Solution Sheet #2

Advanced Cryptography 2021

Solution 1 Primes

1. Note that $\overline{\text{PRIMES}}$ is the language of composite numbers (and 1, but this case is easy to deal with) and we have to show that it's in NP. A trivial witness is to give the factors of the number. Verifying these factors is done in polynomial time.

2. Recall that $\mathbb{Z}_p^*$ is a cyclic group of order $p-1$ if and only if p is prime.

   As a witness, we will use a generator g of $\mathbb{Z}_p^*$ and we check whether it indeed generates $\mathbb{Z}_p^*$. To check that it generates $\mathbb{Z}_p^*$, we need to have full factorization of the prime decomposition of $p-1$. Let $p-1 = \prod_{i=1}^{k} p_i^{\alpha_i}$. Then, we can then check that $g^{(p-1)/p_i} \neq 1$ for all prime divisor p_i and that $p_1^{\alpha_1} \cdot \ldots \cdot p_k^{\alpha_k} = p-1$. Hence, up to now, our witness consists of generator g, primes p_is and their powers α_i.

   The problem is that we need now to be sure that the p_i values are prime. We do this by performing the same technique on each of them (making recursive calls to our own verificatin algorithm). Let $W(p)$ be our witness. We have

$$W(p) = (g, p_1, \ldots, p_k, \alpha_1, \ldots, \alpha_k, W(p_1), \ldots, W(p_k)).$$

```
procedure VERIFY_PRIME(p,W(p))
    If p = 2 return 1                                  ▷ Condition for end of recursion
    W(p) → (g, p_1, ..., p_k, α_1, ..., α_k, W(p_1), ... W(p_k))
    Check p − 1 = ∏_{i=1}^{k} p_i^{α_i}
    Check g^{(p−1)/p_i} ≠ 1 for all i = 1, ..., k
    Check g^{p−1} mod p = 1
    Check Verify_Prime(p_i, W(p_i)) = 1 for all i = 1, ..., k
    Return 1 if all satisfed
    Return 0 if any check fails
end procedure
```

Figure 1: A verification algorithm for PRIMES

Witness size: We need now to compute the size of our witness and make sure it's polynomial. First, note that there can be at most $\log(p)$ prime factors and each of them can be written using $\log(p)$ bits. Let $T(p)$ be the size of our witness. We have

$$T(p) \leq (\log p)^2 + \sum_{i=1}^{j} T(p_i) \ .$$

By the second hint, we get $T(p) = O((\log p)^3)$ which is polynomial in the size of the input.

Computation time: The computation can be divided into three parts:

- Checking $p - 1 = \prod_{i=1}^{k} p_i^{\alpha_i}$ takes $\sum_{i=1}^{k} \alpha_i$ multiplication, where each element can be at most $\log p$ bits. Multiplying two integers a and b takes exactly $\log a \times \log b$ operations, therefore this step takes at most $(\log p)^2 \sum_{i=1}^{k} \alpha_i$ operations. We also know that $\sum_{i=1}^{k} \alpha_i \leq (\log p)^2$, therefore we can bound this step by $(\log p)^4$.
- Each check $g^{\frac{p-1}{p_i}} \neq 1$ takes $O((\log p)^3)$ multiplications, and there can be at most $\log p$ different prime factors; hence this step is bounded by $O((\log p)^4)$ operations.
- Similar operations should be done for the verification of $W(p_i)$ witnesses. We notice that these upper bounds also directly apply to any sub-computations as well, since all elements are also at most $\log p$ size. Therefore for each recursion of $\mathsf{Verify\_Prime}$, the computations are bounded by $O((\log p)^4)$.
- The final step is to show that number of recursions are bounded polynomially. We use the given hint to bound this. Namely, define function $T(p)$ as the number of calls made to $\mathsf{Verify\_Prime}$ with input $(p, W(p))$. We have that $T(p) \leq \log p + \sum_{i=1}^{k} T(p_i)$. Given hint implies that $T(p) \leq (\log p)^3$.

Gathering all together, we conclude that running time of $\mathsf{Verify\_Sign}$ is loosely bounded by $O((\log p)^3) \times O((\log p)^4) = O((\log p)^7)$. Thus, PRIMES $\in$ NP.

Solution 2 Fixed Point Attack on RSA

Since $c^{e^k} \equiv c \pmod N$, then $c^{e^k} \equiv m^e \pmod N \Rightarrow c^{e^{k-1}e} \equiv m^e \pmod N \Rightarrow (c^{e^{k-1}})^e \equiv m^e \pmod N \Rightarrow c^{e^{k-1}} \equiv m \pmod N$.

Hence, it suffices to iterate the RSA encryption until we obtain again the ciphertext. This will give us k. The previous value ($c^{e^{k-1}}$) was the plaintext m.

Nevertheless, it has been shown that the probability for such an attack to succeed is negligible if the primes p and q are chosen at random with a sufficient size (see Rivest-Silverman "Are strong primes needed for RSA").

Solution 3 Turing Machines

1. By definition, we know that the recursively enumerable language requires the existence of a Turing machine, such that it eventually enters a final state q_{accept} (and halts) for all inputs in the language, but it may never halt on the input that is not in the language. Therefore, a recursive language is always recursively enumerable.

2. From the last question, we know that there exists a Turing machine (denoted M) that accepts L, which has two halting states q_{accept} and q_{reject}. We modify M as follows (where M' denotes the modified Turing machine): for all the state transitions involving q_{accept} or q_{reject}, we replace q_{accept} (respectively, q_{reject}) by q_{reject} (resp. q_{accept}). To complete the proof, it suffices to check the following:

- for any $\omega \notin \overline{L}$ (i.e., $\omega \in L$), M' eventually enters the halting state q_{reject} and rejects it;
- for any $\omega \in \overline{L}$ (i.e., $\omega \notin L$), M' eventually enters the halting state q_{accept}, accepts it and halts.