DAT510: Assingment 1

DAT510: Assingment 1
- Abstract
- 1. Introduction
- 2. Design and Implementation
  - 2.1. Apply Encryption
    - Caesar cipher
    - Columnar transposition
      - Example of transposed matrix
    - Decryption
  - 2.2. Evaluate the Avalanche Effect
  - 2.3. Analyze the Avalanche effect
  - 2.4. Optimize Avalanche Effect with Reasonable Computation
  - 2.5. Enhance Security with Block Ciphers
- 3. Discussion
- 4. Conclusion
- References

Abstract

1. Introduction

The main goal of this project is to implement different forms of classical and modern enctpytion methods. These encryption methods should then show how the Avalanch effect differs between them. They should be a practical example of conveying how more advanced encryption methods use the avalanche effect to increase security and robustness of the ciphers.

Classical encryption methods mainly use different variation of substitution and/or transposition ciphers. Substitution ciphers work by replacing letters, while transposition ciphers reorder them. There exists multiple genres of substitution ciphers 1. There are simple ones like the Caesar cipher, which shifts the alphabet by a fixed amount, or more advanced polyalphabetic ciphers like the the Vigenere cipher or other mechanical based rotor machines 2 3. These use multiple alphabets to replace the plaintext, which increases security by breaking up the frequency patterns.

Transposition ciphers encrypt the text by rearranging the letters in a specific order. A columnar transposition cipher reads the letters in a matrix row by row, then reads them out column by column, in the order specified by a key 4. The strenght of the cipher generally depends on the key and lenght of the plaintext. A short message is quickly decrypted by brute force or other frequecy analysis techniques, while the longer the message the harder it is to break given the right conditions 5.

Encryption algorithms must have a avalanch properties to be secure. Common properties are following the Strict Avalanch Criterion (SAC) and the Bit Independence Criterion (BIC). The SAC formalizes a criteria of how the amount that any change in the input affects output. A consistent change of 50% is concidered the best as it maximized the confusion and diffusion properties 6 8. You want a small change to propagate a lot, but if it propagates too much, the way in which it changes can be used to deduce the key. Following the BIC criteria means that the output bits not only change but change statistically independently of each other 7. This way it should completely remove any patterns from the plaintext in the ciphertext.

2. Design and Implementation

In our case, we will implement and combine a simple Caesar cipher and a columnar transposition cipher. The ceasar cipher only adds confusion to the cipher, and columnar transposition only diffusion. By combining them, we hope to increase the security by applying the properties required by the SAC and BIC.

2.1. Apply Encryption

The Ceasar cipher implementation creates a simple character map based on the shift key. It creates a dictionary where the key is the original alphabet letter, and the value is the shifted letter. The `chr()` function is used to create the character from the numerical ASCII representation. To help standardise and increase security all string were converted to uppercase and spaces and other special characters were removed. The range of `A-Z` in a ASCII uppercase are the numbers 65 - 90. To create the caracter map 65 is used as the base, then the shift is used in combination with the index to map to the correct character. Due to the fact that the alphabet gets shifted. The last x number of keys will go past `Z`. To loop back around and continue with `A`, modulo is used to always just get the remainder whis is then added to the base. When used on a text, it loops through each character and for each letter, looks it up in the character map and replaces it with that value.

```
PROCEDURE CAESAR(plaintext, shift):
    character_map = {}
    FOR i = 0 to 25:
        character_map[chr(i + 65)] = chr((i + shift) % 26 + 65)
    END FOR

    ciphertext = ""
    FOR each character in plaintext:
        ciphertext += character_map[character]
    END FOR
    RETURN ciphertext
```

The columnar transposition cipher on ther hand is a bit more complicated. It requires several steps. The key needs to be split into a list of individual characters. A matrix is generated based on the number of rows and columns from the plaintext. If the plaintext is not a multiple of the key, it is padded with "X". The matrix is transposed to swap the rows and columns. The matrix is then read out again row by row, but this time it has dimensions `m x n` instead of `n x m`. The key determines the order of the columns. The index of the key is used to determine the order of the columns. The index is then used to read out the columns in the correct order. The result is then concatenated to form the ciphertext.

```
PROCEDURE COLUMNAR(plaintext, key):
    kl = [] # The key split into a list of individual characters
    cols = len(kl)
    rows = ceil(plaintext) / len(cols)

    # If the plaintext is not a multiple of the key, pad with "X"
    text = plaintext + "X" * (rows * cols - len(plaintext))

    # Create a matrix from the text with dimensions rows x cols
    matrix = []
    FOR i = 0 to rows:
        matrix.append(text[i * cols : (i + 1) * cols])
    END FOR

    # Transpose the matrix then read it out row by row
    matrix = matrix.T

    ciphertext = ""
    FOR i = 1 to cols:
        key_index = kl.index(key[i])
        ciphertext += matrix[key_index][:]
    END FOR
    RETURN ciphertext
```

The order these ciphers are applied is important due to the fact that they bring different properties to the table. The Caesar cipher adds confusion, while the columnar transposition adds diffusion. The order of operations should be the substitution cipher first, and then the transposition cipher[9]. This ordering is due to the fact that confusion removes the recognizable elements from the plaintext and, in some cases, also attempts to disrupt recognizable patterns. Diffusion can then be used to ensure a propper reshuffling to prevent detectable patterns.

The keys and plaintext:

Type	**Value**
Plaintext	Erik Martin Security and Vulnerability in Networks
Transposition key	31425
Substitution key	5

Caesar cipher

The plaintext "Erik Martin Security and Vulnerability in Networks" is first preprocessed which converts it all to uppercase and removed identifying special characters. The result being "ERIKMARTINSECURITYANDVULNERABILITYINNETWORKS". This is then passed to the `caesar()` funciton along with the substitution key. The function then generates a letter map: `{'A': 'F', 'B': 'G', ..., 'Z': 'E'}`. It then loops over each plaintext letter and looks it up in the letter map.

Num	**Plain**	**Cipther**
1	E	J
2	R	W
3	I	N
...	...	...
44	S	X

Afterwards it merges it into a string and the result is "JWNPRFWYNSXJHZWNYDFSIAZQSJWFGNQNYDNSSJYBTWPX".

Columnar transposition

The next step is substitution using columnar transposition. The cipher text is passed along to the columnar() function along with the transposition key. This key is converted to a list of int `[3, 1, 4, 2, 5]`. The amount of columns are `5` due to the length of the key, and number of rows is

$$cols = len(key) = 5$$

$$rows = \lceil \frac{len(plaintext)}{len(key)} \rceil = \lceil \frac{44}{5} \rceil = \lceil 8.8 \rceil = 9$$

The plaintext is then padded with "X" until it is the multiple or `rows*cols` and it becomes "JWNPRFWYNSXJHZWNYDFSIAZQSJWFGNQNYDNSSJYBTWPXX". This is then entered into a numpy matrix and transposed, becomming:

Example of transposed matrix

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
3	J	F	X	N	I	J	Q	S	T
1	W	W	J	Y	A	W	N	S	W
4	N	Y	H	D	Z	F	Y	J	P
2	P	N	Z	F	Q	G	D	Y	X
5	R	S	W	S	S	N	N	B	X

> The key is `[3, 1, 4, 2, 5]`, so the rows it chooses the rows in this order and merges the result into a string.

The rows are extraccted in the order of the key and the result is "WWJYAWNSWPNZFQGDYXJFXNIJQSTNYHDZFYJPRSWSSNNBXX". This is the completed cipher text.

Decryption

Does this process in reverse. It takes the ciphertext and puts it back into the transposed matrix. Since the rows have been shuffled, we have to unshuffle. This means looking at the key and putting, for instance, row 3 into row 1, row 2 into row 4 and so on. `decrypt()` does this by creating a temporary matrix, which it just fill up in the correct order. Afterwards, the matrix is trasposed back and extracted row by row. This returns us with "JWNPRFWYNSXJHZWNYDFSIAZQSJWFGNQNYDNSSJYBTWPXX" which is the same as the result of the `caesar()` function, with the exception of an additional padding character at the end. To reverse the substitution the key passed to `caesar()` is just the key subtracted from the total number of values in the used alphabet

$$decryption\_key = len(alphabet) - key = 26 - 5 = 21$$

After applying the substitution again with the new key. The result is "ERIKMARTINSECURITYANDVULNERABILITYINNETWORKSS". The additional "S" at the end is the result of the padding. To be able to remove the padding at the end, a special padding character can be used. In systems where the result has to be exact and cannot be manually removed, any repeating character that usually isn't in repeating sequences like "X" can be used, but has to be handled by the authentication system.

2.2. Evaluate the Avalanche Effect

The avalanch effect is the percentage of bits that change in the output for making a small change in the input. For instance encrypting two versions of the, mostly, same plaintext:

Text	Cipher
`ERIKMARTINSECURITYANDVULNERABILITYINNETWORKS`	`WWJYAWNSWPNZFQGDYXJFXNIJQSTNYHDZFYJPRSWSSNNBX`
`RRIKMARTINSECURITYANDVULNERABILITYINNETWORKS`	`WWJYAWNSWPNZFQGDYXWFXNIJQSTNYHDZFYJPRSWSSNNBX`

> The difference is character 19: `J -> W`

The difference is difficult to find since they only have 1 character difference. This shows a negligable avalanch effect of $\approx$ 2%, since the number of differing bits is

$$Avalanch\ effect = \frac{num\ changed\ bits}{total\ number\ of\ bits} * 100 = \frac{1}{45} * 100 \approx 2.22\%$$

2 percent is just one character difference, meaning that there is no avalanch effect. A change of one character doesn't cascade and change more bits. It just changes that one letter of the in the cipher text.

2.3. Analyze the Avalanche effect

2.4. Optimize Avalanche Effect with Reasonable Computation

2.5. Enhance Security with Block Ciphers

3. Discussion

Discuss the analysis of your results.

4. Conclusion

A short paragraph that restates the objective from your introduction and relates it to your results and discussion and describe any future improvements in your techniques that you would recommend.

References

1. Substitution types: https://en.wikipedia.org/wiki/Substitution_cipher#Types
2. Polyalphabetic substitution :https://en.wikipedia.org/wiki/Substitution_cipher#Polyalphabetic
3. Mechanical substitution:https://en.wikipedia.org/wiki/Substitution_cipher#Mechanical
4. Columnar transposition:http://practicalcryptography.com/ciphers/columnar-transposition-cipher/#example
5. Transposition Cipher:https://en.wikipedia.org/wiki/Transposition_cipher#General_principle
6. Strict Avalanch Criterion:https://en.wikipedia.org/wiki/Avalanche_effect#Strict_avalanche_criterion
7. Bit Independence criterion:https://en.wikipedia.org/wiki/Avalanche_effect#Bit_independence_criterion
8. Confusion and Diffusion:https://en.wikipedia.org/wiki/Confusion_and_diffusion

9. 3 Classical:https://stavanger.instructure.com/courses/14643/files/folder/Lecture%20Slides?preview=1722525