What is the final design of the program?

The classes reflect five responsibilities:

* key generation
* decryption
* encryption
* string parsing
* file handling

Initial settings are set when the class is instantiated, such as the key to be used for encryption or decryption, or how many chunks to split a message into.

The encryption process:

Read in the public key:

e n

65537 112261115257780114381657619942310817957

Read in a message: ‘abcdef’

Split message into parts, for example 3 character parts: [‘abc’, ‘def’]

‘a’ ‘b’ ‘c’ ‘d’ ‘e’ ‘f’

Translate into BigIntegers using the characters’ Unicode values: [**97**00098**00099**, 100**00101**00102]

Encrypt these integers:

[31495632388743691742356620824474429649, 77990186094313540699241866795966817542]

Write to a file.

The decryption process:

Read in the private key:

d n

108403568657225911123489228638924770753 112261115257780114381657619942310817957

Read in the cipher text:

“31495632388743691742356620824474429649 77990186094313540699241866795966817542”

Split into parts separated by whitespace:

[31495632388743691742356620824474429649, 77990186094313540699241866795966817542]

Decrypt these integers:

[970009800099, 1000010100102]

Translate each section into Unicode characters (remembering to add zeros to the start before splitting)

[‘abc’, ‘def’]

Join the parts together and print the result to the screen:

‘abcdef’

How did I get to this design?

Group discussion

How to understand the key generation algorithm:

1. Generate p and q, then find n and phi.
2. Set e to a fixed, relatively small prime.
3. Keep trying different values for p and q such that phi and e are coprime.
4. Use the recursive Extended Euclid algorithm to find the greatest common divisor.
5. Combines part of the gcd output with phi to get d.
6. Publishes n and e as the public key.
7. Keeps n and d as the private key.

Test driven development

Used the worked examples given to us as expected output.

Particularly challenging to get the right value for d – ended up having to translate an implementation written in python, plus the algorithm written in the lecture slides, to arrive at an algorithm.

Produced single class that got the keys and did the encryption on a given string.

Unexpected Limitations

Discovered that only values up to a certain number may be encrypted accurately. The maximum is dependent on n, because the encryption/decryption processes involve mod n. Any larger value just wraps round. To mitigate this the program splits the message into separate chunks, which are encrypted and decrypted separately from each other. This allows me to be in control of the maximum number that might be sent into the encryption/decryption process, and so avoid wrap-around errors.

Extensive refactoring into classes

First had a Message value object being passed around

Refactored into a class for each responsibility within the overall process.

1. I began by writing the actions the program must perform:
   1. Key generation
   2. Encryption
   3. Decryption
2. I wrote out each of the stages within each action in more detail. For example, KeyGenerator must:
   1. Generate p, q and n, with the correct relationship between phi and e
   2. Find d.
   3. Write the keys to a file.

Refactoring Stage 1: processes

KeyGenerator:

**Creates a public and private key pair**

*Writes them to disk (FileHandler)*

Encrypter:

*Reads a message from a file (FileHandler)*

*Reads a public key from a file (FileHandler)*

**Encrypt the message, given a public key**

*Converts the message to a BigInteger array (StringParser)*

**Encrypts the BigInteger array using the public key**

*Converts the BigInteger array to a string (StringParser)*

*Writes the encrypted version to a file (FileHandler)*

Decrypter:

*Reads a cipher text from a file (FileHandler)*

*Reads a private key from a file (FileHandler)*

**Decrypt the message, given a private key**

*Converts the message to a BigInteger array (StringParser)*

**Decrypts the BigInteger array using the private key**

*Translates the array to a string of Unicode characters (StringParser)*

*Writes the decrypted string to a file (FileHandler)*

Refactoring Stage 2: responsibilities

Encrypter:

Given a public key, converts a BigInteger array to an encrypted BigInteger array.

Decrypter:

Given a private key, converts a cipher BigInteger array to a decrypted BigInteger array.

StringParser Library:

**For encryption:**

Convert a string to BigInteger array using Unicode values

Concatenate a BigInteger array into a string

**For decryption:**

Convert a ciphertext string into a BigInteger array

Convert a BigInteger array into a string using Unicode values

File Handler:

Write a string to a file, replacing the contents

Read a string from a file, handling new lines sensibly

Refactoring Stage 3: move StringParser into Encrypter/Decrypter?

Emergent Classes

An initial value object emerged that is used inside the KeyGenerator to pass values (p, q, n, etc.) from function to function.

Classes for the public and private key allowed the keys to be passed around and handled converting to and from strings.

Things that went well

Group discussion helped to understand the stages of the algorithm.

Test Driven Development helped immensely when implementing the algorithm. Each stage could be built up with confidence as the previous stage had proven to be producing the right output.

Things that could have gone better

Had to rewrite tests when refactoring StringParser.

Incomplete tests – relied too heavily on integration testing rather than individual unit tests.

How the private key is found

How e is found

How d is found

How the prime numbers are found

Process: