**Cryptographic Foundations**

**RSA Encryption Java Application**

1. Researched RSA encryption and figured out how to do the algorithm on paper, once encryption and decryption was achieved, the Programming language Javas BigInteger was then researched.

BigInteger has within it methods to perform all the functions required to achieve RSA Encryption. A bit length has to be determined which grow exponentially with powers of 2, a length of 4096 was selected as it’s the current standard of RSA Encryption implementations in the real world.

Random class is used to determine a random BigInteger which will use its probablePrime method, there is a 2^-100 chance of probablePrime producing a composite number this not likely to ever happen.

Calculations were coded to calculate n and phiOfN, once found p and q are overwritten to hide the original prime values, destroying any easy way of calculating d, until the Java Garbage Collector disposes the values completely.

e is then calculated randomly, calculations for e are continuously carried out until it is greater than 1, less than phiOfN and both it and phiOfN have a greatest common divisor (gcd) of 1.

Every character in the message must be a positive integer, this is achieved by getting the ascii value of each character and storing then in a byte array. This byte array is then converted into a BigInteger representing the plaintext message in a number format.

The message value is put to the power of e which is congruent to the ciphered text mod n. this ciphertext is then sent over the network to the recipient of the message who will use d and n to decrypt the ciphertext.

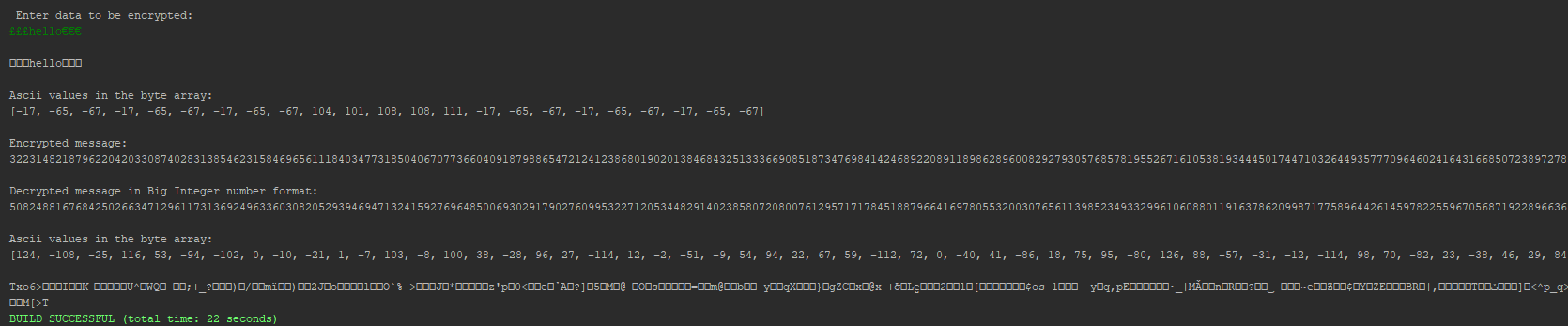
The ciphertext is put to the power of d which is congruent to the decrypted message mod n.

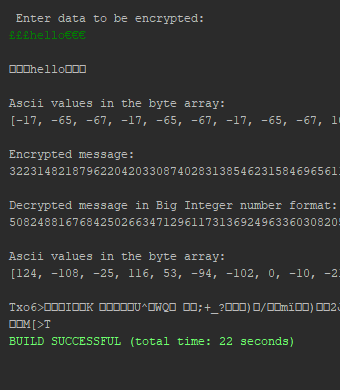
The Decrypted message is still in BigInteger format, this is then converted back to a byte array, the values in the new byte array should match the values in the first byte array. The numbers within are converted back to their ascii counterparts and stored in a message String.

The String when printed should equal the message that was sent.

1. It being a single file application prevents testing from different machines using peer to peer, though if implemented considering this it should work.

Despite specifying UTF-8 the application still uses ascii by default making certain characters like the pound and euro symbol unencryptable, causing the message to get completely messed up in encryption and effectively breaking the algorithm.



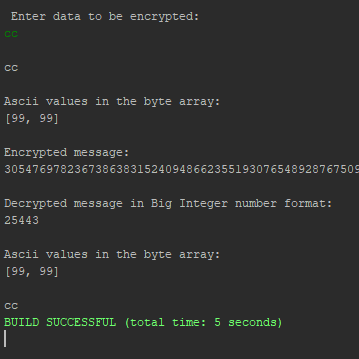
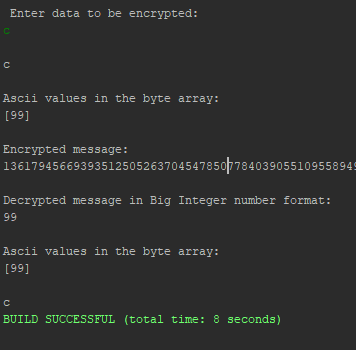


1. Main problem encountered matches the limitation when considering a user can freely type in a pound, Euro or dash symbol, which can potentially break the algorithm.

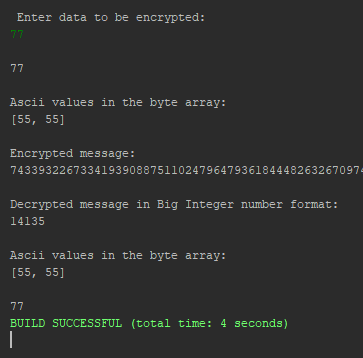
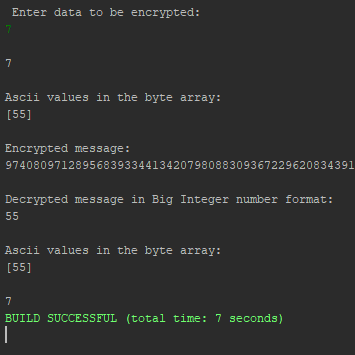
Initial problems where figuring out how text is supposed to be changed into integer data, but this was resolved using ascii, byte Arrays and the BigInteger class.

Additionally when pasting in large pieces of text, the Scanner class cannot handle any CRNL (Carriage Return or New Line) without proceeding with the algorithm. This is evident in example (d) below where one line gets encrypted, sent over and then decrypted, where as the second and any proceeding lines of data are lost. Sticking to one line, the system can handle giant amounts of data.

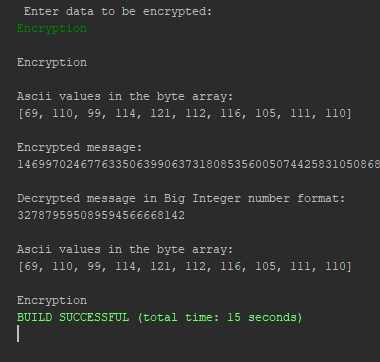
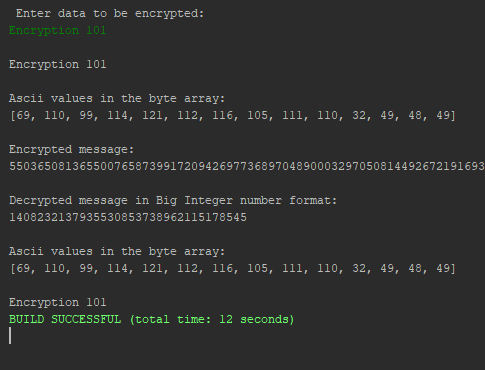
(a) Examples with single and double letters.



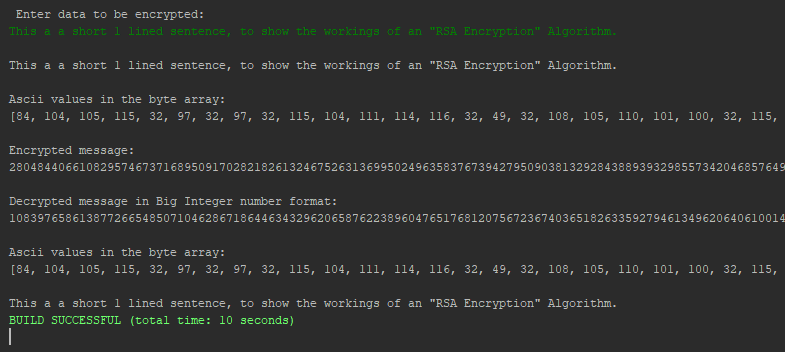
Examples with single and double numbers.



(b) Example on full words and numbers.

(c) Example on a short sentence.



(d) Example using a large piece of text

