In this laboratory exercise you will build a simple public key encryption program based on the RSA algorithm. Once implemented you will build a cryptanalysis tool to "hack" your public key and decode the encrypted message.

Introduction - In 1976, Rivest, Shamir and Adleman introduced a public key encryption system now known as RSA. It is important to understand the difference between public (asymmetric) key and private (symmetric) key encryption systems. In the case of the private key system, the sender and receiver must share the encryption/decryption key along a private (secure) communication line, because anyone in possession of the key would be able to decrypt the message. In the case of public key encryption, all communications between the sender and receiver must be assumed to be open to public view, including a description of the encryption/decryption algorithm itself. Private key cryptosystems are easier to implement and can be made more secure than public key cryptosystems. The value of a public key system is for those situations in which the passing of a private key between sender and receiver is not convenient or practical.

The first step in getting a private message from the sender to the receiver involves the intended receiver creating a public/private key pair. The receiver passes the public key to the sender who uses this key to encrypt the message. The encrypted message is sent to the receiver who, in turn, uses the corresponding private key to decrypt the message. In order for this system to be of practical value two conditions must be true:

1. the public and private keys must be different (asymmetric)
2. the generation of this key pair must be much simpler than the derivation of the private key given the public key.

Mathematical Background - The public/private key pair (e,d) are used to encrypt and decrypt messages using modular exponentiation. Given a plaintext message M which must be less than the magnitude of the modulus n we generate the encrypted message C,

\[ C = M^e \mod n \]

and C is converted back into M by,

\[ M = C^d \mod n \]

It is important to note that the sender must have both the public key value e and the modulus n in order to encrypt the message. It is not practical to assume that the value of n can be kept private because this would mean that the sender and receiver had managed to communicate privately, negating the purpose of using public key encryption.

Generation of the Key Pair (e,d) - We start by choosing to values p and q that are relatively prime and of similar magnitude. This usually means we choose two prime numbers. For a secure public key encryption these would need to be very large numbers, but for demonstration purposes we can use smaller values. The product \( pq = n \), the
modulus. Also from p and q we compute a related value \( \phi = (p-1)(q-1) \). The value \( \phi \) is used to derive a value d (private key) from e (public key), in the following way.

Choose \( e \) such that \( 1 < e < \phi \) and \( e \) and \( \phi \) have no common factors. In other words the greatest common divisor between \( e \) and \( \phi \) is 1.

\[
gcd(e, \phi) = 1 \quad (1)
\]

In order to derive the private key we note that,

\[
e \cdot d \equiv 1 \mod \phi \quad (2)
\]

which means that \( e \cdot d - 1 \) is evenly divisible by \( (p-1)(q-1) \).

**Converting to Code** - We will build a console application for our PKE demo program. Create a new Project named PKE. We will implement the various algorithms as methods within this class and call them from the main method.

We will use long (64 bit integer) data types for our variables. In the main method ask the user to enter values for p and q, such as,

```csharp
Console.Write("Enter p... ");
p = Convert.ToInt64(Console.ReadLine());
Console.Write("Enter q... ");
q = Convert.ToInt64(Console.ReadLine());
```

Now, compute \( n \) and phi (\( \phi \)), also in the main method,

\[
n = p \times q;
\]

\[
\phi = (p - 1) \times (q - 1);
\]

Next, we ask the user to enter a value for \( e \)

```csharp
Console.Write("Enter e... ");
e = Convert.ToInt64(Console.ReadLine());
```

(this should be checked to ensure that \( gcd(e,\phi)=1 \). So we will need to implement a method named gcd() to return the greatest common divisor, for this test.

```csharp
private static long gcd(long a, long b)
{
    long x,y,r;
x=a;
y=b;
while(y!=0)
{
    r = x % y;
    x = y;
    y = r;
}
return x;
}
```

This method is based on the Euclidean algorithm. Now we are ready to derive a private key \( d \) that matches our public key \( e \). There are a number of efficient methods based on
the extended Euclidean algorithm or the least common multiple algorithm but we will use a much simpler method based on a direct application of equation (2).

```java
private static long get_d(long e, long phi) {
    long d = 1;
    long rem = -1;
    long test;
    while (rem != 0) {
        test = e * d;
        test = test - 1;
        rem = test % phi;
        if (rem == 0)
            return d;
        d += 1;
    }
    return 0;
}
```

This method returns 0 if there is an error otherwise it returns a value for \( d \). As a test of your program you may use the following values,

\[
p = 3593 \quad \text{and} \quad q = 5827
\]

Your program should compute the following values:

\[
n = 20936411 \\
\phi = 20926992
\]

two sample (e,d) pairs are,

\[
e = 4201 \quad d = 4807081 \\
e = 10079 \quad d = 4279247
\]

At this point we are ready to encode our message. Download Crypto_01, unzip and run it using the values computed above for \( p, q, e, \) and \( d \). Note that your plaintext message will be converted to an uppercase string with X's in place of blanks. This is done to simplify the conversion of the plaintext string to and from a numeric value.

Now download and run Crypto_02. Run it and enter the specified values to determine the difficulty in deriving the private key from the public key and modulus.
Project 11 - Public Key Encryption

Name _____________________________________________________ Score ________

Complete the tasks of this project, answer the questions below, remove and attach this section of the project form to a copy of your source code. Please DO NOT include a copy of any automatically generated (Designer Code).

1. Using the terms message sender and message receiver, answer the following:

a. Who generates the public key and private key pair? ___________________________

b. Who converts M to C? ________________________

c. Who convert C to M? ________________________

2. What value(s) is(are) sent to the sender? __________________________

3. What value(s) is(are) needed to decrypt the message? _________________________

4. Why is it more difficult for a third party eavesdropper to decrypt the message than it is for the receiver?

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

5. Briefly describe what is needed in order to decrypt an encrypted message without knowing the private key?

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

6. What is the runtime of the encoding and decoding algorithm as a function of the message length and sizes of e and d?

________________________________________________________________________

7. Estimate the runtime of an algorithm used to find the value of d through a brute-force search.

________________________________________________________________________