# A Report On The Implementation Of Elliptic-curve Diffie–Hellman Key Exchange Protocol

Submitted By : Amit Bhowmick(CRS2102)

MTech in Cryptology and Security (2021-23)

Indian Statistical Institute, Kolkata

Date: 8th December 2022

Place: Kolkata

Project supervisor:

Dr Sabyasachi Karati

## Contents

# 1 Abstract

In this project, I did implementations of Elliptic-curve Diffie–Hellman Key Exchange Protocol using P-256 as the underlying curve. Section two of this report contains the notational convention that I used in my code. The next section is all about a small introduction to my total implementation, section four contains the theoretical overview of ECDH, and section five contains each and every implementation detail of my codes followed by the references.

# 2 Notation

In my implementation, I used the following notational convention :

- Every field element is 330 bits binary string out of which 329 bits correspond to the value and the leftmost bit of the string stands for the sign bit. I represent a 330 bits long binary string in a number of base two power thirty. To represent this number I used a long integer array of size eleven.
- Used G, a, b, p, n are same as given in the P-256 curve.

# 3 Introduction

The Elliptic Curve Diffie-Hellman(ECDH), a variant of the Diffie-Hellman, allows two parties that have no prior knowledge of each other to establish a shared secret key over an insecure channel. The Diffie-Hellman works over any group as long as the DLP in the given group is a difficult problem. It is one of the first public key protocols, and it is used to secure a variety of Internet services.

In the implementation of the Elliptic-curve Diffie–Hellman Key Exchange Protocol my code structure is as follows :

- To generate the key in one end the first party generates a 330-bit long binary string such that it's less than n, I convert this string to a number of base two power thirty, suppose the number is A. Now I add G, A times following the elliptic curve group law. This is shared of the Frist party in the public channel.
- In a similar fashion second party also generates a 330-bit long binary string such that it's less than n, I convert this string also to a number of base two power thirty, suppose the number is B. Now I add G, B times following the elliptic curve group law. This is shared of the second party in the channel.

- After obtaining the share of the second party, that is BG, the first party adds BG, A times following the elliptic curve group law and obtains ABG.
- In a similar way the second party computes BAG.
- Then my code printed both ABG and BAG, and I saw both the number are the same, which is the final key obtained from the ECDH algorithm.
- I assumed AB is less than n.

# 4 Elliptic-curve Diffie–Hellman Key Exchange Protocol

This section contains the basic notions of Elliptic-curve followed by the Elliptic-curve Diffie–Hellman Key Exchange Protocol.

## 4.1 Elliptic Curves

There is extensive literature on elliptic curves, as they are both geometric and algebraic objects which arise very naturally in different areas of Mathematics. Recently, they have been used as a theoretical tool in the proof of Fermat's Last Theorem and since then, they are heavily studied in the context of solving Diophantine equations. However, this is outside the scope of this report, Perhaps the most applied context in which elliptic curves have been used so far is Cryptography.

### 4.1.1 Definition

An elliptic curve over Q consists of solutions (x,y) to an equation of the form: $E : Y^2 = X^3 + aX + b$, where a, b $\in$ Q. Moreover, we require that the following quantity, (called the discriminant) is non-zero $\triangle = 4a^3 + 27b^2 \neq 0$.

We think of the elliptic curve E as having a distinguished point called the point at infinity and denoted by $\bigcirc$.

### 4.1.2 Definition

- A point P = (x,y) is a rational point on E if P lies on E (i.e. (x,y) is a solution to the equation describing E) and x,y $\in$ Q.
- We write E(Q) for the rational points together with the point at infinity $\bigcirc$

### 4.1.3 Theorem

**The Group Law**. The set E(Q) forms an abelian group with the binary operation $\oplus$ is defined as follows. Let P = (x1, y1) and Q = (x2,y2) be two rational points, i.e. P, Q

$\in$ E(Q). The line joining P and Q must intersect the curve in a third point, say R. The point R will also be rational, as both the line and the cubic curve E are defined over Q. If then we reflect R to the x-axis we obtain another rational point which we call P $\oplus$ Q.

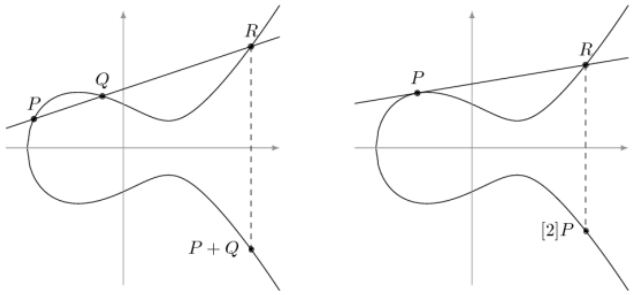


Figure 1: The Group law on an elliptic curve.

### 4.1.4 Remarks

There are some subtleties to address:

- when we add a point P with itself, we take the tangent to the curve at P as shown in the above figure. Such a line will intersect the curve in exactly one other point R. Again, we take the reflection of R in the x-axis to obtain a point which we call [2]P = P $\oplus$ P.
- If the line through P and Q is vertical, (or the tangent through P is vertical), we consider the third point of intersection to be $\bigcirc$. This will get us P $\oplus$ Q = $\bigcirc$, in other words Q is the inverse of P w.r.t. $\oplus$ (or [2]P = O i.e. P has order 2).
- points on a line add up to $\bigcirc$, for example P + Q + R = $\bigcirc$.
- observe the symmetry to the x-axis.

### 4.1.5 Results

Let P = (x,y) $\in$ E(Q). Then, - P = (x,-y).

### 4.1.6 Definition

An elliptic curve over Fp with p $\neq$ 2,3 consists of solutions (x,y) to an equation of the form: $E : Y^2 = X^3 + aX + b$ where a, b $\in$ Fp. Moreover, we require that the following quantity, (called the discriminant) is non-zero $\triangle = 4a^3 + 27b^2 \neq 0 in mod p$.

We think of the elliptic curve E as having a distinguished point called the point at infinity and denoted by $\bigcirc$.

We define P = (x,y) to be an Fp-rational point if P lies on E and x,y ∈ Fp and take E(Fp) to be all of the Fp-rational points, together with the point at infinity.

### 4.1.7 Remarks

These curves will look very different from the ones over Q. The main difference is that E(Fp) is always finite and a discrete set

### 4.1.8 Theorem

**The Group Law** As before, E(Fp) is a group together with $\oplus$, where $\oplus$ is defined exactly as in the rational case.

## 4.2 Cryptography

Cryptography is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents, even if the communication channel is compromised. Here we will discuss the basic notions of the Elliptic-curve Diffie–Hellman Key Exchange Protocol.

### 4.2.1 ECDH

The Diffie-Hellman method proceeds by allowing Alice and Bob to exchange a common secret key via an insecure channel. We have the following steps :

- **initializations** They both have access to the following publicly available information: $E : Y^2 = X^3 + AX + B$ over a fixed field Fp, where p is a prime, a fixed point G ∈ E(Fp) and n the order of G.
- **private key creation** Alice chooses a random element dA ∈ {1,2,...n1} and computes QA := [dA]G := G$\oplus$G$\oplus$G$\cdots\oplus$G (dA times). Alice's secret key is dA and her public key is QA.

  Similarly, Bob chooses a random element dB ∈ {1,2,...n1} and computes QB := [dB]G:= G$\oplus$G$\oplus$G$\cdots\oplus$G (dB times). Bob's secret key is dB and his public key is QB.
- **public key exchange** Alice sends her public key QA to Bob. Bob sends his public key QB to Alice.
- **computing the shared key/secret** Alice computes P := dAQB = dAdBG. Bob computes Q:= dBQA = dBdAG. By commutativity, both Alice and Bob now share P = Q, so for example they can take the x-coordinate of P, denoted x1 ∈ Fp, to be their shared key, which they can then use to encrypt a message.

- **Security** In order for Eve to read the encoded message, she needs to find out x1. She has access to the initial information: E, Fp, G, n. She can intercept QA and QB. However, in order to compute P (and hence x1 ) she needs to know dA or dB. Well, this is equivalent to solving the discrete logarithm problem for G=E(Fp), which we know to be very hard to solve.

# 5 Implementations Details

To implement the above algorithm I construct the following functions :

## 5.1 Integer Addition

This function takes two numbers of base two power thirty and outputs a number of base two power thirty, the code is as follows :

```
void Addition(long int *a, long int *b, long int *d)
{
  long int c[11]={0};
  int i;
  for(i=10; i>=0; i--)
  {
    c[i]+=a[i]+b[i];
    if((c[i]>>30)&1 == 1)
    {
      c[i]=c[i] & 0X3fffffff;
      if(i>0) c[i-1]+=1;
    }
  }
  for(i=0; i<11; i++)
  d[i]=c[i];
}
```

Listing 1: C code

## 5.2 Integer Substraction

This function takes two numbers of base two power thirty and outputs a number of base two power thirty, the code is as follows :

```
void Sub(long int *a, long int *b, long int *c1)
{
  long int b1[11]={0};
  long int c[11] = {0};
  int i, carry;
  for(i=0; i<11; i++) b1[i]=b[i];
```

```
  for(i=0; i<=10; i++)
    b1[i] = b1[i] ^ 0X3fffffff;

  b1[10]+=1;
  carry=(b1[10]>>30)&1;
  b1[10]=b1[10] & 0X3fffffff;
  i=9;
  while(carry != 0 && i>=0)
  {

    b1[i]+=1;
    carry = (b1[i]>>30)&1;
    b1[i]=b1[i] & 0X3fffffff;
    i--;
  }


  for(i=10; i>=0; i--)
  {
    c[i]+=a[i]+b1[i];
    if((c[i]>>30)&1 == 1)
    {
      c[i]=c[i] & 0X3fffffff;
      if(i > 0) c[i-1]+=1;
    }
  }

  for(i=0; i<11; i++)
  c1[i]=c[i];
}
```

Listing 2: C code

## 5.3 Integer Multiplication

This function takes two numbers of base two power thirty and outputs a number of base two power thirty, the code is as follows :

```
void multiplication(long int *a, long int *b, long int *answer)
{
  long int m1[22] = {0}, answer1[22] = {0};
  int i, j, k=21, carry, l;
  for(i=10; i>=0; i--)
  {
    for(j=10; j>=0; j--)
    {
      m1[k] += b[i]*a[j];
```

```
      carry = m1[k]>>30;
      m1[k] = m1[k] & 0x3fffffff;
      m1[k-1]+=carry;
      k--;
    }
    for(l=21; l>=0; l--)
    {
      answer1[l]+=m1[l];
      if((answer1[l]>>30)&1 == 1)
      {
        answer1[l]=answer1[l] & 0X3fffffff;
        if(l>0) answer1[l-1]+=1;
      }
    }


    for(l=0; l<22; l++)
    m1[l]=0;

    k=i+10;
  }
  for(i=0; i<22; i++)
  answer[i]=answer1[i];

}
```

Listing 3: C code

## 5.4 Barrett Reduction

This function takes a number, say a, of base two power thirty and outputs a number of base two power thirty, say c, such that c = a mod p , the code is as follows :

```
void Barrett(long int *x, long int *r1)
{
  long int t[11]={0, 16384, 4095, 1073741823, 1073741567, 1073741759,
   1073741807, 1073741820, 0, 0, 805306368};
  int k=9,i=0;
  long int p[11]={0, 0, 65535, 1073725440, 4096, 0, 0, 63, 1073741823,
   1073741823, 1073741823};
  long int q[22]={0}, r[11]={0};
  long int b_k[11]={0};
  long int x1[22]={0};
  for(i=0; i<22; i++) x1[i]=x[i];
  b_k[0]=1;
  for(i=21; i>=8; i--)
  {
```

```
    q[i]=x1[i-8];
  }
  multiplication(&q[11], &t[0], &q[0]);


  for(i=21; i>=10; i--)
  {
    q[i]=q[i-10];
  }
  for(i=9; i>=0; i--)
  q[i]=0;

  for(i=0; i<=11; i++)
  x1[i]=0;

  multiplication(&q[11], &p[0], &q[0]);

  for(i=0; i<=11; i++)
  q[i]=0;

  Sub(&x1[11], &q[11], &r[0]);

  if(((r[0]>>29) & 1)==1)
  {
    Addition(&r[0], &b_k[0], &r[0]);
  }

  if(compare(&r[0], &p[0]))
  {
    Sub(&r[0], &p[0], &r[0]);
  }

  if(compare(&r[0], &p[0]))
  {
    Sub(&r[0], &p[0], &r[0]);
  }

  for(i=0; i<11; i++)
  r1[i]=r[i];
}
```

Listing 4: C code

## 5.5 Finding Inverse in Finite Field

This function takes a number, say a, of base two power thirty and outputs a number of base two power thirty, say b, such that ab = 1 mod p, the code is as follows :

```
void inverse(long int *x, long int *x1)
{
  long int p[11]={0, 0, 65535, 1073725440, 4096, 0, 0, 63, 1073741823,
   1073741823, 1073741823}, z[11]={0}, z1[22]={0}, p1[11]={0};
  z[10]=1;
  p1[10]=2;
  Sub(&p[0], &p1[0], &p1[0]);
  int i=0,k=0,j=29;
  while (i <330)
  {
    multiplication(&z[0], &z[0], &z1[0]);
    Barrett(&z1[0], &z[0]);
    if((p1[k]>>(j--)) & 1)
    {
      multiplication(&z[0], &x[0], &z1[0]);
      Barrett(&z1[0], &z[0]);
    }
    if(j==-1)
    {
      k++;
      j = 29;
    }
    i++;
  }

  for(i=0; i<11; i++)
  x1[i]=z[i];

}
```

Listing 5: C code

## 5.6 Finite Field Addition

This function takes two numbers, say a, b of base two power thirty and outputs a number of base two power thirty, say c, such that c = (a+b) mod p, the code is as follows :

```
void F_add(long int *a, long int *b, long int *c)
{
  int i;
  long int p[11]={0, 0, 65535, 1073725440, 4096, 0, 0, 63, 1073741823,
   1073741823, 1073741823};
  long int temp1[22]={0}, temp[11]={0};
  Addition(a, b, temp);
  for(i=11; i<22; i++) temp1[i]=temp[i-11];
  Barrett(temp1, c);
```

```
}
```

Listing 6: C code

## 5.7 Finite Field Substraction

This function takes two numbers, say a, b of base two power thirty and outputs a number of base two power thirty, say c, such that c = (a-b) mod p , the code is as follows :

```
void F_sub(long int *a, long int *b, long int *c)
{
  int i;
  long int p[11]={0, 0, 65535, 1073725440, 4096, 0, 0, 63, 1073741823,
   1073741823, 1073741823};
  long int temp1[22]={0}, temp[11]={0};
  Sub(a, b, temp);
  if(((temp[0])>>29) & 1)
  {
    Addition(temp, p, temp);
  }
  for(i=11; i<22; i++) temp1[i]=temp[i-11];
  Barrett(temp1, c);
}
```

Listing 7: C code

## 5.8 Finite field Multiplication

This function takes two numbers, say a, b of base two power thirty and outputs a number of base two power thirty, say c, such that c = (a * b) mod p, the code is as follows :

```
void F_mul(long int *a, long int *b, long int *c)
{
  long int temp1[22]={0}, temp[11]={0};
  long int p[11]={0, 0, 65535, 1073725440, 4096, 0, 0, 63, 1073741823,
   1073741823, 1073741823};
  multiplication(a, b, temp1);
  Barrett(temp1, c);
}
```

Listing 8: C code

## 5.9 Elliptiic Curve Addition

This function takes two points on the elliptic curve, say a, b, as an input and outputs (a + b) by following the group law of the elliptic curve, here '+' is a binary operation as defined in the elliptic curve group, the code is as follows :

```
void ecc_add(long int *a1, long int *a2, long int *b1, long int *b2,
    long int *cc1, long int *cc2)
{
  long int A[11]={0, 0, 65535, 1073725440, 4096, 0, 0, 63, 1073741823,
   1073741823, 1073741820};
  long int B[11]={0, 0, 23238, 225847950, 691960638, 787830234,
   411483237, 121744435, 87756643, 792260856, 668098635};
  long int p[11]={0, 0, 65535, 1073725440, 4096, 0, 0, 63, 1073741823,
   1073741823, 1073741823};
  int i, tem=0;

  long int temp[11], temp1[22], temp2[11], lambda[11];
  long int c1[11], c2[11];


  for(i=0; i<11; i++)
  {
    if(a1[i] != b1[i])
    {
      tem=1;
      break;
    }
  }

  if(tem==0)
  {
    for(i=0; i<11; i++)
    {
      if(a2[i] != b2[i])
      {
        tem=1;
        break;
      }
    }

  }

  if(tem ==0)
  {

    F_mul(a1, a1, temp);
    F_add(temp, temp, temp2);
    F_add(temp2, temp, temp);
    F_add(temp, A, temp);
    F_add(a2, a2, temp2);
    inverse(temp2, temp2);
    F_mul(temp, temp2, lambda);
```

```
  }

  if(tem==1)
  {

    F_sub(b2, a2, temp2);
    F_sub(b1, a1, temp);
    inverse(temp, temp);
    F_mul(temp, temp2, lambda);
  }
  show(lambda, 11);
  printf("\n\n");

  F_mul(lambda, lambda, temp2);
  F_add(a1, b1, temp);
  F_sub(temp2, temp, c1);

  F_sub(a1, c1, temp);
  F_mul(temp, lambda, temp2);
  F_sub(temp2, a2, c2);

  for(i=0; i<11; i++)
  {
    cc1[i] = c1[i];
    cc2[i] = c2[i];
  }

}
```

Listing 9: C code

## 5.10 Elliptiic Curve Scaler Multiplication

This function takes one point on the elliptic curve, say a, and a scaler say c, as an input and outputs ca by adding a, c times by following the group law of the elliptic curve point addition, the code is as follows :

```
void ecc_scaler_mult(long int *a1, long int *a2, long int *n, long int *
    b1, long int *b2)
{

    long int ans1[11], ans2[11];
  int i=0, j=29, k=0, l;
  while((n[k]>>j) & 1 == 0)
  {
      j--;
      if(j== -1)
```

```
      {
    k++;
    j=29;
      }
      i++;
      if(k==11) break;
  }
  j--; if(j == -1) { k++; j=29; } i++;
  ecc_add(a1, a2, a1, a2, ans1, ans2);


  while(i<330)
  {
      if((n[k]>>(j--)) & 1)
      {
    ecc_add(ans1, ans2, a1, a2, ans1, ans2);

      }
      ecc_add(ans1, ans2, ans1, ans2, ans1, ans2);

      if(j==-1)
      {
      k++;
      j = 29;
      }
      i++;
  }
  for(i=0; i<11; i++) b1[i]=ans1[i];
  for(i=0; i<11; i++) b2[i]=ans2[i];

}
```

Listing 10: C code

# 6 References

- Elliptic Curves in Cryptography
- Barrett reduction