Linnéuniversitetet
Kalmar Växjö

Linnaeus University

1DV700 - Computer Security
Assignment 1

Student: Ebbe Karlstad
Personal number: 041025–5434
Student ID: ek224ev@student.lnu.se

Setup Premises

Everything on this assignment will be done on a Lenovo 7 Pro Laptop running Windows 11 with the main web browser being Brave Browser. The code will be written using the VS Code text editor in Python, using the Pylance extension and Flake8 for linting.

Task 1

a) Symmetric encryption is a form of encryption where plain text, or whatever is supposed to be encrypted does so with a key, and when that encrypted message is to be decrypted, it does so with the same key. Since the encryption and decryption key is the same, it is easy to do but less secure and requires a safe way to transfer the key between parties. Asymmetric encryption is a form of encryption where the encrypted message gets encrypted with a public key, and then gets decrypted with a private key, where the public and private keys are different. This process is slower, but also more secure [1].

Encryption and Hashing algorithms are a bit different. When you encrypt something, you scramble the message you want to encrypt, with a key, which will result in a ciphertext. This means that with that key you can also decrypt it, returning it into plain text. Encryption is a two-way process. Hashing on the other hand, converts information using a hash function, and returns a hash key, which cannot be reverted, hence it is a one-way process. Examples of encryption algorithms are RSA and AES, and examples of hashing algorithms are MD5 and SHA256 [2].

Hashing, as mentioned before, is the process of converting information using a hashing algorithm, or a hash function, to create a hash key. Compressing information works a bit differently, by encoding information using fewer bits than the original message. A device that does compression is usually referred to as an encoder, and a device that decompresses information is called a decoder [3].

b) Steganography is often described as hiding something in plain sight. This is usually done by embedding digital information in standard files, this could be embedding code with a message in it into an image file and sending that in an email to the person the message is for [4].

Digital watermarking is very similar to steganography, with one deviation, in digital watermarking, if a message is embedded and hidden inside an image, the message can be either hidden or visible, in steganography, the message is always hidden. [5]

The biggest difference between encryption and steganography/digital watermarking is that encryption hides the message, but not the existence of it, while both steganography and digital watermarking hides the existence of the message itself.

Task 2

a) For this simple substitution cipher, we have the encrypted message: "HKPUFCMHY BHDDXZH" and we want to decrypt it using the table provided in the question. To do this, we simply look at the table and replace the ciphered letters with the corresponding plain text letters, so H becomes e, k becomes n and so on. The final decrypted message is: "encryption message".

b) For the other simple substitution cipher, we have the encrypted message: "NWSRC XQS JXB CWRGABY WKH VWUNWBA" and we want to decrypt it without knowing the key. When I saw this question, I immediately thought of the monoalphabetic cipher, which is simply replacing a letter in the plain text message with another letter to get the ciphered text. Note that the letters used to encrypt this message usually does not have any relation to each other, which makes it harder to crack.
The way to approach these ciphers is to either use the frequency of the most used letters in the encrypted text and replace them with the most frequently used letters in the English language. Another way is to simply notice the combinations of the letters in the shortest words in relation to each other, and try to guess the words based on this, and then replace the letters you 've come up with to figure out the letters of the other words. This is the method I did to find the result. I started with the words XQS and JXB and after a while came up with the words OUT and FOR, which makes sense in a sentence.
After this I replaced X, Q, S, J, X and B with O, U, T, F, O and R, which got me closer to the answer. In the next step of the process, I added the word AND instead of WKH since it made sense and updated the text. At last, I could guess the rest of the words and came up with "WATCH OUT FOR HACKERS AND MALWARE".

This way of encrypting is harder to crack than, let's say, a Caesar Cipher, since there is no direct correlation between the letters being used as the key, but it is still possible. Ways to make it harder to crack is avoiding short words in your message and using fewer common words, but even then, with the right techniques, it can be decrypted. As for making a substitution method "secure enough", I don't think any encryption method ever will be secure enough, and even if we were to find a method which is extremely hard to decrypt without the key, the Vernam Cipher for example, the question is still very hard to answer. [6]

Task 3

For task 3, I read the instructions and started working on my program. I wrote the program in Python, as that is the coding language I'm most familiar with and started implementing the substitution cipher. The substitution cipher is a cipher which replaces the letters within a message in some way, while retaining their position. A great example of this is a Caesar cipher, which is the cipher that replaces all letters with their corresponding ciphered letter some n down the alphabet. Here 'n' is the key. First however, I did a simple menu which prompts the user for various things, such as if they want to encrypt/decrypt, what key they want and if they want to do a substitution cipher or a transposition cipher.

After this, I created a simple script for my Caesar cipher, which goes through all the letters in the text file the user wants to encrypt/decrypt and replaces those letters with their corresponding ciphered letters n steps down the line. This was done by converting the key to an int and reversing it if the mode was 'd' for encryption. I also added a check to ensure it is in the 8-bit range (with % 256), to fulfil the requirements that the user should be able to have an 8-bit key. Then I iterated over every character in the message and avoided everything in the file that I wanted to keep (like spaces and asterisk) and converted them to their ASCII values. After this I shifted the characters using the modulus operator and wrapped the text around in the 8-bit range. At the end I appended the shifted characters to the result and returned it.

After this I started working on the transposition cipher, which was a bit trickier. For this one, I went with the rail-fence cipher. The Rail Fence Cipher is a form of transposition cipher that creates a zigzag pattern with the plaintext to obscure it. I started by defining a function to build a matrix suitable for the Rail Fence Cipher. This matrix was constructed with a specified number of rows and columns, initially filled with empty strings. I chose to only consider characters in the alphabet, to keep the layout from the original plaintext.txt, so as I iterated through the text, I placed each character in the matrix and skipped over the other ones.

For encryption, I read the matrix in a zigzag pattern. I created a function that reads the matrix column-wise and constructs the ciphertext. To decrypt, I first isolated alphabetic characters from the ciphertext and then filled the matrix column-wise. The plaintext was reconstructed by reading the matrix in the zigzag order to put the non-alphabetic characters back in their original order.
In my transposition_cipher function, I determined the number of rows based on the provided key and the number of columns by dividing the length of alphabetic characters by the number of rows, adjusting for any remainder with an additional column if needed. Depending on the mode ('e' for encryption, 'd' for decryption), I then called the appropriate functions to either encrypt or decrypt the text.

For the processing of the files, first, I created a function to open and read the contents of a file. I made sure to handle the file operations using the with statement to ensure that the file is properly closed after its contents are read. Next, I provided an option to choose the cipher method. If the method variable is '1', I used the substitution cipher on the content of the file. If it is '2', I converted the key to an integer since the transposition cipher requires a numerical key, and then applied the transposition cipher.

I included a check for the mode variable. If the mode is 'e', it is encryption, and I set the output file name to "encrypted.txt". If the mode is 'd', it stands for decryption, and I set the output file name to "decrypted.txt". After processing the content with the right cipher and mode, I simply wrote the result to the output file. Once the result was written to the output file, I printed a message to tell the user that the process was completed successfully.

Task 4

For this task, I followed the instructions and downloaded the plaintext.txt file as well as added my secret message, which I chose to be a quote from my favourite series, Mr. Robot. I also added my name at the bottom and uploaded my plaintext file to the designated folder.

Task 5

For task 5, I downloaded four random files from the folder mentioned in the assignment and used an online Caesar cipher calculator to brute-force the text back to its normal form. The ones I successfully cracked were oa222sv.txt and ff222mn.txt. User oa222sv's text file was decrypted using the Caesar cipher with a key of 6, and ff222mn's text file was decrypted, also using the Caesar cipher, but with a key of 5 this time. Both of their encrypted files can be found in my project folder.

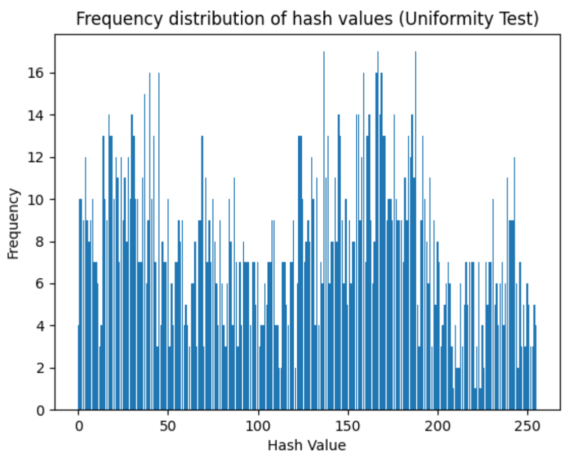
There were a few files I couldn't crack, like user no222hj which from the looks of it, implemented a transposition cipher. I tried breaking it on my own and with multiple online tools but couldn't do it.

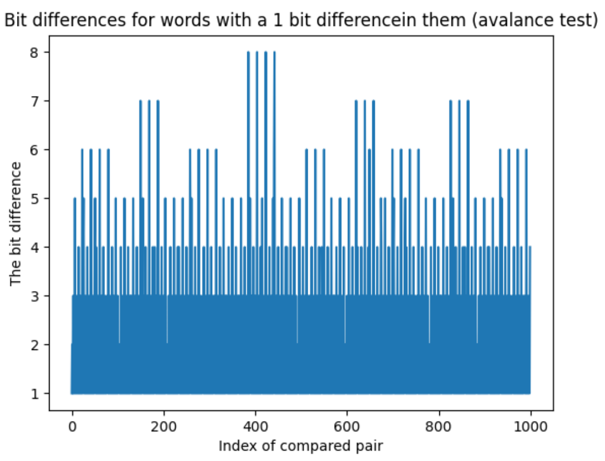
Task 6

a) For this task I implemented a very simple hash function that basically just calculates the ASCII values of the characters in the input string and uses the modulo operation to confine the range of the hash to 8 bits. Note that this method is not secure and would not be appropriate to use in a "real" setting, we would in that case instead use a better and more secure hashing algorithm like SHA. [7]

b) For the uniformity test I tested with a 2000-word long list of random English words. An ideal uniformity test would show an almost evenly flat line, since that means that every hash value occurs almost the same number of times as the others. My uniformity test shows a few deviations, which indicates that the hash function is not very secure. This deviation can lead to an uneven distribution of hash values, potentially resulting in an increased likelihood of hash collisions [8].

For the avalanche test I tested with a 1000-word long list of words ranging from aaaaaaaa, aaaaaaab, aaaaaaac, and so on, up until we reach 1000 words. The ideal avalanche test would be a distribution where the bit differences are centered around half of the hash size. For an 8-bit hash, this would mean an average bit difference of around 4 bits. [8], which the code I wrote, has not. Instead, it has an average bit difference of 2.504, which we also can see in the graphs made using Matplotlib.





c) The main difference between secure hash functions and regular hash functions is in how they are intended to be used. Since cryptographic hash functions are designed to be secure, they can withstand a variety of attacks and satisfy certain security requirements, which regular hash functions don't. Some of these security requirements are:

- Pre-image resistance: It should be difficult to reverse-engineer the original input from its hash value.
- Second pre-image resistance: It should be difficult to find a different input that produces the same hash value as a given input.
- Collision resistance: It should be difficult to find two different inputs that produce the same hash value.
- Pseudo-randomness: The hash function's output should appear random and not reveal any structure or predictability.

The easiest method to show that a hash function isn't secure is to show that it fails to meet the mentioned requirements. For example, it is relatively easy to find collisions due to the limited range of outputs (only 256 possible hash values), which violates the collision resistance property. Furthermore, inputs with the same letters in a different order would return the same hash because the hash is based on the sum of ASCII values, obviously passing the second pre-image resistance test. Pre-image resistance is also weakened by the function's simplicity, since it is easy to attempt every possible combination of characters that add up to a given hash value. [9]

Bibliography

[1] gluttony777, ”GeeksForGeeks,” 22 May 2023. [Online]. Available: https://www.geeksforgeeks.org/difference-between-symmetric-and-asymmetric-key-encryption/. [Använd 13 November 2023].

[2] CoderSaty, ”GeeksForGeeks,” 7 May 2023. [Online]. Available: https://geeksforgeeks.org/difference-between-hashing-and-encryption/. [Använd 13 November 2023].

[3] S. -J. Behrens, ”JavaCodeGeeks,” 26 September 2020. [Online]. Available: https://www.javacodegeeks.com/2020/09/hashing-encryption-encoding-compression-oh-my.html. [Använd 13 November 2023].

[4] ”How does steganography work and does it threaten enterprise data?,” 13 January 2014. [Online]. Available: https://moodle.lnu.se/pluginfile.php/8283582/mod_resource/content/1/How%20does%20steganography%20work%20and%20does%20it%20threaten%20enterprise%20data%3F.pdf. [Använd 13 November 2023].

[5] C. Brook, ”Digital Guardian,” [Online]. Available: https://www.digitalguardian.com/blog/digital-watermarking. [Använd 13 November 2023].

[6] S. L. P. J. M. Charles P. Pfleeger, ”Security in Computing,” i *Security in Computing, Fifth Edition*, Prentice Hall, 2015, p. 863.

[7] ”What is SHA? What is SHA used for?,” Encryption Consulting, [Online]. Available: https://www.encryptionconsulting.com/education-center/what-is-sha/. [Använd 27 November 2023].

[8] ranadeepika2409, ”What are hash functions and how to choose a good hash function,” Geeks For Geeks, 21 Februari 2023. [Online]. Available: https://www.geeksforgeeks.org/what-are-hash-functions-and-how-to-choose-a-good-hash-function/. [Använd 27 November 2023].

[9] ”Security of Hash Functions in Cryptography,” Coding Ninjas, 16 September 2023. [Online]. Available: https://www.codingninjas.com/studio/library/security-of-hash-functions-in-cryptography. [Använd 27 November 2023].