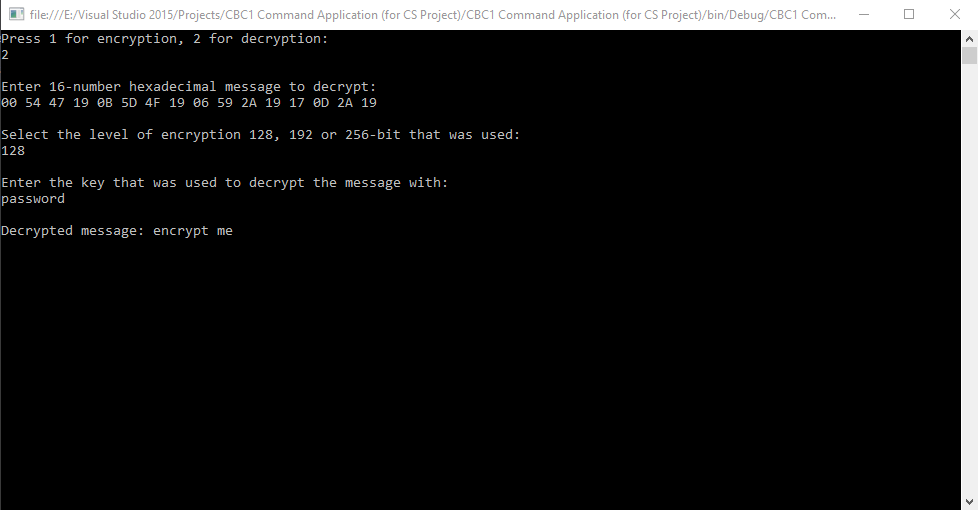
Encryption level: 128-bit

Key chosen: “password”

*Encryption result:*



*Decryption result:*



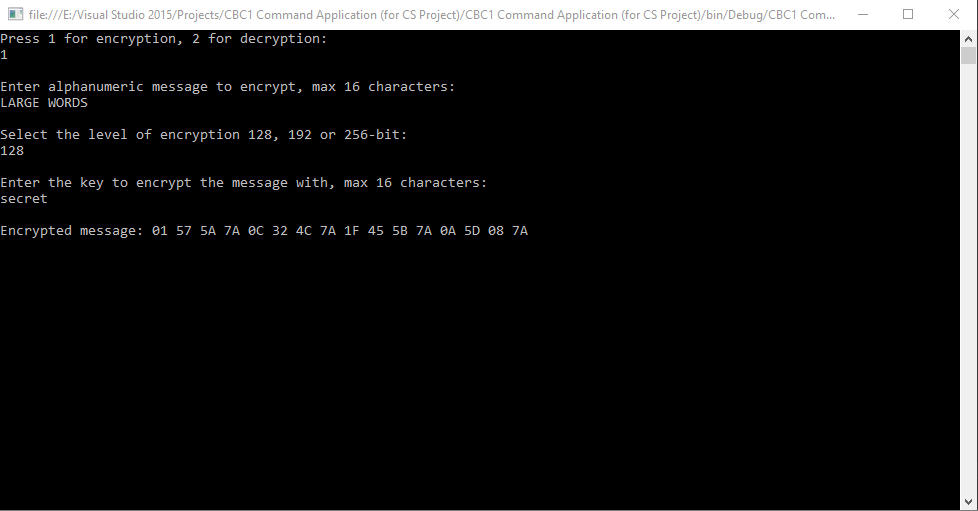
**Case 2**

Message: “LARGE WORDS”

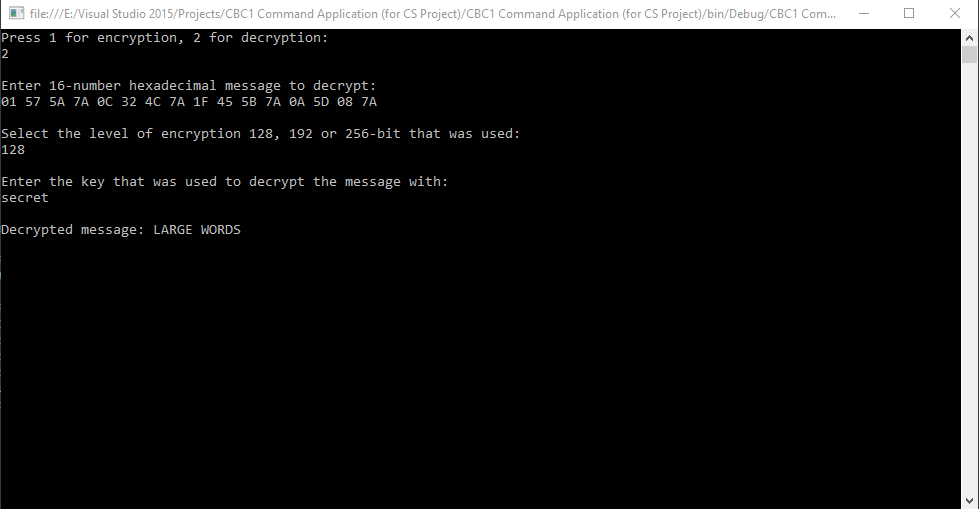
Encryption level: 128-bit

Key chosen: “secret”

*Encryption result:*



*Decryption result:*



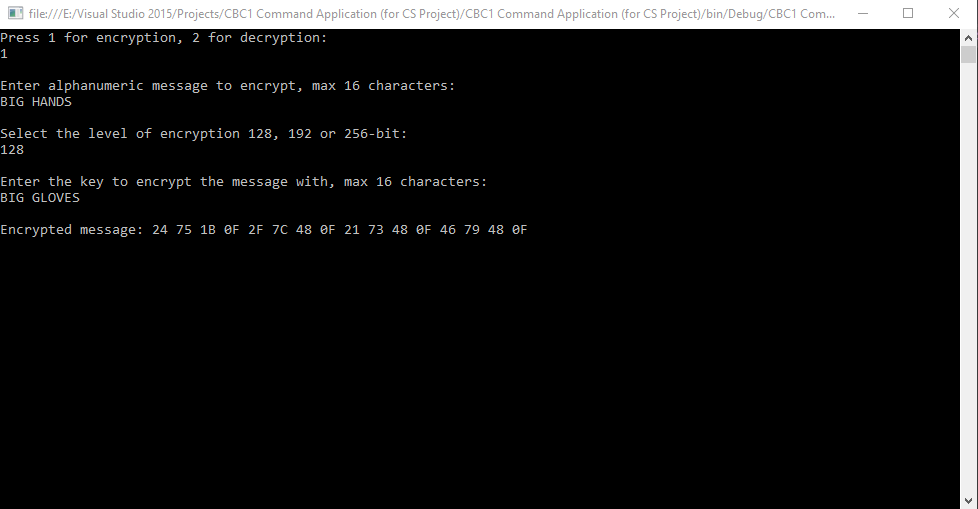
**Case 3**

Message: “BIG HANDS”

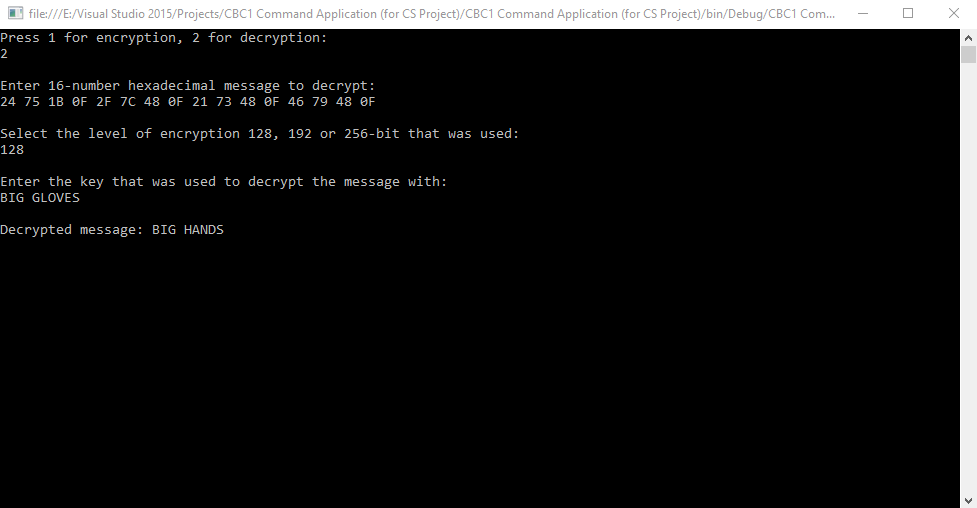
Encryption level: 128-bit

Key chosen: “BIG GLOVES”

*Encryption result:*



*Decryption result:*



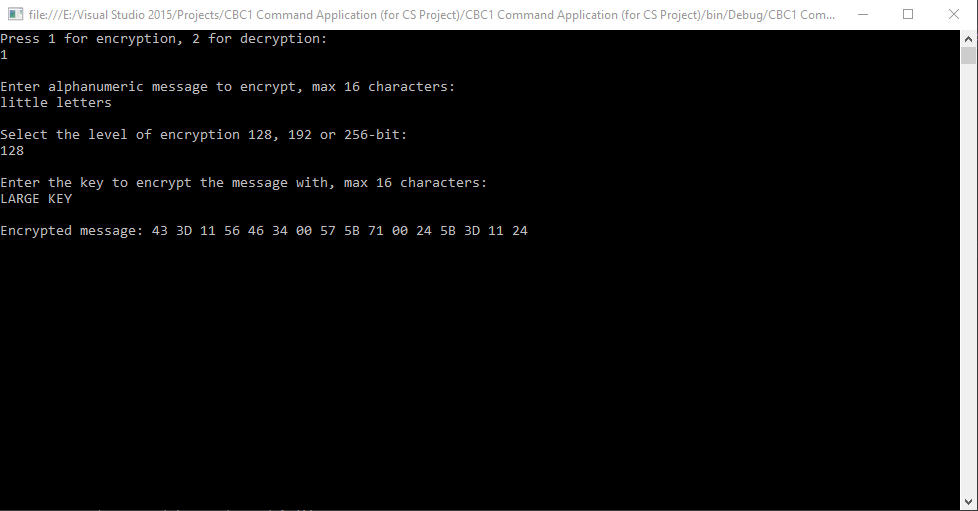
**Case 4**

Message: “little letters”

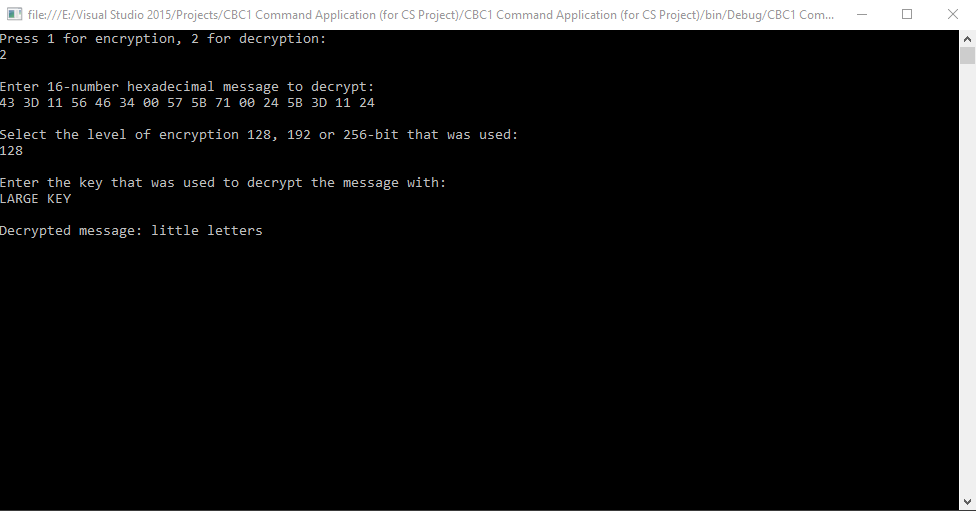
Encryption level: 128-bit

Key chosen: “LARGE KEY”

*Encryption result:*



*Decryption result:*



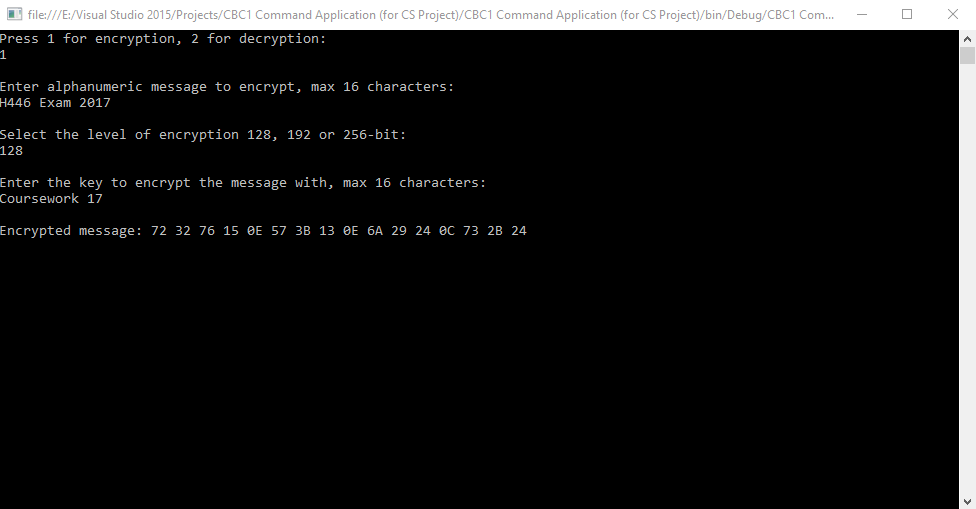
**Case 5**

Message: “H446 Exam 2017”

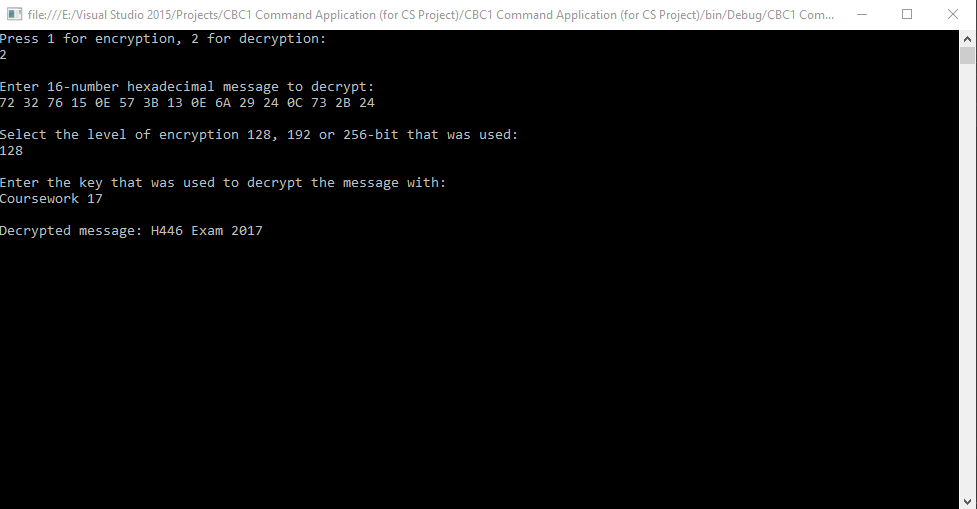
Encryption level: 128-bit

Key chosen: “Coursework 17”

*Encryption result:*



*Decryption result:*



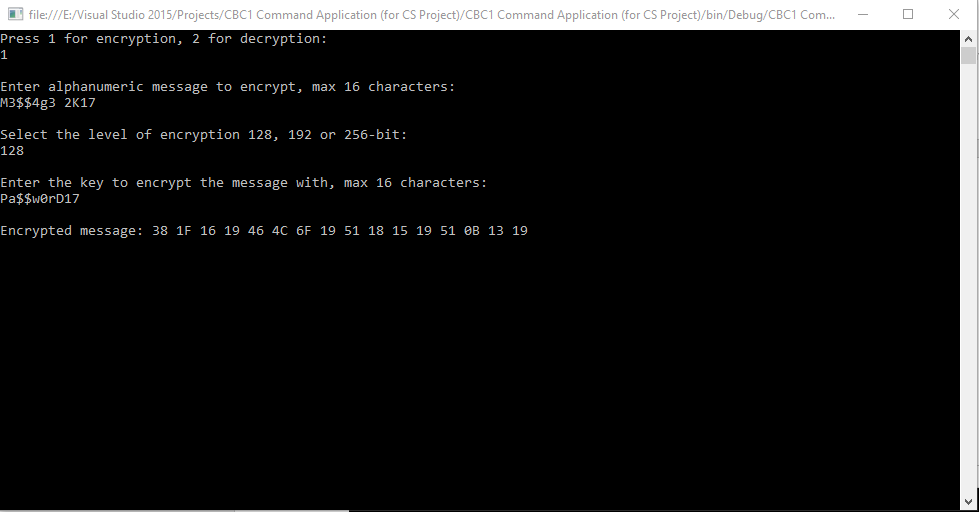
**Case 6**

Message: “M3$$4g3 2K17”

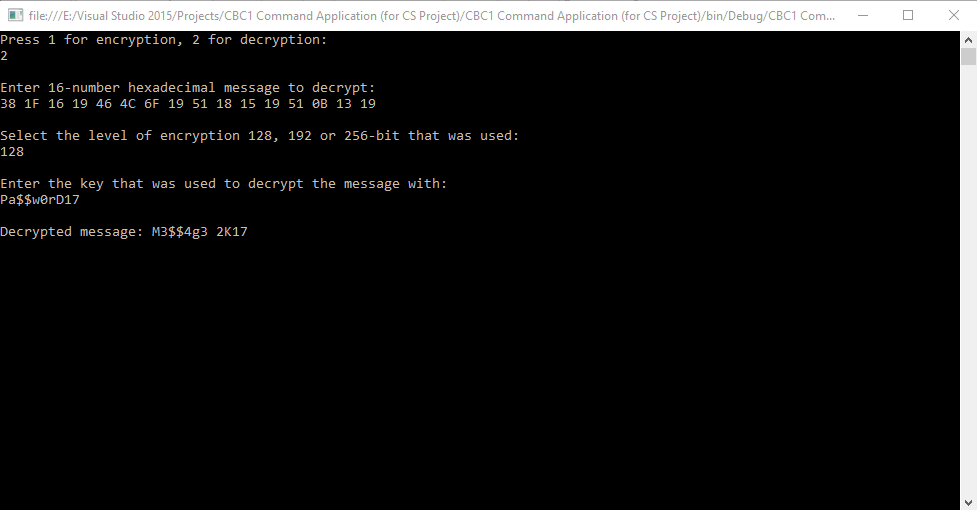
Encryption level: 128-bit

Key chosen: “Pa$$w0rD17”

*Encryption result:*



*Decryption result:*



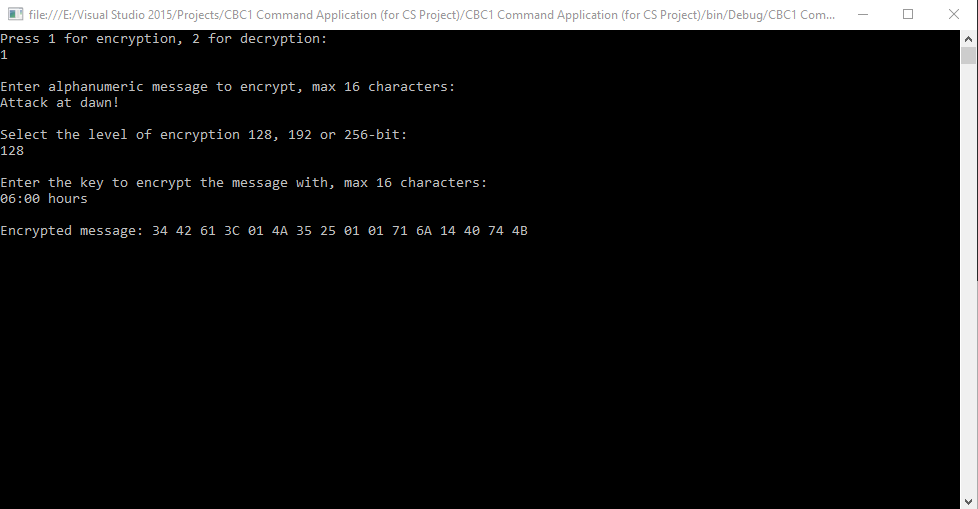
**Case 7**

Message: “Attack at dawn!”

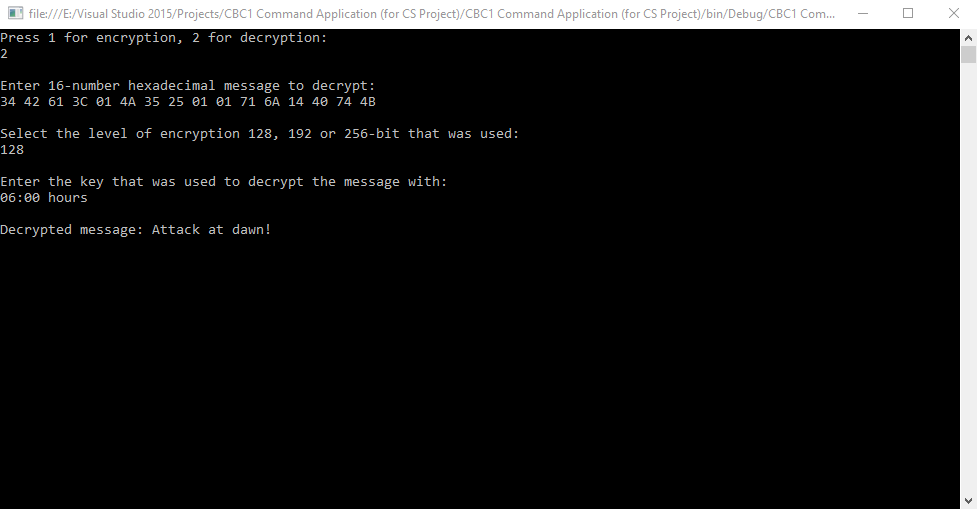
Encryption level: 128-bit

Key chosen: “06:00 hours”

*Encryption result:*



*Decryption result:*

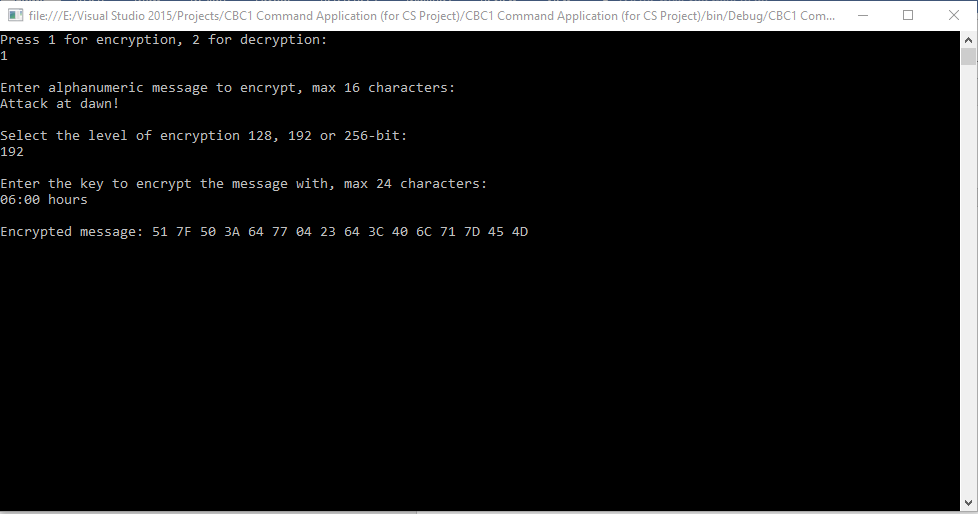


Message: “Attack at dawn!”

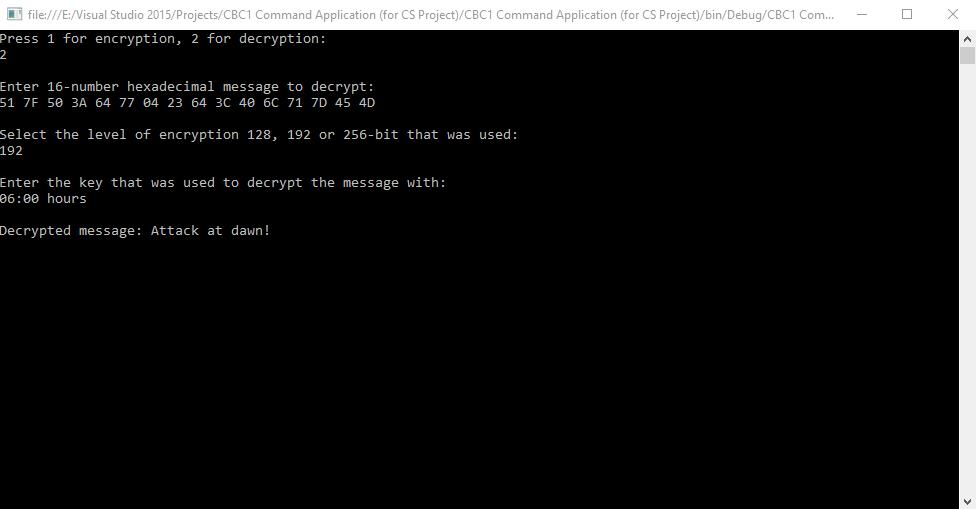
Encryption level: 192-bit

Key chosen: “06:00 hours”

*Encryption result:*



*Decryption result:*

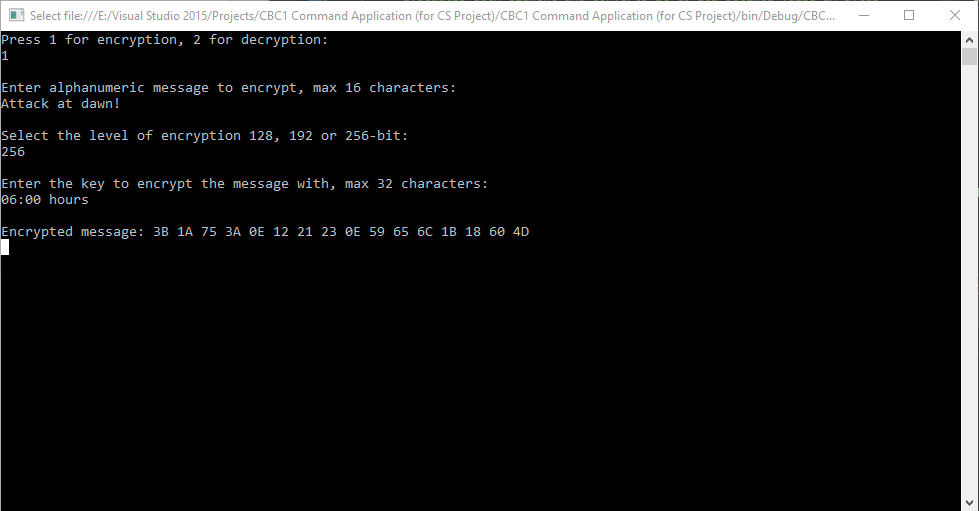


Message: “Attack at dawn!”

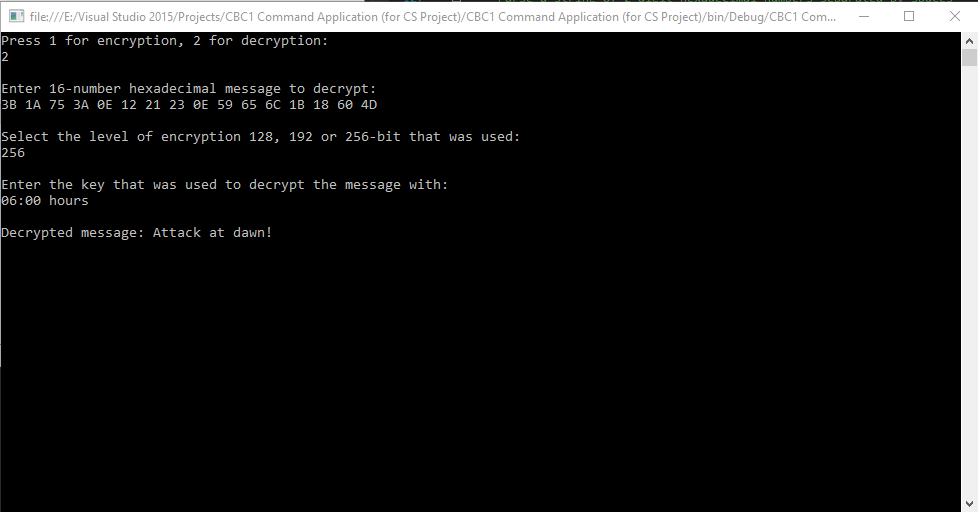
Encryption level: 256-bit

Key chosen: “06:00 hours”

*Encryption result:*



*Decryption result:*



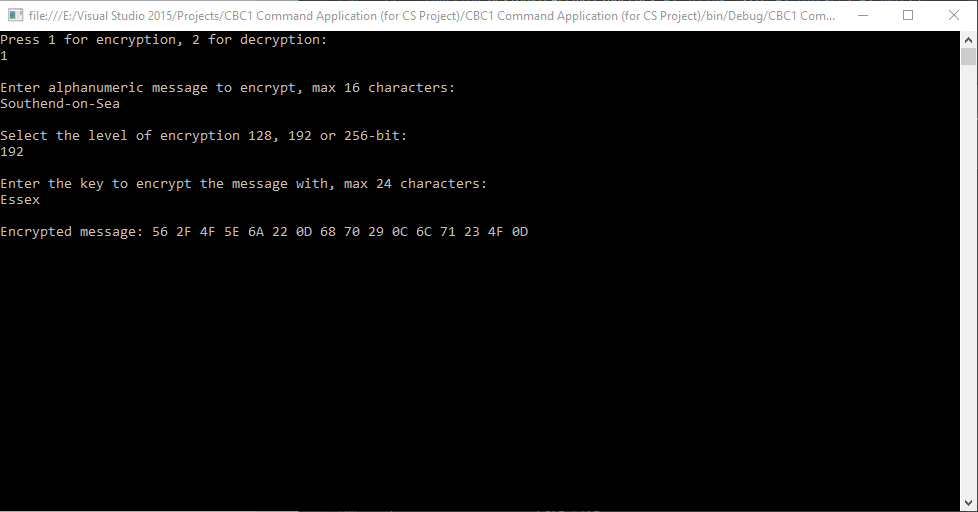
**Case 8**

Message: “Southend-on-Sea”

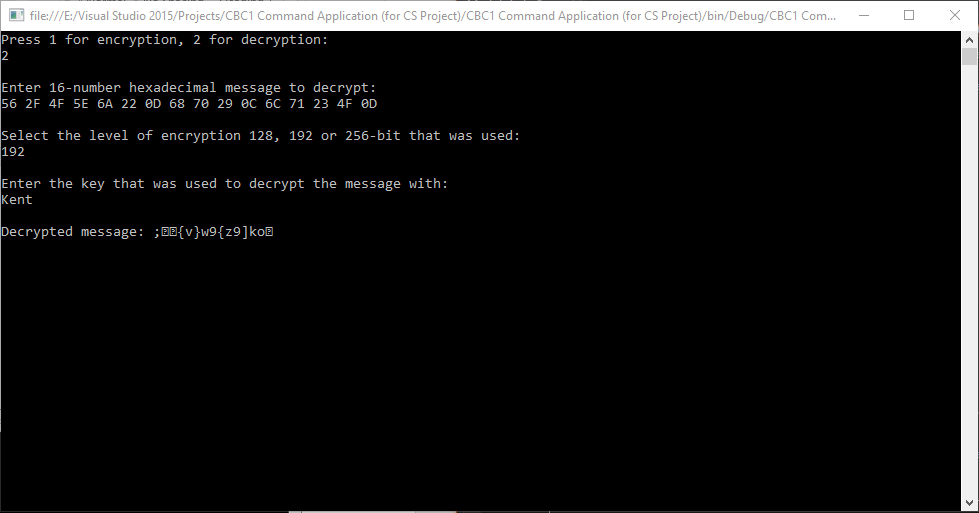
Encryption level: 192-bit

Key chosen: “Essex”

*Encryption result:*



*Decryption result:*



As observed, the decrypted message is unreadable and provides no useful information about the original message. The key is very important during decryption, as it is the only way a message can be decrypted.

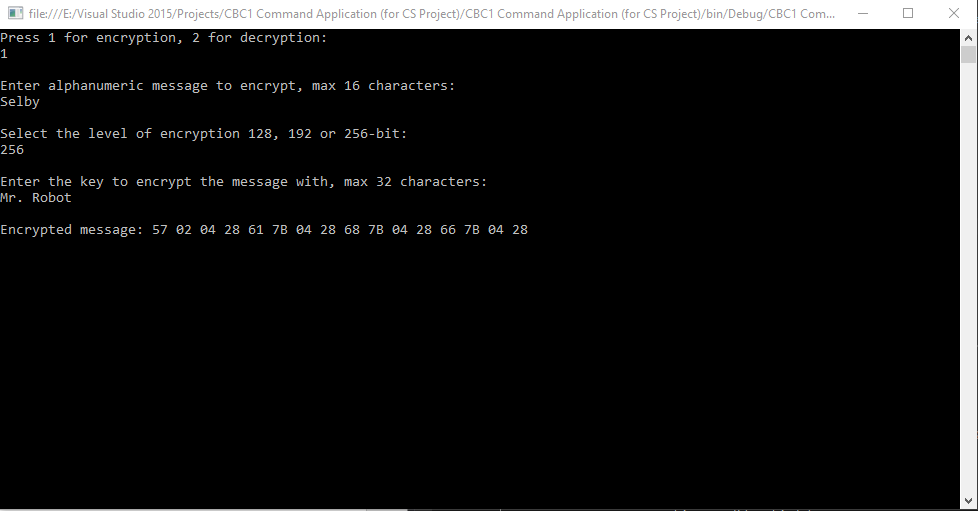
**Case 9**

Message: “Selby”

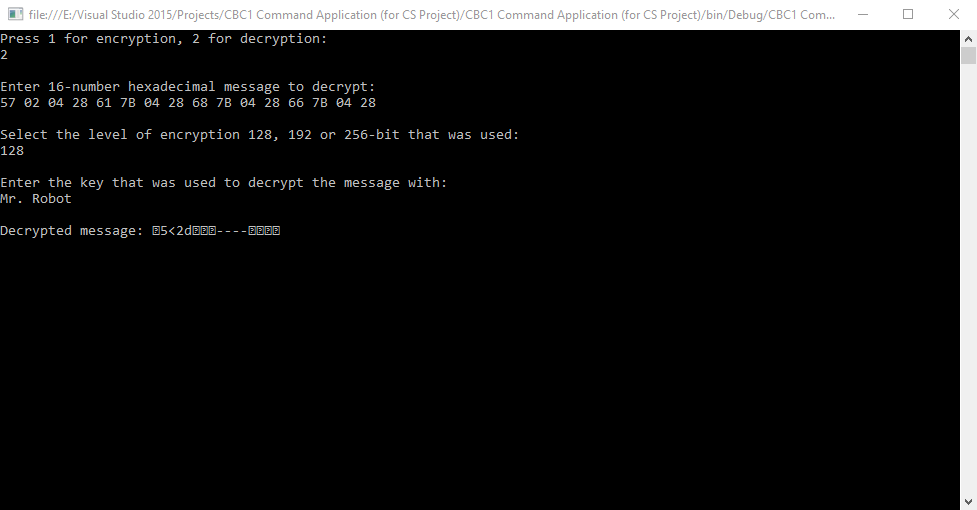
Encryption level: 256-bit

Key chosen: “Mr. Robot”

*Encryption result:*



*Decryption result:*



Like the previous case, the decrypted message is unreadable and provides no useful information about the original message. The correct encryption level is required to correctly decrypt a message, as it specifies how the key will be expanded to match the same expanded key that was used during encryption. Without the correct round keys, the state will never be reverted to its original value.

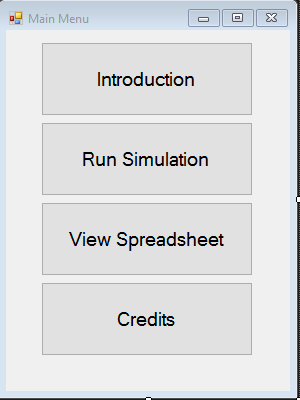
|  |  |  |  |
| --- | --- | --- | --- |
| Requirement objectives met: | ***RO.1*** | ***RO.2*** | ***RO.3*** |
| *Contextual description of the objectives met in this section* | Every algorithm and data structure has been effectively created and tested with varying types of data. It can be assumed the cipher algorithm is working and ready for implementation with the form system. | - | - |

# Development of the forms (Stage 2)

The forms are developed in the Visual Studio IDE, with the intention of linking them fully to the program. Each form has purpose related to the function of the program. The program starts with the default form as the main menu, where the other menus can be reached through the use of the buttons.

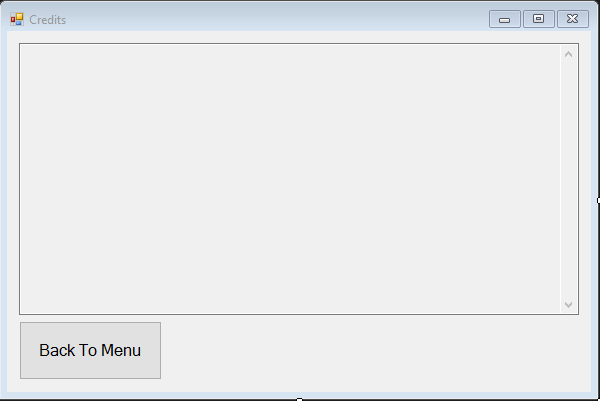
MainMenu.vb

The main menu presents the user with four choices: running the introduction module, running the simulation, viewing the spreadsheet of saved values and viewing the credits. Buttons1-4 handle with navigating the user to the mentioned forms respectively.



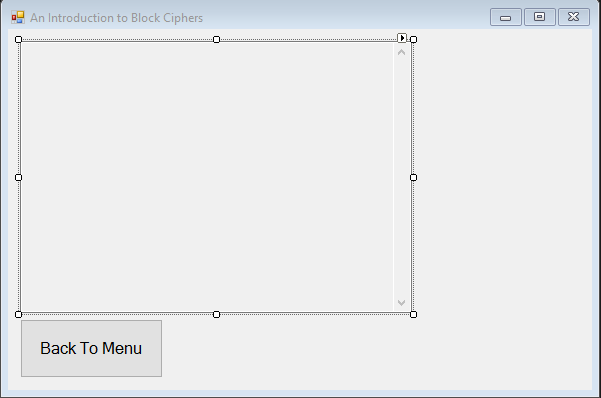
Credits.vb

This window consists of a single read-only text box that will be filled with the details of the citations and references used from the introduction paragraphs.



Intro.vb

The introduction window consists of a single read-only text box which will be filled with informative text briefing the user to the context of the program and the subject area. There is a small space on the right of the text box which can be used to display pictures relevant to the text box’s content e.g. a diagram of a state matrix, which can be visualized as a 4x4 matrix rather than a 2D array.

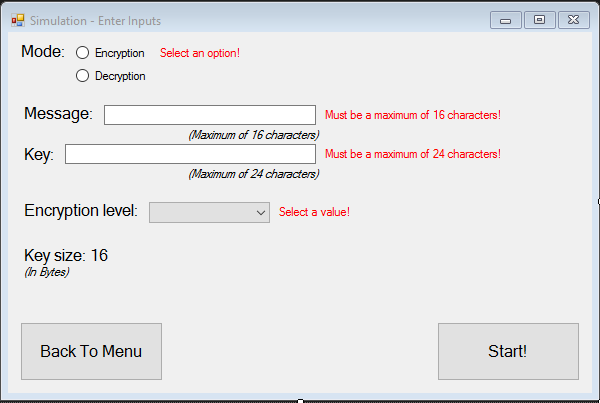


Sim\_EnterInputs.vb

This window is the data capture form for the inputs used in the block cipher, during encryption or decryption. The radio buttons RadioButton1 and RadioButton2 are mutually exclusive selections and set a variable *mode* in the updated block cipher algorithm to take either the value of 0 or 1, deciding the encryption or decryption mode respectively. The textbox TextBox1 contains the data input for the message, and TextBox2 contains the data input for the key. The drop down combo list-box ComboBox1 will be populated with three string values, “128-bit”, “192-bit” and “256-bit” for deciding the encryption level to choose for the cipher. “128-bit”, element zero will be auto-chosen to avoid a data validation error.

The red string labels Error1-4 alongside their relevant data input elements are set to “ “ and hidden during the form load. They only appear when a data validation error occurs, and remind the user they must amend their input before proceeding further.

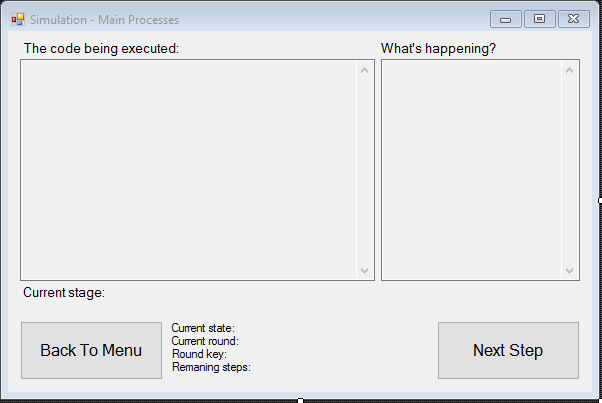
The ‘Start!’ button is only activated once there are no data validation issues, and sends the parameters of the message, the key, the encryption level and the type of encryption as parameters to the main algorithms.



Sim\_Process.vb

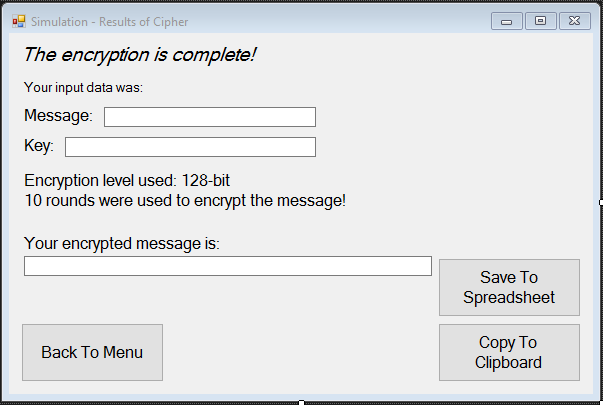
This window contains two text boxes, TextBox1 and TextBox2, which will constantly be updated with new and relevant information during each stage of the encryption. TextBox1 holds a snippet of the current processes’ code, and TextBox2 provides a contextual description of the code. The labels near the ‘Back To Menu’ button, “Current state: ”, “Current round: ”, “Round key: ” and “Remaining steps: ” contain the placeholders for the inspection of vital variables thorough the encryption/decryption process. At the end of each encryption/decryption round, the label “Current state: ” will hold the value “Current state: ” + concatenation of a string of the state at that stage. The same process will be performed for the label “Round key:”. The labels “Current round: ” and “Remaining steps: ” include a concatenation with the round number and a variable which counts down to the total procedures required to encrypt/decrypt the message, respectively.

The button ‘Next Step’ decrements the value of the countdown variable, and the next stage is processed. Each text box and label will be updated according to the latest information, until the countdown variable has reached zero. Then the form will close and Sim\_Results.vb will open, displaying the results of the cipher process.



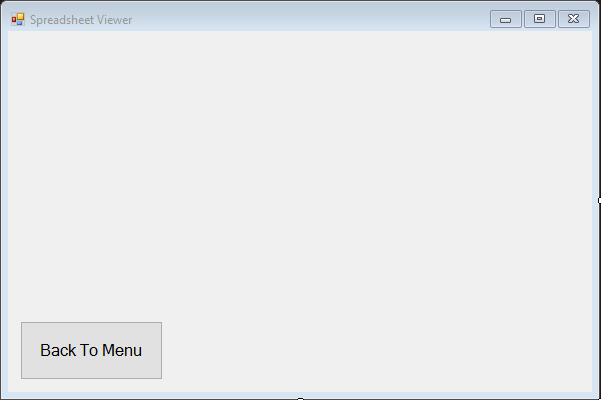
Sim\_Results.vb

This form is the final of its kind for the cipher. It displays the original input values, along with the output of the encryption/decryption. The textboxes TextBox1, TextBox2 and TextBox3 are read-only and contain the values of the original message, the key used and the encrypted message accordingly. The labels below the key’s text box are dynamic and will be updated with relevant information depending on the encryption level that was used. With the use of Button2 and Button3, the user has the option to save the encrypted message to a spreadsheet, or copy it to clipboard respectively.



SpreadsheetViewer.vb

This form will be reserved for the display of entries from the spreadsheet. A list containing all the encrypted key entries from the spreadsheet and a further three buttons are planned to be added. The three new buttons will let the user copy an entry to clipboard, delete an entry from the list and remove all entries from the list, refreshing it.

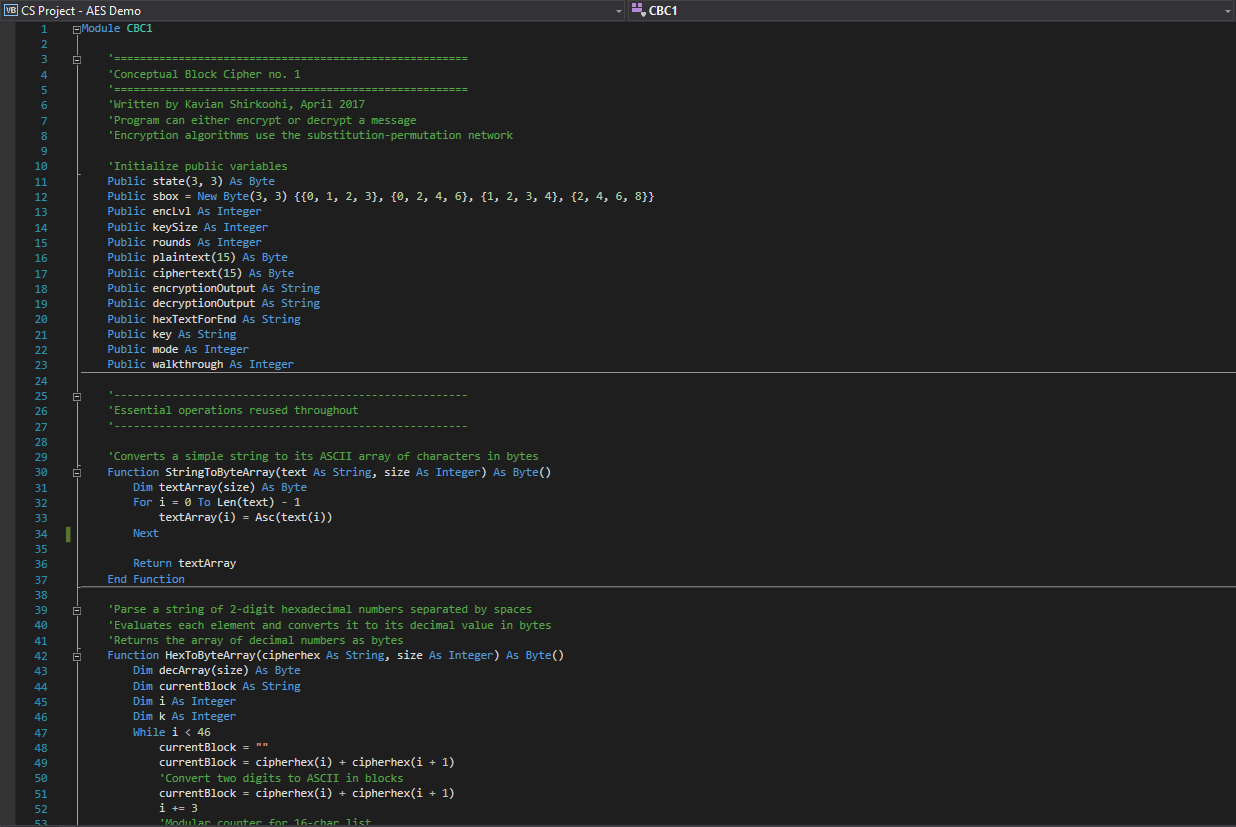


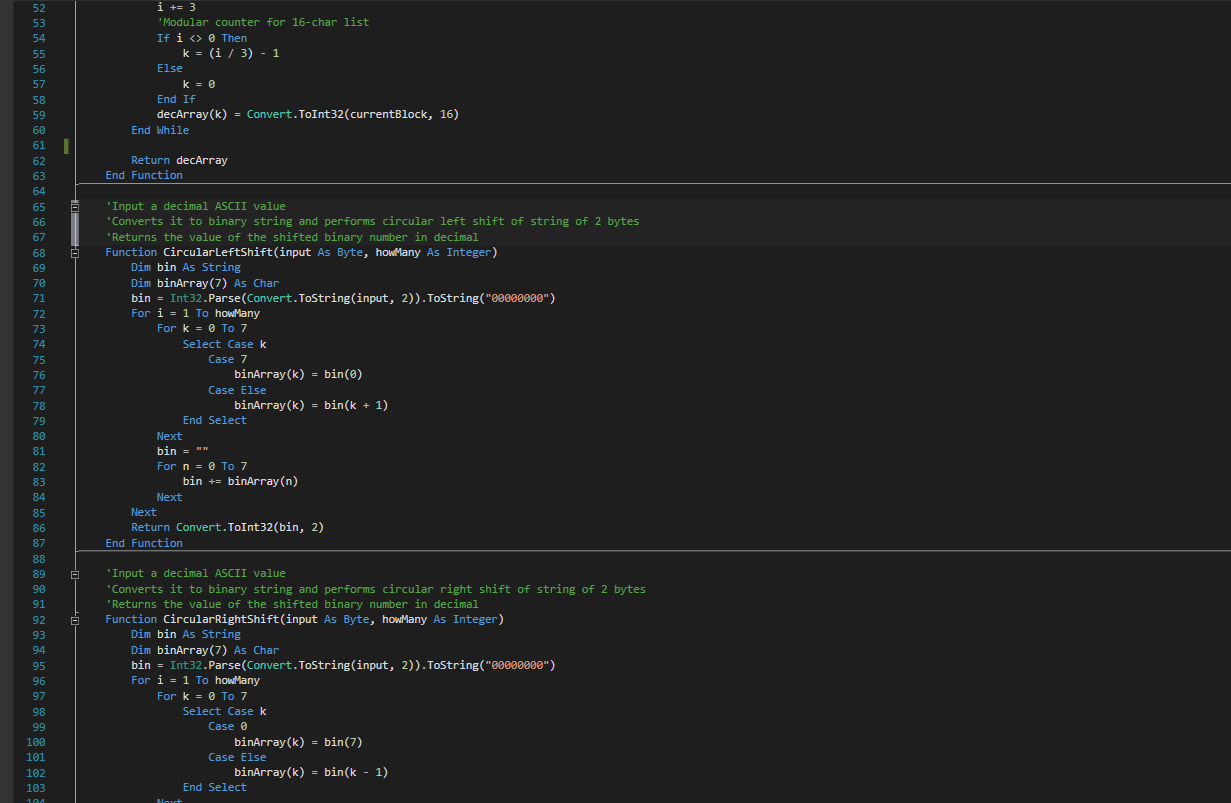
|  |  |  |  |
| --- | --- | --- | --- |
| Requirement objectives met: | ***RO.1*** | ***RO.2*** | ***RO.3*** |
| *Contextual description of the objectives met in this section* | The complete form system of the program has been described and defined. The cipher simply needs to be implemented with the form system for the development to be complete. | The forms are simple and only contain as much information as necessary. This clarifies the users decisions with how to use the program, and makes the end user experience more simple. | - |

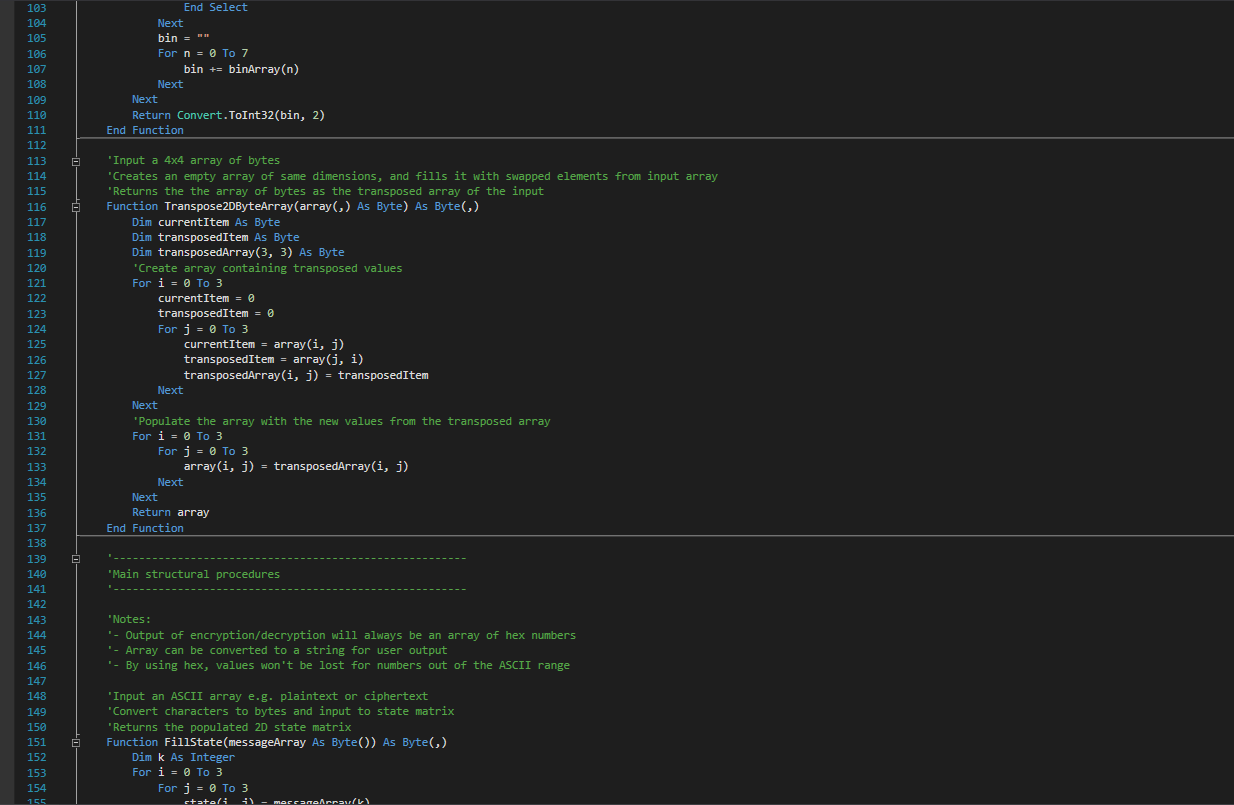
# 

# Implementation of the cipher with the forms (Stage 3)

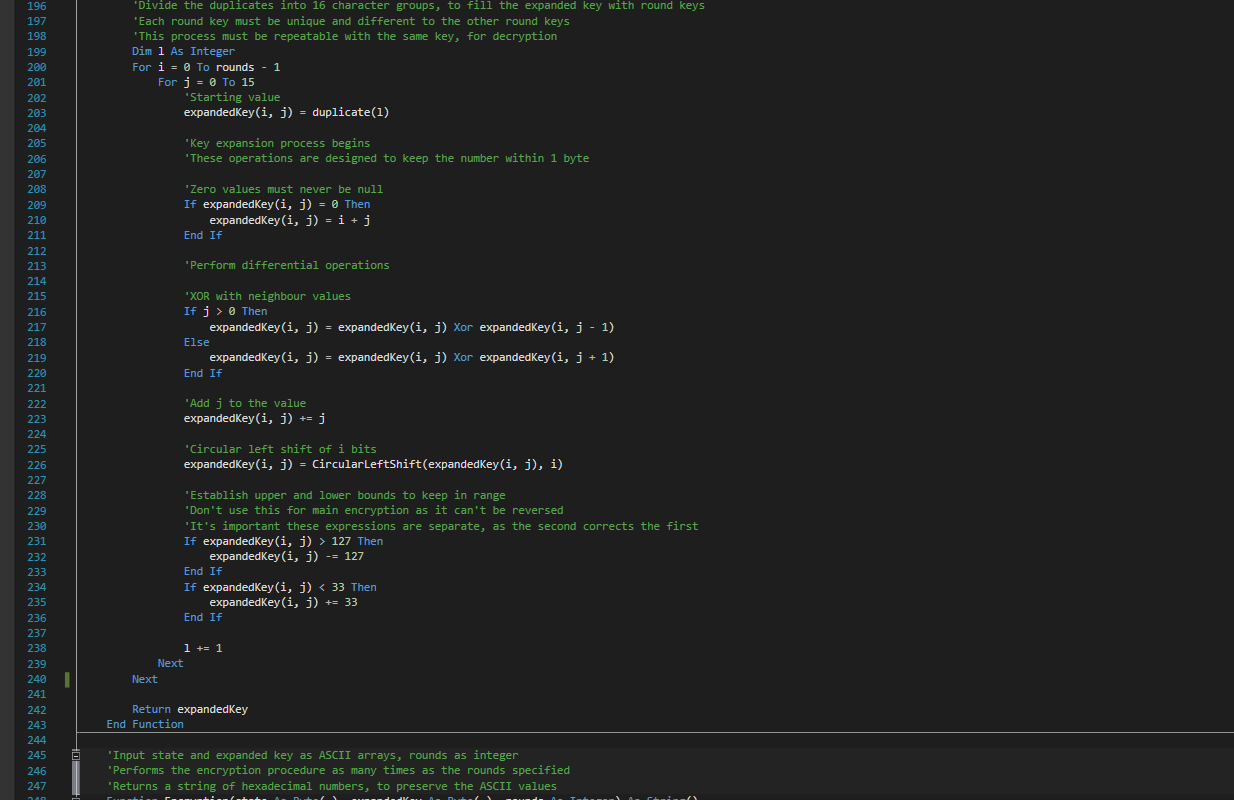
The adjustment of the cipher algorithm was relatively straightforward. The only work that was required, was to convert the console reading and writing statements to the relevant data capture and form element updating features. The adjusted code for the cipher algorithm is shown below:

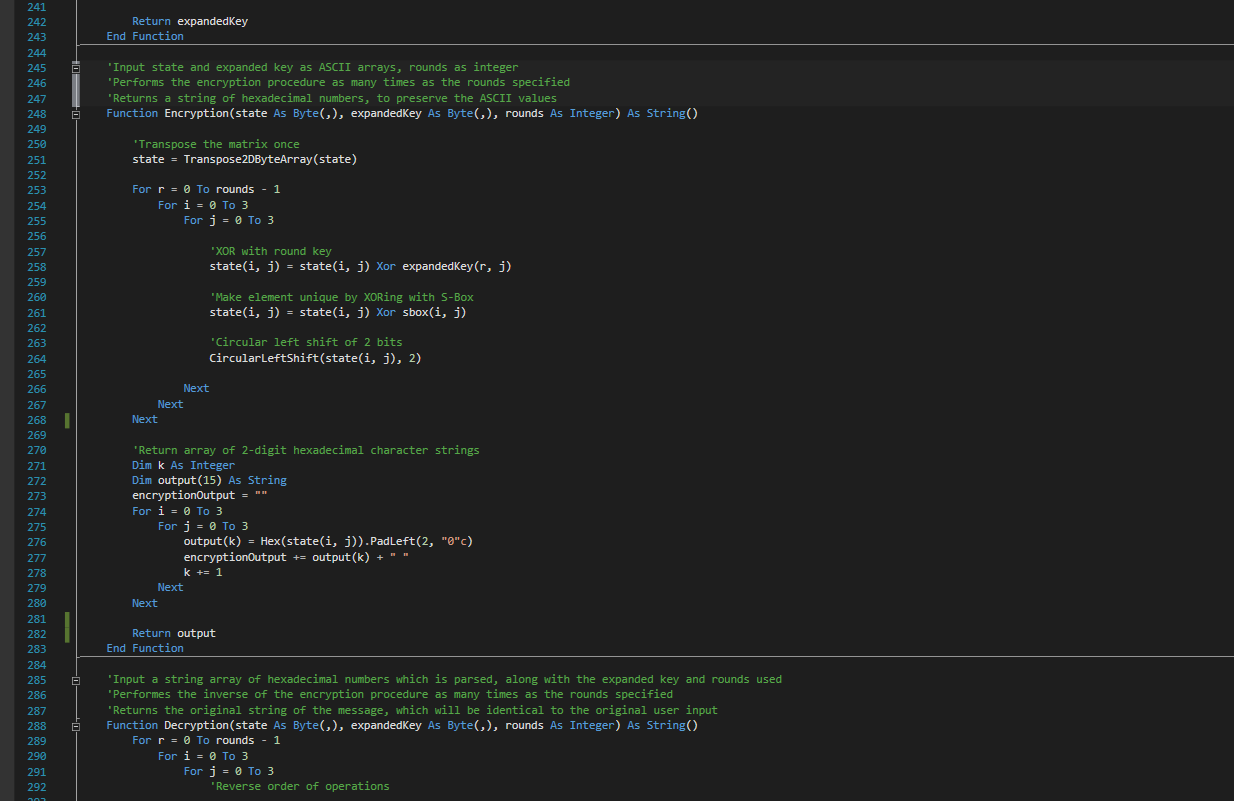


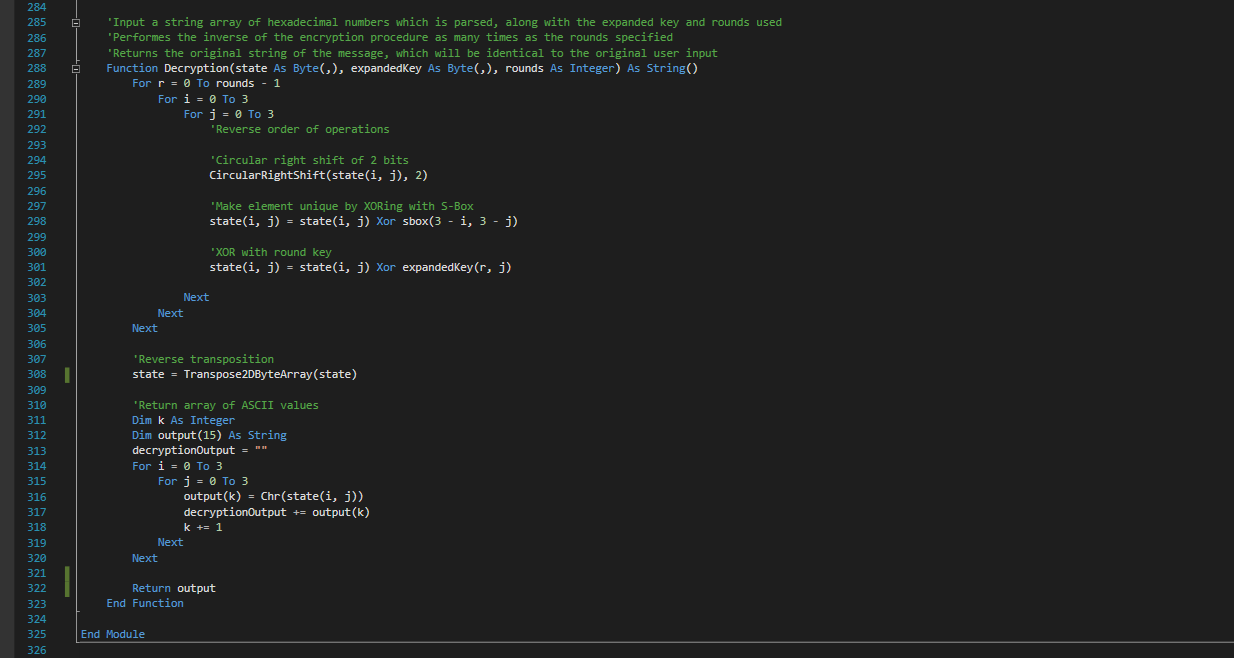






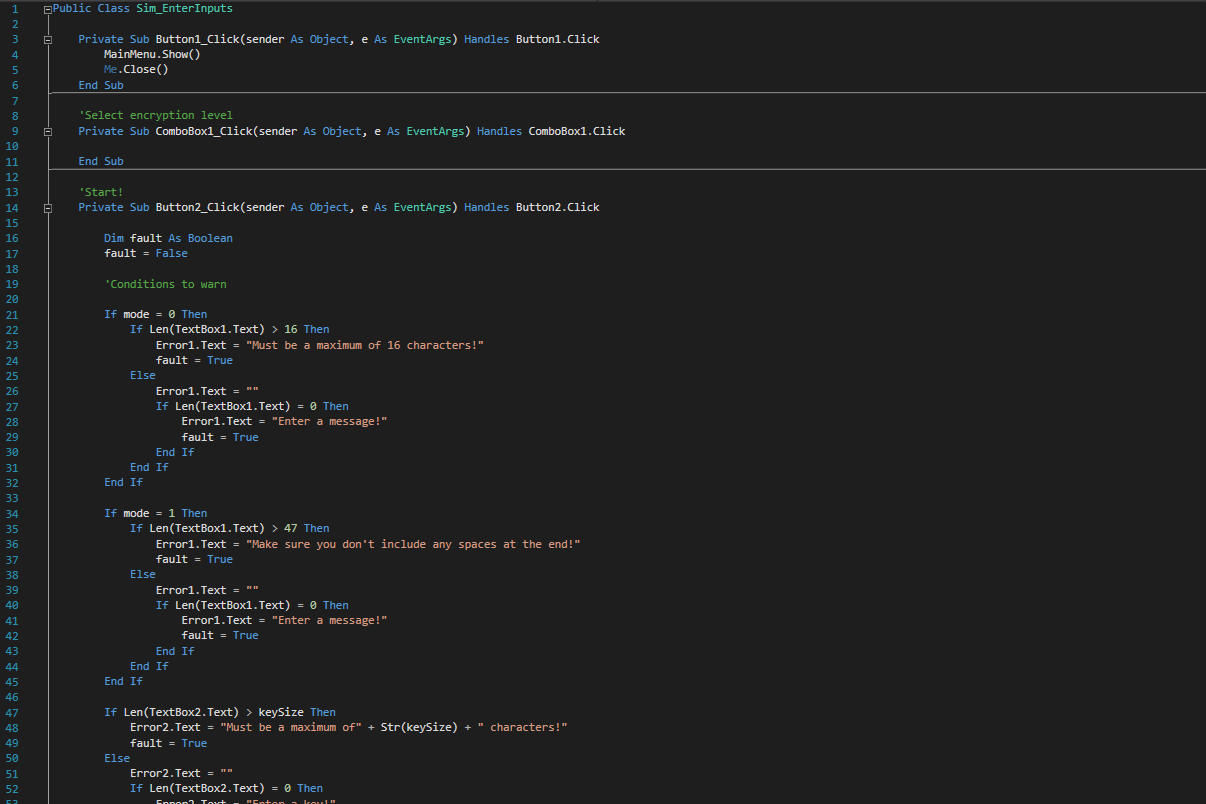


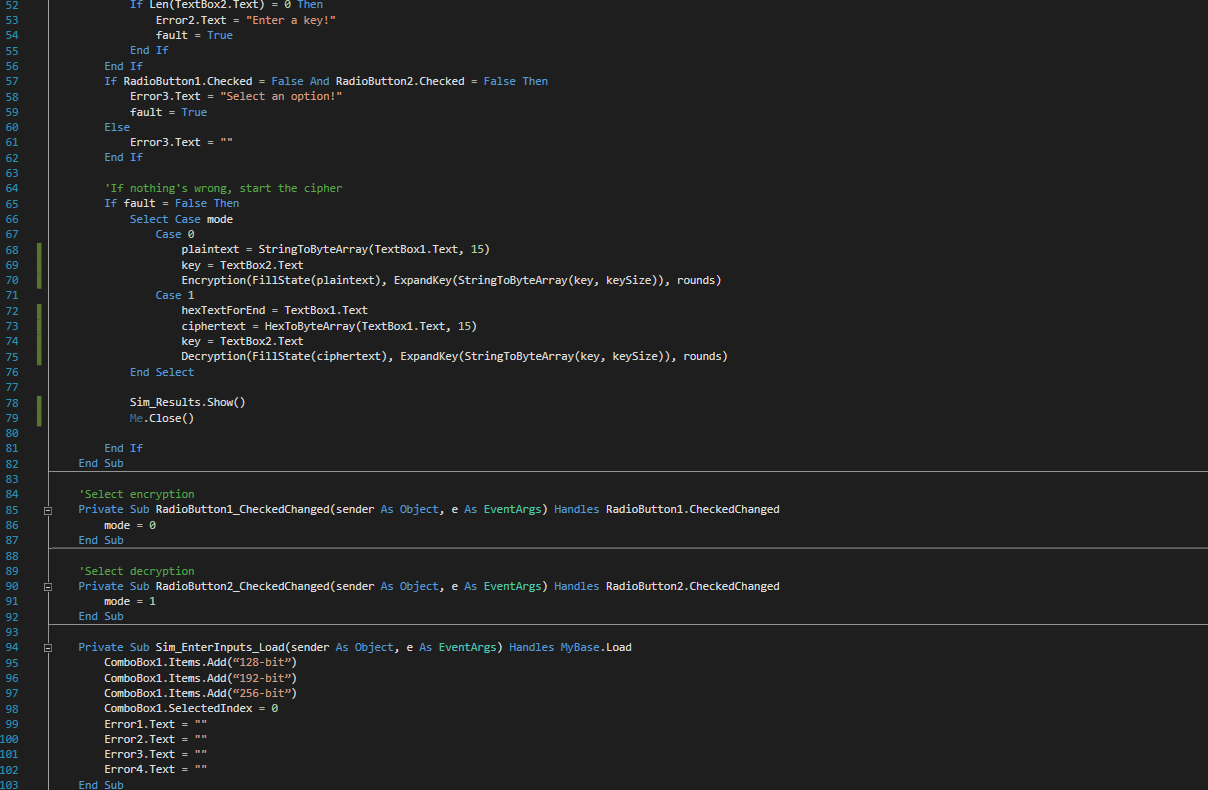


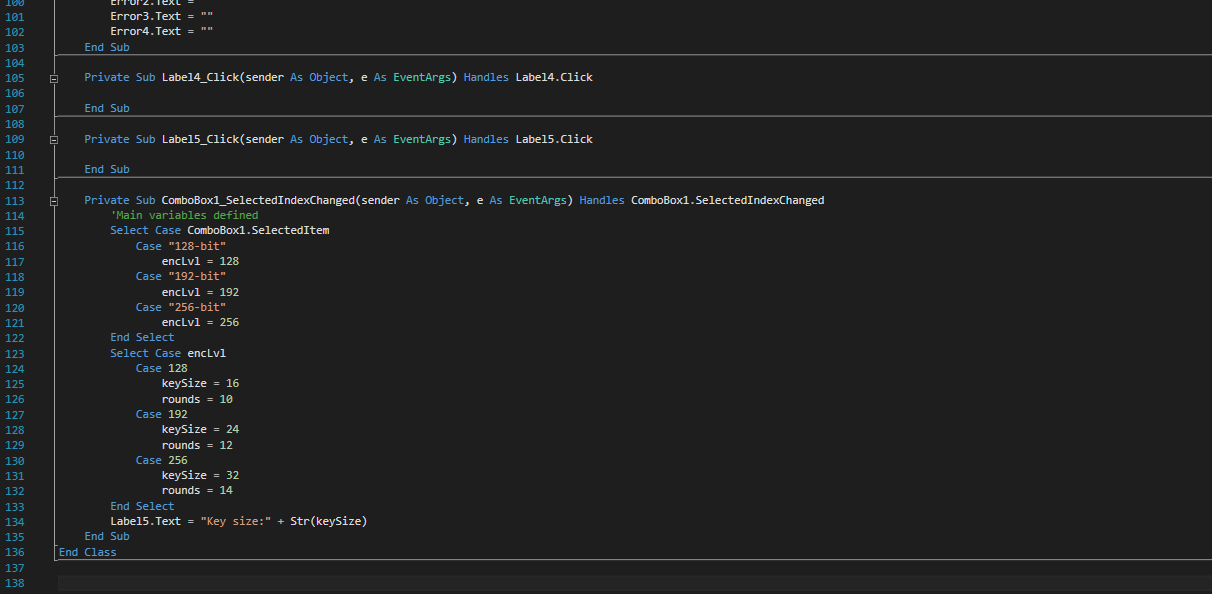


Sim\_EnterInputs.vb

The input and data validation processes were implemented into the execution of the ‘Start!’ button, Button2, and the code for this form was adjusted as necessary to ensure the user input was of the correct data type and range before proceeding to encrypt or decrypt.

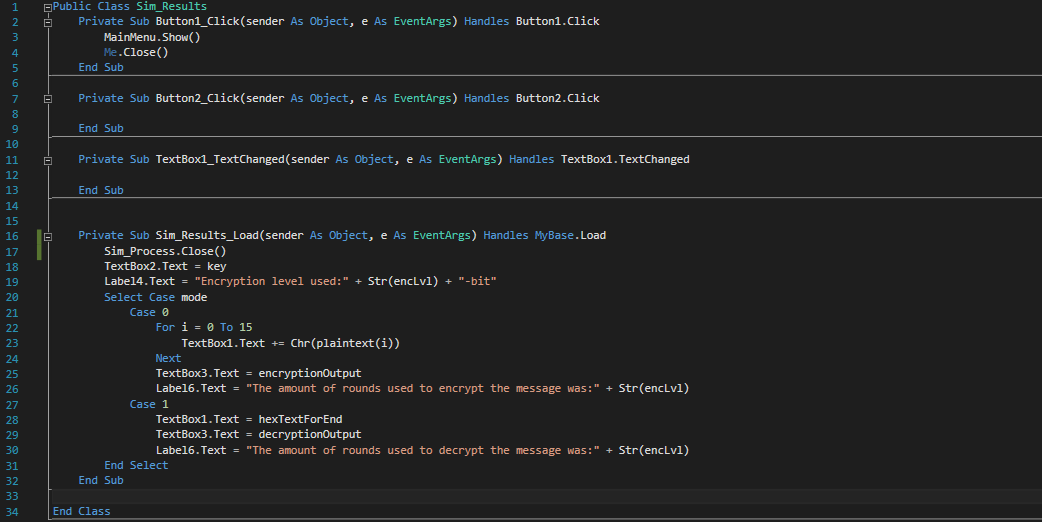






Sim\_Results.vb

The output from the cipher algorithms was adjusted and designed to be passed to this form, where the relevant information for the original values and the encrypted/decrypted result will be displayed.



*Due to a change in requirements and time constraints, the client has requested that the implementation of the walkthrough feature within the code and the spreadsheet integration need no longer be considered. The introduction was self-explanatory and contained enough information for the user to comprehend the results of the encryption and decryption process.*

# Data validation and testing

The encryption and decryption ciphers will now be subject to data validation and testing, to ensure the program is stable for evaluation from the client.

Recall that the five types of data that will be tested and validated will be of the following:

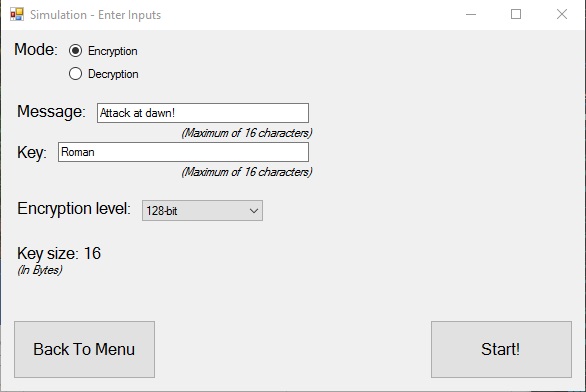
* Valid data
* Valid extreme data.
* Invalid data
* Invalid extreme data.
* Erroneous data of the wrong data type

Valid data

Message: Attack at dawn!

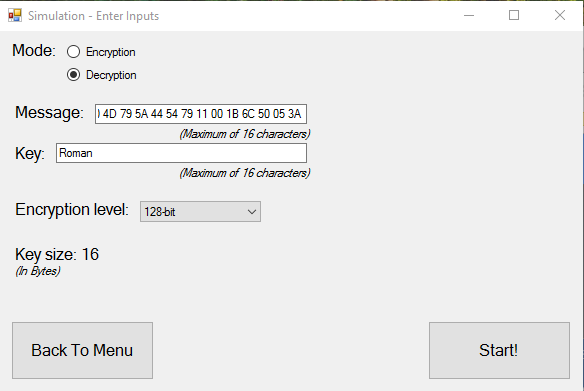
Key: Roman

***Encryption***

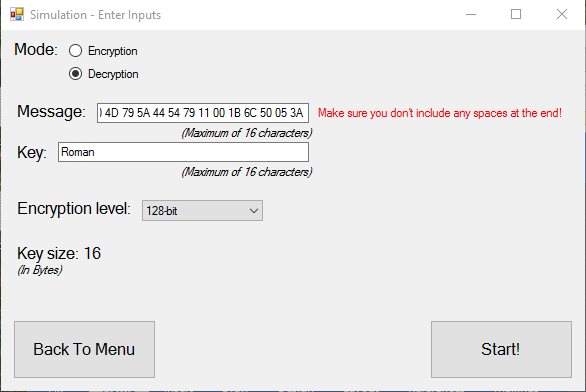


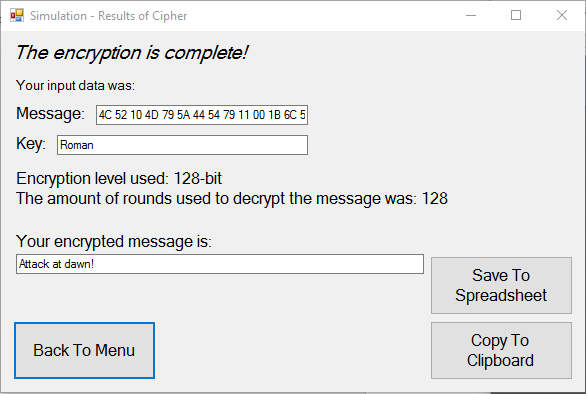


***Decryption***



Note: it is worth mentioning that the error label was triggered for the initial input of the message, as it contained an extra space character which was detected and rejected as input. The ciphertext had to be input again, without the extra space at the end.



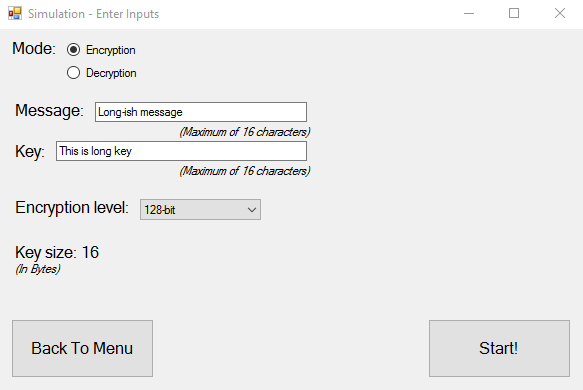


Valid extreme data

Message: Long-ish message (16 characters)

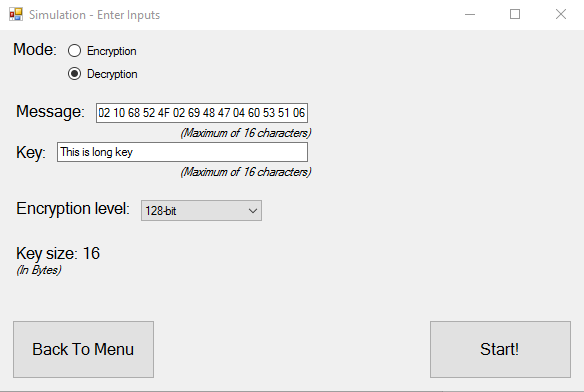
Key: This is long key (16 characters, using 128-bit configuration)

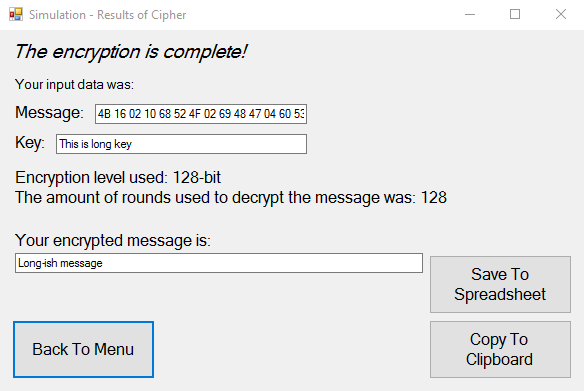
***Encryption***





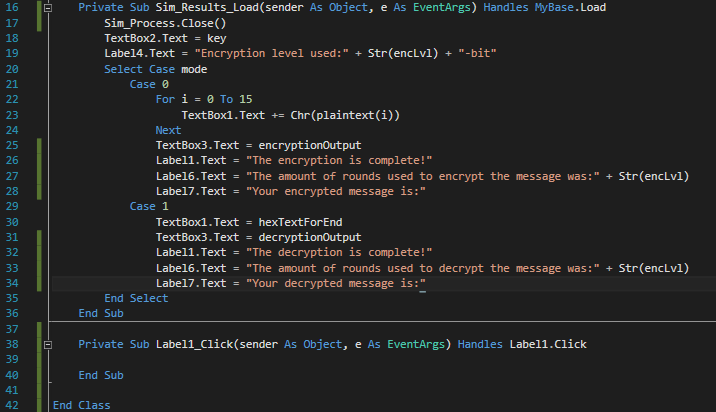
***Decryption***





One slight issue that needs fixing is the persistent label “The encryption is complete!” – this should read “The decryption is complete” during decryption process as it can be confusing to read. It will be amended in the code before the next data value is tested.

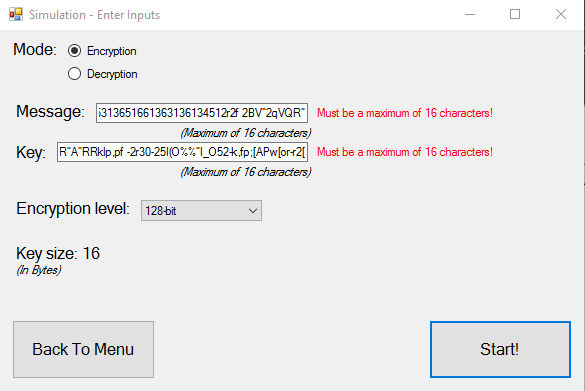
The code snippet displaying the updated lines within the code of the form Sim\_Process is shown below. The issue of the false text for decryption will not appear anymore.



Invalid data

Message: 590-15-oR”\_rlr2-2rl-2lr-lq-2=r2=q2r=-lr2q-=53115313651661363136134512r2f 2BV”2qVQR”

Key: “RER”A”RRklp,pf -2r30-25I(O%%”I\_O52-k,fp;[APw[or-r2[

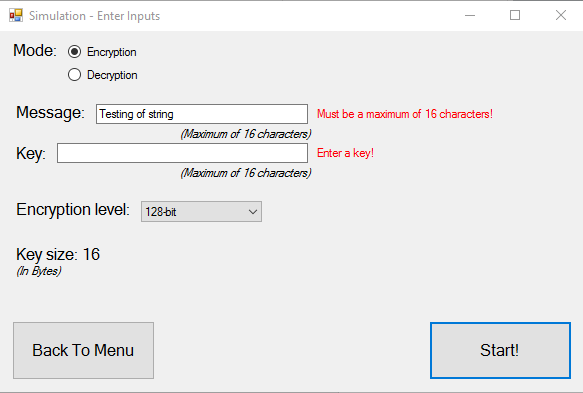


The form immediately displayed an error and the message was not able to be encrypted. This data did not pass the validation check and was rejected by the data capture form before any further action was taken.

Invalid extreme data

Message: Testing of string (17 characters)

Key: (0 characters)



As with before, the form immediately displayed an error and the message was not able to be encrypted. This data did not pass the validation check as the message was over the maximum length, and the key had no characters.

Erroneous data

For this stage, two parts will be conducted. Part 1 will be a test of encrypting a message with a key, then using the wrong key to decrypt it. Part 2 will be a matter of using the wrong encryption level to decrypt a message.

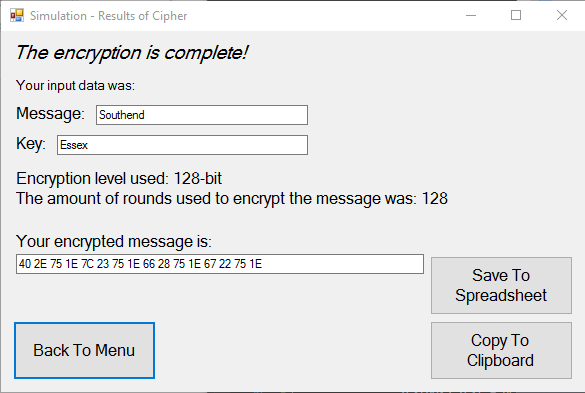
*Part 1*

Message: Southend

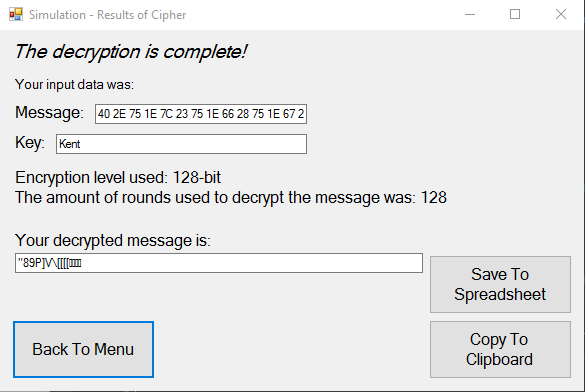
Key: Essex

Wrong key: Kent

***Encryption***



***Decryption***



This demonstrates the security that the key provides to encrypting the message. If the key isn’t known, it isn’t easy to decrypt the message. The required round keys won’t be generated, and the state elements will be XORed with something else.

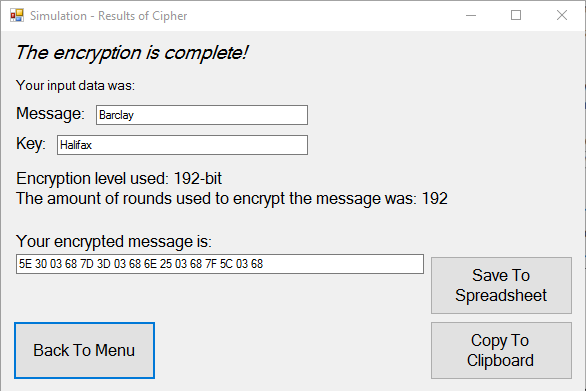
*Part 2*

Message: Barclay

Key: Halifax (192-bit)

A 128-bit encryption level will be used for decryption

***Encryption***

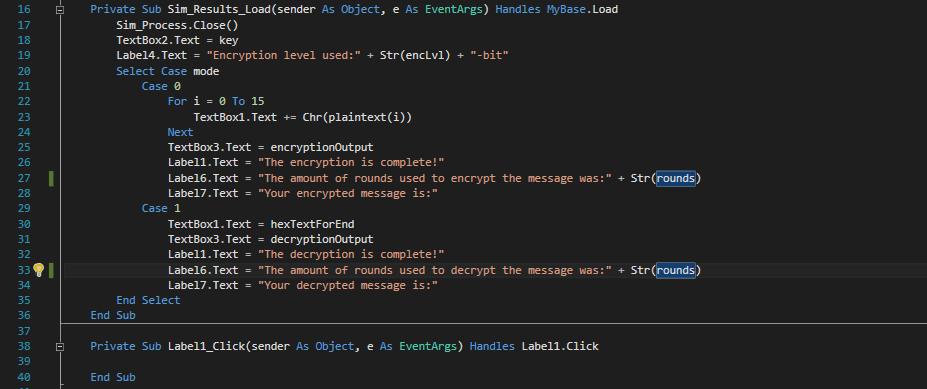


***Decryption***



As expected, if the key size upon encryption does not match the key size upon decryption, the required round keys will not be generated, and the state will be XORed with different values. This will result in meaningless information.

The label for the number of rounds used to decrypt contains false information, it is displaying the encryption level instead of the amount of rounds used. So, it will be amended for the final testing stage:



*With the following data types tested, the program can now be forwarded to the client for the evaluation and beta testing from their approach.*

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement objectives met: | ***RO.1*** | ***RO.2*** | ***RO.3*** |
| *Contextual description of the objectives met in this section* | The program is now complete and ready for evaluation. | - | - |

Evaluation

The program will now be evaluated in its usability and its ability to produce consistent quality data during encryption and decryption of a variety of values. Firstly, the program will be tested for five different messages to encrypt of typical nature that the user would enter the data capture forms. The messages will vary in length and alphanumeric characters, the same for the key being used to encrypt it with.

An evaluation from the client will also be performed, to gain a second opinion from a questionnaire and presenting the client with the program to encrypt and decrypt his messages with. The client will sign the questionnaire to verify that they have carried out the required data evaluations for the cipher.

# Evaluation testing

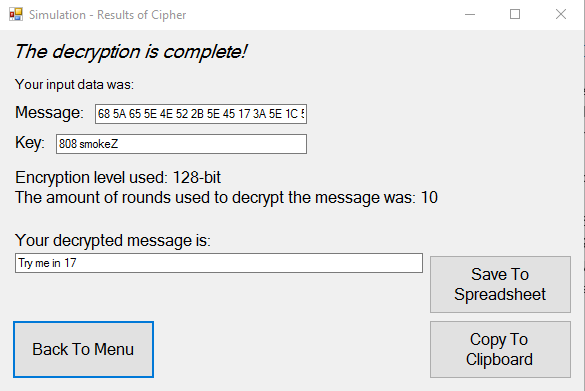
Final development-side testing

To ensure the encryption and decryption methods are consistent, five alphanumeric messages will be encrypted then decrypted, of varying length and encryption levels, with varying and unique keys. The results of encryption and decryption will be displayed in the form.

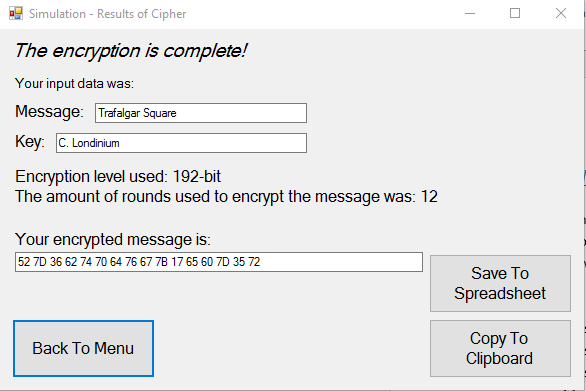
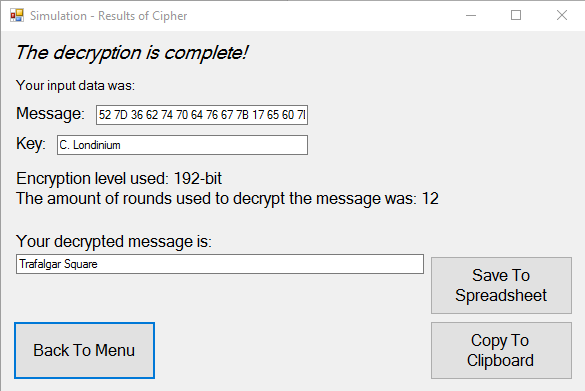
The data for the messages are follows:

* Message 1 - “Try me in 17”, key “808 smokeZ”, encryption level 128-bits
* Message 2 - “Trafalgar Square”, key “C. Londinium”, encryption level 192-bits
* Message 3 – “Lots of money”, key “2000” encryption level 192-bits
* Message 4 - “P versus NP”, key “unsolved”, encryption level 256-bits
* Message 5 – “Descartes”, key “graph”, encryption level 256-bits

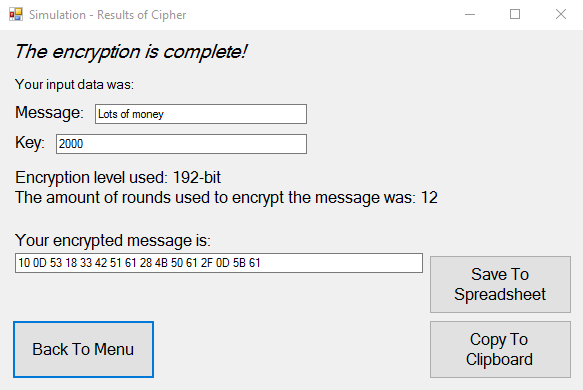
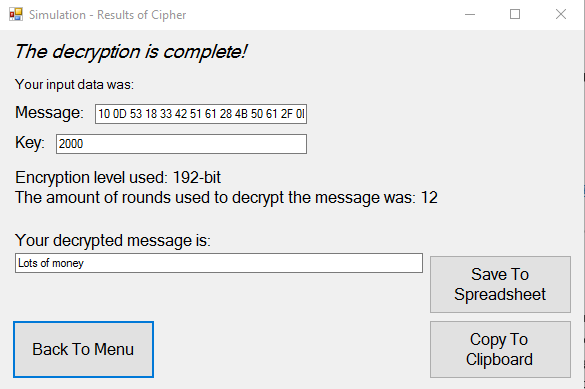
Message 1

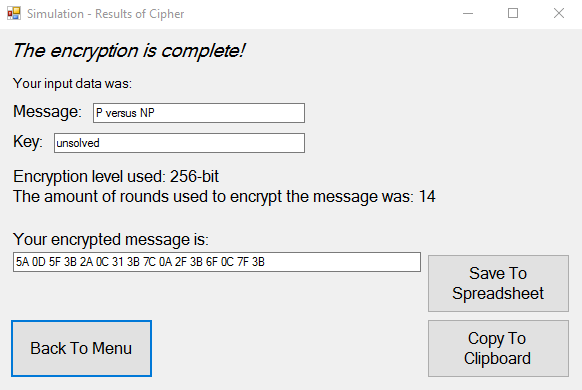
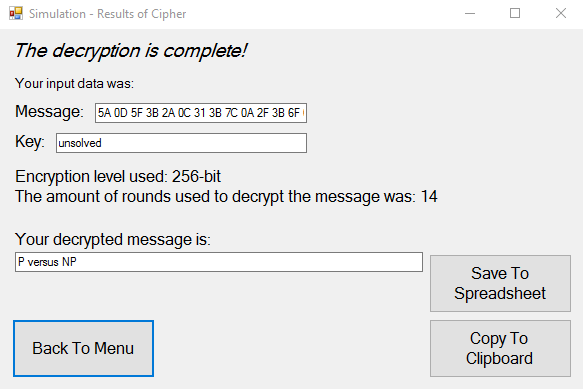
Message 2

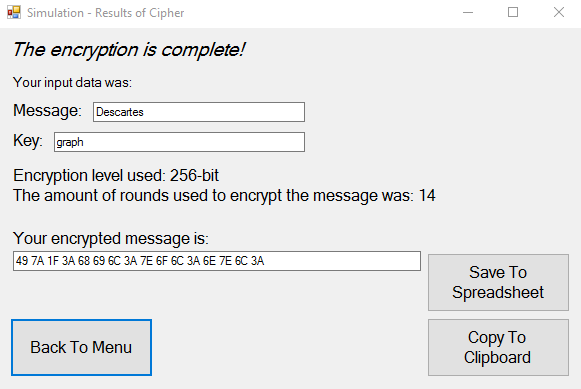
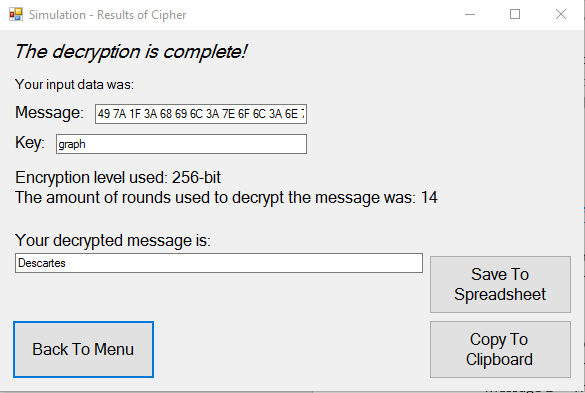
Message 3

Message 4

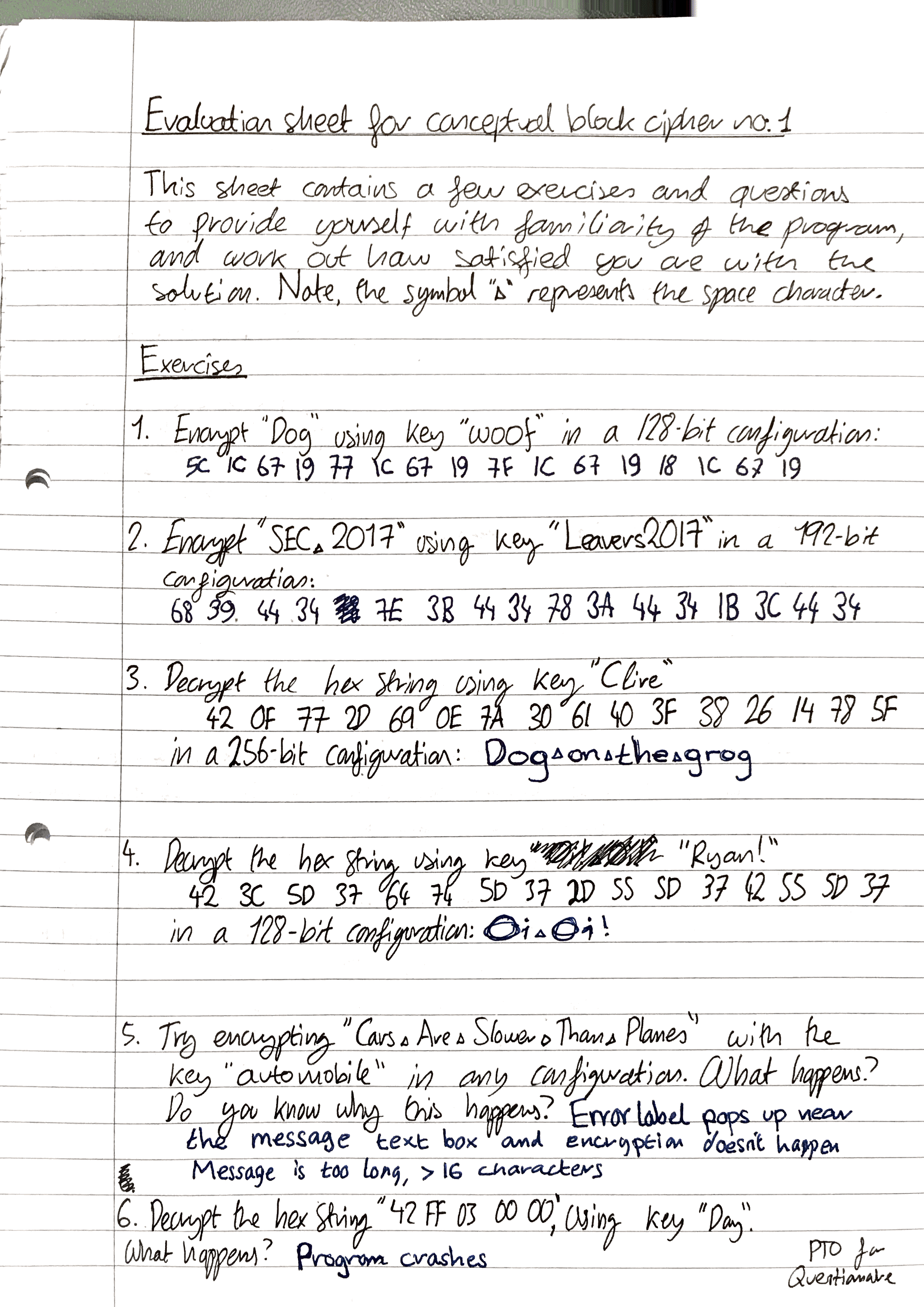
 

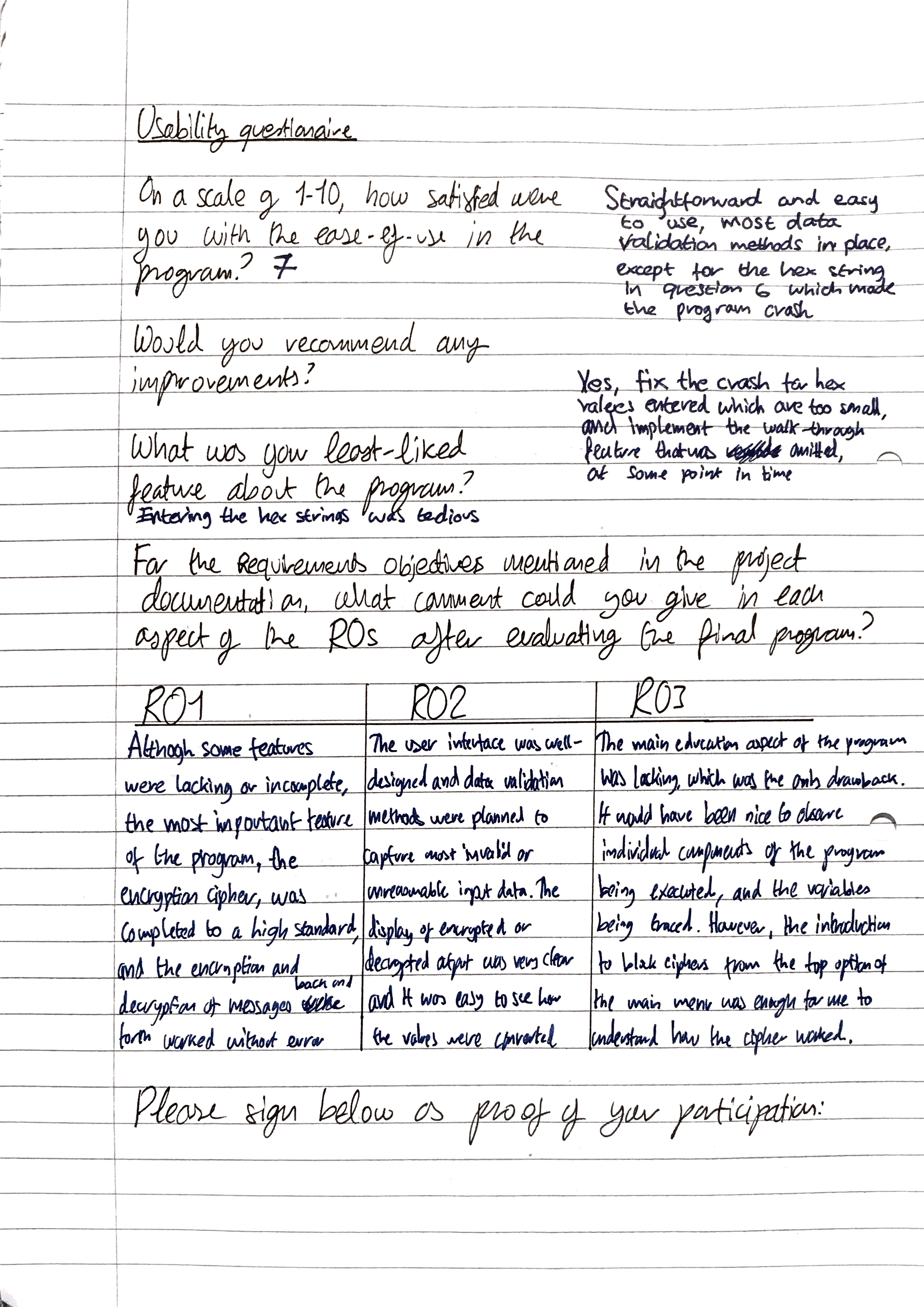
Message 5

Client testing and evaluation

A sheet was prepared containing several exercises and questions for the client to fill in, which were like the exercises carried out during the previous testing stage. Upon usage of the program for the first time, the client filled in the form with his responses. The sheet composed of two pages, the first containing the exercises and the second containing the questionnaire and Requirement Objectives assessment.





The user successfully encrypted and decrypted valid data within the exercises, however he encountered a bug within the last exercise – there was no data validation measure for hexadecimal strings of much shorter length than expected. The code could be amended to fix this, but at this stage it would not affect the functionality of the program as the hexadecimal string will almost always be pasted as a full 47-character string, from the output of the encryption or future spreadsheet values.

The user was satisfied with the program overall, however suggested the walk-through feature be implemented at a later stage.

# Outcome

Towards the end of the development, the cipher walk-through element and the spreadsheet integration was chosen to not be implemented. It took too long to implement and develop within the time-constraints given.

The client was satisfied with the encryption and decryption ciphers as they were, along with the basic introduction to block ciphers page. However, should this program be improved in the future, there are the following features that would need to be added for it to be as effective as it was originally intended:

* A working spreadsheet integration system. This would contain the features to access, retrieve a string from, and update a Microsoft Excel spreadsheet. The entry number would be stored in the first column, the hexadecimal ciphertext strings would be stored in the second column of the spreadsheet, and the encryption level they were encrypted at in the third column. The user would be expected to remember the key for the ciphertext string. If they wanted to encrypt many values without forgetting the key, they could use the same key with multiple messages, and could adjust the key size to mix up the output. Functions would be written within the block cipher program to access the spreadsheet and pass values to and from it. The use of external libraries could be utilised.
* A walk-through of each part of the encryption/decryption process. String arrays could be created storing the code pasted from the source code of the encryption, as well as the description of the code. The elements from these string arrays could be output to the text boxes within the Sim\_Process form, and the values of the state, the round key, the current round, and the current process could be displayed through the labels. The code execution to the next step would be halted until the user clicks the ‘Next Step’ button. An iteration sentry value could be used to determine whether the next step is the output for the encryption/decryption or simply the next procedure to be executed in code.

The client’s feedback was mostly positive, although he was aware that features had been omitted for the final version of the program, he found the encryption and decryption of the messages efficient and insightful. The feedback from the client has been implemented within the final Requirement Objectives of the program.

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement objectives met: | ***RO.1*** | ***RO.2*** | ***RO.3*** |
| *Contextual description of the objectives met in this section* | “Although some features were lacking or incomplete, the most important feature of the program, the encryption cipher, was completed to a high standard, and the encryption and decryption of messages back and forth worked without error.”  ***SUCCESS*** | “The user interface was well-designed and data validation methods were planned to capture invalid or unreasonable input data. The display of encrypted or decrypted output was very clear and it was easy to see how the values were converted”  ***SUCCESS*** | “The main education aspect of the program was lacking, which was the only drawback. It would have been nice to observe the individual components of the program being executed, and the variables being traced. However, the introduction to block ciphers from the top option in the main menu was enough for me to understand how the cipher worked”  ***FAIL*** |

The overall score for the Requirements Objectives was 2/3. Whilst RO. 3 didn’t fare as well as the others, the program is marked as a partial success, as the encryption and decryption algorithms are fully functional and the user interface works with effective data validation features.

# The final program

Although a few features were lacking in the final program, the application still serves as a good block-cipher simulation. The positive aspects of the finished program are the ability to encrypt and decrypt messages, and the modular nature of the code in which the program was written in. Most features are written as functions or subroutines, containing detailed comments describing their purpose and operation, as well as comments within the individual operations of some more-in-depth procedures, such as the hexadecimal string conversion. Two more features that could be added for data validation is a check for non-alphanumeric characters and a flag set if an input string contains one, and an error message generated for hexadecimal strings that are less than 47-characters.

The program can be amended later by the same developer or a new developer (since the modular nature of the code and the comments make it easier for a different programmer to work with the project) to include the features that were missed out, so the program still has a lot of potential.

# Maintainability

Due to the modular nature of the solution, the ability to improve upon the algorithms and components would be relatively easy. Comments are present within the code, and where possible, the code is self-documenting through the effective use of variable and function identifiers. If the cipher algorithm ever needed to be rebuilt or changed, the developer would simply have to copy the current algorithm code and create a new console application for it. The console application would be the most effective form of output and testing during development, as variable values and control structure markers can be easily written to display their output. Once the developer is satisfied with their edits, they may reconvert the algorithm to utilise the Windows forms system.

# Taking the program further

The program could improve its function in two ways. The cipher could be modified to encrypt a value more than once, i.e. multiple applications of the cipher towards continuous inputs of the state matrix and key expansions, so there is an extra layer of security within the encryption of the message. This is like how the real-life block ciphers are used; most operations are performed multiple times upon the same data as the time taken to encrypt the data using the advantages of a block cipher’s encryption/decryption speed is relatively low, compared to other encryption forms.

Another way the cipher could be improved is by extending the initial input limit of 16-characters to as many characters as the user desires. The 16+ character string could be iterated through and, like the key expansion procedure, be separated into 16-character blocks to be input separately to the encryption procedure. The output of the hexadecimal digits for the first block and the second block could be joined together, and so on. The hexadecimal parsing function could be extended to take a string of hexadecimal digits any length, and separate them into 47-character blocks just like the encryption procedure. The same process would be carried out for decryption. Permutations between character blocks could also add another layer of security, although thought would have to be put aside for considering how to remember the permutations for decryption. Perhaps a second key could be used to log the amount of permutations between blocks? There is no limit to how complex the encryption algorithm could become, but for a fact the more complex it would be, the more steps it would need to take and the more processing power from the CPU would be required. Furthermore, there would need to be many variables dedicated to recording the amount of permutations and variations between the encryption steps, otherwise there would be no way to decrypt the message.

Citations

Where relevant, links to resources and information used within this document will be detailed below.

[1] <http://www.telegraph.co.uk/technology/2016/11/24/elite-cyber-defence-college-open-second-world-war-code-breaking/>

[2] <https://en.wikipedia.org/wiki/AES_instruction_set>

[3] <http://searchsecurity.techtarget.com/definition/block-cipher>

[4] <https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#Security>

[5] <https://www.cs.bham.ac.uk/~mdr/teaching/modules/security/lectures/symmetric-key.html>

[6] <https://en.wikipedia.org/wiki/Key_schedule>

[7] <https://en.m.wikipedia.org/wiki/Kerckhoffs%27s_principle>

[8] <https://en.wikipedia.org/wiki/Rijndael_key_schedule>

[9] <https://en.wikipedia.org/wiki/Rijndael_S-box>

[10] <http://www2.cs.man.ac.uk/~raym8/comp38411/main/node33.html>

[11] <http://www2.cs.man.ac.uk/~raym8/comp38411/main/node34.html>