Chapter 1

Prathamesh Dhake 190070048 | Atharva Raut 190070050 | Jaideep Chawla 190110030

March 20, 2023

1 Work Until Now

Not applicable since this is the first week.

2 Work in Week 1

We worked to familiarize ourselves with the elliptic curve cryptography and scalar multiplication algorithms. We studied existing parallelization techniques such as the parallelizing Double and Add method for scalar multiplication.

Let k be a n-bit scalar, and P be the point to be multiplied

Algorithm 1: Double and Add

Input: P, $k = b_{n-1} \parallel b_{n-2} \cdots \parallel b_1 \parallel b_0$
Output: $Q = kP$
$Q = 0$
for $i = 0$ *to* $\ell - 1$ **do**
  $Q = 2Q$
  if $k_i = 1$ **then**
    $Q = Q + P$
return Q

For parallelizing it, we assume we have 2^m processors, with $2^m \leq n$. We break k into 2^m parts,

$$k = k_{2^m}^m \parallel k_{2^m-1}^m \cdots \parallel k_2^m \parallel k_1^m$$

Then compute the smaller products in parallel,

$$k_i^m P \Rightarrow Q_i^m$$

From these smaller products, we can then recursively recombine the Q values to obtain kP. For prime curves, we recombine via doubling,

$$Q_{\frac{j}{2}}^i = 2^{|k_{j-1}^{i+1}|} Q_j^{i+1} + Q_{j-1}^{i+1}$$

Thus, the algorithm for parallelizing can be summarized to be,

Algorithm 2: Parallel Double and Add [1]

Input: P, $k = k_{2^m}^m \parallel k_{2^m-1}^m \cdots \parallel k_2^m \parallel k_1^m$
Output: $Q = kP$
$Q = 0$
for $i = 0$ *to* $2^n - 1$ *in parallel* **do**
  $Q_i^n = d_n^j P$
  for $i = n - 1$ *to* 0 **do**
    for $j = i + 1$ *to* 1 *in parallel* **do**
      $Q_{\frac{j}{2}}^i = f(Q_j^{i+1}, Q_{j-1}^{i+1})$
return Q_0

For implementation, we looked into the suggested library *pycryptodome*, but it contains definitions which utilizes python objects which are troublesome to work due to the restrictions with compyle and hence, would require a lot of work to write the entire library in pure python. Moreover, the library offers curve definitions but does not easily facilitate usage of curves for arithmetic other than for encryption and decryption purposes.

3 Future Work

In the next week, we analyze more algorithms such as Montgomery Ladder, Co-Z Double-and-Add and Addition Chain by Izu and Takagi[2]. We will also be choosing a more suitable library to work with NIST *p-curves*, such as *tinyec*. We will also be identifying potential performance bottlenecks in these algorithms and explore ways to mitigate them.

References

[1] George Gunner. Lecture notes in parallel computing, March 2017.

[2] Tetsuya Izu and Tsuyoshi Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. volume 2274, pages 371–374, 02 2002.