RSA Implementation

Abheek Mondal

A20438046

ECE 543

04 April 2023

Part A

Prime Numbers:

For this part there are 2 programs that were implemented. The first is *primecheck* and *primegen*. Primecheck takes a single argument, a positive integer, and then determines if it is a prime number or not.

This is a Python program that uses the Miller-Rabin primality test to check whether a given number is prime or not. The Miller-Rabin test is a probabilistic algorithm that can determine whether a number is composite with high probability. The program takes an integer argument from the command line and runs the Miller-Rabin test on it with a fixed number of iterations, I have chosen 5. The result is then printed on the console. A thing to keep in mind is that the Miller-Rabin test is not foolproof and there is a small chance that it may incorrectly classify a composite number as prime. However, the probability of error decreases exponentially with the number of iterations, so running the test with a sufficient number of iterations can provide a high level of confidence in the result.

Here are the results using the provided test cases:

```
(base) PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\primecheck.py 32401
True
(base) PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\primecheck.py 3244568
False
(base) PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\primecheck.py 324456823412
False
(base) PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\primecheck.py 1876893421
False
(base) PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\primecheck.py 13
True
(base) PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project>
```

The second program, primegen, takes a single positive integer, which represents the number of bits and produces a prime number of that number of bits.

This is a Python program that generates a random prime number with a specified number of bits using the Miller-Rabin primality test. The program takes an integer argument from the command line specifying the number of bits for the prime to be generated. The program generates a random candidate number with the specified number of bits, sets the high-order bit and the low-order bit to 1 to ensure that the number is odd and has the specified number of bits, and runs the Miller-Rabin test on the candidate. If the candidate passes the test, it is returned as the prime number. The generated prime number may not be cryptographically secure and should not be used for sensitive applications without further testing and verification. The number of iterations in the Miller-Rabin test can be adjusted to increase the security of the generated prime, but this may also increase the running time of the program.

```
(base) PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\primegen.py 1024
106138184192147819631535618998434403775495895715952995590346082435557342272098965898883392892317333969360З
2295458346805709901909619605269091464986499069619839854059153468933344334946044349646314687348598704019278
55477855571698561834348636853205499949366651832891281330614893417021199358087896892144335012421887
(base) PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\primecheck.py 106138184192
1478196315356189984344037754958957159529955903460824355573422720989658988833928923173339693603229545834680
5709901909619605269091464986499069619839854059153468933344334946044349646314687348598704019278554778555716
985618343486368532054999493666518328912813306148934170211993580878968921443350124218
True
```

Part B

For this, there are a total of 3 implemented programs – keygen, encrypt and decrypt.

Keygen generates a public and private key pair given two prime numbers. This is a Python program that generates RSA public and private keys from two prime numbers using the extended Euclidean algorithm and modular arithmetic. The program takes two integer arguments from the command line specifying the prime numbers p and q. The program calculates the modulus n, the totient of n (phi), and the public exponent e, which is the smallest, odd integer greater than 2 that is coprime to phi. The program then calculates the private exponent d, which is the modular inverse of e modulus phi.

```
PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\keygen.py 127 131
Public Key: (16637, 11)
Private Key: (16637, 14891)
PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project>
```

First Number	Sec Number	n	e	d
1019	1021	1040399	7	890023
1093	1097	1199021	5	478733
433	499	216067	5	172109
1061	1063	1127843	7	964903
1217	1201	1461617	7	1250743
313	337	105481	5	41933
419	463	193997	5	154493
15857942311	15924117337	252523734083740945807	11	206610327860693634131
9151366243	16988733197	155470119490359268871	5	62188047785687667773
15362263919	10007043469	153730842819683295011	5	30746168558862797525

The second program is the encrypt, which takes a public key (n, e) and a single character c to encode, and returns the cyphertext corresponding to the plaintext, m. This is a Python program that encrypts a plaintext message using the RSA public key encryption algorithm. The program takes three integer arguments from the command line specifying the modulus n, the public exponent e, and the plaintext message m. The program encrypts the plaintext message using the RSA encryption formula c = m^e (mod n), where c is the resulting ciphertext. The ciphertext is then printed to the console.

```
PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\encrypt.py 16637 11 20
12046
```

n	e	c	m
1040399	7	99	579196
1199021	5	70	871579
216067	5	89	23901
1127843	7	98	871444
1461617	7	113	1411436
105481	5	105	36549
193997	5	85	147738

252523734083740945807	11	119	90376367963112453043
155470119490359268871	5	109	15386239549
153730842819683295011	5	113	18424351793

The last program is decrypt will take a private key (n, d) and the encrypted character and return the corresponding plaintext. This is a Python program that decrypts a ciphertext message using the RSA private key decryption algorithm. The program takes three integer arguments from the command line specifying the modulus n, the private exponent d, and the ciphertext message c. The program decrypts the ciphertext message using the RSA decryption formula m = c^d (mod n), where m is the resulting plaintext message. The plaintext message is then printed to the console.

```
PS Z:\Downloads\Spring 2023\ECE 543\RSA Implementation Project> python .\decrypt.py 16637 14891 12046
20
```

n	d	m	c
1040399	890023	16560	104
1199021	478733	901767	71
216067	172109	169487	101
1127843	964903	539710	119
1461617	1250743	93069	83
105481	41933	78579	76
193997	154493	1583	122

Overall, the RSA implementation was successful. All the implementations performed their tasks accordingly to the given list of requirements and have successfully accomplished all the test cases.