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.
From lecture, we know that $g = gcd(a, b), WLOG,\ a \leqslant b$ and $g = \alpha \cdot a + \beta \cdot b$.
Using $XGCD\ algorithm$ from lecture,

$$B/A = (M \cdot A, R)$$

$$2024/164 = (12 \cdot 164, 2024 - (12 \cdot 164))$$

$$= (12 \cdot 164, 56)$$

$$164/56 = (2 \cdot 56, 52)$$

$$56/52 = (1 \cdot 52, 4)$$

$$52/4 = (13 \cdot 4, 0)$$

$$4/0 = return\ (0, 1, 4)$$

Then, unfolding the recursion, we have:

$$(\alpha', \beta', g), \alpha, \beta$$

$$(\alpha^{'}, \beta', g) = (0, 1, 4), \alpha = 1 - 0(13) = 1, \beta = 0$$

$$(\alpha^{'}, \beta', g) = (1, 0, 4), \alpha = 0 - 1(1) = -1, \beta = 1$$

$$(\alpha^{'}, \beta', g) = (-1, 1, 4), \alpha = 1 - (-1)(2) = 3, \beta = -1$$

$$(\alpha^{'}, \beta', g) = (3, -1, 4), \alpha = -1 - 12(3) = -37, \beta = 3$$

Therefore, final (α, β, g) returned: $(-37, 3, 4)$
Check: $\alpha \cdot a + \beta \cdot b = -37(164) + 3(2024) = 4 = g$
Hence, (g, α, β) when $a = 2024, b = 164$: $(4, -37, 3)$

2. **Asymptotics and Efficient Algorithms (20 points).** Suppose a cryptographic protocol P_n is implemented using αn^2 CPU instructions. We expect the protocol to be broken with $\beta 2^{n/10}$ CPU instructions. α and β are some positive constants, while n is the parameter of the protocol (such as key length in bits). That is, when we set the parameter $n = \sigma$, to use the protocol, the "good guys" need to run $\alpha\sigma^2$ CPU instructions. While to break the protocol, the "bad guys" need to run $\beta 2^{\sigma/10}$ 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.
   Using Moore's Law,
   CPU 8 years into the future will be $2^{\frac{8}{2}} = 2^4 = 16$ 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.

Let γ denote the CPU speed of the current computers.

Thus, to ensure that the same securities still hold, that they cannot be broken for another 8 years,

$$\frac{\beta \cdot 2^{n_0/10}}{\gamma} = \frac{\beta \cdot 2^{n_1/10}}{16 \cdot \gamma}$$

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

$$log_2(16 \cdot 2^{n_0/10}) = log_2(2^{n_1/10})$$

$$log_2 16 + \frac{n_0}{10} log_2(2) = \frac{n_1}{10} log_2(2)$$

$$40 + n_0 = n_1$$

$$n_1 = 40 + n_0$$

(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.

Since today's computers are able to run Γ instructions per second, and it takes αn_0^2 CPU instructions for the "good guys" to run it, the time taken would be $\frac{\alpha n_0^2}{\Gamma}$.

Then, from part (a), new computers would be able to run at $16 \cdot \Gamma$ instructions per second, and thus the time taken would be $\frac{\alpha n_1^2}{16 \cdot \Gamma}$, for new computers.

Then, from part (b), $n_1 = n_0 + 40$,

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

$$= \frac{(n_0 + 40)^2}{16 \cdot n_0^2}$$

$$= \frac{n_0^2 + 80n_0 + 1600}{16n_0^2}$$

$$= \frac{1}{16} + \frac{5}{n_0} + \frac{10}{n_0^2}$$

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

(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.

Using today's computers that can only run Γ instructions per second and the protocol P_n using $n = n_1$ has $\alpha n_1^2 = \alpha(n_0 + 40)^2$ instructions (From part (b)),
The ratio of new run time divided by old run time =

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

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

$$= \frac{n_0^2 + 80n_0 + 1600}{n_0^2}$$

$$= 1 + \frac{80}{n_0} + \frac{1600}{n_0^2}$$

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

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.

   First, we note that $47 \in \mathbb{Z}^*_{1321} = \{1, 2, ..., 1320\}$.
   Then, applying $XGCD(47, 1321)$:

$$B/A = (M \cdot A, R)$$

$$1321/47 = (28 \cdot 47, 5)$$

$$47/5 = (9 \cdot 5, 2)$$

$$5/2 = (2 \cdot 2, 1)$$

$$2/1 = (2 \cdot 1, 0)$$

   Lastly, return (0, 1, 1) since R = 0.
   Then, we can unroll the recursion:

$$(\alpha^{'}, \beta', g), \alpha, \beta$$

   :

$$(0, 1, 1), \alpha = 1 - 2(0) = 1, \beta = 0$$

$$(1, 0, 1), \alpha = 0 - 2(1) = -2, \beta = 1$$

$$(-2, 1, 1), \alpha = 1 - 9(-2) = 19, \beta = -2$$

$$(19, -2, 1), \alpha = -2 - 19(28) = -534, \beta = 19$$

   Finally, return $(\alpha, \beta, g) = (-534, 19, 1)$.
   Thus, $\alpha\ (mod\ p) = 787\ (mod\ p)$ is the multiplicative inverse of 47 in $\mathbb{Z}^*_{1321}$.

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 is a prime, but 16 is not a prime. 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.

Applying XGCD from lecture:
XGCD(16, 63):

$$B/A = M \cdot A + R$$

$$63/16 = 3 \cdot 16 + 15$$

$$16/15 = 1 \cdot 15 + 1$$

$$15/1 = 15 \cdot 1 + 0$$

Lastly, return $(0, 1, 1)$ since R = 0 above.
Then, we can unroll the recursion:

$$(\alpha^{'}, \beta', g), \alpha, \beta$$

:

$$(0, 1, 1), \alpha = 1 - 0(15) = 1, \beta = 0$$

$$(1, 0, 1), \alpha = 0 - 1(1) = -1, \beta = 1$$

$$(-1, 1, 1), \alpha = 1 - (-1)(3) = 4, \beta = -1$$

Finally, return $(\alpha, \beta, g) = (4, -1, 1)$.
We can note that $4 \times 16 + (-1) \times 63 = 1 = g$.
Then, we can also note that the following also follows:
$4 \times 16 \ (mod\ 63) = 1$
$-1 \times 63 \ (mod\ 16) = 1$

Next, we can also denote the following:

$$(16 \times 4 \times 3 + 63 \times (-1) \times 4) \ mod \ 16 = (0 + (-1 \times 63 \ mod \ 16)(4 \ mod \ 16))$$

$$= 1 \times 4 \mod 16$$

(Since $(-1) \times 63 \ (mod \ 16) = 1$ from above)

$$= 4$$

And also:

$$(16 \times 4 \times 3 + 63 \times (-1) \times 4) \ mod \ 63 = (16 \times 4 \ mod \ 63)(3 \ mod \ 63) + 0$$

$$= 1 \times 3 \ mod \ 63$$

(Since $4 \times 16 \ (mod \ 63) = 1$ from above)

$$= 3$$

Thus, we can obtain a solution by doing the following:
$x = 16 \times 4 \times 3 + (-1) \times 63 \times 4 \ (mod \ 1008)$
$x = 948$.

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

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.

First, we note that since $a \in \mathbb{Z}_p^*$ is a quadratic residue, then $a = x^2 \ mod \ p$ for some $x \in \mathbb{Z}_p^*$ (by definition in the question).
Thus,

$$a^{\frac{p+1}{4}} = x^{\frac{p+1}{2}}$$

.

$$\Rightarrow (a^{\frac{p+1}{4}})^2 = (x^{\frac{p+1}{2}})^2$$

(Taking square on both sides)

$$= x^{p+1} \ mod \ p$$

$$= x^{p-1} \cdot x^2 \ mod \ p$$

$$= x^{p-1} \cdot a$$

(Since $a = x^2 \ mod \ p$ if $a \in \mathbb{Z}_p^*$ is a quadratic residue)

$$= 1 \cdot a$$

(Since $x^{p-1} \ mod \ p = 1$ by Fermat's Little Theorem

$$= a$$

Thus, 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.

   Algorithm is as follows:
   Upon receiving $z = f(x_1, x_2)$, check if z is an even number.
   If z is even, output $(2, \frac{z}{2})$ as answer
   Else, output $(0, 0)$ as dummy values, indicating that the algorithm gave up.

   Since $Pr[x_1 \textit{ is even}] = Pr[x_2 \textit{ is even}] = \frac{1}{2}$, $Pr[z = x_1 \cdot x_2 \textit{ is even}] = \frac{3}{4}$. Hence, this algorithm will successfully invert the function h_n with probability $\frac{3}{4}$ and fail with probability $\frac{1}{4}$, as the probability that the input is even is equal to $\frac{3}{4}$.
   Hence, the above is an adversarial algorithm that will successfully invert the function h_n with a constant probability.

Collaborators : Josh Tseng, Rohan Purandare, Nate Johnson, Adam Nasr