Foundations of Cryptography – Problem Set 1
Solution of Question 1

Question 1.
In what complexity class does the problem of inverting one-way permutation reside? Recall that we showed that the problem of inverting one-way functions is in NP.

Answer 1.
In class we proved that if P = NP then there are no one-way functions. By following the exact same proof, we refine the above statement by proving that if P = NP ∩ coNP then there are no one-way permutations.

Given a function $f : \{0,1\}^n \to \{0,1\}^m$, define the language

$$L_f = \{(y, b_1, \ldots, b_k) \mid \exists x \in \{0,1\}^n \text{ s.t. } f(x) = y \text{ and } (b_1, \ldots, b_k) \text{ is a prefix of } x\} \quad .$$

We observed in class that for any polynomial time computable function f, the language L_f is an NP language. We further observed that any polynomial time algorithm for deciding membership in L_f can be used as a subroutine to *always* invert f on *any* input in polynomial time (by finding a preimage bit by bit). Thus, if P = NP then there are no one-way functions.

Now, in the case that the function f is a polynomial time computable *permutation*, the language L_f is also a coNP language. This can be seen, as a witness for the non-membership of $(y, b_1, \ldots, b_k)$ is the *unique* x such that $f(x) = y$ (and $(b_1, \ldots, b_k)$ is not its prefix).

Therefore, for any polynomial time computable permutation f, the language L_f is in NP ∩ coNP, and as before, any polynomial time algorithm for deciding membership in L_f can be used as a subroutine to *always* invert f on *any* input in polynomial time. Thus, if P = NP ∩ coNP then there are no one-way permutations.