Introduction to Cryptography (462)
Homework 06
Alexander Kellermann Nieves
Due: Tuesday, November 21st, 2017 at 2pm Section 3

- Be sure to put your NAME and Section number on the first page.
- If you upload your submission to the myCourses dropbox, I will only accept .pdf format and only the last thing you submit will be accepted.
- This homework is related to Chapter 6 in the Paar and Pelzl (P&P) book.
- **For each question, show the details of your computation unless otherwise specified.**

1. Compute the inverse a^{-1} mod n with Fermat's Theorem (if applicable) or Euler's Theorem:

   (a) $a = 4, n = 7$
   (b) $a = 5, n = 12$
   (c) $a = 6, n = 13$

   1) Since N is prime, the $\phi(7)$ is equal to 6. Therefore:

$$4^6 = 1(\mod 7)$$
$$4^{-1} = 4^5(\mod 7) = 2$$

   2) We can use Euler's Theorem for this one.

$$gcd(5, 12) = 1$$
$$5^{12} = 1(\mod 12)$$
$$5^{-1} = 5^{11}(\mod 12) = 5$$

   3) Let's use Eurler's Theorem again.

$$gcd(6, 13) = 1$$
$$6^{13} = 1(\mod 13)$$
$$6^{-1} = 6^{12}(\mod 13) = 1$$

2. Verify that Euler's Theorem holds in $\mathbb{Z}_m$, $m = 6, 9$, for all elements a for which $gcd(a, m) = 1$. Also verify that the theorem does NOT hold for elements a for which $gcd(a, m) \neq 1$.

   If we let $z_1, z_2, ..., z_{\phi(m)}$ be elements of $\mathbb{Z}_m$. For any $y \in \mathbb{Z}_m$ we can define $y\mathbb{Z}_m$ to equal $yz_1, yz_2, ...yz_{phi(m)}$. Since y has a multiplicative inverse mod m, the mapping $z \rightarrow yz$ mod m is a bijection, and so $y\mathbb{Z}_m = \mathbb{Z}_m \mod m$. [1]

   We can then understand that it follows that

$$\Pi_i z_i = \Pi_i y z_i = y^{\phi(m)} \Pi_i z_i \mod m$$

   If we multiply both sides by

$$(\Pi_i z_i)^{-1}$$

   we get

$$1 = y^{\phi(m)} (\mod m)$$

3. For the affine cipher in Chapter 1, the multiplicative inverse of an element mod 26 can be found as:

$$a^{-1} \equiv a^{11} \bmod 26.$$

   Derive this relation by using Euler's Theorem.

   Answer:

   We know that Euler's theorem is $a^{\varphi(n)} = 1(\mod n)$
   If we consider n = 26 we can find φ by doing the following.

$$\phi(26) = \phi(2)\phi(13)$$
$$1 * 12$$
$$= 12$$

   Now we know that $\phi(26) = 12$
   Let's substitute n = 26 back into Euler's Theorem to get

$$a^{\varphi(26)} = 1(\mod n)$$
$$a^{12} = 1 \mod 26$$

   If we multiply both sides by the inverse we'll get

$$a^{12} \cdot a^{-1} = a^{-1} \mod 26$$

   And simplifying we get

$$a^{11} = a^{-1} \mod 26$$

4. Compute 39^{39} mod 773, using the Square-and-Multiply algorithm. Show the details of your computation.

   First, we have to convert 39 into binary. Which yields

$$(39)10 = 0b100111$$

   Now we apply the Square and multiply method, but at each step we modulo 733 to make the math simpler.

   The first digit is 1, which just yields the number 39.
   The second number becomes a 39.
   On step 3 we get have the digit 0, so we square that, which yields 1521.
   1521 mod 773 = 748
   Step 4 we have the digit 1, So that's 748 squared, multiplied by 39. Mod that by 773 and you get 412.

   Step 5:

$$412^2 = 169,744 \cdot 39 = 6,620,016 \mod 773 = 44$$

   Step 6:

$$44^2 = 1936 \cdot 39 = 75,504 \mod 773 = 523$$

5. Use your favorite programming language to implement the pseudo-code on page 182 of the P&P book for the Square-and-Multiply algorithm. Submit your code in the PDF file as usual.

```
use std::io;

// Author: Alexander Kellermann Nieves
// Email: akn1736@g.rit.edu

fn main(){
    // Create a mutable data type, since we're going to change the
    // value stored in the string
    let mut base = String::new();

    // print line is a macro in rust
    println!("Input base: ");

    // the least readable way to grab a string from stdin. Note that
    // we have to specify that the function needs to borrow the value
    // of base, and not gain ownership of the variable. This is
    // because we need it later on.
    io::stdin().read_line(&mut base)
        .expect("Couldn't read line");

    let mut power = String::new();
    println!("Input power: ");

    io::stdin().read_line(&mut power)
        .expect("Couldn't read line");

    // .trim() is used to get rid of trailing and leading
    // whitespace. For whatever reason, it won't allow the string to
    // be cast as an integer without it.
    let base_num: i64 = base.trim().parse().unwrap();
    let power_num: i64 = power.trim().parse().unwrap();

    // Finally, we pass all that stuff to our lovely function.
    println!("{}", exp_by_square(base_num, power_num));
}

fn exp_by_square(x: i64, n: i64) -> i64{
    // We take a base x, and an exponent n, and run the square and
    // multiply method on it. This is a recursive function.
    if n < 0 {
        return exp_by_square(1 / x, -n);
    }
    else if n == 0 {
        return n;
    }
    else if n == 1 {
        return x;
    }
    else if n % 2 == 0 {
        return exp_by_square( x * x, n / 2);
    }
    else if n % 2 == 1 {
        return x * exp_by_square( x * x, (n - 1) / 2);
    }
    // We needed an else statement to make the compiler happy,
    // it should be literally impossible to get to that point
    else {
        return 0;
    }
}
```

[1] http://www.cs.yale.edu/homes/aspnes/pinewiki/NumberTheory.html