1 Vernam with Two Dice

Our crypto apprentice decided to encrypt messages \( x \in \mathbb{Z}_{12} \) (instead of bits) using the generalized Vernam cipher in the group \( \mathbb{Z}_{12} \). As he did not fully understand the course, he decided to pick a key \( k \) (for each \( x \)) by rolling two dice (with 6 faces numbered from 1 to 6) and setting \( k = k_1 + k_2 \) to the sum of the two faces up \( k_1 \) and \( k_2 \). The encryption of \( x \) with key \( k \) is then \( y = (x + k) \mod 12 \).

Q.1 Why is this encryption scheme insecure?

Q.2 We still use \( k = k_1 + k_2 \). Given a factor \( n \) of 12, we now take \( x \in \mathbb{Z}_n \) and \( y = (x + k) \mod n \). Show that for some values \( n \), this provides perfect secrecy but for others, this does not. (Consider all factors \( n \) of 12.)

Q.3 Finally, the crypto apprentice decides to encrypt a bit \( x \in \{0,1\} \) into \( y = (x + k) \mod 4 \), still with \( k = k_1 + k_2 \) from rolling the two 6-face dice. We assume that \( x \) is uniformly distributed in \( \{0,1\} \). For each \( c \), compute the probabilities \( \Pr[x = 0|y = c] \) and \( \Pr[x = 1|y = c] \).

Q.4 By taking \( \tilde{x} \in \{0,1\} \) as a function of \( c \) such that \( \Pr[x = \tilde{x}|y = c] \) is maximal, compute the probability \( P_e = \Pr[x \neq \tilde{x}] \) (still when \( x \) is uniform in \( \{0,1\} \)).

2 Elliptic Curve Factoring Method

In this exercise, we want to recover the smallest prime factor \( p \) of an integer \( n \).

Given an elliptic curve \( E_{a,b}(p) \) over \( \mathbb{Z}_p \), we denote by \( \mathcal{O} \) the point at infinity. The procedure to add two points \( P \) and \( Q \) which has been seen in class can be implemented as follows:

\[
\text{Add1}(E_{a,b}(p), P, Q)
\]

1. if \( x_P \equiv x_Q \pmod{p} \) and \( y_P \equiv -y_Q \pmod{p} \) (equivalent to \( P = -Q \)) then
2. return \( \mathcal{O} \)
3. end if
4. if \( x_P \equiv x_Q \pmod{p} \) and \( y_P \equiv y_Q \pmod{p} \) (equivalent to \( P = Q \)) then
5. set \( u = (2y_P)^{-1} \pmod{p} \)
6. set \( \lambda = ((3x_P^2 + a) \times u) \mod p \)
7. else
We first consider the following algorithm. (Yes, it uses \( p \) but we will later build on it another algorithm ignoring \( p \).)

\[ \text{Proc1}(p) \]
\[
1: \text{pick some random parameters } a, b \in \mathbb{Z}_p, \text{ define the elliptic curve } E_{a,b}(p) \text{ over } \mathbb{Z}_p \text{ by } y^2 = x^3 + ax + b \text{ and pick a random point } S \text{ on } E_{a,b}(p)
\]
\[
2: \text{set } i = 1
\]
\[
3: \text{while } S \neq \mathcal{O} \text{ do}
\]
\[
4: \quad i \leftarrow i + 1
\]
\[
5: \quad S \leftarrow i.S \text{ with the double-and-add algorithm using } \text{Add1}(E_{a,b}(p), P, Q)
\]
\[
6: \text{end while}
\]

We let \( q \) denote the order of \( E_{a,b}(p) \) over \( \mathbb{Z}_p \). We assume that, due to selecting \( a \) and \( b \) at random, \( q \) is a random number between \( p - 2\sqrt{p} \) and \( p + 2\sqrt{p} \).

\[ \text{Q.1} \text{ Show that Proc1 terminates.} \]

\[ \text{Q.2} \text{ Let } M(q) \text{ be the largest prime factor of } q \text{ and } \alpha_j \text{ be the largest integer such that } j^{\alpha_j} \text{ divides } q. \text{ We assume that the probability that } q \text{ is such that we have } \alpha_j \leq \left\lfloor \frac{M(q)}{j} \right\rfloor \text{ for all prime } j \text{ is “very high”, and that the probability that a random point } P \text{ in } E_{a,b}(p) \text{ has an order multiple of } M(q) \text{ is also “very high”.}
\]

Show that when these two conditions are met, Proc1 terminates with the value \( i = M(q) \).

\[ \text{HINT: Show that when the first condition is met, then } q \text{ divides } M(q)!. \]

\[ \text{HINT2: This question may be a bit harder than the next ones.} \]

In what follows, we assume that this implies that the average number of iterations in Proc1 is \( e^{\sqrt{(1+o(1)) \ln p \ln \ln \ln p}}. \)

\[ \text{Q.3} \text{ We change Proc1 into Proc2 by making computations modulo } n \text{ instead of modulo } p. \text{ When adding two points } P \text{ and } Q, \text{ the test } P = Q \text{ and the test } P = -Q \text{ are still done modulo } p. \]

We temporarily assume that we can easily pick an element in the curve at random in the first step of Proc2. Below, we underline what was changed.

\[ \text{Add2}(E_{a,b}(p, n), P, Q) \]
\[
1: \text{if } x_P \equiv x_Q \pmod{p} \text{ and } y_P \equiv -y_Q \pmod{p} \text{ then}
\]
\[
2: \text{return } \mathcal{O}
\]
\[
3: \text{end if}
\]
\[
4: \text{if } x_P \equiv x_Q \pmod{p} \text{ and } y_P \equiv y_Q \pmod{p} \text{ then}
\]
\[
5: \text{set } u = (2y_P)^{-1} \pmod{n} \text{ (abort with an error message if non invertible)}
\]
\[
6: \text{set } \lambda = ((3x_P^2 + a) \times u) \pmod{n}
\]
\[
7: \text{else}
\]
\[
8: \text{set } u = (x_Q - x_P)^{-1} \pmod{n} \text{ (abort with an error message if non invertible)}
\]
\[
9: \text{set } \lambda = ((y_Q - y_P) \times u) \pmod{n}
\]
\[
10: \text{end if}
\]
\[
11: \text{set } x_R = (\lambda^2 - x_P - x_Q) \pmod{n}
\]
\[
12: \text{set } y_R = ((x_P - x_R) \lambda - y_P) \pmod{n} \]
13: return \( R = (x_R, y_R) \)

\textbf{Proc2}(p, n)
1: pick some random parameters \( a, b \in \mathbb{Z}_n \), define the curve \( E_{a,b}(p, n) \) over \( \mathbb{Z}_n \) by \( y^2 = x^3 + ax + b \), and pick a random point \( S \) on \( E_{a,b}(p, n) \)
2: set \( i = 1 \)
3: \textbf{while} \( S \neq \mathcal{O} \) \textbf{do}
4: \hspace{1em} \( i \leftarrow i + 1 \)
5: \hspace{1em} \( S \leftarrow i \cdot S \) with the double-and-add algorithm using \( \text{Add2}(E_{a,b}(p, n), P, Q) \)
6: \textbf{end while}

We execute in parallel \textbf{Proc1} and \textbf{Proc2} with the same random seed. We let \( S_1 \) (resp. \( S_2 \)) designate the value of the register \( S \) in \textbf{Proc1} (resp. \textbf{Proc2}). Show that at every step, \( x_{S_1} \equiv x_{S_2} \pmod{p} \) and \( y_{S_1} \equiv y_{S_2} \pmod{p} \) until \textbf{Proc2} aborts with an error or terminates.

\textbf{Q.4} Transform \text{Add2} so that any abortion yields a non-trivial factor of \( n \) instead of an error.
\textbf{Q.5} Further transform \text{Add2} so that it does not need \( p \) any longer.
\textbf{HINT:} look at what can go wrong if we do the comparisons modulo \( n \).
\textbf{Q.6} Observe that the first step of \textbf{Proc2} cannot be done efficiently. Transform this step to make it doable efficiently and without using \( p \).
\textbf{HINT:} pick \( S \) first!
\textbf{Q.7} Show that the probability that \textbf{Proc2} terminates with an abortion is “very high” based on the assumptions from \textbf{Q.2}. Deduce that we can find the smallest prime factor \( p \) of \( n \) with complexity \( e^{\sqrt{(1+o(1)) \ln p \ln \ln p}} \).
\textbf{HINT:} we do not expect any probability computation, just identify cases when the algorithm does not abort and heuristically justify that this is unlikely to happen.