

Using Diffie-Hellman, Ciphers, and Hashes

This assignment uses three of the tools we have learned about in class over the past few weeks - DES symmetric encryption, cryptographic hashes, and Diffie-Hellman key establishment. You will use the .NET implementations of the algorithms to build a secure file transfer application.

1 Introduction: Sending Files Securely

A common issue when sending files over email is security. Say you want to email a sensitive file to somebody else, but you don't have any previously shared secret or code which will enable you to send it securely. The tool you will develop for this assignment addresses this issue, allowing you to securely encrypt and send a file to the other person without having any previously secure communication mechanism.

The fundamental steps in establishing a shared DES key and Initialization Vector (IV) is shown in Figure 1.

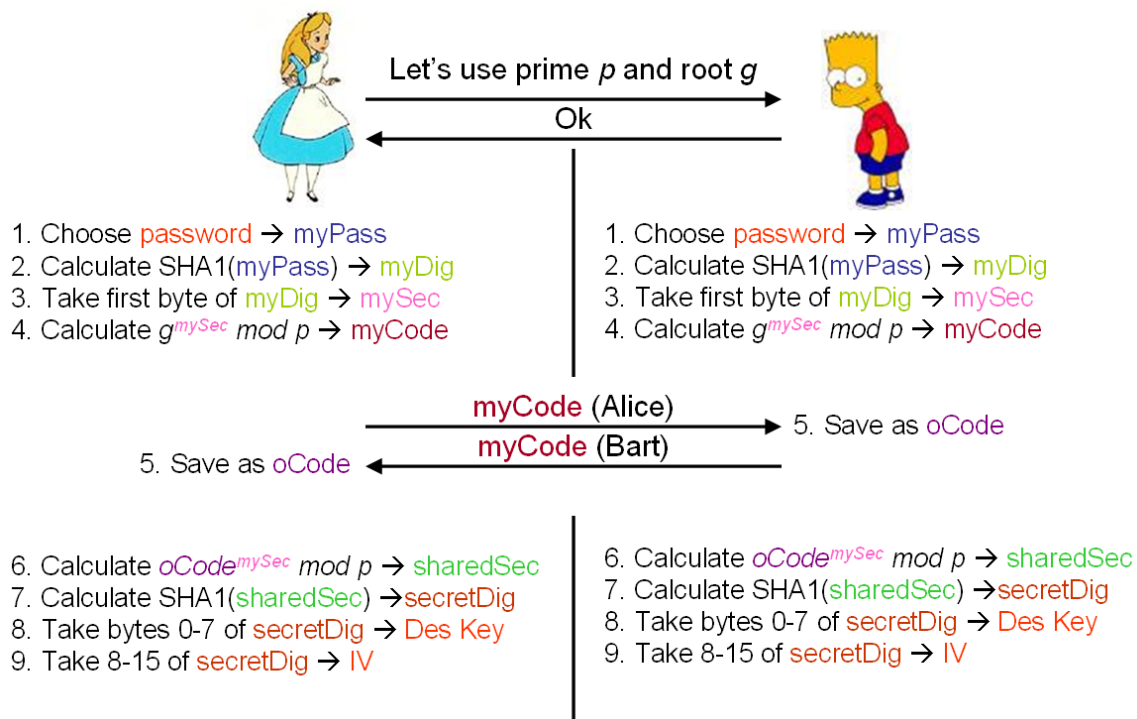


Figure 1: Fundamental steps in establishing a DES key and IV via Diffie-Hellman key exchange

Let's now explain the steps involved in establishing the shared DES key and IV as shown in the figure. The process is symmetric, that is both sides perform the same steps and in the end arrive upon the same shared secret key and IV.

Alice and Bart begin by agreeing upon a (non-secret) prime number p and a primitive root g

- Ideally g should be a primitive root of p , but the algorithm will work even if it is not
- The communication establishing p and g does not need to be secure and so can be sent via unsecured email

Once they have agreed on p and g they can start the steps to calculate the secret DES key and IV.

1. Alice chooses a secret text password called *myPass*.
2. Alice calculates the SHA1 digest of myPass (hashing myPass using SHA1) and saves it as *myDig*.
3. Alice takes the first byte of myDig and saves it as *mySec*.

Note: Using just the first byte reduces the number of possible keys to 256, significantly weakening the protocol. I want you to do this to make it easier for you to debug your application and results.

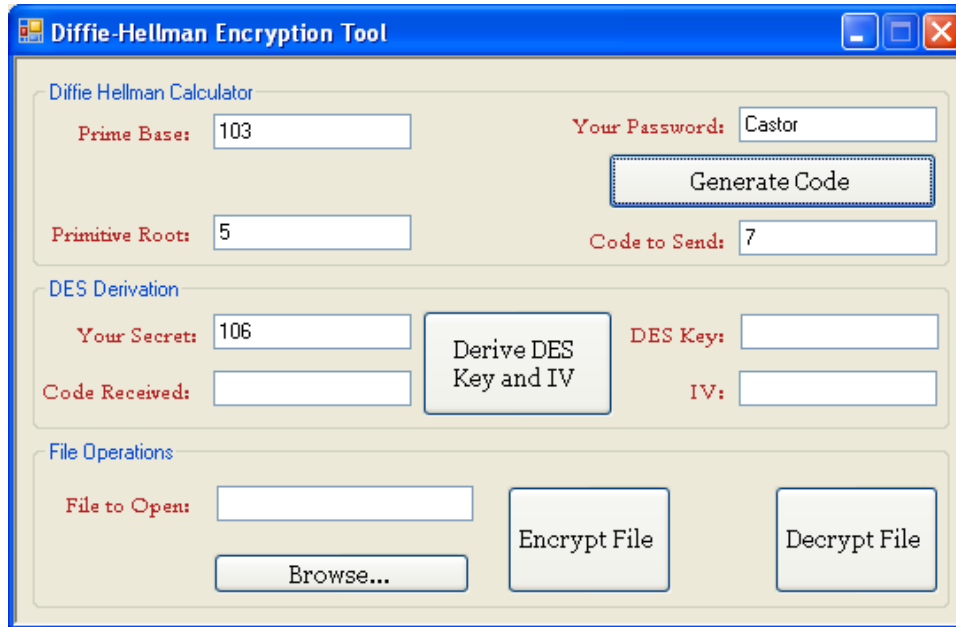
4. Alice calculates $g^{mySec} \bmod p$ and saves it as *myCode*. Alice sends myCode to Bart.
5. Bart has done the same steps (1-4), so Alice receives a code from Bart too. Alice calls the code she receives from Bart *oCode*.
6. Alice calculates $oCode^{mySec} \bmod p$ and saves it as *sharedSec*. This is the shared secret between Alice and Bart.
7. Alice calculates the SHA1 digest of sharedSec (hashing sharedSec) and saves it as *secretDig*.
8. Alice takes bytes 0 – 7 of secretDig and saves them as the DES Key.
9. Alice takes bytes 8 – 15 of secretDig and saves them as the IV.

Bart carries out the exact same steps as Alice and thereby arrives at the same DES key and IV as Alice. They can then use the DES key and IV to encrypt and decrypt files.

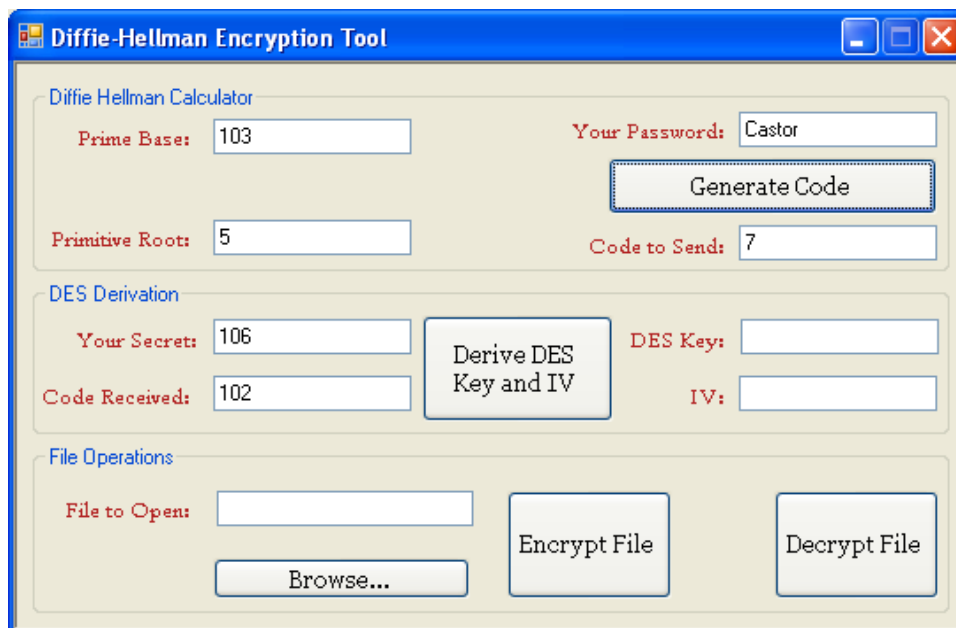
2 What to do

Your job is to develop an application which will enable Alice and Bart to perform the above steps. Let's consider a sample run of the algorithm where $p = 103, g = 5$, Alice chooses her password "Castor" and Bart chooses his password as "Pollux".

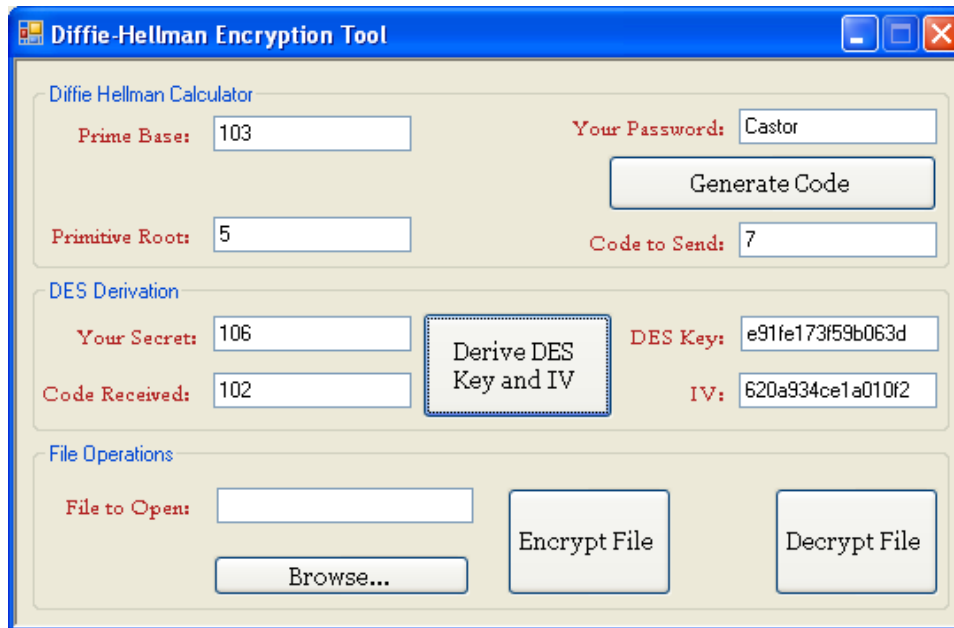
1. Alice enters in the prime 103, primitive root 5, and her password “Castor”. She clicks on the “Generate Code” button to generate two things:
 - (a) Her secret *mySec*. It is derived from taking the Unicode encoding of her password, hashing it, and taking the first byte. That value is shown in the text box labeled “Your Secret”. As shown in the window, her secret value is 106.
 - (b) The code she should send to Bart. That is calculated by $5^{106} \bmod 103 = 7$. It is shown in the text box labeled “Code to Send.”



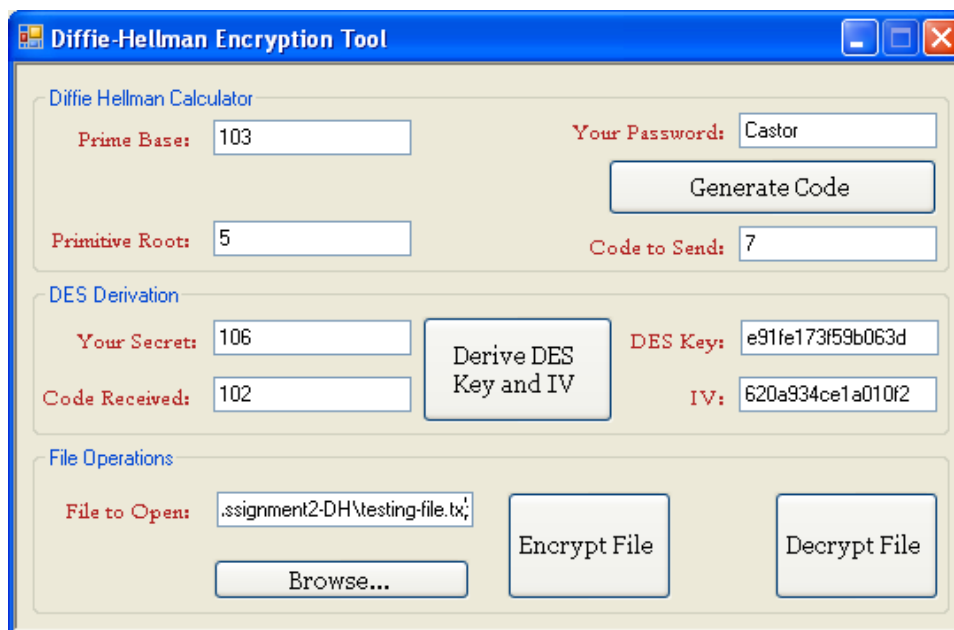
2. Alice receives the value 102 from Bart (the value Bart will receive by entering his password “Pollux” into his copy of the tool). She enters the value in the text box labeled “Code Received”.



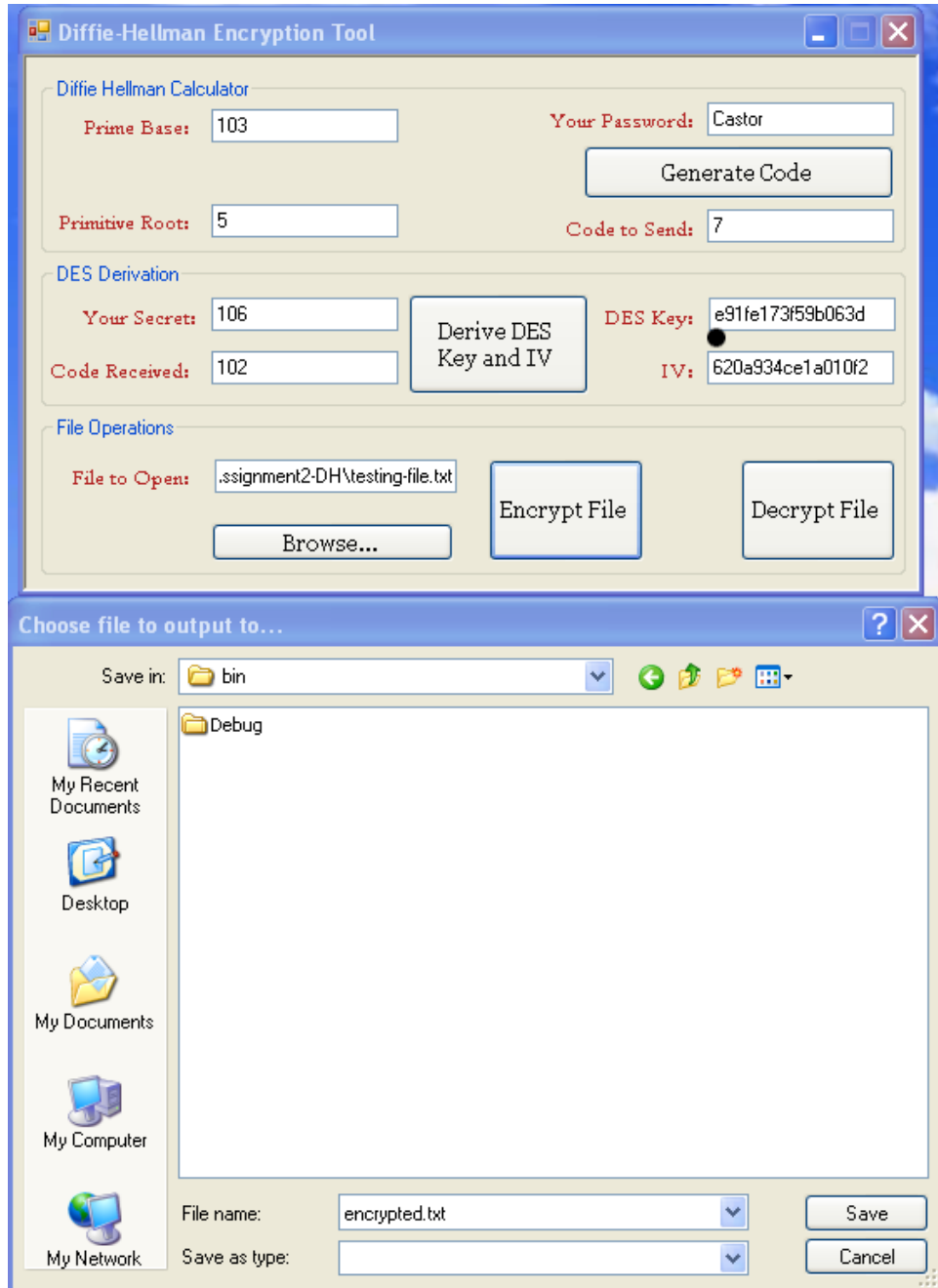
3. Alice clicks on “Derive DES Key and IV” to calculate the shared secret ($102^{106} \bmod 103 = 1$). The tool then computes the SHA1 digest of 1 (as a Unicode string). The first eight bytes of the result are shown in (hexadecimal notation) in the text box labeled “DES Key” (it’s e91fe173f59b063d). The second eight bytes of the result are shown in the text box labeled “IV” (it’s 620a934ce1a010f2).



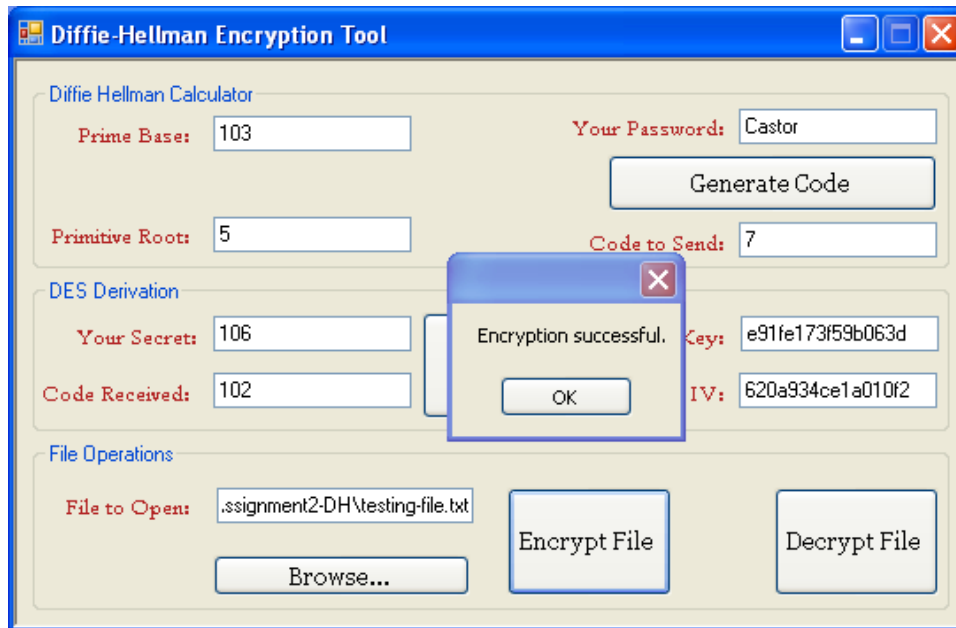
4. Alice then chooses the file to encrypt using the tool by clicking on the “Browse” button and selecting an existing file. She chooses `testing-file.txt` to encrypt.



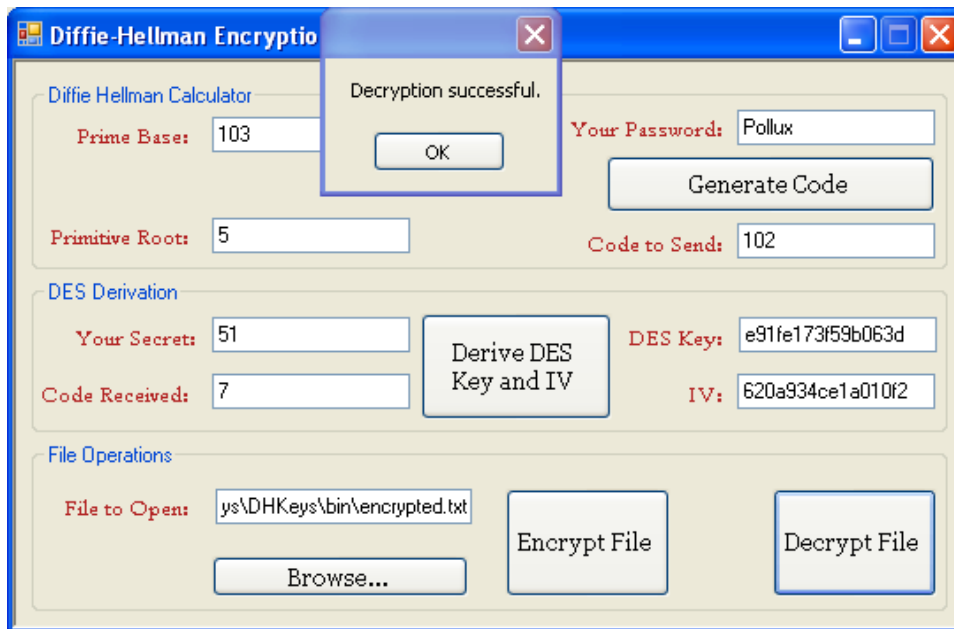
5. Alice clicks on the “Encrypt File” button and selects a file name to store the encrypted file.



6. After clicking on “OK”, the tool encrypts the file and shows a “success” screen.



Alice can then send the file to Bart securely. When Bart receives the file he can use the “Decrypt File” button to decrypt it using the same steps that Alice used:



3 Testing Your Application

Your tool should be able to encrypt and decrypt both text and binary files. Managing binary files will require you to use the `BinaryWriter` and `BinaryReader` stream management classes in C#. You can read more about them in the online documentation.

To help you test your application, there are two encrypted files on the course web page. One is a text file and one is a PDF (binary) file. They have been encrypted with the same keys that Alice and Bart used in the above example ($p = 103$, $g = 5$, Alice's secret "Castor", Bob's secret "Pollux"). You may use them to test your decryption routines.

Be sure to test your application with other example primes and primitive roots. A helpful web site with a Java application to derive primitive roots of prime number is found at: <http://www.math.mtu.edu/mathlab/COURSES/holt/dnt/quadratic4.html>.

4 A Note on Modular Exponentiation

You will find that the standard data types in C# are not sufficient to calculate large modular exponents the simple way (calculating g^a and then calculating the modulus with p). You will end up with an overflow in many cases. A more efficient, bit oriented algorithm exists (you can read about it online), but since we are dealing with relatively small exponents (16 bits long) we can calculate modular exponents in a reasonable amount of time recursively. A prepared function in C# to calculate recursive modular exponentiation is given on the course web page. You may copy it into your application code and use it as is.

5 What to turn in by 10 April 2011 at 8:30am

Each group should turn in the following items (one submission per group). Electronic submissions should be uploaded to the Telem website, transferred to me by USB disk on key, or sent via email to the course email ([ise328@gmail](mailto:ise328@gmail.com)) in a single ZIP or RAR file.

1. A project report file called README.txt containing the names of the students in your group and a one- or two-sentence description of each group members contributions to the project and an estimate of the number of hours each member contributed. The file must also contain any special instructions needed to run your program. **Skipping this will result in a 5 point penalty.**
 - The group will receive a **single grade** for the project. The information about individual contributions will be used to help gauge the *difficulty and effectiveness* of the assignment.
2. An electronic submission of your source code. This should be single, compressed directory in ZIP or RAR format. It also should contain a **compiled, executable version of the program**. The classes should be implemented according to the interface described above **using C#**.
 - Note: Due to gmail's security requirements, you can't send EXE files inside of ZIP archives. To submit an EXE, either use Telem, compress the directory with RAR, or submit the EXE in person by USB disk on key.

Collaboration between groups and copying from internet, any online resource, source code sites, or other resources **is forbidden**. Automated code checking tools may be used to check for copying or sharing of source code.

5.1 Grading Guide

I will grade your applications by testing it with encryptions, keys, and files of my choosing. I will grade based on the following criteria:

- Diffie-Hellman Calculations: 30%
- DES Key and IV Derivation: 30%
- File Encryption and Decryption: 40% (text files and binary files)