COMP2700 Lab 11 Solutions

Exercise 1.

If every pair out of n = 120 employees requires a distinct key, we need in sum

$$n \cdot \frac{(n-1)}{2} = 120 \cdot \frac{120-1}{2} = 7140$$

key pairs. Note that each of these key pairs need to be exchanged in a secure way.

Exercise 2.

One way to solve this is to calculate the gcd directly:

$\gcd(n, 12) = 1$ for $n = 1,5,7,11$ so $\Phi(12) = 4$

$\gcd(n, 15) = 1$ for $n = 1,2,4,7,8,11,13,14$ so $\Phi(15) = 8$

Alternatively, we can find the canonical factorisation of m and use the following formula

$$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1})$$

where

$$m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$$

For m=12, we have $m = 2^2 \times 3$, so $\Phi(m) = (2^2 - 2^1) \times (3^1 - 3^0) = 2 \times (3-1) = 4$.

For m=15, we have $m = 3 \times 5$, so $\Phi(m) = (3-1) \times (5-1) = 8$.

Exercise 3.

a) 7 is prime, so use Fermat's Little Theorem. $a^{-1} \equiv a^{n-2} mod\ n$
$4^{-1} \equiv 4^{7-2} \equiv 4^5 \equiv 2\ mod\ 7$.

b) Since 12 is a composite, Fermat's theorem is not applicable. However, $\gcd(5,12) = 1$ so Euler's theorem is applicable. Recall that $\Phi(12) = 4$, so applying Euler's theorem:
$$a^{-1} \equiv a^{\Phi(n)-1} mod\ n$$
$$\Rightarrow 5^{-1} \equiv 5^{\Phi(12)-1} \equiv 5^{4-1} \equiv 5^3 \equiv 5\ mod\ 12.$$

c) 13 is prime, so use Fermat's theorem: $6^{-1} \equiv 6^{13-2} \equiv 6^{11} \equiv 11\ mod\ 13$.

Exercise 4.

First calculate $\Phi(26)$, which is 12. If $\gcd(a, 26) = 1$ (which is a requirement for the affine cipher), then using Euler's theorem, we have $a^{\Phi(26)} \equiv a^{12} \equiv 1 \; mod \; 26$. This implies $a \cdot a^{11} \equiv 1 \; mod \; 26$, so a^{11} is the inverse of a modulo 26.

Exercise 5.

To find the order of an element a in a multiplicative group Z_p^*, compute all $a^i \; mod \; 7$ for all $1 \leq i \leq p-1$ and count the number of distinct values produced. For example, if $a = 2$, in Z_7^* we have:

$2^1 \equiv 2 \; mod \; 7$, $2^2 \equiv 4 \; mod \; 7$, $2^3 \equiv 1 \; mod \; 7$, $2^4 \; mod \; 7 \equiv 2$, $2^5 \equiv 4 \; mod \; 7$,

$2^6 \equiv 1 \; mod \; 7$.

We see that there are only three distinct values from the exponentiations mod 7, i.e., 1, 2 and 4, so the order of 2 is 3 in this case.

The order of each element is given in the following table:

a	1	2	3	4	5	6
$ord(a)$	1	3	6	3	6	2

A primitive element is an element of the group whose order is the same as the cardinality of the group. In this case, the cardinality of Z_7^* is 6, so the primitive elements are 3 and 5.

Exercise 6.

Only $e_2 = 49$ is valid exponent. This is because we require the public exponent to satisfy $\gcd(e, \Phi(n)) = 1$ where $n = p \times q$ and $\Phi(n) = (p-1) \times (q-1)$. In this case, $\Phi(n) = 40 \times 16 = 640$. Among the two parameters e_1 and e_2, only e_2 satisfies $\gcd(e_2, \Phi(n)) = 1$.

Exercise 7.

a) In this case we have $n = p \times q = 33$ and $\Phi(n) = 2 \times 10 = 20$. Note that $\gcd(7,20) = 1$ so $e = d^{-1} \; mod \; \Phi(n)$ exists, i.e., $e = 3$. To encrypt x, compute $x^e mod \; n = 5^3 mod \; 33 = 26$.

b) $n = 5 \times 11 = 55$, $\Phi(n) = 4 \times 10 = 40$. The private exponent is $d = e^{-1} mod \; \Phi(n) = 3^{-1} mod \; 40 = 27$. The decryption of y is computed as $y^d \; mod \; n = 9^{27} \; mod \; 55 = 4$.

Note that to reduce the size of intermediate results in calculating $9^{27} \; mod \; 55$, we can break it down to computing $9^{13} \; mod \; 55$ first, and then use the equality $9^{27} = (9^{13})^2 \cdot 9 \; mod \; 55$. We will look at this kind of reduction again in Lab 12.

Exercise 8.

Let $y_1 = x_1^e \ mod \ n$ and $y_2 = x_2^e \ mod \ n$. Then $y_1 y_2 = x_1^e x_2^e \ mod \ n \equiv (x_1 x_2)^e \ mod \ n$. That is, the multiplication of the two ciphertexts coincides with the encryption of the multiplication of the underlying plaintexts.

Exercise 9.

a.

```
>>> from Crypto.PublicKey import RSA
>>> key = RSA.generate(1024)
>>> pt = 10
>>> # encrypt using pow(pt, key.e, key.n)
>>> ct = pow(pt, key.e, key.n)
>>> # decrypt using pow(ct, key.d, key.n)
>>> pow(ct, key.d, key.n) == pt
True
>>>
```

b.

```
>>> a = 10
>>> b = 20
>>> x = pow(a, key.e, key.n)
>>> y = pow(b, key.e, key.n)
>>> z = (x * y) % key.n
>>> a*b == pow(z, key.d, key.n)
True
```