Algebraic Cryptography
Homework 2

Due Monday, 18 September 2017

Problem 1. Explain in detail the *square and multiply* algorithm for computing modular exponentiation efficiently. Use this method to compute

$$2956^{2039} \pmod{5219}.$$

Note: you can use a computer (a standard scientific calculator should suffice, otherwise you are not using an efficient algorithm!) to do each of the multiplications and modular reductions, but I should be able to see the algorithm you described at work in the example.

Proof. To compute a modular exponentiation such as $a^e \pmod{n}$ via the square and multiply method, do the following. Write e in binary as $e = \sum_{j=0}^{k} c_j 2^j$ where $c_j = 0, 1$. Then compute the sequence $a \pmod{n}, a^2 \pmod{n}, a^{2^2} \pmod{n}, \ldots, a^{2^k} \pmod{n}$ by squaring the previous term, and then reducing modulo n. Finally, notice that

$$a^e \pmod{n} = a^{\sum_{j=0}^{k} c_j 2^j} \pmod{n} = \prod_{j=0}^{k} a^{c_j 2^j}.$$

To compute the product on the right, notice that if $c_j = 0$, this term contributes nothing to the product, and if $c_j = 1$, we have already computed that term. To conclude, simply multiply the product term-by-term, reducing modulo n at each step.

We now show this with the example provided in the question. Notice that $2039 = (11111110111)_2$, and so we compute:

$$\begin{aligned}
2956^2 &\pmod{5219} = 8737936 \pmod{5219} = 1330 \pmod{5219} \\
1330^2 &\pmod{5219} = 1768900 \pmod{5219} = 4878 \pmod{5219} \\
4878^2 &\pmod{5219} = 23794884 \pmod{5219} = 1463 \pmod{5219} \\
1463^2 &\pmod{5219} = 23794884 \pmod{5219} = 579 \pmod{5219} \\
579^2 &\pmod{5219} = 335241 \pmod{5219} = 1225 \pmod{5219} \\
1225^2 &\pmod{5219} = 1500625 \pmod{5219} = 2772 \pmod{5219}
\end{aligned}$$

$$\begin{aligned}
2772^2 \pmod{5219} &= 7683984 \pmod{5219} = 1616 \pmod{5219} \\
1616^2 \pmod{5219} &= 2611546 \pmod{5219} = 1956 \pmod{5219} \\
1956^2 \pmod{5219} &= 3825936 \pmod{5219} = 409 \pmod{5219} \\
409^2 \pmod{5219} &= 167281 \pmod{5219} = 273 \pmod{5219}
\end{aligned}$$

Then

$$\begin{aligned}
2956^{2039} \pmod{5219} &= \prod_{\substack{j=0 \\ j\neq 3}}^{10} 2956^{2^j} \pmod{5219} \\
&= 2956 \cdot 1330 \cdot \prod_{\substack{j=2 \\ j\neq 3}}^{10} 2956^{2^j} \pmod{5219} \\
&= 1573 \cdot 4878 \cdot \prod_{j=4}^{10} 2956^{2^j} \pmod{5219} \\
&= 1164 \cdot 579 \cdot \prod_{j=5}^{10} 2956^{2^j} \pmod{5219} \\
&= 705 \cdot 1225 \cdot \prod_{j=6}^{10} 2956^{2^j} \pmod{5219} \\
&= 2490 \cdot 2772 \cdot \prod_{j=7}^{10} 2956^{2^j} \pmod{5219} \\
&= 2762 \cdot 1616 \cdot \prod_{j=8}^{10} 2956^{2^j} \pmod{5219} \\
&= 1147 \cdot 1956 \cdot \prod_{j=9}^{10} 2956^{2^j} \pmod{5219} \\
&= 4851 \cdot 409 \cdot \prod_{j=10}^{10} 2956^{2^j} \pmod{5219} \\
&= 8 \cdot 273 \pmod{5219} \\
&= 2184 \pmod{5219}
\end{aligned}$$

■

Problem 2. All the exercises for §1 of Chapter 2. There are 15, but they should each take less than 1 minute. You do not need to provide proofs.

Solution. 1. $\binom{n}{3} = \frac{n(n-1)(n-2)}{6} = \Theta(n^3)$.

2. $10 \ln^3 n + 20n^2 = \Theta(n^2)$.

3. The number of monomials in x, y, z of total degree at most n. Notice that these have the form $x^i y^j z^k$ with $i+j+k \leq n$. Thus, it is clear that this is $O(n^3)$ since each of i, j, k is between 0 and n. However, it is even $\Theta(n^3)$. Indeed, notice that we can choose any values of i, j, k with each between 0 and $\lfloor \frac{n}{3} \rfloor$. Of which there are approximately (i.e., asymptotically) $\frac{n^3}{27}$.

4. The number of polynomials in x of degree at most n whose coefficients are either 0 or 1. There are exactly 2^{n+1} of these, so it is $\Theta(2^n)$.

5. The number of polynomials in x of degree at most $n-1$ whose coefficients are between 0 and n. There are exactly n^{n+1} of these, so it is $\Theta(n^n)$

6. The area of a fixed shape after it's magnified by a factor of n. Assuming we are talking about a shape in the plane, this is $O(n^2)$. *However*, it should be noted that this is *not* $\Theta(n^2)$; for example, take something whose boundary has Hausdorff dimension strictly less than 2, like the Koch snowflake (boundary has dimension $\log_3 4$).

7. The amount of memory space a computer requires to store the number n is $\lceil \log_2 n \rceil$ with is $\Theta(\ln n)$.

8. The amount of memory space a computer requires to store n^2 is $\lceil \log_2 n^2 \rceil = \lceil 2 \log_2 n \rceil$ with is $\Theta(\ln n)$.

9. The sum of the first n positive integers is $\frac{n(n+1)}{2} = \Theta(n^2)$.

10. The number of bits in the sum of the squares of the first n positive integers. Well, the sum of the squares of the first n positive integers is $\frac{n(n+1)(2n+1)}{6}$ which is $\Theta(n^3)$. But we want the number of bits, which is the ceiling of log_2 of this number. In particular, the number of bits is $\Theta(\ln n)$ (since $log_2 n^3 = 3log_2 n$).

11. $(m^2 + 2m - 3)(n + \ln^2 n + 14) = O(m^2 n)$.

12. $2m \ln^2 n + 3m^2 \ln n = O(m^2 \ln^2 n)$.

13. The largest n-digit number to the base m is $O(m^n)$.

14. The maximum number of circles or radius $\frac{1}{n}$ that fit into a circle of radius m without overlapping. The area of k circles of radius $\frac{1}{n}$ is $\frac{\pi k}{n^2}$. If this exceeds the area πm^2 of the big circle, some of the circles necessarily overlap. Therefore, we have the relationship $\frac{\pi k}{n^2} \leq \pi m^2$, which yields $k \leq m^2 n^2$. Thus this value is $O(m^2 n^2)$. In fact, as I argued in class, it is asymptotically equal to $\frac{\pi m^2 n^2}{2\sqrt{3}}$.

■

Problem 3. Exercises 1,2,3 from §2 of Chapter 2. Exercise 2 does not require proofs, but the others do.

Proof. (a) Suppose a k-bit integer a is divided by an l-bit integer b (where $l \leq k$) to get a quotient q and a remainder r:

$$a = qb + r, 0 \leq r < b.$$

Notice that the length of qb is the sum of the lengths of q and b. Moreover, the length of r is less or equal to that of qb. Since the length of a sum is the maximum of the length, we see that the length of $a = qb + r$ is just the sum of the length of bq. Thus q has $k - l$ bits. (Of course, it is always possible for this to be off a bit, depending on whether or not there was carrying, but not by more than that.

(b) (a) The sum of n numbers, each of length at most k. The maximum value this sum can have is $n2^k$. Upon taking the logarithm, we find this is $O(k + \ln n)$.

(b) The length of this is dominated by the n^4 term, which has length $O(\ln n)$.

(c) A polynomial in n of degree k: $a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0$. Again, the length is dominated by the largest term n^k, which has length $O(\ln n)$.

(d) The product of all prime numbers of k or fewer bits. Let π denote the prime counting function. Then by the Prime Number Theorem, $\pi(n) \asymp \frac{n}{\ln n}$. So, the number primes with exactly k-bits is approximately

$$\pi(2^{k+1}) - \pi(2^k) \asymp \frac{2^{k+1}}{(k+1)\ln 2} - \frac{2^k}{k \ln 2}.$$

The length of the product of all these is just k times of the number of them. Then the length of the product of all with k or fewer bits, is just the sum of the length of the product of all primes with exactly j bits as j goes from 1 to k. Thus, we find

$$\sum_{j=1}^{k} \frac{j 2^{j+1}}{(j+1)\ln 2} - \frac{2^j}{\ln 2} \asymp 2^{k+1},$$

so this is $O(2^k)$.

(e) We will use Stirling's approximation to say that

$$(n^2)! \approx \sqrt{2\pi} n (\frac{n^2}{e})^{n^2}.$$

Taking the logarithm of the left-hand side, we can ignore the factor of $\sqrt{2\pi}$, and we are left with

$$\ln n + 2n^2 lnn - n^2 = O(n^2 lnn).$$

(f) The n-th Fibonacci number F_n. By Binet's formula,

$$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}},$$

and hence F_n is the result of rounding $\frac{\varphi^n}{\sqrt{5}}$. The length of this latter number is $O(n)$.

(c) From the previous problem, the length of F_n is asymptotically equal to $\log_2 \frac{\varphi^n}{\sqrt{5}} = n \log_2 \varphi - \frac{1}{2} \log_2 5 \asymp n \log_2 \varphi$.

■

After having proven the Division Algorithm on the last homework, you now know that the proof of existence of the quotient and remainder is nonconstructive, instead relying on the well-ordering of a certain set of integers. In this homework, we will analyze an actual division method.

Problem 4. The *Newton–Raphson method* is a numerical method for computing the zeros of a real-valued differentiable function. The method proceeds as follows for the function f:

- Guess a value x_0 for the zero of f.
- for $n \geq 1$, set $x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}$.

This numerical scheme does not always converge, but it often works in applications, and in fact, convergence is often quadratic.

Now consider the following. To calculate the quotient and remainder from a division with dividend a and divisor b, it suffices to calculate only one or the other, because the other can be easily calculated once one is known. Moreover, the quotient is the floor of the $\frac{a}{b}$ (i.e., greatest integer less or equal to $\frac{a}{b}$). Therefore, if we can calculate $\frac{a}{b}$ sufficiently accurately, then we can compute the quotient (since it's the integer part). Moreover, $\frac{a}{b} = a \cdot \frac{1}{b}$, so if we can calculate $\frac{1}{b}$ with high precision, then we can do division by multiplication!

So, suppose we want to divide by b. We can approximate $\frac{1}{b}$ by applying the Newton–Raphson method to the function $f(x) := \frac{1}{x} - b$.

(a) Show that the iterations of the Newton–Rapshon method can be achieved solely with the addition and multiplication of known quantities.

(b) Define the error of the n-th approximant to be $e_n = bx_n - 1$. Prove that $e_{n+1} = -e_n^2$.

(c) Using the previous error calculation, what is the valid range (in terms of b) for your initial guess in order to ensure convergence? Explain how to easily choose an initial guess (expressed in binary).

(d) If your initial guess is good to one (binary) significant figure, how many iterations do you have to compute in order to ensure your guess is good to 500 binary significant figures (i.e., 500 bits of precision)?

Proof. (a) Since our function is $f(x) = \frac{1}{x} - b$, iterations of the Newton–Raphson method have the form:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{\frac{1}{x_n} - b}{-\frac{1}{x_n^2}} = x_n + (x_n - x_n^2 b) = x_n(2 - x_n b).$$

(b) Notice that

$$\begin{aligned} e_{n+1} &= bx_{n+1} - 1 \\ &= bx_n(2 - x_n b) - 1 \\ &= -b^2 x_n^2 + 2bx_n - 1 \\ &= -(bx_n - 1)^2 \\ &= -e_n^2. \end{aligned}$$

(c) In order for the error to be decreasing (hence convergence of x_n), we need $|e_0| < 1$. In particular, we need $|bx_0 - 1| < 1$, or equivalently, $0 < bx_0 < 2$, or equivalently, $0 < x_0 < \frac{2}{b}$. So, how do we choose an x_0 in this range? Consider the expression for b in binary, and suppose it has l-bits. Thus $2^{l-1} \leq b < 2^l$. Therefore, $2^{-l} < \frac{1}{b} \leq 2^{-l+1}$. So using $x_0 = \frac{2^{-l}+2^{-l+1}}{2} = 2^{-l-1} + 2^{-l}$ (which is trivial to express in binary) is a valid and explicit choice for the initial guess.

(d) So, I realized I did this incorrectly in class, because the error is *not* just the difference $x_n - \frac{1}{b}$, in fact, $x_n - \frac{1}{b} = \frac{e_n}{b}$. So, to get 500 bits of precision, making sure $e_n < 2^{-502}$ is sufficient. In particular, given our initial guess, we can verify that $-\frac{1}{2^2} \leq bx_0 < \frac{1}{2}$ and thus the error $|e_n| < 2^{-2^n}$. So choosing $n = 9$ iterations suffices for our purposes.

■