Takehome Project 2 Written Solutions

Written Part

5. Let's begin the written portion of this assignment by studying the correctness of the MV-Elgamal.

   (a) Let E be an elliptic curve modulo a prime p and $P \in E(\mathbb{F}_p)$ be a point not equal to $\mathcal{O}$. Prove that the order of P is 2 if and only if it's y coordinate is equal to 0.

   Proof. Let $P = (x, y)$. Since $P \neq \mathcal{O}$ we may assume we get to step 3 of the algorithm, and observe that $P + P = \mathcal{O}$ if and only if $x = x$ and $y \equiv -y \mod p$. This occurs precisely when $y \equiv 0 \mod p$ (notice here we are using that $p \neq 2$ in an important way, as $1 \equiv -1 \mod 2$). □

   (b) Use part (a) to explain why in the key creation phase of MV-Elgamal it was insisted that Q_A must have a nonzero y coordinate.

   Proof. When encrypting a message we assume the x and y coordinates of S are nonzero—this is because we must be able to divide by them to recover the message, as explained in part (c). But S is a multiple of Q_A. If Q_A has 0 for it's y-coordinate, then Q_A has order 2, so that it's only multiples are $\mathcal{O}$ and itself. But S can't be either of these points, because in the first case there are no x and y coordinates, and in the second, we would have $y_S = 0$. In either case we would not be able to encrypt a message with S. □

   (c) Prove the correctness of MV-Elgamal. (Be sure to explain why x_S, y_S must be nonzero).

   Proof. We first observe that:

   $$T = n_A R = n_A(kP) = k(n_A P) = kQ_A = S.$$

   Therefore $(x_T, y_T) = (x_S, y_S)$. As we ensured that x_S and y_S are nonzero (modulo p), we see that so are x_T and y_T, and therefore it is in fact possible to compute x_T^{-1} and y_T^{-1}. The identity $x_T^{-1} x_S = 1$ is immediate, as is $y_T^{-1} y_S = 1$. Therefore we observe that

   $$m_1' = x_T^{-1} c_1 = x_T^{-1} x_S m_1 = m_1 \quad \text{and} \quad m_2' = y_T^{-1} c_2 = y_T^{-1} y_S m_2 = m_2.$$

   □

   (d) The original Elgamal and the Elliptic Curve Elgamal essentially wrap together a Diffie-Hellman key exchange and a symmetric cipher into one procedure. Explain why this is also the central philosophy of MV-Elgamal. What plays the roll of the shared secret? What about the symmetric key?

   Proof. The Diffie-Hellman component of MV-Elgamal can be summarized as follows: Alice has a secret multiplier n_A, while Bob has k. Alice multiplies P by her secret multiplier, and sends it to Bob, who does the same to arrive at S. Conversely, Bob multipies P by his secret multiplier and sends it to Alice, who then multiplies by hers to arrive at T. As we saw above, $T = S = kn_A P$ is the shared secret.

This shared secret is used to construct a symmetric cipher as follows: The coordinates of S (or T) are invertible elements $x_S, y_S \in \mathbb{F}_p^*$, that Alice and Bob both secretly know. The use these values as symmetric keys for the *multiplication mod p* cipher. That is, Bob multipies his message(s) by the coordinate(s), to produce the ciphertext, and Alice divides to cipher(s) by the coordinate(s) to recover the plaintext. □

6. Let's now discuss the efficiency.

   (a) Compute the time complexity of the following algorithms (in terms of the number of bits of your prime), proving your answer is correct. You may assume the basic operations $+, -, \times, \div$ and choosing a random number are all $\mathcal{O}(1)$.

      i. `doubleAndAddSmall` (you may assume you are scaling by a number less than the order of your base point)

      Proof. We first observe that `addPoints` is $\mathcal{O}(\log p)$. Steps 1-4 are all a constant number of basic operations. The slowest scenario is if we get to step 5, we do a constant number of basic operations and compute a single inverse modulo p, which using the extended Euclidean algorithm takes $\mathcal{O}(\log p)$. So in total, we have a constant number of basic operations plus one inverse, totaling in $\mathcal{O}(\log p)$.
      With this in hand, we observe that the main loop in `doubleAndAddSmall` runs $\log_2 n$ times, and each time the loop runs `addPoints` at most 2 times. This means that we run $\mathcal{O}(\log n \log p)$, but we want the complexity in terms of p. For this we use that

      $$n \leq |E(\mathbb{F}_p)| \leq p + 2\sqrt{p} = \mathcal{O}(p),$$

      where for the inequality we use Hasse's bound. In particular, the time complexity is:

      $$\mathcal{O}(\log n \log p) = \mathcal{O}(\log p \log p) = \mathcal{O}((\log p)^2).$$

      □

      ii. `generateEllipticCurveAndPoint`

      Proof. Randomly choosing x, y, A is $\mathcal{O}(1)$, and then computing B is a constant number of operations, so each time we generate `[A,B]` and `[x,y]` is $\mathcal{O}(1)$. The question is how many times we may have to do this to get an actually elliptic curve. But the discriminant Δ varies essentially randomly, so it has a probability of $1/p$ of being 0. In particular, we get an elliptic curve with probability of $(p-1)/p$, so (as in Project 1 Problem 6(d)), it should take $p/p - 1 \leq 2 = \mathcal{O}(1)$ guesses to get an elliptic curve. So the total time complexity is $\mathcal{O}(1)$. □

      iii. `MVParameterCreation` (Recall you computed the time complexity of `findPrime` in Project 1

      Proof. We first run `findPrime` which takes $\mathcal{O}((\log p)^2)$. Then `generateEllipticCurveAndPoint` which takes $\mathcal{O}(1)$. Then we make sure that the point we generate doesn't have a y-coordinate equal to 0. If it does, we run `generateEllipticCurveAndPoint` again. How many times should we expect to do this? Well the y-coordinate is a random number between 0 and $p-1$, so there is a $p - 1/p$ probability that it is nonzero, meaning that it should take $p/(p-1) < 2 = \mathcal{O}(1)$ guesses to find a suitable elliptic curve and point. Therefore the overall time complexity is $\mathcal{O}((\log p)^2)$. □

iv. `MVKeyCreation`

Proof. In the main loop we first pick a random number n in $\mathcal{O}(1)$), and then scale our point computing nP, which we saw is $\mathcal{O}((\log p)^2)$. This is the entire time complexity, because we run the loop $\mathcal{O}(1)$ times.
To see this, notice we will restart this if $nP = \mathcal{O}$ or has y-coordinate equal to 0. There are at most 3 points with y-coordinate equal to 0 (because there at most 3 solutions to $x^3 + Ax + B = 0$), so we must avoid 4 bad points. There are $\mathcal{O}(p)$ points on E, so we can estimate a probability of $(p-4)/p$ that we avoid these points. Therefore it should take about $p/(p-4) \leq 2 = \mathcal{O}(1)$ guesses to find a good value of n (using that p is a prime larger than 11, which in all practical cases it will be). □

v. `MVEncrypt`

Proof. In the main loop we choose a random number k in $\mathcal{O}(1)$, and then scale P and Q_A by k, taking $\mathcal{O}((\log p)^2 + (\log p)^2 = \mathcal{O}((\log p)^2)$.
We next show this loop runs $\mathcal{O}(1)$ times. Recall that we need $kP \neq \mathcal{O}$, and $kQ_A \neq 0$ as well as not having any coordinate equal to 0, and similarly for kQ_A. There are at most 3 points with y-coordinate equal to 0, and at most 2 with x-coordinate equal to zero (using that $y^2 = B$ has at most 2 solutions), so there are 6 points to avoid. Since $|E(\mathbb{F}_p)| = \mathcal{O}(p)$, this has a probability of $(p-6)/p$ for each point that kQ_A is good, and a probability of $(p-1)/6$ that kP is good. This gives a probability of , so a probability of $(p-6)(p-1)/p^2$ for both to be 'good' points. Therefore, it should take approximately:

$$\frac{p^2}{(p-6)(p-1)} = \mathcal{O}(1),$$

guesses (indeed, the value gets smaller as p grows). This means the main loop runs in $\mathcal{O}((\log p)^2)$. After the main loop we do a constant number of basic operations (2 multiplications and 2 reductions mod p), so the overall complexity is $\mathcal{O}((\log p)^2)$. □

vi. `MVDecrypt`

Proof. We first scale the point R by n_A, which takes $\mathcal{O}((\log p)^2)$. We then compute 2 inverses, in $\mathcal{O}(\log p)$, and do 2 multiplications which is $\mathcal{O}(1)$. Therefore the dominant term is $\mathcal{O}((\log p)^2)$. □

(b) What is the message expansion of MV-Elgamal?

Proof. The message is a pair (m_1, m_2) with $m_i \in \mathbb{F}_p^*$. If p is approximately k-bits in size, then this means we can send messages up to $2k$-bits long. To send a message one must send a point on $E(\mathbb{F}_p)$ (R) and two elements of $\mathbb{F}_p^*$ (c_1 and c_2). In particular, this is 4 numbers k-bits long. Therefore it takes a cipher approximately $4k$ bits long to send approximately $2k$ bits of data, resulting in a 2-1 message expansion. □

(c) Suppose Alice has an algorithm to efficiently compute square roots modulo p (such things certainly exist). Explain a way that Bob could compress the ciphertext by sending 1 bit in place of the y-coordinate of R, and prove that your method works. What would the new message expansion be?

Proof. Notice that if $R = (x, y)$ then the coordinates x and y satisfy the equation $y^2 = x^3 + Ax + B$ (where A and B are public). Therefore once you know x, you know the value of y^2. Assuming you know how to compute square roots modulo p, then you can narrow the value of y down to two values, the two square roots of y^2 (HW2 Problem 8). We know that these are y and $p - y$ modulo p. Notice that if $y < p/2$ then $p - y > p - p/2 = p/2$. In particular, exactly one of the square roots is less that $p/2$, and the other is greater.

To exploit this, Bob instead of sending the point $R = (x, y)$, can send the pair (x, b) where $b \in \{0, 1\}$ is a bit. Then the agreed upon rule could be:

$$b = \begin{cases} 0 & 0 \leq y < p/2 \\ 1 & p/2 \leq y < p \end{cases}$$

Then Alice can compute y and $p - y$ from x as described in the previous paragraph, and deduce which one is y given the value of b.

(Alternatively, as p is odd, one can observe that y is even if and only if $p - y$ is odd, and the extra bit can encode whether the y-coordinate is even or odd.)

(Alternatively, the extra bit could encode whether the y-coordinate is the larger or smaller square root of y^2.)

Since we don't have to send the y-coordinate, Bob only needs to send three elements of $\mathbb{F}_p$ (plus one bit) as the ciphertext. In particular, it takes $3k + 1$ bits to encode a message $2k$ bits long, so that the message expansion becomes 1.5-1 (plus one extra bit for b). □

7. Let's conclude with some potential attacks on MV-Elgamal.

   (a) Show that if Eve can solve the Elliptic Curve Discrete Log Problem she can use it to crack MV-Elgamal.

   Proof. Eve can consult an ECDLP oracle, presenting the oracle with P and $n_A P = Q_A$ in $E(\mathbb{F}_p)$. The oracle will then present Eve with the value n_A. This is all Eve needs to crack MV-Elgamal. Indeed, with this in hand, if Eve intercepts Bob's message (R, c_1, c_2), where $R = kP$ and $(c_1, c_2) = (x_S m_1, y_S m_2)$ where $S = kQ_A$. Then Eve can compute $n_A R = n_A kP = kQ_A = S$, can compute x_S^{-1} and y_S^{-1}, and multiply them by c_1 and c_2 respectively to recover the message. □

   (b) Show that if Eve can solve the Elliptic Curve Diffie Hellman Problem, she can use it to crack MV-Elgamal

   Proof. Using the notation of Table 1: Eve has access to $Q_A = n_A P$ and $R = kP$, as well as the cipher text $c_1 = x_S m_1$ and $c_2 = y_S m_2$, since these are sent over a public channel. Even can then consult with an ECDHP oracle, to obtain $kn_A P = kQ_A = S$. Eve can then compute x_S^{-1} and y_S^{-1} using the extended Euclidean algorithm and therefore easily obtain $x_S^{-1} c_1 = x_S^{-1} x_S m_1 = m_1$ and $y_S^{-1} c_2 = y_S^{-1} y_S m_2 = m_2$. □

(c) Eve knows the elliptic curve E, and intercepted the two peices of ciphertext c_1, c_2. Explain how Eve could use this information to relate m_1 and m_2 in a polynomial equation (modulo p). In particular, if Eve knows one of the peices of plaintext, she can use that to recover the other peice by solving an equation modulo p. (The moral here is, both peices of plain text must be unique and private with each message. For example: Bob should never leave one blank just because he doesn't need the space.)

Proof. We have relations $c_1 = xm_1$ and $c_2 = ym_2$ for some point (x, y) on a publicly known elliptic curve. In particular, since $c_i, m_i \in \mathbb{F}_p^*$ we can deduce that $(x, y) = (c_1/m_1, c_2/m_2)$ (where here division is done modulo p). Therefore the following equation (or rather congruence modulo p) holds:

$$\left(\frac{c_2}{m_2}\right)^2 = \left(\frac{c_1}{m_1}\right)^3 + A\left(\frac{c_1}{m_1}\right) + B.$$

Therefore, if we know intercept the ciphertext, and know m_1, we can use this equation to compute $\left(\frac{c_2}{m_2}\right)^2$, and therefore taking square roots get two possible values for c_2/m_2. This allows us to solve for m_2 narrowing down the search for the plaintext to two values. Conversely, knowing m_2 we would have $x = \frac{c_1}{m_1}$ in a cubic equation (mod p). The cubic formula therefore reduces this computation to taking cube roots and square roots, and gives 3 possible values for x, and therefore for m_1. □