

# Diffie-Hellman and Encryption Project

## Purpose

In this project, you will gain experience working with existing libraries to perform encryption using a 256-bit key. To generate the key, you will implement a part of the Diffie-Hellman algorithm. In doing so, you will also learn how to work with very large numbers that are too large to store in a standard data type such as integer or long.

## Objectives

Students will be able to:

- Implement the Diffie-Hellman Key Exchange.
- Employ existing algorithms to encrypt and decrypt data.
- Apply number theory concepts to solve real-world problems.

## Technology Requirements

Technology requirements will be found on the class GitHub README (Click or copy the link below)  
<https://github.com/GiveThanksAlways/CSE-539-Applied-Cryptography-2021-Fall-B>

## Directions

The first part of this assignment involves the creation of a 256-bit key for performing encryption. You will be implementing the Diffie-Hellman key exchange protocol. Typically, this protocol involves two parties concurrently sharing values and generating a key. In this assignment, you will be given all necessary values immediately and will not be required to send values over any channel. In this way, you can perform calculations as a single party.

Here is a brief reminder of how Diffie-Hellman works:

- 1) Alice and Bob agree on values for  $g$  and  $N$ .
- 2) Alice randomly picks  $x$  and Bob randomly picks  $y$ .
- 3) Alice computes  $g^x \bmod N$  and sends it to Bob (call this  $gx$ ).
- 4) Bob computes  $g^y \bmod N$  and sends it to Alice (call this  $gy$ ).
- 5) Alice computes  $(gy)^x \bmod N$ , and Bob computes  $(gx)^y \bmod N$ . These two values are equivalent and are the key.

The encryption algorithm you will be using in this project is AES (i.e., Rijndael encryption). You can use the AES class in the C# System.Security.Cryptography namespace. You will be using the 256-bit key mode. In order to perform the encryption, you will also need an IV. In this exercise, you will employ a 128-bit IV passed via command line in hex. Here is the list of values you will receive from the command line arguments, in order:

- 1) 128-bit IV in hex
- 2)  $g_e$  in base 10
- 3)  $g_c$  in base 10
- 4)  $N_e$  in base 10
- 5)  $N_c$  in base 10
- 6)  $x$  in base 10
- 7)  $g^y \bmod N$  in base 10
- 8) An encrypted message  $C$  in hex
- 9) A plaintext message  $P$  as a string

Hint: You may not need all of these values to perform the computations.

Note: To facilitate debugging, the values for  $g$  and  $N$  are given as a pair of values, the exponent ( $e$ ) and the constant ( $c$ ). You can calculate the value for either using the formula  $2^e - c$ . For example, if you were given  $g_e = 250$ ,  $g_c = 207$ ,  $N_e = 256$ , and  $N_c = 189$ , your values for  $g$  and  $N$  should be as follows:

g =  
1809251394333065553493296640760748560207343510400633813116524750123642  
650417

N =  
1157920892373161954235709850086879078532699846656405640394575840079131  
29639747

To perform these calculations without encountering an overflow, you **cannot** use int (32-bit), long (64-bit), or decimal (96-bit). Instead, you must make use of C#'s BigInteger struct, which supports any arbitrarily long integer.

After calculating the key, your program must perform a decryption of the given ciphertext bytes and an encryption of the given plaintext string. Your program should output these values as a comma separated pair (the decrypted text followed by the encrypted bytes). Here is a full example:

```
Command: dotnet run "A2 2D 93 61 7F DC 0D 8E C6 3E A7 74 51 1B 24 B2" 251 465  
255 1311 2101864342  
8995936589171851885163650660432521853327227178155593274584417851704581  
358902 "F2 2C 95 FC 6B 98 BE 40 AE AD 9C 07 20 3B B3 9F F8 2F 6D 2D 69 D6 5D  
40 0A 75 45 80 45 F2 DE C8 6E C0 FF 33 A4 97 8A AF 4A CD 6E 50 86 AA 3E DF"  
AfYw7Z6RzU9ZaGUloPhH3QpfA1AXWxnCGAXAwk3f6MoTx
```

```
Expected output: uUNX8P03U3J91XsjCqOJ0LVqt4l4B2ZqEBfX1gCGBH4hH,3D E9 B7  
31 42 D7 54 D8 96 12 C9 97 01 12 78 F7 A2 4F 69 1A FF F4 42 99 13 A1 BD 73 52 E5  
48 63 33 7A 39 BF C5 25 AD 53 26 53 0D E4 81 51 D1 3E
```

## Submission Directions for Project Deliverables

Upload your Program.cs file to Gradescope.