30 April | 15

Elliptic Curve Cryptography

Implementation in C++

Pooja Garg

This manual describes how to configure and use this implementation.

Table of Contents

A quick start

To configure

1- **Install gmp**

Download gmp library from here- https://gmplib.org/ - DOWNLOAD.

Please note that we enable c++ also.

Steps for installation on a Unix-like system-

i. Unzip gmp-6.0.0a.tar.bz2
ii. In the terminal, go to parent directory of gmp-6.0.0
iii. ./configure --enable-cxx
iv. make
v. make check
vi. sudo make install

2- Install Givaro library

Download givaro 3.8.0 from here- https://forge.imag.fr/frs/?group_id=187

Replace these two files- **givpoly1io.inl, givpoly1io.inl** at "*givaro-3.8.0/src/library/poly1/*" with those in folder "*assets*" of ECC.zip

Steps for installation on a Unix-like system-

i. Unzip givaro-3.8.0.tar
ii. In the terminal, go to parent directory of givaro-3.8.0
iii. ./configure --prefix=/tmp/givaro-exec
iv. make; make install

Now, we are ready to go.

Content of ECC.zip

1. **src/**
   a. ellipticCurve.h, ellipticCurve.cpp
   Definition of data structures- ExtensionField, ecPoint, ellipticCurve, ellipticCurveFq
   b.
   c. ECC.h, attacksECC.cpp
   Definition of 2 elliptic curve attacks- Pollard Pho, Pohlig Hellman
   d. ECC.h, ECC.cpp
      - Definition of data structures- Key, SignatureECDSA, SignatureELGAMAL
      - Definition of Elliptic Curve Cryptography systems- Diffie Hellman Key exchange; Digital Signature Schemes- ELGAMAL, ECDSA; Public Key Encryption- ELGAMAL

2. **examples/**
   Contains examples for finite field arithmetic, elliptic curve arithmetic, attacks, elliptic curve cryptography systems. For each example, there is a driver file with extension .cpp, a script file with extension .sh, and an input file with name as that of .cpp file and appended by I.

3. **assets/**

Contains two files that modifies Givaro Library for easy input of a polynomial

4. **manual.pdf**

This file

To compile and run

Before compilation using script file, modify the environment variable 'HOME' in that script file to the location where you unzipped the givaro library in your system.

That is change
HOME=Users/poojagarg/Downloads/c_library/givaro-3.8.0
to
HOME="Location of givaro library in your system"

Steps:

1- Compile source files using script file- compile.sh in the directory src.
   a. Open terminal in the parent directory of ECC
   b. cd src
   c. ./compile.sh

2- Compile example file 'X.cpp' using script file- X.sh in the directory examples
   a. cd examples
   b. ./X.sh
   c. To run: ./X < XI

For example:

- To compile ECarithmetic.cpp,
  ./ECarithmetic.sh
- To run ECarithmetic,
  ./ECarithmetic < ECarithmeticI

Files that end with I are input files for the example programs.

For clarity purposes, all such commands are given in examples/commands.sh
Also, you can run all the examples together with the following commands-

1. chmod +x commands.sh
2. ./commands.sh

To give our own input while the program runs, we simply run ./ECarithmetic.

About Givaro Library

In the joint CNRS-INRIA / INPG-UJF project APACHE, Givaro is a C++ library for arithmetic and algebraic computations. Its main features are implementations of the basic arithmetic of many mathematical entities: Primes fields, Extensions Fields, Finite Fields, Finite Rings, Polynomials, Algebraic numbers, Arbitrary precision integers and rationals(C++ wrappers over gmp)

We use following data structures and associated operations of Givaro library-

1. ZpzDom<Integer>: For representation of prime field Zp, where p is prime
2. Poly1Dom: For representation of polynomials that belong to Fp[x] , where p is prime and using associated operations
3. Integer: For representation of integer of arbitrary precision(which uses gmp internally) and using associated operations.

Finite Field Arithmetic

Class "ExtensionField" represents a finite field, Fp^m, where p is prime and m>0. Size of finite field is p^m. User gives an irreducible polynomial of degree 'm' and each element of field is represented as a polynomial in Fp[x] of degree < m. Arithmetic operations defined on these elements use the operations directly from Givaro library.

A slight change is made in Givaro library to input a polynomial. Now, there are 2 ways-
Let Fq[X] be an object of class ExtensionField, and A be some element of Fq

1- Fq[X].readElement(A,false)
   Enter degree of polynomial, followed by pair (index, co-efficient). Enter -1 to stop.
   For example, to input: X^506+3*X^2+2, write
   506,506,1,2,3,0,2,-1
   This method is suitable for such kind of polynomials that have sparse number of terms but huge degree.
2- Fq[X].readElement(A) or Fq[X].readElement(A,true)
   Enter degree of polynomial, followed by all the coefficients, starting from highest degree.
   For example, to input: 2*X^3+7*x, write
   3,2,0,7,0

Check src/ellipticCurve.h to know what all operations are there and examples/FiniteFieldArithmetic.cpp to see how to do finite field arithmetic.

Elliptic Curve Arithmetic

Class **ecPoint** represents a point on Elliptic Curve.
It's data members are

Identity: bool

x, y: Element of Extension Field

If a point is identity, identity is set to true, x and y are arbitrary. Otherwise, identity is false and point is (x,y)

Class **ellipticCurve** represents Elliptic Curve over extension field Fp^m, pointed by Kptr, defined by co-efficient a,b,c with 3 possible equations:

type 0: E/K, char(K)!=2: $y^2 = x^3 + ax + b$; hence c is not defined

type 1: non-supersingular $E/F2^m$: $y^2 = x^3 + ax + b$,

type 2: supersingular $E/F2^m$: $y^2 + cy = x^3 + ax + b$

To input an elliptic curve, please check the constructor ellipticCurve()
Prime p is of arbitrary precision, but m can not be more than a long because element of extension field is represented in polynomial basis form, though type of m is defined to be Integer for handy calculations.

To check examples program for Elliptic Curve of type 1 and type 2, use input file that ends with _2_1_I and _2_2_I for type 1 and type 2 respectively.

Class **ellipticCurveFq** represents abelian group E(Fp^(m*d)), with Elliptic Curve defined over extension field Fp^m. This class also defines the operation on this group like addition of points, inverse of a point.

To input an object E(Fq) of type ellipticCurveFq, we can either use an object of ellipticCurve over field K such that K is contained in Fq, or we can create an entirely new object.
To input, please check the constructor ellipticCurveFq() and ellipticCurveFq(ellipticCurve*)

Check src/ellipticCurve.h to know what all operations are there and examples/ECarithmetic.cpp to see how to do arithmetic of E(Fq).

Attacks

src/attacksECC.h declare 2 attacks- Pollard rho attack, Pohlig Hellman Attack.

Function Prototype:

1- void pollardRho(Integer& result, ecPoint& P, ecPoint& Q,Integer n,ellipticCurveFq& E_Fq,int L);

It computes result such that Q=result*P.
Assumption: n is prime, n=order(P) in E(Fq) represented by E_Fq, L is number of partition functions

2- void pohligHellman(Integer& result, ecPoint& P, ecPoint& Q,Integer n,ellipticCurveFq& E_Fq);

It computes result such that Q=result*P.
n=order(P) in E(Fq) represented by E_Fq.
It uses function pollardRho to compute each smaller instance of ECDLP

examples/attacksECCdriver.cpp discusses how to use pohlig hellman method.
Pollard rho is similar, provided order(P) is prime.

Note: Since Pollard Rho is a probabilistic algorithm, and Pohlig Hellman also uses Pollard rho to solve it's smaller instance, attack might fail to give any answer.

In those scenario, you may try-
1- to change the partition function defined as

```
/*result=H(input) where 0=<result<L. used by function pollardRho to compute the partition*/
Integer& H(Integer& result,ecPoint& input, int L, ellipticCurveFq& E_Fq);
```

2- choose your own initial random number c1, d1. For that you will have to change the definition of pollard rho method.

Pollard Pho method

Algorithm used- As given in book "Guide to Elliptic Curve Cryptography" by *Darrel Hankerson, Alfred Menezes, Scott Vanstone*

Algorithm: Pollard's rho algorithm for the ECDLP (single processor)

INPUT: $P \in E(\mathrm{F}_q)$ of prime order n, $Q \in \langle P \rangle$.

OUTPUT: The discrete logarithm $l = \log_P Q$.

1 Select the number L of branches (e.g., $L = 16$ or $L = 32$).

2 Select a partition function $H : \langle P \rangle \rightarrow \{1,2,\ldots, L\}$.

3 For j from 1 to L do

  3.1 Select $a_j, b_j \in_R [0, n-1]$.

  3.2 Compute $R_j = a_j P + b_j Q$.

4 Select $c', d' \in_R [0, n-1]$ and compute $X' = c'P + d'Q$.

5 Set $X'' \leftarrow X'$, $c'' \leftarrow c'$, $d'' \leftarrow d'$.

6 Repeat the following:

  6.1 Compute $j = H(X')$. Set $X' \leftarrow X' + R_j$, $c' \leftarrow c' + a_j \bmod n$, $d' \leftarrow d' + b_j \bmod n$.

  6.2 For i from 1 to 2 do

    Compute $j = H(X'')$.

    Set $X'' \leftarrow X'' + R_j$, $c'' \leftarrow c'' + a_j \bmod n$, $d'' \leftarrow d'' + b_j \bmod n$.

Until $X' = X''$.

7 If $d' = d''$ then return("failure");

Else compute $l = (c' - c'')(d'' - d')^{-1} \bmod n$ and return(l).

Pohlig Hellman

Algorithm: Pohlig Hellman algorithm for the ECDLP

INPUT: $P \in E(\mathbb{F}_q)$ of order n, $Q \in \langle P \rangle$.

OUTPUT: The discrete logarithm $l = \log_P Q$.

1 Factor n=$\prod p$^e

2 For each prime factor p,

Compute Po=(n/p)P. Hence, Order of Po=p.

Compute Qo=(n/p)Q

Use Pollard rho to solve Qo=Zo*Po

For t=1 to e-1

Compute Q_t = $(n/p^{(t+1)})*(Q-z_0P-z_1pP-z_2p^2P-\cdots-z_{t-1}p^{t-1}P)$

Use Pollard rho to compute $z_t = \log_{P_0} Q_t$

$l_i = z_0 + z_1 p + z_2 p^2 + \cdots + z_{e-1} p^{e-1}$

3 Use Chinese Remainder theorem to compute l, such that $l \equiv l_i \pmod{p_i^{e_i}}$

Elliptic Curve Cryptosystems

Algorithms are implemented as given in the book “Elliptic Curves Number Theory And Cryptography” by *Lawrence C. Washington*
src/ECC.h declares the following ECC systems.

Diffie-Hellman Key Exchange

Input: basepoint P in E(Fq), n=order(P)
Output: Ka- public Key with Alice, Kb- public Key with Bob
a and b: secret integer by Alice and Bob respectively

Digital Signature Schemes

1 ECDSA

2 ElGamal

*Self explanatory

Public Key Encryption: ElGamal

ElGamal Encryption

Input: message represented as a Point in E(Fq), basepoint P in E(Fq) represented by E_Fq, n=order(P)
Output: Encrypted message {M1,M2}

ElGamal Decryption

Input: Encrypted message {M1,M2}, basepoint P in E(Fq) represented by E_Fq, n=order(P)
Output: message represented as a Point in E(Fq)

References

Books referred:

1. "Guide to Elliptic Curve Cryptography" by *Darrel Hankerson, Alfred Menezes, Scott Vanstone*
2. "Elliptic Curves Number Theory And Cryptography" by *Lawrence C. Washington*

Libraries Used:

1. gmp, with C++ support
2. Givaro