This part looks the simplest part to write, but it was extremely difficult to grasp around the Edward Curve because I couldn't understand how to multiply s *G mentioned inside the specifications. When I was able to understand that I needed to use exponentiation to achieve s*G I would finally able to multiply a point by the secret key.

Now I ran into another problem where I had no idea whether my Part 3 private and public keys worked and moved straight onto Part 4 to see if I could get it to work. Unfortunately, my part 3 had an error in there because I had the Edward Point add function signs flipped. I was using modInverse and intended but this bug wasted a lot of time for me because I did not have any test vectors.

The way I ended up testing my Part 3 public key by taking the public key point and adding it with the negation of itself to get the neutral element of O := (0,1). Once I had this issue fixed, I was able to have my Part 3 working correctly and Part 4 working at the same time since I already the code for Part 4 to test my initial Part 3.

Overall I feel that this elliptic curve key pair generation was the most difficult because it used Edwards Curve which was hard to find information online. Everything seemed to be about Bitcoin's Secp256k1 curve.

The pk.txt inside Part3 folder shows an example of what the output was when I used the passphrase "test" to create the private key.