Problem Set for Unit 7

Prof. Amit Chakrabarti
Computer Science Dept
Dartmouth College

Here are some problems about the **RSA cryptosystem** and related topics. Have the modular arithmetic lecture notes handy as you work on this set.

By now you are familiar with what needs to be submitted towards graded homework and when.

The symbols $\mathbb{N}, \mathbb{N}^+, \mathbb{Z}, \mathbb{Z}_m, \mathbb{Z}_m^*, \mathbb{Q}, \mathbb{R}$, and $\phi(m)$ have their usual meanings.

PS7-1

Dr. Speedy proposes a cryptosystem that would work faster than RSA by working modulo a large *prime*.

- Bob chooses a public key of the form (P, e), where P is a very large (say 300-digit) prime.
- When Alice wants to send a message $M \in \mathbb{Z}_P$ to Bob, she will send $C := M^e \pmod{P}$.
- Bob has a secret key d such that $ed \equiv 1 \pmod{P-1}$; using it, he decrypts $M' := C^d \pmod{P}$.

a. Show that Dr. Speedy's cryptosystem is sane, in the sense that $M' = M$ always.

b. Why aren't we all using Dr. Speedy's cryptosystem instead of RSA, which is more complicated?

PS7-2

The security of RSA would be compromised if you could find an algorithm $\mathscr{A}$ to quickly compute $\phi(N)$, given N. We believe that factoring is hard, but why should computing ϕ be hard?

Prove that if computing ϕ were easy—i.e., algorithm $\mathscr{A}$ exists—then $\mathscr{A}$ can be used to quickly factor the RSA modulus N. You'll need to use the fact that N is the product of *exactly two* primes.

PS7-3 *HW*

Dr. Tricksy proposes a cryptosystem that works just like RSA, except that the modulus N is chosen as the product of *ten* distinct primes, not two. "Using ten primes makes it five times as secure as RSA," they say.

What's wrong with Dr. Tricksy's idea? [5 points]

PS7-4 *HW*

As you have seen, the basic operation of RSA encryption and decryption is *modular exponentiation*. In this problem, you will develop a fast algorithm for carrying out this operation. Suppose that we want to compute $a^k \bmod n$, where $a \in \mathbb{Z}_n$, and n and k are very large integers (say 1024 bits each, which is about 308 digits).

Let's say you have a function `modmult(a,b,n)` that returns $ab \bmod n$ (code shown below). Your goal is to write a function `modpow(a,k,n)` that returns $a^k \bmod n$. Here's a bad way to do it.

```
def modmult(a, b, n):
    """multiply a and b modulo n, assuming n > 0"""
    return (a * b) % n

def modpow_bad(a, k, n):
    """compute a**k modulo n, assuming k >= 0, n > 0"""
    result = 1
    for i in range(k):
       result = modmult(result, a, n)
    return result
```

a. What's bad about the `modpow_bad` function above?

b. In class (see the posted slides), we used a clever method to compute $a^{42} \bmod n$, based on the decomposition $42 = 32 + 8 + 2$. Explain how you would compute $a^{83} \bmod n$ along similar lines. Don't write code. Instead, write something analogous to what you see on the posted slides for $a^{42} \bmod n$.

c. Give a very short proof that $\forall\, x \in \mathbb{R}\ \forall\, n \in \mathbb{N}$,

$$x^n = \begin{cases} (x^2)^{\lfloor n/2 \rfloor}, & \text{if } n \text{ is even,} \\ x \cdot (x^2)^{\lfloor n/2 \rfloor}, & \text{if } n \text{ is odd.} \end{cases}$$

d. The above equation captures the general idea behind the tricks for quickly computing $a^{42} \bmod n$ and $a^{83} \bmod n$. Based on the equation, write a much more efficient Python function `modpow(a, k, n)`.

Warning: It should go without saying that the Honor Code forbids you from looking at modular exponentiation code online if you intend to submit this problem for credit.

e. Use your code to evaluate $5921400673^{6626043712} \bmod 9999988887$. [1+1+2+4+1 points]

PS7-5

In one of the cases of our in-class proof of the Decryption Theorem for the RSA cryptosystem (see the posted slides), we argued that a certain congruence was true modulo p and also true modulo q and said that *therefore* it was true modulo pq. This is a baby step towards a beautiful theorem that you'll now prove.

Suppose that $m, n \in \mathbb{Z}$ are such that $m \geq 2$, $n \geq 2$, and $\gcd(m, n) = 1$.

a. Generalize the argument to show that if $x \equiv y \pmod{m}$ and $x \equiv y \pmod{n}$, then $x \equiv y \pmod{mn}$.

b. Prove that the function $f : \mathbb{Z}_{mn} \to \mathbb{Z}_m \times \mathbb{Z}_n$ given by $f(x) = (x \bmod m,\ x \bmod n)$ is injective.

c. Conclude that f is therefore surjective.
Give a precise reason! Naturally, you need something in addition to the just-derived fact that f is injective.

d. Conclude that given any two values $a, b \in \mathbb{Z}$, the system of congruences

$$x \equiv a \pmod{m}$$
$$x \equiv b \pmod{n}$$

in the unknown x has one and only one solution modulo mn. This is called the Chinese Remainder Theorem. Read the Rosen textbook for the story behind this name; it dates back to ancient China.

PS7-6 ^{HW}

Suppose that $m, n \in \mathbb{Z}$ are such that $m \geq 2$, $n \geq 2$, and $\gcd(m, n) = 1$.

a. Prove that the function f defined in ***PS7-5*** is also a bijection from $\mathbb{Z}^*_{mn}$ to $\mathbb{Z}^*_m \times \mathbb{Z}^*_n$.
You may use the results of ***PS7-5*** without proof, provided you swear that you have solved ***PS7-5*** first!

b. Conclude that $\phi(mn) = \phi(m)\phi(n)$. [7+1 points]

PS7-7

Consider arithmetic modulo m, where $m \geq 3$. We say that $a \in \mathbb{Z}_m$ is a *square root of unity* modulo m if $a^2 \equiv 1 \pmod{m}$. For any $m \geq 3$, there are always two such square roots, namely, 1 and $m - 1$: these are called "trivial" square roots. If $a \not\equiv \pm 1 \pmod{m}$, we say that a is a *nontrivial* square root of unity.

a. Prove that if there exists a nontrivial square root of unity modulo m, then m is composite.
Hint: Take a look at ***PS5-9*** ^{HW}.

b. Now suppose that b is a nontrivial square root of unity modulo m. The existence of b implies that m is composite, but this by itself doesn't tells us how to *find* a nontrivial divisor of m. (A nontrivial divisor is a positive divisor besides 1 and m.) That's where the following result comes in.
Prove that either $\gcd(m, b - 1)$ or $\gcd(m, b + 1)$ is a nontrivial divisor of m.

PS7-8 ^{EC}

The security of RSA would be compromised if you could find an algorithm $\mathscr{B}$ to quickly compute the (secret) decryption key d, given the (public) encryption key (N, e). Again, we believe that factoring is hard, but why should this computation be hard? In this problem, you'll prove if algorithm $\mathscr{B}$ exists, then $\mathscr{B}$ can be used to quickly factor N. Unfortunately, the proof will rest on an advanced theorem that is beyond the scope of this course (talk to me if you want to learn more).

Suppose that $m \in \mathbb{N}^+$ is divisible by at least two different odd primes.

a. Prove that there exists a nontrivial square root of unity modulo m.
Hint: Use the Chinese Remainder Theorem.

b. Suppose $t \in \mathbb{N}^+$ is divisible by $\phi(m)$ and $t = 2^r s$, where $r \in \mathbb{N}$ and s is odd. For each $a \in \mathbb{Z}_m$, consider the sequence

$$b_0 = a^s \bmod m,\ b_1 = a^{2s} \bmod m,\ b_2 = a^{2^2 s} \bmod m,\ \ldots,\ b_r = a^{2^r s} \bmod m\,.$$

We'll say that a is *useful* if at least one of the following conditions holds:

- $b_r \neq 1$;
- $b_0 \neq 1$ and $\forall\, j \in \{0, 1, \ldots, r\} :\ b_j \neq m-1$.

Prove that if a is useful, then from the sequence of b_i values one can find a nontrivial divisor of m.
Hint: This has something to do with nontrivial square roots of unity.

c. Here is the advanced theorem you need now: "At least half of the elements of $\mathbb{Z}_m$ are useful."
Based on this theorem and your results above, show that if m is an RSA modulus (i.e., a product of two distinct odd primes), then algorithm $\mathscr{B}$ can be used to factor m quickly.