Applications of RSA, Digital Certificate, SSS to Enable Multi-Level Decryption and Private Key Sharing

Jaehyeon Yoon

Abstract

RSA, Digital Certificate, and SSS (Shamir's Secret Sharing) are some of the most widely used methods of encrypting and decrypting data in current days. RSA encrypts and decrypts data using a mathematical characteristic of prime numbers: it is extremely difficult to prime factorize large numbers. Digital Certificate is the applied version of RSA and is used to prove one's identification. In addition, SSS encrypts and decrypts data using a characteristic of a polynomial graph: the only way to model the exact graph is to know the coordinate values of the k+1 number of points on the graph when it is in kth degree.

However, despite the simplicity in their mathematical mechanism, they come short in some aspects which may be crucial for modern data encryption and decryption applications. For instance, Digital Certificates cannot be used to validate digital identities of more than one person. Also the traditional form of SSS is extremely inefficient as it requires a new complex encryption process to be performed for every single piece of new data, especially when encrypting a long sentence or a large number.

This paper details the process in which the traditional RSA, SSS and digital certificate work. This paper also will detail how the two major shortcomings of them were resolved as well as functioning codes in which the two new implementations were tested of their functionality.

1. Introduction

Motivation

As modern technology evolves rapidly, the frequency and the technology behind digital frauds has evolved along with it. Therefore, individuals' digital identity and the security of it is

at jeopardy. Currently, there are many ways of protecting individuals' digital identity through

data encryption. Some of them are RSA, Digital Certificate and SSS (Shamir's Secret Sharing).

Although these methods have proven their effectiveness over the past years, they still have areas of improvements to fill. They are inadequate and sometimes, incapable, for use under some circumstances. For instance, to encrypt a list of separate data with SSS, one would have to go through a complicated mathematical process to create a new polynomial for every single data in the list. In other words, it is a 'single-use product'. Also, when encrypting a large data, the math that needs to be done can be extremely difficult and inefficient. Additionally, current Digital Certificates are incapable of validating more than one person's digital identity at once.

So, this research was conducted in order to resolve the limitations of the traditional forms of RSA, SSS and Digital Certificate. The new implementation of Digital Certificate allows more than one person to validate their digital identity at once. The new implementation of RSA and SSS allows a much more efficient encryption and decryption process especially when the data is large and/or separately listed.

Design of the Paper

Section 2 details the necessary background knowledge on RSA, Shamir's Secret Sharing, Digital Certificate along with mathematical step by step procedures proofs of each method. Section 3 is about the implementation of RSA and Digital Certificate which allows more than one person to validate their digital identity. Section 4 deals with the implementation of RSA and SSS which allows people to obtain a private key only when more than a certain number decides to do so. In section 3 and 4, an actual code that demonstrates such implementation is included. The conclusion section will summarize and wrap up the whole thing. Lastly, in the appendix section, a code that demonstrates RSA method is included.

2. Background

2.1 RSA

RSA, Riverst Shamir Adlemn, is one of the most widely used methods of data encryption. It involves a public key and a private key. The public key is available to the public, hence the name. Anyone can use it to encrypt data and send the encrypted data to the individual who possesses a paired private key. Since the decryption process requires both the private and the public key, the single person who has the private key will be able to decrypt the data. Since the only person who can encrypt the data is the person with the private key, it is virtually impossible for anyone else to 'hi-jack' the data. [1]

Step 1. Randomly generate two prime numbers: q and p. **Ex) q=31, p=17**

Step 2. Compute N and $\varphi(N)$.

Ex) $N = q \times p = 527$, $\varphi(N) = (q-1)(p-1) = 480$

[Lemma 2.1]

Let N = q×p. Then, φ(N) = Number of integers less than N that is coprime with N = (q-1)(p-1)

[Proof 2.1]

1. Because N is a product of two prime numbers, the only integers that are less than N and are not in coprime with N have to have either q or p as their factors.

2. There will be a 'p' number of integers that is less than N that has q as their factor. 1*q, 2*q, 2*q

... p*q.

3. There will be an 'q' number of integers that is less than N that has p as their factor. 1*p, 2*p, 3*p

... q*p.

4. Therefore, there will be 'q+p -1' number of integers that are less than N and that are not in coprime with N. Because q*p is counted twice, one has to be cancelled out, hence the subtraction of 1.

5. If there are q+1-1 numbers that are less than N and that are not in coprime with it, the number of integers that are less than N and ARE in coprime is 'qp - (q+p-1)'.

6. qp - (q+p-1) = qp - q - p +1 = (q-1)(p-1) ■

Step 3. Find an integer e, that is co-prime to φ(N) and 3 < e < n-1. Ex) e = 7

Step 4. Calculate d. de ≡ 1(mod φ(N)) Ex) d = 343

(∵ 7*343 = 2401 = 1 mod(480))

Step 5. d = private key, e= public key

Step 6. Encryption

Use the formula C=me mod(N), where m is the original 'message', e is the 'public key', and c is the 'encrypted message'. Ex) m = 123, C = 1237 = 30 (mod 527)

Step 7. Decryption

Use the formula m=Cd mod(N), where m is the original 'message', C is the 'encrypted message', and d is the 'private key' of the receiver. Ex) Cd = 30343 = 123 = m (mod 527)

[Proof 7.2] [2]

We want to prove that m=Cd mod(N)

Cd mon(n) = med mod(n) {from 6.1 C=me mod(N) * d}

Euleor's theorem: If m and n are coprime to each other, mφ(n)+1= m mod (n) or mφ(n)= 1 mod (n)

mφ(n)+1= m(q-1)(p-1)+1 = m mod (n) {2.1, φ(N) = (q-1)(p-1) }

med mod(n) = mkφ(n)+1 mod(n) = mkφ(n)m1 mod(n) = (mφ(n))km1 mod(n)

(mφ(n))km1 mod(n) = (1)km mod(n) = m mod(n) ■

RSA, though it is not the simplest method of data encryption, it is used in VPNs, digital

communication channels, etc. Among them, RSA serves to encrypt the users' digital signatures, or their digital identities, relating to the digital certificate.

Figure 1: The simulation of RSA by C++. The details can be found in the Appendix A.

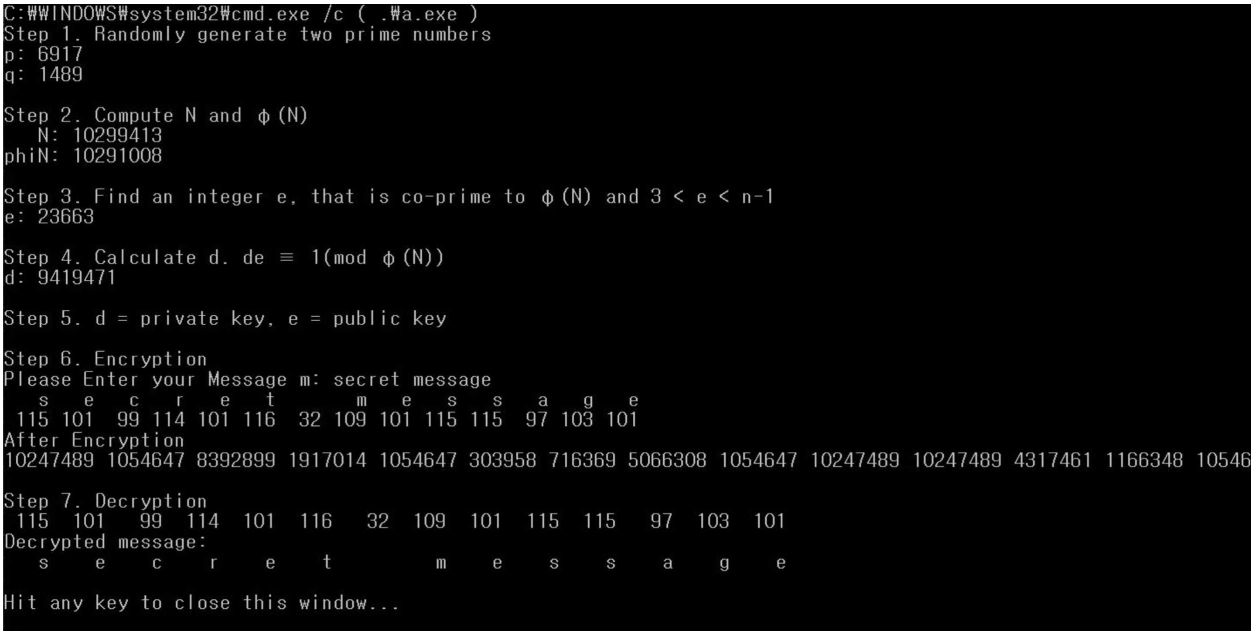


Figure 1: The simulation of RSA by C++. The details can be found in the Appendix A.

1.2 Shamir's Secret Sharing

Shamir's Secret Sharing, often regarded as SSS, is a method of encrypting data through separating the data into n pieces. Such pieces only can restore the initial data if K number of them are available. (n>k) The mechanism works upon a characteristic of polynomials: to complete a polynomial of nth power, there has to be n+1 coordinate points. In other words, the initial polynomial (nth power) can only be restored if there are n+1 points on the cartesian coordinate plane. Reversing the characteristic, we can obtain the constant of the initial polynomial (nth power) if there are n+1 coordinate points.

Step 1. Decide a number of 'keys' needed to decrypt a data, or the value of 'n'. **Ex) n=5**

Step 2. Decide a message and convert it into a numerical value. The converted numerical value will be

the constant term of the final function. (y intercept) **Ex) Message: "!" = 33 (ASCII decimal value)**

= (0,33)

Step 3. Randomly generate 'n-1' number of points on the cartesian coordinate plane.

Ex) (1,2) (2,4) (7,8) (5,0)

Step 4. Form a polynomial of 'n-1'th power that hits all of the randomly generated points and the coordinate for the secret numerical value, with Lagrange Interpolation.

Ex) Randomly generated coordinate values: (1,2) (2,4) (7,8) (5,0)

Secret numerical value: (0,33)

Interpolated polynomial:

$$f(x) = \left(\left(\left(\frac{227(x-5)}{420}+\frac{19}{60}\right)(x-7)-\frac{1}{5}\right)(x-2)+2\right)(x-1)+2$$

$$= \frac{227x^4}{420}-\frac{818x^3}{105}+\frac{15157x^2}{420}-\frac{6283x}{105}+33$$

Step 5. Randomly choose 'n' or more number of coordinate points from the created polynomial which will be the 'keys' used to remodel the polynomial. However, the series of coordinate values must not contain the coordinate value of f(0) as it is the secret message itself.

Step 6 Decryption. If at least n number of 'keys' are available, the exact same polynomial can be recreated with Lagrange Interpolation, thus the constant term which is the 'decrypted data'.

The minimum number of 'keys' that will be required to obtain the initial polynomial and thus the data can be determined through fluctuating the degree of the power of the polynomial: the value of 'n'.

Much like other encryption methods, SSS is highly useful in encrypting data with high sensitivity and importance.

For instance, a nuclear missile launch code would not be safe to only have a single person deciding whether or not to launch a nuclear missile. In such a case, SSS can be implemented as there would have to be at least 'n' number of people agreeing to the launch, making the process much safer than just having a single person to launch the missile. On top of that, it can be implemented in voting procedures as well.

If there has to be at least 'k' number of votes in order for someone to be elected, a polynomial of 'k-1'th power that has a constant term that indicates the fact that the person was elected can be designed. As per the SSS algorithm, the only way for the person to be elected, or to have the indication that the person is elected, is to have at least 'k' number of people to vote for the person which would prevent electoral fraud. [3]

Digital Certificate

Digital certificates are an applied version of RSA and are used to authenticate digital signatures. When the sender, who encrypts a data with their private key, sends the encrypted data to the receiver, who decrypts the data using the sender's private key, their digital identity can be validated if the decrypted data corresponds to the original data prior to the encryption.

This works upon the same logic as the RSA method, just reversed. For RSA, the sender will encrypt the data using the public key and the actual data, then the receiver will decrypt the data

using the private key. For the case of a digital certificate, the sender encrypts and sends the data using their private key, then, the receiver decrypts it using the sender's public key.

Step 1: Generate the value of N, the private key, and the public key. For simplicity, the value of N, the private key and the public key from the RSA section will be used for example. Refer to the procedure of generating the keys above.

Ex) Public key: 343 Private Key: 7 N: 527

Step 2: Convert a message into a numerical value.

Ex) Message: "A" Numerical value: 65 (ASCII decimal value)

Step 3 Encryption: The sender encrypts the converted message using the formula C=md mod(N). Within the formula, C represents the encrypted message, m represents the original message, and d represents the sender's private key Then, the sender sends the original message, the encrypted message and the sender's public key to the receiver.

Ex) The encrypted message: C= 657 mod (N) = 657 mod (527)= 482

Step 4 Decryption: When the receiver decrypts the encrypted message that the sender sent with the formula m=Cd mod(N), if the decrypted message corresponds to the original message that the sender sent, the receiver can validate the sender's digital identity. Within the formula, m represents the decrypted message, C represents the encrypted message, and d represents the sender's public key.

Ex) m= 482343 mod(N) = 482343 mod(527) = 65

3 Problem-1: More than one person cannot validate their digital identity.

Problem statement: All of the 'k' number of people need to sign their signatures in order to validate a contract. (k>1) However, a pandemic has forced them to send their digital signatures online. How can they guarantee that each others' digital signatures are authentic and are not from possible hackers?

The initial idea:

1. Let the data be 'm'
2. Encrypt the data 'm' with person 1's private key (d1) -> m^{d1}
3. Encrypt the already-encrypted data with person 2's private key(d2) -> m^{d1*d2}
4. Decrypt the data using both person's public keys and check if the data corresponds, thus proving both person 1 and person 2's identity.

The initial idea had an error when it was executed. For instance, person 2 would not be able to know the original data as it was encrypted with person 1's private key prior to person 2's encryption. So, the following is the second iteration that resolved such an error.

The second idea:

1. A program will generate a set of public and private keys, in which the sum of d1 and d2 equals d. **Ex) Public key: e=7, Private key: d=343, Message: m=123**

d1 = 300, d2 = 43

2. Person 1 would encrypt the data 'm' using 'd1'.

Ex) md1 = 123300 = 1 mod(527)

3. Person 2 would encrypt the data 'm' using 'd2'.

Ex) md2 = 12343 = 30 mod(527)

4. The two encrypted 'm' would be multiplied and form a final encrypted data.

Ex) md1*md1=md1+d2=md = 30 mod(527)

5. The final data can be decrypted using the generated public key.

Ex) m=Ce=307=123 mod(n)

The C++ implementation of Problem1 can be found in the Appendix B.

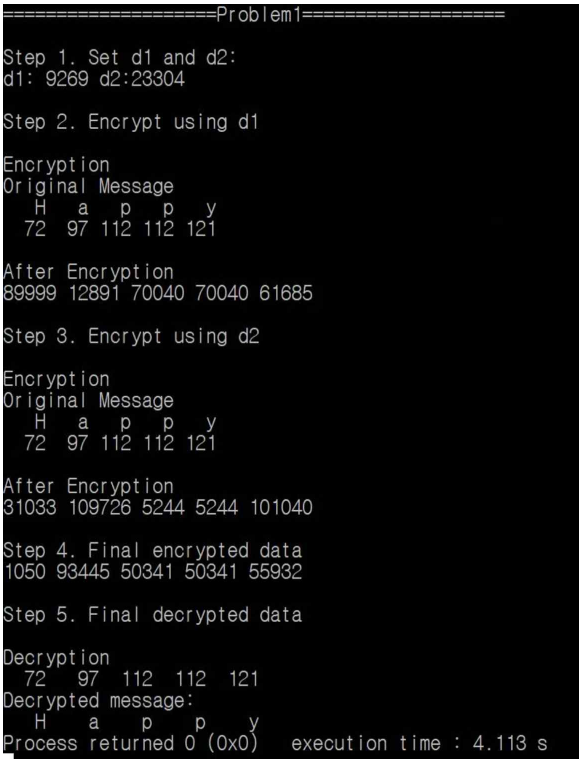


Figure 2: The simulation of Problem by C++. The details can be found in the Appendix B.

4. Problem 2: Large data needed to be decrypted only when at least N out of K number of people agree.

Problem statement: A founder of a company is trying to encrypt the company's classified information, so that the information can be obtained only when 'n' out of 'k' number of executives agree. The founder decides to use SSS to encrypt the information only to realize that when the information is turned into a numerical value, it is astronomically large which will be highly inefficient when SSS is used to encrypt and decrypt it. Also, the information was in the form of a list ('x' number of elements in the list), meaning that the founder needed to model 'x'

number of distinct polynomials according to the SSS algorithm for each of the separately listed information which is even more inefficient. How can the founder use SSS to encrypt a large and/or separately listed data efficiently? Also, how can the executives decrypt the encrypted data with such a method?

Initial idea:

1. A public key is generated from two prime numbers: 'a' and 'b'. Using the public key, a data is encrypted.
2. Generate two Polynomials of n-1th power that each has a constant of 'a' and 'b'. The two polynomials need to be intersected at 'n' number of points.
3. Both constants of the two polynomials can be found when all of the 'n' number of intersections are gathered.
4. Following the RSA private and public key generation algorithm that was mentioned above, use the two prime numbers 'a' and 'b' to generate the private that pairs with the public key that was used to encrypt the data beforehand.
5. Decrypt the data using the generated private key, thus obtaining the company's classified information.

Second idea:

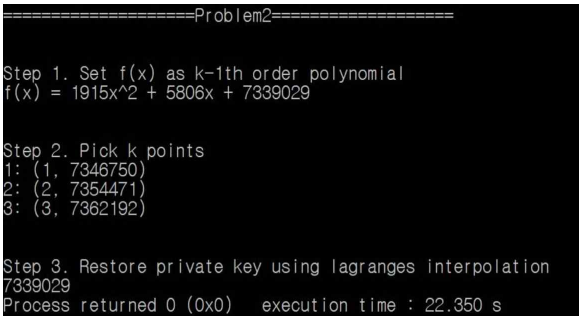
1. Randomly generate 'n-1' number of points on the cartesian coordinate plane.

Ex) (5,1) (5,2) (-3,4) (2,0)

2. Form a polynomial of 'n-1'th power that hits all of the randomly generated points and has a constant term of a private key.

Ex) Randomly generated coordinate values: (5,1) (4,2) (-3,4) (2,0)

Private key: (0,343)



Interpolated polynomial:

3. Randomly choose 'n' or more number of coordinate points from the created polynomial which will be the 'keys' used to remodel the polynomial. However, the series of coordinate values must not contain the coordinate value of f(0) as it is the private key itself.
4. If at least 'n' number of 'keys' are available, the exact same polynomial can be recreated with Lagrange Interpolation, thus the constant term which is the private key.

5. The found private key can be used to decrypt data that was encrypted with its corresponding public key, following the traditional RSA algorithm.

The C++ implementation of Problem 2 can be found in the Appendix C.

Figure 3: The simulation of Problem by C++. The details can be found in the Appendix C.

$$f(x) = (x-5)\left((x-4)\left(\left(-\frac{121}{42}(x-2)-\frac{97}{840}\right)(x+3)-\frac{5}{56}\right)-1\right)+1 = -\frac{121x^4}{42}+\frac{6421x^3}{280}-\frac{11593x^2}{840}-\frac{29761x}{140}+343$$

5 Conclusion

RSA, SSS, and Digital Certificate are some of the most widely used methods of data encryption and decryption due to the simple yet effective mathematical mechanisms that drive them. However, they are not applicable under every condition. For instance, when more than one person is trying to validate their digital identity, Digital Certificate is incapable of doing so. Also, traditional forms of SSS can be extremely inefficient when a list of data has to be encrypted and decrypted individually

Throughout the paper, such shortcomings were resolved through clever alteration of RSA, SSS and Digital certificate. To allow for more than one person to validate their digital identity using Digital Certificate, a property of exponent was implemented. Moreover, the advantageous characteristics of RSA and SSS were combined to allow for the encryption to last for as many times as it is required to: SSS with the alteration doesn't have to be a 'single-use product'.

As the codes below show, the altered version of RSA, SSS, and Digital Certificate are not just theoretical ideas. They are functional methods of data encryption that are not only more efficient but also capable of encrypting data under unexpected but plausible circumstances. For these reasons, it would be captivating to see them being developed further and implemented into real-life applications.

References

[1] Josh Lake, Comparitech, "What is RSA encryption and how does it work?"
URL:
https://www.comparitech.com/blog/information-security/rsa-encryption/

[2] Margaret Rouse, TechTarger, "RSA algorithm (Rivest-Shamir-Adleman)"
URL:
https://searchsecurity.techtarget.com/definition/RSA

[3] Adi Shamir, Massachusetts Institute of Technology, "How to Share a Secret"
URL:
https://cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf

Appendix A. C++ implementation of RSA

```
class RSA_ENTRY
{
public:
long long d, e, N;string s;
RSA_ENTRY() {
srand(time(0));
printf("Step 1. Randomly generate two prime numbers\n");
long long p = pickRandomPrime();long long q = pickRandomPrime();printf("p: %lld\n", p);
printf("q: %lld\n", q);
printf("\nStep 2. Compute N and ?(N)\n");N = p*q;
long long phiN = (p-1)*(q-1);printf("N: %lld\n", N);
printf("phiN: %lld\n", phiN);
printf("\nStep 3. Find an integer e, that is co-prime to ?(N) and3 < e < n-1\n");
e = coprimeWith(phiN, N);printf("e: %lld\n", e);
printf("\nStep 4. Calculate d. de ??1(mod ?(N))\n");d = findInverse(e, phiN);
printf("d: %lld\n", d);

printf("\nStep 5. d = private key, e = public key\n");
printf("Please Enter your Message m: ");getline(cin s);
}
vector<long long> Encrypt() {
printf("\nEncryption\n");
printf("Original Message\n");
for (int i=0; i<s.size(); ++i) {printf("%4c", s[i]);
}
printf("\n");
for (int i=0; i<s.size(); ++i) {printf("%4d", s[i]);
}
printf("\n\nAfter Encryption\n");
vector<long long> encrypt(s.size());
for (int i=0; i<s.size(); ++i) {
encrypt[i] = myPow(s[i], e, N);
printf("%lld ", encrypt[i]);
}
return encrypt;
}
```

```
vector<long long> Decrypt(vector<long long> encrypt){printf("\n\nDecryption\n");
vector<long long> decrypt(encrypt.size());
for (int i=0; i<decrypt.size(); ++i) {
decrypt[i] = myPow(encrypt[i], d, N);printf("%4lld ", decrypt[i]);
}
printf("\nDecrypted message:\n");
for (int i=0; i<decrypt.size(); ++i) {printf("%4c ", decrypt[i]);
}
return decrypt;
}
```

Appendix B. C++ implementation of Problem1 (multi level RSA)

```
void Problem1(RSA_ENTRY* rsa)
{ printf("\n\n====================Problem1 ==================\n");
printf("\nStep 1. Set d1 and d2:\n");
long long d1 = rand()%(rsa->e), d2 = (rsa->e)-d1;
printf("d1: %lld d2:%lld\n", d1, d2);
printf("\nStep 2. Encrypt using d1\n"); rsa->e = d1;
auto e1 = rsa->Encrypt();

printf("\n\nStep 3. Encrypt using d2\n"); rsa->e = d2;
auto e2 = rsa->Encrypt();

printf("\n\nStep 4. Final encrypted data\n");
vector<long long> e;
for (int i=0; i<e1.size(); i++)
{ e.push_back((e1[i]*e2[i])%(rsa->N));
printf("%4lld ", e[i]);
}

printf("\n\nStep 5. Final decrypted data"); rsa->Decrypt(e);
}
```

Appendix C. C++ implementation of Problem2 (Private Key Sharing)

```
double lagrangesInterpolate(pair<long long, long long> f[], int xi, int n)
{
double result = 0;
for (int i=0; i<n; i++)
{
double term = f[i].second;
for (int j=0;j<n;j++)
{
if (j!=i)
term = term*(xi - f[j].first)/double(f[i].first - f[j].first);
}
```

```
result += term;
}
return result;
}
void Problem2(RSA_ENTRY* rsa) {
int K = 3;
long long x2_coeff = rand()%10002+1, x_coeff = rand()%10002+1, constant = rsa->d;
printf("\n\n====================Problem2=====================\n");
printf("\n\nStep 1. Set f(x) as k-1th order polynomial\n"); printf("f(x) = %lldx^2 + %ll
+ %lld\n", x2_coeff, x_coeff, constant);
pair<long long, long long> f[] = {
{1,x2_coeff*(1) + x_coeff*(1) + constant},
{2,x2_coeff*(2) + x_coeff*(2) + constant},
{3,x2_coeff*(3) + x_coeff*(3) + constant}
};
printf("\n\nStep 2. Pick k points\n");
for (int i=0; i<3; i++) {
printf("%d: (%lld, %lld)\n", i+1, f[i].first, f[i].second);
}
printf("\n\nStep 3. Restore private key using lagranges interpolation\n"); cout << (long long)lagrangesInterpolate(f, 0, 2);
}
```

Appendix D. C++ implementation of helper functions

```
// Return whether num is prime or not
bool isPrime(long long num)
{
for (long long i=2; i*i<=num; i++)
{
if (num%i==0) return false;
}
return true;
}
// Return whether a and b is coprime or not
bool isCoprime(long long a, long long b)
{
for (long long i=2; i<=min(a, b); i++)
{
if (a%i==0 && b%i==0) return false;
}
return true;
}
// Pick a random prime number between 2~10002
long long pickRandomPrime()
{
while(1)
{
int randomNumber = rand()%10000+2;
if (isPrime(randomNumber)) return randomNumber;
}
}

// Return a random number that is coprime with phiN within a range of 4~N-1
long long coprimeWith(long long phiN, long long N)
{
while(1)
{
long long randomNumber = rand()%(N-1);
if (randomNumber<=3) continue;



if (isCoprime(randomNumber, phiN))
return randomNumber;
}
}

// Find a modular multiplicative inverse of e in mod phiN
long long findInverse(long long e, long long phiN)
{
int i = 1;

while(1)
```

```
{
if (i*e%phiN != 1)
{
i = i+1;
}
else
{
return i;
}
}
}

// return a^b (mod phiN)
long long myPow(long long a, long long b, long
long phiN) {
long long ret = 1;
for (int i=0; i<b; i++)
{
ret *= a;
ret %= phiN;
}
return ret;
}
```