William Duke and Ian Shaneyfelt
Professor Chen Wang
CSC 4562 - Mobile Sec
1 May 2024

CSC4562 Programing Project

1. For this problem I chose to implement a simplified DES in python. For the encryption function, the plaintext is divided into blocks of 8 characters each, and each block goes through 16 rounds of a permutation function. The permutation function adds the ASCII value of each character in a block to the corresponding character in the key. The value of this is then mod by 256 and converted back into a character. The decryption function applies the inverse permutation function to each block of the ciphertext for 16 rounds. The inverse function subtracts the ASCII value of the key from the ciphertext mod 256. This project helped us understand the basic principles of symmetric key cryptography and gain hands-on experience with the encryption and decryption processes. Here are the results of running the code.

   ```
   Plaintext: plaintext test
   Key: 0123456789abcdef
   Ciphertext: 707cc281c299c2aec384c385c3a87430c294c295c2b3c384
   Decrypted plaintext: plaintext test
   ```

2. For this part of the project I used python because I am very familiar with the language and comfortable with the syntax especially for mathematics. To start off with the prime checker I created a function utilizing Fermat's Little Theorem to quickly check for primes, however I am aware of the flaw that there are several edge cases but in instance I value run time over edge cases. Now onto RSA proper I created 3 mathematical functions one to find the greatest common divisor and the other two to calculate inverse modulus which I found on a stackoverflow forum. I chose to code my own versions of these functions instead of using imports as I believed it would be more beneficial to my understanding of the material. After that I simply followed the steps of RSA encryption in creating the variables e, d, p, q, phi, and n, separated into functions to enhance readability. I chose to use arrays as the data structure to store the keys in as it makes accessing the variables more efficient. Encoding the values of the message did trip me up as encoding a to 0 means it will never change in the way my values are calculated, mostly multiplication, but other than that no real problems on creating this code.

3. For this project I decided to use python so I could take advantage of these libraries: hashlib, time, random, string, and matplotlib.pyplot. To solve this puzzle I made a function called solve_puzzle that randomly generates Alice's P value of B bits long. Next Bob makes a message that is a random selection of 20 ascii letters and/or digits. This

length was chosen since the largest number of bits is 16, meaning that there are 2^{16} possible choices of P, so I need to make the possible choices of M much higher. After crafting an M, M gets hashed using the python's hashlib library to convert the message to a sha256 hash. Then the last B bits of the hash are compared to P. If they aren't equal, a new M is made until one that solves P is found. Once a “correct” M is found the program saves this time and does 9 more trials and averages the time. This happens for P lengths of 4, 8, 12, and 16 bits. Finally, the average time to solve for each P length is plotted on a line graph where the x-axis is lengths of P and the y-axis the average time to solve in milliseconds. After running the program it generates a line graph that shows that the time to solve correlates exponentially to the length of P. This is expected since as the length of P increases, the probability of a hash matching P decreases exponentially due to the exponential increase in possible P choices.