Classic Crypto and Crypto Math

Dana Thomson

September 2021

Description:
This is the first part of the cryptography portfolio. This document covers the classic crypto and crypto math sections of the portfolio. The classic crypto section delves into affine ciphers, vigenere ciphers, affine attacks, vigenere attack, hill ciphers, and one time pass ciphers. Each of the cipher type can both encrypt or decrypt a given text. The crypto math section delves into useful algorithms for cryptography. This includes algorithms that are commonly included in math libraries and ones which are not. This document will serve as a user guide and walkthrough.

Algorithms and Libraries used:
I used 2 python libraries random and numpy. Random is used for the random.randint function. This function is used for pseudo random number generation. The numpy library is used for dot and matmul functions. The dot product function is used in finding the vigenere key and the matmul is used for hill cipher.

Function and Program Structure:
My system has 3 files, main.py, classiccrypto.py, and cryptomath.py. Main contains the user menu and input. It is a starting point for the system. Classiccrypto contains functions for affine encryption/decryption, vigenere encryption/decryption, hill encryption/decryption, one time pass encryption/decryption, frequency analysis, affine brute force attack, affine known plain text attack, and vigenere attack. The Cryptomath file contains function for gcd, egcd, and mod inverse. The cryptomath functions are used throughout the classiccrypto functions as helper functions. I will describe the functions in more detail now.

main.py
Main: Prints the user menu and waits for user input. If the input is not a valid option the menu will print again and wait for valid user input.

classiccrypto.py affine: Affine asks the user to input alpha and beta values and then asks whether the user would like to encrypt or decrypt. If the user chooses encrypt affine_encrypt is called and if the user chooses decrypt affine_decrypt is called.

affine_encrypt: Affine encrypt takes the user given alpha and beta values as parameters. The user is then asked for the file name that contains the plain text. If the name given is not a valid file name the program will return to the main menu and user input. If the file is valid and opened then every character is checked to see if it is in the valid char string after calling .upper() on the letter. Then each letter is the converted from ascii to 0-25 values, multiplied by alpha, and beta is added to the result. The resulting value mod 26 is then added to the encrypted string. Once all characters in the file have been encrypted the encryption is written to encrypted_affine.txt.

affine_decrypt: Affine decrypt takes the user given alpha and beta values as parameters. The alpha value is sent to the cyrpto math mod inverse function to get the decryption alpha value, dalpha. The user is then asked for the file name that contains the cipher text. If the name given is not a valid file name the program will return to the main menu and user input. If the file is valid then each letter is the converted from ascii to 0-25 values, beta is subtracted, and then dalpha is multiplied to get the result. The resulting value mod 26 is then added to the decrypted string. Once all characters in the file have been decrypted the decryption is written to decrypt_affine.txt.

affine_brute_force_attack: This function asks the user for a file with the cipher text. It then tries all 312 options for an affine cipher alpha and beta values. Each decryption is done on 30 characters and printed to the file affine_brute.txt along with the alpha and beta values used. This allows the user to look through and see the alpha and beta values that correspond to the decrypted text.

affine_given_plain_text: This function asks the user to input two characters of known plain text and the two corresponding characters in the cipher text. Then the function creates a system of linear equations with these values. First we solve for alpha by subtracting the equations. Then beta is found using the alpha value.

vigenere: This function asks the user for the cipher key and then if the user would like to encrypt or decrypt. If encrypt is chosen the function vigenere_encrypt is called. If decrypt is chosen the function vigenere_decrypt is called.

vigenere_encrypt: This function asks the user to input the file name of the plain text. For each letter which is in the valid chars, the letter value(0-25) is added to by the current key value(key[i]) to get the new encrypted letter. The key location is then incremented and if it is now the length of the key string it is reset to zero. Once all the text has been encrypted it is written to encrypted_vigenere.txt

vigenere_decrypt: This function asks the user for the filename containing the cipher text. Then for each letter in the cipher text the current key value(key[i]) is subtracted from the letter. This decrypted letter is added to the decrypted string and the key location is incremented by one. If the key location is the

length of the key then it is reset to zero. Once all the cipher text has been decrypted it is written to decrypt_vigenere.txt

vigenere_attack: This function asks the user to enter the file name which contains the cipher text. It then calls the vigenrere_find_key_length function and the vigenere_find_key function.

vigenere_find_key_length: This function takes the cipher text as a parameter. Then using a shifted copy of the cipher text it looks for the number of coincedences, number of letters which are the same in both texts. The function repeats this for each possible shift of the text, shifting to the left one character each time. If the number of coincidences is greater than the previous max and the current location, number of shifts, is not a multiple of the max then the current coincidences become the max and the max location is set to the current location. The algorithm doesn't set the max on multiples because if the key length is shorter than the text then all multiples of the key length should have high coincidences. It is most likely for the smallest multiple that gave a new max value to be the key length. The max location gives us the key length. This is printed to the screen.

vigenere_find_key: This function takes the cipher text and key length to find the key value. For each value 0-n, where n is the key length, a new text is made with just the characters in positions mod n. This text is then used to create a frequency analysis which returns an array. We will use this frequency array to determine the most likely character for this position of the key. To do this we take the dot product of the previously described array with an array containing the English language frequency analysis values. To find which letter is correct we must take this dot product 26 times, each time shifting the English language frequency array to the left one space and wrapping the shifted item. The dot product which produces the highest values is the letter in that position of the key. This process is repeated for the length of the key and the key is printed once each value has been found.

hill_cipher: This function asks the user to input the e for encryption or d for decryption. If e is input then hill_cipher_encrypt is called and if d is input then hill_cipher_decrypt is called.

hill_cipher_encrypt: This function uses a 3x3 matrix M as the encryption matrix. The user is asked to input the file name of the plain text. Then the characters are grouped into strings of three characters. Then the string of three letters is put into an array using their numeric, 0-25, values. This is then multiplied by the encryption matrix to produce the encrypted array. Each value in the encrypted array is added to the encrypted text. If the length of the cipher text is not divisible by 3 then the last string is padded with random characters. It is up to the person reading the decrypted text to realize the characters are padding. The encrypted text is written to encrypted_hill.txt.

hill_cipher_decrypt: This function uses the matrix N as the decryption matrix. N is the inverse matrix of encryption matrix M. First the user is asked to enter the file name of the cipher text. Then each character is read into a string in groups of three. This string is used to assign value to an array using the numberic, 0-25, values of each letter. The array is then multiplied by the matrix N to get the decrypted array. The decrypted array is then added to the decryption text. The decrypted text is written to decrypt_hill.txt.

one_time_pad: This function asks the user to enter e for encryption and d for decryption. Then the user is asked to input a integer seed for pseudo random number generation. If e was input one_time_pad_encrypt is called and if d was input one_time_pad_decrypt is called.

one_time_pad_encrypt: This function takes an integer seed as a parameter, this seed is used to garuentee the same random numbers are generated for encryption and decryption. The user is asked to input the filename with the plain text. Then random.seed(seed) is called to seed the random number generation. For each letter in the plain text the random generated number is added to the value. The mod 26 of this result is added to the encrypted text. The full encryption is written to encrypted_opt.txt.

one_time_pass_decrypt: This function asks the user to input the filename of the cipher text. Then the seed parameter is used to seed random number generation through the random.seed function. Then for each letter in the cipher text the random number generated is subtracted from the letter to get the decrypted value. The full decrypted text is written to decrypt_otp.txt.

frequency_calculation: This function calculates the percentage of usage of each letter in the english language for a user given input file. The function loops through each character in the text adding it to the corresponding dictionary position. Then each count is divided by the total number of characters counted. The dictionary is printed to show the percentage corresponding to each letter.

frequency_calculation_text_param: This function takes the input text as a parameter and does the same calculation as the above function but uses an array instead of a dictionary. The array is returned at end of the function. This function was added to allow for easier use in algorithms while the above function is used for user display.

cryptomath.py
gcd: This function takes two integers as input. First it checks that the first int, a, is larger than the second int, b. If it isn't then a and b are swapped. Then while b does not equal zero a becomes b and b becomes a mod b. Finally be return a which is the greatest common divisor.

extended_gcd: This function takes two integers a and b as input. First it checks that the first int, a, is larger than the second int, b. If it isn't then a and b are swapped. The egcd finds the gcd and the linear combination of a and b which will generate the gcd. To do this while b is not zero a becomes b, b becomes a mod b, linearX is calculated to be x0 minus x1 times the quotient of a and b, and linearY is calculate y0 minus y1 times the quotient. Then x0, y0 are set to x1, y1 and x1, y1 are set to linearX, linearY. Then a, x0, and y0 are returned. These provide the gcd and the linear combination values for a and b.

mod_inverse: This function calls the extended_gcd function if the gcd is one then there is no possible inverse. If it is not one then the inverse is returned as x mod n. Where n is the mod number.

How to compile:
To run this program simply run the main.py file. Be sure classiccrypto.py and cryptomath.py are in the same directory as main.py. I have been running the program through the command line using the command 'python main.py'.

Testing and Verification:
To test this program I have ran each encryption and decryption algorithm on small then large input texts to verify that they are working correctly. For each attack algorithm I ran them multiple times to determine that they would find the correct answers in different situations. For the crypto math functions I tested agaisnt known values provided in the book and online.