A-level Computer Science practical project

(7517/C)

Teacher standardisation

Summer 2017

Project E

Note: this is a real project submitted by a candidate for A-level Computer Science. Every effort has been made to respect the anonymity of the source of this project and those who have contributed to it, without compromising the work or invalidating it. However, some personal details may remain, if this is the case we ask the reader to respect their privacy.

Copyright © AQA and its licensors. All rights reserved.
AQA Education (AQA) is a registered charity (number 1073334) and a company limited by guarantee registered in England and Wales (number 3644723). Our registered address is AQA, Devas Street, Manchester M15 6EX.

Analysis

1. Background To/ Identification Of Problem
    1. Background:
       BGS is a school located in the north west that teaches pupils from the age of 4 to 18. The school teaches a variety of subjects at several levels, including GCSE and A Level.
    2. Identification of Problem:
       The Computing department are beginning to use a new A Level syllabus and the head of the department, Mr. M, is looking for new materials to use and interactive ways of teaching the new topics. One of these new topics is Encryption (Appendix 1.1). This new topic requires the students to learn about two different ciphers- the Caesar Cipher and the Vernam Cipher. Public/Private Key Encryption (Asymmetric Encryption) and Bitmapped Graphics are also on the specification (Appendix 1.2 and Appendix 1.3) and while they are not new topics, it would be beneficial for the students to have new ways of learning about them. To demonstrate public/private key encryption, a specific example of this, such as RSA, could be used. A way of combining the bitmapped graphics and encryption would be to have a demonstration of steganography, an alternative to encryption, where the message is hidden in some way, and this can be done by changing the values of some of the bits in a bitmapped image. Although the topics can be taught using the textbook, it would be more fun and effective if the students could actually have a go at steganography and encryption. In the case of steganography, it would be useful for the students to be able to physically load up an image and hide their own messages in it, and they would also discover things that may not be as obvious from only reading about bitmapped images, for example, the insignificance of certain bit on its own; they would come to this realisation when they see the image does not change much when they hide their message in certain bits. I have researched the structure of bitmapped files to identify how steganography might be achieved (Appendix 1.4). With the ciphers, a program could be used by the teacher to make practise questions easily. It could also be used by the students in lesson time to work independently with encoding and decoding messages, to help them in understanding how each of the ciphers work and how effective and secure they are.

2. Description Of The Current System
   Since it is new to the syllabus, there isn't a current system for teaching about encryption. However, without an interactive tool, the department would teach the topic using the textbook available, which contains all of the necessary information and several exercises that the students can do to practise encoding and decoding. Previously, with other topics, notes have also been produced, which are accessible on the department's website (Appendix 1.5). These are distributed in the lesson and discussed, before the students are given practise or past paper questions to answer about the topic. While the images topic has not changed and those lessons could still be taught in this way using the old resources, it would not be ideal for encryption, because there are no past papers on it; there are only 2 example questions available from the Additional Specimen Papers officially released by the exam board (Appendix 1.6).

3. Research Into Steganography and Cryptography
   Steganography and cryptography are methods of preventing people from reading messages. Steganography hides the message itself, so that it is not visible to everyone. It used to be very common in ancient civilisations such as Greece, Babylon and Egypt. In particular, it was easier than using cryptography for the Egyptians, whose alphabet consisted of hieroglyph symbols representing phonetics rather than single letters, which made it difficult to change. Similarly,

steganography is also favoured in places like China, where one character can mean a whole word.

An example of steganography is microdots, where the data to be hidden is shrunk down to a barely visible size. Originally, the microdot was a small disc which could be hidden in letters or on images. A microscope was needed to read the microdot. Microdots can also be used digitally; they can be placed in other images or documents where they can't be detected. On a computer, it is easier to read a located microdot, as it can easily be zoomed in on.

A solely digital method of steganography is to use a bitmap file- certain bits in the file can be flipped to represent a message. Since some individual bits on their own have little significance, it can be difficult to spot where the colour of a pixel has changed; a pixel may only change by one shade of the same colour, which may be undetectable unless the image is zoomed in to pixel level.

Cryptography hides the meaning of the message, by making it unreadable to people who don't know how to break the encryption; while the message may obviously be there and visible to everyone, its meaning is still protected. Cryptography is a lot easier with the more conventional alphabets, such as the Roman alphabet, where letters are grouped together to represent sounds and words.

The Caesar cipher and the Vernam cipher are both types of substitution cryptography, where letters are swapped for other letters.

The Caesar cipher uses two alphabets, the second alphabet being shifted along a chosen number of letters. This shifted alphabet is lined up with the original, and each letter on the original is substituted by the one lined up on the shifted. For example, encrypting “hello” with the Caesar wheel setting shown (shift 19) produces the ciphertext “AXEEH”.

The cipher is quite a weak cipher, as the number of possible keys is small. Also, since each letter is encrypted with the same shift, relative frequency analysis can be done on the ciphertext to find out what some of the characters have been encrypted as. Simply knowing what one letter is encrypted as can reveal the whole message's encryption shift.

The Vernam cipher uses binary representation of characters, XOR-ing one character from the plaintext with one character from the key to produce the encrypted value. Each character in the plaintext is encrypted using the character at the corresponding position in the key. If the key is shorter than the plaintext then it is repeated, for example, for a plaintext of length 9 with a 4 character key, characters 1, 5 and 9 in the plaintext would all be encrypted using character 1 in the key.

The Vernam cipher can be done using the ASCII values of characters, however it was originally done using a 5-bit system, the Baudot code (Appendix 1.6). The original Baudot code did not have control signals, which meant that the result was almost always a character that could be

printed out; for example, encrypting “A” (Baudot code 10000) using a key of “F” (Baudot code 01101) produces the Baudot code 11101, which can be converted to the letter “D” for printing. However, the Baudot code 00000, which was produced by XOR-ing a letter with itself, represented a NULL value, which couldn not be printed. A 5 key keyboard was originally used to type messages in the code- one key for each bit. The Baudot code was later adapted to work with a typewriter-like keyboard, and control characters were added to the code.

The Vernam cipher is considered unbreakable when it is used properly- by using a one-time pad, a random key that is the same length as the plaintext to avoid repeating it later in the message, which is destroyed after it has been used. With the key used to encrypt it, the ciphertext can easily be decrypted by XOR-ing again.

The Vernam cipher is also considered to have perfect secrecy- obtaining the ciphertext without the key does not leak any extra information about the plaintext.

While these methods are fine for hiding text, encryption on computers is usually done in blocks of bits, rather than characters. However, encryption is most easily demonstrated using text and these are the two methods required for the specification.

The problem with using ciphers such as the Vernam cipher and the Caesar cipher is how to exchange the keys, particularly in electronic communications. If the key used for any of these ciphers is intercepted by someone other than those intended to read the messages, they would be able to decrypt every message encrypted in the cipher and sent along a connection, making the exchange unsecure.

A solution to this is asymmetric encryption or public-key encryption, where there are two keys – one to encrypt a message and another to decrypt it on the other end.

RSA is an example of this type of encryption. It relies on the sender and the receiver each having a public key made up of two numbers, N and e, which can be known by anyone and a private key, made up of two numbers, N and d, which should be kept secret from everyone. The sender encrypts the data with the receiver’s public key, and the receiver uses his private key to decrypt it when he receives it. RSA encryption is a one-way function (Ciphertext = Plaintext^e MOD N) so the public key cannot be used to decrypt the message - the encryption is secure as long as no one but the intended recipient of the data knows the private key. For example, to encrypt the decimal ASCII value of the letter “K” (75), with the public key N 7471, e = 83:

Encryption:
Ciphertext = 75^{83} % 7471 = 342

Attempt to decrypt with the public key:
Plaintext = 342^{83} % 7471 = 5012 ≠ “K”

Decryption with private key N = 7471, d = 347:
Plaintext = 342^{347} % 7471 = 75 = “K”

Note that the encryption only works with the private key, trying to decrypt with the public key that it was encrypted with does not work.

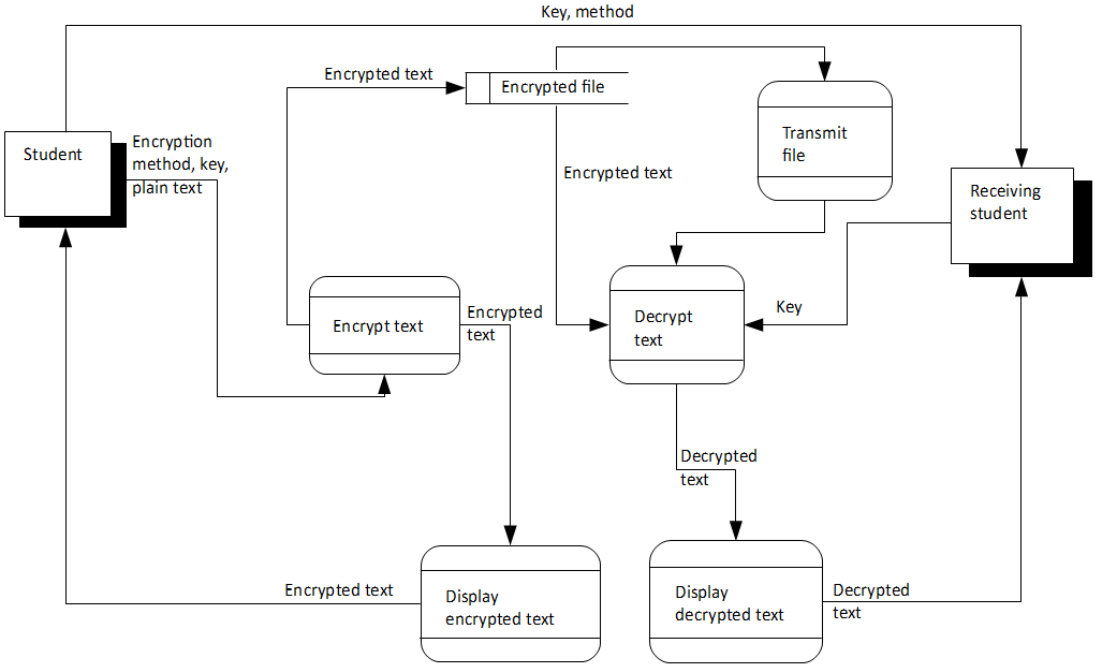
4. Identification Of The Prospective User(s)

   The Computing students will be the primary users for the system. Since it is to be used as a teaching aid, it should be easy for the students to use from the start, without requiring special training. The teachers may also use it to demonstrate things on the board, or to produce practise questions for the students to do. In both cases, it can be presumed that the users will have a good amount of computer experience.

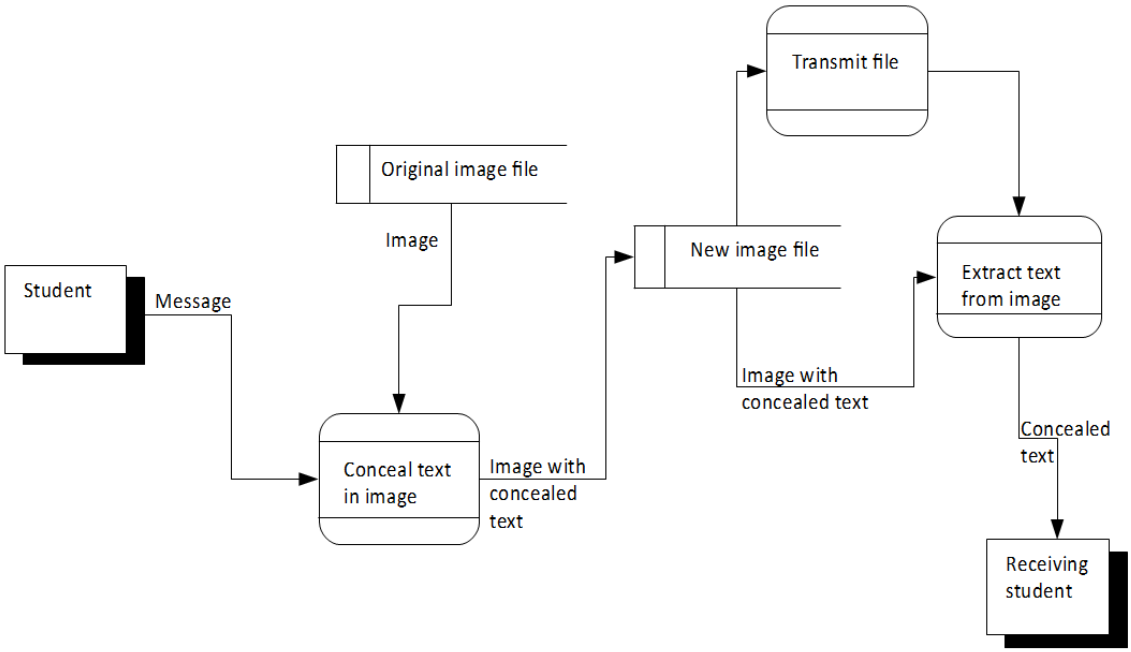
5. Identification Of User Needs
   1. The program should be able to encrypt and decrypt text using the Vernam and Caesar ciphers- it could show an animation/ the process on screen to help the students understand what is happening
   2. The program should be able to generate public and private keys so that data can be encrypted and decrypted using RSA
   3. The students should be able to encrypt and decode their own messages- this would be more fun and engaging
   4. The students should also be able to change the key of the ciphers
   5. It should be possible to hide messages in a picture using steganography- it would be good if they could see how the bit patterns were changed
   6. The program should display the image and be able to zoom in to see if the colours change
   7. Students should be able load or save their encrypted messages/images so they can email them to their friends, for them to decrypt
   8. Students should be able to connect via TCP/IP with a friend and be able to send messages and files across the connection

6. Data Flow Diagrams

1. Proposed System- Caesar Cipher, Vernam Cipher and RSA



2. Proposed System- Steganography



7. Objectives Of The Project
    1. User Need: The program should be able to encrypt and decrypt text using the Caesar cipher- it could show an animation/ the process on screen to help the students understand what is happening
       Specific Objectives:
        1. Allow the user to switch between encrypting and decrypting with the Caesar cipher
        2. Allow the user to enter a specific shift
        3. Allow the user to enter a message to encrypt
        4. Allow the user to enter a message to decrypt
        5. Demonstrate visually how the cipher is working by means of an animation.
    2. User Need: The program should be able to encrypt and decrypt text using the Vernam cipher- it could show an animation/ the process on screen to help the students
       Specific Objectives:
        1. Allow the user to switch between encrypting with the Vernam cipher and decrypting with the Vernam cipher
        2. Allow the user to enter a key of their choice
        3. Allow the user to enter a message to encrypt
        4. Allow the user to enter a message to decrypt
        5. Demonstrate on screen, how the cipher is working, by showing it working through each letter of the message
    3. User Need: The program should be able to generate public and private keys so that data can be encrypted and decrypted using RSA
       Specific Objectives:
        1. Generate public and private key
        2. Encrypt using public key
        3. Decrypt using private key
        4. Be able to send the public key to another computer across the network
    4. User Need: It should be possible to hide messages in a picture using steganography- it would be good if they could see how the bit patterns were changed
       Specific Objectives:
        1. Allow the user to load up a chosen bitmap image
        2. Allow the user to enter a message to hide in the image
        3. Retrieve a hidden message from an image
        4. Provide a zoom function which allows the user to zoom into the pixels to inspect
        5. Let the user compare the old image with the new image to see if a hidden message can be spotted
        6. Show the hexadecimal values of pixel colours
    5. User Need: Students should be able load or save their encrypted messages/images so they can email them to their friends, for them to decrypt.
       Specific Objectives:
        1. Have a save function to enable the user to save a text file containing their encrypted/decrypted messages
        2. Have a save function to enable the user to save a bitmap file containing a hidden message- as a new file rather than over the original
        3. Have a load function to enable the user to load a text file containing an encrypted message.
        4. Have a load function to enable the user to load a bitmap file either containing a message or to hide a message in it.

5. Allow the user to send messages to someone else across the network using TCP/IP

Note that in this section I have expanded upon the user needs to develop the objectives but I have grouped some user needs together, hence the number of major points (5) is less than the number of user needs (8).

8. Chosen Solution

The Vernam Cipher and the Caesar Cipher both work with text so a console based program could be possible for performing these. However, objectives 1.5 and 2.5 would be difficult to achieve as the demonstration will require a graphical interface to implement it properly. Furthermore, objectives 4.1, 4.4 and 4.5 all require images to be loaded and viewed so the program will use a GUI.
The interface will need to be intuitive so that students can get started in the lessons almost straight away. A complicated interface would require a lot more time to be spent on training the students to use it- time that may not be used or available in the current system and so shouldn't be required by the new system.

To produce a program with a GUI, I could use either Python or Java. I have more experience in programming with Python which could make the general development of the program easier, however, the GUI would have to be programmed completely manually as there are no development tools available to help produce the code. I have manually programmed a GUI in Python before but it was time consuming and difficult to get everything organised on the screen.
Java, on the other hand, has several IDEs available that offer a GUI Builder; I have some experience using the builder provided by the Netbeans IDE, which allows for the use of either Java Swing or Java AWT, two GUI modules for Java. Java Swing is the obvious choice for any GUI development in Java, as it offers a larger choice of components and it is better maintained by Oracle.

The system needs to establish and work with a TCP/IP connection to complete objective 5.5. Both programming languages have modules that allow this to be implemented – Python has the socket module and Java offers the java.net module for this. I have never used either of these modules, however, neither of them seem particularly difficult to use. They look very similar in terms of how they are used to create a TCP/IP connection.

For structuring the code itself, both Python and Java allow for object-oriented programming, so the program can be structured to define each different cryptographic method as a separate entity. The GUI-related methods can be kept separate from the processing methods more easily as the GUI can be managed by a separate class to the ciphers and steganography. Java is more strongly object-oriented and will allow for better object-oriented programming practises such as encapsulation.

The system will be developed in Java, as it will be easier and quicker to produce the GUI to fit a design, using the GUI Builder. Since it will be easier to produce to a design, it will be able to be quicker and easier to use for the students – this is required as they will only be using the system for a couple of lessons and so it wouldn't be worth training them to use it.
It will also make it easier to organise and structure the different parts of the code during development, as Java is strongly object-oriented.

Table of analysis performed:

Date	Activity	Actions
Oct	Initial chat with Mr M	Found out overview as to requirements and suggestion to investigate specification and past exam papers.
Nov	Looked at specification and past paper questions.	Form out list of proposed objectives to clear with Mr M
Dec	Chat with Mr M	Discussion had with Mr M to show current proposed ideas and concepts to be covered. Advice given to research more into steganography and cryptography. Look to create a proposed list of objectives.
Dec	Chat with Mr M	Proposed list of objectives modified and agreed (list provided above in analysis)

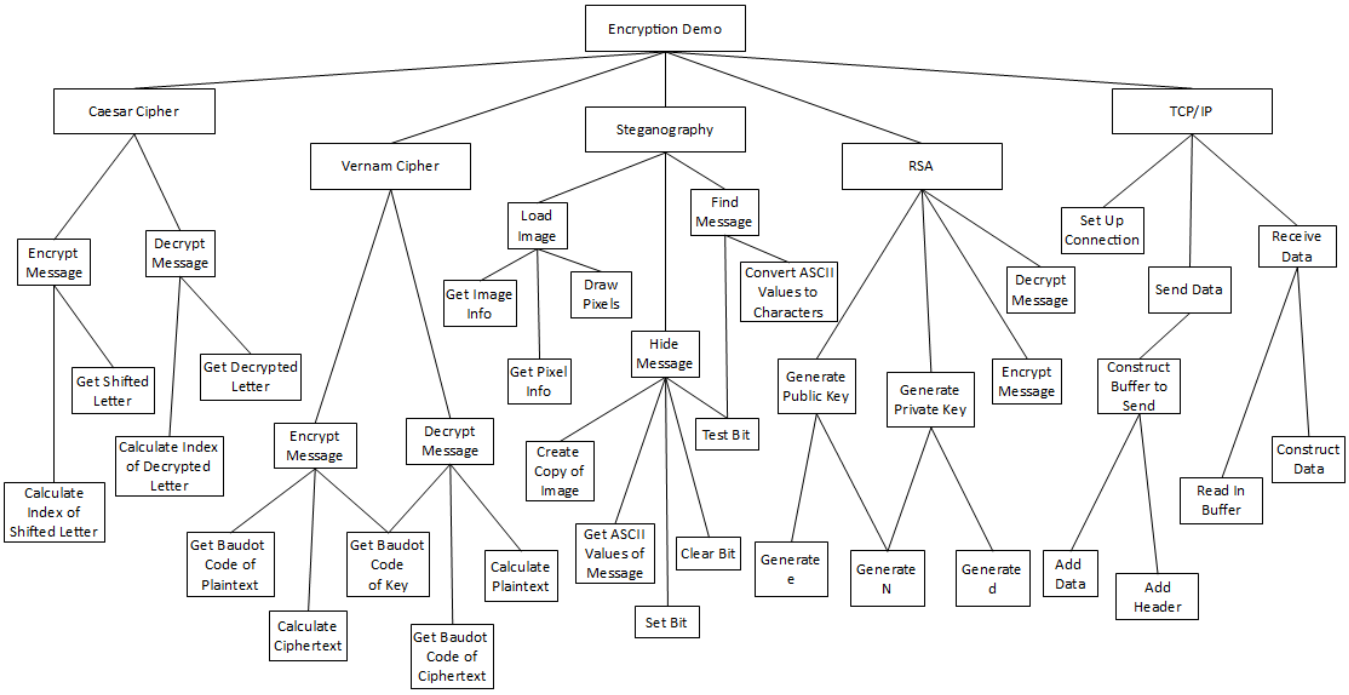
Documented Design

1. General Explanation

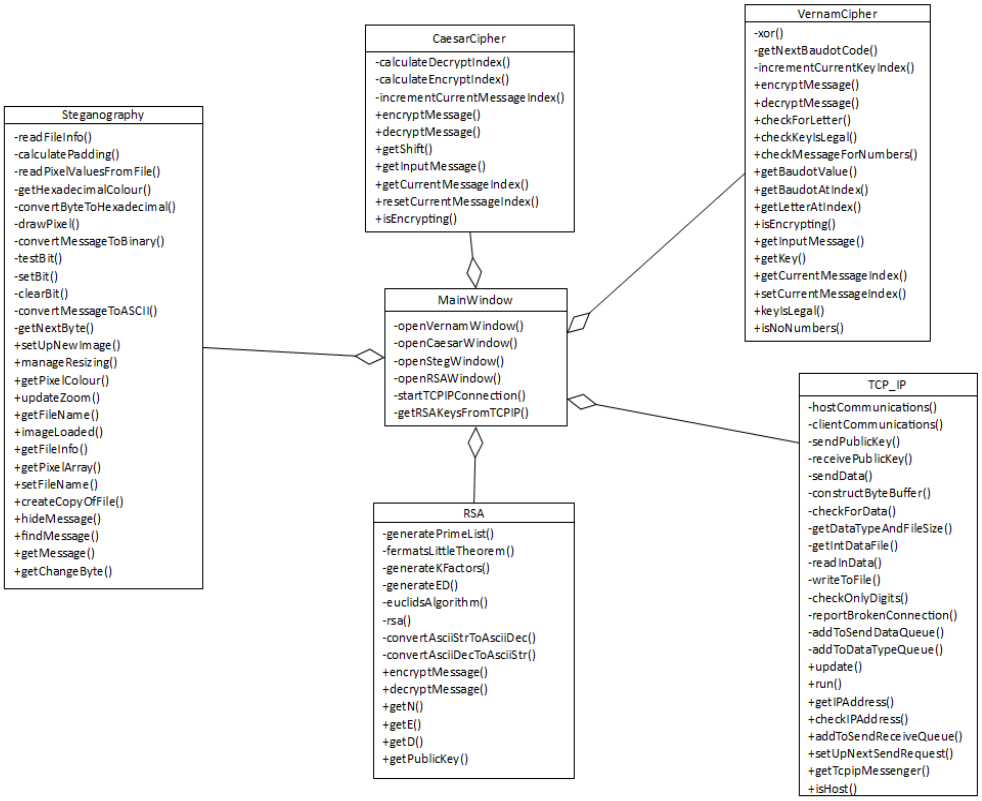
The system will be developed in Java, and since Java is heavily object-oriented, it will take advantage of this by using classes in the structure of the code. The system will use forms, and will use the Swing module for producing these. The system will be able to establish and use a TCP/IP connection and to do this, it will use the java.net module. The system will be able to save and load data in text files. In addition to text files, the system will be able to load a bitmap file and manipulate the bits of it to hide a message and read particular bits to find a message.

2. Description of Modular Structure of System

This diagram is a breakdown of the required functionality of each main part of the system.



3. <u>Class Definitions (diagram) and Details of Object Behaviours and Methods</u>
   This shows how the system will be implemented as classes in Java. It shows the key methods planned for the program but these may be broken down further when I start to develop it.



4. <u>File Organisation, Record Structure and Processing</u>
   The system will use two different types of files- text files and bitmapped images.
   It will be able to load in messages from text files to encrypt or decrypt and also stored encrypted messages in text files. This will be used with the Caesar Cipher, Vernam Cipher and RSA.
   Bitmapped images will be used for Steganography- the system will load up a bitmapped image and be able to hide a message or find a message already hidden in it. To do these three things, it will need to know the structure of a bitmapped file, as shown in Appendix 1.4.

5. Identification of Validation Required

Form	Data Item	Check Type	Check Details
Main Menu	Host IP Address	Format	Must be in format of 4 numbers separated by dots (.)
Main Menu	Host IP Address	Range	Each quarter of the IP address must be a number between 0-255
Main Menu	Host IP Address	Length	The IP address can be a max of 15 characters, including the dots between each number
Vernam Cipher	Input Text	Length	The text to encrypt/decrypt must be at least one character long
Vernam Cipher	Key	Format	The key may only contain letters from A-Z
Vernam Cipher	Key	Length	The key must be at least one character long (preferably the same length as the input text)
Caesar Cipher	Input Text	Length	The text to encrypt/decrypt must be at least one character long
Steganography	File to Load	Format	File to load must be a valid bitmap file
Steganography	Byte to Hide Message in	Format	Option must be “Red”, “Green”, or “Blue”
Steganography	Input Message	Format	The message may only contain ASCII characters
Steganography	Input Message	Length	The message to hide must at least one character long
RSA	Public Key	Length	Both parts of the public key must be at least one digit long
RSA	Public Key	Format	The public key can only be made up of numbers
RSA	Input Message	Format	The message may only contain ASCII characters
RSA	Input Message	Length	The message to encrypt/decrypt must be at least one character long
Load Window	File to Load	Format	The file to load must be a text file or bitmap file
Save Window	File Name	Format	The file name must only contain legal characters and have the right suffix for the file type

6. Identification of Suitable Algorithms for Data Transformation

Vernam Cipher:

Encrypting: This algorithm encrypts a message using the Vernam Cipher. The Vernam cipher takes a string of characters for the message and a string of characters for the key. For each letter in the message, the Vernam Cipher gets the next letter from the key, converts both letters to their Baudot codes and XORs them to produce the cipher text.

1. If the next character is a letter, get the next letter of the key to encrypt with.
    - Letters are converted to Baudot and encrypted, but other characters are added to the ciphertext as they are.
2. Get the Baudot codes for the letter to encrypt and for the letter from the key.
3. XOR the two Baudot codes to get the ciphertext.
4. Increment the pointer to the next character in the message and the pointer to the next character in the message.
    - To increment the pointer for the message, it is simply adding 1 to the pointer's current value.
    - The key's pointer is slightly different, because it is possible that it will have to loop back to the start to encrypt the whole message if the key is shorter than the message. Looping back to the start of the key is easily done by modular dividing the current pointer + 1 by the length of the key: new key index = (current key index + 1) % length of the key

```
newKeyIndex ← (currentKeyIndex + 1) MOD keyLength
```

Decrypting: This algorithm decrypts a message in the Vernam Cipher. It takes a string of Baudot codes and the key that was originally used to encrypt the message. It takes each Baudot code, one at a time, gets the Baudot code for the next letter in the key, and XORs the two codes. This produces a third Baudot code which maps back to the original letter that was encrypted.

1. Get the next Baudot code to decrypt.
    - Since the Baudot codes only represent letters, any other characters are copied to the ciphertext as they are. This means that the offset for the start of a code is not always the same for each message and it will have to be searched for – iterate through the message until a 0 or a 1 is found.
    - This method means that 0 and 1 cannot be allowed in the plaintext as they are not letters and would be copied to the ciphertext as they are, and this would cause problems at this point in decryption.
    - So, once a 0 or a 1 has been found, it can be presumed that it is part of a Baudot code, and the next four characters after it should be the rest of the code- use the five characters as the Baudot code to decrypt.

```
Set the Baudot code B to equal an empty string
WHILE the current position pointer in the message
string is less than the length of the message and the
length of B is less than 5
   Set the nextChar to equal the character in the
message string at the current position
   IF nextChar is a 0 or a 1 THEN
         //including the current character, add the
next five characters to B
      FOR X ← 0 to 4
```

```
            B ← B + the character at the current position
pointer
            Increment the current position pointer
        ENDFOR
    ELSE
    B ← B + nextChar
            Increment the current position pointer
            Return B
    ENDIF
ENDWHILE
Return B
```

2. Get the Baudot code representing the next letter in the key to decrypt with.
3. XOR the two Baudot codes together to get the Baudot code of the original letter.
   - This is the same as the XOR function for encrypting in the Vernam Cipher.
4. Get the letter represented by this resulting Baudot code.

<u>Caesar Cipher</u>:

Encrypting: This algorithm encrypts a message using the Caesar Cipher. The Caesar Cipher takes a number, the shift, and swaps every letter in the message for the letter this number of positions down from it.

1. Get the index in an array of the alphabet of the letter to encrypt.
   - Each letter of the alphabet is stored as an element in an array.
   - For example, “A” would be at index 0, “B” would be at index 1, “C” at index 2, etc.
2. Calculate the position of the cipherletter by adding the shift to the index of the original letter.
   - The index of the encrypted letter = the index of original letter + shift
   - But, this produces indices out of range with letters at the end of the alphabet- if an index goes past 25, it should loop back round to 0, e.g. encrypting “Z” with shift 1 produces an index of 26, which should loop back to index 0, and the letter “A”.
   - The solution to this problem is modular dividing the index by 26 (e.g. 26 mod 26 = 0).

   ```
   The index of the encrypted letter ←(the index of the
   original letter + shift) MOD 26
   ```

3. The encrypted letter can now be obtained by accessing the element in the array at the calculated index.

Decrypting: This algorithm decrypts a message using the Caesar Cipher. The Caesar uses the given shift to work out the original letter for each of the letters in the encrypted text.

1. Get the index of the encrypted letter in the array of the alphabet.
2. Calculate the index of the decrypted letter in the array of the alphabet.
   - Index of the decrypted letter = index of the encrypted letter – shift % 26
   - If the letter that has been encrypted is towards the end of the alphabet, the above calculation produces a negative index. In this case, taking the

magnitude of the calculated index from 26 (or adding the entire value, with sign, to 26) gives the index of the original letter.

```
Index of decrypted letter ← (index of encrypted letter –
shift) MOD 26
IF index of decrypted letter is less than zero THEN
      Index of decrypted letter ← 26 + index of decrypted
letter
ENDIF
```

3. Access the element at the calculated index to get the decrypted letter.

Steganography Hide Message Algorithm: This algorithm takes a string of characters to hide in a bitmapped image, converts each of them to their 8 bit binary ASCII values and then iterates through each pixel's values in the pixel array and performs the appropriate procedure (set or clear) for the value to store the next bit of the string of characters.

1. Convert the characters in the message to their 8 bit binary ASCII values.
   - Get the integer value of each character and convert it to binary.
   - Add a terminator byte to the end of the message (00000000 will be suitable as it is a value that is unlikely to appear in the message anyway because it represents null)- this will used when finding the message to detect when the whole message has been read.
2. Iterate through each bit of the message and each pixel of the file, test the file bit to hide the message bit in and clear or set it depending on the result:

```
FOR each bit B in the message
      Move to the pixel in the file to store B in
      Set P to equal the current value of the byte to
      store B in
      //Test the position in the byte where the message
      bit will be stored
      Set result R to equal (2^bit position to change) AND P
      IF R equals the bitmask THEN
            IF B equals 0 THEN
                  //Clear the bit
                  Write (255 – 2^bit position to change) AND P to
            file
            ENDIF
      ELSE
            IF B equals 1 THEN
                  //Set the bit
                  Write (2^bit position to change) OR P to file
            ENDIF
      ENDIF
ENDFOR
```

RSA Key Generation: This algorithm selects two prime numbers at random and uses them as the foundation for creating a public and private key pair for encrypting and decrypting messages using RSA.

1. Generate list of prime numbers- since 2 is the only even prime number, iterate through odd numbers up to a limit and use a primality test to decide whether it is prime- add it to the list if it is.

- Fermat's Little Theorem states that a number p is prime if (x^p – x) is divisible by p (where x is any integer between 2 and p-1). Every possible value of x should be tried, although as soon as one value is found where (x^p – x) is not divisible by p, then p can be declared not prime.
- For larger numbers, the algorithm is obviously less efficient, because there are more possible values of x to check. In the formal use of RSA, extremely large prime numbers are used to make decryption without the key even harder, but for a demonstration of the concept of asymmetric encryption, restricting the system to a smaller range of prime numbers will not be a problem.
- In addition, there are numbers that Fermat's Little Theorem passes as prime although they are actually not- these are called Carmichael Numbers and the first Carmichael Number is 561. Restricting the generation of prime numbers to numbers under 561 will avoid these.

```
Initialise array to store prime numbers in
//Check each odd number below 561 for primality
FOR each odd integer I between 3 and 560 inclusive
      Set isPrime to true
      //Use Fermat's Little Theorem
      FOR each integer X between 2 and I-1
            IF X^I - x mod I is not 0 THEN
                  Set isPrime to false
                  Break the loop
            ENDIF
      ENDFOR
      IF isPrime is true THEN
            Add I to list of primes
      ENDIF
ENDFOR
```

2. Two numbers from the list of primes are picked at random- these are p and q.
3. Calculate p*q = N. N is one half of the public key and one half of the private key.
4. Phi is also calculated by (p-1)(q-1). (Euler's totient of N → number of positive integers less than N that are coprime to N ← the greatest common divisor between two numbers is 1 if they are coprime (see below))
5.

```
Initiate empty list of factors and K
Set M to a random integer between 1 and 6 inclusive
Set tempK to (phi*M) + 1
//attempt create a list of factor pairs of K
WHILE the length of list of factors is 0
      FOR each integer Z between 2 and the square root of K
            IF K mod Z equals 0 THEN
                  Add [Z, K div Z] to list of factors
            ENDIF
      ENDFOR
ENDWHILE
Return K
```

6. e and d are a pair of numbers whose product equals K, i.e. they are a factor pair of K. They also have to be coprime to phi.
   - K is a number such that (K mod phi = 1) and K must not be prime.

- Possible values of K can be obtained by generating multiples of phi and adding 1 to them. We can generate random numbers to multiply phi by until a value is found that matches the requirements, rather than iterate through a series of numbers and multiply phi by each in turn. This will greatly reduce the number of calculations in many cases, since K is already random- the algorithm can stop when a value is found and a value of K may be found sooner.
- To check that K is not prime, attempt to generate a list of pairs of its factors- if a list cannot be produced, K is prime. If a list is produced, a value of K has been found and the algorithm can finish.
- A way of getting the factors of K is to iterate through every integer between 2 and the square root of K and divide K by each of them- the result will be an integer if the divisor is a factor – the result and the divisor are added to the list as the next factor pair.
- Fermat's Little Theorem could be used again here to test K's primality but since the values of K are likely to be very large, it would be very inefficient, and the list of factor pairs will be used again in step 6 anyway.
- Iterate through the list of factor pairs until a pair is found where both numbers in the pair are coprime to phi – this can be done by checking the greatest common divisor (gcd) between each of the numbers and phi (if they are coprime, the gcd will be 1).
- Euclid's Algorithm is a recursive algorithm that calculates the greatest common divisor between two numbers, a and b.
    - If a is 0, return b.
    - If b is 0, return a.
    - Otherwise, calculate the remainder, r, when a is divided by b and call the function again with b and r.
- So, when the gcd of each number and phi is returned as 1, the values of e and d can be set.

```
EuclidsAlgorithm(a, b)
      IF a equals 0 THEN
            Return b
      ENDIF
      IF b equals 0 THEN
            Return a
      ENDIF
      Set r to equal a mod b
      Return EuclidsAlgorithm(b, r)

FOR each pair of factors P in the list of factors
      IF EuclidsAlgorithm(P[0], phi)equals 1 and
EuclidsAlgorithm(P[1], phi) equals 1 THEN
            Set e to equal P[0]
            Set d to equal P[1]
            Break the loop
      ENDIF
ENDFOR
```

7. e and N make up the public key, and d and N make up the private key.

8. User Interface Design (HCI) Rationale

The system will use a graphical user interface. The interface will follow a consistent design across all of the different screens- for example, every screen will have the same colour scheme of a light blue background, dark blue border and white buttons.

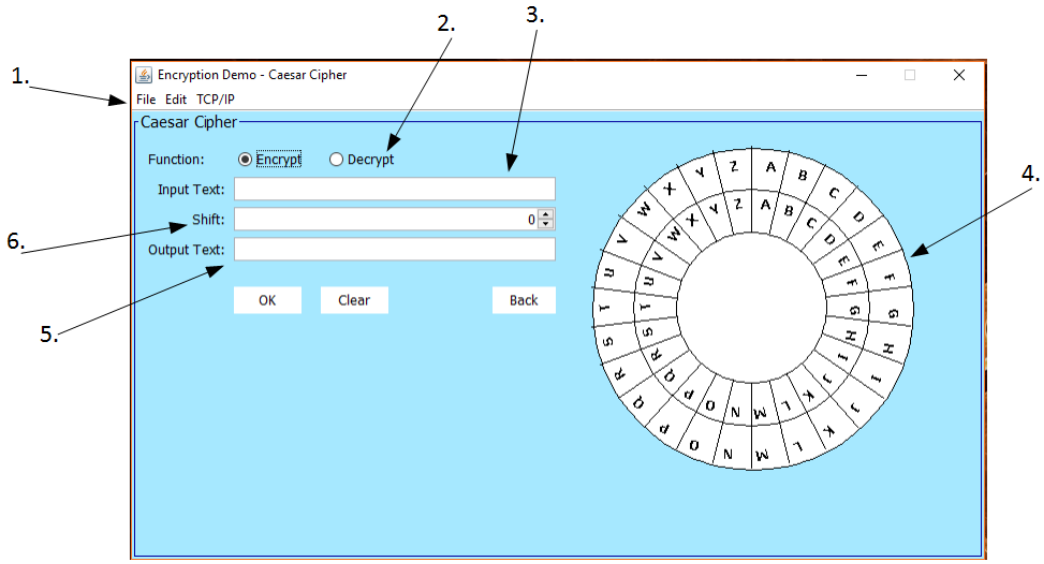
Consistency in the layout of the windows will be achieved by having reoccurring controls in the same place, for example, each of the encryption methods will have a button that submits the input data to the program and starts the requested method (encrypt or decrypt) and this button will be in the same place for each of them. The windows will mainly have text boxes for inputting a message and the key to encrypt or decrypt it with (although the Caesar Cipher will use a spinner to input the shift to use) and these will be laid out in the same way for each – the message input will be at the top and the shift or key input beneath it. The outcome of the chosen method will be displayed in a final text box beneath these.

The Vernam and Caesar Cipher windows will each display an aid to help the user understand the working in the current operation being performed. The Vernam Cipher will display the Baudot codes used in encrypting and decrypting and their matching letters in a table for reference, while the Caesar Cipher will display the Caesar wheel showing the currently selected shift setting. In both cases, the aids will be displayed on the right hand side of the GUI.
In addition, every window will have a toolbar at the top that will offer functions common to each of the different features of the system, for example, the user will be able to send and receive data from a TCP/IP connection and save and load data in text files, regardless of whether they are using any of the encryption methods or Steganography.

The GUI will use common components, such as buttons and combo boxes as people are used to using these and they will make it easier to use the system.

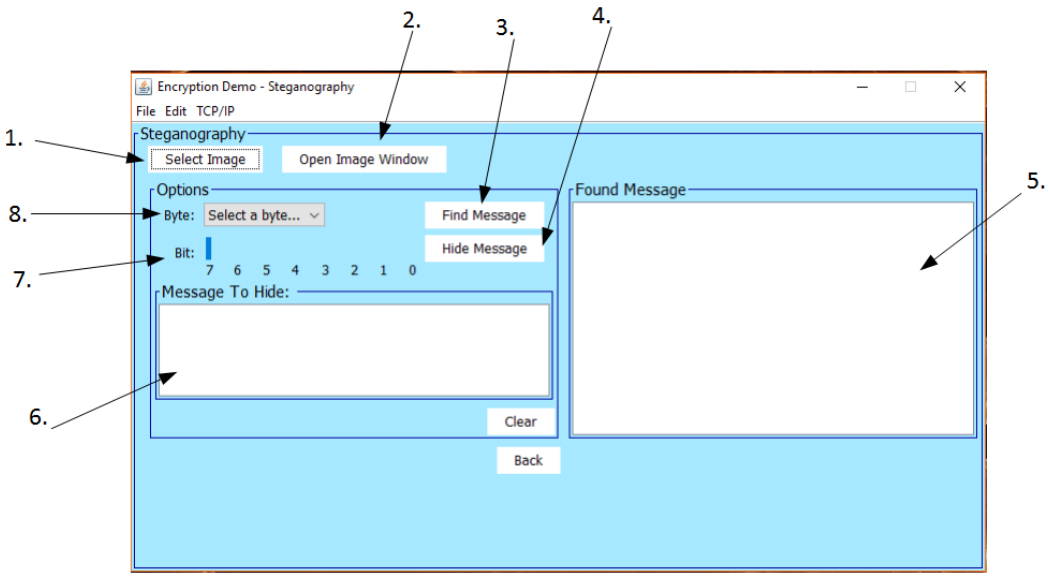
9. <u>UI Sample of Planned Data Capture and Entry Designs/ UI Sample of Planned Valid Output Designs</u>

<u>Caesar Cipher Sample:</u>



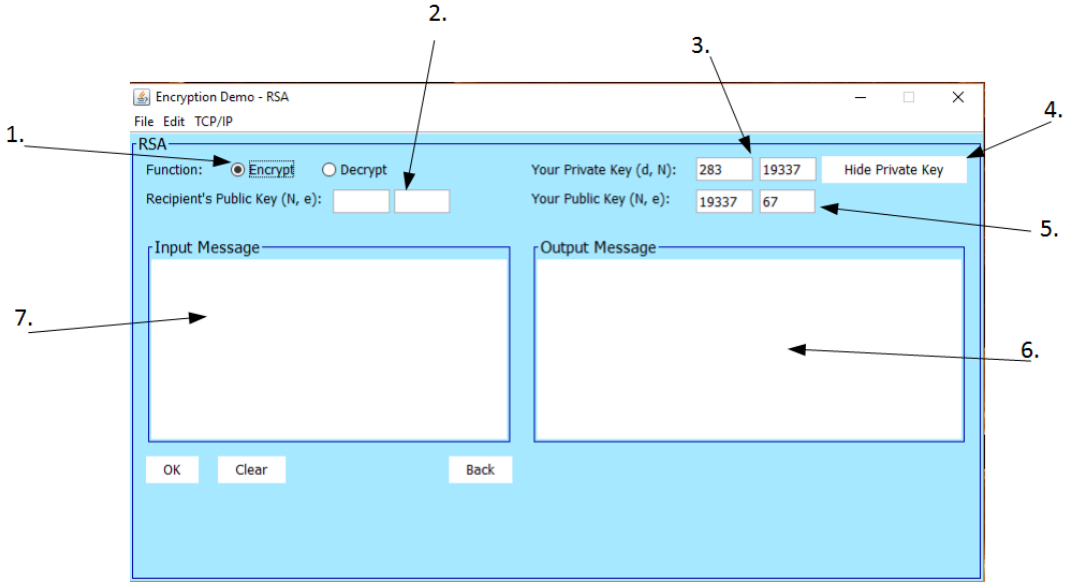
1. The toolbar will be visible on every method's screen, and will allow the user to perform common tasks for all methods, such as loading and saving files and using TCP/IP.
2. The user will select whether they want to encrypt or decrypt their message by selecting one of the radio buttons displayed.
3. Simple textbox for inputting the message to work with (restrict to small message sizes as the demonstration will take time to run?)
4. The Caesar wheel will provide a graphic display of the shift currently chosen by the user. Whenever the shift is changed using the spinner (see below) the wheel will update to show this change. The wheel will also be used for demonstration when encryption or decryption is being performed- the current letter being processed will be highlighted along with the corresponding plaintext or ciphertext.
5. As the message is encrypted or decrypted letter by letter, the output textbox will update to include the latest resulting letter in the total result.
6. Simple textbox for outputting the result to the user.

Steganography Sample:



1. Opens a dialog for the user to pick which bitmap file they want to load and use for Steganography.
2. The chosen image will be displayed in a separate window, which will open automatically when a new image is selected. If the window is closed, the user will be able to reopen the window by clicking this button.
3. The user will be able to extract a message hidden in the currently loaded bitmap file by clicking the Find Message button. In the event that there is no message in the image, the user will be notified of this as well.
4. The user will be able to hide a message in a similar way to finding the message. However, they will need to first input which byte of each pixel they want to hide the message in, which bit position in the byte they want to use and the message to hide before a message can be hidden in the image.
5. When a message has been found in the image, it will be displayed on the right of the screen.
6. For inputting the message to hide in the image, the user will use this textbox. This will be larger than the one for the Caesar Cipher (and also the Vernam Cipher) as there won't be an animation shown to slow down the process, making it feasible for the user to use a longer message.
7. The bit position will be chosen using a slider set up to display how the bit positions are actually laid out in a byte. The slider will need to have a default value; this will be bit 7, the most significant bit. The program won't object if the user doesn't change the slider as it will function using the default value.
8. For the byte to use, the user will pick one of three options from a combo box- Red, Green or Blue.

RSA Sample:



1. In the same way as on the Caesar Cipher sample, the user will choose whether to encrypt or decrypt by selecting one of the two radio buttons.
2. The user will enter the public to encrypt a message with using two text boxes, one for each number in the key. When a TCP/IP connection is started, the public key of the user's partner will automatically be filled in in these boxes, although the user will be still be able to choose another key instead.
3. On the right of the screen, the user's own keys will be displayed. The private key will be displayed above the public key.
4. Since the private key is meant to be kept secret, the user will able to hide their private key using a toggle button. Clicking the button again will display the private key again.
5. The user's public key will be displayed underneath their private key. The numbers will be displayed the same way round as they should be entered for encryption for coherence.
6. An encrypted or decrypted message will be displayed on the right of the screen, similar to a discovered message in the Steganography sample.
7. The message that the user wants to encrypt or decrypt will be entered in the textbox on the left. Like with the Steganography sample, there won't be an animation for RSA, so the user can work with longer messages without it taking too long to do.

10. Use of Libraries

For the system to be able to make a TCP/IP connection and have a GUI, there are two main libraries that will be used – java.net and javax.swing respectively.

To develop the TCP/IP interface, the ServerSocket and Socket classes of java.net will be used. The ServerSocket sets up a socket on the host end of the communications to listen for a client requesting to connect. The Socket sets up the client end of the connection and attempts to connect to the host specified by the given IP address on the specified port. These must, of course, match the host's socket.
The differentiating between the host and client will only extend as far as starting a TCP/IP connection in the system, as both ends will be able to send and receive data, regardless of whether they were host or client in the initial setup.

Swing is a GUI manager for Java which offers the common components such as buttons, labels, and text entries that people are used to using. However, other classes can also inherit from these components, allowing for custom components to be created, using polymorphism. This will be needed to display the Caesar wheel and the bitmapped image being used for Steganography, as Swing doesn't offer a canvas-like component to draw on as a standard component, but one can be created by having a class inherit from the JPanel class and override its paintComponent() procedure.

Testing

1. Test Plan

Test No.	Form	Purpose	Test Data	Expected Result	Type	Result
1	Vernam Cipher	Check that a valid message can be input to be encrypted.	I will enter “hello” as a message.	Message accepted	Normal	Pass
2	Vernam Cipher	Check that an invalid message can not be entered for encryption	I will enter “12345” as a message.	Message rejected	Erroneous	Pass
3	Vernam Cipher	Check that a message of legal characters that can’t be encrypted is handled properly	I will enter “!!!!” as a message.	Message accepted and encrypted as just “!!!!”.	Extreme	Pass
4	Vernam Cipher	Check that a valid key is accepted	I will enter “abcdef” as a key.	Key accepted	Normal	Pass
5	Vernam Cipher	Check that an invalid key cannot be input	I will enter “678!:)” as a key.	Key rejected	Erroneous	Pass
6	Vernam Cipher	Check that the right Baudot code is used for each letter in the message	I will enter “hello” as a message.	The system should use the codes “11001”, “01000”, “11011”, “11011” and “11100”	Normal	Pass
7	Vernam Cipher	Check that the right Baudot code is used for each letter in the key	I will enter “after” as a key.	The system should use the codes “10000”, “01101”, “10110”, “01000”, “00111”	Normal	pass
8	Vernam Cipher	Check that the system correctly encrypts the input message using the given key	I will enter “hello” as a message and “after” as a key.	Message encrypted correctly – the result should be “01001”, “00101”, “01101”, “10011”,	Normal	Pass

Test No.	Form	Purpose	Test Data	Expected Result	Type	Result
				“11011” and the animation should show this correctly.		
9	Vernam Cipher	Check that system correctly gets the next Baudot code from a message to decrypt	I will use the message “111110000 00111011 10010100, 110111010000 100011000 100110101!” encrypted using the key “secretkey”.	Correct Baudot codes are taken from the message and decrypted correctly.	Normal	Pass
10	Caesar Cipher	Check that a message of legal characters that can’t be encrypted is handled properly	I will enter “<>:45@” as a message, and use shift 7.	Message accepted and encrypted as “<>:45@”	Extreme	Pass
11	Caesar Cipher	Check that the correct index is calculated for getting the encrypted letter from plaintext	I will use “A lion escaped from the zoo” as a message and 10 as the shift.	Calculated indices should be 10, 21, 18, 24, 14, 2, 12, 10, 25, 14, 13, 15, 1, 24, 22, 3, 17, 14, 9, 24, 24	Normal	Pass
12	Caesar Cipher	Check that the correct index is calculated for getting the original letter from ciphertext	I will use “PMTTW EWZTL” as a message and 8 as the shift.	Calculated indices should be 7, 4, 11, 11, 14, 22, 14, 17, 11, 3	Normal	Pass
13	Caesar Cipher	Check that the currently selected shift is shown on the Caesar wheel correctly	I will use shift 5.	Each letter on the wheel lines up with the correct cipherletter (“A” should line up with “F”)	Normal	Pass

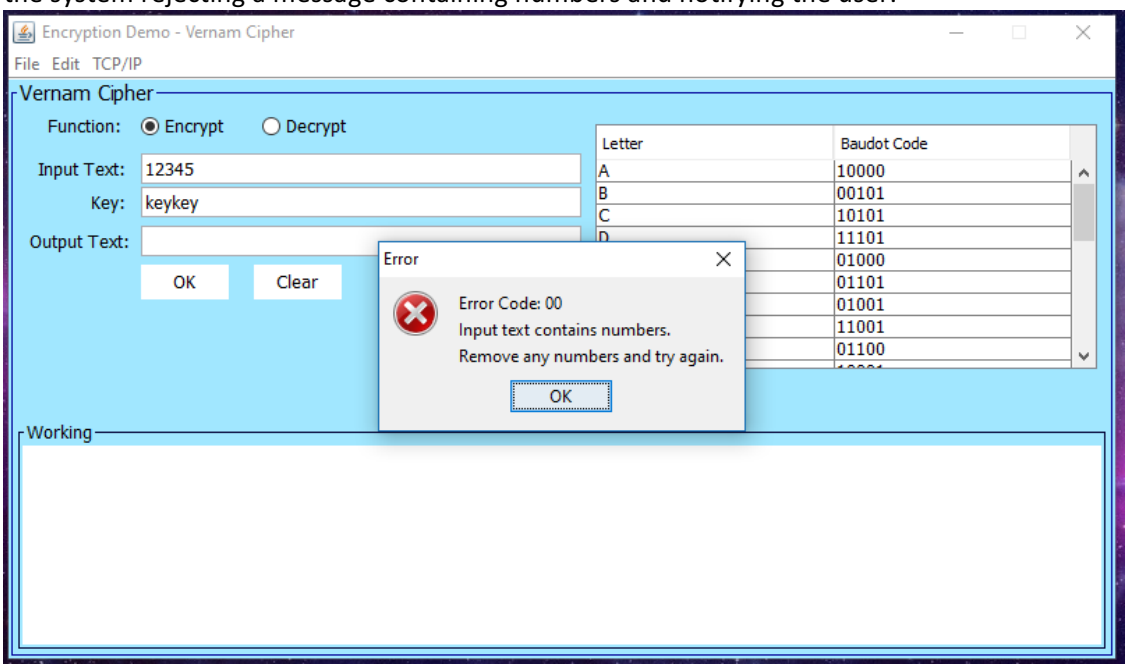
Test No.	Form	Purpose	Test Data	Expected Result	Type	Result
14	Caesar Cipher	Encrypt a whole message	I will enter the message “Bonjour” with shift 8.	Message encrypted to “JWVRWCZ”	Normal	Pass
15	Caesar Cipher	Decrypt a whole message	I will enter the message “URYYB JBEYQ” with shift 13.	Message decrypted to “hello world”	Normal	Pass
16	Steganography	Check that a valid bitmapped image can be loaded	I will choose to load up the bitmapped image “artemis.bmp”.	Chosen file is accepted and displayed correctly	Normal	Pass
17	Steganography	Check that an invalid file is not accepted by the system	I will attempt to load up the file “message1.txt”.	Chosen file is rejected	Erroneous	Pass
18	Steganography	Check that a valid message is accepted for hiding in a bitmap file	I will enter “inconspicuous secret message” as a message to hide.	Message accepted	Normal	Pass
19	Steganography	Check that an invalid message is not accepted for hiding in the bitmapped image	I will enter “¡Hola!” as a message to hide.	Message rejected	Erroneous	Pass
20	Steganography	Check that a message hidden in an image is found and reconstructed correctly	I will search for a message in an image hiding the message “hello world :)” in bit 4 of the blue bytes.	Message found	Normal	Pass
21	Steganography	Check zoom feature correctly shows message.	I will look for signs of a message hidden in bit position 7 (leftmost bit) of the green bytes.	Green pixels should be visible on the image with the message hidden in it	Normal	Pass
22	Steganography	Check that message is encoded in correct selected bit.	I will use an image hiding the message “hello!!” in bit 7 of the green bytes.	The binary values of the pixels should only show changes	Normal	Pass

Test No.	Form	Purpose	Test Data	Expected Result	Type	Result
				made to the most significant bit of the second byte of each pixel		
23	RSA	Check that numbers are correctly identified as prime	The system will test all numbers between 2 and 561 exclusive for being prime.	Prime list should not contain non-prime numbers	Normal	Pass
24	RSA	Check that valid values for K are produced for the chosen values of p and q	I will use the values 73 and 7 for p and q respectively.	Values of K should be non-primes and should be equal to one plus a multiple of phi	Normal	Pass
25	RSA	Check that message encrypted with a public key is decrypted correctly using the corresponding private key	Message: “0000000 5120002700001 000 0000001000000 072 9007290000100 001 0006400001000 000 0001000010000 100 2160000100001 001 25” Public Key: 36769, 3 Private Key: 60407, 36769	Message is correctly decrypted	Normal	Pass
26	RSA	Check that a valid message is accepted for encryption	I will enter “meet at the fountain.” as a message.	Message accepted	Normal	Pass
27	RSA	Check that an invalid message is rejected for encryption	I will enter “meet at the café!”	Message rejected	Erroneous	pass

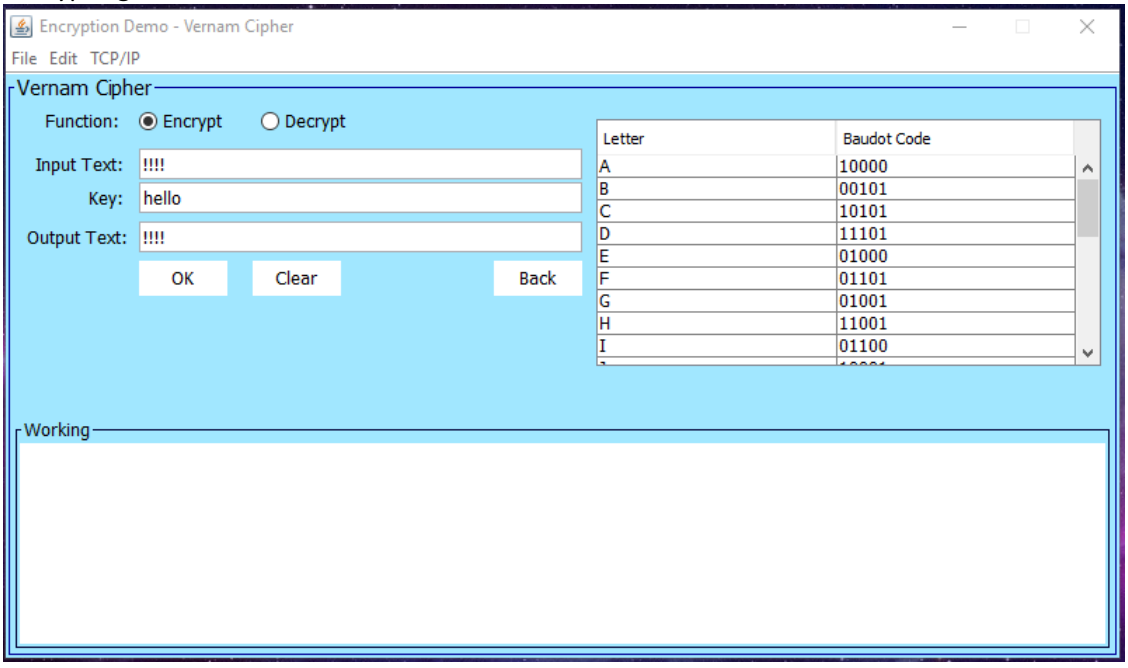
Test No.	Form	Purpose	Test Data	Expected Result	Type	Result
28	RSA	Check that a valid message for decryption is accepted	I will enter “00010000102400 01000031250000 02430243” as a message	Message accepted	Normal	Pass
29	RSA	Check that an invalid message for decryption is rejected	I will enter “000001010608291!” as a message	Message rejected	Erroneous	Pass
30	Main Menu (TCP/IP)	Check that a valid IP address is accepted	I will enter “192.168.0.6” as an IP address	IP address accepted	Normal	Pass
31	Main Menu (TCP/IP)	Check that an invalid IP address is not accepted	I will enter “192.168.300.270” as an IP address	IP address rejected	Erroneous	Pass
32	Main Menu (TCP/IP)	Check boundary IP addresses are accepted	I will enter “255.255.255.255” and “0.0.0.0” as IP addresses	IP addresses accepted	Extreme	Pass
33	Main Menu (TCP/IP)	Check that a connection can be successfully made	N/A	Connection is successfully started	Normal	Pass
34	Main Menu (TCP/IP)	Check that the public key of the other user is correctly received	Connection between two users, one with public key “47921, 7” and one with public key “75301, 3”.	Public key is received	Normal	Pass
35	Main Menu (TCP/IP)	Check that a sent message is successfully received	I will send the message “TQXXA!” across the connection	Message received	Normal	Pass
36	Main Menu (TCP/IP)	Check that a sent file is successfully received	I will send the file “artemis.bmp” across the connection	File received	Normal	Pass

2. Test Results

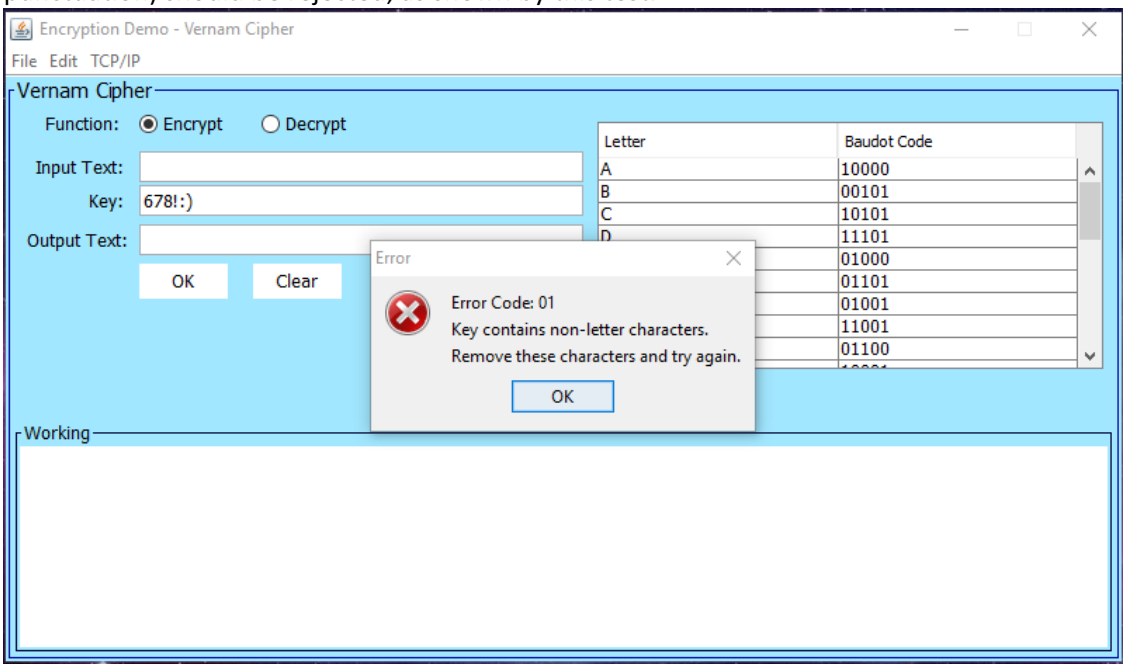
Test 2: The Vernam cipher cannot take numbers in the plaintext message- there are no Baudot codes for numbers so they would have to be encrypted as themselves, but encrypting 1s and 0s as themselves would make it impossible for the system to differentiate between part of a Baudot code and an actual 1 or 0 when decrypting. The screenshot shows the system rejecting a message containing numbers and notifying the user.



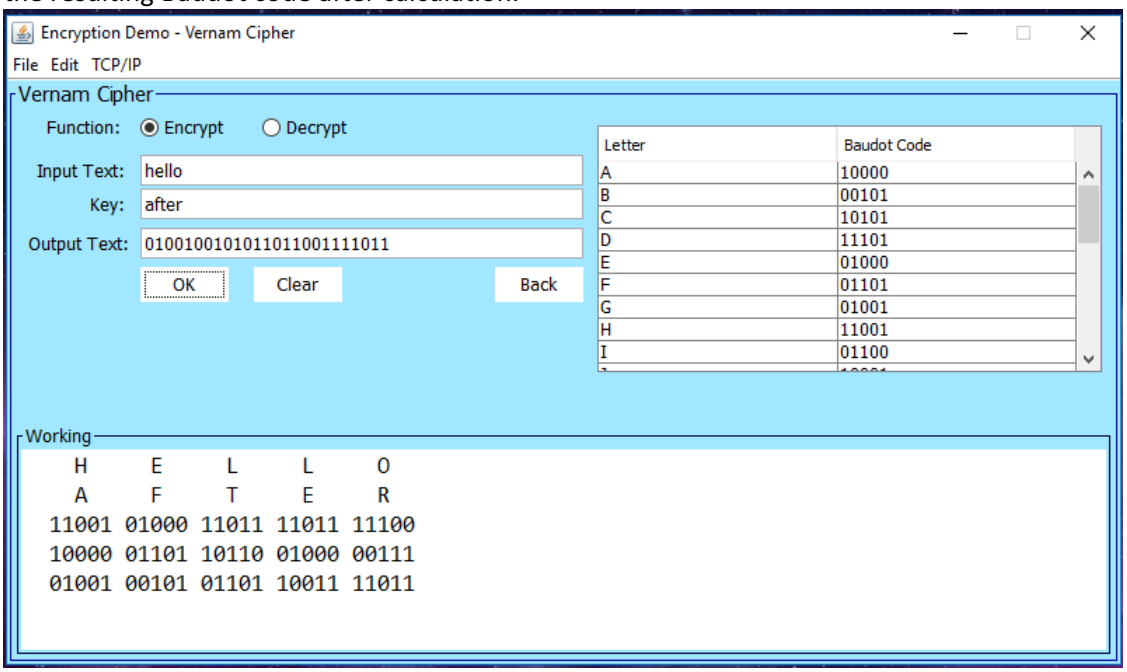
Test 3: Punctuation also isn't represented by the Baudot code used by the system, but since it won't affect the decryption, the system should be able to handle it as legal characters. The screenshot shows the punctuation in an input message being handled by the system encrypting it as itself.



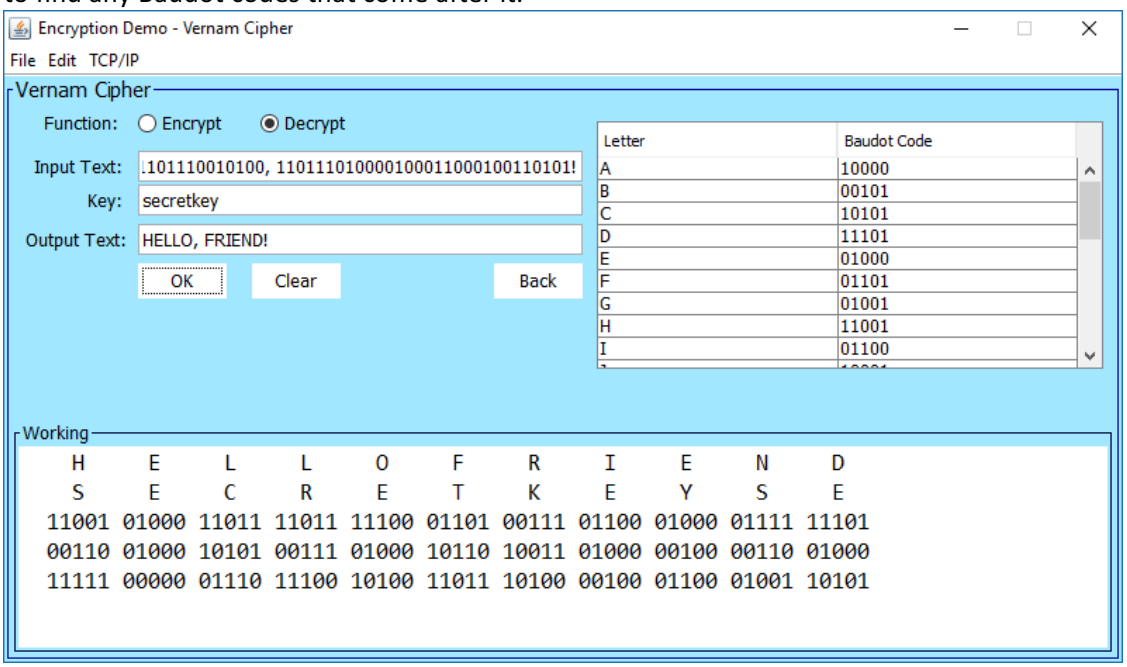
Test 5: The key for Vernam cipher encryption should only contain characters that can be represented by Baudot codes as any other characters don't have a corresponding code. Any keys containing characters that are not represented by Baudot codes (such as numbers and punctuation) should be rejected, as shown by this test.



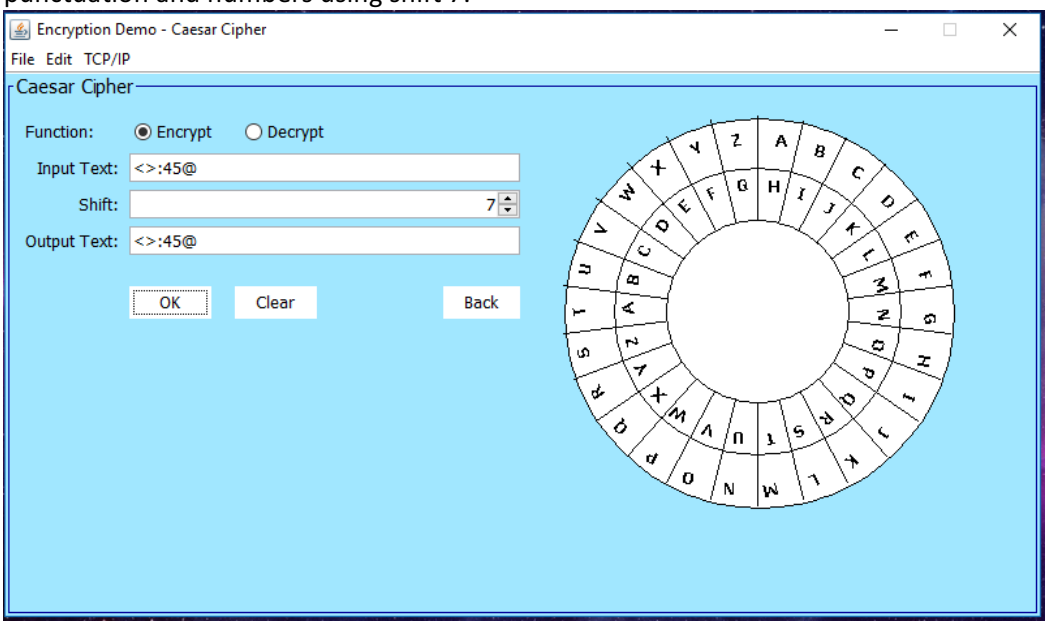
Test 8: To encrypt a message, the system should go through the message one character at a time. If the character is a letter, the Baudot code for it should be found. The Baudot code of the letter in the key to encrypt it with should also be obtained. The two codes should then be XOR-ed together to produce a third code, the ciphertext.
This test checks that the right codes are produced after the XOR calculation. The text box at the bottom of the window has three rows of Baudot codes- the first row being from the letter in the message, the second row's code is the key's Baudot code and the bottom row is the resulting Baudot code after calculation.



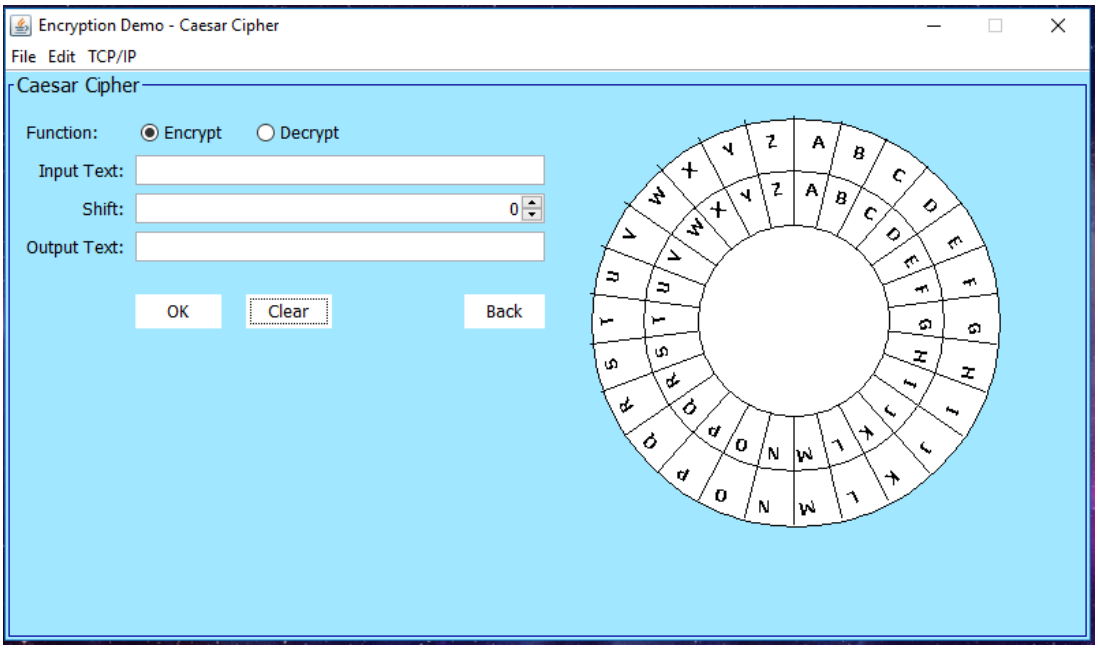
Test 9: When the system decrypts a message, it has to extract the five bit Baudot codes one at a time. However, if there was punctuation in the plaintext, this will be in ciphertext to decrypt (punctuation doesn't have corresponding Baudot codes so is encrypted as itself), and may mean that not all Baudot codes start at an offset in the message that is a multiple of five, which would make obtaining each Baudot code easier. Instead, punctuation has to be checked for before the system tries to get the next code, in order for it to be able to decrypt the message. This test shows a ciphertext with punctuation included to check that the system can correctly handle characters not part of a Baudot code and continue correctly to find any Baudot codes that come after it.

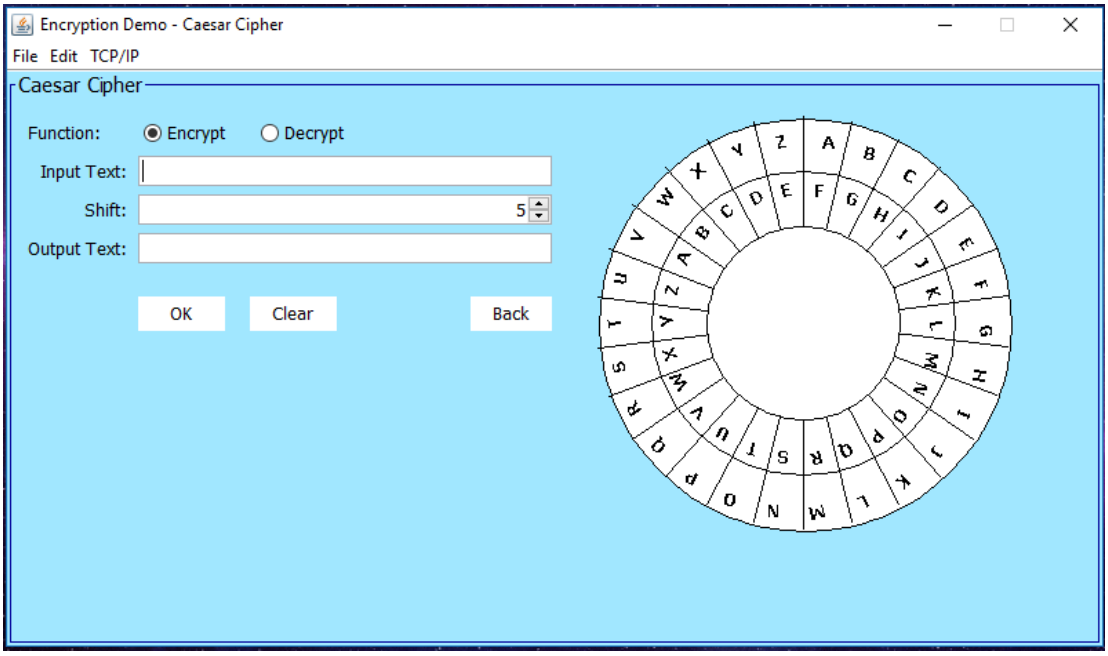


Test 10: In the Caesar cipher, only letters can be encrypted, but the system should be able to handle other characters, such as numbers and punctuation- these should be identified and encrypted as themselves, as shown by this screenshot that attempts to encrypt a string of punctuation and numbers using shift 7.

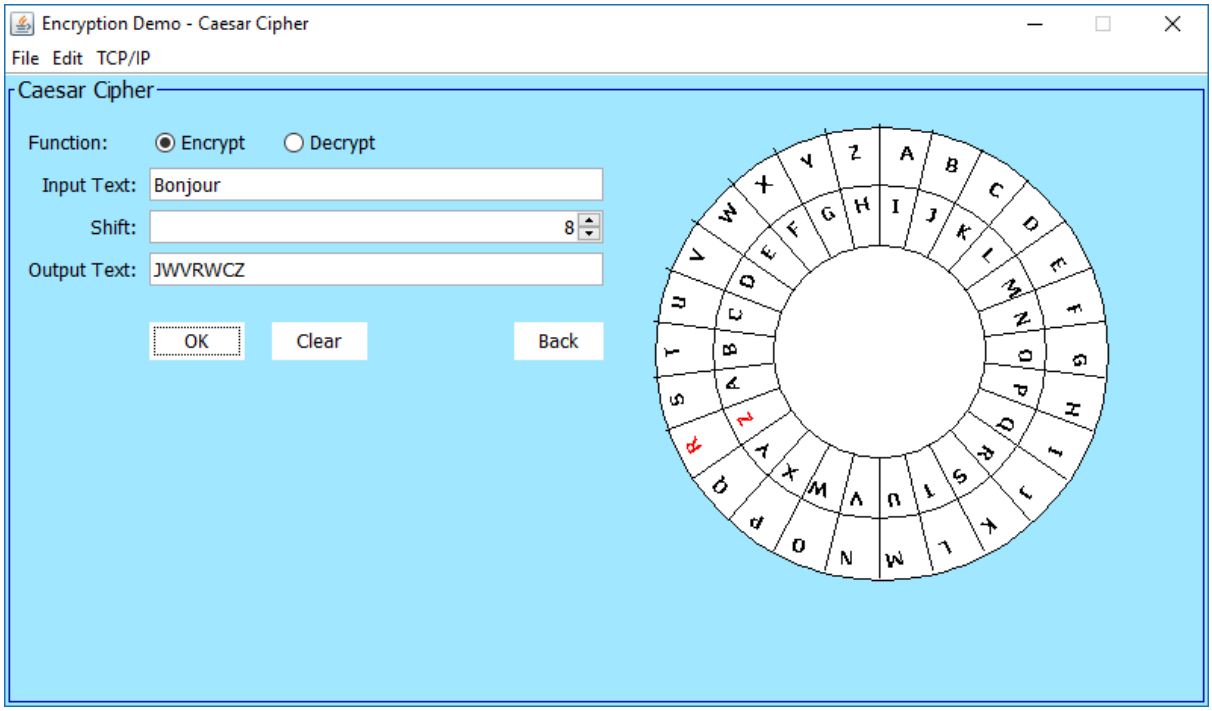


Test 13: For the animation, the Caesar wheel should show the selected setting, by moving the inner wheel round to line up the cipherletters with the corresponding plainletters, according to the selected setting. The first screenshot shows the wheel when no shift, or 0, is selected, and the second shows the wheel redrawn to display the letter pairings when a shift of 5 is selected.

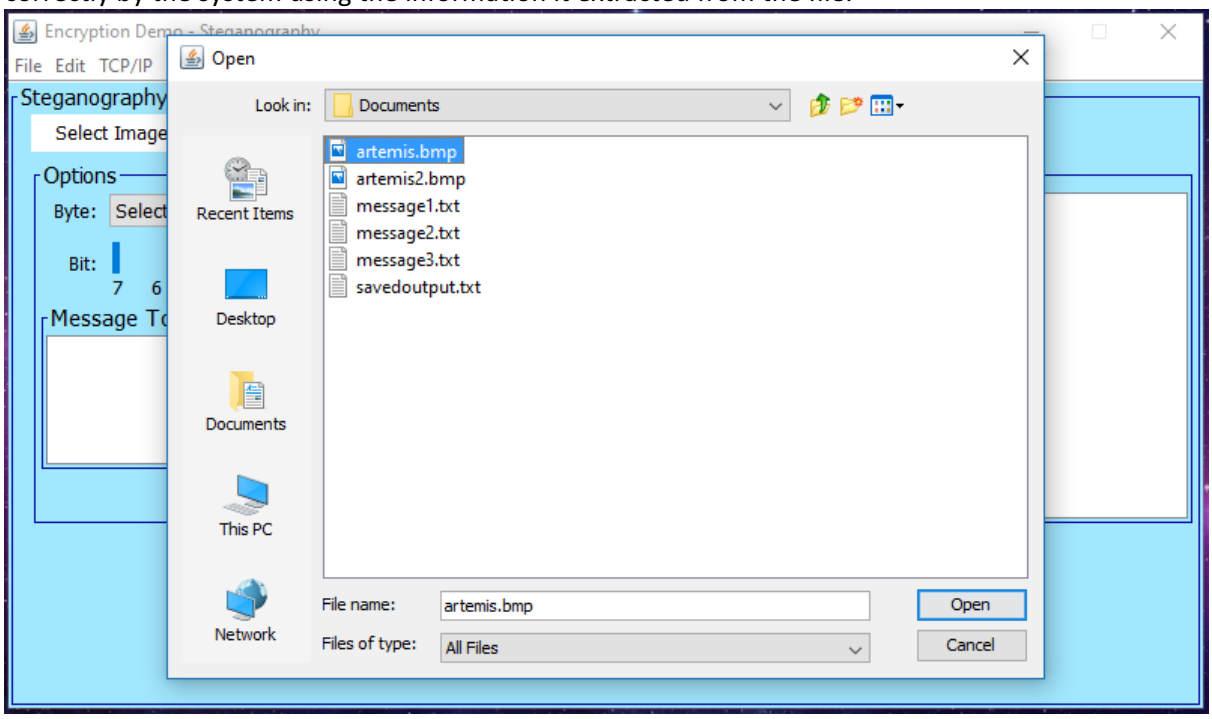


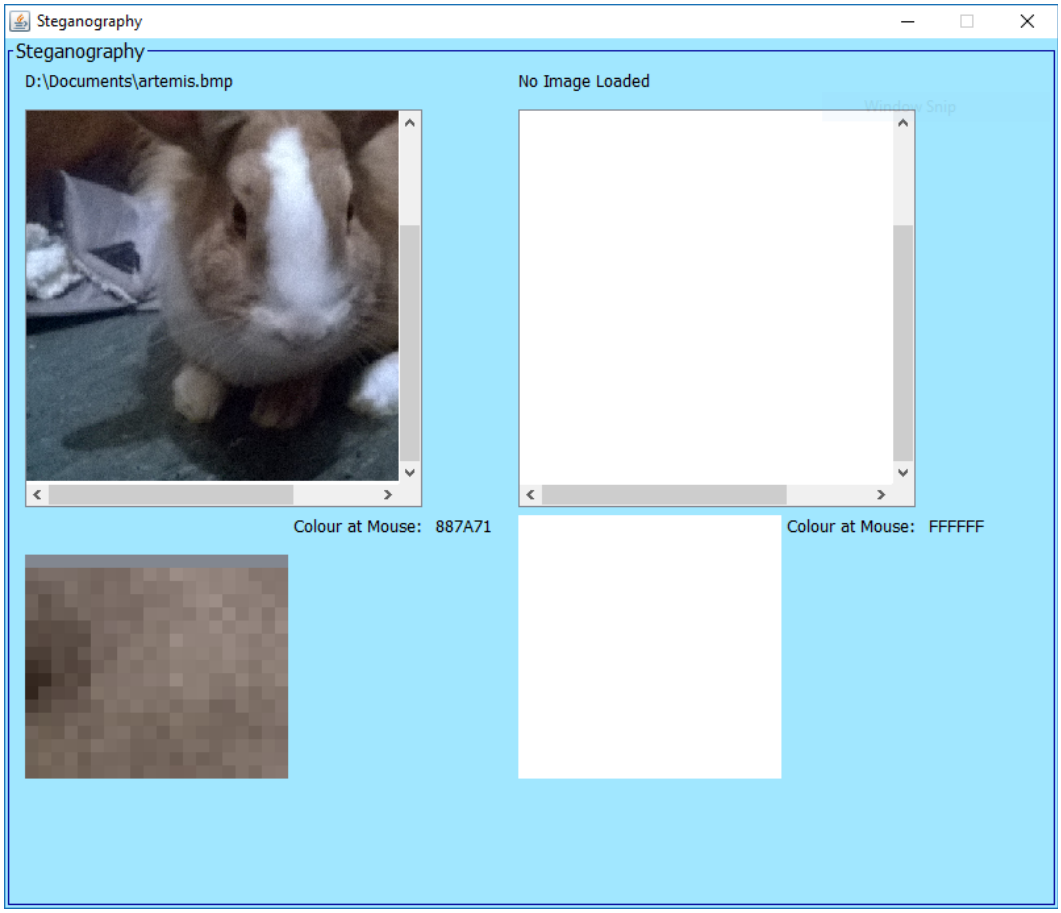


Test 14: This test checks that the system can encrypt a whole message correctly in the Caesar Cipher. The screenshot shows “Bonjour” correctly encrypted to “JWVRWCZ” using shift 8.

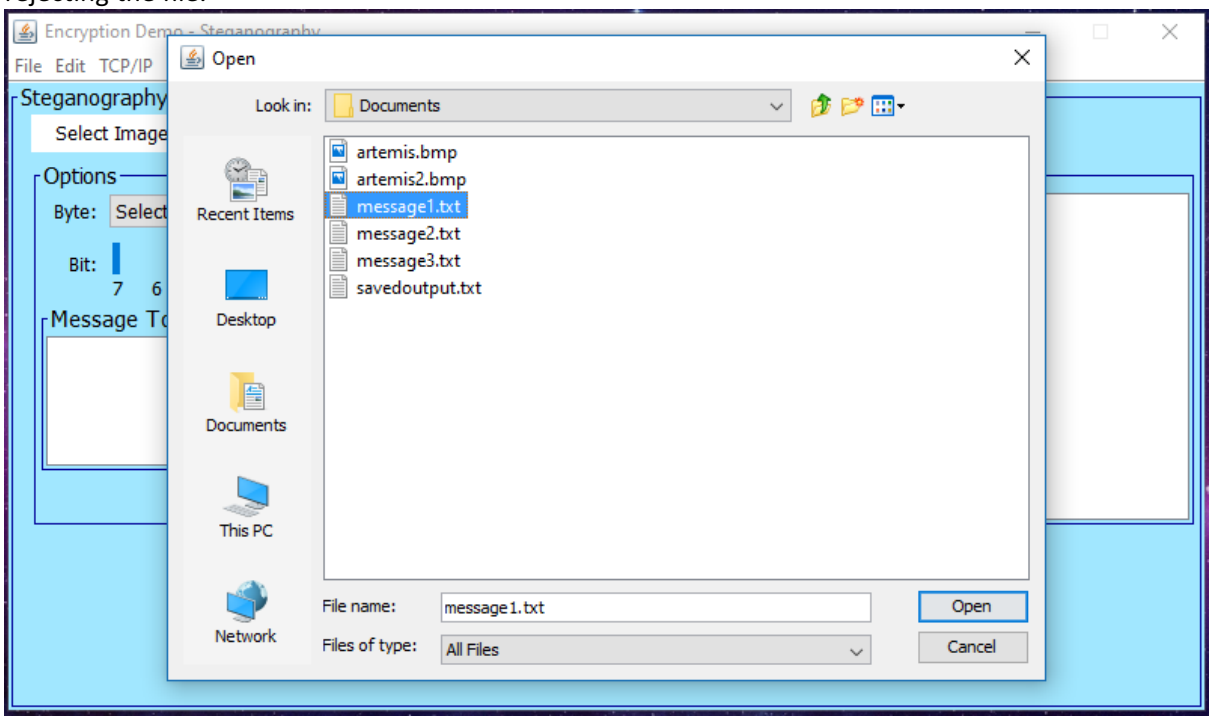


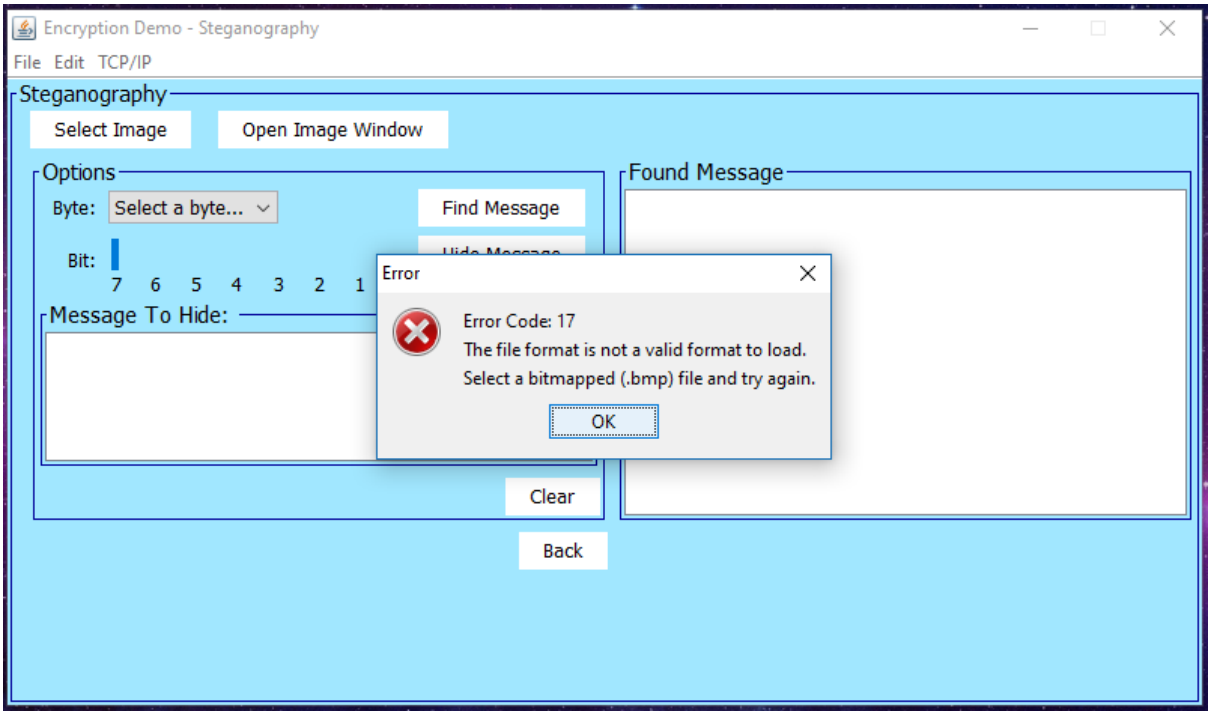
Test 16: For Steganography, the chosen image file must be of the valid structure so that the system can get the required information to load the image, hide a message in the right place in the file and find a message correctly in the file. The first screenshot shows artemis.bmp, a valid bitmap file, being chosen to be loaded and the second shows the image being loaded correctly by the system using the information it extracted from the file.



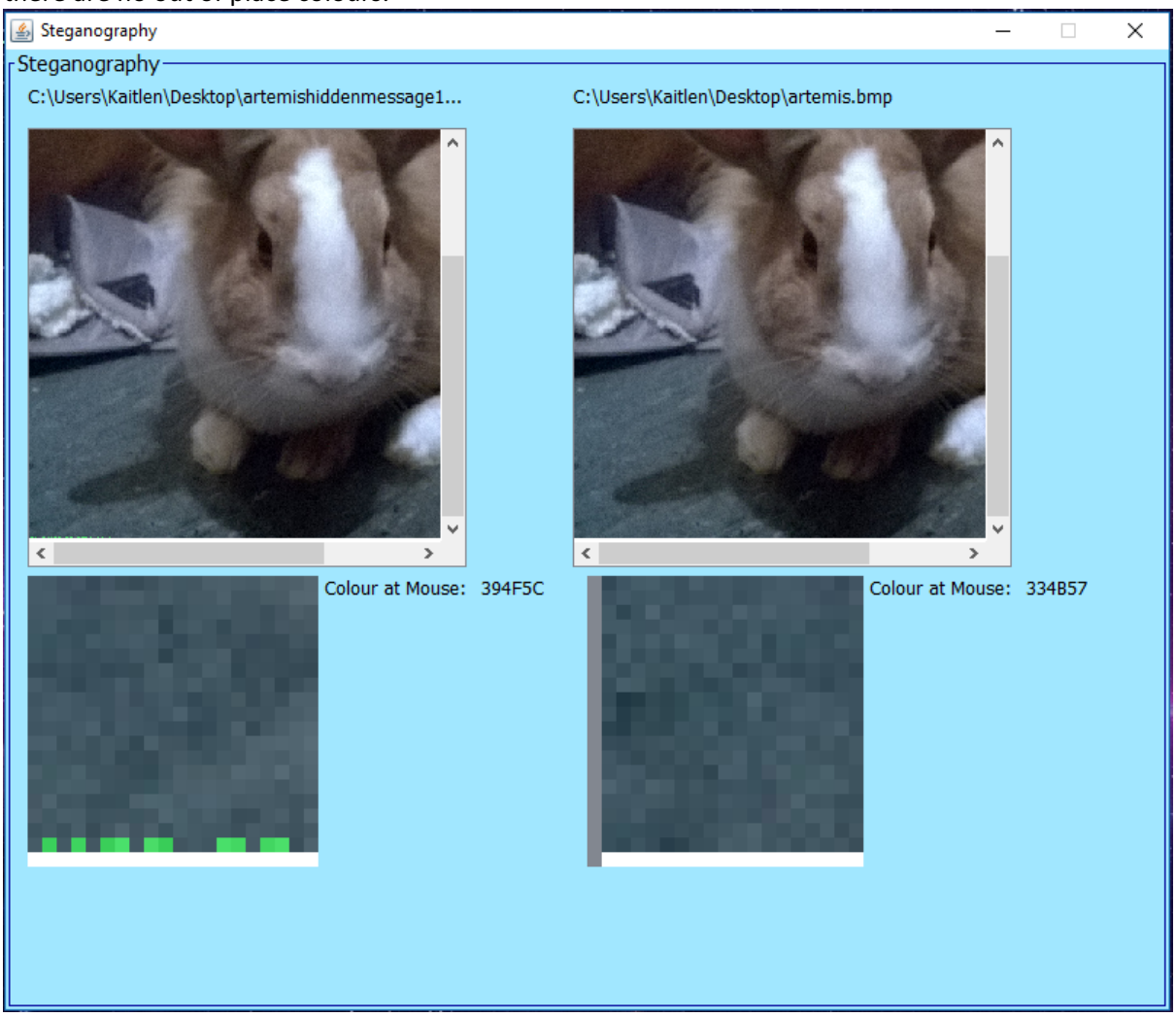


Test 17: This test checks that the system identifies and handles invalid files correctly. The first screenshot shows message1.txt (an invalid file because it is a text file and not a bitmapped image) being chosen to be loaded. The second screenshot shows the system rejecting the file.




Encryption Demo - Steganography
File Edit TCP/IP
Steganography
Select Image
Open Image Window
Options
Byte:
Select a byte...
Find Message
Bit:
7 6 5 4 3 2 1
Message To Hide:
Found Message
Error
Error Code: 17
The file format is not a valid format to load.
Select a bitmapped (.bmp) file and try again.
OK
Clear
Back

Test 21: Using the zoom function of the system, it should be possible in some cases to see where the message has been stored in the image, by extreme changes to the colours of the pixels.
For this test, the message was hidden in the most significant bit of each byte used to store the message. While this is not ideal for actually hiding the message, as it is obvious that the message is there, it makes it easier to test that the zoom feature is working correctly, as shown in the screenshot. The image on the left has a message hidden in it, and the changed bits are visible, as the green pixels, in the zoomed-in image of the bottom left corner beneath the image. On the right, is the original image - the zoomed-in section underneath it is same part of the image (i.e. the bottom left corner), but as there is no message hidden, there are no out of place colours.



Test 22: This test checks that the system hides the message in the selected bit of the selected byte. This was carried out by examining and comparing the binary values of an image with a message hidden in it and the original image (the same images as those in Test 21), as shown in the first screen shot. The bytes storing the colour values of the images' pixels starts at address 36 (in hexadecimal) and any bytes that differ between the two images are highlighted in red. The image containing the hidden message is the bottom file in the screenshot.

The second image shows, according to the binary values of the characters being hidden and the binary values of the bytes in the original image that will be used to hide the characters, which bytes in the file are expected to be changed in order to store each bit of the message. The bit being stored and the expected change to the byte are highlighted in green. The final column of the table can be completed by comparing these expected values to the bytes displayed for the changed image in the first screenshot – doing this confirmed that the only changes made to bytes in the file were the ones expected.

artemis.bmp

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dum
00000030	00000000	00000000	00000000	00000000	00000000	00000000	01011100	01010000	00111000	01010110	01001010	00110010	01100000	01010100	00111100	01011001	...
00000040	01001100	00110110	01001011	00111110	00101000	01001110	01000011	00101101	01011010	01001111	00111001	01011011	01010000	00111010	01101001	01011110	L6K
00000050	01001000	01100110	01011011	01000101	01101011	01100000	01001010	01100000	01010101	00111111	01100110	01011011	01000101	01011011	01010000	00111010	Hf[
00000060	01100100	01011001	01000011	01011111	01010100	00111110	01100010	01010111	01000011	01011001	01001110	00111010	01101011	01100000	01001010	01100010	dYC
00000070	01010111	01000011	01101000	01011100	01001010	01011000	01001101	00111001	01100000	01010101	01000001	01100101	01011010	01000100	01100001	01010110	WCh
00000080	01000000	01101010	01011111	01001001	01100010	01010111	01000011	01100101	01011010	01000110	01100001	01010110	01000010	01101010	01011111	01001011	@j[
00000090	01011001	01001110	00111010	01101101	01100010	01001110	01101011	01100000	01001100	01100000	01010101	01000001	01011101	01010010	00111110	01100100	YN::
000000a0	01011001	01000101	01011111	01010100	01000000	01100100	01011001	01000101	01011011	01010000	00111100	01101000	01011010	01000111	01100001	01010011	YE_
000000b0	01000000	01100010	01010100	01000010	01101000	01011010	01001000	01011110	01010010	01000000	01100100	01011000	01000110	01011101	01010001	00111111	@bT
000000c0	01011111	01010011	01000001	01101010	01011111	01001011	01011001	01001110	00111010	01011111	01010100	01000000	01011011	01010001	00111111	01011011	_SA
000000d0	01010001	00111111	01100001	01010111	01000101	01010110	01001100	00111010	01011010	01010011	01000000	01011110	01010100	01000010	01011000	01001110	Q?a
000000e0	00111100	01011101	01010000	01000000	01100000	01010011	01000011	01011100	01010000	00111110	01011100	01010000	00111110	01100111	01011011	01001001	<]P
000000f0	01011011	01001111	00111101	01100001	01010101	01000011	01100101	01011001	01000111	01101100	01100000	01001110	01101011	01100001	01001111	01100100	[O=

artemishiddenmessage1.bmp

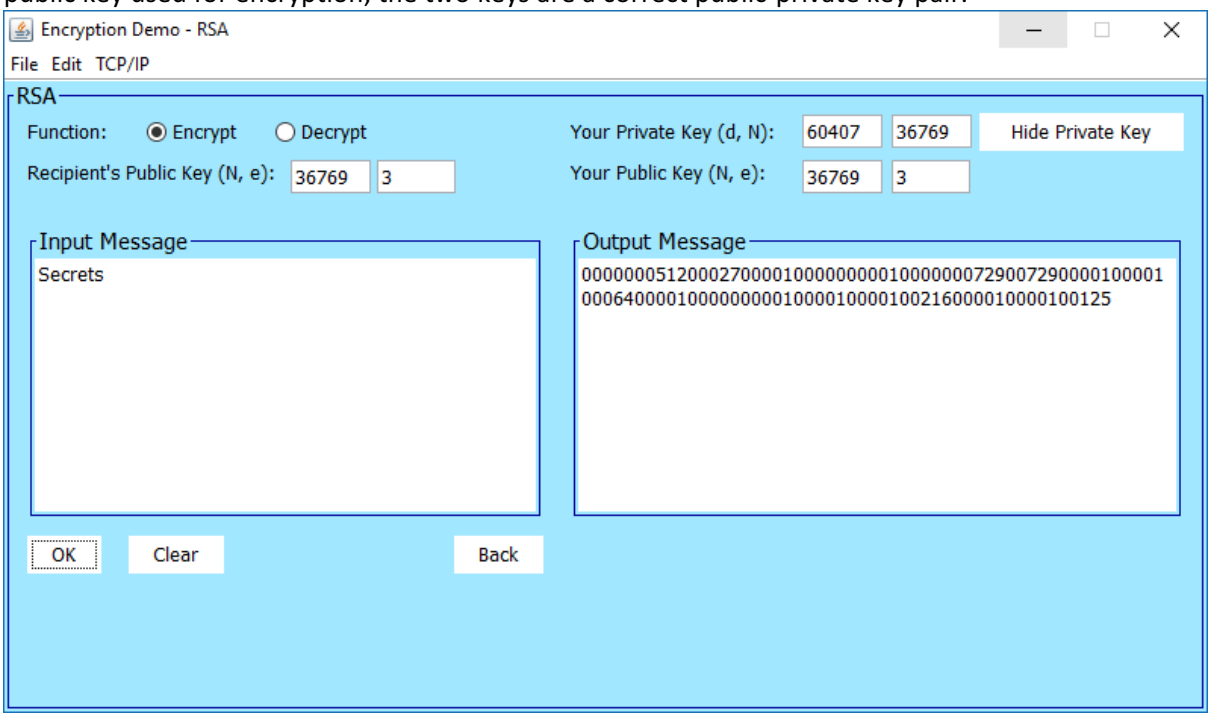
Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dum
00000030	00000000	00000000	00000000	00000000	00000000	00000000	01011100	01010000	00111000	01010110	11001010	00110010	01100000	11010100	00111100	01011001	...
00000040	01001100	00110110	01001011	10111110	00101000	01001110	01000011	00101101	01011010	01001111	00111001	01011011	01010000	00111010	01101001	01011110	L6K
00000050	01001000	01100110	11011011	01000101	01101011	11100000	01001010	01100000	01010101	00111111	01100110	01011011	01000101	01011011	11010000	00111010	Hf[
00000060	01100100	01011001	01000011	01011111	11010100	00111110	01100010	01010111	01000011	01011001	11001110	00111010	01101011	11100000	01001010	01100010	dYC
00000070	01010111	01000011	01101000	11011100	01001010	01011000	11001101	00111001	01100000	01010101	01000001	01100101	01011010	01000100	01100001	01010110	WCh
00000080	01000000	01101010	11011111	01001001	01100010	11010111	01000011	01100101	01011010	01000110	01100001	11010110	01000010	01101010	11011111	01001011	@j[
00000090	01011001	01001110	00111010	01101101	01100010	01001110	01101011	01100000	01001100	01100000	11010101	01000001	01011101	11010010	00111110	01100100	YN::
000000a0	01011001	01000101	01011111	11010100	01000000	01100100	11011001	01000101	01011011	11010000	00111100	01101000	11011010	01000111	01100001	01010011	YE_
000000b0	01000000	01100010	01010100	01000010	01101000	11011010	01001000	01011110	01010010	01000000	01100100	01011000	01000110	01011101	01010001	00111111	@bT
000000c0	01011111	01010011	01000001	01101010	11011111	01001011	01011001	01001110	00111010	01011111	01010100	01000000	01011011	11010001	00111111	01011011	_SA
000000d0	01010001	00111111	01100001	01010111	01000101	01010110	01001100	00111010	01011010	01010011	01000000	01011110	11010100	01000010	01011000	01001110	Q?a
000000e0	00111100	01011101	01010000	01000000	01100000	01010011	01000011	01011100	01010000	00111110	01011100	01010000	00111110	01100111	01011011	01001001	<]P
000000f0	01011011	01001111	00111101	01100001	01010101	01000011	01100101	01011001	01000111	01101100	01100000	01001110	01101011	01100001	01001111	01100100	[O=

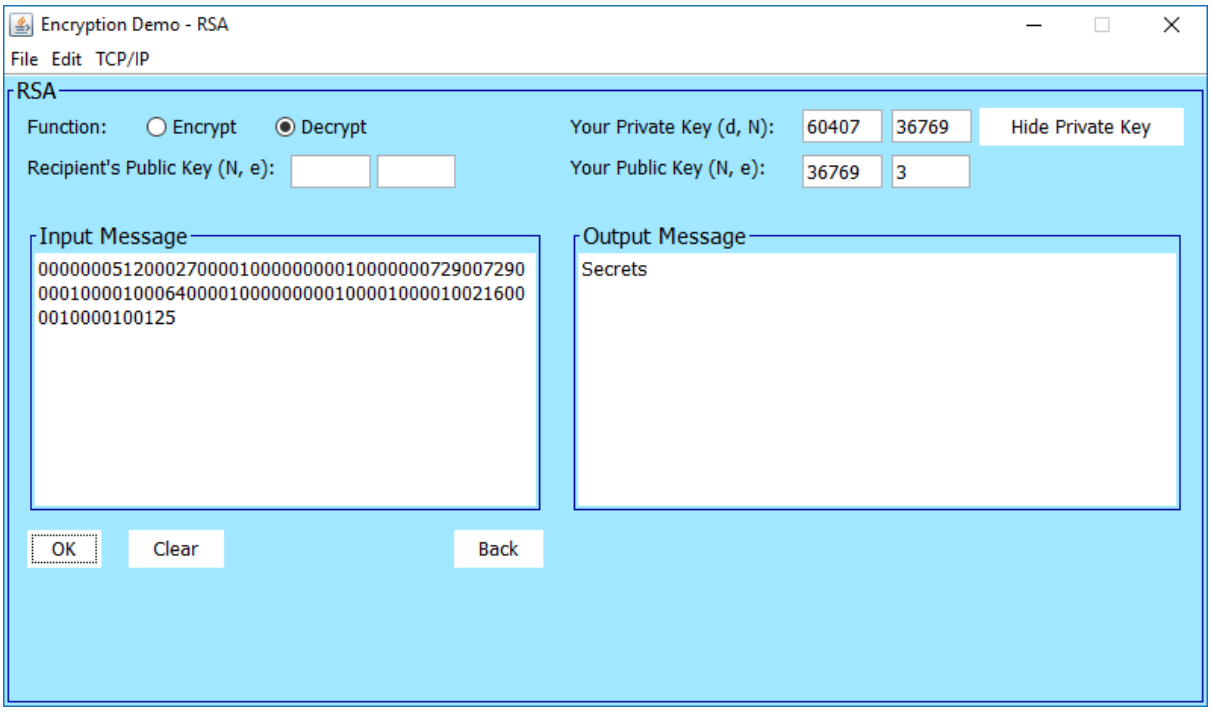
ASCII Character	Hex Value of Each Character (ASCII)	Message bits	Byte Address in File (in hex)	Value in Original Image	Expected Value in Changed Image	Value Changed?
h	68 (01101000)	0	37	01010000	01010000	No
		1	3a	01001010	11001010	Yes
		1	3d	01010100	11010100	Yes
		0	40	01001100	01001100	No
		1	43	00111110	10111110	Yes
		0	46	01000011	01000011	No
		0	49	01001111	01001111	No
		0	4c	01010000	01010000	No
e	65 (01100101)	0	4f	01011110	01011110	No
		1	52	01011011	11011011	Yes
		1	55	01100000	11100000	Yes
		0	58	01010101	01010101	No
		0	5b	01011011	01011011	No
		1	5e	01010000	11010000	Yes
		0	61	01011001	01011001	No
		1	64	01010100	11010100	Yes
l	6c (01101100)	0	67	01010111	01010111	No
		1	6a	01001110	11001110	Yes
		1	6d	01100000	11100000	Yes
		0	70	01010111	01010111	No
		1	73	01011100	11011100	Yes
		1	76	01001101	11001101	Yes
		0	79	01010101	01010101	No
		0	7c	01011010	01011010	No
l	6c (01101100)	0	7f	01010110	01010110	No
		1	82	01011111	11011111	Yes
		1	85	01010111	11010111	Yes
		0	88	01011010	01011010	No
		1	8b	01010110	11010110	Yes
		1	8e	01011111	11011111	Yes
		0	91	01001110	01001110	No
		0	94	01100010	01100010	No
o	6f (01101111)	0	97	01100000	01100000	No
		1	9a	01010101	11010101	Yes
		1	9d	01010010	11010010	Yes
		0	a0	01011001	01011001	No
		1	a3	01010100	11010100	Yes
		1	a6	01011001	11011001	Yes
		1	a9	01010000	11010000	Yes
		1	ac	01011010	11011010	Yes
!	21 (00100001)	0	af	01010011	01010011	No
		0	b2	01010100	01010100	No
		1	b5	01011010	11011010	Yes
		0	b8	01010010	01010010	No
		0	bb	01011000	01011000	No
		0	be	01010001	01010001	No
		0	c1	01010011	01010011	No
		1	c4	01011111	11011111	Yes
!	21 (00100001)	0	c7	01001110	01001110	No
		0	ca	01010100	01010100	No
		1	cd	01010001	11010001	Yes
		0	d0	01010001	01010001	No
		0	d3	01010111	01010111	No
		0	d6	01001100	01001100	No
		0	d9	01010011	01010011	No
		1	dc	01010100	11010100	Yes
null (terminator byte)	00 (00000000)	0	df	01001110	01001110	No
		0	e2	01010000	01010000	No
		0	e5	01010011	01010011	No
		0	e8	01010000	01010000	No
		0	eb	01010000	01010000	No
		0	ee	01011011	01011011	No
		0	f1	01001111	01001111	No
		0	f4	01010101	01010101	No

Test 24: As part of the key generation for RSA, several intermediate values need to be calculated to produce the keys' actual values. Two of these are p and q, two prime numbers chosen at random from the list of prime numbers generated by the system. Subtracting one from each of these and multiplying the results together produces a third number, phi.
Phi is used with another value, K, where K mod phi = 1 and K must not be prime as a pair of K's factors are used to produce half of the public key and half of the private key.
Possible values of K can be found by finding multiples of phi, adding one to them and checking that the result is not prime.
The screenshot shows the values produced when p and q are specified as 73 and 7 respectively. Here, two values of K are produced, 865 and 1729. The screenshot shows how these are calculated by finding a multiple of phi (in these cases by multiplying by 2 and 4) and adding one to this. It then displays the factors of K to prove that it does have some, meaning it is not prime.

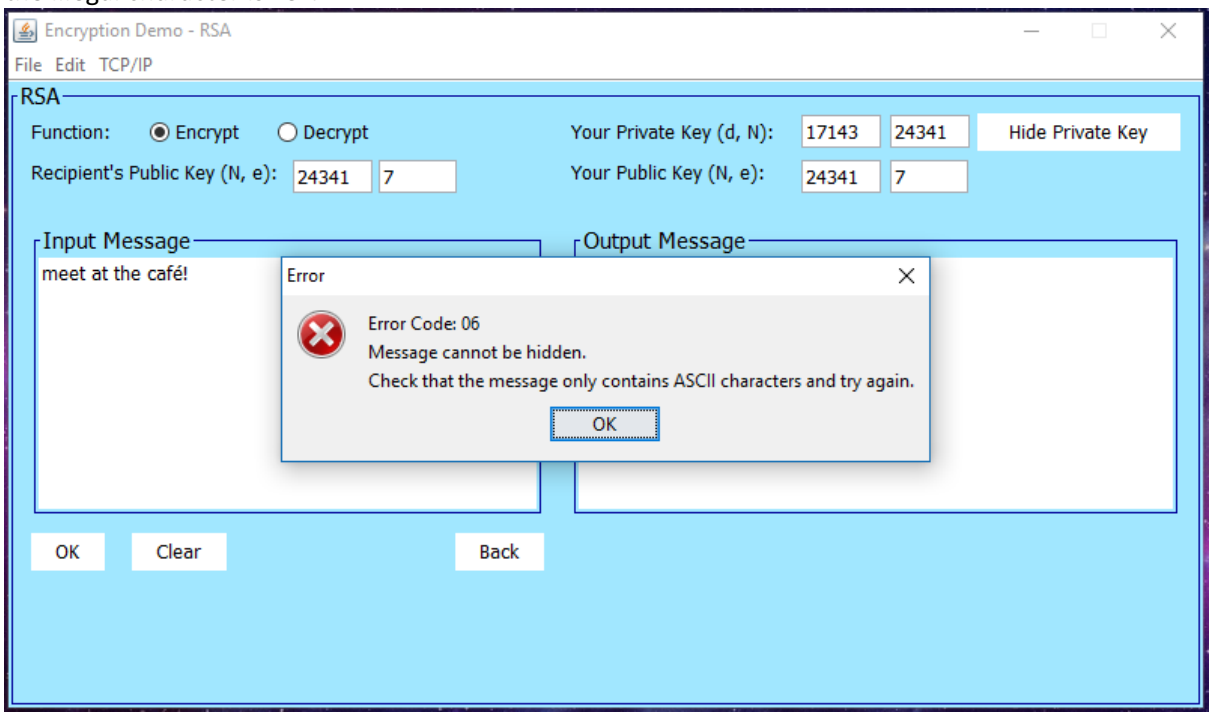
```
p: 73
q: 7
phi = 432
phi * 2 = 864
phi * 2 + 1 = 865
K: 865
Factors:
[5, 173]
phi * 4 = 1728
phi * 4 + 1 = 1729
K: 1729
Factors:
[7, 247]
[13, 133]
[19, 91]
```

Test 25: This test checks the public and private keys are a correct pair, by checking that a message encrypted with the public key can actually be decrypted with the private key that is given to go with it.
The first screenshot shows the original message being encrypted using the given public key. The second screenshot shows the ciphertext from the first screenshot being decrypted. The system automatically uses the private key it has generated- the private key displayed in the top right of the screen.
Since the original message was successfully decrypted using the matching private key for the public key used for encryption, the two keys are a correct public-private key pair.

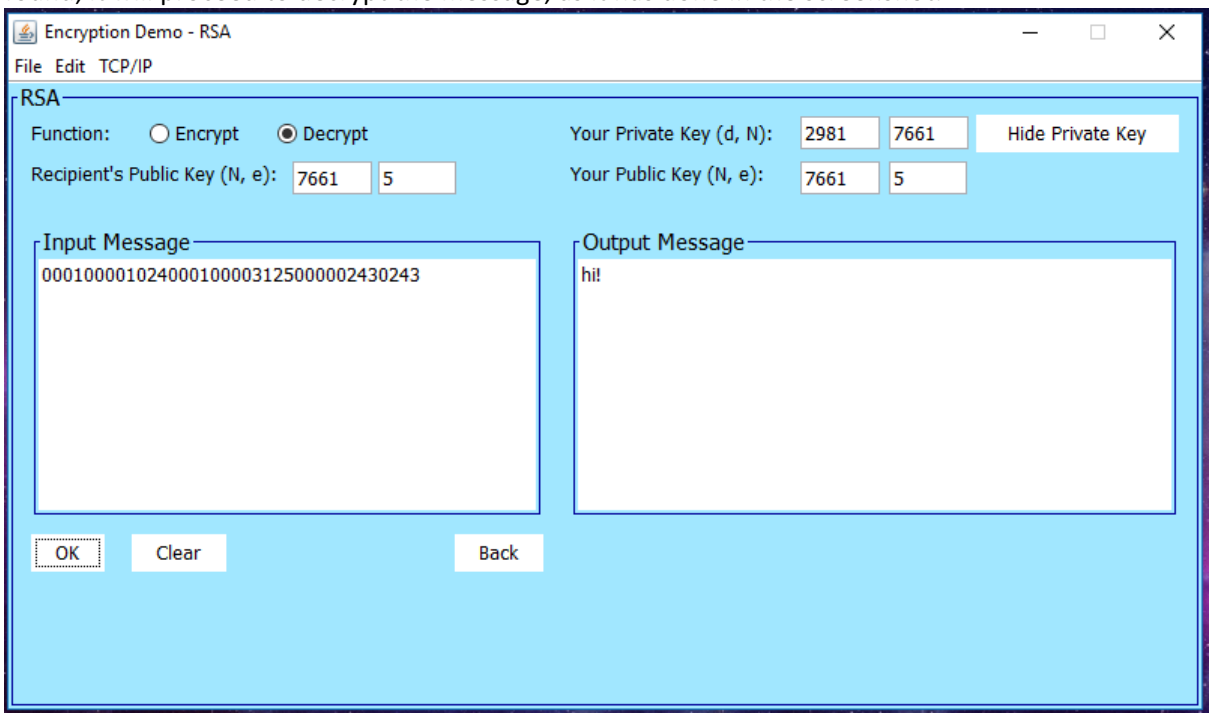




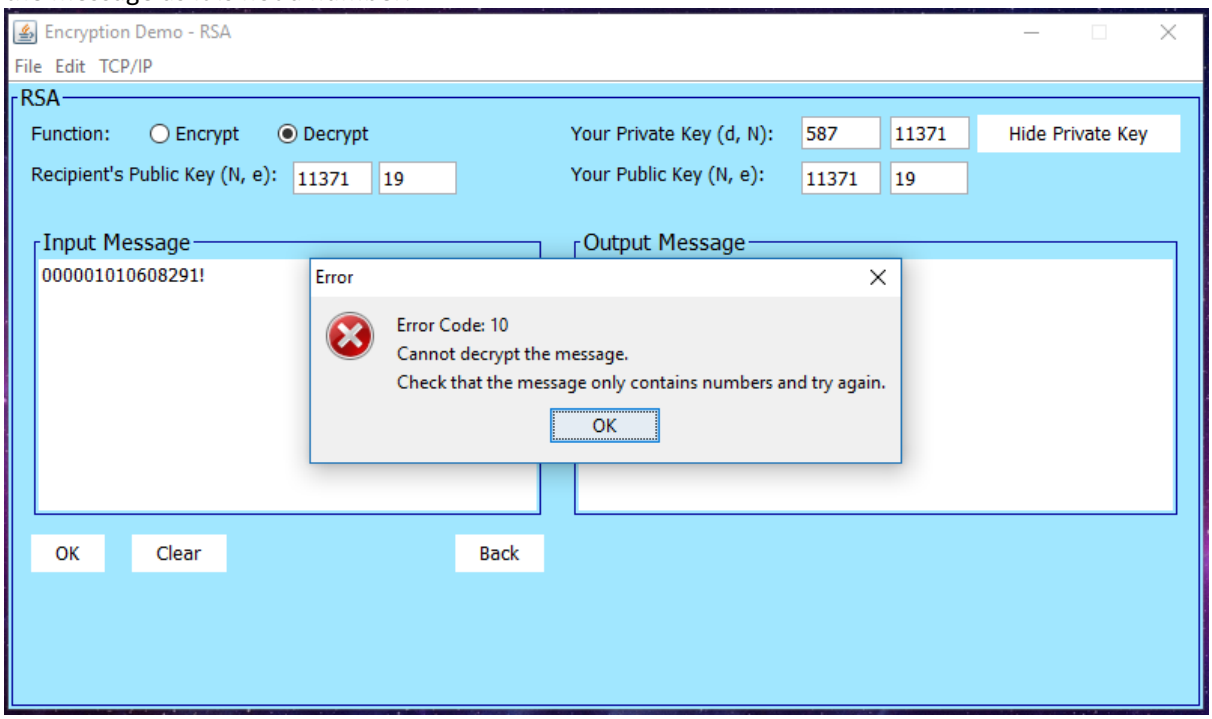
Test 27: The message to encrypt using RSA cannot have any non-ASCII characters in it; this test checks that non-ASCII characters are caught correctly and rejected. In this screenshot, the illegal character is “é”.



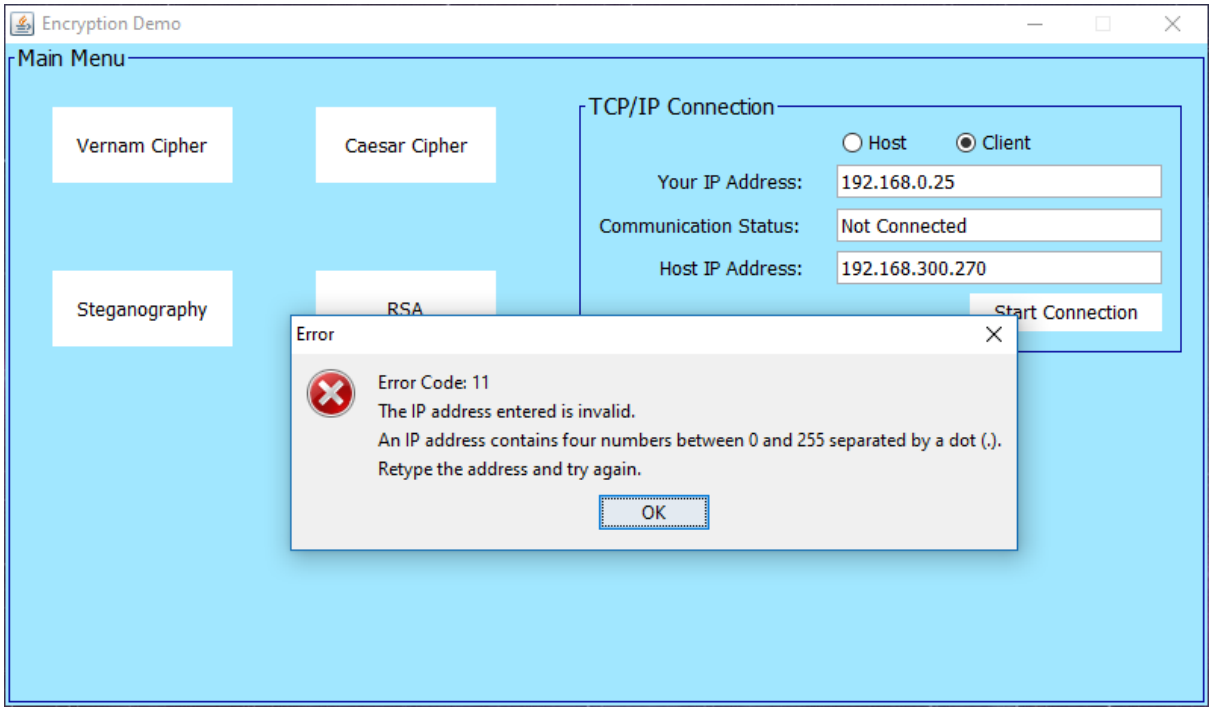
Test 28: For RSA decryption, there must only be numbers in the ciphertext, as all characters that are accepted in a plaintext message are encrypted to a number. This test checks that the system accepts a standard, correct ciphertext for decryption. The system checks for problems with the input message, and reports any it finds to the user, so if no problems are found, it will proceed to decrypt the message, as it has done in the screenshot.



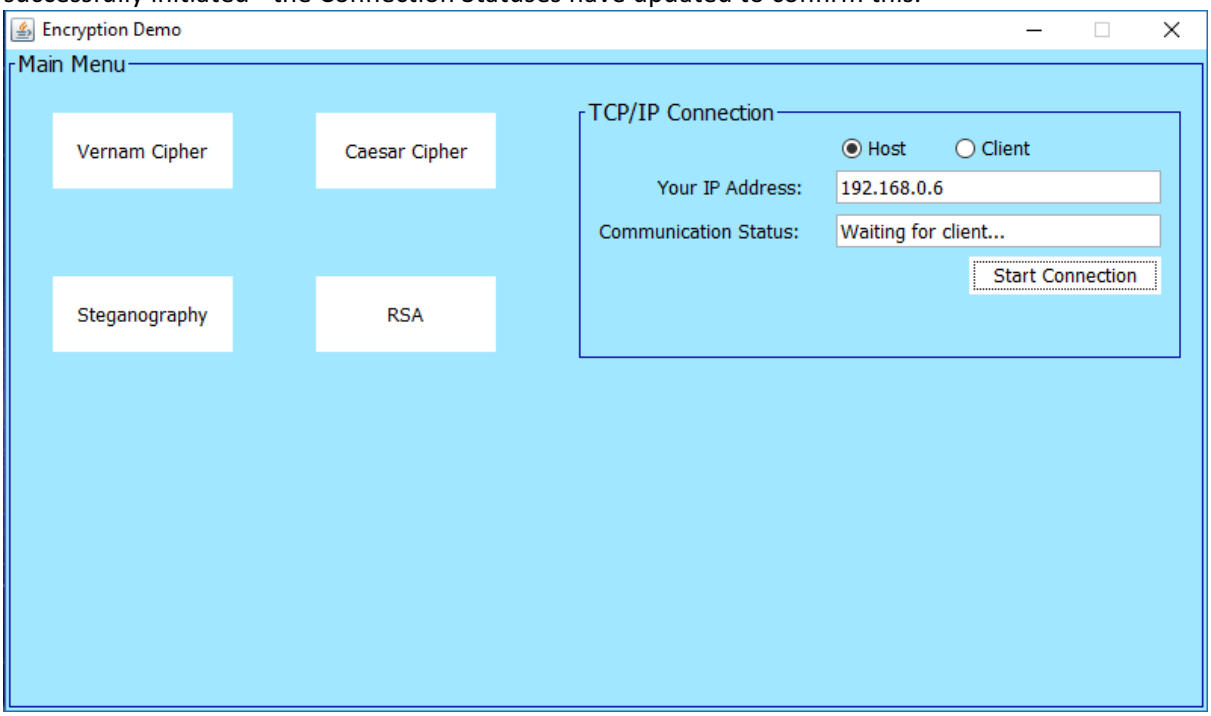
Test 29: This test checks that the system can correctly catch invalid input for decryption and report it. In this screenshot, the exclamation mark at the end of the ciphertext invalidates the message as it is not a number.

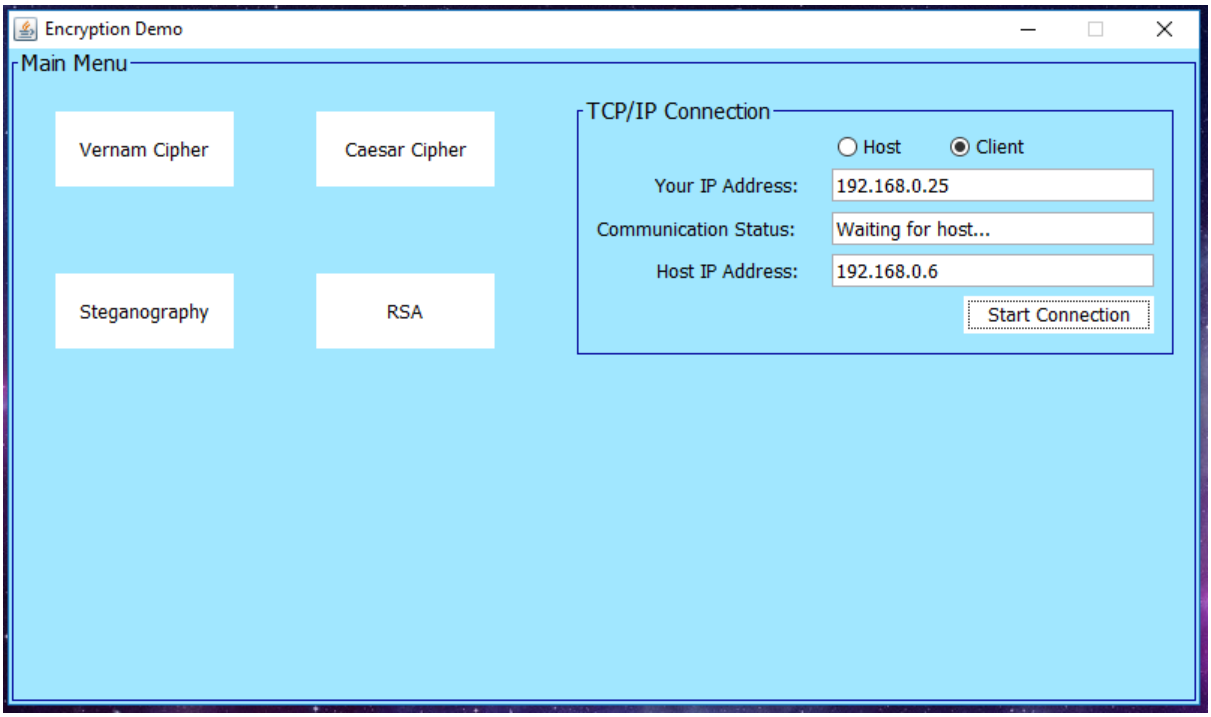


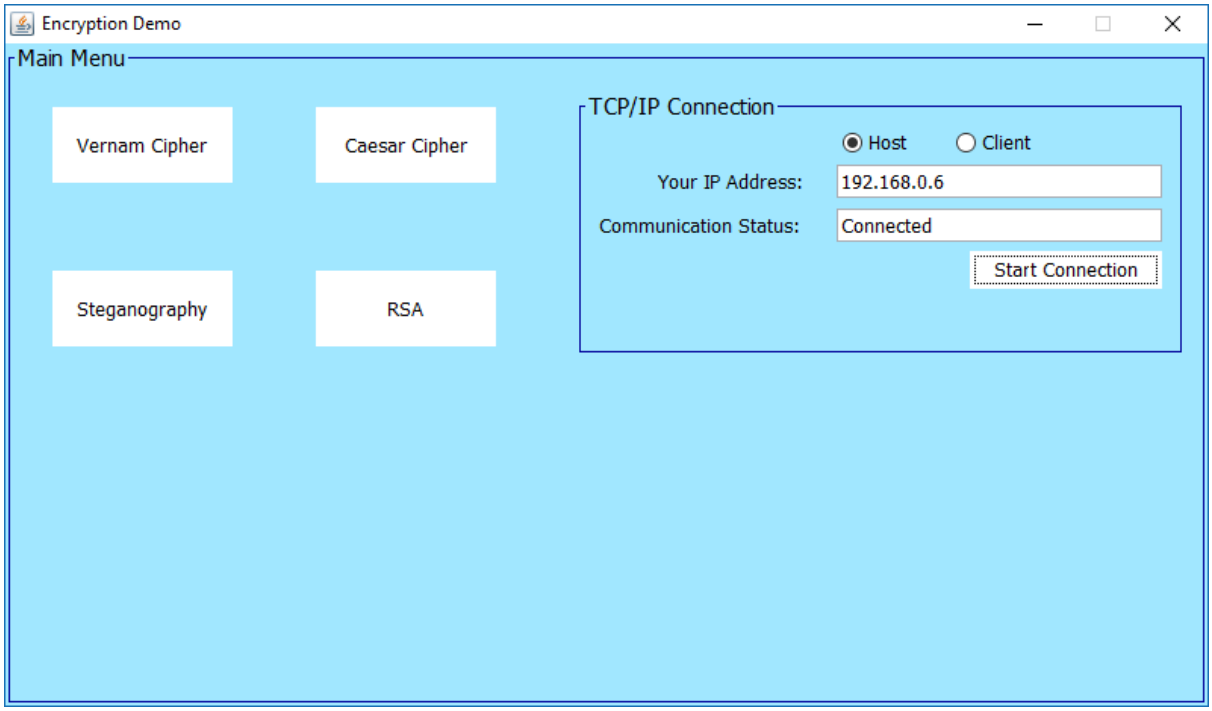
Test 31: This test checks that system correctly catches invalid IP addresses. An IP address is made of four numbers, each less than 255, so the IP address used in this screenshot is not valid as 300 and 270 are more than 255.

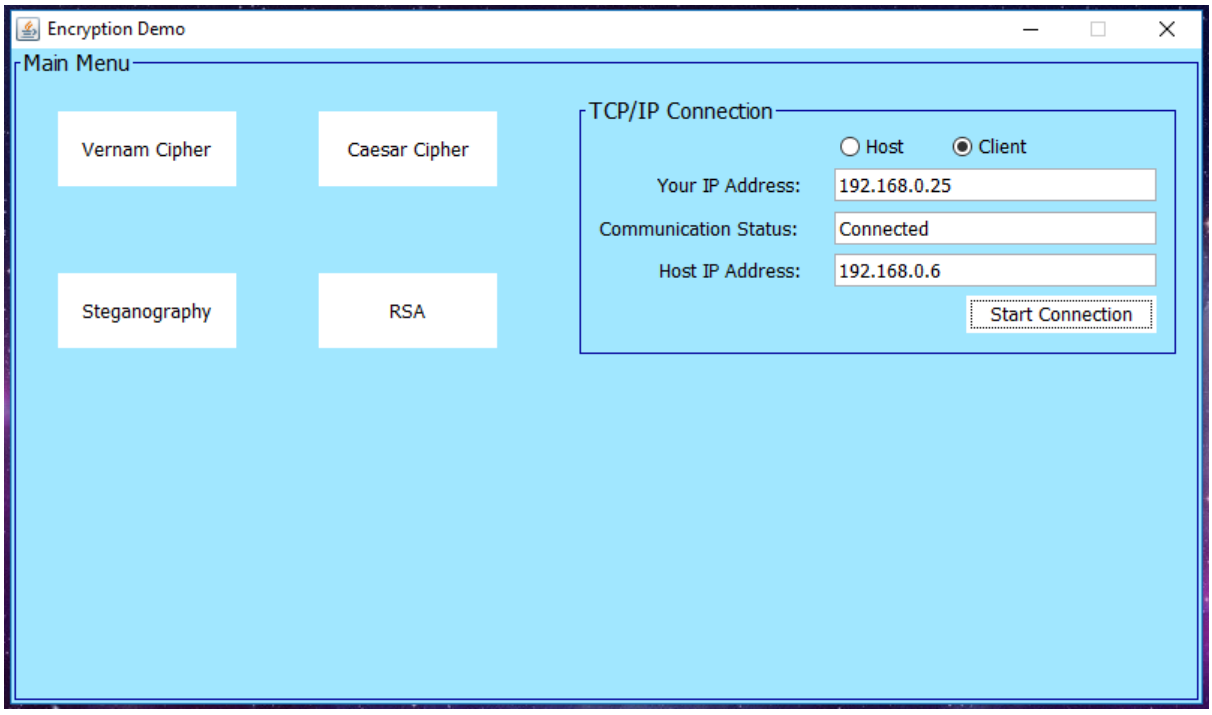


Test 33: This test checks that a TCP/IP connection can be successfully made when the required information is provided correctly.
The first two screenshots show the screens on either of the connection (the first being the Host's screen and the second being the Client's screen) when they have requested a connection be made but not established one yet, as the Connection Statuses indicate.
The final two screenshots show the screens on either end of the connection (the first being the Host's screen and the second being the Client's screen) when the connection has been successfully initiated - the Connection Statuses have updated to confirm this.

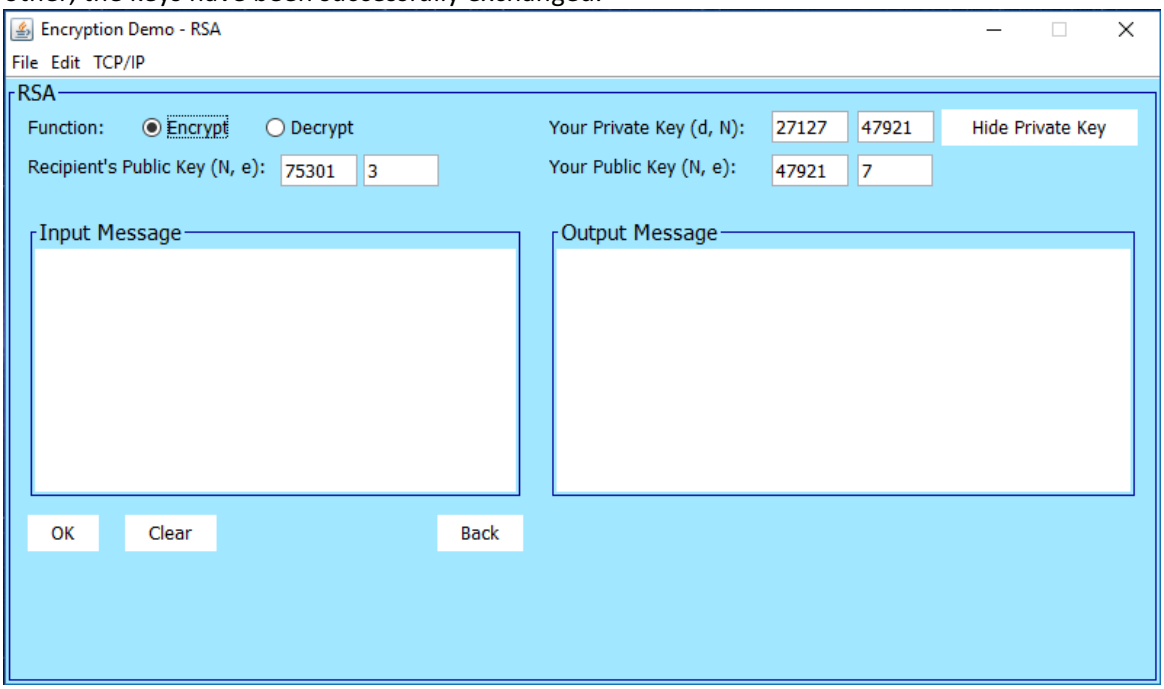


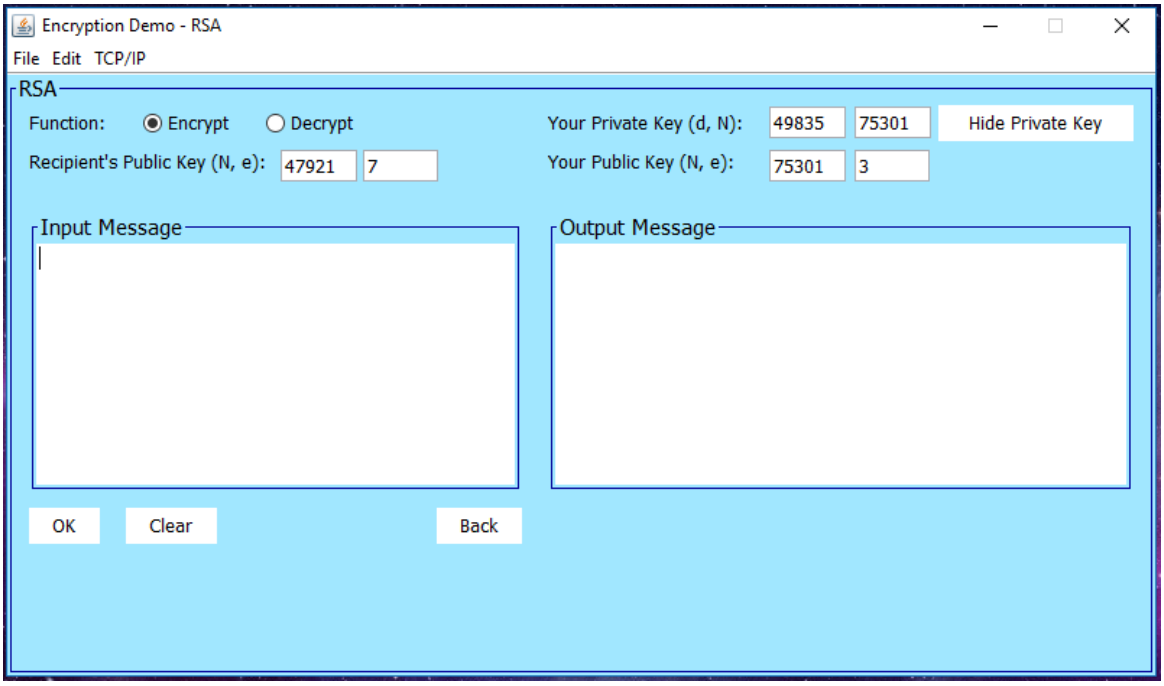



Encryption Demo
Main Menu
Vernam Cipher
Caesar Cipher
Steganography
RSA
TCP/IP Connection
Host
Client
Your IP Address:
192.168.0.6
Communication Status:
Connected
Start Connection

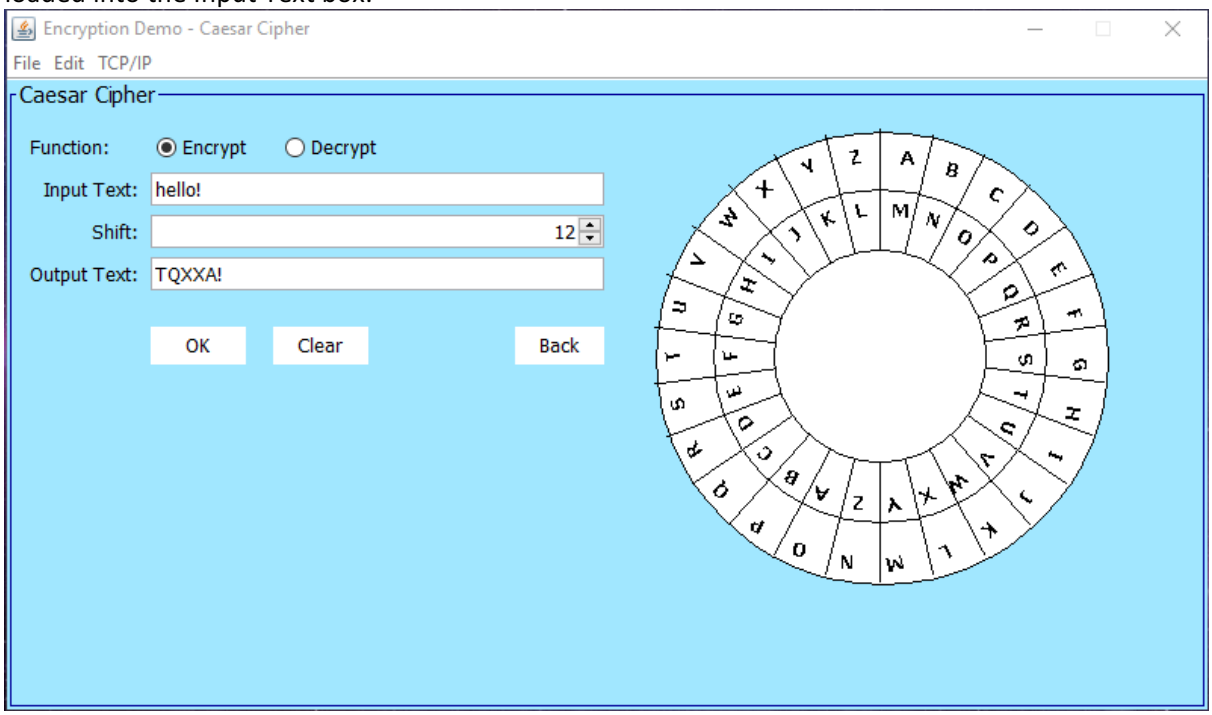

Encryption Demo
Main Menu
Vernam Cipher
Caesar Cipher
Steganography
RSA
TCP/IP Connection
Host
Client
Your IP Address:
192.168.0.25
Communication Status:
Connected
Host IP Address:
192.168.0.6
Start Connection

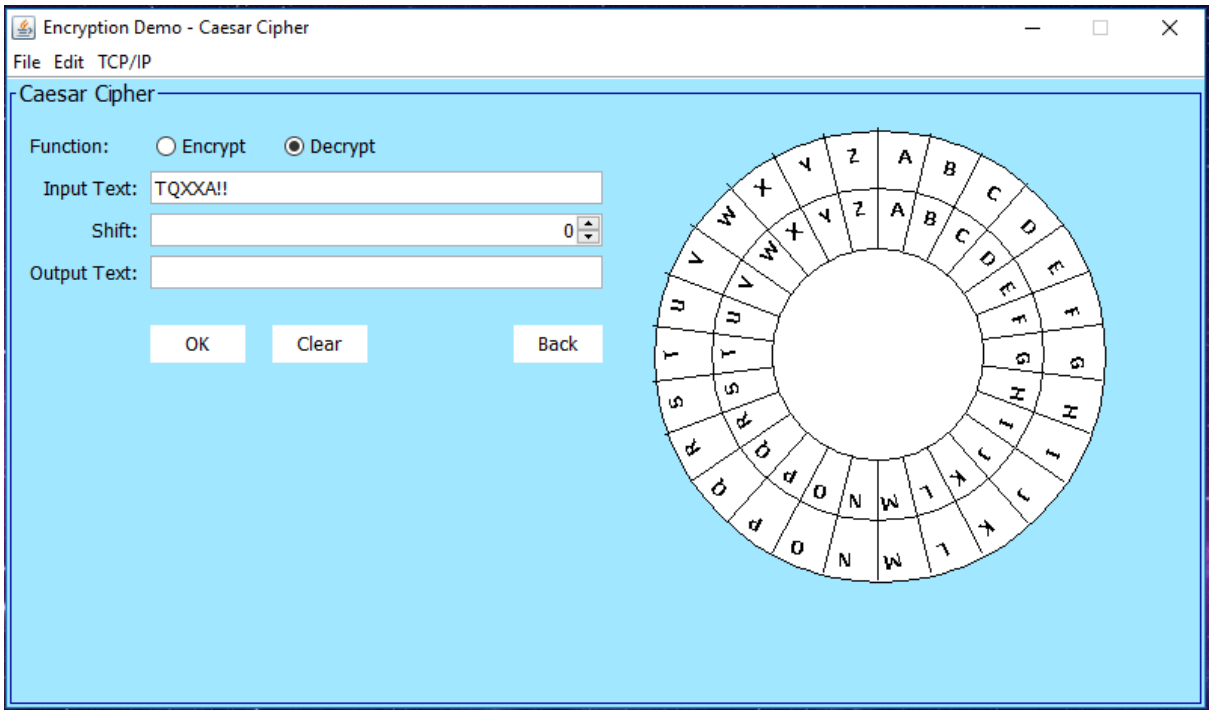
Test 34: When a TCP/IP connection is first initiated, the two ends automatically exchange RSA public keys, which are automatically displayed in the input boxes for the public key on the RSA screen. This test checks that the public keys are exchanged correctly and then displayed correctly on the screen.
The first screenshot shows the RSA screen of the Host. The public key on the left is the Client's public key and the public key on the right is the Host's own public key. Likewise, in the second screenshot, which shows the Client's screen, the left public key is the Host's key, as displayed in the first screenshot on the right. The public key on the right of the second screenshot is the Client's, which is also displayed on the left of the Host's screen.
Since the public key on the left of one screen matches the public key on the right of the other, the keys have been successfully exchanged.

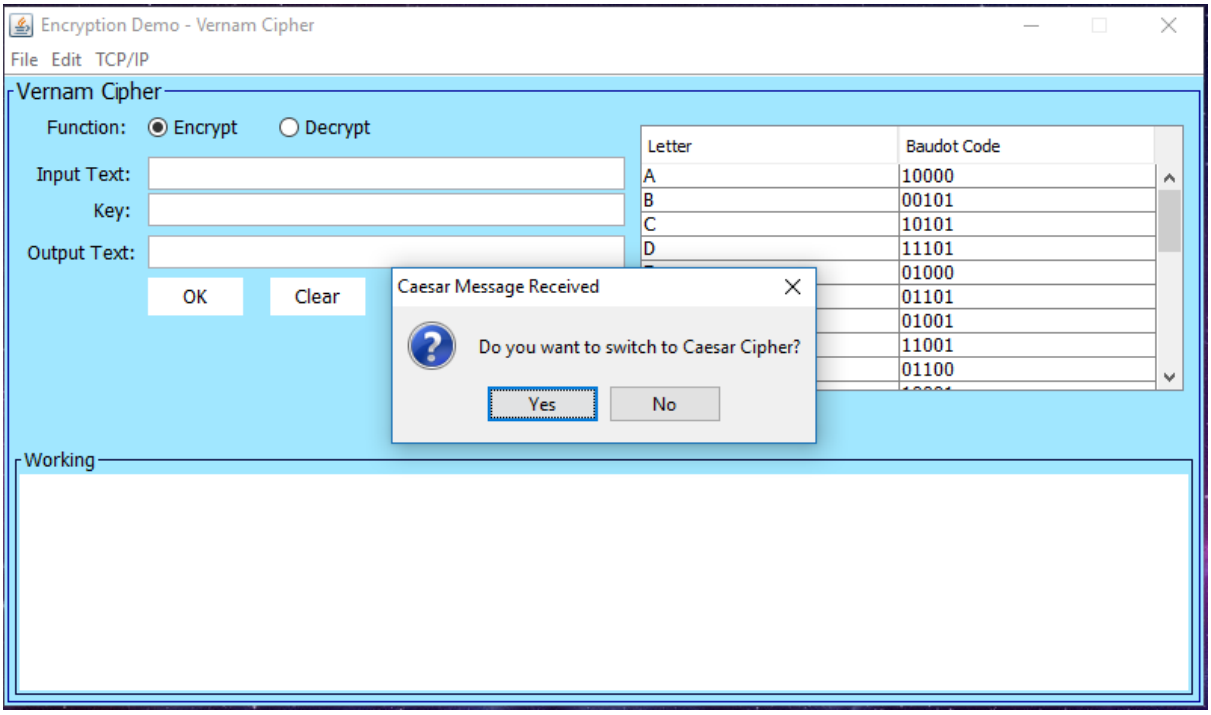




Test 35: This test checks that a message can be sent from one user to another and that the correct method used to encrypt the message is identified on the receiving end to load it in the right screen.
The first screenshot shows the message to send in the Output Text box of the sender's Caesar Cipher screen.
In the second screenshot, the message has been received by the other user, but since they have received a Caesar Cipher encrypted message while they are on the Vernam Cipher screen, they are asked if they want to switch to the Caesar Cipher.
In the third screenshot, the user chose to switch screens and the received message has been loaded into the Input Text box.






Encryption Demo - Vernam Cipher
File Edit TCP/IP
Vernam Cipher
Function: Encrypt Decrypt
Input Text:
Key:
Output Text:
OK
Clear
Letter
Baudot Code
A 10000
B 00101
C 10101
D 11101
01000
01101
01001
11001
01100
Caesar Message Received
Do you want to switch to Caesar Cipher?
Yes
No
Working

Technical Solution – Overview Guide

1. System Overview

The system is programmed in Java and makes use of the javax.swing library for creating the GUI.

It allows two users to establish a connection over TCP/IP and send messages and files to each other; it uses the java.net library for doing this.

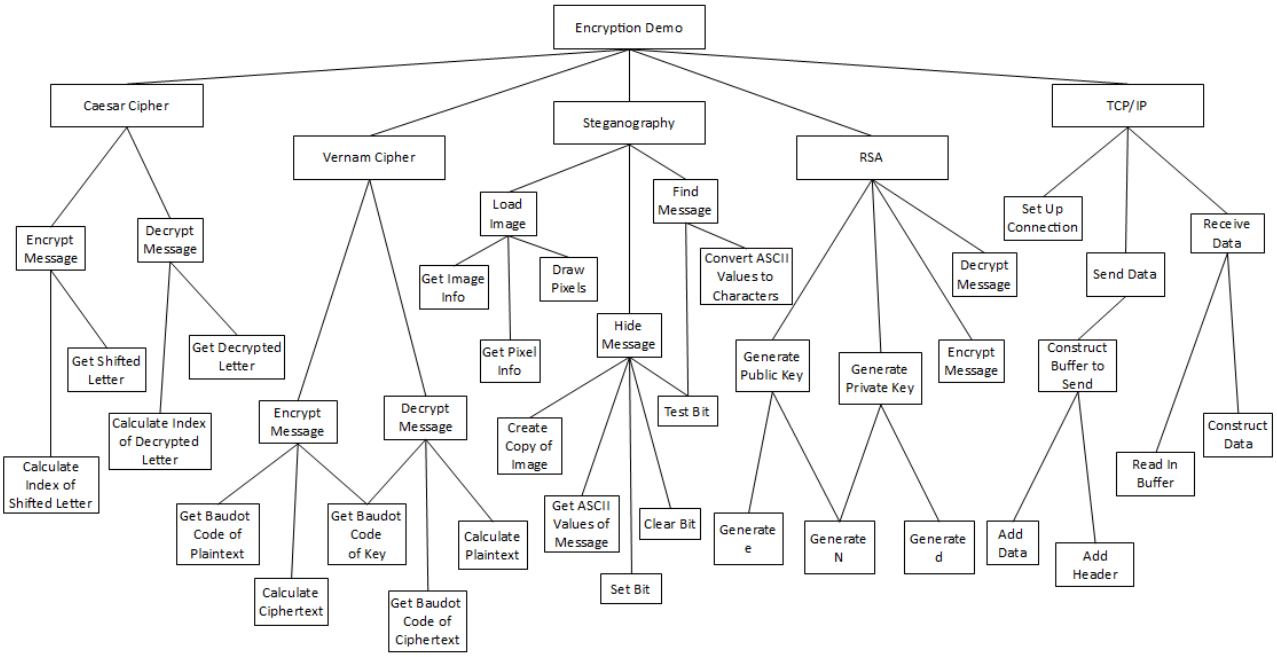
The system demonstrates the Caesar and Vernam Cipher with animation to show the working of the process. It also demonstrates the concept of Asymmetric Encryption, using RSA, and automatically generates the public and private keys for the user to use in encryption and decryption.

The system also allows the user to hide a message in a bitmapped file, by changing the values of the files, and can also find messages hidden in bitmapped images. The system can load bitmapped images, which it does by loading the data required for displaying the image into arrays for quicker access.

In addition to being able to load and write to bitmapped files for Steganography, it can also load and save messages to text files.

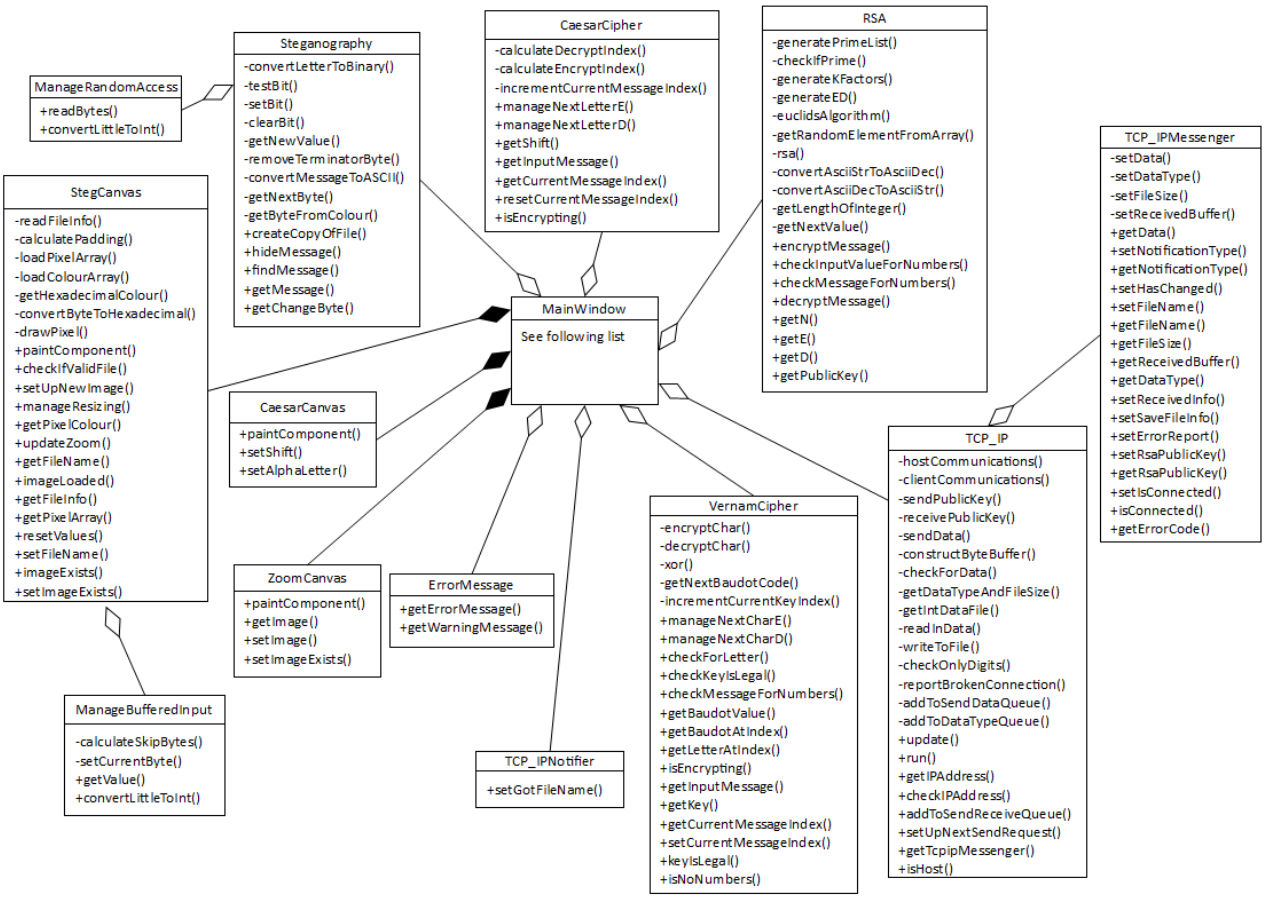
Module Structure Chart

The module structure chart shows the basic structure of each main class in the system to show, generally, what each can do.



Class Diagram

This class diagram shows each of the classes involved in the program code and the procedures and functions that each can perform. Since the MainWindow contains so many procedures, these are listed separately below the diagram.



The procedures of the MainWindow class are:

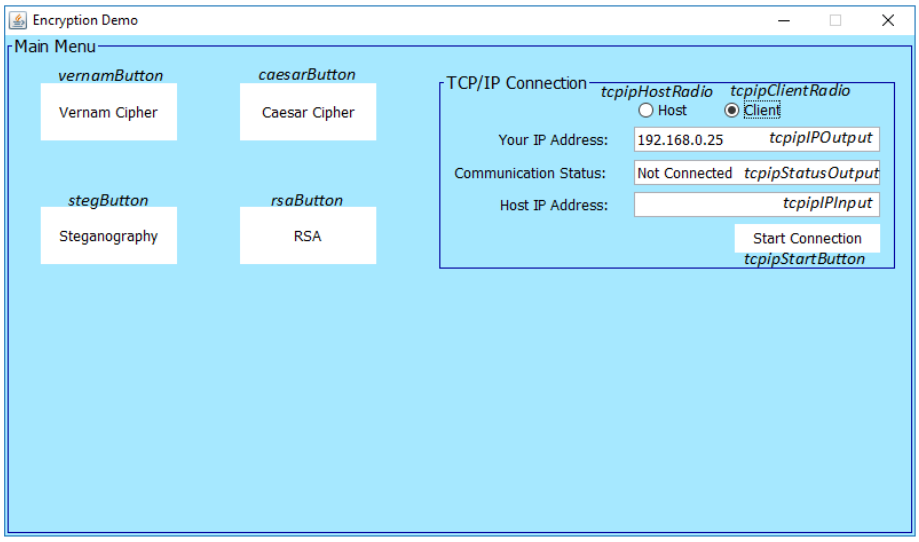
-update()
-displayConfirmDialog()
-loadTextFile()
-getLoadFileName()
-updateCurrentPanelInput()
-showIncomingFileDialog()
-getFileNameForTcpip()
-getSaveAsFileName()
-saveTextFile()
-loadTextFile()
-displayErrorMessage()
-displayWarningMessage()
-fillVernamTable()
-displayVernamWorkings()
-diplayChosenPanel()
-clearCurrentDisplay()
-loadNewStegImage()
-setStegCanvas1Info()
-setStegCanvas2Info()

-clearStegCanvas2Info()
-refreshStegImageWindow()
-copyFromTextEntry()
-clearRSAPrivateKey()
-showRSAKeys()
-getTextToSave()
-checkMessageIsASCII()
-vernamButtonActionPerformed()
-caesarButtonActionPerformed()
-caesarBackButtonActionPerformed()
-caesarShiftInputStateChanged()
-stegButtonActionPerformed()
-stegSelectImageButtonActionPerformed()
-stegBackButtonActionPerformed()
-stegHideMessageButtonActionPerformed()
-stegFindMessageButtonActionPerformed()
-stegClearButtonActionPerformed()
-vernamBackButtonActionPerformed()
-vernamOkButtonActionPerformed()
-vernamClearButtonActionPerformed()
-mainTBFSaveAsActionPerformed()
-mainTBFLoadActionPerformed()
-stegCanvas1MouseMoved()
-tcpipStartButtonActionPerformed()
-tcpipHostRadioActionPerformed()
-tcpipClientRadioActionPerformed()
-mainTBTSFileActionPerformed()
-mainTBTReceiveActionPerformed()
-mainTBEClearActionPerformed()
-stegImageWindowButtonActionPerformed()
-stegCanvas2MouseMoved()
-rsaOkButtonActionPerformed()
-rsaButtonActionPerformed()
-rsaBackButtonActionPerformed()
-mainTBTSMsgActionPerformed()
-mainTBFMainMenuActionPerformed()
-rsaClearButtonActionPerformed()
-mainTBFExitActionPerformed()
-mainTBECopyInputActionPerformed()
-mainTBECopyOutputActionPerformed()
-rsaHidePrivateKeyButtonActionPerformed()
+main()

2. Annotated Listings of Program Code

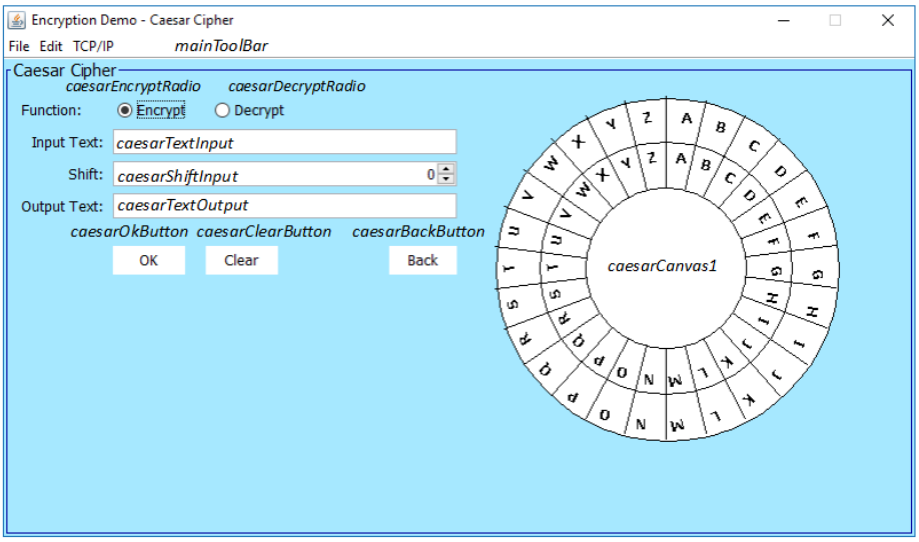
The following screenshots show each of the forms in the system and I have annotated the controls referenced by the program code on each form. I have also listed the classes that are relevant to each form. In addition to those listed for each screenshot, every form uses TCP_IP, TCP_IPMessenger and ErrorMessage. With the exception of the Main Menu form, every form also uses TCP_IPNotifier.

Main Menu:



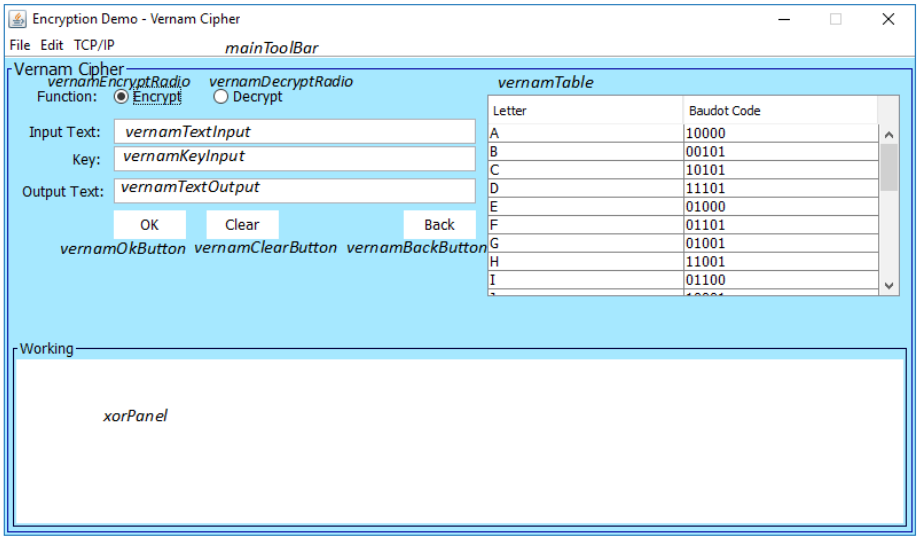
Caesar Cipher:

Associated Code: CaesarCipher, CaesarCanvas



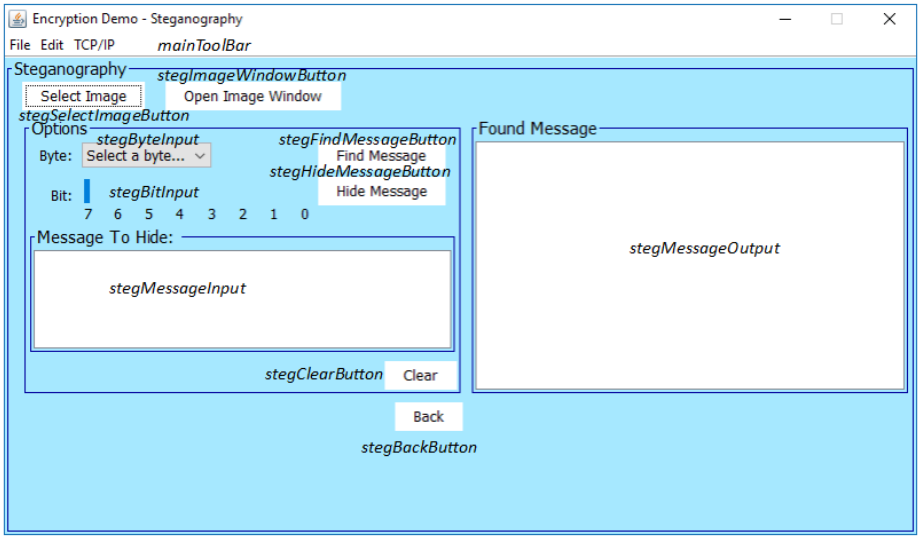
Vernam Cipher:

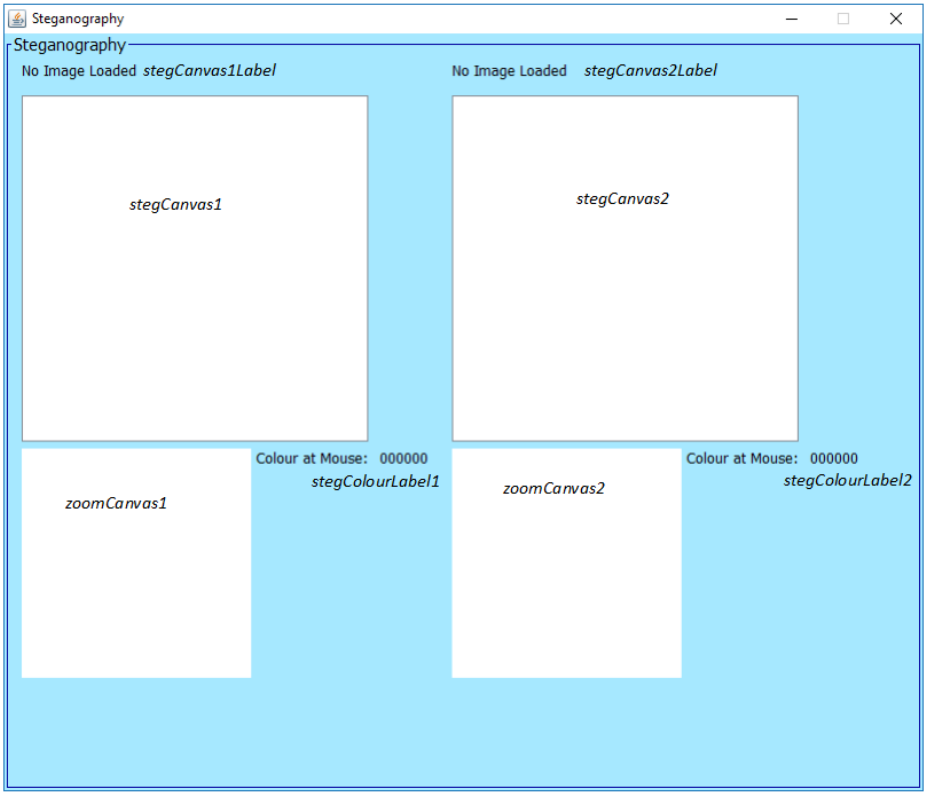
Associated Code: VernamCipher



Steganography:

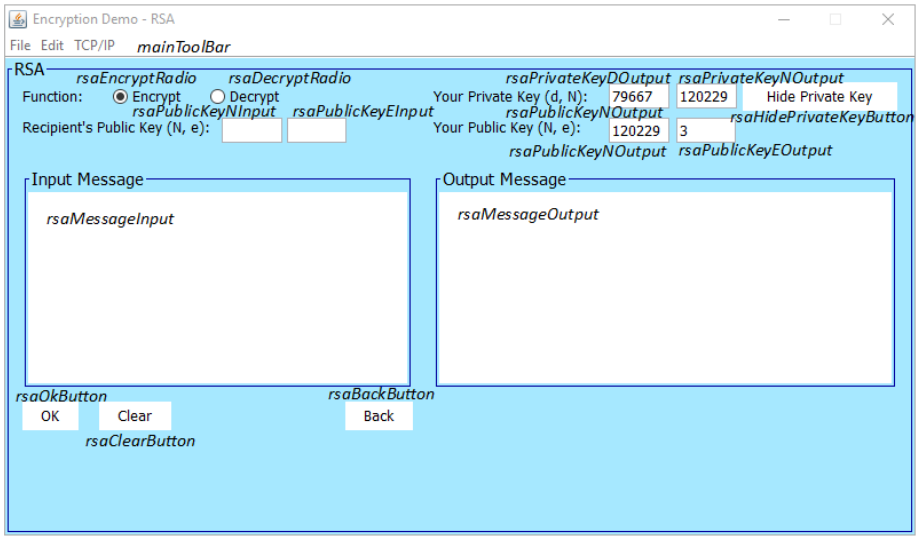
Associated Code: Steganography, StegCanvas, ManageRandomAccess, ManageBufferedInput, ZoomCanvas





RSA:

Associated Code: RSA



MainWindow.java

```
package main.main;

import java.awt.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;
```

```
import javax.swing.JOptionPane;

public class MainWindow extends javax.swing.JFrame implements Observer {

    //layout for the main window is card layout- the panels are swapped
around
    private CardLayout card;
    //which panel is currently being displayed
    private String currentCard;
    //whether there is a connection currently in place
    private boolean tcpipIsRunning;
    //tcpip thread
    private methods.methods.TCP_IP tcpip;
    //pass notifications to the tcpip thread (if it's running)
    private main.methods.TCP_IPNotifier tcpipNotifier;
    //stores the public and private key of the user and manages encryption
and decryption
    private methods.methods.RSA rsaManager;

    /**
     * Creates new form MainWindow
     */
    public MainWindow() {
      this.card = new CardLayout();
      this.currentCard = "mainMenuPanel";
      this.tcpipIsRunning = false;
      this.tcpipNotifier = new main.methods.TCP_IPNotifier();
      this.rsaManager = new methods.methods.RSA();

      initComponents();
      //starting on main menu so hide the toolbar as it's not required
      mainToolBar.setVisible(false);
    }
@Override
    public void update(Observable o, Object arg) {
      //this is run when tcpipMessenger notifies MainWindow that something
has changed in it

      //tcpipMessenger stores data to pass between the TCP/IP thread and
the main thread
      main.methods.TCP_IPMessenger tcpipMessenger =
tcpip.getTcpipMessenger();
      //update the connection status
      tcpipIsRunning = tcpipMessenger.isConnected();
      if (tcpipIsRunning) {
          tcpipStatusOutput.setText("Connected");
      } else {
          tcpipStatusOutput.setText("Not Connected");
      }
      //interpret the notificationType value (what sort of data will be
processed next
      int notificationType = tcpipMessenger.getNotificationType();
      //data is the data that is passed from the TCP/IP thread to the main
thread
      String data = tcpipMessenger.getData();
      //confirmed stores true if the user chooses Yes on a prompt box
      boolean confirmed;
      //what to do for each change
      switch (notificationType) {
          case (0):
            //vernam message received
```

```
            vernamTextInput.setText(data);
            vernamDecryptRadio.setSelected(true);
            //ask user if they want to switch to the vernam panel if
they're not on it
            if (!currentCard.equals("vernamPanel")) {
                confirmed = displayConfirmDialog("Do you want to switch to
Vernam Cipher?", "Vernam Message Received");
                if (confirmed) {
                  displayChosenPanel("vernamPanel", "Vernam Cipher");
                }
            }
            break;
          case (1):
            //caesar message received
            caesarTextInput.setText(data);
            caesarDecryptRadio.setSelected(true);
            //ask user if they want to switch to the caesar panel if
they're not on it
            if (!currentCard.equals("caesarPanel")) {
                confirmed = displayConfirmDialog("Do you want to switch to
Caesar Cipher?", "Caesar Message Received");
                if (confirmed) {
                  displayChosenPanel("caesarPanel", "Caesar Cipher");
                }
            }
            break;
          case (2):
            //text file received -
            //ask the user if they want to update the current panel's input
box
            confirmed = displayConfirmDialog(
                  "Do you want to load the text file\ninto the current
window's input field?",
                  "Text File Received");
            if (confirmed) {
                //loadText is the contents of the text file to load into
the window
                String loadText =
loadTextFile(tcpipMessenger.getFileName());
                updateCurrentPanelInput(loadText);
            }
            break;
          case (3):
            //bitmap file received-
            //ask the user if they want to switch to Steg and load the file
            //dialogMessage is the text that the dialog box should display
            String dialogMessage;
            if (!currentCard.equals("stegPanel")) {
                //ask user if they want to switch to the steg panel if
they're not on it
                //the image will not be loaded if they don't
                dialogMessage = "Do you want to switch to
Steganography\nand load the bitmap file?";
            } else {
                dialogMessage = "Do you want to load the bitmap file?";
            }
            confirmed = displayConfirmDialog(dialogMessage, "Bitmap File
Received");
            if (confirmed) {
                displayChosenPanel("stegPanel", "Steganography");
                loadNewStegImage(tcpipMessenger.getFileName());
```

```
            }
            break;
          case (4):
            //rsa message received
            rsaMessageInput.setText(data);
            rsaDecryptRadio.setSelected(true);
            //ask the user if they want to switch to the rsa panel if
they're not on it
            if (!currentCard.equals("rsaPanel")) {
                confirmed = displayConfirmDialog("Do you want to switch to
RSA?", "RSA Message Received");
                if (confirmed) {
                  displayChosenPanel("rsaPanel", "RSA");
                  //only decrypt the message if the panel is displayed
                  rsaOkButton.doClick();
                }
            }
            break;
          case (5):
            //get the filepath for the new text file
            showIncomingFileDialog("text");
            getFileNameForTcpip(".txt");
            break;
          case (6):
            //get the filepath for the new bitmap file
            showIncomingFileDialog("bitmap");
            getFileNameForTcpip(".bmp");
            break;
          case (7):
            //the other user's public key has been received-
            //update the rsa panel with it
            //publicKey is an array storing the other user's public key
            Integer[] publicKey = tcpipMessenger.getRsaPublicKey();
            rsaPublicKeyNInput.setText(publicKey[0] + "");
            rsaPublicKeyEInput.setText(publicKey[1] + "");
            break;
          case (8):
            //error has occurred on the tcpip thread-
            //notify the user
            displayErrorMessage(tcpipMessenger.getErrorCode());
          default:
            break;
      }
    }

    private boolean displayConfirmDialog(String message, String title) {
      //ask user question in message parameter and return their response

      //intVal is the value of the user's response
      int returnVal;
      returnVal = JOptionPane.showConfirmDialog(
            this,
            message,
            title,
            JOptionPane.YES_NO_OPTION);
      if (returnVal == JOptionPane.YES_OPTION) {
          return true;
      }
      return false;
    }
```

```
    private void loadTextFile() {
      //gets the name of the text file that the user wants to load
      //and loads the contents into the current panel's input box

      if (!currentCard.equals("mainMenuPanel")) {
          //filePath is the entire file path and name of the file to load
          String filePath = getLoadFileName();
          //check that a text file has been selected
          if (filePath.endsWith(".txt")) {
            //loadText stores the contents of the file to load
            String loadText = loadTextFile(filePath);
            updateCurrentPanelInput(loadText);
          } else if (!filePath.equals("")) {
            //the file is not a valid text file
            //this is only reached if a file was selected-
            //i.e. there is no error message for cancelling the action
            displayErrorMessage(16);
          }
      }
    }

    private String getLoadFileName() {
      //this procedure gets the name of the file that the user wants to
load
      //returns a blank string if they cancel the action

      //filePath is the entire path and name of the file to load
      String filePath = "";
      //returnVal is the value of the user's reponse to the dialog box
      int returnVal = loadFileWindow.showOpenDialog(this);
      if (returnVal == javax.swing.JFileChooser.APPROVE_OPTION) {
          filePath = loadFileWindow.getSelectedFile().getAbsolutePath();
      }
      return filePath;
    }

    private void updateCurrentPanelInput(String loadText) {
      //load the text into the right input text box

      switch (currentCard) {
          case ("vernamPanel"):
            vernamTextInput.setText(loadText);
            break;
          case ("caesarPanel"):
            caesarTextInput.setText(loadText);
            break;
          case ("stegPanel"):
            stegMessageInput.setText(loadText);
            break;
          case ("rsaPanel"):
            rsaMessageInput.setText(loadText);
            break;
          default:
            break;
      }
    }

    private void showIncomingFileDialog(String fileType) {
      //show a dialog to tell the user what type of file they are receiving
```

```
      JOptionPane.showMessageDialog(
            this,
            "Please enter a file name for the incoming " + fileType + "
file.",
            "Incoming file",
            JOptionPane.INFORMATION_MESSAGE);
    }

    private void getFileNameForTcpip(String fileSuffix) {
      //get the file name for an incoming file
      //and notify the tcpip thread that a file name has been got

      tcpip.getTcpipMessenger().setFileName(getSaveAsFileName(fileSuffix));
      tcpipNotifier.setGotFileName();
      tcpipNotifier.notifyObservers();
    }

    private String getSaveAsFileName(String fileSuffix) {
      //get the user to input a file name to save a file as
      //and add the right suffix (.txt or .bmp)
      //returns empty string if the user cancels the action

      //returnVal is the value of the user's response to the dialog box
      int returnVal = saveFileWindow.showSaveDialog(this);
      //filePath is the entire path and name to save the file under
      String filePath;
      if (returnVal == javax.swing.JFileChooser.APPROVE_OPTION) {
          filePath = saveFileWindow.getSelectedFile().getAbsolutePath();
          //if the user hasn't added the file extension, do so
          if (!filePath.endsWith(fileSuffix)) {
            filePath += fileSuffix;
          }
      } else {
          filePath = "";
      }
      return filePath;
    }

    private void saveTextFile(String saveText, String filePath) {
      //save the given text to a new text file at the location of the
filePath

      try {
          try {
            //create new file to write to
            File file = new File(filePath);
            file.createNewFile();
          } catch (IOException e) {
            displayErrorMessage(-1);
          }
          //output stream to write the data to the new file
          FileOutputStream newFile = new FileOutputStream(filePath);
          //convert the string to bytes before it can be written to the
file
          byte[] byteData = saveText.getBytes();
          //write to the file
          newFile.write(byteData);
          newFile.flush();
          newFile.close();
      } catch (IOException e) {
```

```
            displayErrorMessage(-1);
        }
    }

    private String loadTextFile(String filePath) {
        //load and return the contents of the text file at filePath

        //contents of the file to load
        String loadText = "";
        try {
            //input stream to read data from the file
            BufferedReader file = new BufferedReader(new
FileReader(filePath));
            //read the text file line by line and append to load text until
line reads null
            //line stores the next line read in from the file
            String line;
            while ((line = file.readLine()) != null) {
                loadText += line;
            }
            file.close();
        } catch (IOException e) {
            loadText = "";
            displayErrorMessage(-1);
        }
        return loadText;
    }

    private void displayErrorMessage(int errorCode) {
        //display error message that matches given error code

        //errorMessage is the message to display on the error message dialog
        String errorMessage =
main.methods.ErrorMessage.getErrorMessage(errorCode);
        JOptionPane.showMessageDialog(
                this,
                "Error Code: " + String.format("%02d", errorCode) + "\n" +
errorMessage,
                "Error",
                JOptionPane.ERROR_MESSAGE);
    }

    private void displayWarningMessage(int warningCode) {
        //displays a warning message for the user- not critical errors

        //warningMessage is the message to display on the warning dialog
        String warningMessage =
main.methods.ErrorMessage.getWarningMessage(warningCode);
        JOptionPane.showMessageDialog(
                this,
                "Warning Code: " + String.format("%02d", warningCode) + "\n" +
warningMessage,
                "Warning",
                JOptionPane.WARNING_MESSAGE);
    }

    private void fillVernamTable() {
        //fill the table on the vernamPanel to show the
        //corresponding Baudot code for each letter of the alphabet
```

```
        //x is a counter to iterate through the entire alphabet (an array in
VernamCipher)
        for (int x = 0; x < 26; x++) {
            //get values to put on current row
            //next to character to add to the table
            char letter = methods.methods.VernamCipher.getLetterAtIndex(x);
            //corresponding baudot code to add to the table
            String baudot = methods.methods.VernamCipher.getBaudotAtIndex(x);
            //put values on current row
            vernamTable.getModel().setValueAt(letter, x, 0);
            vernamTable.getModel().setValueAt(baudot, x, 1);
        }
    }

    private void displayVernamWorkings(char plainChar, char keyChar, String
encryptedBaudot) {
        //get the current text in the box and split into the different lines
into an array
        //row order: plainChar, keyChar, blank line, messageBaudot,
keyBaudot, encryptedBaudot

        //currentText is the current contents of xorPanel
        String[] currentText = xorPanel.getText().split("\n");
        //the baudot code of the next plaintext character to add to the panel
        String messageBaudot =
methods.methods.VernamCipher.getBaudotValue(plainChar);
        //the baudot code the next character in the key to add to the panel
        String keyBaudot =
methods.methods.VernamCipher.getBaudotValue(keyChar);
        //add to the current text
        currentText[0] += " " + "  " + plainChar + "  " + "\n";
        currentText[1] += " " + "  " + keyChar + "  " + "\n";
        currentText[2] += " " + messageBaudot + "\n";
        currentText[3] += " " + keyBaudot + "\n";
        currentText[4] += " " + encryptedBaudot;
        //reset the panel and show the new current text for each line
        xorPanel.setText("");
        //x iterates through each row to update the panel
        for (int x = 0; x < currentText.length; x++) {
            xorPanel.setText(xorPanel.getText() + currentText[x]);
        }

    }

    private void displayChosenPanel(String panelName, String title) {
        //display the given panel and change the window's title to match

        card = (CardLayout) (mainPanel.getLayout());
        card.show(mainPanel, panelName);
        if (!panelName.equals("mainMenuPanel")) {
            mainToolBar.setVisible(true);
            if (!this.getTitle().endsWith(title)) {
              this.setTitle("Encryption Demo - " + title);
            }
        } else {
            //if the main menu is being displayed, hide the toolbar
            //and reset the window's title
            mainToolBar.setVisible(false);
            this.setTitle("Encryption Demo");
        }
        currentCard = panelName;
```

```
    }

    private void clearCurrentDisplay() {
      //empty any text fields on the currently displayed panel

      switch (currentCard) {
          case ("vernamPanel"):
            vernamTextInput.setText("");
            vernamKeyInput.setText("");
            vernamTextOutput.setText("");
            xorPanel.setText(" \n \n \n \n \n");
          case ("caesarPanel"):
            caesarTextInput.setText("");
            caesarTextOutput.setText("");
            caesarShiftInput.setValue(0);
            caesarCanvas1.repaint();
            break;
          case ("stegPanel"):
            stegMessageInput.setText("");
            stegMessageOutput.setText("");
          case ("rsaPanel"):
            rsaMessageInput.setText("");
            rsaMessageOutput.setText("");
          default:
            break;
      }
    }

    private void loadNewStegImage(String filePath) {
      //set up stegCanvas1 to display

      if (stegCanvas1.checkIfValidFile(filePath)) {
          setStegCanvas1Info(filePath);
          stegImageWindow.setVisible(true);
          //stegCanvas2 still needs to be the same size as stegCanvas1 even
if
          //it's not displaying an image since the scrollbars of the two
are linked
          //canvasWidth is the width in pixels that the canvas needs to
resize to
          int canvasWidth = stegCanvas1.getFileInfo().get(3);
          //canvasHeight is the height in pixel that the canvas needs to
resize to
          int canvasHeight = stegCanvas1.getFileInfo().get(4);
          stegCanvas2.manageResizing(canvasWidth, canvasHeight);

          //clear anything on stegCanvas2 as it won't match stegCanvas1
anymore
          clearStegCanvas2Info();

          //refresh the window
          refreshStegImageWindow();
      } else {
          //file is not a valid bitmap file
          displayErrorMessage(17);
      }
    }

    private void setStegCanvas1Info(String filePath) {
      //set up the info required for stegCanvas1 to load a newly selected
image
```

```
      //(presumes it has been given an image)

      stegCanvas1.resetValues();
      stegCanvas1.setFileName(filePath);
      stegCanvas1.setUpNewImage();
      stegCanvas1.setImageExists(true);
      stegCanvas1Label.setText(filePath);
      stegCanvas1Label.setToolTipText(filePath);
    }

    private void setStegCanvas2Info(String filePath) {
      //set up the info required for stegCanvas2 to load a newly selected
image
      //(presumes it has been given an image)

      stegCanvas2.setFileName(filePath);
      stegCanvas2.setImageExists(true);
      stegCanvas2.resetValues();
      stegCanvas2Label.setText(filePath);
      stegCanvas2Label.setToolTipText(filePath);
    }

    private void clearStegCanvas2Info() {
      //clear stegCanvas2 when a new image is loaded up

      stegCanvas2.setImageExists(false);
      stegCanvas2.resetValues();
      stegCanvas2Label.setText("No Image Loaded");
      stegCanvas2Label.setToolTipText("No Image Loaded");
    }

    private void refreshStegImageWindow() {
      //refresh the window after the canvases have been given new images to
display

      stegCanvas1.repaint();
      stegCanvas2.repaint();
      stegImageWindow.revalidate();
    }

    private void copyFromTextEntry(javax.swing.JTextField textEntry) {
      //copies the entire contents of the given textField

      textEntry.selectAll();
      textEntry.copy();
    }

    private void copyFromTextEntry(javax.swing.JTextArea textEntry) {
      //copies the entire contents from the given textArea

      textEntry.selectAll();
      textEntry.copy();
    }

    private void clearRSAPrivateKey() {
      //clear the textFields displaying the private key to hide it

      rsaPrivateKeyNOutput.setText("");
      rsaPrivateKeyDOutput.setText("");
    }
```

```
    private void showRSAKeys() {
      //fill the text field displaying the user's public and private keys

      rsaPrivateKeyDOutput.setText("" + rsaManager.getD());
      rsaPrivateKeyNOutput.setText("" + rsaManager.getN());
      rsaPublicKeyNOutput.setText("" + rsaManager.getN());
      rsaPublicKeyEOutput.setText("" + rsaManager.getE());
    }

    private String getTextToSave() {
      //get the text from the output box of the panel currently displayed

      //save text stores the text to save to a file
      String saveText;
      switch (currentCard) {
          case ("stegPanel"):
            saveText = stegMessageOutput.getText();
            break;
          case ("vernamPanel"):
            saveText = vernamTextOutput.getText();
            break;
          case ("caesarPanel"):
            saveText = caesarTextOutput.getText();
            break;
          case ("rsaPanel"):
            saveText = rsaMessageOutput.getText();
            break;
          default:
            saveText = "";
      }
      return saveText;
    }

    private boolean checkMessageIsASCII(String message) {
      //checks that a message is only made of ASCII characters

      //isASCII will stay true if the message is only ASCII
      boolean isASCII = true;
      //i iterates from 0 to the end of the message
      for (int i = 0; i < message.length(); i++) {
          /**
           * if any character's binary is more than 127 (max value of 7 bit
           * binary) then the message is not completely ASCII and not valid
           */
          //charValue is the integer value of the next character
          int charValue = message.charAt(i);
          if (charValue > 127) {
            isASCII = false;
            break;
          }
      }
      return isASCII;
    }

    private void vernamButtonActionPerformed(java.awt.event.ActionEvent
evt) {
      displayChosenPanel("vernamPanel", "Vernam Cipher");
      fillVernamTable();
    }
```

```
    private void caesarButtonActionPerformed(java.awt.event.ActionEvent
evt) {
      displayChosenPanel("caesarPanel", "Caesar Cipher");
    }

    private void
caesarClearButtonActionPerformed(java.awt.event.ActionEvent evt) {
      clearCurrentDisplay();
    }

    private void caesarBackButtonActionPerformed(java.awt.event.ActionEvent
evt) {
      displayChosenPanel("mainMenuPanel", "");
    }

    private void caesarOkButtonActionPerformed(java.awt.event.ActionEvent
evt) {
      //performs the caesar cipher

      //clear the output box
      caesarTextOutput.setText("");
      //create new CaesarCipher object for processing the message
      methods.methods.CaesarCipher caesar = new
methods.methods.CaesarCipher(
            (int) caesarShiftInput.getValue(),
            caesarTextInput.getText(),
            caesarEncryptRadio.isSelected());

      if ((caesar.getShift() % 26 == 0)) {
          //warn the user that the entered shift has no effect
          displayWarningMessage(1);
      }
      //method called by the timer every time it ticks
      java.awt.event.ActionListener processNextLetter = new
java.awt.event.ActionListener() {
          @Override
          public void actionPerformed(java.awt.event.ActionEvent e) {
            //inputText is the message to encrypt or decrypt
            String inputText = caesar.getInputMessage();
            //end of the input message
            if (caesar.getCurrentMessageIndex() == inputText.length()) {
                return;
            } else {
                if (caesar.isEncrypting()) {
                  //encrypt letter
                  // get char to output and plain char
                  //outputPlain stores the output character and the plain
character
                  char[] outputPlain = caesar.manageNextLetterE();
                  //character to add to the output
                  char outputChar = outputPlain[0];
                  //plaintext character
                  char plainChar = outputPlain[1];
                  //give outputChar to output text field
                  caesarTextOutput.setText(caesarTextOutput.getText() +
outputChar);
                  //give plainChar to the canvas - if it's a letter, the
letter
                  //will be highlighted, otherwise it will be ignored by
the canvas
```

```
      caesarCanvas1.setAlphaLetter((Character.toUpperCase(plainChar)));
                } else {
                  //decrypt letter
                  //character to add to the output
                  char outputChar = caesar.manageNextLetterD();
                  caesarTextOutput.setText(caesarTextOutput.getText() +
Character.toLowerCase(outputChar));
                  caesarCanvas1.setAlphaLetter((outputChar));
                }
                //request that the Canvas is redrawn (doesn't do it
immediately)
                caesarCanvas1.repaint();
            }
          }
      };
      //length of the message
      int messageLength = caesar.getInputMessage().length();
      if (messageLength > 0) {
          //create a time delay for the animation so it doesn't all happen
at once
          javax.swing.Timer timeDelay = new javax.swing.Timer(1500,
processNextLetter);
          timeDelay.setInitialDelay(1500);
          timeDelay.start();

          if (caesar.getCurrentMessageIndex() == messageLength) {
            //end the timer when it has reached the end of the message
            timeDelay.stop();
          }
      } else {
          //no message to en/decrypt
          displayErrorMessage(3);
      }
    }

    private void caesarShiftInputStateChanged(javax.swing.event.ChangeEvent
evt) {
      //update the caesar wheel's state when the shift is changed

      //shift value chosen by the user for encrypting/decrypting
      int shift = (Integer) caesarShiftInput.getValue();
      caesarCanvas1.setShift(shift);
      caesarCanvas1.repaint();
    }

    private void stegButtonActionPerformed(java.awt.event.ActionEvent evt)
{
      displayChosenPanel("stegPanel", "Steganography");
    }

    private void
stegSelectImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
      //get the name of the file to load
      //and load it (presumes that a bitmap has been selected)

      //entire path and name of the bitmap file to load
      String filePath = getLoadFileName();
      if (!filePath.equals("")) {
          loadNewStegImage(filePath);
      }
```

```
    }

    private void stegBackButtonActionPerformed(java.awt.event.ActionEvent
evt) {
      displayChosenPanel("mainMenuPanel", "");
    }

    private void
stegHideMessageButtonActionPerformed(java.awt.event.ActionEvent evt) {
      //create a copy of the currently loaded bitmap file and hide the
message in it

      //create steganography object to process the copying of the image and
hiding the message
      methods.methods.Steganography steganography = new
methods.methods.Steganography(
            stegMessageInput.getText(),
            stegBitInput.getValue(),
            (String) stegByteInput.getSelectedItem());
      if (stegCanvas1.imageLoaded()) {
          if (steganography.getMessage().length() > 0) {
            if (checkMessageIsASCII(steganography.getMessage())) {
                if (steganography.getChangeByte() > 0) {
                  //user input of name of new file (to hide message in)
                  //entire path and name to save the file as
                  String filePath = getSaveAsFileName(".bmp");
                  if (!filePath.equals("")) {
                      //copy the file and hide the message in the copy
                      if (!filePath.equals(stegCanvas1.getFileName())) {

      steganography.createCopyOfFile(stegCanvas1.getFileName(), filePath);
                        setStegCanvas2Info(stegCanvas1.getFileName());
                        //file info is the information required by a canvas
to draw the image
                        ArrayList<Integer> fileInfo =
stegCanvas1.getFileInfo();
                        stegCanvas1.resetValues();
                        //hide the message in the copy of the image
                        steganography.hideMessage(filePath, fileInfo);
                        setStegCanvas1Info(filePath);
                        //refresh the window
                        stegCanvas1.repaint();
                        stegCanvas2.repaint();
                        stegImageWindow.revalidate();
                      } else {
                        //attempt to overwrite original file
                        displayErrorMessage(8);
                      }
                  }
                } else {
                  //no byte selected
                  displayErrorMessage(7);
                }
            } else {
                //message isn't ascii
                displayErrorMessage(6);
            }
          } else {
            //no message entered
            displayErrorMessage(5);
          }
```

```
        } else {
            //no image loaded
            displayErrorMessage(4);
        }
    }

    private void
stegFindMessageButtonActionPerformed(java.awt.event.ActionEvent evt) {
        //find a hidden message in the currently loaded image

        //create new steganography object to process finding the message
        methods.methods.Steganography steganography = new
methods.methods.Steganography();
        if (stegCanvas1.imageLoaded()) {
            //if changeByte of stegCanvas.fileInfo is 0, the picture has
never been edited
            //so there is no message
            if (stegCanvas1.getFileInfo().get(0) > 0) {
              steganography.findMessage(stegCanvas1.getPixelArray(),
stegCanvas1.getFileInfo());
              //foundMessage is the message that has been found in the image
              String foundMessage = steganography.getMessage();
              stegMessageOutput.setText(foundMessage);
            } else {
              //there is no message in the image!
              JOptionPane.showMessageDialog(
                    this,
                    "There is no message in this image.",
                    "No Message To Find",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        } else {
            //no image loaded to find a message in
            displayErrorMessage(4);
        }
    }

    private void stegClearButtonActionPerformed(java.awt.event.ActionEvent
evt) {
        clearCurrentDisplay();
    }

    private void vernamBackButtonActionPerformed(java.awt.event.ActionEvent
evt) {
        displayChosenPanel("mainMenuPanel", "");
    }

    private void vernamOkButtonActionPerformed(java.awt.event.ActionEvent
evt) {
        //performs the vernam cipher

        xorPanel.setText(" \n \n \n \n \n");
        vernamTextOutput.setText("");
        //create new vernam object to process the en/decryption
        methods.methods.VernamCipher vernam = new
methods.methods.VernamCipher(
              vernamEncryptRadio.isSelected(),
              vernamTextInput.getText(),
              vernamKeyInput.getText().replace(" ", ""));

        //action listener for the timer
```

```
      java.awt.event.ActionListener processNextLetter = new
java.awt.event.ActionListener() {
          @Override
          public void actionPerformed(java.awt.event.ActionEvent e) {
            //inputText is the message to en/decrypt
            String inputText = vernam.getInputMessage();
            if (vernam.getCurrentMessageIndex() == inputText.length()) {
                return;
            } else if (vernam.isEncrypting()) {
                //if encrypting, no numbers in plaintext
                if (vernam.isNoNumbers()) {
                  if (vernam.keyIsLegal()) {
                      //encrypt the next char if it's a letter or return
the plainChar
                      //array containing the cipher character to add, the
key character used
                      //and the plain character that was encrypted
                      String[] newCharKeyPlain = vernam.manageNextCharE();
                      //character to add to the ciphertext
                      String charToAdd = newCharKeyPlain[0];
                      //key character that was used to encrypt if plain
char was a letter
                      String keyChar = newCharKeyPlain[1];
                      //the plainchar that was processed this time
                      String plainChar = newCharKeyPlain[2];
                      if (!charToAdd.equals("" + plainChar)) {
                        //display workings if it didn't return the char
                        //(it returned a ciphertext instead)
                        displayVernamWorkings(plainChar.charAt(0),
keyChar.charAt(0), charToAdd);
                      }
                      //add the charToAdd to output no matter what
                      vernamTextOutput.setText(vernamTextOutput.getText() +
charToAdd);
                  } else {
                      vernam.setCurrentMessageIndex(inputText.length());
                      //key is not legal
                      displayErrorMessage(1);
                  }
                } else {
                  vernam.setCurrentMessageIndex(inputText.length());
                  //numbers in input plaintext
                  displayErrorMessage(0);
                }
            } else if (vernam.keyIsLegal()) {
                //array containing the plainchar, key character used for
decryption
                //and the baudot code from the input message
                String[] charKeyBaudot = vernam.manageNextCharD();
                //decrypt letter
                //plaintext character to add
                String valueToAdd = charKeyBaudot[0];
                if ((valueToAdd.length() == 1) &
(vernam.checkForLetter((int) valueToAdd.charAt(0)))) {
                  //character in the key used for decryption (if a baudot
code was processed)
                  String keyChar = charKeyBaudot[1];
                  //baudot code processed from the message if there was one
                  String baudotCode = charKeyBaudot[2];
                  displayVernamWorkings(valueToAdd.charAt(0),
keyChar.charAt(0), baudotCode);
```

```
                }
                vernamTextOutput.setText(vernamTextOutput.getText() +
valueToAdd);
            } else {
                vernam.setCurrentMessageIndex(inputText.length());
                //non-letter characters in key
                displayErrorMessage(1);
            }
        }
    };
    if ((vernam.getInputMessage().length() > 0)
          & (vernam.getKey().length() > 0)) {
        if (vernam.keyIsLegal() & vernam.isEncrypting()) {
          if (!(vernam.getKey().length() >=
vernam.getInputMessage().length())) {
                //key should be same length as message
                displayWarningMessage(0);
            }
        }
        //timer to delay the animation so it doesn't all happen at once
        javax.swing.Timer timeDelay = new javax.swing.Timer(1500,
processNextLetter);
        timeDelay.setInitialDelay(1500);
        timeDelay.start();
        if (vernam.getCurrentMessageIndex() ==
vernam.getInputMessage().length()) {
            //end the timer when it has reached the end of the message
            timeDelay.stop();
        }
    } else {
        //missing input text/key
        displayErrorMessage(2);
    }
  }
  private void
vernamClearButtonActionPerformed(java.awt.event.ActionEvent evt) {
    clearCurrentDisplay();
  }

  private void mainTBFSaveAsActionPerformed(java.awt.event.ActionEvent
evt) {
    //saves the output text from Vernam/Caesar/Steg to a text file

    if (!currentCard.equals("mainMenuPanel")) {
        //get the filepath for the new file
        String filePath = getSaveAsFileName(".txt");
        if (!filePath.equals("")) {
            //need to know exactly which method is being used to know where
to
            //get the text to put in the file
            String saveText = getTextToSave();
            //having got the required info, can now save the file (or
attempt to)
            saveTextFile(saveText, filePath);
        }
    }
  }

  private void mainTBFLoadActionPerformed(java.awt.event.ActionEvent evt)
{
```

```
      //load the contents of a text file to the input text box of the
current method being used

      loadTextFile();
    }

    private void stegCanvas1MouseMoved(java.awt.event.MouseEvent evt) {
      //update the zoom canvas for stegCanvas1 to show the area around the
cursor

      //get mouse position on screen (not on canvas)
      //x coordinate of mouse on screen
      int x = evt.getXOnScreen();
      //y coordinate of mouse on scren
      int y = evt.getYOnScreen();
      //get the colour at the coordinates of the mouse
      String pixelColour = stegCanvas1.getPixelColour(x, y);
      stegCanvas1.updateZoom(x, y, zoomCanvas1);
      stegColourLabel1.setText(pixelColour);
    }

    private void tcpipStartButtonActionPerformed(java.awt.event.ActionEvent
evt) {
      //initiate a TCP/IP connection with another user

      //isHost is true if this user will be host
      boolean isHost = tcpipHostRadio.isSelected();
      //check that there is no connection already set up
      if (!tcpipIsRunning) {
          if (isHost) {
            //start connection as the host
            tcpip = new methods.methods.TCP_IP(isHost, this,
rsaManager.getPublicKey());
            tcpipStatusOutput.setText("Waiting for client...");
            //put the tcpip on a separate thread to the rest of the program
so it doesn't hang the program
            tcpip.start();
            //tcpipNotifier used by mainWindow to tell the tcpip thread
            //when the user has entered a file name to save an incoming
file
            tcpipNotifier.addObserver(tcpip);
          } else {
            //start connection as the client
            //get the IP address of the user to connect to
            String ipAddress = tcpipIPInput.getText();
            if (methods.methods.TCP_IP.checkIPAddress(ipAddress)) {
                tcpip = new methods.methods.TCP_IP(isHost, ipAddress, this,
rsaManager.getPublicKey());
                tcpipStatusOutput.setText("Waiting for host...");
                //put the tcpip on a separate thread to the rest of the
program so it doesn't hang the program
                tcpip.start();
                //tcpipNotifier used by mainWindow to tell the tcpip thread
                //when the user has entered a file name to save an incoming
file
                tcpipNotifier.addObserver(tcpip);
            } else {
                //invalid IP address
                displayErrorMessage(11);
            }
```

```
            }
        } else {
            //connection already exists
            displayErrorMessage(13);
        }
    }

    private void tcpipHostRadioActionPerformed(java.awt.event.ActionEvent
evt) {
        //hide the host IP input because it is not required for connecting as
a host

        tcpipIPInputLabel.setVisible(false);
        tcpipIPInput.setVisible(false);
    }

    private void tcpipClientRadioActionPerformed(java.awt.event.ActionEvent
evt) {
        //show the host IP input for a client to enter the IP address

        tcpipIPInputLabel.setVisible(true);
        tcpipIPInput.setVisible(true);
    }

    private void mainTBTSFileActionPerformed(java.awt.event.ActionEvent
evt) {
        //sets up a request to send a file over TCP/IP

        if (tcpipIsRunning) {
            //entire path and name to save file as
            String filePath = getLoadFileName();
            if (!filePath.equals("")) {
              if (filePath.endsWith(".txt")) {
                  tcpip.setUpNextSendRequest(filePath, 2);
              } else if (stegCanvas1.checkIfValidFile(filePath)) {
                  tcpip.setUpNextSendRequest(filePath, 3);
              } else if (!filePath.equals("")) {
                  //the file is not a valid file type to send
                  displayErrorMessage(15);
              }

            }
        }
    }

    private void mainTBTReceiveActionPerformed(java.awt.event.ActionEvent
evt) {
        //request for TCP/IP thread to check for incoming data

        if (tcpipIsRunning) {
            tcpip.addToSendReceiveQueue(1);
        }
    }

    private void mainTBEClearActionPerformed(java.awt.event.ActionEvent
evt) {
        //clears the contents of the text fields on the currently displayed
panel

        clearCurrentDisplay();
```

```
    }

    private void
stegImageWindowButtonActionPerformed(java.awt.event.ActionEvent evt) {
      //reopen the window displaying the bitmap chosen for steganography
      //but only if there is an image chosen to display

      if (stegCanvas1.imageExists()) {
          stegImageWindow.setVisible(true);
      }
    }

    private void stegCanvas2MouseMoved(java.awt.event.MouseEvent evt) {
      //update the zoom canvas for stegCanvas2 to show the area around the
cursor

      //get mouse position on screen (not on canvas)
      //x coordinate of mouse on screen
      int x = evt.getXOnScreen();
      //y coordinate of mouse on screen
      int y = evt.getYOnScreen();
      //get the colour at the coordinates of the mouse
      String pixelColour = stegCanvas2.getPixelColour(x, y);
      stegCanvas2.updateZoom(x, y, zoomCanvas2);
      stegColourLabel2.setText(pixelColour);
    }

    private void rsaOkButtonActionPerformed(java.awt.event.ActionEvent evt)
{
      //performs RSA

      if (rsaEncryptRadio.isSelected()) {
          //encrypting the input message
          //get the recipient's public key from the input boxes

          //N value of the public key to use as a string to check if the
input
          //is a number
          String recipientN = rsaPublicKeyNInput.getText();
          //e value of the public key to use as a string to check if the
input
          //is a number
          String recipientE = rsaPublicKeyEInput.getText();
          //array of the two String values to pass to
checkInputForNumbers()
          String[] inputValues = {recipientN, recipientE};
          //check that the values for the key are valid -
          //will throw an error if they can't be parsed to integer
          if (rsaManager.checkInputValueForNumbers(inputValues)) {
            //get N as an integer to use
            int N = Integer.parseInt(recipientN);
            //get e as an integer to use
            int E = Integer.parseInt(recipientE);
            //message to encrypt
            String message = rsaMessageInput.getText();
            if (checkMessageIsASCII(message)) {
                //the result of encrypting the message
                String encryptedMessage =
rsaManager.encryptMessage(message, E, N);
                rsaMessageOutput.setText(encryptedMessage);
```

```
            } else {
                //message is not completely ASCII
                displayErrorMessage(6);
            }
        } else {
            //key isn't valid/ doesn't exist
            displayErrorMessage(9);
        }

    } else {
        //decrypting the input message
        //message to decrypt
        String message = rsaMessageInput.getText();
        if (rsaManager.checkMessageForNumbers(message)) {
            //result of decrypting the message
            String decryptedMessage = rsaManager.decryptMessage(message);
            rsaMessageOutput.setText(decryptedMessage);
        } else {
            //Can't decrypt the message
            displayErrorMessage(10);
        }
    }
  }

  private void rsaButtonActionPerformed(java.awt.event.ActionEvent evt) {
    displayChosenPanel("rsaPanel", "RSA");
    showRSAKeys();
  }

  private void rsaBackButtonActionPerformed(java.awt.event.ActionEvent
evt) {
    displayChosenPanel("mainMenuPanel", "");
  }

  private void mainTBTSMsgActionPerformed(java.awt.event.ActionEvent evt)
{
    //request to send a message over TCP/IP

    //data stores the data to pass to the TCP/IP thread to send
    String data;
    //dataType indicates what method the data is from (0-VC, 1=CC, 4=RSA)
    int dataType = -1;
    //get the right text to send and signal what type it is to tcpip
    switch (currentCard) {
        case ("vernamPanel"):
            data = vernamTextOutput.getText();
            dataType = 0;
            break;
        case ("caesarPanel"):
            data = caesarTextOutput.getText();
            dataType = 1;
            break;
        case ("rsaPanel"):
            data = rsaMessageOutput.getText();
            dataType = 4;
            break;
        default:
            data = "";
            break;
    }
    if (!data.equals("")) {
```

```
            tcpip.setUpNextSendRequest(data, dataType);
        }
    }

    private void mainTBFMainMenuActionPerformed(java.awt.event.ActionEvent
evt) {
        displayChosenPanel("mainMenuPanel", "");
    }
    private void rsaClearButtonActionPerformed(java.awt.event.ActionEvent
evt) {
        clearCurrentDisplay();
    }

    private void mainTBFExitActionPerformed(java.awt.event.ActionEvent evt)
{
        System.exit(0);
    }

    private void mainTBECopyInputActionPerformed(java.awt.event.ActionEvent
evt) {
        //Copy function (not mapped to Ctrl-C as there are two different copy
procedures)

        //copy the entire contents of the input field of the currently
displayed panel
        switch (currentCard) {
            case ("vernamPanel"):
              copyFromTextEntry(vernamTextInput);
            case ("caesarPanel"):
              copyFromTextEntry(caesarTextInput);
            case ("stegPanel"):
              copyFromTextEntry(stegMessageInput);
            case ("rsaPanel"):
              copyFromTextEntry(rsaMessageInput);
            default:
              break;
        }
    }

    private void
mainTBECopyOutputActionPerformed(java.awt.event.ActionEvent evt) {
        //Copy function (not mapped to Ctrl-C as there are two different copy
procedures)

        //copy the entire contents of the output field of the currently
displayed panel
        switch (currentCard) {
            case ("vernamPanel"):
              copyFromTextEntry(vernamTextOutput);
            case ("caesarPanel"):
              copyFromTextEntry(caesarTextOutput);
            case ("stegPanel"):
              copyFromTextEntry(stegMessageOutput);
            case ("rsaPanel"):
              copyFromTextEntry(rsaMessageOutput);
            default:
              break;
        }
    }
```

```
    private void
rsaHidePrivateKeyButtonActionPerformed(java.awt.event.ActionEvent evt) {
      //hides the private key if the button is toggled on

      if (rsaHidePrivateKeyButton.isSelected()) {
          clearRSAPrivateKey();
          rsaHidePrivateKeyButton.setText("Show Private Key");
      } else {
          showRSAKeys();
          rsaHidePrivateKeyButton.setText("Hide Private Key");
      }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
      try {
          for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Windows".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;

            }
          }
      } catch (ClassNotFoundException ex) {
          java.util.logging.Logger.getLogger(MainWindow.class
                .getName()).log(java.util.logging.Level.SEVERE, null, ex);

      } catch (InstantiationException ex) {
          java.util.logging.Logger.getLogger(MainWindow.class
                .getName()).log(java.util.logging.Level.SEVERE, null, ex);

      } catch (IllegalAccessException ex) {
          java.util.logging.Logger.getLogger(MainWindow.class
                .getName()).log(java.util.logging.Level.SEVERE, null, ex);

      } catch (javax.swing.UnsupportedLookAndFeelException ex) {
          java.util.logging.Logger.getLogger(MainWindow.class
                .getName()).log(java.util.logging.Level.SEVERE, null, ex);
      }

      /* Create and display the form */
      java.awt.EventQueue.invokeLater(new Runnable() {
          public void run() {
            new MainWindow().setVisible(true);

          }
      });
    }
    // Variables declaration
    private javax.swing.JButton caesarBackButton;
    private javax.swing.JButton caesarButton;
    private javax.swing.ButtonGroup caesarButtonGroup1;
    private gui.components.CaesarCanvas caesarCanvas1;
    private javax.swing.JButton caesarClearButton;
    private javax.swing.JRadioButton caesarDecryptRadio;
    private javax.swing.JRadioButton caesarEncryptRadio;
    private javax.swing.JButton caesarOkButton;
    private javax.swing.JPanel caesarPanel;
```

```
    private javax.swing.JSpinner caesarShiftInput;
    private javax.swing.JTextField caesarTextInput;
    private javax.swing.JTextField caesarTextOutput;
    private javax.swing.JPanel canvasPanel1;
    private javax.swing.JPanel canvasPanel2;
    private javax.swing.JScrollPane canvasScroll1;
    private javax.swing.JScrollPane canvasScroll2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel20;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel6;
    private javax.swing.JPanel jPanel7;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
    private javax.swing.JScrollPane jScrollPane5;
    private javax.swing.JScrollPane jScrollPane6;
    private javax.swing.JPopupMenu.Separator jSeparator1;
    private javax.swing.JPopupMenu.Separator jSeparator2;
    private javax.swing.JFileChooser loadFileWindow;
    private javax.swing.JPanel mainMenuPanel;
    private javax.swing.JPanel mainPanel;
    private javax.swing.JMenuItem mainTBEClear;
    private javax.swing.JMenuItem mainTBECopyInput;
    private javax.swing.JMenuItem mainTBECopyOutput;
    private javax.swing.JMenu mainTBEdit;
    private javax.swing.JMenuItem mainTBFExit;
    private javax.swing.JMenuItem mainTBFLoad;
    private javax.swing.JMenuItem mainTBFMainMenu;
    private javax.swing.JMenuItem mainTBFSaveAs;
    private javax.swing.JMenu mainTBFile;
    private javax.swing.JMenu mainTBTCPIP;
    private javax.swing.JMenuItem mainTBTReceive;
    private javax.swing.JMenuItem mainTBTSFile;
    private javax.swing.JMenuItem mainTBTSMsg;
    private javax.swing.JMenu mainTBTSend;
    private javax.swing.JMenuBar mainToolBar;
    private javax.swing.JButton rsaBackButton;
    private javax.swing.JButton rsaButton;
    private javax.swing.ButtonGroup rsaButtonGroup1;
    private javax.swing.JButton rsaClearButton;
    private javax.swing.JRadioButton rsaDecryptRadio;
    private javax.swing.JRadioButton rsaEncryptRadio;
```

```
    private javax.swing.JToggleButton rsaHidePrivateKeyButton;
    private javax.swing.JTextArea rsaMessageInput;
    private javax.swing.JTextArea rsaMessageOutput;
    private javax.swing.JButton rsaOkButton;
    private javax.swing.JPanel rsaPanel;
    private javax.swing.JTextField rsaPrivateKeyDOutput;
    private javax.swing.JTextField rsaPrivateKeyNOutput;
    private javax.swing.JTextField rsaPublicKeyEInput;
    private javax.swing.JTextField rsaPublicKeyEOutput;
    private javax.swing.JTextField rsaPublicKeyNInput;
    private javax.swing.JTextField rsaPublicKeyNOutput;
    private javax.swing.JFileChooser saveFileWindow;
    private javax.swing.JButton stegBackButton;
    private javax.swing.JSlider stegBitInput;
    private javax.swing.JButton stegButton;
    private javax.swing.JComboBox<String> stegByteInput;
    private gui.components.StegCanvas stegCanvas1;
    private javax.swing.JLabel stegCanvas1Label;
    private gui.components.StegCanvas stegCanvas2;
    private javax.swing.JLabel stegCanvas2Label;
    private javax.swing.JButton stegClearButton;
    private javax.swing.JLabel stegColourLabel1;
    private javax.swing.JLabel stegColourLabel2;
    private javax.swing.JButton stegFindMessageButton;
    private javax.swing.JButton stegHideMessageButton;
    private javax.swing.JFrame stegImageWindow;
    private javax.swing.JButton stegImageWindowButton;
    private javax.swing.JTextArea stegMessageInput;
    private javax.swing.JTextArea stegMessageOutput;
    private javax.swing.JPanel stegPanel;
    private javax.swing.JButton stegSelectImageButton;
    private javax.swing.ButtonGroup tcpipButtonGroup1;
    private javax.swing.JRadioButton tcpipClientRadio;
    private javax.swing.JRadioButton tcpipHostRadio;
    private javax.swing.JTextField tcpipIPInput;
    private javax.swing.JLabel tcpipIPInputLabel;
    private javax.swing.JTextField tcpipIPOutput;
    private javax.swing.JPanel tcpipPanel;
    private javax.swing.JButton tcpipStartButton;
    private javax.swing.JTextField tcpipStatusOutput;
    private javax.swing.JButton vernamBackButton;
    private javax.swing.JButton vernamButton;
    private javax.swing.ButtonGroup vernamButtonGroup1;
    private javax.swing.JButton vernamClearButton;
    private javax.swing.JRadioButton vernamDecryptRadio;
    private javax.swing.JRadioButton vernamEncryptRadio;
    private javax.swing.JTextField vernamKeyInput;
    private javax.swing.JButton vernamOkButton;
    private javax.swing.JPanel vernamPanel;
    private javax.swing.JTable vernamTable;
    private javax.swing.JTextField vernamTextInput;
    private javax.swing.JTextField vernamTextOutput;
    private javax.swing.JTextArea xorPanel;
    private gui.components.ZoomCanvas zoomCanvas1;
    private gui.components.ZoomCanvas zoomCanvas2;
    // End of variables declaration
}
```

ErrorMessage.java

```
package main.methods;
```

```
public class ErrorMessage {

    public static String getErrorMessage(int errorCode) {
      //provides the text to display when an error occurs

      //errorMessage stores the message to return for displaying
        String errorMessage;
        switch (errorCode) {
            //Vernam Cipher codes
            case (0):
                errorMessage = "Input text contains numbers.\nRemove any
numbers and try again.";
                break;
            case (1):
                errorMessage = "Key contains non-letter characters.\nRemove
these characters and try again.";
                break;
            case (2):
                errorMessage = "Input text/key is missing.\nEnter the
message and the key and try again.";
                break;
            //Caesar Cipher codes
            case (3):
                errorMessage = "Input text is missing.\nEnter the message
and try again.";
                break;
            //Steganography codes
            case (4):
                errorMessage = "No image loaded.\nLoad an image before
continuing.";
                break;
            case (5):
                errorMessage = "There is no message to hide in the
image.\nEnter a message and try again.";
                break;
            case (6):
                errorMessage = "Message cannot be hidden.\nCheck that the
message only contains ASCII characters and try again.";
                break;
            case (7):
                errorMessage = "No byte has been selected to hide the
message in.\nSelect a byte and try again.";
                break;
            case (8):
                errorMessage = "The new file cannot overwrite the
original.\nChange the file location or filename and try again.";
                break;
            //RSA codes
            case (9):
                errorMessage = "The recipient's public key is not
valid.\nEnter two values consisting of only digits and try again.";
                break;
            case (10):
                errorMessage = "Cannot decrypt the message.\nCheck that the
message only contains numbers and try again.";
                break;
            //TCP/IP codes
            case (11):
                errorMessage = "The IP address entered is invalid.\nAn IP
address contains four numbers between 0 and 255 separated by a dot
(.).\nRetype the address and try again.";
```

```
                break;
            case (12):
                errorMessage = "Unable to make a TCP/IP connection.";
//general inability
                break;
            case (13):
                errorMessage = "A connection already exists.\nTo make a new
connection, restart the program.";
                break;
            case (14):
                errorMessage = "The connection has been broken.\nTo make a
new connection, return to the main menu.";
                break;
            case (15):
                errorMessage = "The file format is not a valid format to
send.\nSelect a text (.txt) or bitmap (.bmp) file and try again.";
                break;
            //Save/Load codes
            case (16):
                errorMessage = "The file format is not a valid format to
load.\nSelect a text (.txt) file and try again.";
                break;
            case (17):
                errorMessage = "The file format is not a valid format to
load.\nSelect a bitmapped (.bmp) file and try again.";
                break;
            case (18):
                errorMessage = "The file cannot be saved in the chosen
location.\nSelect another location and try again.";
                break;
            default:
                errorMessage = "An error has occurred."; //unanticipated
error occurs
                break;
        }
        return errorMessage;

    }

    public static String getWarningMessage(int warningCode) {
      //gets the warning message to display

      //warningMessage stores the message to return to display
        String warningMessage;
        switch (warningCode) {
            case (0):
                warningMessage = "For perfect security, the key should be
the same length as the message to encrypt.";
                break;
            case (1):
                warningMessage = "The chosen shift has no effect.";
                break;
            default:
                warningMessage = "";
                break;

        }
        return warningMessage;
    }
}
```

TCP_IP.java

```
package methods.methods;

import java.io.*;

import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.Observable;
import java.util.Observer;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

public class TCP_IP extends Thread implements Observer {

    //size of the byte array used to receive data
    private final int maxBufferSize = 65536; // max possible buffer size

    //lets the class know whether it is to run hostCommunications or
clientCommunications
    private boolean isHost;
    //inputted ip address of the host to connect to (only for client)
    private String hostName;
    //public key to send to tcpip partner for rsa encryption
    Integer[] rsaPublicKey;
    //socket for the connection
    private Socket sock;
    //has a connection been made?
    private boolean isConnected = false;
    //stores whether the next task is to send something or check for
incoming data
    private BlockingQueue<Integer> sendReceiveQueue;
    //stores the data to send for each send request in sendReceiveQueue
    private BlockingQueue<String> sendDataQueue;
    //stores the type of data for each element of sendDataQueue
    private BlockingQueue<Integer> dataTypeQueue;
    //notifies the main window of received data and errors
    private main.methods.TCP_IPMessenger tcpipMessenger;

    public TCP_IP(boolean isHost, String hostName, Observer observer,
Integer[] rsaPublicKey) {
        //constructor for tcp ip connection as client

        //init messenger
        this.tcpipMessenger = new main.methods.TCP_IPMessenger();

        //init queues
        this.dataTypeQueue = new LinkedBlockingQueue<Integer>();
        this.sendDataQueue = new LinkedBlockingQueue<String>();
        this.sendReceiveQueue = new LinkedBlockingQueue<Integer>();

        //init properties requiring the values of the parameters
        this.isHost = isHost;
        this.hostName = hostName;
        tcpipMessenger.addObserver(observer);
        this.rsaPublicKey = rsaPublicKey;

    }
```

```
    public TCP_IP(boolean isHost, Observer observer, Integer[]
rsaPublicKey) {
        //constructor for tcp ip connection as host

        //init messenger
        this.tcpipMessenger = new main.methods.TCP_IPMessenger();

        //init queues
        this.dataTypeQueue = new LinkedBlockingQueue<Integer>();
        this.sendDataQueue = new LinkedBlockingQueue<String>();
        this.sendReceiveQueue = new LinkedBlockingQueue<Integer>();

        //init properties requiring the values of the parameters
        this.isHost = isHost;
        tcpipMessenger.addObserver(observer);
        this.rsaPublicKey = rsaPublicKey;
    }

    @Override
    public void update(Observable o, Object arg) {
        //main window will notify this class when it has got a file name to
save a received file under
      //fileName is the entire path and name to save a received file under
        String fileName = tcpipMessenger.getFileName();
        if (!fileName.equals("")) {
            //save the file and notify the main window
            writeToFile(fileName, tcpipMessenger.getReceivedBuffer(),
tcpipMessenger.getFileSize());
            tcpipMessenger.setReceivedInfo(fileName,
tcpipMessenger.getDataType());
        }
    }

    @Override
    public void run() {
      //executed when the TCP/IP thread is run
        if (!isConnected) {
            if (isHost) {
                hostCommunications();
                sendPublicKey();
                receivePublicKey();
            } else {
                clientCommunications();
                receivePublicKey();
                sendPublicKey();
            }
        }
      //main program for the TCP/IP thread- waits for sending/receiving
requests
        while (true) {
            if (!sendReceiveQueue.isEmpty()) {
                try {
                //nextRequest is an integer indicator of what the next
request is
                //(0 for sending, 1 for receiving)
                    int nextRequest = sendReceiveQueue.take();
                    if (nextRequest == 0) {
                        sendData();
                    } else {
                        checkForData();
```

```
                    }
                } catch (InterruptedException e) {
                    tcpipMessenger.setErrorReport(-1);
                }
            }
        }
    }

    private void hostCommunications() {
      //start connection as host

        try {
            //open connection on port 10000
            int port = 10000;
            //server is the socket to open connection on
            ServerSocket server = new ServerSocket(port);
            //times out after a minute
            server.setSoTimeout(60000);
            //wait for client to connect
            sock = server.accept();
            isConnected = true;
            tcpipMessenger.setIsConnected(true);
        } catch (IOException e) {
            tcpipMessenger.setErrorReport(12);
        }
    }

    private void clientCommunications() {
      //start connection as client

        try {
            //request connection on port 10000 with the host
            int port = 10000;
            sock = new Socket(hostName, port);
            isConnected = true;
            tcpipMessenger.setIsConnected(true);
        } catch (UnknownHostException e) {
            tcpipMessenger.setErrorReport(11);
        } catch (IOException e) {
            tcpipMessenger.setErrorReport(12);
        }
    }

    private void sendPublicKey() {
        //sends the user's rsa public key

        try {
          //data is sent via TCP/IP using an output stream
            DataOutputStream dos = new
DataOutputStream(sock.getOutputStream());
            //set up a buffer to send the two values in 4 bytes each
            ByteBuffer sendBuffer = ByteBuffer.allocate(8);
            for (int x = 0; x < 2; x++) {
                sendBuffer.putInt(rsaPublicKey[x]);
            }
            dos.write(sendBuffer.array());
            dos.flush();
        } catch (IOException e) {
            reportBrokenConnection();
        }
```

```
    }

    private void receivePublicKey() {
        //receives the other user's rsa public key that they have sent

        try {
            //array to store the received public key in as integers
            Integer[] publicKey = new Integer[2];
            //data is received via TCP/IP using an input stream
            DataInputStream dis = new
DataInputStream(sock.getInputStream());
            //public key is originally received in binary
            byte[] receivedKey = new byte[8];
            dis.read(receivedKey);
            //split the 8 bytes read in into the two values
            publicKey[0] =
java.nio.ByteBuffer.wrap(Arrays.copyOfRange(receivedKey, 0,
4)).order(java.nio.ByteOrder.BIG_ENDIAN).getInt();
            publicKey[1] =
java.nio.ByteBuffer.wrap(Arrays.copyOfRange(receivedKey, 4,
8)).order(java.nio.ByteOrder.BIG_ENDIAN).getInt();
            //notify mainWindow that the public key has been received
            tcpipMessenger.setRsaPublicKey(publicKey);
            tcpipMessenger.setNotificationType(7);
            tcpipMessenger.setHasChanged();
        } catch (IOException e) {
            reportBrokenConnection();
        }
    }

    private void sendData() {
      //sends data (file or message) across TCP/IP

        //sets up a buffer with the data and its metadata to send across
TCP/IP
        ByteBuffer sendBuffer;
        try {
            //output stream for sending the file across TCP/IP
            DataOutputStream dos = new
DataOutputStream(sock.getOutputStream());
            //get data to send
            String data = sendDataQueue.take();
            //get the data type of the data to send
            int dataType = dataTypeQueue.take();
            if (!(dataType == 2) & !(dataType == 3)) {
                //sending a message
            //convert the message to bytes to add to sendBuffer
                byte[] message = data.getBytes();
                //create the buffer to send across tcp ip
                sendBuffer = constructSendBuffer(dataType, message.length,
message);
            } else {
                //sending a file
                File sendFile = new File(data);
                //length of the file to send
                int fileLength = (int) sendFile.length();
                //input stream for reading in the file to send
                BufferedInputStream bis = new BufferedInputStream(new
FileInputStream(sendFile));
                //get the data from the file
                byte[] fileBuffer = new byte[fileLength];
```

```
                bis.read(fileBuffer);
                //create the buffer to send across tcp ip
                sendBuffer = constructSendBuffer(dataType,
fileBuffer.length, fileBuffer);
            }
            //send the contents of the buffer
            dos.write(sendBuffer.array());
            dos.flush();
        } catch (IOException e) {
            reportBrokenConnection();
        } catch (InterruptedException e) {
            //use default error message
            tcpipMessenger.setErrorReport(-1);
        }
    }

    private ByteBuffer constructSendBuffer(int dataType, int fileSize,
byte[] data) {
        //creates the buffer to send through tcp ip
        //buffer structure = data type; file size; data

        //sendBuffer stores all of the information that will be sent across
TCP/IP
        ByteBuffer sendBuffer = ByteBuffer.allocate(fileSize + 8);
        //add the data type first
        sendBuffer.putInt(dataType);
        sendBuffer.position(4);
        //add the size of the data being sent
        sendBuffer.putInt(fileSize);
        sendBuffer.position(8);
        //add the data to send
        sendBuffer.put(data, 0, fileSize);

        return sendBuffer;
    }

    private void checkForData() {
        //reads in incoming data

        try {
            //input stream for reading in data from TCP/IP
            DataInputStream dis = new
DataInputStream(sock.getInputStream());
            //get data type and file size to know what and how much data is
incoming
            int[] dataFile = getDataTypeAndFileSize(dis);
          //integer indicator of the type of data received
            int dataType = dataFile[0];
          //number of bytes the data uses in the buffer
            int fileSize = dataFile[1];
            //get the data
            ByteBuffer receivedBuffer = readInData(fileSize, dis);
            if (!(dataType == 2) & !(dataType == 3)) {
                //if the data is a message, convert to a string
                //and notify main window to update the relevant input text
box
            //message received from TCP/IP
                String message = new String(receivedBuffer.array());
                tcpipMessenger.setReceivedInfo(message, dataType);
            } else {
```

```
                //data type 2 = txt file- get filename by sending 5 to main
window
                //data type 3 = bmp file- get filename by sending 6 to main
window
                //i.e. notification type = data type + 3
                tcpipMessenger.setNotificationType(dataType + 3);
                //receivedBuffer, fileSize and dataType are local to this
                //procedure so store them in tcpipMessenger so they can be
                //accessed by the update procedure of this class when the
main
                //window sends a file name to save the data under
                tcpipMessenger.setSaveFileInfo(receivedBuffer, fileSize,
dataType);
                //reset file name and notify the main window that the user
needs to input a file name
                tcpipMessenger.setFileName(null);
                tcpipMessenger.setHasChanged();
            }
        } catch (IOException e) {
            reportBrokenConnection();
        }
    }

    private int[] getDataTypeAndFileSize(DataInputStream dis) throws
IOException {
        //read in a byte array containing the data type and file size

        //byte array to store the bytes of the the two integers to read in
        byte[] dataFile = new byte[8];
        dis.read(dataFile);
        return getIntDataFile(dataFile);
    }

    private int[] getIntDataFile(byte[] dataFile) {
        //get the integer values of the data type and file size
        //integer value of the data type (converted from bytes)
        int dataType =
java.nio.ByteBuffer.wrap(Arrays.copyOfRange(dataFile, 0,
4)).order(java.nio.ByteOrder.BIG_ENDIAN).getInt();
        //integer value of the file size (converted from bytes)
        int fileSize =
java.nio.ByteBuffer.wrap(Arrays.copyOfRange(dataFile, 4,
8)).order(java.nio.ByteOrder.BIG_ENDIAN).getInt();
        //array storing the two to return from this function
        int[] intDataFile = {dataType, fileSize};
        return intDataFile;
    }

    private ByteBuffer readInData(int fileSize, DataInputStream dis) throws
IOException {
        //reads in data from the data input stream of the socket

        //keep a count of how many bytes read
        int bytesRead = 0;
        //create buffer to store read data before writing to file
        ByteBuffer sendBuffer = ByteBuffer.allocate(fileSize);
        while (bytesRead < fileSize) {
            //get next chunk of data
            //received data stores the next chunk data read in from the
stream
            byte[] receivedBytes = new byte[maxBufferSize];
```

```
            //bytesReadNow is the actual number of bytes read from the
stream
            int bytesReadNow = dis.read(receivedBytes);
            bytesRead += bytesReadNow;

            //add to data buffer
            sendBuffer.put(receivedBytes, 0, bytesReadNow);
            sendBuffer.position(bytesRead);
        }
        return sendBuffer;

    }

    private void writeToFile(String fileName, ByteBuffer sendBuffer, int
noOfBytes) {
      //saves received files to a file name specified by the user
        try {
            //create new file to write to
            File file = new File(fileName);
            file.createNewFile();
            //output stream used to write data to the file
            FileOutputStream fos = new FileOutputStream(fileName);
            //write the contents of the buffer to the new file
            fos.write(sendBuffer.array(), 0, noOfBytes);
            fos.flush();
            fos.close();
        } catch (FileNotFoundException e) {
            tcpipMessenger.setErrorReport(-1);
        } catch (IOException e) {
            reportBrokenConnection();
        }
    }

    public static String getIPAddress() {
        //get the ip address of this computer

        try {
          //ipAddress is the string format of this computer's IP address
            String ipAddress = (String)
InetAddress.getLocalHost().getHostAddress();
            return ipAddress;
        } catch (UnknownHostException e) {
        }
        return "";
    }

    public static boolean checkIPAddress(String ipAddress) {
        //check that an input IP address is valid

        //isValid is true if the IP address is valid
        boolean isValid = true;
        //array that stores each section of the IP address (split by ".")
        String[] splitIP = ipAddress.split("\\.");
        //has to have four parts to be valid
        if (splitIP.length == 4) {
            for (int x = 0; x < splitIP.length; x++) {
                //if not all numbers, IP can't be valid
                if (checkOnlyDigits(splitIP[x])) {
                    //numbers have to be between 0 and 255 inclusive
                    //negatives should be caught by checkOnlyDigits (hyphen
isn't a digit)
```

```
                    if (Integer.parseInt(splitIP[x]) >= 256) {
                  isValid = false;
                  break;
                }
                } else {
                    isValid = false;
                    break;
                }
            }
        } else {
            isValid = false;
        }
        return isValid;
    }

    private static boolean checkOnlyDigits(String ipSection) {
        //check each quarter of the IP address is only made of digits
before
        //trying to parse the int

        //if the section of IP address passed to this function is only made
of numbers
        //isDigits is true
        boolean isDigits = true;
        for (int x = 0; x < ipSection.length(); x++) {
            if (!Character.isDigit(ipSection.charAt(x))) {
                isDigits = false;
                break;
            }
        }
        return isDigits;
    }

    private void reportBrokenConnection() {
        //connection has been broken

        tcpipMessenger.setIsConnected(false);
        tcpipMessenger.setErrorReport(14);
    }

    public void addToSendReceiveQueue(int value) {
        //adds a value to sendReceiveQueue to indicate whether the next
task is sending or receiving
        //0 = send, 1 = receive

        sendReceiveQueue.offer(value);
    }

    private void addToSendDataQueue(String data) {
      //adds the data to send to the send queue
        sendDataQueue.offer(data);
    }

    private void addToDataTypeQueue(int dataType) {
        //adds a value to dataTypeQueue to indicate what type the next data
to send
        dataTypeQueue.offer(dataType);
    }

    public void setUpNextSendRequest(String data, int dataType) {
```

```
        //set values required to create a new send request
        //needs the data to send and its data type
        addToSendDataQueue(data);
        addToDataTypeQueue(dataType);
        addToSendReceiveQueue(0);
    }

    public main.methods.TCP_IPMessenger getTcpipMessenger() {
        return tcpipMessenger;
    }

    public boolean isHost() {
        return isHost;
    }
}
```

TCP_IPMessenger.java

```
package main.methods;

import java.nio.ByteBuffer;
import java.util.Observable;

public class TCP_IPMessenger extends Observable {

    //data to send to MainWindow
    private String data;
    //sent to MainWindow so it knows what to do next
    private int notificationType;
    //MainWindow gets file name from user and stores it here
    private String fileName = null;
    //size of the received buffer (to write to a file)
    private int fileSize;
    //received buffer to write to a file
    private ByteBuffer receivedBuffer;
    //sent to TCP_IP so it knows what it's receiving
    private int dataType;
    //stores error code to give to main window to report to the user
    private int errorCode;
    //public key is received by TCP_IP and sent to MainWindow to update the
display
    private Integer[] rsaPublicKey;
    //whether a connection currently exists between the user and another
user
    private boolean isConnected = false;

    public void setReceivedInfo(String data, int notificationType) {
      //set the values required by the main window to update the display
with the received data
      //it needs the data and the type of data so it can update the right
display

      setData(data);
      setNotificationType(notificationType);
      //notify the main window
      setHasChanged();
    }

    public void setSaveFileInfo(ByteBuffer receivedBuffer, int fileSize,
int dataType) {
      //set the values required by TCP_IP.update() to save the file
```

```
      //it needs the file (receivedBuffer), the file's size and the type of
file

      setReceivedBuffer(receivedBuffer);
      setFileSize(fileSize);
      setDataType(dataType);
    }

    public void setErrorReport(int errorCode) {
      //reports an error back to MainWindow for it to display the
appropriate dialog

      setNotificationType(8);
      this.errorCode = errorCode;
      setHasChanged();
    }

    public void setHasChanged() {
      setChanged();
      notifyObservers();
    }

    private void setData(String data) {
      this.data = data;
    }

    public String getData() {
      return data;
    }

    public void setNotificationType(int notificationType) {
      this.notificationType = notificationType;
    }

    public int getNotificationType() {
      return notificationType;
    }

    public void setFileName(String fileName) {
      this.fileName = fileName;
    }

    public String getFileName() {
      return fileName;
    }

    private void setFileSize(int fileSize) {
      this.fileSize = fileSize;
    }

    public int getFileSize() {
      return fileSize;
    }

    private void setReceivedBuffer(ByteBuffer receivedBuffer) {
      this.receivedBuffer = receivedBuffer;
    }

    public ByteBuffer getReceivedBuffer() {
      return receivedBuffer;
    }
```

```
    private void setDataType(int dataType) {
      this.dataType = dataType;
    }

    public int getDataType() {
      return dataType;
    }

    public void setRsaPublicKey(Integer[] rsaPublicKey) {
      this.rsaPublicKey = rsaPublicKey;
    }

    public Integer[] getRsaPublicKey() {
      return rsaPublicKey;
    }

    public void setIsConnected(boolean isConnected) {
      this.isConnected = isConnected;
    }

    public boolean isConnected() {
      return isConnected;
    }

    public int getErrorCode() {
      return errorCode;
    }
}
```

TCP_IPNotifier.java

```
package main.methods;

import java.util.Observable;

public class TCP_IPNotifier extends Observable {
    //used by MainWindow to notify the tcpip thread that a file name has
been obtained
    //from the user to save a file as

    //the fileName itself is stored in tcpipMessenger
    //gotFileName is true when the user enters and confirms a filename to
save an
    //incoming file as
    private boolean gotFileName;

    public void setGotFileName() {
      //change the value of gotFileName and notify observers of the change

      gotFileName = !gotFileName;
      setChanged();
    }
}
```

CaesarCipher.java

```
package methods.methods;

import java.util.Arrays;

public class CaesarCipher {
```

```
    //letterList used to get the index of the input character and the
character it is en/decrypted to
    private final static Character[] letterList = {'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
'U', 'V', 'W', 'X', 'Y', 'Z'};
    //the shift to use in the en/decryption
    private int shift = 0;
    //message to en/decrypt
    private String inputMessage;
    //how far through the message it currently is
    private int currentMessageIndex = 0;
    //whether it is to encrypt or decrypt
    private boolean isEncrypting = true;

    public CaesarCipher(int shift, String inputMessage, boolean
isEncrypting) {
      this.shift = shift;
      this.inputMessage = inputMessage;
      this.isEncrypting = isEncrypting;
    }

    public char[] manageNextLetterE() {
      //gets the next character and encrypts it with the shift if it's a
letter
      //or encrypts as itself otherwise

      //attempt to find the index of the plainletter in the alphabet - if
the
      //character is not a letter, it won't be found so a negative value
will
      //be returned
      //returnArray stores the values needed to be returned to the main
program (cipherletter and plainletter)
      char[] returnArray = new char[2];
      //next character to process
      char nextChar =
Character.toUpperCase(inputMessage.charAt(currentMessageIndex));
      //where in letterList nextChar is (negative if it isn't)
      int charIndex = Arrays.asList(letterList).indexOf(nextChar);
      //can only encrypt letters so check for a negative index
      if (charIndex >= 0) {
          //encryptIndex is the calculated value of where the ciphertext is
in letterList
          int encryptIndex = calculateEncryptIndex(charIndex);
          //encryptedLetter is the ciphertext
          char encryptedLetter = letterList[encryptIndex];
          returnArray[0] = encryptedLetter;
      } else {
          //if a negative index was returned, just add the character to the
          //ciphertext
          returnArray[0] = nextChar;
      }
      incrementCurrentMessageIndex();
      returnArray[1] = nextChar;
      return returnArray;
    }

    private int calculateEncryptIndex(int charIndex) {
      //calculate index of plainletter in the alphabet encryption across
the
```

```
      //ends of the alphabet is managed by % 26 to "loop round"

      return (charIndex + shift) % 26;
    }

    public char manageNextLetterD() {
      //gets the next character and decrypts it with the shift if it's a
letter
      //or decrypts as itself otherwise

      //attempt to find the index of cipherletter in the alphabet - if the
      //character is not a letter, it won't be found so a negative value
will
      //be returned
      //returnChar is the plaintext character to return to main program
      char returnChar;
      //the next character in the message to process
      char nextChar =
Character.toUpperCase(inputMessage.charAt(currentMessageIndex));
      //where in letterList nextChar is located (negative if it's not)
      int charIndex = Arrays.asList(letterList).indexOf(nextChar);
      //can only decrypt letters so check for a negative index
      if (charIndex >= 0) {
          //calculate index of plainletter in letterList
          int decryptIndex = calculateDecryptIndex(charIndex);
          //the plaintext letter
          char decryptedLetter = letterList[decryptIndex];
          returnChar = decryptedLetter;
      } else {
          //if a negative value, return the character as it is
          returnChar = nextChar;
      }
      incrementCurrentMessageIndex();
      return returnChar;
    }

    private int calculateDecryptIndex(int charIndex) {
      //decrypt index is the index of a plaintext letter from a ciphertext
letter
      int decryptIndex = (charIndex - shift) % 26;
      if (decryptIndex < 0) {
          //negative indices here mean the encryption was across the ends
of
          //the alphabet - this is solved by taking the magnitude of the
          //index from 26 (or adding the entire value to 26)
          decryptIndex = 26 + decryptIndex;
      }
      return decryptIndex;
    }

    public int getShift() {
      return shift;
    }

    public String getInputMessage() {
      return inputMessage;
    }

    public int getCurrentMessageIndex() {
      return currentMessageIndex;
    }
```

```
    private void incrementCurrentMessageIndex() {
      currentMessageIndex++;
    }

    public void resetCurrentMessageIndex() {
      currentMessageIndex = 0;
    }

    public boolean isEncrypting() {
      return isEncrypting;
    }
}
```

CaesarCanvas.java

```
package gui.components;

public class CaesarCanvas extends javax.swing.JPanel {

    //the usual colour of the letters
    private final java.awt.Color letterColour;
    //if currently processing a message, the current letter and its
corresponding shift letter
    //will be highlighted in a different colour
    private final java.awt.Color currentLetterColour;
    //array of the alphabet for drawing the letters on the wheel
    private final Character[] alphabet = {'A', 'B', 'C', 'D', 'E', 'F',
'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
'V', 'W', 'X', 'Y', 'Z'};
    //center to outer edge of outer ring of the wheel
    private final int wheelRadiusOuter;
    //width of the letter rings
    private final int wheelWidth;
    //center to outer edge of inner ring of the wheel
    private final int wheelRadiusInner;
    //point to center the wheel around on the canvas(x = y = midpoint)
    private final int midpoint;
    //diameter of the wheel
    private final int canvasDimensionsOuter;
    //diameter of the inner ring's circle
    private final int canvasDimensionsInner;

    //the shift currently selected on the GUI
    //so the canvas can redraw the inner letters to show the shift on the
wheel
    private int shift = 0;
    //the letter to highlight on repainting
    private char alphaLetter;

    public CaesarCanvas() {
      //highlight the current letter in red
      this.currentLetterColour = java.awt.Color.RED;
      //all other letters will be black
      this.letterColour = java.awt.Color.BLACK;
      //initiate dimensions of the wheel
      this.wheelRadiusOuter = 150;
      this.wheelWidth = 40;
      this.wheelRadiusInner = wheelRadiusOuter - wheelWidth;
      this.midpoint = wheelRadiusOuter;
      this.canvasDimensionsOuter = wheelRadiusOuter * 2;
      this.canvasDimensionsInner = wheelRadiusInner * 2;
```

```
    }

    @Override
    public void paintComponent(java.awt.Graphics g) {
      //draw the Caesar Wheel

      super.paintComponent(g);
      //object used to draw on the canvas
      java.awt.Graphics2D g2d = (java.awt.Graphics2D) g.create();
      //draw the rings to display the alphabets on
      //draw fill colour of the circle
      //circles are drawn by specifying the top left corner, width and
height
      //of a square that the circle should fill
      g2d.setColor(java.awt.Color.WHITE);
      //int x, int y, int width, int height
      g2d.fillOval(2, 2, canvasDimensionsOuter - 1, canvasDimensionsOuter -
1);
      //draw edge colour of the outer edge of the wheel
      g2d.setColor(java.awt.Color.BLACK);
      g2d.drawOval(2, 2, canvasDimensionsOuter - 1, canvasDimensionsOuter -
1);

      //draw outer edge of the inner ring
      //get coordinates of the top left corner of the square to fill
      int innerWheelx = midpoint - wheelRadiusInner;
      int innerWheely = midpoint - wheelRadiusInner;
      //only draw the circumference as the whole wheel has already been
filled white
      g2d.drawOval(innerWheelx, innerWheely, canvasDimensionsInner,
canvasDimensionsInner);

      //draw the letters onto the wheel
      //the index in alphabet of the plaintext letter to draw on outer
wheel
      int alphaIndex = 0;
      //index in alphabet of the corresponding ciphertext letter to draw on
inner wheel
      int shiftIndex = 0;
      for (int num = 0; num < 26; num++) {
          //draw a line separating the next letter to draw from the last
drawn letter
          //calculate the angle to draw the next separating line at in
radians
          double radAngle = num * ((2 * Math.PI) / 26);
          //calculate the point on the circumference of the wheel where the
          //next separating line intersects the circumerference
          int x = (int) (midpoint + (wheelRadiusOuter *
Math.sin(radAngle)));
          int y = (int) (midpoint - (wheelRadiusOuter *
Math.cos(radAngle)));
          //draw the separating line
          g2d.drawLine(midpoint, midpoint, x, y);
          //calculate the coordinates to draw the next letters at
          double alphax = midpoint + ((wheelRadiusInner + 20) *
Math.sin(radAngle - (Math.PI / 26)));
          double alphay = midpoint - ((wheelRadiusInner + 20) *
Math.cos(radAngle - (Math.PI / 26)));
          double shiftx = midpoint + ((wheelRadiusInner - 20) *
Math.sin(radAngle - (Math.PI / 26)));
```

```
          double shifty = midpoint - ((wheelRadiusInner - 20) *
Math.cos(radAngle - (Math.PI / 26)));

          //index of the letter to draw on the outer ring
          //outer ring letters don't move
          alphaIndex = num - 1;
          if (alphaIndex < 0) {
            //drawing the outer ring letters starts at Z and then to A, B,
C...
            alphaIndex = 25;
          }

          //calculate the corresponding shift letter based on the shift
          shiftIndex = (alphaIndex + shift) % 26;
          //letters are rotated as they go round the wheel-
          //calculate the angle to draw the next letters at
          double rotAngle = (Math.PI / 13) - (num) * (Math.PI / 13);

          if (num <= 25) {
            //set apppearance of the letters
            g2d.setFont(new java.awt.Font("Tahoma", java.awt.Font.BOLD,
13));
            //set the rotation of the alpha letter around a point -
            //-2 and +3 are adjustment numbers to rotate it better
            g2d.rotate(-rotAngle, alphax - 2, alphay + 3);
            //check whether the letter is to be highlighted
            if (alphabet[alphaIndex].equals(alphaLetter)) {
                g2d.setColor(currentLetterColour);
            }
            //draw alpha letter
            g2d.drawString(alphabet[alphaIndex] + "", (float) alphax - 2,
(float) alphay + 3);
            //have to rotate back before re-rotating for the shift letter
            g2d.rotate(rotAngle, alphax - 2, alphay + 3);
            //rotate for shift letter
            g2d.rotate(-rotAngle, shiftx - 1, shifty + 1);
            //draw shift letter
            g2d.drawString(alphabet[shiftIndex] + "", (float) shiftx - 2,
(float) shifty - 2);
            //reset the rotation for the next letter
            g2d.rotate(rotAngle, shiftx - 1, shifty + 1);
            //reset colour for the next letter
            g2d.setColor(letterColour);
          }

      }
      //calculate the top left corner of the square to create the inner
circle
      //of the wheel
      int midWheelx = midpoint - wheelRadiusInner + wheelWidth;
      int midWheely = midpoint - wheelRadiusInner + wheelWidth;

      //draw white filled circle in the middle of the wheel for appearance
      g2d.setColor(java.awt.Color.WHITE);
      g2d.fillOval(midWheelx, midWheely, (wheelRadiusInner - wheelWidth) *
2, (wheelRadiusInner - wheelWidth) * 2);
      g2d.setColor(java.awt.Color.BLACK);
      g2d.drawOval(midWheelx, midWheely, (wheelRadiusInner - wheelWidth) *
2, (wheelRadiusInner - wheelWidth) * 2);
    }
```

```
    public void setShift(int shift) {
      this.shift = shift;
    }

    public void setAlphaLetter(char alphaLetter) {
      this.alphaLetter = alphaLetter;
    }

}
```

VernamCipher.java

```
package methods.methods;

import java.util.Arrays;

public class VernamCipher {

      // list of letters in the alphabet
      private final static Character[] letterList = { 'A', 'B', 'C', 'D',
'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
                  'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
'Y', 'Z' };
      // binary representation of each letter in the alphabet in 5 bits =
Baudot
      // Code
      private final static String[] baudotList = { "10000", "00101",
"10101", "11101", "01000", "01101", "01001", "11001",
                  "01100", "10001", "10011", "11011", "01011", "01111",
"11100", "11111", "10111", "00111", "00110", "10110",
                  "10100", "11110", "01110", "01010", "00100", "11010" };

      // is it encrypting or decrypting
      private boolean isEncrypting;
      // message to en/decrypt
      private String inputMessage;
      // key to use
      private String key;
      // the current position in the message
      private int currentMessageIndex;
      // the current position in the key
      private int currentKeyIndex;
      // is the key legal
      private boolean keyIsLegal;
      // the message cannot have any numbers if encrypting
      private boolean noNumbers = true;

      public VernamCipher(boolean isEncrypting, String inputMessage, String
key) {
            currentMessageIndex = 0;
            currentKeyIndex = 0;
            this.isEncrypting = isEncrypting;
            this.inputMessage = inputMessage;
            this.key = key;
            checkKeyIsLegal();
            // cannot have numbers in message if encrypting
            if (isEncrypting) {
                  checkMessageForNumbers();
            }
      }

      public String[] manageNextCharE() {
```

```
            // manages the next char during Encryption
            // char to add is the ciphertext baudot code if plaintext is a
letter
            // or the original char otherwise
            String charToAdd;
            // returnArray returns the character to add to the input and
the keyChar
            // (keyChar needed by displayVernamWorkings())
            String[] returnArray = new String[3];
            // check the next char is a letter - if it is, encrypt it
            // the next character in the message to process
            char plainChar = inputMessage.charAt(currentMessageIndex);
            // integer ASCII value of plainChar
            int asciiValue = (int) plainChar;
            if (checkForLetter(asciiValue)) {
                  plainChar = Character.toUpperCase(plainChar);
                  // get key character to XOR with
                  char keyChar =
Character.toUpperCase(key.charAt(currentKeyIndex));
                  // encrypt the current letter
                  charToAdd = encryptChar(plainChar, keyChar);
                  // increment the key index - if this becomes
                  // bigger than the key's length, % key's length
                  // to "loop round" to beginning of key again
                  incrementCurrentKeyIndex();
                  returnArray[1] = "" + keyChar;
                  returnArray[2] = "" + plainChar;
            } else {
                  // if the plainChar is not a letter, just add it
                  // to the ciphertext as it is
                  charToAdd = "" + plainChar;
                  // there is no keyChar and it won't be accessed so send a
blank string
                  returnArray[1] = "";
                  // plainChar is accessed, this needs to be the same as
charToAdd
                  returnArray[2] = charToAdd;
            }
            returnArray[0] = charToAdd;
            currentMessageIndex += 1;
            return returnArray;
      }

      private String encryptChar(char plainChar, char keyChar) {
            // xor the baudot of the message char with the baudot of the
key char

            // baudot code of plainChar
            String plainBaudot = getBaudotValue(plainChar);
            // baudot code of keyChar
            String keyBaudot = getBaudotValue(keyChar);
            // xor plainBaudot and keyBaudot to get encrypted value
            String encrypted = xor(plainBaudot, keyBaudot);
            return encrypted;
      }

      public String[] manageNextCharD() {
            // returns the decrypted character(or the character as it is)
            // and the key character and baudot code
            // array of values to return to main program
            String[] returnArray = new String[3];
```

```
            // new plaintext character to add to the current plaintext
            String valueToAdd;
            // manages the next char during Decryption
            // the next baudot code for decryption (may not actually be a
baudot
            // code)
            String baudotCode = getNextBaudotCode(inputMessage);
            if (baudotCode.length() == 5) {
                  // the character in the key to use to decrypt the code
                  char keyChar =
Character.toUpperCase(key.charAt(currentKeyIndex));
                  // XOR baudot with key to get original letter
                  valueToAdd = "" + decryptChar(baudotCode, keyChar);
                  returnArray[1] = "" + keyChar;
                  returnArray[2] = baudotCode;
                  incrementCurrentKeyIndex();
            } else {
                  // add what ever is in the baudotCode to the plaintext
                  valueToAdd = baudotCode;
                  // the values of these won't be accessed so send a blank
string for each
                  returnArray[1] = "";
                  returnArray[2] = "";
            }
            returnArray[0] = valueToAdd;
            return returnArray;
      }

      private char decryptChar(String cipherBaudot, char keyChar) {
            // xors the baudot of the message char with the baudot of the
key char

            // baudot code of the key
            String keyBaudot = getBaudotValue(keyChar);
            // xor keyBaudot with cipherBaudot to get plainBaudot
            String plainBaudot = xor(cipherBaudot, keyBaudot);
            // find letter that is represented by plainBaudot
            // where in baudotList and letterList the plain baudot code is
            int plainBaudotIndex =
Arrays.asList(baudotList).indexOf(plainBaudot);
            // the letter in the alphabet that corresponds to the plain
baudot code
            char plainLetter =
Arrays.asList(letterList).get(plainBaudotIndex);
            return plainLetter;
      }

      private String xor(String inputBaudot, String keyBaudot) {
            // xors the two binary values
            // first value to use for xor
            int a = Integer.parseInt(inputBaudot, 2);
            // second value to use for xor
            int b = Integer.parseInt(keyBaudot, 2);
            // result of xoring the two values
            String result = Integer.toBinaryString(a ^ b);
            // current length of the result (has to be 5 eventually)
            int resultLength = result.length();
            if (resultLength < 5) {
                  // make sure the returned result is 5 bits long
                  for (int i = 0; i < (5 - resultLength); i++) {
                        result = "0" + result;
```

```
                }
            }
            return result;

        }

        public boolean checkForLetter(int asciiValue) {
            // check that the passed integer is the ascii representation of
a letter

            // isLetter is false if the asciiValue doesn't represent a
letter
            boolean isLetter = false;

            // upper = 65-90 lower = 97-122
            if ((asciiValue >= 97) && (asciiValue <= 122) || (asciiValue >=
65) && (asciiValue <= 90)) {
                isLetter = true;
            }
            return isLetter;
        }

        public void checkKeyIsLegal() {
            // key can only be made up of letters - go through each
character and check they're all letters

            // if the key is a valid key, keyIsLegal is true
            keyIsLegal = true;
            for (int x = 0; x < key.length(); x++) {
                // the next char in the key to check
                char currentChar = key.charAt(x);
                if (!(checkForLetter((int) currentChar))) {
                    keyIsLegal = false;
                    break;
                }
            }
        }

        public void checkMessageForNumbers() {
            // check that the message doesn't contain any numbers

            // noNumbers is true if the message doesn't contain any numbers
            noNumbers = true;
            for (int x = 0; x < inputMessage.length(); x++) {
                // ascii value of the next character in the message to
check
                int asciiCode = (int) (inputMessage.charAt(x));
                // numbers are between 48 and 57 inclusive in ASCII
                if ((48 <= asciiCode) & (asciiCode <= 57)) {
                    noNumbers = false;
                    break;
                }
            }
        }

        private String getNextBaudotCode(String inputMessage) {
            // get the 5 bit pattern for decrypting without getting any
punctuation as well

            // the baudot code to return
            String baudotCode = "";
```

```
            while ((baudotCode.length() < 5) && (currentMessageIndex <
inputMessage.length())) {
                  // the next char in the string
                  String nextChar =
inputMessage.charAt(currentMessageIndex) + "";

                  // if a 1 or 0 is detected, presume to be the start of
the next baudot code and get the next 4 bits of the code
                  if ((nextChar.equalsIgnoreCase("1")) ||
(nextChar.equalsIgnoreCase("0"))) {
                        for (int x = 0; x < 5; x++) {
                              baudotCode +=
inputMessage.charAt(currentMessageIndex);
                              currentMessageIndex++;
                        }
                  } else {
                        baudotCode += nextChar;
                        currentMessageIndex++;
                        return baudotCode;
                  }
            }
            return baudotCode;
      }

      public static String getBaudotValue(char letter) {
            // get the baudot value of the letter passed to the function

            // where in letterList and baudotList letter is
            int letterIndex = Arrays.asList(letterList).indexOf(letter);
            // the baudot code at letterIndex in baudotList
            String letterBaudot = getBaudotAtIndex(letterIndex);
            return letterBaudot;
      }

      private void incrementCurrentKeyIndex() {
            currentKeyIndex = (currentKeyIndex + 1) % key.length();
      }

      public static String getBaudotAtIndex(int index) {
            return Arrays.asList(baudotList).get(index);
      }

      public static char getLetterAtIndex(int index) {
            return Arrays.asList(letterList).get(index);
      }

      public boolean isEncrypting() {
            return isEncrypting;
      }

      public String getInputMessage() {
            return inputMessage;
      }

      public String getKey() {
            return key;
      }

      public int getCurrentMessageIndex() {
            return currentMessageIndex;
      }
```

```
        public void setCurrentMessageIndex(int currentMessageIndex) {
              this.currentMessageIndex = currentMessageIndex;
        }

        public boolean keyIsLegal() {
              return keyIsLegal;
        }

        public boolean isNoNumbers() {
              return noNumbers;
        }
}
```

Steganography.java

```
package methods.methods;

import java.io.*;
import java.util.ArrayList;

public class Steganography {

    //object used for reading from files
    private methods.utils.ManageRandomAccess readManager;
    //message to hide in an image
    private String message;
    //bit in chosen byte to use to hide message
    private int changeBit;
    //byte to use to hide message in
    private int changeByte;

    public Steganography(String message, int changeBit, String chosenByte)
{
      //constructor to hide a message

      this.message = message;
      this.changeBit = changeBit;
      this.changeByte = getByteFromColour(chosenByte);
      this.readManager = new methods.utils.ManageRandomAccess();
    }

    public Steganography() {
      //constructor to find a message

      this.readManager = new methods.utils.ManageRandomAccess();
    }

    public void createCopyOfFile(String origFileName, String newFileName) {
      //make a copy of the image so the original can remain unchanged after
      //hiding a message

      try {
          try {
            //create new file to write to
            File file = new File(newFileName);
            file.createNewFile();
          } catch (IOException e) {
          }
          //open original to read from, and new to write to
          RandomAccessFile origFile = new RandomAccessFile(origFileName,
"rw");
```

```
            FileOutputStream newFile = new FileOutputStream(newFileName);

            //read the file size from the metadata of the bitmap
            origFile.seek(2);
            //binary value of filesize
            byte[] fileSizeBytes = new byte[4];
            fileSizeBytes = readManager.readBytes(origFile, 4);
            //decimal value of filesize
            int fileSize = readManager.convertLittleToInteger(fileSizeBytes);
            origFile.seek(0);

            //read 4 bytes at a time from original file
            byte[] fourBytes = new byte[4];
            //how many bytes have been read from the file
            int bytesRead = 0;

            while (bytesRead < fileSize) {
              origFile.read(fourBytes);
              newFile.write(fourBytes, 0, 4);
              bytesRead += 4;
            }
            newFile.flush();
            newFile.close();
            origFile.close();

        } catch (IOException e) {
        }
    }

    public void hideMessage(String fileName, ArrayList<Integer> fileInfo) {
        //hides the message in the file

        File file = new File(fileName);
        //get the position of the first byte to hide the message in
        int filePosition = fileInfo.get(2) + changeByte - 1;
        //width in pixels of the image
        int width = fileInfo.get(3);
        //number of extra bytes at the end of each row to make the length a
multiple of 4
        int padding = fileInfo.get(7);
        //keeps track of how many pixels in a row have been already been used
        int pixelCount = 0;
        try {
            //open the file to read values from and write values to
            RandomAccessFile raFile = new RandomAccessFile(file, "rw");
            for (int i = 0; i < message.length() + 1; i++) {
              //for each char:
              //binary value of the next letter
              String binaryLetter;
              if (i < message.length()) {
                  //get the next letter in the message to convert to binary
                  char letter = message.charAt(i);
                  //convert char to 8 bit binary
                  binaryLetter = convertLetterToBinary(letter);
              } else {
                  //add terminator byte to the end of the message so
                  //findMessage knows where to stop
                  binaryLetter = "00000000";
              }
              //for each bit in binary value of char:
              for (int x = 0; x < binaryLetter.length(); x++) {
```

```
                try {
                  //get the value in the file at the current position
                  raFile.seek(filePosition);
                  //decimal value of the next value read from the file
                  int fileValue =
readManager.convertLittleToInteger(readManager.readBytes(raFile, 1));
                  //return pointer to the beginning of the byte for writing
to
                  raFile.seek(filePosition);
                  // test the bit to change from the fileValue
                  int testResult = testBit(fileValue, changeBit);
                  //get the bit to write to the file
                  int writeBit = Integer.parseInt("" +
binaryLetter.charAt(x));
                  //calculate whether a new value is required to store the
bit
                  //and return what it is (256 if not required)
                  int newValue = getNewValue(testResult, writeBit,
fileValue, changeBit);
                  //the file only needs changing if newValue is less that 8
bits (newValue <=255)
                  //if newValue is larger than 8 bits then the file doesn't
need changing
                  if (newValue < 256) {
                      raFile.writeByte(newValue);
                      raFile.seek(filePosition);
                  }
                  pixelCount++;
                  filePosition += 3;
                  if (pixelCount == width) {
                      filePosition += padding;
                      pixelCount = 0;
                  }
                } catch (IOException e) {
                }
            }
          }
          //write the changed bit and byte to the file so findMessage can
find the message
          try {
            //create a 2 digit integer comprised of the changed byte and
            //bit side by side- the changed byte and bit can be obtained by
            //dividing and modding the 2 digit number by 10 respectively
            int changeByteBit = Integer.parseInt("" + changeByte + "" +
changeBit);
            //use the empty bytes at the beginning of the bitmap to write
it in
            raFile.seek(6);
            raFile.writeByte(changeByteBit);
          } catch (IOException e) {
          }
          try {
            raFile.close();
          } catch (IOException e) {
          }

      } catch (FileNotFoundException e) {
      }
    }

    private String convertLetterToBinary(char letter) {
```

```
      // convert a character to an 8 bit binary pattern

      //get binary value of letter
      String letterValue = Integer.toBinaryString((int) letter);
      //make sure binary value is 8 bits long
      String binaryValue = String.format("%8s", letterValue).replace(" ",
"0");
      return binaryValue;
    }

    private int testBit(int fileByte, int changeBit) {
      //check whether a bit is a 1 or a 0

      //mask to test the bit with
      int testMask = (int) Math.pow(2, changeBit);
      //and the byte with the test mask and check the result
      int result = fileByte & testMask;
      if (result == testMask) {
          // bit is a 1
          return 1;
      } else {
          //bit is a 0
          return 0;
      }
    }

    private int setBit(int fileByte, int changeBit) {
      //make the bit a 1

      //mask to set the bit with
      int setMask = (int) Math.pow(2, changeBit);
      // or the byte with the setMask
      int result = fileByte | setMask;
      return result;
    }

    private int clearBit(int fileByte, int changeBit) {
      //make the bit a 0

      //mask to clear the bit with
      int clearMask = 255 - (int) Math.pow(2, changeBit);
      //and the byte with the clearMask
      int result = fileByte & clearMask;
      return result;
    }

    private int getNewValue(int testResult, int writeBit, int fileValue,
int changeBit) {
      //get what value the current position in the file should hold

      //init newValue larger than 8 bits
      //newValue is the value that the file value should be changed to (256
if no change)
      int newValue = 256;
      if (testResult == 1) {
          if (writeBit == 0) {
            //clear the bit if it's 1 because writeBit is not a 1
            newValue = clearBit(fileValue, changeBit);
          }
      } else if (writeBit == 1) {
          //set the bit if it's 0 because writeBit is a 1
```

```
            newValue = setBit(fileValue, changeBit);
        }
        return newValue;
    }

    public void findMessage(byte[] pixelArray, ArrayList<Integer> fileInfo)
{
        //extracts the hidden message from the image using
StegCanvas.pixelArray

        //store binary of the message in binMessage to be converted to ASCII
        String binMessage = "";
        //length of the pixel array
        int pixelArrayLength = pixelArray.length;
        //get which byte of each pixel and
        //which bit of the byte is used to hide the message
        //byte the message is hidden in
        int changedByte = fileInfo.get(0);
        //bit in the byte used to hide the message
        int changedBit = fileInfo.get(1);
        //width in pixels of the image
        int width = fileInfo.get(3);
        //extra bytes at the end of each row to make the length a multiple of
4
        int padding = fileInfo.get(7);
        //changedByte = 1, 2 or 3 but x has to start as 0, 1 or 2
        int x = changedByte - 1;
        //keep track of how far through a row it is
        //(if at the end, x needs to skip the padding)
        int imageCol = 0;
        //zeroCount keeps track of how many 0s have been encountered- 8
signals
        //the end of the message
        int zeroCount = 0;
        //bitCount makes sure that zeroCount only counts 8 if the bits are
part
        //of the same byte-if bitCount reaches 8, the counters reset
        int bitCount = 0;
        while (x <= (pixelArrayLength)) {
            //store the current byte in a byte[] of 4
            //to convert it from little endian to an integer
            byte[] currentByte = new byte[4];
            currentByte[0] = pixelArray[x];
            //next value read from the file
            int currentInteger =
readManager.convertLittleToInteger(currentByte);
            //find out if the required bit is a 0 or 1
            int testResult = testBit(currentInteger, changedBit);
            if (testResult == 1) {
              binMessage += "1";
              //any chain of 0s has been broken so reset zeroCount
              zeroCount = 0;
            } else {
              binMessage += "0";
              //add another 0 to the count
              zeroCount += 1;
            }
            bitCount += 1;
            if (bitCount == 8) {
              if (zeroCount == 8) {
                  //don't need the terminator byte so can remove this
```

```
                //before converting to characters
                binMessage = removeTerminatorByte(binMessage);
                break;
            } else {
                bitCount = 0;
                zeroCount = 0;
            }
        }
        imageCol++;
        x += 3;
        if (imageCol == width) {
          // at the end of the row so x skips past the padding
          imageCol = 0;
          x += padding;
        }
    }
    //convert the message before returning it the main program to be
displayed on the GUI
    message = convertMessageToASCII(binMessage);
  }

  private String removeTerminatorByte(String binMessage) {
    //copies the message to a new string but leaves off the last 8 bits
to
    //get rid of the terminator byte

    //newBinMessage is the message after the last byte has been taken off
    String newBinMessage = "";
    for (int x = 0; x < binMessage.length() - 8; x++) {
        newBinMessage += binMessage.charAt(x);
    }
    return newBinMessage;
  }

  private String convertMessageToASCII(String binMessage) {
    //convert the string of binary codes back into ASCII

    //charMessage is the message after it has been converted from binary
to characters
    String charMessage = "";
    for (int x = 0; x < binMessage.length(); x += 8) {
        //get the next 8 bits to decode
        String nextByte = getNextByte(binMessage, x);
        //calculate the decimal value of the combined 8 bits
        int byteInt = Integer.parseInt(nextByte, 2);
        //convert decimal to ASCII character
        charMessage += (char) byteInt;
    }
    return charMessage;
  }

  private String getNextByte(String binMessage, int index) {
    //starting at the indexth bit in the binary string, take the next 8
    //bits to get the next byte to convert to ASCII

    //nextByte is the next 8 bits to convert to ASCII
    String nextByte = "";
    for (int x = index; x < index + 8; x++) {
        nextByte += binMessage.charAt(x);
    }
```

```
        return nextByte;
    }

    private int getByteFromColour(String colour) {
        //the GUI takes the byte the user wants to hide their message in as a
        //string containing the colour that the byte represents but the
        //computer has to work with the position that the byte is in each
pixel

        //changeByte is the byte position in each pixel that should be
changed (3, 2, 1)
        int changeByte;
        switch (colour) {
            case ("Red"):
                changeByte = 3;
                break;
            case ("Green"):
                changeByte = 2;
                break;
            case ("Blue"):
                changeByte = 1;
                break;
            default:
                changeByte = 0;
                break;
        }
        return changeByte;
    }

    public String getMessage() {
        return message;
    }

    public int getChangeByte() {
        return changeByte;
    }

}
```

ManageRandomAccess.java

```
package methods.utils;

import java.io.IOException;
import java.io.RandomAccessFile;

public class ManageRandomAccess {

        public byte[] readBytes(RandomAccessFile raFile, int noOfBytes)
throws IOException {
                // read the next "noOfBytes" bytes from the file

                // byte value is the byte array to read the next value from the
file
                // into
                byte[] value = new byte[4];
                raFile.read(value, 0, noOfBytes);
                return value;
        }

        public int convertLittleToInteger(byte[] value) {
                // requires 4 bytes in value
```

```
            // convert value to an integer from little endian
            int intValue =
java.nio.ByteBuffer.wrap(value).order(java.nio.ByteOrder.LITTLE_ENDIAN).get
Int();
            return intValue;
      }

}
```

StegCanvas.java

```
package gui.components;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.Robot;

import java.io.*;

import java.util.ArrayList;
import java.util.Arrays;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

public class StegCanvas extends javax.swing.JPanel {

      // name of the current image to be/being displayed
      private String fileName;
      // is there an image to paint on the canvas
      private boolean imageExists;
      // has the image finished being painted on the canvas
      private boolean imageLoaded;
      // stores each byte representing the pixels of the image for quicker
access
      private byte[] pixelArray;
      // stores changedByte, changedBit, pixelOffset, width, height,
pixelNo,
      // colourDepth, paddding
      private ArrayList<Integer> fileInfo;
      // stores the hexadecimal value of each pixel to make redrawing the
image
      // quicker
      private ArrayList<ArrayList<String>> colourArray = new
ArrayList<ArrayList<String>>();

      public StegCanvas() {
            this.fileName = "";
            this.fileInfo = null;
            this.imageLoaded = false;
            this.imageExists = false;
      }

      @Override
      public void paintComponent(java.awt.Graphics g) {
            // paints the image on the canvas if there is an image
            // paints the canvas white if there is no image
            super.paintComponent(g);
            // object used for drawing on the canvas
```

```
            java.awt.Graphics2D g2d = (java.awt.Graphics2D) g.create();

            if (imageExists) {
                  imageLoaded = false;
                  if (fileInfo == null) {
                        setUpNewImage();
                  } else {
                        // width in pixels of the image
                        int width = fileInfo.get(3);
                        // height in pixels of the image
                        int height = fileInfo.get(4);

                        // iterate through the colourArray and draw a pixel
for every colour value
                        for (int row = height - 1; row >= 0; row--) {
                              for (int col = 0; col < width; col++) {
                                    // get the hexadecimal colour of the
pixel to be drawn
                                    String colour = colourArray.get(height
- 1 - row).get(col);
                                    drawPixel(colour, row, col, g2d);
                              }
                        }
                        imageLoaded = true;
                  }
            } else {
                  // clear the canvas if there isn't an image to load
                  g2d.setColor(Color.white);
                  g2d.clearRect(0, 0, this.getWidth(), this.getHeight());
            }
      }

      public boolean checkIfValidFile(String fileName) {
            // checks that the bitmap file is valid
            // valid bitmap files start with the ASCII values for 'B' and
'M' (66 and 77)
            // isValid is false if the first two bytes do not indicate a
valid file

            boolean isValid = false;
            if (fileName.endsWith(".bmp")) {
                  try {
                        // read the first two bytes of the file
                        BufferedInputStream file = new
BufferedInputStream(new FileInputStream(fileName));
                        // byte array to store the first two bytes of the
file while they are checked
                        byte[] startBytes = new byte[2];
                        file.read(startBytes, 0, 2);
                        // if the read values are 66 and 77, the file is
valid
                        if (Arrays.toString(startBytes).equals("[66, 77]"))
{
                              isValid = true;
                        }
                        file.close();
                  } catch (IOException e) {
                  }
            }
            return isValid;
      }
```

```
	public void setUpNewImage() {
		// sets up the steg canvas to display a new image

		try {
			// input stream allows data to be read from files
			BufferedInputStream file = new BufferedInputStream(new
FileInputStream(fileName));
			// readManager provides functions for reading from the
file and converting from little endian
			methods.utils.ManageBufferedInput readManager = new
methods.utils.ManageBufferedInput();

			// get the fileInfo for the new image at fileName
			fileInfo = readFileInfo(file, readManager);
			// width in pixels of the image
			int width = fileInfo.get(3);
			// height in pixels of the image
			int height = fileInfo.get(4);
			// the number of bytes required after each row of pixels
in the file to make the row length a multiple of 4
			int padding = fileInfo.get(7);

			// resize the canvas so it will fit
			manageResizing(width, height);

			// set up the pixel array and colour array
			loadPixelArray(file, readManager);
			loadColourArray(width, padding, readManager);

			file.close();
		} catch (IOException e) {
		}

	}

	private ArrayList<Integer> readFileInfo(BufferedInputStream file,
methods.utils.ManageBufferedInput readManager) {
		// read in the required information to draw and process the
image from the bitmap
		// needs changedByte, changedBit, pixelOffset, width, height,
pixelNo, colourDepth, paddding

		fileInfo = new ArrayList<Integer>();

		// for an image hiding a message, the position of the byte and
the bit in the byte used
		// to hide the message is stored in a single byte near the
beginning of the file

		// get binary of value (starting at byte 6 in file, 1 byte
long)
		byte[] binChangedByteBit = readManager.getValue(file, 6, 1);
		// convert little endian binary to integer
		int changedByteBit =
readManager.convertLittleToInt(binChangedByteBit);
		// position of the byte is the first digit of changedByteBit
		int changedByte = changedByteBit / 10;
		// position of the bit is the second digit of changedByteBit
		int changedBit = changedByteBit % 10;
		// add to fileInfo ArrayList
```

```
            fileInfo.add(changedByte); // 0
            fileInfo.add(changedBit); // 1

            // get binary value of pixel offset
            byte[] binPixelO = readManager.getValue(file, 10, 4);
            // convert binPixelO to integer from little endian
            int pixelOffset = readManager.convertLittleToInt(binPixelO);
            fileInfo.add(pixelOffset); // 2

            // binary value of the width of the image
            byte[] binWidth = readManager.getValue(file, 18, 4);
            // convert binWidth to integer from little endian
            int width = readManager.convertLittleToInt(binWidth);
            fileInfo.add(width); // 3

            // binary value of the height of the image
            byte[] binHeight = readManager.getValue(file, 22, 4);
            // convert binHeight to integer from little endian
            int height = readManager.convertLittleToInt(binHeight);
            fileInfo.add(height); // 4

            // number of pixels in the image
            int pixelNo = width * height;
            fileInfo.add(pixelNo); // 5

            // binary value of the colour depth
            byte[] binColourD = readManager.getValue(file, 28, 2);
            // convert binColourD to integer from little endian
            int colourDepth = readManager.convertLittleToInt(binColourD);
            fileInfo.add(colourDepth); // 6

            // number of extra bytes required after each row to make row
length a multiple of 4
            int padding = calculatePadding(colourDepth, width);
            fileInfo.add(padding); // 7

            return fileInfo;

      }

      private int calculatePadding(int colourDepth, int width) {
            // the rows have to be made up of a number of bytes divisble by
4
            // if they're not, padding is used to make up the extra bytes

            // convert colour depth from bits to bytes
            int colourDepthBytes = colourDepth / 8;
            // padding is the number of bytes that will make up the paddig
            int padding;
            // get how many bytes over a multiple of 4 the row length is
            int modValue = (colourDepthBytes * width) % 4;
            if (modValue != 0) {
                  // padding makes up the difference between 4 and modValue
                  padding = 4 - modValue;
            } else {
                  // no padding if modValue is 0 as the row length is
already a
                  // multiple of 4
                  padding = 0;
            }
            return padding;
```

```
	}

	private void loadPixelArray(BufferedInputStream file,
methods.utils.ManageBufferedInput readManager) {
		// load the bytes of each pixel into an array to avoid reading
from the file every time
		// which byte position in the file the pixel value start at

		int pixelOffset = fileInfo.get(2);
		// width in pixels of the image
		int width = fileInfo.get(3);
		// height in pixels of the image
		int height = fileInfo.get(4);
		// colour depth of the bitmap
		int colourDepthBits = fileInfo.get(6);
		// how many extra bytes are at the end of each row
		int padding = fileInfo.get(7);

		// calculate number of bytes in the pixel array (inc any
padding)
		int arrayBufferSize = height * (width * (colourDepthBits / 8) +
padding);

		// pixel array stores the binary values of every pixel in the
image
		pixelArray = readManager.getValue(file, pixelOffset,
arrayBufferSize);
	}

	private void loadColourArray(int width, int padding,
methods.utils.ManageBufferedInput readManager) {
		// stores the hexadecimal representation of each pixel's colour
in a 2d array to make it quicker to repaint

		// length of the pixel array
		int pixelArrayLength = pixelArray.length;
		// x is a counter to keep track of how far through pixel array
it is
		int x = 0;
		// imageRow keeps track of which row number it is on
		int imageRow = 0;
		// imageCol keeps track of which column number it is on
		int imageCol = 0;
		// colourArray is a 2d arraylist to store the hexadecimal value
of
		// each pixel's colour
		colourArray.add(new ArrayList<String>());
		while (x <= (pixelArrayLength) - 3) {
			// presumes the colourDepth of the file is 24 bits
			// get the hexadecimal colour of the next pixel
			// pixelColour is the hexadecimal value to add to
colourArray
			String pixelColour = getHexadecimalColour(x,
readManager);
			// add to the 2d array
			colourArray.get(imageRow).add(pixelColour);
			imageCol++;
			x += 3;
			if (imageCol == width) {
				colourArray.add(new ArrayList<String>());
				imageCol = 0;
```

```
                        imageRow++;
                        x += padding;
                    }
                }
            }

        private String getHexadecimalColour(int byteCount,
methods.utils.ManageBufferedInput readManager) {
            // get the value of each colour channel (blue, green, red),
            // convert each to hexadecimal,
            // construct as one string in order RGB to paint the canvas
with

            // channel stores the next byte of the pixel (has to be length
4 to be able to convert to integer from little endian)
            byte[] channel = new byte[4];
            channel[0] = pixelArray[byteCount];
            // get hexadecimal value for each colour channel of the current
pixels
            // blueChannel is the hexadecimal value for the amount of blue
in the pixel's colour
            String blueChannel = convertByteToHexadecimal(channel,
readManager);
            byteCount++;
            channel[0] = pixelArray[byteCount];
            // greenChannel is the hexadecimal value for the amount of
green in the pixel's colour
            String greenChannel = convertByteToHexadecimal(channel,
readManager);
            byteCount++;
            channel[0] = pixelArray[byteCount];
            // redChannel is the hexadecimal value for the amount of red in
the
            // pixel's colour
            String redChannel = convertByteToHexadecimal(channel,
readManager);
            byteCount++;
            // colour is the concatenation of redChannel, greenChannel and
            // blueChannel to make a valid hexadecimal colour
            String colour = "#" + redChannel + greenChannel + blueChannel;

            return colour;
        }

        private String convertByteToHexadecimal(byte[] value,
methods.utils.ManageBufferedInput readManager) {
            // converts the byte array to hexadecimal
            // presumes it has been given an array of length 4

            String hexValue = String.format("%02X",
readManager.convertLittleToInt(value));
            return hexValue;
        }

        private void drawPixel(String colour, int imageRow, int imageCol,
java.awt.Graphics2D g2d) {
            // set the colour of the next pixel
            // draw the pixel as a line of length 1

            g2d.setColor(java.awt.Color.decode(colour));
            g2d.drawLine(imageCol, imageRow, imageCol + 1, imageRow);
```

```
	}

	public void manageResizing(int width, int height) {
		// resize the canvas to fit the image
		// there are no limits to the resizing of the canvas -
MainWindow
		// contains
		// the canvas in a scrollpane to manage this

		// canvasDimension is the required size of the canvas to fit
the image
		Dimension canvasDimension = new Dimension(width + 2, height +
2);

		// window is the GUI window, required for redrawing it after
resizing
		JFrame window = (JFrame)
(SwingUtilities.getWindowAncestor(this));

		// resize the parent
		this.getParent().setPreferredSize(canvasDimension);
		this.getParent().revalidate();
		// resize canvas
		this.setPreferredSize(canvasDimension);
		this.repaint();
		window.pack();

	}

	public String getPixelColour(int x, int y) {
		// gets the colour of the pixel at the cursor (hexadecimal)

		// pixelColour stores the hexadecimal value of the pixel at x,
y on the screen
		String pixelColour = "";
		try {
			// create a rectangle of 1x1 at (x, y)
			Rectangle rect = new Rectangle(x, y, 1, 1);
			// take screenshot of contents of rectangle
			java.awt.image.BufferedImage image = new
Robot().createScreenCapture(rect);
			// get colour at (0, 0) in screenshot - convert
java.awt.Color class to hex- no simple method to do this
			pixelColour = "" + Integer.toHexString(image.getRGB(0, 0)
& 0xFFFFFF).toUpperCase();
			// display on screen
			while (pixelColour.length() < 6) {
				pixelColour = "0" + pixelColour;
			}

		} catch (java.awt.AWTException e) {
		}
		return pixelColour;
	}

	public void updateZoom(int x, int y, ZoomCanvas zoomCanvas) {
		// updates the connected to zoom canvas to show a zoomed in
version
		// of the area around the mouse

		int squareSize = 10; // for 20x20 square
```

```
            try {
                  // create square with (x, y) as center (around the mouse
point)
                  Rectangle square = new Rectangle(x - squareSize, y -
squareSize, squareSize * 2, squareSize * 2);
                  java.awt.image.BufferedImage image = new
Robot().createScreenCapture(square);
                  // image will be made to fit the zoom canvas so it will
be zoomed in
                  zoomCanvas.setImage(image);
                  zoomCanvas.setImageExists(true);
                  zoomCanvas.repaint();
            } catch (java.awt.AWTException e) {
            }
      }

      public String getFileName() {
            return fileName;
      }

      public boolean imageLoaded() {
            return imageLoaded;
      }

      public ArrayList<Integer> getFileInfo() {
            return fileInfo;
      }

      public byte[] getPixelArray() {
            return pixelArray;
      }

      public void resetValues() {
            // clear the values to load a new image
            fileInfo = null;
            pixelArray = new byte[0];
            colourArray.clear();
      }

      public void setFileName(String fileName) {
            this.fileName = fileName;
      }

      public boolean imageExists() {
            return imageExists;
      }

      public void setImageExists(boolean imageExists) {
            this.imageExists = imageExists;
      }
}
```

ManageBufferedInput.java

```
package methods.utils;

import java.io.*;

public class ManageBufferedInput {
      // currentByte keeps track of where in the file the pointer currently
is to
```

```
	// correctly work out how many bytes might need to be skipped to get
to the right position to read the next value
	private int currentByte;

	public ManageBufferedInput() {
		this.currentByte = 0;
	}

	public byte[] getValue(BufferedInputStream file, int offset, int
noOfBytes) {
		// gets the specified number of bytes (noOfBytes) from the
given file starting at offset

		// offset parameter is the starting byte of the required value
in the file
		// offset of file.read is where to start writing the value in
the
		// bytearray

		// to be converted from little endian, bytearrays have to be 4
bytes
		// byte array to read the next value from the file into
		byte[] value;
		if (noOfBytes >= 4) {
			value = new byte[noOfBytes];
		} else {
			value = new byte[4];
		}
		// skipBytes is the number of bytes that need to be skipped
over from the current position to get to the required offset
		int skipBytes = calculateSkipBytes(offset);
		try {
			file.skip(skipBytes);
			file.read(value, 0, noOfBytes);
			setCurrentByte(noOfBytes + skipBytes);
		} catch (IOException e) {
		}
		return value;
	}

	private int calculateSkipBytes(int offset) {
		// offset is starting byte of value to read
		// presumes the byte to read is further on than the current
position
		// (can't go backwards through the file)

		// skipBytes is the number of bytes that need to be skipped
over from the current position to get to the required offset
		int skipBytes = offset - currentByte;
		return skipBytes;
	}

	public int convertLittleToInt(byte[] value) {
		// converts a byte array of four bytes from a little endian
value to a integer
		// intValue is the integer value of the given bytearray
		int intValue =
java.nio.ByteBuffer.wrap(value).order(java.nio.ByteOrder.LITTLE_ENDIAN).get
Int();
		return intValue;
	}
```

```
	private void setCurrentByte(int extraBytes) {
		currentByte += extraBytes;
	}
}
```

ZoomCanvas.java

```
package gui.components;

public class ZoomCanvas extends javax.swing.JPanel {

	// displays a zoomed in version of the area around the cursor
	// image to display
	private java.awt.image.BufferedImage image;
	// does an image exist to display
	private boolean imageExists;
	// preferred size for the canvas (length of one side)
	// height and width should be the same size as the image is always
square
	private int canvasSize;

	public ZoomCanvas() {
		this.canvasSize = 200;
		this.imageExists = false;

	}

	@Override
	public void paintComponent(java.awt.Graphics g) {
		// draws the zoomed-in area of the image

		super.paintComponent(g);
		if (imageExists) {
			// display the image
			g.drawImage(image, 0, 0, canvasSize, canvasSize, null);
		}
	}

	public void setImage(java.awt.image.BufferedImage image) {
		this.image = image;
	}

	public void setImageExists(boolean imageExists) {
		this.imageExists = imageExists;
	}
}
```

RSA.java

```
package methods.methods;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Collections;

public class RSA {

	// private key = d, N
	// public key = N, e
	// primeList is a list of prime numbers which is used to randomly
generate
```

```
	// p and q
	private ArrayList<Integer> primeList;
	// p and q are two random prime numbers to generate the public and
private
	// key with
	private int p;
	private int q;
	// N makes up half of the public and private key (is p*q)
	private int N;
	// phi is the totient of N (equal to (p-1)*(q-1))
	private int phi;
	// e and d make up the other halves of the public and private key
	// respectively
	private int e;
	private int d;
	// store N and e in an array to make the public key easier to pass
around
	// the system
	private Integer[] publicKey = new Integer[2];

	public RSA() {
		// generate the public and private key values
		primeList = generatePrimeList();
		p = getRandomElementFromArray(primeList);
		q = p;
		while (q == p) {
			q = getRandomElementFromArray(primeList);
		}
		phi = (p - 1) * (q - 1);
		N = p * q;
		// e and d are generated and returned together as an array
		int[] edArray = generateED();
		e = edArray[0];
		d = edArray[1];
		publicKey[0] = N;
		publicKey[1] = e;
	}

	private ArrayList<Integer> generatePrimeList() {
		// iterates through each integer in a given range (restricted
to <=560)
		// and uses Fermat's Little Theorem to identify primes and add
them to an array
		// !! watch for Carmichael with Fermat's Little Theorem and
		// numbers > 560
		ArrayList<Integer> primeList = new ArrayList<>();
		// don't add 2 because it allows 6 to be created for N which is
too
		// small for encryption
		// start on 3, limit of 561 because of Carmichael numbers,
		// += 2 because even numbers aren't prime after 2
		for (int x = 3; x < 561; x += 2) {
			if (checkIfPrime(x)) {
				primeList.add(x);
			}
		}
		return primeList;

	}

	private boolean checkIfPrime(int p) {
```

```
            // using Fermat's Little Theorem
            // returns true if prime, false if not

            // BigInteger version of p - used for modding with BigIntegers
            BigInteger biP = new BigInteger("" + p);
            for (int x = 2; x < p - 1; x++) {
                  // for larger values of x and p, x^p is too big to store
in an
                  // Integer so use BigInteger instead
                  BigInteger biX = new BigInteger("" + x);
                  // if (x^p)-x is divisible by p then p is prime
                  // result of (x^p)-x % p (0 if p is prime)
                  BigInteger fLTResult =
((biX.pow(p)).subtract(biX)).mod(biP);
                  if (!(fLTResult.toString().equals("0"))) {
                        return false;
                  }
            }
            // if it's not returned false then FLT says p is prime although
there
            // are composite numbers that pass as prime with this
(Carmichael
            // numbers, starting with 561)

            // so, presuming it is not Carmichael:
            return true;
      }

      private ArrayList<int[]> generateKFactors() {
            // K is a number such that K mod phi = 1
            // stores a calculated value while it is checked for being a
valid K
            int tempK = 0;
            // factorList is a list of factors of the generated value of K
            ArrayList<int[]> factorList = new ArrayList<>();
            while (factorList.size() < 1) {
                  // randomly check possible values of K less than or equal
to
                  // (phi*6)+1
                  // multiplier is how much to multiply phi by to generate
K
                  int multiplier = (int) (Math.random() * 5);
                  // tempK is the value of K being tested
                  tempK = (phi * multiplier) + 1;
                  for (int x = 2; x <= Math.sqrt(tempK); x++) {
                        if (tempK % x == 0) {
                              // factorPair is two numbers that times
together to produce K
                              int[] factorPair = { x, (int) (tempK / x) };
                              factorList.add(factorPair);
                        }
                  }
            }
            return factorList;
      }


      private int[] generateED() {
            // e and d have to be factors of K and coprime to phi
            // list of all factor pairs of K
            ArrayList<int[]> KFactorList = generateKFactors();
```

```
            int noOfFactorPairs = KFactorList.size();
            for (int x = 0; x < noOfFactorPairs; x++) {
                  // the current factor pair being checked for being
coprime to phi
                  int[] factorPair = KFactorList.get(x);
                  // euclid's algorithm returns the gcd of 2 numbers
                  // this is 1 for coprimes
                  if ((euclidsAlgorithm(factorPair[0], phi) == 1) &
(euclidsAlgorithm(factorPair[1], phi) == 1)) {
                        return factorPair;
                  }
            }
            // if the loop ends and this function has not returned,
            int[] errorArray = { 0, 0 };
            return errorArray;
      }

      private int euclidsAlgorithm(int a, int b) {
            // find the greatest common divisor of a and b (recursive)

            // base statements
            if (a == 0) {
                  return b;
            }
            if (b == 0) {
                  return a;
            }
            // a = b.q + r where q is any coefficent and r is the remainder
of a/b
            int r = a % b;
            // repeat with b and r
            return euclidsAlgorithm(b, r);
      }

      private int getRandomElementFromArray(ArrayList<Integer> list) {
            // get element randomly from the list of possible values
            Collections.shuffle(list);
            // Math.random() produces a number between 0 and 1
            // times Math.random to get value between 0 and eList length to
index
            // x is a random number to use an index in the given arrayList
            int x = (int) (Math.random() * (list.size() - 1));
            return list.get(x);
      }

      public String encryptMessage(String message, int e, int N) {
            // encrypt the message using e and N as the public key
            // stores the produced ciphertext
            String encryptedMessage = "";
            // get a string of the integer ascii codes of each char in the
message
            String asciiMessage = convertAsciiStrToAsciiDec(message);
            // get how long each encrypted value has to be
            int encryptedLength = getLengthOfInteger(N);
            for (int x = 0; x < asciiMessage.length(); x++) {
                  // get the next value to encrypt
                  int valueToEncrypt = Integer.parseInt("" +
asciiMessage.charAt(x));
                  // encrypt the value and make it the right length
                  String valueToAdd = String.format(("%0" + encryptedLength
+ "d"), rsa(valueToEncrypt, e, N));
```

```
                    encryptedMessage += valueToAdd;
              }
              return encryptedMessage;
        }

        public boolean checkInputValueForNumbers(String[] inputValues) {
              // check that the values obtained from an input are parseable
to
              // integers
              // uses a String[] so that the public key can be passed all at
once
              // but the function will work with a single element in the
array

              // isValid is true if the values are parseable to integers
              boolean isValid = true;
              // iterate through each element in the inputValues parameter
              for (String value : inputValues) {
                    if (!value.equals("")) {
                          for (int x = 0; x < value.length(); x++) {
                                // the ascii code of the next value in the
string
                                int asciiCode = (int) (value.charAt(x));
                                // numbers are between 48 and 57 inclusive in
ASCII
                                if ((48 > asciiCode) || (asciiCode > 57)) {
                                      isValid = false;
                                      return isValid;
                                }
                          }
                    } else {
                          isValid = false;
                          return isValid;
                    }
              }
              return isValid;
        }

        public boolean checkMessageForNumbers(String inputMessage) {
              // creates a String[] containing the inputMessage to send to
              // checkInputValueForNumbers
              // to check the message only contains numbers

              // array containing the inputMessage
              String[] messageArray = new String[1];
              messageArray[0] = inputMessage;
              return checkInputValueForNumbers(messageArray);
        }

        private String convertAsciiStrToAsciiDec(String inputMessage) {
              // convert a string of ascii characters into their integer
values
              // and forces each value to be 3 digits long
              // presumes message is only ascii- this will be checked in the
main program
              // string to store the integer ascii values of all the
characters in the
              // message
              String asciiMessage = "";
              for (int x = 0; x < inputMessage.length(); x++) {
                    // convert the next character to its ascii integer code
```

```
                  String valueToAdd = String.format("%03d", (int)
(inputMessage.charAt(x)));
                  asciiMessage += valueToAdd;
            }
            return asciiMessage;
      }

      private String convertAsciiDecToAsciiStr(String inputMessage) {
            // convert a string of numbers into ascii characters
            // presumes inputMessage only contains numbers

            // string of characters converted from integers
            String asciiMessage = "";
            for (int z = 0; z < inputMessage.length(); z += 3) {
                  // convert the next value to a character
                  char nextChar = (char)
(Integer.parseInt(getNextValue(inputMessage, z, 3)));
                  asciiMessage += nextChar;
            }
            return asciiMessage;
      }

      private int getLengthOfInteger(int number) {
            // largest value produced by rsa is N-1 so the largest length
of value
            // produced by rsa is N.length- need to know how long each
encrypted
            // chunk is to decrypt it so force them all to be N.length to
make it
            // easier - also used to check that chunks are the right length

            // number of digits in the given number
            int valueLength = ("" + number).length();
            return valueLength;
      }


      private String getNextValue(String message, int startIndex, int
length) {
            // get the chunk of string to en/decrypt
            // length is the length of chunk required, startIndex is the
position
            // in the message to start at
            // nextValue is the next part of message to en/decrypt
            String nextValue = "";
            for (int i = 0; i < length; i++) {
                  try {
                        nextValue += message.charAt(startIndex + i);
                  } catch (StringIndexOutOfBoundsException e) {
                        // if there isn't enough characters left in the
string, return
                        // what there is
                        return nextValue;
                  }
            }
            return nextValue;
      }

      public String decryptMessage(String message) {
            // decrypts message by taking a chunk of length encryptedLength
at a
```

```
            // time and using the rsa function with the private key length
of the
            // next chunk to decrypt
            int encryptedLength = getLengthOfInteger(N);
            // stores the decrypted values of the message
            String decryptedMessage = "";
            // stores the last chunk if it's not long enough to decrypt
properly
            String endString = "";
            // next value to decrypt
            String nextValue;
            for (int x = 0; x < message.length(); x += encryptedLength) {
                  nextValue = getNextValue(message, x, encryptedLength);
                  if (nextValue.length() == encryptedLength) {
                        // integer version of nextValue
                        int valueToDecrypt = Integer.parseInt(nextValue);
                        // plaintext of nextValue
                        String decrypted = "" + rsa(valueToDecrypt, d, N);
                        // add the newly calculated plaintext to the rest
of the
                        // plaintext
                        decryptedMessage += decrypted;
                  } else {
                        endString = "" + nextValue;
                  }
            }
            // convert mesage to ascii and add any leftovers to the end as
these
            // can't be converted to ascii
            // plaintext message in characters
            String asciiMessage =
convertAsciiDecToAsciiStr(decryptedMessage);
            asciiMessage += endString;
            return asciiMessage;
      }

      private int rsa(int value, int exp, int N) {
            // V = (value^exp) mod N to get the plain/ciphertext of value
            // uses BigInteger as some of the values produced by value^exp
can be be
            // too big for int
            // BigInteger version of value
            BigInteger biValue = new BigInteger("" + value);
            // BigInteger version of exp
            BigInteger biExp = new BigInteger("" + exp);
            // BigInteger version of N
            BigInteger biN = new BigInteger("" + N);
            // result of the calculation (plain or ciphertext of value)
            BigInteger result = biValue.modPow(biExp, biN);

            return Integer.parseInt(result.toString());
      }

      public int getN() {
            return N;
      }

      public int getE() {
            return e;
      }
```

```
        public int getD() {
                return d;
        }

        public Integer[] getPublicKey() {
                return publicKey;
        }
}
```

Appraisal

1. Comparison of Project Performance Against Objectives

Objective 1.1: Allow the user to switch between encrypting and decrypting with the Caesar cipher
This objective has been met. The user can choose whether to encrypt or decrypt a message using the shift they have input, by selecting the option from a pair of radio buttons.

Objective 1.2: Allow the user to enter a specific shift
This objective has also been met; the user can enter a shift using a spinner box. The Caesar wheel display is also updated to show the setting.

Objective 1.3: Allow the user to enter a message to encrypt
This objective has been met. The user can type a message directly into the box for encrypting or they can either load the contents of a text file as the message or choose to use an incoming message from a TCP/IP connection as the message to encrypt.

Objective 1.4: Allow the user to enter a message to decrypt
This objective has also been met. The user can input a message for decryption in exactly the same way as for encryption.

Objective 1.5: Demonstrate visually how the cipher is working by means of an animation.
This objective has been met- the system uses a Caesar wheel to demonstrate the workings of the cipher. During encryption and decryption, the current letter and the result of encrypting or decrypting it are highlighted on the wheel.

Objective 2.1: Allow the user to switch between encrypting with the Vernam cipher and decrypting with the Vernam cipher
This objective has been met. The user can switch between encryption and decryption in the same way as they can with the Caesar cipher- selecting the option from two radio buttons.

Objective 2.2: Allow the user to enter a key of their choice
This objective has been met. The user can enter their own key. They are also warned when their key is shorter than the message to encrypt, because if the key is repeated when encrypting then the encryption is not perfect.

Objective 2.3: Allow the user to enter a message to encrypt
This objective has been met. The user can choose a method in three different ways, the same as with the Caesar cipher; enter a message manually, load up the contents of a text file, or use an incoming message from TCP/IP.

Objective 2.4: Allow the user to enter a message to decrypt
This objective has also been met. The user can enter a message for decrypting using any of the same three ways available for entering a message for encrypting.

Objective 2.5: Demonstrate on screen, how the cipher is working, by showing it working through each letter of the message
This objective has been met. The demonstration shows the user the XOR calculation for each letter in the message and the result of this. Every calculation made in the current operation is displayed in a box at the bottom of the screen for the user to look through.

Objective 3.1: Generate public and private key
This objective has been met. A new public and private key pair is automatically generated when the program is run. The generation algorithm produces most of the required values from two prime numbers randomly selected from a list of prime numbers, generated using Fermat's Little Theorem. The algorithm also has to calculate the greatest common divisor between several different pairs of numbers, for which it uses Euclid's Algorithm.

Objective 3.2: Encrypt using public key
This objective has been met. The user can either enter the public key to use, or, if a TCP/IP connection has been initiated, they can use the public key of their partner.

Objective 3.3: Decrypt using private key
This objective has also been met. The system automatically uses the private key of the user to decrypt the message, so decrypting a message encrypted for someone else will not be successful.

Objective 3.4: Be able to send the public key to another computer across the network
This objective has been met. When the user makes a connection with another user, the public keys of each are automatically exchanged. On the RSA screen, the input boxes for the public key to use are automatically filled with the public key of the user's TCP/IP connection partner.

Objective 4.1: Allow the user to load up a chosen bitmap image
This objective has been met. The user can choose which bitmapped image to use for Steganography. When the image is initially read in and drawn, the colour value of each pixel is stored in an array in the program, as when the image is redrawn, reading the values from the array is quicker than reading them straight from the file again.

Objective 4.2: Allow the user to enter a message to hide in the image
This objective has been met. The user can enter a message to hide, and also specify the position in each pixel's value to use to hide it.

Objective 4.3: Retrieve a hidden message from an image
This objective has been met. The user can request for the program to find a message hidden in the file. If a message is found, this is displayed onscreen, but if there is no message, the user is notified.

Objective 4.4: Provide a zoom function which allows the user to zoom into the pixels to inspect
This objective has been met. In the window displaying the image, there is another display that shows a zoomed in portion of the image. This portion is a specified area currently around the mouse, and is updated when the mouse is moved around the image.

Objective 4.5: Compare the old image with the new image
This objective has also been met. When the user hides a message in an image, a copy of the image is created, and the message is hidden in this, leaving the original as it is, allowing for the two versions to be compared. After the message has been hidden, the new image is automatically loaded up at the side of the original, allowing for the two to be compared.

Objective 4.6: Show the hexadecimal values of pixel colours

This objective has been met. The colour of the pixel at the current mouse position is displayed onscreen in hexadecimal and this value updates whenever the mouse moves.

Objective 5.1: Have a save function to enable the user to save a text file containing their encrypted/decrypted messages
This objective has been met. The user can choose to save the result of their most recent encryption or decryption to a text file.

Objective 5.2: Have a save function to enable the user to save a bitmap file containing a hidden message- as a new file rather than over the original
This objective has been met. When the user wants to hide a message, they are prompted to enter a name for the new bitmap file that will contain the hidden message. The name of the original file is not a valid name for this, as it would overwrite the original file.

Objective 5.3: Have a load function to enable the user to load a text file containing an encrypted message.
This objective has been met. The user can choose to load the message to use for decryption from a text file, and the input box for the message to use will automatically be filled in with the message.

Objective 5.4: Have a load function to enable the user to load a bitmap file either containing a message or to hide a message in it.
This objective has been met. The user can choose which bitmapped image to use for Steganography, either hiding a message or finding a message.

Objective 5.5: Allow the user to send messages to someone else across the network using TCP/IP
This objective has been met. The user can connect with another user using TCP/IP and send the results of their most recent encryption or decryption to them as a message or a text file. The other user can receive these and choose whether to load them up as the input message for their next encryption or decryption.

2. User Feedback

I have obtained feedback from both Mr. M and a group of students who are currently studying the new A Level Specification.

Mr. M (discussion):
Mr. M was pleased to see that I had managed to incorporate all of the encryption we had discussed and also include TCP/IP, which had been an extension idea for the project.

For the Vernam Cipher, he commented that the layout was clear. He also said that it was useful to see the Baudot codes, so that a student can see the codes being used in the XOR process. He also commented that it was good that the Working box scrolls for longer messages, although he suggested that it would be nice if the box could display the words “XOR” and “EQUALS” to the left of the binary patterns. He also liked the fact that it displays a warning for short keys as it emphasises an important point about the cipher.

With the Caesar Cipher, he commented that the consistency with the Vernam Cipher interface was good. He found the animation very clear and said that students should easily be able to see how the cipher works.

He was impressed with the Steganography for the range of options available, including selecting the byte and bit to hide the message in. He was pleased to see that the validation to hide a message without properly setting all options was thorough. He commented that the facility to compare images and zoom in will enable students to see how the message has been hidden and the effect of changing the bit and byte used to hide it- the zoom feature will be very helpful with this. He suggested that on the screen displaying the two images that there should be some way to identify which contains the message and which doesn't. He accepted that this could be done with the system as it stands, since the file names are displayed above each image, but suggested that a clearer way of flagging it might be useful.

He was impressed that the RSA works as the maths behind it is complex. He said that Asymmetric Encryption was a topic that students find hard so it will be useful to have a tool to demonstrate it with. He appreciated that RSA is harder to demonstrate because the maths is likely to be beyond a lot of students but suggested that it would be helpful if some of the maths used to generate the keys could be viewed by students who wanted to, but remained hidden from most.
He commented that, for this encryption method, the implementation of the TCP/IP connection is particularly helpful, as it enables the automatic exchange of public and private keys, which conveys the benefit of an asymmetric encryption. He also said it was great to see the messages being transmitted across the network because this puts the topic into a real context, and makes more obvious the fact that one key can be used for encryption and a matched key for decryption.

With the TCP/IP, he found the setup of a connection straightforward. He commented that it was nice that they have to input the IP address to connect to, as this is another opportunity to reference another part of the syllabus. He also said that the validation worked well. He suggested two things – having a send button on the form itself and for the system to allow more than two people connect together, and have a repository for the RSA keys in this situation.

In a final comment, he said that I had made a nice job of it and that he can see that it is going to be genuinely useful to demonstrate techniques next year.

<u>Analysis of Feedback:</u>
Mr. M was clearly happy with the layout and contents of the Vernam Cipher interface. His suggestion about adding "XOR" and "EQUALS" to the Vernam Cipher Working box are similar to an idea I had for improving the interface, which was to add labels next to each row to identify what each one showed. However, with the current system, the students have to know that the Vernam Cipher works by XOR-ing two binary patterns together so Mr. M's suggestion would be more useful to the students and also more effective in demonstrating how the cipher works.

Mr. M was also very happy with the Caesar Cipher interface, because this is laid out to match that of the Vernam Cipher to make the system consistent and easier to work with.

From the discussion, it is clear that the window for displaying the images for Steganography is not as clearly laid out as intended. Mr. M's suggestion for adding labels to the window to identify which image contains the hidden message and which doesn't is a good idea that would improve this, since, unless the students are familiar with the names of the files involved, it can be difficult to determine which is which. It could be argued that when the two images

are displayed side by side, the one on the left is always the one with the message hidden it but this could be made more sensible anyway, as when an image is initially loaded, it is loaded on the left, so when a message is hidden it, the copy could loaded on the right, instead of moving the original to the right and loading the copy on the left, as the system currently does.

Mr. M was also impressed by the fact that the RSA encryption worked, especially since the maths behind the key generation is quite difficult. His idea for having the option for students to view the maths behind RSA, although a nice idea, doesn't really seem essential for improving the system for its purpose (as the students are not required to know how specific examples of asymmetric encryption work and seeing the maths is unlikely to benefit students).

He liked the fact that the connection for the TCP/IP was straightforward and gave students the opportunity to input the IP address of who they wanted to connect with, as this covered even more of the syllabus. Mr. M suggested adding a send button to the form, instead of having the option only available from the toolbar, which would make it easier to send messages and files across the connection. He also suggested making it possible for more than two people to connect together at the same time; this could work but it may also make the connection more complicated and harder for the students to use.

Students (questionnaire):
The questionnaires that the students filled in are in Appendix 2.1

Analysis of Feedback:
The overall feedback from the students was very positive. While some students didn't have time to try everything and finish the questionnaire, their comments about what they did get to use were encouraging.

All students agreed that the Vernam Cipher and the Caesar Cipher interfaces were straightforward to use. One student commented that the layout of the Vernam Cipher interface was good and that the Caesar Cipher interface was presented in an effective way. The Caesar Cipher's animation was also clearly a success, as two specifically commented that this had helped them understand the cipher.

Steganography didn't seem to be as successful in helping the students with bitmapped graphics, as a couple of students seemed less than certain that they had a better understanding afterwards.

The RSA had mixed responses from those who had chance to use it in the time they had with the system. Some of the students felt that they did have a better understanding of asymmetric encryption and that the interface was easy enough to understand how to use, but one student felt the complete opposite, although this could be due to the fact that the students are currently studying AS Level and haven't covered RSA or Asymmetric Encryption in lessons beyond a brief overview before using the program.

Almost all of the students who looked at the TCP/IP reported that it was straightforward to set up a connection; one student managed to conclude this from just looking at the interface as he didn't actually connect with anyone.

One student made the suggestion that there could be an option to view instructions added to the interface. He didn't explain what instructions he would want, but it could be presumed

that he meant instructions for using the system since the idea formed part of his comments on the ease of using the interfaces. Another idea for this could be instructions on using the methods generally, independent of the system. In either or both cases, it would be a good idea as the students would have quicker access to any information they might require.

Feedback authenticated by Mr. M: _____

3. Possible Extensions

It would be useful for the Working box in the Vernam Cipher interface to have labels and/or the words "XOR" and "EQUALS" (as suggested by Mr. M in his feedback) to make it clear and easier to understand.

For encrypting and decrypting with the Vernam cipher, the students could have the option of using the ASCII codes of characters over using the Baudot codes- while the Baudot codes are useful for teaching the concept of the Vernam cipher and were the codes that were originally used with it, it would be useful for the students to have experience with ASCII codes being used as they will be expected to use these in the exam and not the Baudot codes.

In both the Vernam cipher and the Caesar cipher, there could be an option to encrypt or decrypt without the animation. When students have become familiar with the ciphers, they may want to use the system as a way of checking their own encrypting, but with the current system, they would have to wait through the entire animation for the full result to be displayed.

Mr. M's suggestion to have clearer identifiers of which image is which in the Steganography interface would be useful as it would make the interface easier to use and understand. As the system stands, the file names are displayed with the images but it is not clear from this which image contains the message and which doesn't.

The program could allow the user to choose whether to use their private key or an input private key for RSA decryption- this would allow the students to see that a private key can only decrypt messages encrypted with the matching public key.

It would be useful for the user to have a way of viewing all received data from a TCP/IP connection – at the moment, if the user chooses not to use a message that they have received, they have no way of returning to it and using it later.

It would be easier for the users if there was a way of breaking TCP/IP connections manually – there is currently no way for users to break a connection unless one shuts down the program, which is not always ideal – for example, shutting down the program and restarting generates new keys.

Mr. M suggested that a button for sending messages and files across a TCP/IP connection could be placed on the forms instead of having to use the menu on the toolbar. This would make using the system quicker and easier to use, and the same could also be done with a button for receiving messages and files.

One of the students suggested that an Instructions section would be a useful addition to the system. These instructions could be general information on using the ciphers involved or more specific instructions for using the system. Both would be useful to the students as it would make it quicker for them to get any help they need with the topics or system.

4. Conclusion of Project

It is clear from the feedback from Mr. M and the students that the system will be useful in lessons, for each of the topics involved (Encryption, Bitmapped Graphics and Asymmetric Encryption).

There are some minor changes that could be made to the program to make it easier to use and understand; these are feasible and should be simple enough changes to make.

There are also some ideas that can be added to the system to extend the system's abilities and how far it will be useful to the students.

Appendix 1

1. The new specification requires students to be able to encrypt and decrypt messages in both the Caesar and the Vernam cipher and also be able to compare the two.

4.5.6.10 Encryption

Content	Additional information
Understand what is meant by encryption and be able to define it.	Students should be familiar with the terms cipher, plaintext and ciphertext. Caesar and Vernam ciphers are at opposite extremes. One offers perfect security, the other doesn't. Between these two types are ciphers that are computationally secure – see below. Students will be assessed on the two types. Ciphers other than Caesar may be used to assess students' understanding of the principles involved. These will be explained and be similar in terms of computational complexity.
Be familiar with Caesar cipher and be able to apply it to encrypt a plaintext message and decrypt a ciphertext. Be able to explain why it is easily cracked.	
Be familiar with Vernam cipher or one-time pad and be able to apply it to encrypt a plaintext message and decrypt a ciphertext. Explain why Vernam cipher is considered as a cypher with perfect security.	Since the key ***k*** is chosen uniformly at random, the ciphertext *c* is also distributed uniformly. The key ***k*** must be used once only. The key ***k*** is known as a one-time pad.
Compare Vernam cipher with ciphers that depend on computational security.	Vernam cipher is the only one to have been mathematically proved to be completely secure. The worth of all other ciphers ever devised is based on computational security. In theory, every cryptographic algorithm except for Vernam cipher can be broken, given enough ciphertext and time.

2. Public/Private Key Encryption appears on both the new specification and the old specification. Students are only required to how asymmetric encryption works but a specific encryption method will be used to demonstrate the concept.

4.9.3.2 Internet security

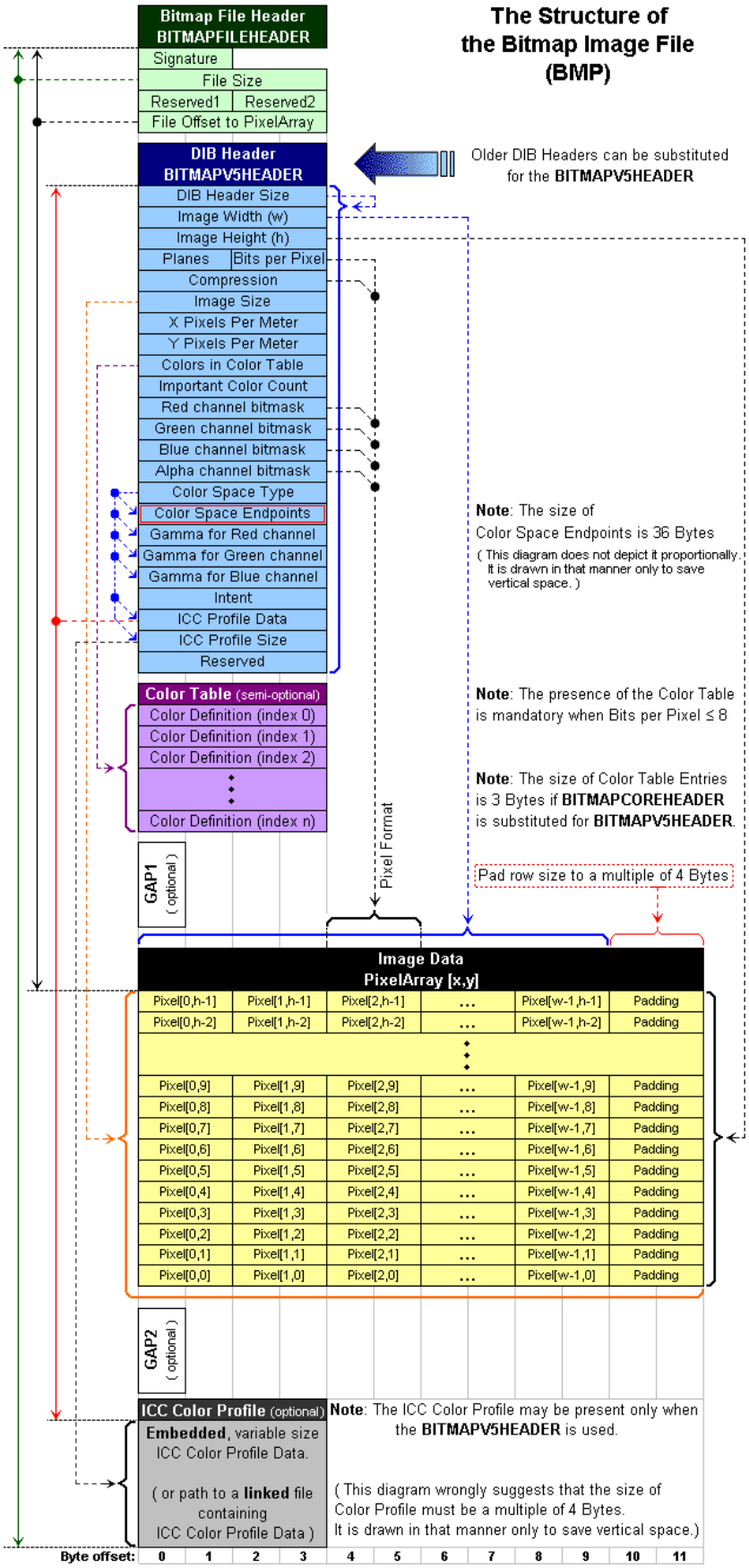
Content	Additional information
Understand how a firewall works (packet filtering, proxy server, stateful inspection).	
Explain symmetric and asymmetric (private/public key) encryption and key exchange.	
Explain how digital certificates and digital signatures are obtained and used.	
Discuss worms, trojans and viruses, and the vulnerabilities that they exploit.	
Discuss how improved code quality, monitoring and protection can be used to address worms, trojans and viruses.	

3. The new specification includes bitmapped graphics. These also featured in the old specification and the requirements have not changed.

4.5.6.4 Bitmapped graphics

Content	Additional information
Explain how bitmaps are represented.	
Explain the following for bitmaps: • resolution • colour depth • size in pixels.	Resolution of an image is expressed directly as *width of image in pixels* by *height of image in pixels* using notation *width* x *height*. Alternatively, resolution can be expressed in *number of dots per inch* where a dot is a pixel. *colour depth = number of bits stored for each pixel*. *resolution in pixels = width in pixels* x *height in pixels*.
Calculate storage requirements for bitmapped images and be aware that bitmap image files may also contain metadata.	Ignoring metadata, *storage requirements = resolution x colour depth* where resolution is expressed in *width in pixels* x *height in pixels*.
Be familiar with typical metadata.	eg width, height, colour depth.

4. Bitmap files contain several bytes of meta data such as the width and height of the image and the number of bits used to represent each pixel later in the file.



5. The Computing department have their own website where they put the notes for each topic on the specification, which are available to the students, for example, this shows the notes on

bitmapped images. While these were written for the old specification, the requirements have not changed, so they can still be used in lessons.

BGSB Computing Department Website

1st Form
2nd Form
3rd Form
GCSE ICT
GCSE Comp
AS
A2

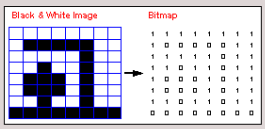
Robot Club

Computer Room Rules

Home

COMP1: Images and Sound

1. Complex types of data such as images and sound occupy a large amount of memory when they are stored. There are many different methods that can be used to store these types of data.
2. The two most common systems for representing graphics are the bit-map and vector systems.
3. Using the bit-map system, an image is composed of a large number of small dots called pixels. Each pixel has a colour associated with it and the colour of a pixel is represented in memory by a binary number. A black and white bitmap image could be stored with the colour of each pixel represented in one bit using a 0 for black and a 1 for white.



To store a colour image more bits are required for each pixel as each pixel could be one of many colours. High quality colour images often allow a choice of 16 million different colours. For this number of colours each pixel must be represented by a 24-bit binary number.

4. The amount of memory required to store the colour of a pixel depends upon the number of different colour that make up an image.
   - In a black and white image there are only two possible colours for a picture. The colour of each pixel can therefore be represented in one bit (set to 0 for black, 1 for white or vice-versa).
   - In a 256-colour image, each pixel's colour is represented by 8 bits (1 byte). This is because you can represent 256 different numbers in a byte.
   - To calculate how many bits you need to store a particular number of colours, keep dividing the number of colours by two until you get to (approximately) 1. Count how many times you divided by two. This is the number of bits required. If you want your answer in number of bytes per pixel instead of bits, divide by 8 at the end as one byte=eight bits.
5. The size of a file containing a bitmap image can be calculated like this :

*File Size = No. of pixels horizontally in image * No. of pixels vertically in image * No. of bytes per pixel*

6. AQA have produced two collections of additional questions to accompany the two specimen papers for the new specification. The additional questions papers contain extra questions for the new topics on the syllabus. For encryption, these are question 4 from the AS paper and question 10 from the A Level paper.

6

0 4 Figure 1 shows a message being encrypted using a Caesar cipher.

Figure 1

Plaintext: MONKEY

Ciphertext: FHGDXR

0 4 . 1 Encrypt the plaintext "COMPUTING" using the Caesar cipher with the settings shown in **Figure 1**.

[1 mark]

0 4 . 2 Decrypt the ciphertext "IRMAHG" using the Caesar cipher with the settings shown in **Figure 1**.

[1 mark]

0 4 . 3 Using the Vernam cipher method, the plaintext "SOS" is to be encrypted. "S" will be encoded using 8-bit ASCII as 01010011 and "O" as 01001111.

The key 10111001 00110101 00011010 will be used to perform the encryption.

Perform this encryption, showing how you have worked out what the ciphertext would be from the plaintext.

[3 marks]

5

Paper 2 V1.0

Turn over

1 0 **Figure 7** shows a message being encrypted using a Caesar cipher.

Figure 7

Wheel settings to use

Plaintext: CROCODILE
↓
Ciphertext: ODAOAPUXQ

1 0 . 1 Decrypt the ciphertext "QXQBTMZF" using the Caesar cipher with the settings shown in **Figure 7**.

[1 mark]

1 0 . 2 Using the Vernam cipher method, the plaintext "RUN" is to be encrypted. "RUN" will be encoded using 8-bit ASCII, according to the ASCII table in **Figure 8**.

Figure 8

Letter	ASCII Code	Letter	ASCII Code	Letter	ASCII Code
A	01000001	J	01001010	S	01010011
B	01000010	K	01001011	T	01010100
C	01000011	L	01001100	U	01010101
D	01000100	M	01001101	V	01010110
E	01000101	N	01001110	W	01010111
F	01000110	O	01001111	X	01011000
G	01000111	P	01010000	Y	01011001
H	01001000	Q	01010001	Z	01011010
I	01001001	R	01010010		

The key 10111001 01001101 01000001 will be used to perform the encryption.

Perform this encryption, showing how you have worked out what the ciphertext would be from the plaintext.

[3 marks]

1 0 . 3 Explain why messages encrypted using a Caesar cipher can be easily cracked, and why the Vernam cipher is, under certain conditions, considered to be perfectly secure.

[4 marks]

8

7. The original Baudot code focused on representing the characters more than the layout of the message; the Baudot-Murray code created later includes various control characters, but for teaching the principles of the Vernam cipher, the Baudot code will be sufficient.

V	IV			I	II	III	Con-ti-nen-tal	Gray	Let.	Fig.	V	IV		I	II	III	UK
							-	-									-
		A	1	•					A	1				•			
		É	&	•	•				/	1/				•	•		
		E	2		•				E	2					•		
		I	o		•	•			I	3/					•	•	
		O	5	•	•	•			O	5				•	•	•	
		U	4	•		•			U	4				•		•	
		Y	3			•			Y	3						•	
	•	B	8			•			B	8		•				•	
	•	C	9	•		•			C	9		•		•		•	
	•	D	0	•	•	•			D	0		•		•	•	•	
	•	F	f		•	•			F	5/		•			•	•	
	•	G	7		•				G	7		•			•		
	•	H	h	•	•				H	¹		•		•	•		
	•	J	6	•					J	6		•		•			
	•	*Figure*	*Blank*						*Fig.*	*Bl.*		•					
•	•	*Erasure*	*Erasure*						*	*	•	•					
•	•	K	(	•					K	(	•	•		•			
•	•	L	=	•	•				L	=	•	•		•	•		
•	•	M	)		•				M	)	•	•			•		
•	•	N	N°		•	•			N	£	•	•			•	•	
•	•	P	%	•	•	•			P	+	•	•		•	•	•	
•	•	Q	/	•		•			Q	/	•	•		•		•	
•	•	R	–			•			R	–	•	•				•	
•		S	;			•			S	7/	•					•	
•		T	!	•		•			T	²	•			•		•	
•		V	'	•	•	•			V	¹	•			•	•	•	
•		W	?		•	•			W	?	•				•	•	
•		X	,		•				X	9/	•				•		
•		Z	:	•	•				Z	:	•			•	•		
•		t	.	•					–	.	•			•			
•		*Blank*	*Letter*						*Bl.*	*Let.*	•						

Appendix 2

1. Six AS Level students gave feedback on the system in the form of a questionnaire. A couple of students weren't able to give feedback on all parts of the system as they didn't have enough time to use every part.

Bibliography

Crypto Museum, 2012. *The Vernam Cipher*. [online] Available at: http://www.cryptomuseum.com/crypto/vernam.htm [Accessed 24 September 2015].

Grantham B. 2007. *Bitmap Steganography: An Introduction*. [pdf] Available at: http://vanilla47.com/PDFs/Cryptography/Steganography/Bitmap%20Steganography%20-%20An%20Introduction.pdf [Accessed 10 October 2015].

Kessler, G. C., 2001. *Steganography: Hiding Data Within Data*. [online] Available at: http://www.garykessler.net/library/steganography.html [Accessed 25 September 2015].

Singh, S. 2000. *The Code Book*. London: Fourth Estate.

Wikipedia contributors. *Baudot Code*. [online] Available at: https://en.wikipedia.org/wiki/Baudot_code [Accessed 24 September 2015].

Wikipedia contributors. *BMP File Format*. [online] Available at: https://en.wikipedia.org/wiki/BMP_file_format [Accessed 25 September 2015].