Ira A. Fulton Schools of Engineering
Arizona State University

CSE 539: Applied Cryptography

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

Learners 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

This project requires the knowledge of the following tools and technologies:

- .NET Core 8.0
- Basic Understanding of C#
- Zybooks IDE for editing your code

Directions

Accessing ZyLabs

You will complete and submit your work through zyBooks's zyLabs. Follow the directions to correctly access the provided workspace:

1. Go to the Canvas submission space, "**Submission: Diffie Hellman and Encryption Project**"

2. Click the “**Load Submission…in new window**” button.
3. When ready, review the provided code and develop your work where instructed.

Project Description

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 mod N and sends it to Bob (call this gx).
4. Bob computes g^y mod N and sends it to Alice (call this gy).
5. Alice computes $(gy)^x$ mod N, and Bob computes $(gx)^y$ mod N. These two values are equivalent and are the key.

The encryption algorithm you will use 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. To perform the encryption, you will also need an IV. In this exercise, you will employ a 128-bit IV passed via the 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 mod 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 = 11579208923731619542357098500868790785326998466564056403945758400791312 9639747

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 args0 args1 args2 args3 args4 args5 args6 args7 args8

Where,

args0= "A2 2D 93 61 7F DC 0D 8E C6 3E A7 74 51 1B 24 B2"

args1= 251

args2= 465

args3= 255

args4= 1311

args5= 2101864342

args6= 89959365891718518851636506604325218533272271781555932745844178517045813589 02

args7= "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"

args8= AfYw7Z6RzU9ZaGUloPhH3QpfA1AXWxnCGAXAwk3f6MoTx

So that final Command looks like

```
dotnet run "A2 2D 93 61 7F DC 0D 8E C6 3E A7 74 51 1B 24 B2" 251 465 255 1311 2101864342 899593658917

185188516365066043252185332722717815559327458441785170458135890 2 "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
```

Note: You may encounter extra line breaks while copying the args or command so make sure to remove that while pasting to the command.

Expected output: uUNX8P03U3J91XsjCqOJ0LVqt4I4B2ZqEBfX1gCGBH4hH,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

You are given an unlimited attempts to submit your best work. The number of attempts is given to anticipate any submission errors you may have in regards to properly submitting your best work within the deadline (e.g., accidentally submitting the wrong paper). It is not meant for you to receive multiple rounds of feedback and then one (1) final submission. Only your most recent submission will be assessed.

You must submit your Diffie Hellman and Encryption Project deliverable through zyBooks's zyLabs. Carefully review the submission directions outlined in this overview document to earn credit for your work correctly. Learners may not email or use other means to submit any assignment or project for review, including feedback, and grading.

1. Please use the **program.cs** file provided in your zyBooks workspace to complete your 'Diffie Hellman and Encryption Project' program, ensuring that you do not change the class name.
    a. Execute your code by running the below command in the terminal located at the bottom of the IDE.
        i. dotnet run "A2 2D 93 61 7F DC 0D 8E C6 3E A7 74 51 1B 24 B2" 251 465 255 1311 2101864342 8995936589171851885163650660432521853327227178155593274584417851704581358902 "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" AfYw7Z6RzU9ZaGUIoPhH3QpfA1AXWxnCGAXAwk3f6MoTx
2. When you are ready to submit your completed work, click on "**Submit for grading**" located on the bottom left.
3. You will know you have completed the project when feedback appears below the workspace.
4. If needed: to resubmit the project in zyLabs
    a. Edit your work in the provided workspace.
    b. Click "**Submit for grading**" again at the bottom of the screen.

The Diffie-Hellman and Encryption Project includes one (1) deliverable:

- **Project Code:** Use the zyBooks's zyLabs workspace to complete and submit your program.cs file.

Your submission will be reviewed by the course team and then, after the due date has passed, your score will be populated from zyBooks into your course grade.

Evaluation

The auto-grader will execute your code by passing values similar to the example through the command line. The grader will compare the plain text and the cipher text produced by your code with the expected plain and cipher text. If these match, you will receive **100 points**. Otherwise, you will receive **0 points** with corresponding error messages.