Concurrent Elliptic Curve Cryptography

Philip Moss Robinson
Advisor : Dr. David Bover
Computer Science Department
Western Washington University

Abstract

Elliptic Curve Cryptography (ECC) is a means of using properties inherent to Elliptic Curves to hide information such that it is computationally exhaustive/impractical to breach. For this project we produced a practical ECC library-module with the goal of using the concurrent paradigms built into the **Erlang** programming language.

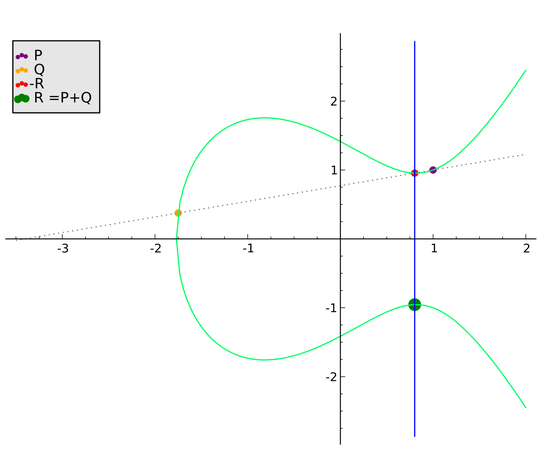
A consequence to most ciphers which can be parallelized is vulnerability to frequency attacks. This non-sequential implementation adopts arbitrary N-Bit block sizes so the system is not permeable to addressable-tupple frequency attacks. The working solution was derived using *Koblitz*s point mapping algorithm and *Massey-Omura* encryption methods.

Elliptic Curves

The fields we work in with, Elliptic Curves with non-zero discriminants, are closed over a point addition operator $+_E$ and are of the form:

$$\{(x, y) \in \mathbf{F} | y^2 + a_1xy + a_3y = x^3 + a_2x^2 + Ax + B\} \cup \{(\infty, \infty)\}$$

The point $\Omega = (\infty, \infty)$ represents the identity element for this field. Point addition is graphically represented bellow.



The graphics of the fields above are over **R** or **Q**, however, we want our points in **Zp** because it is easy to interpret data as integers.
Our homomorphism $H : E_{\mathbf{Q}} \to E_{\mathbf{Zp}}$

$$(\left[\frac{m}{n}, \frac{r}{s}\right] \in E_{\mathbf{Q}})$$

$$\left(\frac{m}{n} \equiv X(mod\ P)\right) \wedge \left(\frac{r}{s} \equiv Y(mod\ P)\right) \Rightarrow (X, Y) \in E_{\mathbf{Zp}}$$

Massey-Omura *Encryption*

1. Data = $Where - is - my - Cabbage$
2. Hex = 57686572652d69732d6d792d436162626167652020
3. Split = $['576865','72652d','69732d','6d792d','436162','626167','652020']$
4. Ints = $[5728357, 7497005, 6910765, 7174445, 4415842, 6447463, 6627360]$

Public Info

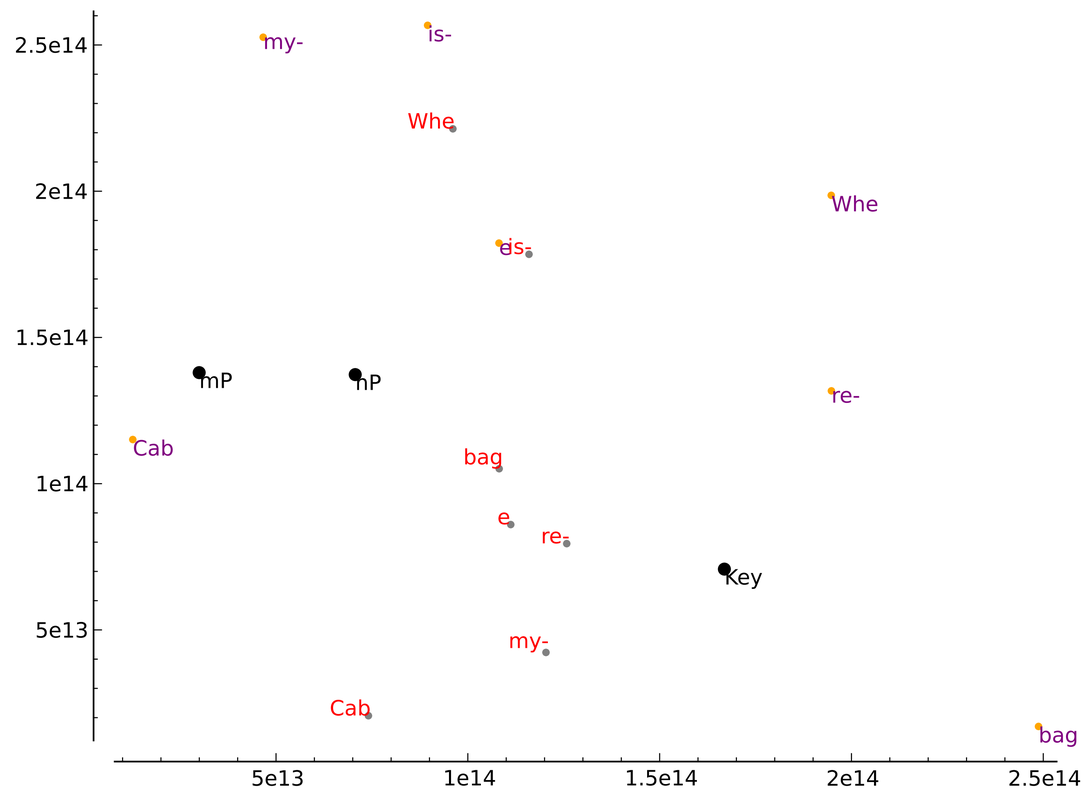
- $y^2 = x^3 + 2x + 2$
- P = $(36165185351809, 107306345876646)$
- Group = $\mathbf{F}_{281474976710731}$

PersonA

1. N = 236854279155507 is a random integer
2. N*P = $(70644839002251, 137298511691771)$
3. Sends N*P
4. Rec. M*P = $(29969551944534, 137970218947240)$
5. K = N*MP = $(166875418105473, 70783621696893)$

PersonB

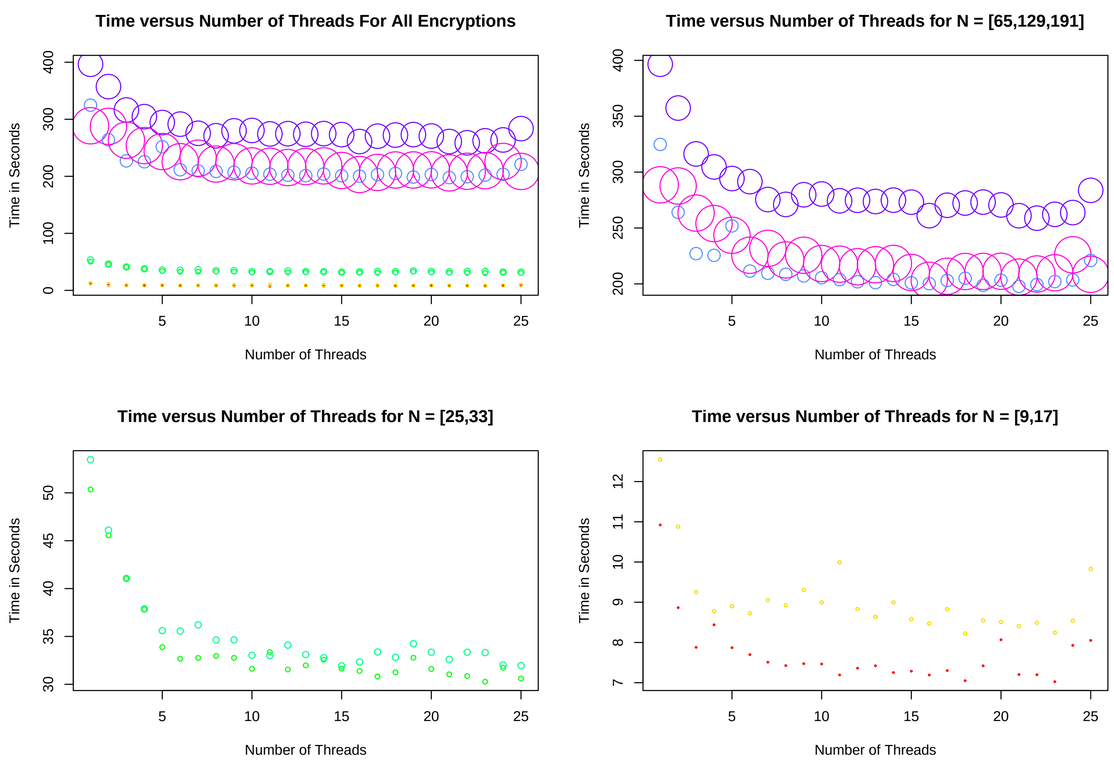
1. M = 112843882915608 is a random integer
2. M*P = $(29969551944534, 137970218947240)$
3. Rec. N*P = $(70644839002251, 137298511691771)$
4. Sends M*P
5. K = M*NP = $(166875418105473, 70783621696893)$



Gray Points represent **decrypted** message
Orange Points represent **encrypted** message

As you can see the message is scattered in a seemingly random order. The time complexity of breaking this system is equivalent to the **RSA** large primes problem; however, **ECC**'s advantage is its more reasonable growth space complexity.

Concurrency in Encryption



The chosen encryption and point mapping methods produce points from integers, and are not altered by prior generated points. This invites an *Embarrassingly Parallel* implementation.

The greatest concern with techniques like this is frequency attacks. If points do not effect their neighbors then it is simple to compare it's frequency with those of known strings. The cheap means of addressing this used is encrypting on block sizes not divisible by 8 (and there fore not character divisible). This emulates the desired behavior that renders frequency attacks useless.

In the Image above the point/circle size represents the N-Bit encryption, and the colors are constant over a listed N-bit encryption. As is quite visible there is most definitely a time difference found, especially for $[65, 129, 191]$-Bit encryption. As expected, after a given number of threads, there is no longer any speedup.