CS 2051: Honors Discrete Mathematics
Spring 2023 Homework 7 Supplement outline

Sarthak Mohanty

Title: ECC is the new RSA

Overview

Elliptic Curve Cryptography (ECC) is one of the most powerful cryptosystems in use today. Companies are using ECC everything to secure everything from our customers' HTTPS connections to how we pass data between our data centers. In fact, based on currently understood mathematics, ECC provides a significantly more secure foundation than first generation public key cryptography systems like RSA, for reasons we'll explore in detail later. The specific implementations you might learn in this supplement aren't as important as the general framework, understanding, and ideas youll develop.

Cryptography Primer

Symmetric vs Asymmetric Systems

In cryptography, there are two main types of encryption algorithms: symmetric and asymmetric. Symmetric encryption uses the same key to both encrypt and decrypt the message, while asymmetric encryption uses different keys for encryption and decryption.
Symmetric encryption algorithms are generally faster and more efficient, but they require the sender and the recipient to share a secret key. The most prominent example is known as AES[1]
Asymmetric, or public-key encryption algorithms, on the other hand, are slower and less efficient, but they allow the sender to send a message securely to the recipient without sharing a secret key. Examples include RSA, Diffie-Helman, and ECCDH.

Trapdoor Functions: RSA

Trapdoor functions are an essential component of asymmetric encryption algorithms. A trapdoor function is a function that is easy to compute in one direction, but difficult to compute in the opposite direction without knowledge of additional information, such as a secret key.
RSA (Rivest-Shamir-Adleman) is a commonly used asymmetric encryption algorithm that relies on the mathematical properties of large prime numbers. Here are the steps involved in generating an RSA public and private key pair:
The security of RSA is based on the difficulty of factoring large composite numbers. Factoring is the process of finding the prime factors of a composite number, which is believed to be a computationally difficult problem.

[1] some of your classmates have chosen this as their group project!

Part 1: Diffie-Helman

Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange is a cryptographic protocol that allows two parties to generate a shared secret key over an insecure communication channel. The protocol is named after its inventors, Whitfield Diffie and Martin Hellman.
There is a classic story of how diffie-helman (and in general) works, of which a modified version is below:

> Professor Brito

In actuality, the Diffie-Helman is a bit more complicated than this. I don't have time to write it all out, but luckily the textbook has a very good explanation of the concepts.

0.1 The Discrete Log Program and Attacks

Brute Force

All public key schemes are in theory susceptible to a brute force attack. Try all possible values. This takes $\mathcal{O}(n)$ time.

Baby-Step Giant-Step algorithm

This algorithm trades off space to improve the running time over brute force. The idea is to write $x = i\lceil n \rceil + j$ for $0 \leq i, j \leq \lceil n \rceil$. For simplicity let $m = \lceil n \rceil$. First compute g^j for all $0 \leq j \leq m$. Then we compute $y \cdot g^{-im}$ for each $0 \leq i \leq m$. Notice that both can be done in $O(\sqrt{n})$ time. If we could find a collision $y \cdot g^{-im} = g^j$, then $x = im + j$. To find this collision efficiently we store all g^j (for all $0 \leq j \leq m$) by hash-sets and when we compute $y \cdot g^{-im}$ we simply check if that value is already present in the set. This finds the collision in $O(\sqrt{n})$ time (as we need to check if the value is present in the set only $O(\sqrt{n})$ times). Overall this algorithm uses $O(\sqrt{n})$ space and time.

Pollard rho algorithm

This is a randomized algorithm that works in $O(\sqrt{n})$ expected time and with $O(1)$ space requirement. We descirbe the algorithm for prime modulus. All the operations described in this section work mod p.
The idea is to divide $\{1, \ldots n-1\}$, which are the possible xs into three sets, say, S_0, S_1 and S_2 of roughly equal size. In practice we do it by taking $x \pmod 3$ ie. $x \in S_i$, where $i = x \pmod 3$. We do a random walk starting at $x_0 = g^{a_0} y^{b_0}$ where $a_0 = b_0 = 0$. To generate x_{i+1} from x_i we follow the following rule:

$$x_{i+1} = \begin{cases} x_i^2 \text{ if } x_i \in S_0 \\ yx_i \text{ if } x_i \in S_1 \\ gx_i \text{ if } x_i \in S_2 \end{cases}$$

Corresponding to these x_i's we also maintain the a_i, b_i such that $x_i = g^{a_i} y^{b_i}$. Now suppose we find i and j, $i \neq j$, such that $x_i = x_j$, or $g^{a_i} y^{b_i} = g^{a_j} y^{b_j}$. Then $g^{a_i - a_j} = y^{b_j - b_i} \pmod n$. Substituting $y = g^x$, we get the equation $a_i - a_j = x(b_j - b_i)$. We then solve this using a linear congruence solver to obtain x.
So all we need to so now is to find i and j, $i \neq j$, such that $x_i = x_j$. This is done using the classic Floyd's Hare and Tortoise algorithm (by which the algorithm gets the ρ in its name). Essentially there are two pointers, one that jumps two steps and another that jumps only one. If there are i and j, such that $x_i = x_j$, then these pointers would meet at the same point, giving us i and j.

General Number Field Sieve

These factoring algorithms get more efficient as the size of the numbers being factored get larger. The gap between the difficulty of factoring large numbers and multiplying large numbers is shrinking as the number (i.e. the key's bit length) gets larger. As the resources available to decrypt numbers increase, the size of the keys need to grow even faster. This is not a sustainable situation for mobile and low-powered devices that have limited computational power. The gap between factoring and multiplying is not sustainable in the long term.
All this means is that RSA is not the ideal system for the future of cryptography. In an ideal Trapdoor Function, the easy way and the hard way get harder at the same rate with respect to the size of the numbers in question. We need a public key system based on a better Trapdoor. In 1985, such a system was found, revolving around an (then) arcane field known as elliptic curves.

In this part, you'll fully implement the Diffie-Helman key exchange and try to break it on small numbers.
Class Actor:

- `generate_key(elements, relation)`
- `generate_shared_key(elements, relation)`

Class BadActor

- `brute_force`: This function takes in the paraameters for the discrete log problem, and returns the correct answer using brute force
- `baby_step_giant_step`: Same thing as above, but with baby step giant step.

Part 2: Enter Elliptic Curves

These algorithms are faster and less computationally intensive than the naive approach of just guessing pairs of known primes.
The standard form for an elliptic curve is

$$y^2 = x^3 + Ax + B,$$

where A and B are constants. This will be referred to as the Weierstrass equation for an elliptic curve. For reasons that will be discussed later, also include the point $\{\infty\}$ in this function.

The Group Law

[2] One important aspect of elliptic curves is that we can start with two points, or even one point on an elliptic curve, and produce another point.

[2] explanation for the same is https://www.math.brown.edu/reschwar/M1540B/elliptic.pdf

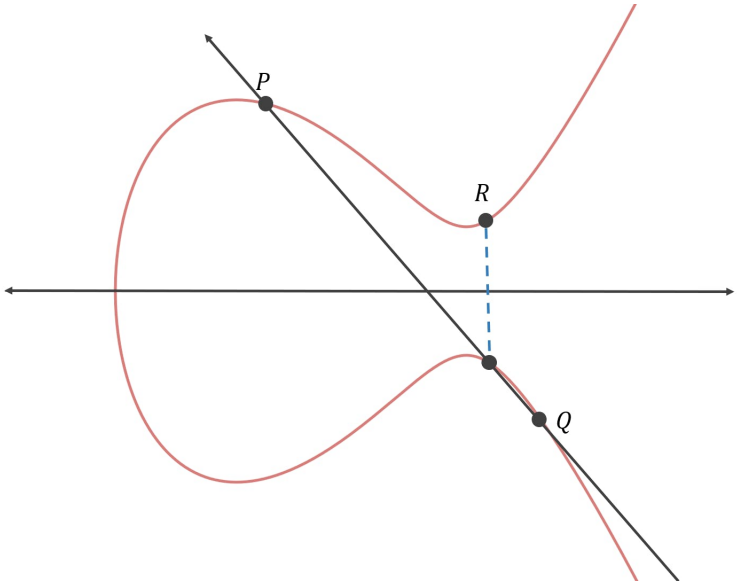

Figure 1: Caption

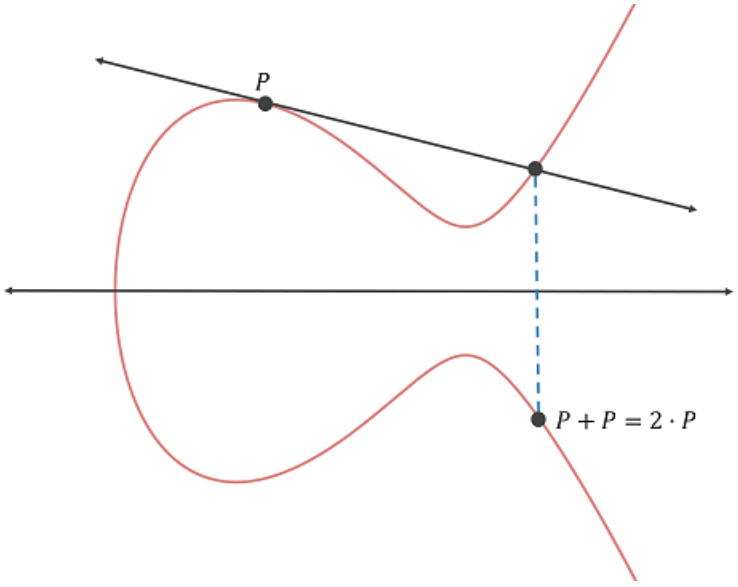

Figure 2: Caption

In this part, you'll implement point addition and point multiplication on elliptic curves
In this part, you'll implement the following functions:

- `point_addition(elliptic_curve, P, Q)`: This function takes in an elliptic curve
- `point_scalar_multiplication(elliptic_curve, k, P))`: This function takes in an elliptic curve of the form described, a scalar integer k, and a point P, and repeatedly multiplies this result to output a number.

You can run `visualize_addition` and `visualize_multiplication` to generate the corresponding visualizations mentioned above. Notes: This part is just algebra and geometry. Since I know it's been a while since many of you touched this, here's a few steps you get you started.

- For point addition, we assumed that $P \neq Q \neq \infty$. For this reason, we can draw the line L through P_1 and P_2. Its slope is
$$m = \frac{y_2 - y_1}{x_2 - x_1}.$$
If $x_1 = x_2$, then L is vertical, so the resulting point is just ∞. Now let's assume that $x_1 \neq x_2$. The equation of L is then
$$y = m(x - x_1) + y_2.$$
To find the intersection of this line with the elliptic curve, we can substitute the above equation into the formula for an elliptic curve
$$y^2 = x^3 + ax + b.$$
Finally, we can reflect the above point across the x-axis to obtain the point $P_3 = (x_3, y_3)$.
- For point scalar multiplication, we have that $P = Q = (x_1, y_1)$. When two points on a curve are very close to each other, the line through them approximates a tangent line. Therefore, when the two points coincide, we take the line L through them to be the tangent tine. Implicit differentiation allows us to find the slope m of L:
$$2y\frac{dy}{dx} = 3x^2 + A, \quad \text{so} \quad m = \frac{dy}{dx}\frac{3x_1^2 + A}{2y_1}.$$
If $y_1 = 0$ then the line is vertical and we set $P_1 + P_2 = \infty$, as before. Otherwise, we plug in values and solve as before.

Part 3: In The Field

The elliptic curve cryptography (ECC) uses elliptic curves over the finite field $\mathbb{F}_p$ (where p is prime and $p > 3$). For example, the "Bitcoin curve" secp256k1 takes the form:

$$y2 \equiv x3 + 7 \pmod{p}$$

In this part, you'll visualize elliptic curves over finite fields, as well as implement point addition and point multiplication over these fields

In this part, you'll implement the following functions:

- `generate_point_cloud(elliptic_curve, P, Q)`
- `point_addition(elliptic_curve, P, Q)`: This function takes in an elliptic curve
- `point_scalar_multiplication(elliptic_curve, k, P))`: This function takes in an elliptic curve of the form described, a scalar integer k, and a point P, and repeatedly multiplies this result to output a number.

Part 4: ECC in Action: Elliptic-curve Diffie Helman

You've devleoped some undrestanding of how elliptic curves work, but it's naturarl to doubt the connection between them. This is where yopu're wrong. Note that... this is called *order independence*. This makes it really easy to modify diffie helman to accomodate our elliptic curves, as follows:

The Elliptic-curve Discrete Log Function

For carefully chosen (by cryptographers) finite fields and elliptic curves, the ECDLP problem has no efficient solution.

The multiplication of elliptic curve points in the group $\mathbb{F}_p$ is similar to exponentiation of integers in the group $\mathbb{Z}_p$ (this is known as multiplicative notation) and this is how the ECDLP problem is similar to the DLP problem (discrete logarithm problem).

See this video for implementation details

In this part, you'll implement a elliptic-curve diffie helman key exchange and try to break it on small numbers. You should directly call the methods you constructed in Part 2. You should replicate the methods you created in Part 1, this time accomodating for the elliptic-curve element.

Class Actor:

- `generate_key(elements, relation)`
- `generate_shared_key(elements, relation)`

Class BadActor

- `brute_force(elements, relation)`: This function takes in a relation (represented as a set of tuples) and returns whether or not the relation (taken over the set of elements) is a valid partial order.
- `baby_step_giant_step(elements, relation)`: This function takes in the same arguments are before, but this time returns whether or not the relation is a valid equivalence relation.

Conclusion

RSA vs ECC

Since RSA and ECC essentially serve the same purpose, there have been many heated discussions as to which implentation to use. The main pro of using ECC is the fact that it uses fewer memory and CPU resources, important as mobile computing becomes more ubiquitous. However, there are a few flaws with ECC. First,

there are only a few curves that work and the NIST is in contorl of most of them, but people distrust NIST. Second,

Finally, neither ECC nor RSA are secure against quantum computers.

Real World Application

Even with all those downsides, more and more companies are using ECC. Two prominent examples:

- ChatGPT
- ProtonVPN

to do: explain how you can see ecc implentation on website
assymetric not often used since takes long time. Instead, key exchange is done using assymetric andthen symmetric is used to send the actual messages, as can be seen here

Submission Instructions (10 pts)

After you fill the appropriate functions, submit the following files to Gradescope and make sure you pass all test cases:

- `diffie_helman.py`
- `elliptic_curves.py`
- `ECCDH.py`

Notes

- The autograder may not reflect your final grade on the assignment. We reserve the right to run additional tests during grading.
- Do not import additional packages, as your submission may not pass the test cases or manual review.

References

why infinity is added here