Course Project
EECS 475
Fall 2014
Implementing an Elliptic Curve Cryptosystem

November 5, 2014

1 Introduction

For this project you will use the GMP multiprecision C/C++ library to implement a variant of the Elliptic Curve Cryptosystem called the simplified Elliptic Curve Integrated Encryption Scheme with point compression. The due date for the project is 5 pm, Thursday December 4.

From class you know that the elliptic curve $E_p(a, b)$ is the set of points (x, y) satisfying the equation

$$y^2 = x^3 + ax + b,$$

where x, y, a, b are in the field GF(p); in addition, $E_p(a, b)$ contains one other point, the *point at infinity* $\mathcal{O}$. (The equation above is correct as long as p is not a power of 2 or 3. We also require that $4a^3 + 27b^2 \neq 0$ for the definitions below to work.)

If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ are two points on $E_p(a, b)$ (and neither is $\mathcal{O}$), the the sum of these points $R = P + Q$ is the point $R = (x_R, y_R)$ defined as follows.

1. If $P \neq Q$ and $x_P \neq x_Q$, define

$$\begin{aligned} \Delta &= (y_Q - y_P)(x_Q - x_P)^{-1} \\ x_R &= \Delta^2 - x_P - x_Q \\ y_R &= -y_P + \Delta(x_P - x_R) \end{aligned}$$

2. If $P = Q$ and $2y_p \neq 0$, define

$$\begin{aligned} \Delta &= (3x_P^2 + a)(2y_P)^{-1} \\ x_R &= \Delta^2 - 2x_P \\ y_R &= -y_P + \Delta(x_P - x_R) \end{aligned}$$

3. In other cases, $P + Q = \mathcal{O}$.

If either P or Q (or both) is $\mathcal{O}$, we define $P + \mathcal{O} = \mathcal{O} + P = P$.

As we have seen, an elliptic curve with the binary operation + is a group. It may not be a cyclic group, but a deep result (that we will not prove in this course) states that it is either cyclic or is the Cartesian product of two cyclic groups. The elliptic curve group you will use for this assignment is cyclic.

Most researchers believe that the discrete log problem is hard for cyclic elliptic curve groups (at least for some parameters p, a, b) and, hence, they are suitable for cryptographic applications such a public key cryptography, key distribution, and digital signatures.

Suppose $E_p(a, b)$ is a cyclic group with generator $G = (x_G, y_G)$ and $|E_p(a, b)| = N$. Since we use additive notation, rather than the usual multiplicative notation, we use multiples of points P on the curve, rather than powers. That is, for any nonnegative integer X, we define

$$X \cdot P = \underbrace{P + P + \cdots + P}_{X \text{ addends}}.$$

Suppose $R = (x_R, y_R)$ is a point on $E_p(a, b)$. Then so is $-R = (x_R, p - y_R)$, since $p - y_R \equiv -y_R \pmod{p}$ and $y_P^2 \equiv (-y_P)^2 \pmod{p}$. There are no other values for y other than these two such that (x_R, y) is a point on the curve. Now the sum of y_R and $p - y_R$ is p, an odd number, so one of these values is even and the other is odd. This gives us a method to compress the representation of point on an elliptic curve. Rather than storing a pair (x_R, y_R), let b_R be just the bit ($y_R \bmod 2$) and store the *compressed point* $\gamma(R) = x_R b_R$, the concatenation of the binary representation of x_R with the bit b_R. (For some purposes we also view this as the integer $2 \cdot X_R + b_R$.) $\gamma(R)$ uses slightly more than half number of bits needed to explicitly store the coordinates of R.

Can we recover $R = (x_R, y_R)$ from $\gamma(R) = x_R b_R$ in a reasonable amount of time? Since

$$y_R^2 = x_R^3 + a x_R + b$$

from the elliptic curve equation, this means we must be able to find square roots efficiently in $GF(p)$. It turns out that if p is a prime and $p \equiv 3 \pmod{4}$, computing square roots is easy. In this case, if $z \in GF(p)$ is a *quadratic residue* (i.e., has square roots), one of its square roots is

$$z^{(p+1)/4} \bmod p$$

and the other is

$$p - z^{(p+1)/4} \bmod p.$$

One of these is even and the other is odd, so take whichever one has b_R as its low order bit. This allows us to compute the *decompression function* $\delta(x_R b_R) = (x_R, y_R)$.

There are three algorithms associated with the Elliptic Curve Cryptosystem: the key generation algorithm, which is a randomized algorithm generating a private key/public key pair; the encryption algorithm, which is a randomized algorithm for computing a ciphertext C from a plaintext M using the public key; and the decryption algorithm for obtaining the plaintext M for a ciphertext C using the private key. In this version, we will assume that $p < 2^k$ but that $2^k - p$ is very small. Here $k + 1$ will be the block length.

Key Generation Algorithm

1. Generate a random integer Y, $0 \le Y \le N - 1$.
2. Compute $P = Y \cdot G = (x_P, y_P)$ using repeated squaring. (Since we are using additive notation here and below, we are really using repeated doubling.)

Return: Public key $(E_p(a, b), G, P)$ and private key $(E_p(a, b), G, Y)$.

Encryption Algorithm

Given: plaintext block M (a bit string of length $k + 1$) and the public key.

1. Generate a random integer X, $0 \le X \le N - 1$.

2. Compute $Q = X \cdot G = (x_Q, y_Q)$ and $R = X \cdot P = (X \cdot Y) \cdot G = (x_R, y_R)$ using repeated squaring.

3. Set $C_1 = \gamma(Q)$ and $C_2 = M \oplus \gamma(R)$. ($\oplus$ is bitwise XOR).

Return: Ciphertext $C = (C_1, C_2)$.
Notice that this definition differs from the usual El Gamal encryption algorithm where M would be an element of the group. The difficulty with doing this is that not every bit string of the appropriate length is actually a group element (that is, a point on the elliptic curve). The reason the definition above works is that $\gamma(R)$ looks random, so we use it in the same way we would use a random bit string in the one-time pad. Another version of the Elliptic Curve Cryptosystem (which you will *not* use in this project) defines

$$C_2 = M \cdot \gamma(R) \ (mod\ p).$$

In this case, the length of the block M is has k rather than $k+1$ bits and we view it as an element of $GF(p)$.

Decryption Algorithm

Given: Ciphertext (C_1, C_2) and the private key.

1. Compute $R = Y \cdot \delta(C_1) = (x_R, y_R)$ using the square root algorithm and repeated squaring.

2. Set $M = C_2 \oplus \gamma(R)$.

Return: Plaintext M.

Fields and Curves Used for this Assignment.

The elliptic curve you will be using is called w-255-mers. It is one of the curves investigated by Joppe W. Bos, Craig Costello, Patrick Longa and Michael Naehrig in a paper titled "Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis." This is Report 2014/130 in the Cryptology ePrint Archive at: http://eprint.iacr.org/2014/130.

You will be working over the field GF(p), where the prime $p = 2^{255} - 765$. As remarked in class, we usually pick prime powers that are nearly a power of 2, and you can see that p is very close to 2^{255}. The binary representation of $2^{255} - 1$ is 255 consecutive 1's. Subtracting 764 from this simply flips a few of the lower order bits.) In hexadecimal, p is

`7FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFD03`.

The probability that a randomly chosen 255-bit string represents an integer less than p is $p/2^{255} = 1 - 765/2^{255}$, which is very close to 1. Thus, p is a very convenient modulus to use with 256-bit blocks (since M has one more bit than p). In decimal, p is

`57896044618658097711785492504343953926634992332820282019728792003956564819203`

The elliptic curve is $y^2 = x^3 + ax + b$, where $a = -3$ and b is

`−51BD`

in hexadecimal, or

`−20925`

in decimal. The number of points on the curve is

`7FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 864A3828 3AD2B3DF AB8FAC98 3C594AEB`

in hexadecimal, or

```
57896044618658097711785492504343953926473211886304323019964499781607006751467
```

in decimal. This is a prime number so the group is cyclic.

The generator we use is (x_G, y_G), where x_G is

```
1D
```

and y_G is

```
404B36E1 A74DEF81 37F29FA5 9849D32E 32E4337C 7ED795BB 8F8F8C42 55313DB1
```

in hexadecimal, or x_G is

```
29
```

and y_G is

```
290809146170973680206379888020415280234770502728059337185551585749631637416 17
```

in decimal.

2 Implementation Using GMP

GMP is an abbreviation for GNU Multiprecision Library. It is a library of C functions for arbitrary precision (or multiprecision) arithmetic on integers, rational numbers, and floating-point numbers. There is an interface that allows you to use these functions in C++ programs. The documentation for GMP is at

```
http:http://gmplib.org/manual/
```

You will not need to know very much about GMP, but you should look over the sections GMP Basics, Integer Functions, and C++ Class Interface.

For the assignment you will need to find a computer or network with a full installation of the g++ compiler. CAEN machines running Linux are your best bet. GMP is included with the g++ installation.

You then need to download three files from the course CTools web site:

```
ec.cpp
ec_ops.h
ec_ops.cpp
```

These files already contain a number of the include statements, definitions, and functions you will need to use GMP in implementing and elliptic curve cryptosystem. The file `ec_ops.h` contains a number parameter definitions for some of the constants mentioned in the previous sections and three class definition headers for the classes `Zp, ECpoint`, and `ECsystem`. `Zp` is the class for doing arithmetic in GF(p). Notice that the parameters of type `mpz_class` are multiprecision integers, meaning they may be so long that they must be stored in several words of memory. `ECpoint` is the class for defining an elliptic curve point (essentially a boolean indicating whether it is the point at infinity, and, if not, a pair (x, y) where x and y are both `Zp` objects). `ECsystem` is the class for elliptic curve cryptosystems; it supports encryption and decryption. The file `ec_ops.cpp` has definitions for the methods used by these classes. You will have to take some time to figure out what these methods do. Some are, for example, definitions of overloaded operators so you can add `Zp` values, use them in standard input and output, etc.

The file `ec.cpp` is the only file that is not complete. The definitions of seven functions have been left blank. You will have to write these parts. The methods you will have to implement are the following.

1. For class `Zp` you will implement a method `inverse()` that uses the Extended Euclidean Algorithm to return the inverse mod PRIME.
2. For class `Zp` you will implement a method `power()` that computes a power mod PRIME using repeated squaring.
3. For class `ECpoint` you will define an operation + (clearly overloading) that adds another point.
4. For class `ECpoint` you will implement a method `repeatSum` that adds a point p to itself v times.
5. For the class `ECpoint` you will implement a method `pointDecompress` for decompressing a compressed point.
6. For class `ECsystem` you will implement the `encrypt` operation whose parameters are the public key `pb` (a point on the elliptic curve) and a plaintext `pm`.
7. For class `ECsystem` you will implement the `decrypt` operation.

The strategy you should pursue is to implement these methods one at a time in the order given. You may want to comment out the parts you have not yet implemented and rewrite `main` temporarily to test your code.

To compile your program and link to the GMP library you will type

```
g++ ec.cpp -lgmpxx -lgmp
```

You will not need to write a lot of code to complete this assignment, but there are some conceptual problems you will have to overcome, so you should not put this off to the final few days. If you are not an experienced C++ programmer, you may need to consult the GSI and professor on some of the finer points of object oriented programming.