Homework 4

1. **An Example of Extended GCD Algorithm (20 points).** Recall that the extended GCD algorithm takes as input two integers a, b and returns a triple (g, α, β), such that

$$g = \gcd(a, b), \text{ and } g = \alpha \cdot a + \beta \cdot b.$$

Here $+$ and $\cdot$ are integer addition and multiplication operations, respectively.

Find (g, α, β) when $a = 2024, b = 164$.

Solution.

$$\begin{aligned} 2024 &= 12 \cdot 164 + 56 \\ 164 &= 2 \cdot 56 + 52 \\ 56 &= 1 \cdot 52 + 4 \\ 52 &= 13 \cdot 4 + 0 \end{aligned}$$

This tells us $\gcd(2024, 164) = 4$.

We can rewrite them as:

$$\begin{aligned} 56 &= 2024 - 12 \cdot 164 \\ 52 &= 164 - 2 \cdot 56 \\ 4 &= 56 - 1 \cdot 52 \end{aligned}$$

Now by doing substitution, we get:

$$\begin{aligned} 4 &= 56 - 1 \cdot 52 \\ &= 56 - 1 \cdot (164 - 2 \cdot 56) \\ &= 56 - 1 \cdot 164 + 2 \cdot 56 \\ &= -1 \cdot 164 + 3 \cdot 56 \\ &= -1 \cdot 164 + 3 \cdot (2024 - 12 \cdot 164) \\ &= -1 \cdot 164 + 3 \cdot 2024 - 36 \cdot 164 \\ &= 3 \cdot 2024 - 37 \cdot 164 \end{aligned}$$

We get $\alpha = 3$ and $\beta = -37$.

We can verify that $4 = 3 \cdot 2024 - 37 \cdot 164$

The answer is $(4, 3, -37)$

2. **(20 points).** Suppose a cryptographic protocol P_n is implemented using αn^2 CPU instructions, where α is some positive constant. We expect the protocol to be broken with $\beta 2^{n/10}$ CPU instructions.

   Suppose, today, everyone in the world uses the primitive P_n using $n = n_0$, a constant value such that even if the entire computing resources of the world were put together for 8 years, we cannot compute $\beta 2^{n_0/10}$ CPU instructions.

   Assume Moore's law holds. That is, every two years, the amount of CPU instructions a CPU can run per second doubles.

   Remark: This problem explains why we demand that our cryptographic algorithms run in polynomial time and it is exponentially difficult for adversaries to break the cryptographic protocols.

   (a) (5 points) Assuming Moore's law, how much faster will the CPUs be 8 years into the future compared to now?

   Solution.
   The CPUs will be $2^{8/2} = 16$ times faster.

   (b) (5 points) At the end of 8 years, what choice of n_1 will ensure that setting $n = n_1$ will ensure that the protocol P_n for $n = n_1$ cannot be broken for another 8 years? (Recall that currently, setting $n = n_0$ ensures that the adversaries need to run $\beta 2^{n_0/10}$ instruction to break the protocol, which they are unable to do even in 8 years.)

   Intuition: Since future computers (8 years later from today) are now faster (based on your answer in part (a)), we need to set our parameters to a larger value to ensure that securities still hold. Your task is to determine how large this new parameter n_1 needs to be compared to the current parameter n_0. Your answer should be an equation for n_1, in terms of n_0 and/or other variables.

   Hint: Start by assuming that the old computers are able to run Γ instruction over an 8-year period. And the new computers are able to run $x \cdot \Gamma$ instruction over an 8-year period.

   Solution.

   Currently, it is given that $\frac{\beta 2^{n_0/10}}{\Gamma} = $ 8-years. We want $\frac{\beta 2^{n_1/10}}{16\Gamma} = $ 8-years. That is

$$16 \cdot 2^{n_0/10} = 2^{n_1/10} \iff n_1 = n_0 + 40$$

(c) (5 points) What will be the run-time of the protocol P_n using $n = n_1$ on the new computers as compared to the run-time of the protocol P_n using $n = n_0$ on today's computers? (Recall that P_n is implemented using αn^2 CPU instructions.)

Hint: Start by assuming that today computers are able to run Γ instructions per second.

Your answer should be a ratio of the new run time divided by the old run time.

Solution.

Today, suppose, a honest person runs Γ instructions per second. The run-time of P_{n_0} on today's computers is

$$\frac{\alpha n_0^2}{\Gamma}$$

8-years from now, honest people will run 16Γ instructions per second on new computers. The run-time of P_{n_1} on those computers is

$$\frac{\alpha n_1^2}{16\Gamma} = \frac{\alpha(n_0+40)^2}{16\Gamma} = \frac{\alpha(n_0/4+10)^2}{\Gamma}$$

The ratio of second-run-time to first-run-time is

$$\left(\frac{1}{4} + \frac{10}{n_0}\right)^2$$

Observe: This ratio can be much less than 1. That is, in future, honest people will be able to run the protocol faster!

(d) (5 points) What will be the run-time of the protocol P_n using $n = n_1$ on today's computers as compared to the run-time of the protocol P_n using $n = n_0$ on today's computers? (Recall that P_n is implemented using αn^2 CPU instructions.)

Your answer should be a ratio of the new run time divided by the old run time.

Solution.

Suppose the honest person did not upgrade his CPU. So, he is running Γ instructions per second in the future as well. Then, the running time of P_{n_1} on this computer is

$$\frac{\alpha n_1^2}{\Gamma} = \frac{\alpha(n_0+40)^2}{\Gamma}$$

Ratio of this time to run-time of P_{n_0} is

$$\frac{(n_0+40)^2}{n_0^2} = \left(1 + \frac{40}{n_0}\right)^2$$

Observe: If n_0 is much larger than 40 then the running-time of P_{n_1} on old processors is not much different from running-time of P_{n_0} on old processors!

3. **Finding Inverse Using Extended GCD Algorithm (20 points).** In this problem, we shall work over the group $(\mathbb{Z}^*_{1321}, \times)$. Note that 1321 is a prime. The multiplication operation $\times$ is "integer multiplication $\mod 1321$."

   Use the Extended GCD algorithm to find the multiplicative inverse of 47 in the group $(\mathbb{Z}^*_{1321}, \times)$.

   Solution.

$$
\begin{aligned}
1321 &= 28 \cdot 47 + 5\\
47 &= 9 \cdot 5 + 2\\
5 &= 2 \cdot 2 + 1\\
2 &= 2 \cdot 1
\end{aligned}
$$

$$
\begin{aligned}
1 &= 5 - 2 \cdot 2\\
&= 5 - 2 \cdot (47 - 9 \cdot 5)\\
&= -2 \cdot 47 + 19 \cdot 5\\
&= -2 \cdot 47 + 19 \cdot (1321 - 28 \cdot 47)\\
&= 19 \cdot 1321 - 534 \cdot 47
\end{aligned}
$$

   Since $1 = 19 \cdot 1321 - 534 \cdot 47$. The inverse of 47 is -534, which is 787.

4. **Another Application of Extended GCD Algorithm (20 points).** Use the Extended GCD algorithm to find $x \in \{0, 1, 2, \ldots, 1007\}$ that satisfies the following two equations.

$$x = 3 \mod 63$$
$$x = 4 \mod 16$$

Note that 63 and 16 are not primes. However, we have the guarantee that 63 and 16 are relatively prime, that is, $\gcd(63, 16) = 1$. Also, note that the number $1007 = 63 \cdot 16 - 1$.

Solution.
We first run extended GCD Algorithm on 63 and 16.

$$63 = 3 \cdot 16 + 15$$
$$16 = 1 \cdot 15 + 1$$
$$15 = 15 \cdot 1 + 0$$

$$\begin{aligned} 1 &= 16 - 1 \cdot 15 \\ &= -1 \cdot 63 + 4 \cdot 16 \end{aligned}$$

Since we know that $1 = -1 \cdot 63 + 4 \cdot 16$. We know that

$$-1 \cdot 63 + 4 \cdot 16 \mod 63 = 4 \cdot 16 \mod 63 = 1 \mod 63$$

and

$$-1 \cdot 63 + 4 \cdot 16 \mod 16 = -1 \cdot 63 \mod 16 = 1 \mod 16.$$

Let us look at

$$4 \cdot -1 \cdot 63 + 3 \cdot 4 \cdot 16.$$

$$4 \cdot -1 \cdot 63 + 3 \cdot 4 \cdot 16 \mod 63 = 3 \cdot 4 \cdot 16 \mod 63 = 3 \mod 63.$$

$$4 \cdot -1 \cdot 63 + 3 \cdot 4 \cdot 16 \mod 16 = 4 \cdot -1 \cdot 63 \mod 16 = 4 \mod 16.$$

The answer is therefore $4 \cdot -1 \cdot 63 + 3 \cdot 4 \cdot 16$, or -60. Observe that if we add $63 \cdot 16$, then nothing $\mod 63$ or $\mod 16$ changes. Therefore, the answer is $-60 + 1008 = 948$. Note that this is the same as converting -60 into an element of the integer ring mod 1008.

5. **Square Root of an Element (20 points).** Let p be a prime such that $p = 3 \mod 4$. For example, $p \in \{3, 7, 11, 19 \dots\}$.

We say that x is a square-root of a in the group $(\mathbb{Z}_p^*, \times)$ if $x^2 = a \mod p$. We say that $a \in \mathbb{Z}_p^*$ is a quadratic residue if $a = x^2 \mod p$ for some $x \in \mathbb{Z}_p^*$. Prove that if $a \in \mathbb{Z}_p^*$ is a quadratic residue then $a^{(p+1)/4}$ is a square-root of a.

(Remark: This statement is only true if we assume that a is a quadratic residue. For example, when $p = 7$, 3 is not a quadratic residue, so $3^{(7+1)/4}$ is not a square root of 3.)

Solution.

Since $a \in \mathbb{Z}_p^*$ is a quadratic residue, we have $a = x^2 \mod p$ for some $x \in \mathbb{Z}_p^*$. Then, since $x \in \mathbb{Z}_p^*$, we have $x^{p-1} \mod p = 1$, and so:

$$
\begin{aligned}
& a^{\frac{(p+1)}{4}} = x^{\frac{p+1}{2}} \mod p \\
\implies & \left(a^{\frac{(p+1)}{4}}\right)^2 = \left(x^{\frac{p+1}{2}}\right)^2 = x^{p+1} = x^{p-1} \times x^2 = 1 \times a = a
\end{aligned}
$$

This proves that $a^{\frac{p+1}{4}}$ is a square-root of a.

6. **Weak One-way Functions (20 points).** Define $S_n = \{0,1\}^n \setminus \{ \{0,1\} \}$. That is, S_n is all n-bit numbers except 0 and 1. Let $h_n \colon S_n \times S_n \to \{0,1\}^{2n}$ be the product function $f(x_1, x_2) = x_1 \cdot x_2$.

   Present an adversarial algorithm $\mathcal{A} \colon \{0,1\}^{2n} \to S_n \times S_n$ that successfully inverts this function with a constant probability when $(x_1, x_2) \xleftarrow{\$} S_n \times S_n$. Compute the probability of your algorithm successfully inverting the function h_n.

   Hint: Intuitively, to invert the function is equivalent to finding one factor of a number. Can you find a factor that shows up with constant probability?

   Hint: Your algorithm is allowed to fail with constant probability. This also means you are allowed to design an algorithm that sometimes (with constant probability) "gives up" and outputs wrong/arbitrary/dummy values.

   Solution.

   The solution is based on the observation that the probability that an element in S_n is divisible by 2 is $\frac{1}{2}$ because

$$S_n = \{2, 3, 4, 5, 6, \cdots, 2^n - 2, 2^n - 1\}$$

   Adversarial Algorithm $\mathcal{A} \colon \{0,1\}^{2n} \to S_n \times S_n$:

   $y \leftarrow \{0,1\}^{2n}$
   if $y \mod 2 == 0$ **then**
      return $(y/2, 2)$
   else
      return $(1, 1)$
   end if

$$\begin{aligned}
\mathbb{P}[\mathsf{success}] &= \mathbb{P}[(x_1 \mod 2 = 0) \cup (x_2 \mod 2 = 0)] \\
&= 1 - \mathbb{P}[(x_1 \mod 2 \neq 0) \cap (x_2 \mod 2 \neq 0)] \\
&= 1 - \mathbb{P}[x_1 \mod 2 \neq 0] \cdot \mathbb{P}[x_2 \mod 2 \neq 0] \\
&= 1 - \left(\frac{1}{2}\right)^2 \\
&= \frac{3}{4}
\end{aligned}$$

Collaborators :