Cryptography and Security 2017

Exercise Sheet 7

Exercise 1 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 $\mathbf{Z}_p$, 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:

$\mathsf{Add1}(E_{a,b}(p), P, Q)$

if $x_P \equiv x_Q \pmod p$ and $y_P \equiv -y_Q \pmod p$ (equivalent to $P = -Q$) **then**
    return $\mathcal{O}$
end if
if $x_P \equiv x_Q \pmod p$ and $y_P \equiv y_Q \pmod p$ (equivalent to $P = Q$) **then**
    set $u = (2y_P)^{-1} \bmod p$
    set $\lambda = ((3x_P^2 + a) \times u) \bmod p$
else
    set $u = (x_Q - x_P)^{-1} \bmod p$
    set $\lambda = ((y_Q - y_P) \times u) \bmod p$
end if
set $x_R = (\lambda^2 - x_P - x_Q) \bmod p$
set $y_R = ((x_P - x_R)\lambda - y_P) \bmod p$
return $R = (x_R, y_R)$

We first consider the following algorithm. (Yes, it uses p but we will later build on it another algorithm ignoring p.)

$\mathsf{Proc1}(p)$

pick some random parameters $a, b \in \mathbf{Z}_p$, define the elliptic curve $E_{a,b}(p)$ over $\mathbf{Z}_p$ by $y^2 = x^3 + ax + b$ and pick a random point S on $E_{a,b}(p)$
set $i = 1$
while $S \neq \mathcal{O}$ **do**
    $i \leftarrow i + 1$
    $S \leftarrow i.S$ with the double-and-add algorithm using $\mathsf{Add1}(E_{a,b}(p), P, Q)$
end while

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

1. Show that Proc1 terminates.

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

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

   HINT: Show that when the first condition is met, then q divides $M(q)!$.

   HINT^2: 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 $e^{\sqrt{(1+o(1))\ln p \ln\ln p}}$.

3. We change Proc1 into Proc2 by making computations modulo n instead of modulo p. When adding two points P and Q, the test $P = Q$ and the test $P = -Q$ 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.

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

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

   We execute in parallel Proc1 and Proc2 with the same random seed. We let S_1 (resp. S_2) designate the value of the register S in Proc1 (resp. 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 Proc2 aborts with an error or terminates.

4. Transform Add2 so that any abortion yields a non-trivial factor of n instead of an error.

5. Further transform Add2 so that it does not need p any longer.

   HINT: look at what can go wrong if we do the comparisons modulo n.

6. Observe that the first step of Proc2 cannot be done efficiently. Transform this step to make it doable efficiently and without using p.

   HINT: pick S first!

7. Show that the probability that Proc2 terminates with an abortion is "very high" based on the assumptions from 2. Deduce that we can find the smallest prime factor p of n with complexity $e^{\sqrt{(1+o(1))\ln p \ln\ln p}}$.

   HINT: we do not expect any probability computation, just identify cases when the algorithm does not abort and heuristicaly justify that this is unlikely to happen.

Exercise 2 Weak Keys of DES

We say that a DES key k is *weak* if DES_k is an involution. Exhibit four weak keys for DES.
Reminder: Let $\mathcal{S}$ be a finite set and let f be a bijection from $\mathcal{S}$ to $\mathcal{S}$. The function f is an *involution* if $f(f(x)) = x$ for all $x \in \mathcal{S}$.

Note: PC1 and PC2 are permutations you don't need to know.

Exercise 3 Complementation Property of DES

Given a bitstring x we let $\overline{x}$ denote the bitwise complement, i.e., the bitstring obtained by flipping all bits of x.

1. Prove that
$$\mathsf{DES}_{\overline{K}}(\overline{x}) = \overline{\mathsf{DES}_K(x)}$$
for any x and K.

2. Deduce a brute force attack against DES with average complexity of 2^{54} DES encryptions.
Hint: Assume that the adversary who is looking for K is given a plaintext block x and the two values corresponding to $\mathsf{DES}_K(x)$ and $\mathsf{DES}_K(\overline{x})$.

Exercise 4 A Weird Mode of Operation

In this exercise, we assume that we have a block cipher C and we use it in the following mode of operation: to encrypt a sequence of blocks $x_1, \ldots, x_n$, we initialize a counter t to some IV value, then we compute
$$y_i = t_i \oplus C_K(x_i)$$
for every i where K is the encryption key and $t_i = \mathsf{IV} + i$. The ciphertext is
$$\mathsf{IV}, y_1, \ldots, y_n$$
Namely, IV is sent in clear.

1. Is this mode of operation equivalent to something that you already know? Say why?
2. Does the IV need to be unique?
3. What kind of security problem does this mode of operation suffer from?