|  |
| --- |
| Encryption, Decryption & Man-In-The-Middle Attack Implementation Using Python |
| Overview |
| Aayush Garg | aayushgarg.bu@gmail.com |

Table of Contents

[Objective 3](#_Toc532912197)

[Algorithms 3](#_Toc532912198)

[Additional algorithms 3](#_Toc532912199)

[Elgamal algorithm 4](#_Toc532912200)

[Baby Step Giant Step algorithm 4](#_Toc532912201)

[Programming approach 5](#_Toc532912202)

[Snapshot displaying Main Menu 5](#_Toc532912203)

[UML diagrams 6](#_Toc532912204)

[Instructions to execute the program 8](#_Toc532912205)

[Figure displaying all the files opened using Spider IDE 8](#_Toc532912206)

[Figure displaying the location of Run File button on Spyder IDE 9](#_Toc532912207)

[Snapshots to represent functionality of available options 10](#_Toc532912208)

[Snapshot displaying Generate Keys functionality 10](#_Toc532912209)

[Snapshot displaying Encrypt Message functionality 10](#_Toc532912210)

[Snapshot displaying Decrypt Message functionality 11](#_Toc532912211)

[Snapshot displaying Break Encryption functionality 11](#_Toc532912212)

[Snapshot displaying an Exit from the program 12](#_Toc532912213)

[Snapshot displaying the file structure 12](#_Toc532912214)

[Snapshots to represent input validations 13](#_Toc532912215)

[Snapshot displaying the recommendation to generate Keys in case of no file found 13](#_Toc532912216)

[Snapshot displaying Invalid input message and displaying the option to re-enter 13](#_Toc532912217)

[Snapshot displaying message on entering input value to encrypt greater than the upper bound of the public key component generated 14](#_Toc532912218)

[Limitations and Future work 14](#_Toc532912219)

# Objective

The objective of the project is to encrypt and decrypt data using asymmetric encryption. Also, a Man-In-The-Middle attack technique to decipher the encrypted message to get the original message has also been demonstrated.

# Algorithms

Elgamal algorithm has been implemented to achieve Keys generation, Encryption of data and Decryption of encrypted data.

Baby Step Giant Step algorithm has been implemented to conduct a Man-In-The-Middle attack by deciphering the encrypted message to get the original message.

# Additional algorithms

Below techniques/algorithms were utilised to smoothen the process:

1. Extended Euclidean algorithm – to find the multiplicative inverse of **‘a’** modulo **‘b’**, i.e. in algebraic field extensions.
2. Exponentiation algorithm – to find the modular exponentiation, i.e. exponentiation performed over a modulus.

# Elgamal algorithm

Also known as Asymmetric Key Encryption has been utilised to encrypt and decrypt messages shared between two parties, making it cryptographically secure to perform the task.

Usually, the person who needs to receive encrypted message generates and publishes Public Key which is utilized by the sender for encryption of the message to be sent, along with a generated Header. The receiver of message i.e. the one whose public key has been utilised to encrypt the message uses his/her private key and sender’s generated Header to decrypt the encrypted message to original plain text.

# Baby Step Giant Step algorithm

The algorithm is utilised to implement a Man-in-the-middle attack technique by computing the discrete logarithm. Throughout this implementation the algorithm is used to figure out ‘l’ in equation, a = b^l mod p

# Programming approach

The project has been developed to minimise user input and store the information in files that can be used as per usage. The user is only required to enter the message to be encrypted and rest everything which includes Keys, Encrypted Message and generated Header are stored in below two files:

* KEY.txt
* ENC.txt

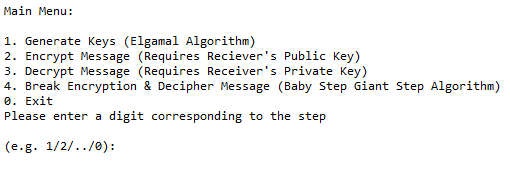
The names of above files can be changed in the code easily as these are just initialized once and are being passed as arguments.

Below available options are visible to the user as the Main Menu (an indexed option list) for which the user enters the digit corresponding to the option to be performed and press Enter:

* Generate Keys (Elgamal Algorithm)
* Encrypt Message (Requires Reciever's Public Key)
* Decrypt Message (Requires Receiver's Private Key)
* Break Encryption & Decipher Message (Baby Step Giant Step Algorithm)
* Exit

The program continues to execute until the user enters 0 for Exit.

### Snapshot displaying Main Menu



Please note more snapshots are available towards the end of this document which will provide a brief idea of what options are presented to the user and their outcomes respectively.

# UML diagrams

|  |
| --- |
| **Main** |
| -usrInpt: int  common: Common(keyFileName: str, encFileName: str)  keyGeneration: KeyGeneration(objCommon: Common())  encryption: Encryption(objCommon: Common())  decryption: Decryption(objCommon: Common())  breakEncryption: BreakEncryption(objCommon: Common()) |
| Main()  \_\_str\_\_(): str  Start(): None |

|  |
| --- |
| **Common** |
| -i: int  keyFile: str  encFile: str |
| Common(keyFileName: str, encFileName: str)  \_\_str\_\_(): str  IsInteger(element: str): int  IsDictionary(element: str): dict  AreElementsIntegers(userEntry: str, count: int): list  WriteToFile(filename: str, content: dict): None  GetRandomPrime(limit: int): int  CheckPrime(randomInteger: int): bool  GetExponentiation(x: int, e: int, y: int): int  FindFile(filename: str): bool  ReadFromFile(filename: str): str |

|  |
| --- |
| **KeyGeneration** |
| -limit: int  -keysFileName: str  common: Common() |
| KeyGeneration(objCommon: Common())  \_\_str\_\_(): str  GenerateAndStoreKeys(): None  -GenerateKeys(limitForPrime: int): dict  -GetSecret(randomPrimeValue: int): int  -GetPrimitiveElement(secretKey: int): int  GetKeysDictionary(primitveElement: int, primitiveRaisedSecretModPrime: int, randomPrime: int, secret: int): dict  ReadReceiversKeys(private: bool): list |

|  |
| --- |
| **Encryption** |
| -encMsgFileName: str  common: Common()  keyGeneration: KeyGeneration(objCommon: Common()) |
| Encryption(objCommon: Common())  \_\_str\_\_(): str  EncryptMessage(): None  -GetEncryptedMessageAndHeader(msg: int, primitiveElement: int, primitiveRaisedSecretModPrime: int, randomPrime: int): dict  ReadEncryptedMessageAndHeader(): tuple |

|  |
| --- |
| **Decryption** |
| common: Common()  keyGeneration: KeyGeneration(objCommon: Common())  encryption: Encryption(objCommon: Common()) |
| Decryption(objCommon: Common())  \_\_str\_\_(): str  DecryptMessage(): None  PerformExtendedEuclidean(element: int, prime: int): list  GetInverseModPrime(element: int, prime: int): int |

|  |
| --- |
| **BreakEncryption** |
| common: Common()  encryption: Encryption(objCommon: Common())  keyGeneration: KeyGeneration(objCommon: Common())  decryption: Decryption(objCommon: Common()) |
| BreakEncryption(objCommon: Common())  \_\_str\_\_(): str  BreakEncryptionGetMessage(): None  -PerformBabyStepGiantStep(encryptedMessage: int, header: int, primitiveElement: int, primitiveRaisedSecretModPrime: int, randomPrime: int): None  -ComputePowerVariable(answerVar: int, baseVar: int, prime: int): int |

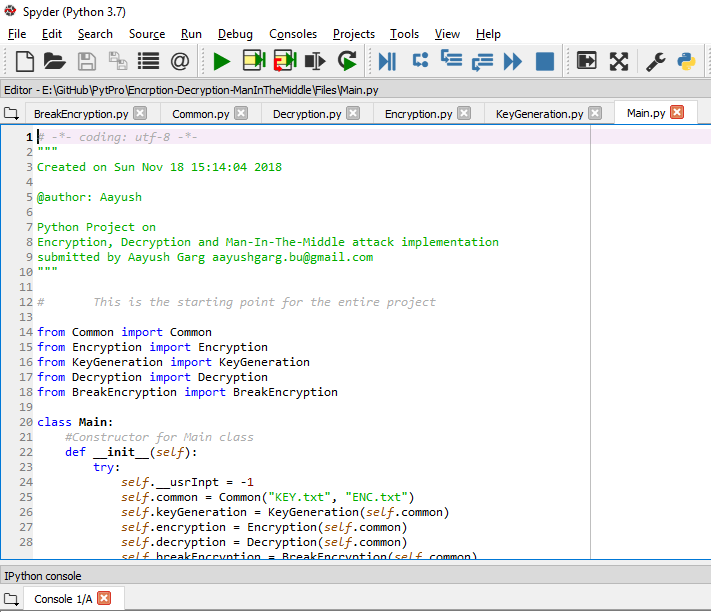
# Instructions to execute the program

All the files to be placed in a single folder are as mentioned below:

1. Main.py
2. Common.py
3. KeyGeneration.py
4. Encryption.py
5. Decryption.py
6. BreakEncryption.py

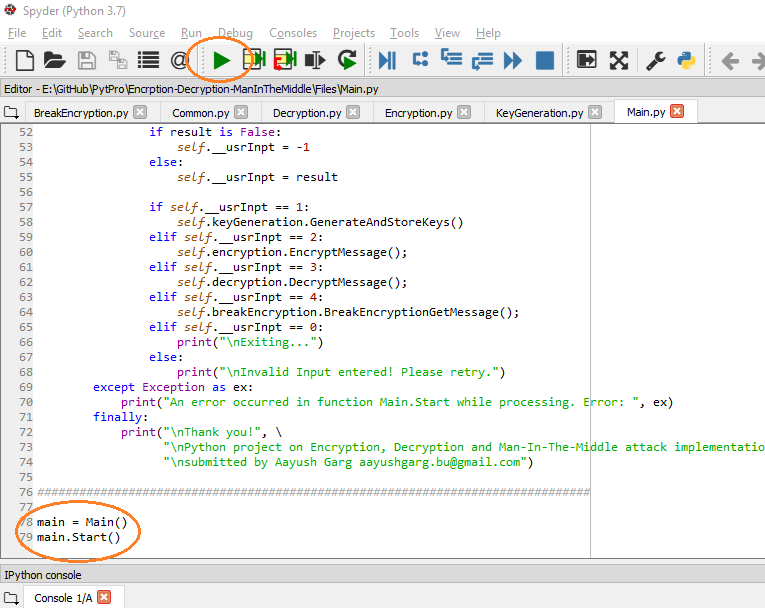
The entry point to program is Main class in the Main.py file. All the files are to be opened in an IDE supporting Python e.g. in current scenario Spyder IDE with Anaconda is being used.

### Figure displaying all the files opened using Spider IDE



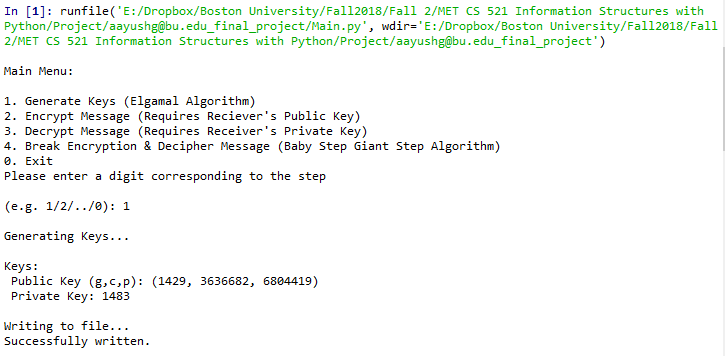
Main.py is then executed to run the program as the object of class Main is being initialised in this file and function Start() executes the program. In the current scenario F5 (Green Play button) is pressed on Main.py file which starts the interpretation of the code and launches the program.

### Figure displaying the location of Run File button on Spyder IDE

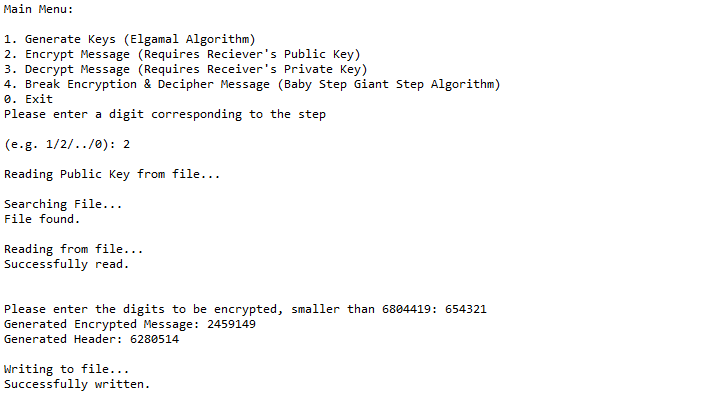


# Snapshots to represent functionality of available options

## Snapshot displaying Generate Keys functionality



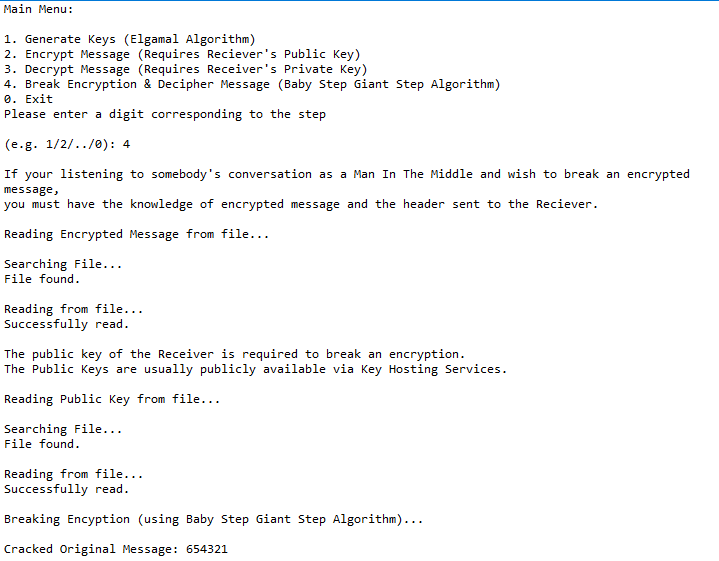
## Snapshot displaying Encrypt Message functionality



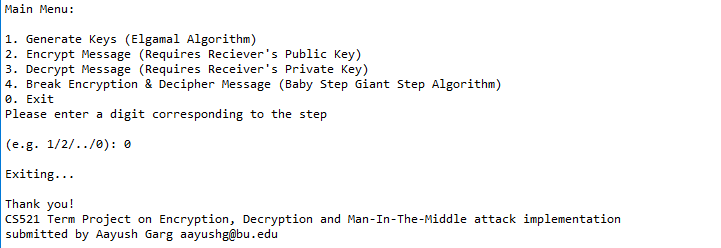
## Snapshot displaying Decrypt Message functionality



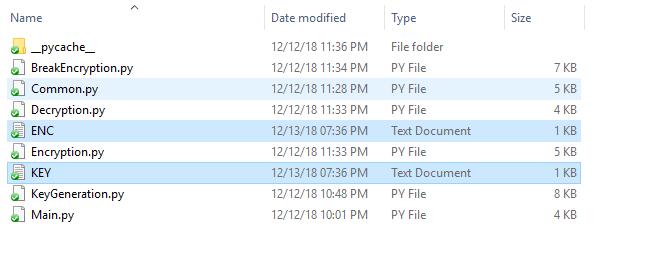
## Snapshot displaying Break Encryption functionality



## Snapshot displaying an Exit from the program

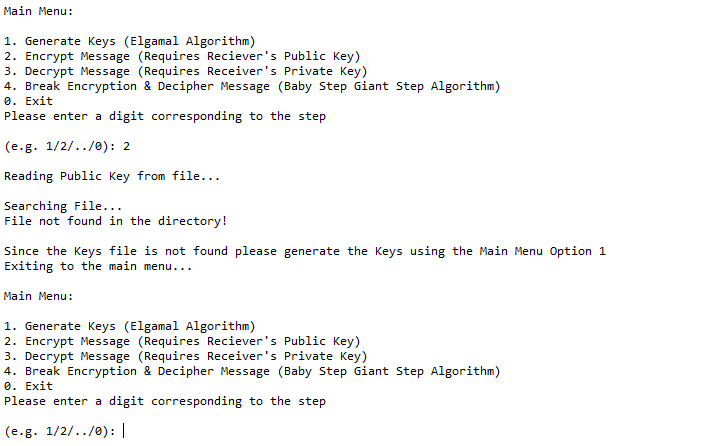


## Snapshot displaying the file structure

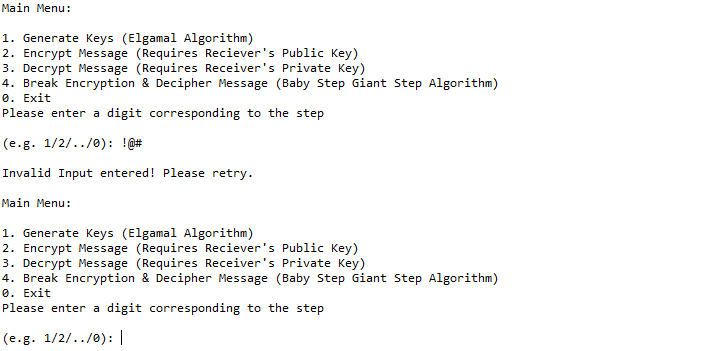


# Snapshots to represent input validations

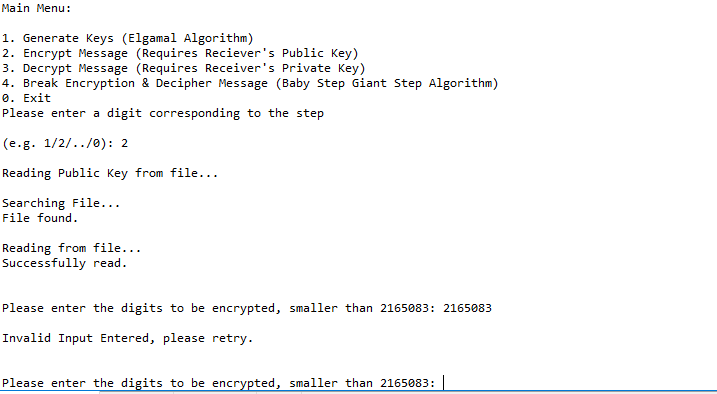
## Snapshot displaying the recommendation to generate Keys in case of no file found



## Snapshot displaying Invalid input message and displaying the option to re-enter



## Snapshot displaying message on entering input value to encrypt greater than the upper bound of the public key component generated



# Limitations and Future work

As of now the solution only supports the digits and numerical values to be encrypted and decrypted. The next phase of development will be the introduction of alphabets and string messages to be encrypted and decrypted.