1. I successfully implemented a Birthday attack on a hash function as can be seen below. Section a is a semantically similar message to the original in the textbook. I generated $2^{m/2}$ of these messages by systematically replacing " " with " \b " because "\b" is the notation for backspace. This I would end up with messages that would look the same in a terminal but still have different hashes. From there I created an amusingly incorrect fraudulent message, you can find it is section b, and generated different versions of it until one of the fraudulent hashes matched a pregenerated one. It should be noted that the hash that I used and broke was a simple 8bit hash created from the previous Toy DES assignment; I wasn't confident in my laptop's ability to consistently generate and hold 2^{32} hashes in a timely manner.

    a. "More efficient attacks are possible by employing cryptanalysis **\b** to specific hash functions. When a collision attack is discovered and is found to be faster than a birthday attack, a hash function is often denounced as "broken". The NIST hash function competition was largely induced by published collision attacks against two very commonly used hash functions, MD5 and SHA-1. The collision attacks against MD5 have improved so much that, as of 2007, it takes just a few seconds on a regular computer. Hash collisions created this way are usually constant length and largely unstructured, so cannot directly be applied to attack widespread document formats or protocols."

    b. "Least efficient attacks are possible **\b** by employing cryptanalysis to specific hash functions. When a collision attack is discovered and is found to be slower than a birthday attack, a hash function is often denounced as "perfect". The NASA hash function competition was largely induced by published collision attacks against two very commonly used hash functions, MARIO and SONIC-1. The collision attacks against MARIO have improved so much that, as of 2157, it takes just a few years on a regular computer. Hash collisions created this way are usually exponential length and largely structured, so can directly be applied to attack widespread document formats and protocols."

    c. Code for Birthday Attack algorithm is located at https://github.com/angelson1992/MakeupExam/blob/master/src/BirthdayAttacker.java
    d. Code for simple hasher made from DES is located at https://github.com/angelson1992/MakeupExam/blob/master/src/SimpleHash.java

2. Elliptic Curves and ECC

a. 10-12: Find all points on $E_{11}(1,6)$.

X	$y^2 = x^3 + x + 6$ mod 11	Is in QR(11)	Y
0	6	No	
1	8	No	
2	16 = 5	Yes	4, 7
3	36 = 3	Yes	5, 6
4	74 = 8	No	
5	136 = 4	Yes	2, 9
6	228 = 8	No	
7	356 = 4	Yes	2, 9
8	526 = 9	Yes	3, 8
9	744 = 7	No	
10	1016 = 4	Yes	2, 9
∞			∞

So the points should be (2, 4), (2, 7), (3, 5), (3, 6), (5, 2), (5, 9), (7, 2), (7, 9), (8, 3), (8,8), (10, 2), (10, 9), (∞, ∞)

b. 10-13: What are the negatives of the following EC points on Z_{17}?

i. P = (5, 8)
   1. -P = -(5, 8) = **(5, -8)**

ii. Q = (3, 0)
   1. -Q = -(3, 0) = **(3, 0)**

iii. R = (0, 6)
   1. -R = -(0, 6) = **(0, -6)**

c. 10-14: For $E_{11}(1, 6)$ a.k.a $y^2 = x^3 + x + 6$ mod 11, consider point G = (2, 7). Compute the multiples of G from 2G to 13G

i. $2G = (m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(5, 2)**

1. $M = (3x^2+a/2y) = 13/14 = 13*14^{-1} \bmod 11 = 13*4 \bmod 11 = 8$

ii. $3G = G + 2G = (2, 7) + (5, 2) = (m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(8, 3)**

1. $M = (y_2-y_1/x_2-x_1) = (7-2)/(2-5) = 5/-3 = 5 * (-3)^{-1} \bmod 11 = 5 * 7 \bmod 11 = 2$

iii. $4G = G + 3G = (2, 7) + (8, 3) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(10, 2)**

1. $M = (y_2-y_1/x_2-x_1) = (3-7)/(8-2) = -4/6 = -4 * (6)^{-1} \bmod 11 = -4 * 2 \bmod 11 = 3$

iv. $5G = G + 4G = (2, 7) + (10, 2) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(3, 6)**

1. $M = (y_2-y_1/x_2-x_1) = (2-7)/(10-2) = -5/8 = -5 * (8)^{-1} \bmod 11 = -5 * 7 \bmod 11 = 9$

v. $6G = G + 5G = (2, 7) + (3, 6) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(7, 9)**

1. $M = (y_2-y_1/x_2-x_1) = (6-7)/(3-2) = -1/1 = -1 * (1)^{-1} \bmod 11 = -1 * 1 \bmod 11 = 10$

vi. $7G = G + 6G = (2, 7) + (7, 9) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(7, 2)**

1. $M = (y_2-y_1/x_2-x_1) = (9-7)/(7-2) = 2/5 = 2 * (5)^{-1} \bmod 11 = 2 * 9 \bmod 11 = 7$

vii. $8G = G + 7G = (2, 7) + (7, 2) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(3, 5)**

1. $M = (y_2-y_1/x_2-x_1) = (2-7)/(7-2) = -5/5 = -5 * (5)^{-1} \bmod 11 = -5 * 9 \bmod 11 = 10$

viii. $9G = G + 8G = (2, 7) + (3, 5) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(10, 9)**

1. $M = (y_2-y_1/x_2-x_1) = (5-7)/(3-2) = -2/1 = -2 * (1)^{-1} \bmod 11 = -2 * 1 \bmod 11 = 9$

ix. $10G = G + 9G = (2, 7) + (10, 9) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(8, 8)**

1. $M = (y_2-y_1/x_2-x_1) = (9-7)/(10-2) = 2/8 = 2 * (8)^{-1} \bmod 11 = 2 * 7 \bmod 11 = 3$

x. $11G = G + 10G = (2, 7) + (8, 8) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(5, 9)**

1. $M = (y_2-y_1/x_2-x_1) = (8-7)/(8-2) = 1/6 = 1 * (6)^{-1} \bmod 11 = 1 * 2 \bmod 11 = 2$

xi. $12G = G + 11G = (2, 7) + (5, 9) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(2, 4)**

1. $M = (y_2-y_1/x_2-x_1) = (9-7)/(5-2) = 2/3 = 2 * (3)^{-1} \bmod 11 = 2 * 4 \bmod 11 = 8$

xii. $13G = G + 12G = (2, 7) + (2, 4) = ((m^2 - x_1 - x_2, m(x_1-x_3)-y_1) =$ **(∞, ∞)**

1. $M = (y_2-y_1/x_2-x_1) = (4-7)/(2-2) = -3/0 = -\infty$

d. 10-15: This problem performs elliptic curve encryption/decryption. The cryptosystem parameters are $E_{11}(1, 6)$ and $G = (2, 7)$. B's private key is $n_B = 7$.

i. Find B's public key P_B.

1. $P_B = n_B * G = 7 * G =$ **(7, 2)**

a. The math for this was done in the previous problem, 10-14.

ii. A wishes to encrypt the message $P_m = (10, 9)$ and chooses the random value $k = 3$. Determine the ciphertext C_m.

1. $C_m = \{kG, P_m + kP_B\}=$ **$\{(8,3), (10, 2)\}$**
   a. $kG = 3*(2,7) = (8, 3)$ because this math was done in the previous question, 10-14.
   b. $kP_B = 3*P_B = P_B + 2P_B= (7, 2) + (2, 7) = (3, 5)$
      i. $2P_B = (m^2 - x_1 - x_1, m(x_1-x_3)-y_1) = (2, 7)$
         1. Where $m = (3x^2+a/2y) = 148*4^{-1} = 4 \bmod 11$
      ii. $P_B+2P_B= (7,2)+(2,7)= (m^2-x_1-x_2, m(x_1-x_3)-y_1)= (3,5)$
         1. Where $m = (y_2-y_1/x_2-x_1) = (7-2)/(2-7) = 5*(-5)^{-1} = 5*2 = 10$
   c. $P_m+kP_B = (10,9)+(3,5) = (m^2-x_1-x_2,m(x_1-x_3)-y_1) = (10,2)$
      i. Where $m = (y_2-y_1/x_2-x_1) = (5-9)/(3-10) = -4*(-7)^{-1} \bmod 11 = -4*3 \bmod 11 = 10$

iii. Show the calculation by which B recovers P_m from C_m.

1. $C_m = \{kG, P_m + kP_B\} = \{(8,3), (10, 2)\}$. Also $P_B = n_B*G$.
2. $C_m = \{kG, P_m + k(n_B*G)\}$
3. $C_m = \{kG, P_m + (kG)*n_B\}$
4. Thus $P_m = P_m + (kG)*n_B - (kG)*n_B$
5. Thus $P_m = P_m + kP_B - (kG)*n_B$ and B can construct $(kG)*n_B$ with the information that it has available.

6. $P_m = P_m + kP_B - (kG)*n_B = (10, 2)-(kG)*n_B = (10, 2)-(3, 5) =$ **$(10,9)$**
   a. $(kG)*n_B = (8, 3)*7 = (3, 5)$
      i. $(8,3)*2 = (m^2-x_1-x_1,m(x_1-x_3)-y_1) = (7, 9)$
         1. $m = (3x^2+a/2y) = 1$
      ii. $(8,3)*4 = (7,9)*2 = (m^2-x_1-x_1,m(x_1-x_3)-y_1) = (2, 4)$
         1. $m = (3x^2+a/2y) = 7$
      iii. $(8,3)*8 = (2,4)*2 = (m^2-x_1-x_1,m(x_1-x_3)-y_1) = (5, 9)$
         1. $m = (3x^2+a/2y) = 3$
      iv. $(8,3)*7= (5,9)-(8,3)= (m^2-x_1-x_2,m(x_1-x_3)-y_1) = (3, 5)$
         1. $m = (y_2-y_1/x_2-x_1) = 7$
   b. $(10, 2)+(3, -5) = (m^2 - x_1 - x_2, m(x_1-x_3)-y_1) = (10, 9)$
      i. $m = (y_2-y_1/x_2-x_1) = 1$

3. 31531 is most likely prime according to Miller-Rabin. Thus it makes sense that Pollard-Rho failed.

   520482 is obviously not prime considering it’s even and Miller-Rabin can’t even generate the needed numbers because of this. Running Pollard-Rho several times I get 520482 = 3 * 2 * 223 * 389

   485827 is most likely prime according to Miller-Rabin. Thus it makes sense that Pollard-Rho failed.

   15485863 is most likely prime according to Miller-Rabin. Thus it makes sense that Pollard-Rho failed.

   a. Code for Miller-Rabin algorythm is located at https://github.com/angelson1992/MakeupExam/blob/master/src/Miller_Rabin_Algo.java
   b. Code for Pollard-Rho algorythm is located at https://github.com/angelson1992/MakeupExam/blob/master/src/Pollard_rho_algo.java