CSE 539 MCS Portfolio Report

Kiyoko Mangrobang
CSE 539: Applied Cryptography
Arizona State University
kmangrob@asu.edu

ABSTRACT

In the following paper, I will describe the four projects required for completing CSE 539 Applied Cryptography. The sections within this report include an introduction or a summary of the four projects, an explanation of the solutions, the results for each project, lastly the lessons learned and skills I acquired for each required project. The four projects included within this report are the Steganography and Cryptanalysis project; Hash project; Diffie-Hellman and Encryptions project; and RSA project.

1. INTRODUCTION

1.1 Project 1: Steganography and Cryptanalysis Project

The overall goal of Project 1 was to attain experience using steganography, cryptanalysis, and random number generators. Within this project, I worked at the byte level of files to create a bitmap image from scratch. This project included two parts; the first part was the steganography portion which included programmatically modifying a file to hide a message inside the bitmap image. The second part of this project was applying cryptanalysis, in which I broke the encryption by finding the key through a weakness within the random number generator.

1.2 Project 2: Hash Project

The overall goal for Project 2 was to apply cryptographic libraries to hash strings using the MD5 library. I obtained the necessary experience to utilize many of the available modern hash functions in C#'s hashing library. I learned how to convert bytes from a string and then applied salt to that string for further secure hashing through the application of an MD5 hashing function. This project also taught me how to find/identify collisions in a hash function using a birthday attack.

1.3 Project 3: Diffie-Hellman and Encryption Project

The overall goal for Project 3 was to attain experience utilizing existing libraries to implement encryption using a 265-bit key. I implemented part of the Diffie-Hellman algorithm to generate the key. Within this project, I learned how to work with extremely large numbers that were too large to store in a standard data type, such as integer or long. For my program to accommodate these arbitrarily larger integers, I utilized C#'s BigInteger struct to support these numbers.

DOI: http://dx.doi.org/10.1145/12345.67890

1.4 Project 4: RSA Project

The overall goal for Project 4 was to acquire first-hand experience in implementing the RSA algorithm. This project tested my ability to work with extremely large prime numbers, given that all of the provided prime numbers were greater than 200 bits. I learned how to implement the Extended Euclidean algorithm and solve for the value of the greatest common divisor. I also learned how to encrypt and decrypt using the RSA algorithm within this project.

2. EXPLANATION OF SOLUTIONS

2.1 Steganography and Cryptanalysis Project

2.1.1 Project 1 Explanation of Solutions: Part 1

I began part one of this project by examining the bmp file structure constructed through header bytes. The goal of part one was to create a bitmap from scratch and hide a secret message inside an image's header bytes without changing its bitmap. I solved this problem through number theory concepts by taking two bits at a time and applying XOR on each byte. I kept the first 26 bytes intact, so I would not alter the 18 general header bytes, two width bytes, two height bytes, and four additional header bytes. The remaining 48 bytes are where I hid the secret message by converting the bits with XOR. These bytes are split up into 16 sets of three bytes per 1 pixel, where each set contains the color BGR (Blue Green Red) format.

2.1.2 Project 1 Explanation of Solutions: Part 2

The goal of part two of this project was to find the encryption key through a weakness in the random number generator. I was presented with a problem scenario that states that an encryption program randomly generates its key. This problem scenario in the program also states that it employs the current time as the seed. Within this project, I had to pass the plaintext and ciphertext inside the command line as arguments to output an integer value. This integer was the value used to seed the random number generator within the encryption program. Through the utilization of DateTime stamping, I was able to solve this problem and identify the seed.

2.2 Hash Project

Project 2 required me to employ an outdated hashing algorithm, MD5 (message-digest algorithm). The overall goal of the hash project was to find collisions using a birthday attack. The application of hashing on passwords is typically assumed to be irreversible. However, the implementation and usage of rainbow tables can create scenarios in which two users with the same password would have the same hash. To help mitigate this issue, you could add a salt to the password. Due to the addition of a salt, users with the same password will have different hashes. Salting the password hashes increases the overall security of passwords stored in a rainbow table [1]. I found the collisions in a hash function by employing the Birthday Paradox and salting the password hashes.

2.3 Diffie-Hellman and Encryption Project

Project three required the implementation of the Diffie-Hellman Key Exchange protocol. The encryption algorithm used in this project is AES (i.e., Rijndael encryption) and utilized the AES class in the C# System.Security.Cryptography namespace. Professor Partha Dasgupta, Ph.D. briefly discusses an example of this algorithm in the lecture video *Certificates on the Web* during week five of this course [2]. To perform the encryption algorithm for this project, I utilized the 256-bit key mode and 128-bit IV passed via the command line in hex. This project required a program to intake nine values from the command line as arguments and would employ the algorithm computations to encrypt and decrypt data.

2.4 RSA Project

Project four required me to understand how to construct the RSA algorithm and how to verify the validity of my output values. To verify I understood the mechanics of the RSA algorithm, I referred to Professor Partha Dasgupta, Ph.D. explanation of this algorithm in the lecture video *Commutative Encryption and Mental Poker* during week seven of this course [3]. The overall goal of the RSA project was to successfully encrypt and decrypt the implementation of the RSA algorithm through the Extended Euclidean algorithm to calculate the value of the greatest common divisor. Similarly to Project 3 (Diffie-Hellman and Encryption Project) I was given prime numbers in the form (e, c). I used the equation $2^e - c$ to calculate the value of the prime number. To implement the RSA algorithm, I had to calculate several values. These values include the prime numbers p and q, e coprime to $phi(n)$, and d such that $e * d \ mod \ phi(n) = 1$. I was required to intake the following six command line arguments in this specific order: p_e in base 10; p_c in base 10; q_e in base 10; q_c in base 10; Ciphertext in base 10; Plaintext message in base 10. Employing the Extended Euclidean algorithm was required to calculate the value of d. I verified this d value through the equation $\mathrm{e} * \mathrm{d} \bmod \mathrm{phi}(\mathrm{n}) = 1$. Only after completing the calculations of the specified four values was I able to decrypt the given ciphertext and encrypt the given plaintext. The output of my program included the decrypted plaintext and encrypted plaintext as a comma-separated pair.

3. RESULTS

3.1 Steganography and Cryptanalysis Project

3.1.1 Project 1 Results: Part 1

I started part one of this project by using the provided code to print out the bytes of the bitmap from scratch. I then constructed a function to copy the first 26 bytes of the bitmap and then performed XOR on the remaining 48 bytes in groups of 2 bits. Those 48 bytes allowed me to hide 12 bytes of data by converting the base size of each byte. I struggled at first to correctly apply this conversion, but after solving this algorithm utilizing BitConverter I finally completed the steganography portion of this project.

3.1.2 Project 1 Results: Part 2

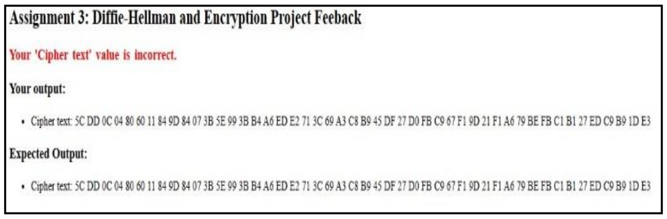
By carefully examining the provided conversion and encryption code in part two, I formulated a similar function for Encrypting a secret string. I applied DateTime to timestamp to my code to exploit the random generator's weakness. The weakness is that C#'s (int)ts.TotalMinutes function is pseudo-random, meaning that if I know the seed value, I can construct the exact random sequence used to make the encryption key. These functions allowed me to identify the value used to seed the random number generator and build the encryption key within the encryption program hence solving the cryptanalysis portion of this project.

3.2 Hash Project

I started this project by writing three simple helper functions. I wrote a randomize function to generate a random plaintext only using alphanumeric characters in the string [A-Z] [a-z] [0-9]. The second function was to compute and salt the MD5 hash. The last helper function is where I checked my MD5 hash table for any collisions within its dictionary. The main idea needed to solve the hash project's problem is to concatenate the salt to a random string. Next, I inputted these salted strings into the hash function I developed. To prevent a birthday attack when applying my hash function, I doubled the number of bits used when hashing my bits [4]. The required output of Project 3 was to identify the two different plaintext values, which held the same hash values and returned those values.

3.3 Diffie-Hellman and Encryption Project

To start Project 3, I created a function to read and parse the nine input parameters written in the command line. Within this function, I wrote a switch case for each of the nine parameters to get parsed accordingly. The three parameters parsed as strings were 128-bit IV in hex, a plaintext message P as a string, and an encrypted message C in hex. The parameters parsed as integers were g_e in base 10, g_c in base 10, N_e in base 10, and N_c in base 10. The remaining parameters were all parsed as BigIntegers. These included x in base 10, g^y mod N in base 10, and the N used in this mod. To accommodate the arbitrarily larger integers, I utilized C#'s BigInteger struct to prevent any encounters of a bit overflow for these last parameters. One issue I faced when submitting my project was the format in which I was printing my output. Both my decrypted plaintext value and ciphertext output values were correct separately. Displayed below in Figure 1 is a screenshot of the feedback that the auto-grader provided. It states that my cipher text value is incorrect even though the output and the expected values are equivalent.



Assignment 3: Diffie-Hellman and Encryption Project Feeback

Your 'Cipher text' value is incorrect.

Your output:

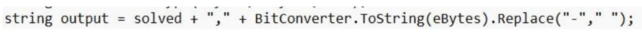
- Cipher text: 5C DD 0C 04 80 60 11 84 9D 84 07 3B 5E 99 3B B4 A6 ED E2 71 3C 69 A3 C8 B9 45 DF 27 D0 FB C9 67 F1 9D 21 F1 A6 79 BE FB C1 B1 27 ED C9 B9 1D E3

Expected Output:

- Cipher text: 5C DD 0C 04 80 60 11 84 9D 84 07 3B 5E 99 3B B4 A6 ED E2 71 3C 69 A3 C8 B9 45 DF 27 D0 FB C9 67 F1 9D 21 F1 A6 79 BE FB C1 B1 27 ED C9 B9 1D E3

Figure 1. Auto-grader feedback with whitespace.

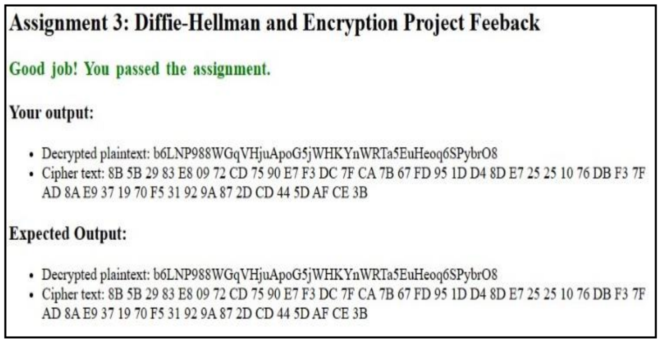
To figure out why my project was not passing, I carefully worked through my code, checking all of the calculations I employed. After many painful hours of confusion and requesting help from my peers, I finally located my issue. The fixed code is displayed in Figure 2 below.



```
string output = solved + "," + BitConverter.ToString(eBytes).Replace("-"," ");
```

Figure 2. Code where I found my error.

The format for concatenating the decrypted plaintext and cipher text was supposed to be connected as follows “decrypted plaintext,cipher text” While my program was printing out “decrypted plaintext, cipher text” This slight difference was my error. A single white space I added after the comma caused the failure of my program and not any of my implemented code calculations. Displayed on the next page, Figure 3 shows the auto-grader feedback after removing the white space in-between the decrypted plaintext and the cipher text.



Assignment 3: Diffie-Hellman and Encryption Project Feeback

Good job! You passed the assignment.

Your output:

- Decrypted plaintext: b6LNP988WGqVHjuApoG5jWHKYnWRTa5EuHeoq6SPybrO8
- Cipher text: 8B 5B 29 83 E8 09 72 CD 75 90 E7 F3 DC 7F CA 7B 67 FD 95 1D D4 8D E7 25 25 10 76 DB F3 7F AD 8A E9 37 19 70 F5 31 92 9A 87 2D CD 44 5D AF CE 3B

Expected Output:

- Decrypted plaintext: b6LNP988WGqVHjuApoG5jWHKYnWRTa5EuHeoq6SPybrO8
- Cipher text: 8B 5B 29 83 E8 09 72 CD 75 90 E7 F3 DC 7F CA 7B 67 FD 95 1D D4 8D E7 25 25 10 76 DB F3 7F AD 8A E9 37 19 70 F5 31 92 9A 87 2D CD 44 5D AF CE 3B

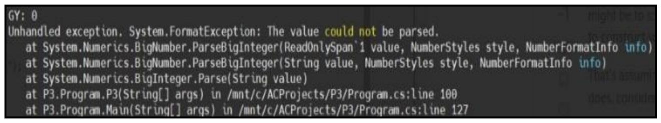
Figure 3. Auto-grader feedback without whitespace.

3.4 RSA Project

To begin Project 4, I created a BigInteger helper function to calculate the prime greatest common divisor utilizing the Euclidean algorithm. Every variable used within this function was initialized as a BigInteger to accommodate these arbitrarily larger integers, similar to Project 3 (Diffie-Hellman and Encryption project). I read and parsed the input parameters from the command line inside my string function. Once I parsed these values, I executed both calculations of the prime numbers p and q using the BigInteger.Subtract() method. Next, I utilized the BigInteger.Multiply() method to calculate e coprime. Lastly, I calculated d such that $e * d \; mod \; phi(n) = 1$ utilizing my BigInteger helper function. After completing the calculations of these four specified values, I decrypted the given ciphertext and encrypted the given plaintext. I saved the decrypted ciphertext as a string and the encrypted plaintext as a string. Then I concatenated them together with a comma. I completed this project by printing out this concatenated string of both ciphertext and plaintext.

4. CONTRIBUTIONS & LESSONS LEARNED/SKILLS ACQUIRED

The projects included in this portfolio report were all completed individually. One lesson that I learned while working on the Steganography and Cryptanalysis Project was the power and real-world application of XOR on bits. Before beginning this project, my knowledge of the application of bitwise XOR was limited to truth tables. Now I have learned how to apply this to encrypt and hide secret messages within the bitmap of an image. When running the Steganography and Cryptanalysis project, I encountered an unhandled exception on my local computer, as referred to in Figure 4 below.



```
GY: 0
Unhandled exception. System.FormatException: The value could not be parsed.
   at System.Numerics.BigNumber.ParseBigInteger(ReadOnlySpan`1 value, NumberStyles style, NumberFormatInfo info)
   at System.Numerics.BigNumber.ParseBigInteger(String value, NumberStyles style, NumberFormatInfo info)
   at System.Numerics.BigInteger.Parse(String value)
   at P3.Program.P3(String[] args) in /mnt/c/ACProjects/P3/Program.cs:line 100
   at P3.Program.Main(String[] args) in /mnt/c/ACProjects/P3/Program.cs:line 127
```

Figure 4. Project 1 unhandled exception.

However, when I submitted my project to the auto-grader it returned correctly. Working around this exception, I checked my project's output by submitting a zip of my project after each iteration change. I completed Project 1 this way, even though verifying my output was unfavorable.

The Hash project taught me how to prevent birthday party attacks and increase security when storing hashed passwords. Learning how hash tables can be applied to storing data such as passwords is a powerful tool when developing and securing a website's login system.

Additional skills I acquired while completing the following projects was the ability to apply algorithms I have only learned about in theory to real-world problem solutions. Working through these projects, I was reminded of the importance of formatting when concatenating and printing output values. A single white space is powerful enough to cause an issue in your output especially when submitting projects through an auto-grader. I experienced this in Project 3, Diffie-Hellman and Encryption project, and was reminded in Project 4, RSA Project.

5. REFERENCES

[1] Partha Dasgupta, PhD. Constructing Rainbow Tables. C*oursera*. Retrieved April 1, 2023, from https://www.coursera.org/learn/cse539appliedcryptography/lecture/EGWAF/constructing-rainbow-tables

[2] Partha Dasgupta, PhD. Certificates on the Web. C*oursera*. Retrieved April 20, 2023, from https://www.coursera.org/learn/cse539appliedcryptography/lecture/OoLIj/certificates-on-the-web

[3] Partha Dasgupta, PhD. Commutative Encryption and Mental Poker. C*oursera*. Retrieved April 28, 2023, from https://www.coursera.org/learn/cse539appliedcryptography/lecture/IQifj/commutative-encryption-and-mental-poker

[4] Partha Dasgupta, PhD. Constructing and Preventing Birthday Attacks. C*oursera*. Retrieved March 24, 2023, from https://www.coursera.org/learn/cse539appliedcryptography/lecture/LHrtm/constructing-and-preventing-birthday-attacks