**El Gamal over ECC using Lopez-Dahab (LD) projective coordinates**

**Jared Garey & Cristian Tapiero**

**Abstract**

TODO

**Introduction**

Elliptic Curve Cryptography is a more efficient use of bits when compared to Feisel cypher-based cryptosystems. However, there are more complex math operations, which makes it more difficult to design hardware for. The most complex operation is a division or inversion. To avoid any division or inverse operations the math is done in a projective coordinate system instead of the default affine coordinate system. The Lopez-Dahab paper introduces an altered algorithm from the traditional projective coordinate system algorithms we learned in class, and looks at the performance benefits. The only altered algorithm is for the point doubling operation. We will implement the LD point doubling algorithm in a full El Gamal Elliptic Curve Cryptosystem. We will show that the resulting points are indeed the same when using the default LD point doubling algorithm, and compare the resources used in hardware between the two designs.

**Notations / Abbreviations**

LD – Lopez-Dahab

ECC – Elliptical Curve Cryptography or Cryptosystem

In the equations, a capital X is equal to a root α in the primitive polynomial. A lowercase x is a variable for the equation.

**Contributions / Division of labor**

We both individually read the paper on LD, and verified our Singular and Verilog code. We worked together on our project proposal and plans for what we wanted to do, as well as debugging several specific bugs in our code.

Christian put together the Singular and Verilog code, pulling from homework 3. He wrote most of the simulations. He also did the key generation (in singular), encryption (in singular and Verilog), and decryption (in singular and Verilog).

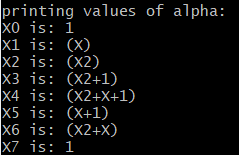
Jared wrote the point generation testbench and singular code to translate a point from the projective to the affine coordinate system. He also wrote the final paper, including gathering extra data and cleaning up project files.

**Project Description and Results**

Singular Coding and Design

We started our cryptosystem design by choosing an elliptic curve ECC and primitive polynomial P(X). We chose to use k=3 for this design. Instead of increasing the bit length, we chose to focus on implementing a full cryptosystem and kept the bit length shorter.

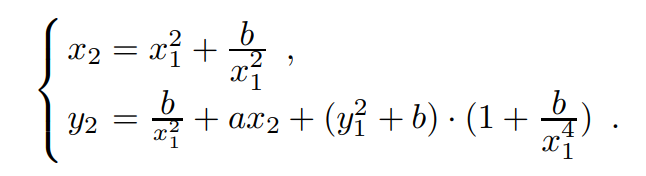
Generating our alpha (root) values from the primitive polynomial, we get the following.



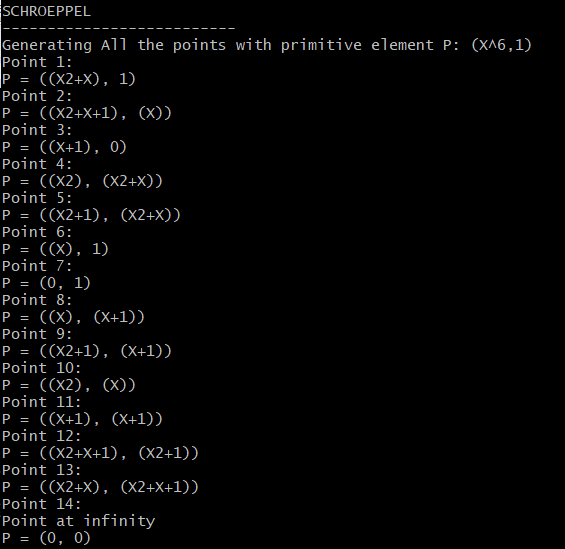
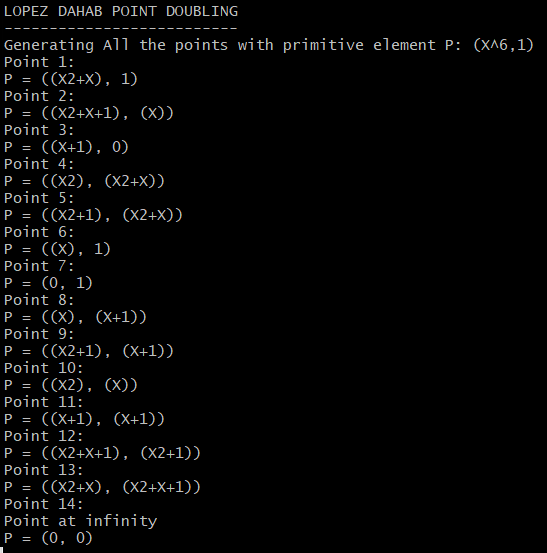
We will use (X6, 1) for our generator point in both singular and Verilog code. We also tried and verified the other generator point (X,1). Just as in homework 3, we also generate all the points on the curve by substituting in x=all values of α into ECC, then factoring the resulting polynomial. This will tell us what all the possible points are on the curve, with 2 points for each value of alpha.

The first unique task for this project was to design the LD algorithm in Singular. P1 is the generator point. P2 is generated by doing the point doubling algorithm (P+P). This is the operation that is unique for the Lopez-Dahab algorithm. The rest of the points are generated the same, which is point addition (P+Q).

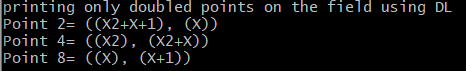
The LD point doubling algorithm and proof are given in the Lopez-Dahab paper. The formula we implemented is given below.



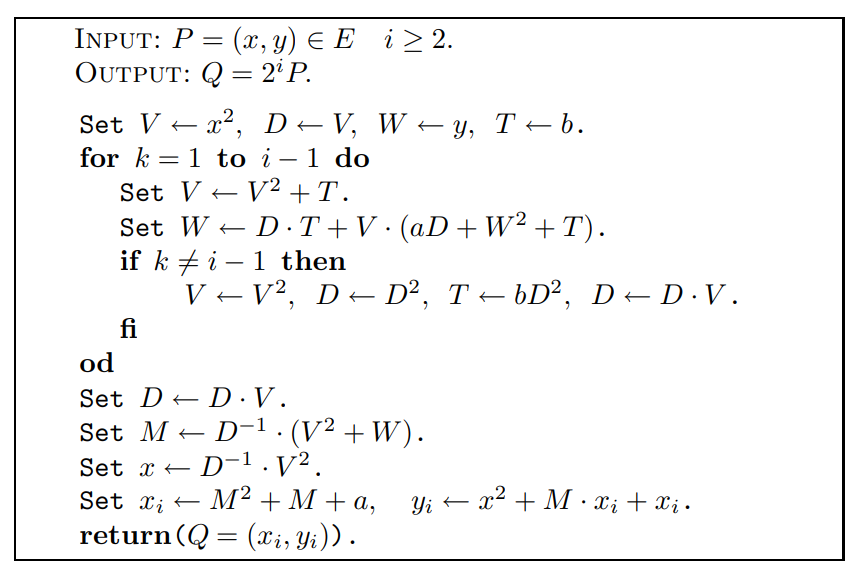
Below are the generated points, verifying both algorithms give us the same result.

To further verify the point doubling algorithm, we did repeated point doubling. These points match what is generated using point addition. This verification was more useful to us because it is far less operations to get to further points, which is what will be done in a full cryptosystem. We experimented several more point doubling operations that what is shown here.

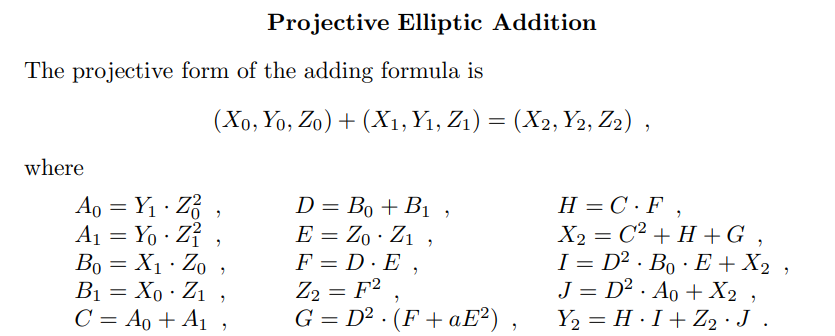


We also briefly looked at the repeated point doubling algorithm given in the LD paper, but did not implement it. This would be interesting to look into and compare efficiency for if more work is ever done.

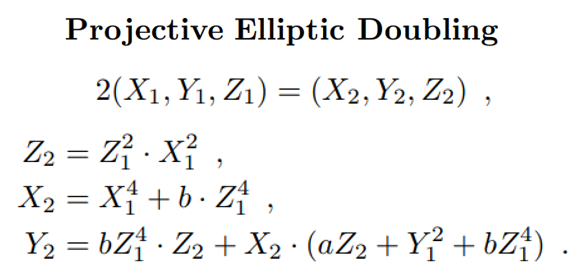


Hardware Implementation

For the Hardware design we chose to use the same Montgomery multiplier for the Galois Field multiplication that we used for homework 3. The addition in the Galois Field was simply a bitwise XOR. The implementation for point addition P+Q=R is given in the equations below (given in the LD paper). We did not implement point addition in homework 3, so implementing this was new. It wasn’t obvious if these equations given in the LD paper were equivalent to the traditional equations for projective coordinate point addition, but we think that it is equivalent. There is also an improved algorithm for when Z­1 = 1, but we did not implement it.



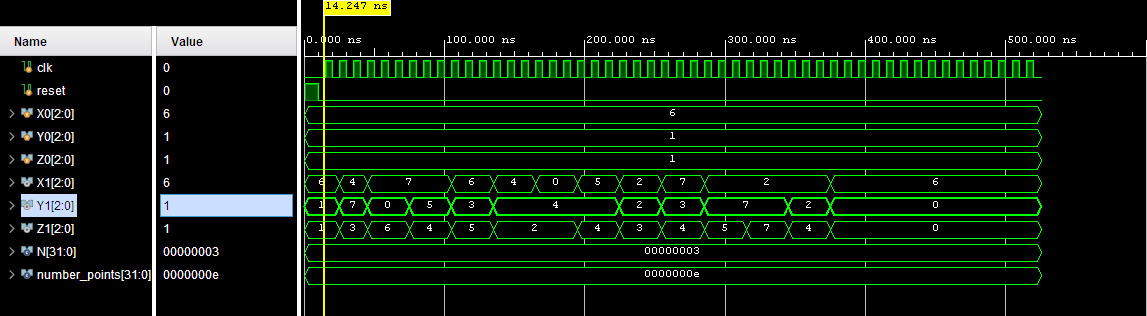
The point doubling P+P=R algorithm has changed to the equations below (given in the LD paper). With both the point addition and doubling algorithms, the number of squaring operations has increased, which provides more benefit to us because that is where the LD algorithm is used.



We simulated both hardware implementations of the algorithms. For the point addition TODO

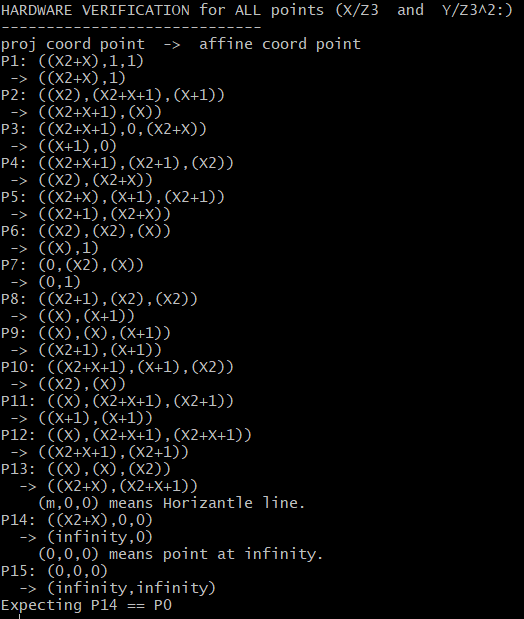
For the point doubling testbench, we did 2 test cases.

Below is the point generation simulation of our Verilog code. This testbench file was very similar to a singular script. We implemented a for loop with a point double for the first point, and point addition afterwards. The generator and starting point is P0(X,Y,Z) = (6,1,1) or (X2 + X, 1, 1). We generated points to P14, which is a point intersecting the line at infinity. P15 should give us the starting point P1.

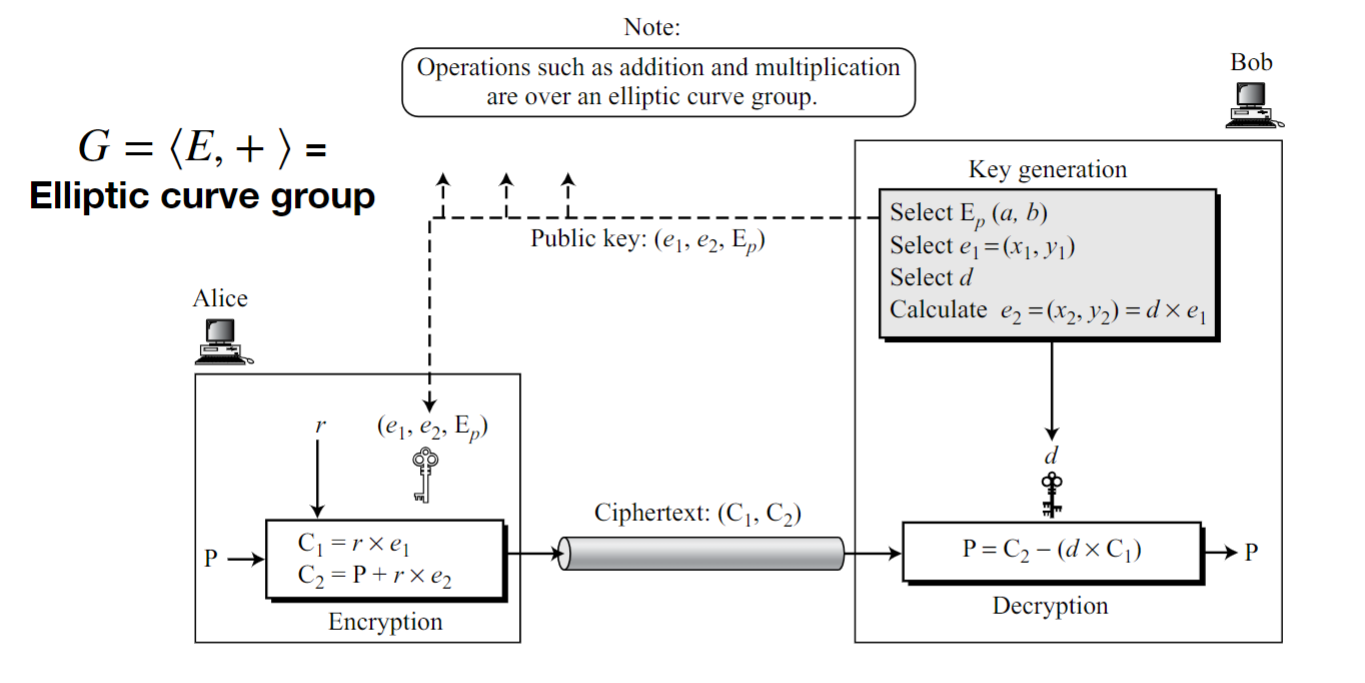
****

This type of operation was useful so we created a statemachine to cycle through the generated points a number of times.

Using Singular, we translated our points from LD projective to the affine coordinate system using the equations below. These points are all the same as the calculated points using singular. One difficulty we had but didn’t solve was on P15, the last point. This point should be equal to the first point (the generator point), but instead our code gave us (0,0,0). We had to take this into consideration later when choosing inputs for encryption and decryption.

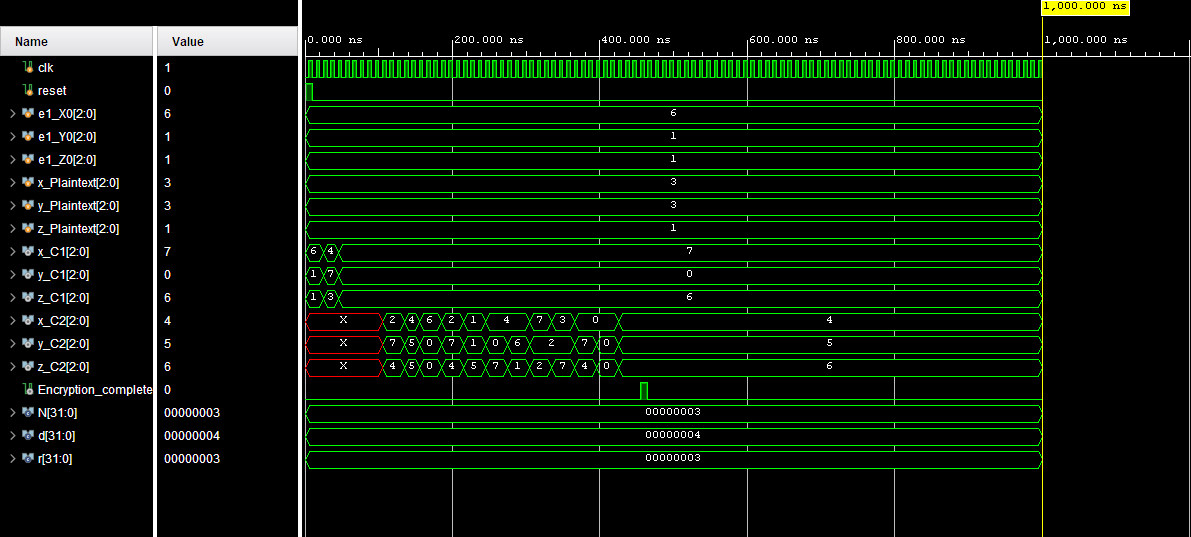


Now that we have fully proved our algorithms work for all points, we implemented the El Gamal Cryptosystem (block diagram is shown below). We used singular to try out several values for our key generation (p, e1, d, and e2). Because we had an issue with P15 not being P1, we had to avoid higher values for the keys.



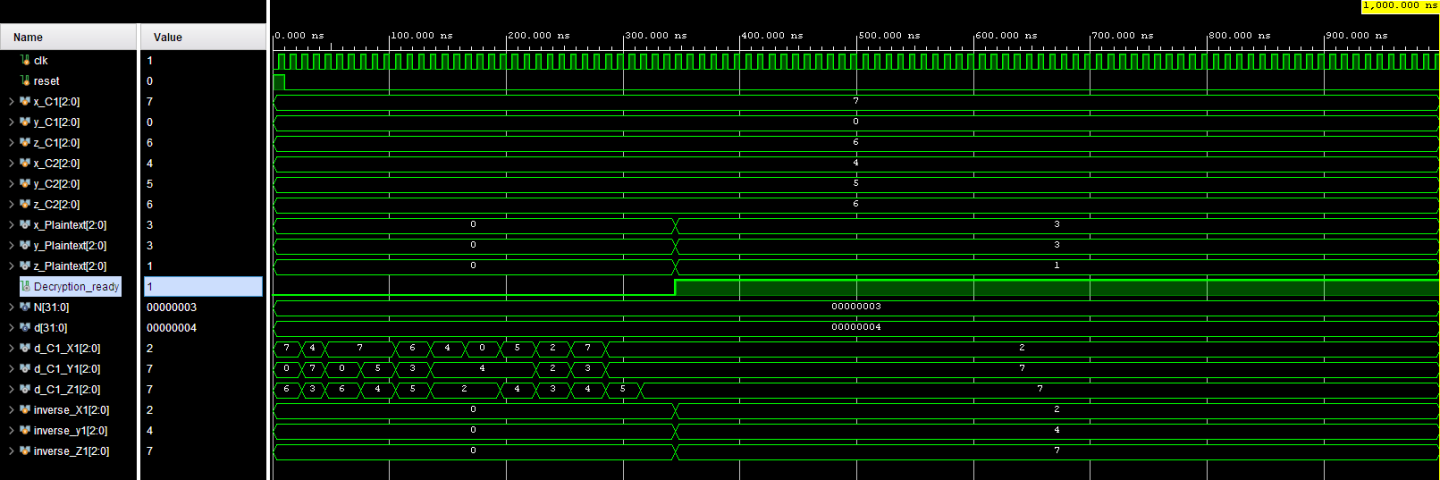
Below is the encryption and decryption simulation of our hardware design. As shown in the block diagram above, the receiver Bob does the key generation. For this we chose e1 = (6,1,1), which always has to be a point generator, and then we calculated e2 = (4,6,1) using the point generator circuit and iterating d times. We selected d = 4.

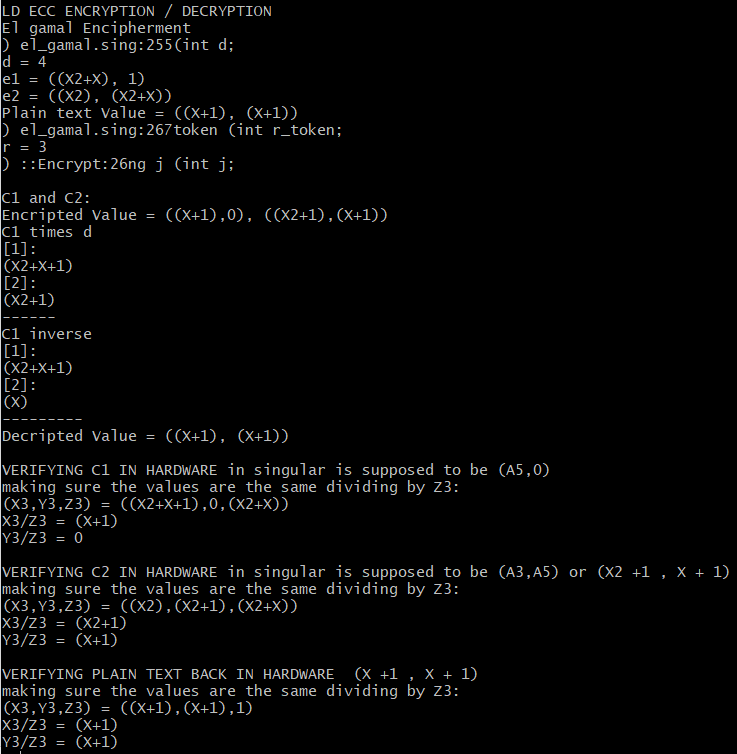
For encryption by Alice, we started with a plaintext P = (3,3,1) and chose a private receiver key of r = 3. For decryption by Bob, we used the previously chosen value d = 4. The math of El Gamal and proof of how is works is described in the class lecture slides so we won’t go into it further here.

****

Using singular, we verified that the values calculated by hardware were correct. Below is the singular script output. The script output shows that we’re expecting C1\*d = (7,5), and [C­1\*d]-1= (7,2). Hardware gives us C1 = (7,0,6), C2 = (4,5,6). Transforming that to the

TODO: need to get a better simulation picture





**Concepts learned**

Most of what we learned for this project came from studying the LD paper. This algorithm for point doubling is very simple and easy to implement. It focuses on improving just one aspect of projective coordinate system arithmetic: point doubling. It increases the number of field multiplications in order to only slightly reduce the number of field inversions. This was interesting because they didn’t start designing an algorithm from the ground up, but instead improved an already existing algorithm.

We learned that when analyzing performance of an algorithm, you have to take into consideration how common your algorithm will be used. The paper brings up in the beginning that if the cost ratio of inversion to multiplication is very small (< 3), then using projective coordinates isn’t going to be as efficient. This stuck out because even though it’s unlikely one would use an elliptical curve where this is true, it’s still possible. And inversion isn’t impossible in hardware, just very costly.

We also learned from implementing the El Gamal cryptosystem that it’s important to test and treat edge cases. Our design isn’t very useful because we aren’t doing something correct with the infinity value. To avoid this, we determined we needed to keep our key generation numbers small.

When implementing hardware encryption and decryption you have to be careful with which algorithms you use. We initially implemented a point generation state machine that would take an input point and use it to generate another point N away. This was a mistake because we forgot that the point generation circuit had to use a generator point, and would give incorrect values when given other points.

There are some things we still don’t understand. Why does e1 have to be a generator point? Why is e1 a public key if it is a generator point? And why is the point addition implementation in hardware different than what is in our lecture slides? Given time, we would be able to continue learning these things.

**Conclusion**

TODO

**References**

[1] J. L´opez, R. Dahab, and R. Dahab, “Improved Algorithms for Elliptic Curve Arithmetic in GF(2 n).” Springer-Verlag, 1998, pp. 201–212.

[2]  P. Barrett, “Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor,” in Proceedings of Advances In Cryptology. London, UK, UK: Springer-Verlag, 1987, pp. 311–323. [Online]. Available: <http://dl.acm.org/citation.cfm?id=36664.36688>

[3] Priyak Kalla slides and book on hardware Cryptography

TODO

OTHER TODO: Make a results table. Get block diagram of hw, pseudocode for sw, show the 2 point generators, curve eq we chose, describe projective coordinate system. It’s the same as before, just with an equivalent algorithm that has some advantages. Compare resources and add that. In README, make sure to say they need to change the filepath in testbench files.