**Security and Vulnerability in Networks**

University of Stavanger

# DAT 510: Assignment 1

# Encryption Analysis and Optimization: Understanding Avalanche Effect and Enhancing Security with Block Ciphers.

**Author**: Daniel Linfon Ye Liu

**Year**: 2024

# Index

# Abstract

In this assignment, we evaluated the avalanche effect across various encryption combinations to enhance data security. We tested four different encryption schemes: Caesar - Transposition, Caesar - Transposition - Vigenère Autokey, Caesar - Transposition - AES in CTR, and Caesar - Transposition - Vigenère Autokey - AES in CTR. Our procedure involved modifying a single character in the plaintext and analyzing the resulting avalanche effect after multiple encryption rounds. Results demonstrated that while simpler schemes like Caesar - Transposition yielded low diffusion, more complex combinations such as Caesar - Transposition - Vigenère Autokey - AES in CTR produced near-perfect avalanche effects, reaching up to 100% diffusion. As encryption complexity increased, the processing time also increased significantly. Our analysis highlights that combining block ciphers, strengthens the security and mitigates weaknesses inherent in transposition and substitution methods. Future improvements may focus on optimizing processing efficiency while maintaining high diffusion.

# 1. Introduction

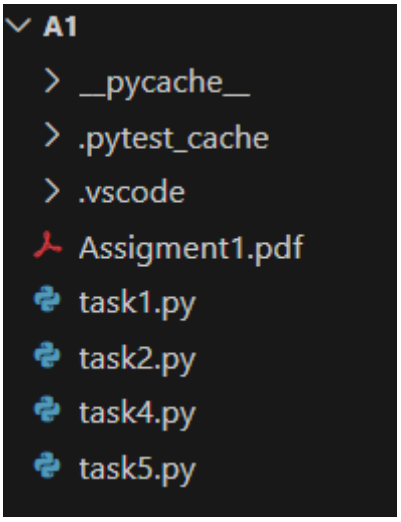
In this project, we will explore classical cryptographic techniques by implementing and analyzing transposition and substitution ciphers. The main goal is to apply these methods to a plaintext, composed of the student's name and the course name, using a custom numeric key derived from the student's phone number.

The project focuses on two possible encryption sequences: transposition followed by substitution or substitution followed by transposition. I have chosen the second one and it will be evaluated to determine security in terms of the Avalanche Effect, which measures how small changes in the plaintext affect the ciphertext. Furthermore, the encryption process will be optimized by balancing the Avalanche Effect with computational efficiency and the optimal number of encryption rounds will be identified to improve the security and efficiency of the system.

Finally, to strengthen the security of the chosen cipher combination, we will implement a block cipher operation mode, in my case the Counter (CTR). The effectiveness of these modes in mitigating the weaknesses inherent in transposition and substitution ciphers will be analyzed, providing a comprehensive view of their performance and security.

# 2. Design and Implementation

I have decided to create a separate script for each task (except 2 and 3 which are linked) using **Python** in **Visual Studio Code**. Each task builds on the previous one, enhancing the complexity and introducing advanced encryption techniques. Despite this, I think this organization is more suitable for my project to specifically evaluate each section.




***working directory***

## 2.1. Apply Encryption

In this initial task, I implemented basic cryptographic algorithms, **Caesar Cipher** and **Transposition Cipher**. I have opted for **Option B**, which consists of first applying the substitution cipher to the original text and then the transposition cipher to the resulting text. This encryption sequence offers several advantages in terms of security:

- Each letter of the original text is replaced by another, which eliminates **letter frequency patterns** and makes it difficult for an attacker to deduce the encryption key or the original content of the text.
- After the substitution cipher has been applied, the transposition cipher applies an **additional layer of security** by scrambling any patterns that may have been left behind after the substitution, making cryptographic analysis even more difficult.

## Implementation

Plain text: **Daniel Linfon Security and Vulnerability in Networks**
Phone number: **34036**
Key for Substitution: **6**
Key for Transposition: **24135**

```python
def caesar_cipher(text, shift):
    """
    Applies Caesar cipher to the text with a specified shift.

    Parameter:
    - text: The text to be encrypted.
    - shift: The number of positions to shift in the alphabet.

    Returns:
    - The encrypted text using Caesar cipher.
    """
    encrypted_text = []

    for char in text:
        shift_base = ord('A') if char.isupper() else ord('a')
        encrypted_text.append(chr((ord(char) - shift_base + shift) % 26 + shift_base))

    return ''.join(encrypted_text)
```

***task1.py - Caesar cipher code***

To perform the substitution (Caesar Cipher Algorithm), each letter in the text will be shifted 6 positions forward in the alphabet. The resulting text is:

**JGTOKRROTLUTYKIAXOZEGTJBARTKXGHOROZEOTTKZCUXQY**

```python
def transposition_cipher(text, key):
    """
    Performs a row transposition cipher based on the provided key.

    Parameter:
    - text: The text to be encrypted.
    - key: The transposition key used to rearrange the text.

    Returns:
    - The encrypted text after applying the transposition.
    """

    num_cols = len(key)
    columns = {}

    for i in range(1, num_cols + 1): # Every column
        words = ''
        for j in range(1, len(text)+1): # Every char
            if j % num_cols == i or (j % num_cols == 0 and i == num_cols):
                words += text[j-1]

        columns[i] = words

    columns_copy = {}
    ciphertext = ""

    for pos,digit in enumerate(key):
        digit = int(digit)
        columns_copy[digit] = columns[pos+1]

    for num in range(1,len(key)+1):
        ciphertext += columns_copy[num]

    return ciphertext
```

***task1.py - Transposition cipher code***

To make the transposition, first, we split the text into blocks the same size as the key length, in this case, 5.

**Key —> 24135**

**JGTOK**
**RROTL**
**UTYKI**
**AXOZE**
**GTJBA**
**RTKXG**
**HOROZ**
**EOTTK**
**ZCUXQ**
**Y**

**According to the key**
Column 1: **TOYOJKRTU**
Column 2: **JRUAGRHEZY**
Column 3: **OTKZBXOTX**
Column 4: **GRTXTTOOC**
Column 5: **KLIEAGZKQ**

If we concatenate the columns, the resulting encrypted message is:

```
TOYOJKRTUJRUAGRHEZYOTKZBXOTXGRTXTTOOCKLIEAGZKQ
```

```
PS C:\Users\danie\Desktop\Ing. Informatica\CUARTO\DAT510_SVN\A1> python .\task1.py

*** Task 1: Apply Encryption ***
Introduce the plain text: Daniel Linfon Security and Vulnerability in Networks
Introduce the last five digits of your phone number: 34036
Plain text: Daniel Linfon Security and Vulnerability in Networks
Caesar key: 6
Transposition key: 24135
After cleaning the text: DANIELLINFONSECURITYANDVULNERABILITYINNETWORKS
Caesar cipher text: JGTOKRROTLUTYKIAXOZEGTJBARTKXGHOROZEOTTKZCUXQY
Encrypted message: TOYOJKRTUJRUAGRHEZYOTKZBXOTXGRTXTTOOCKLIEAGZKQ
PS C:\Users\danie\Desktop\Ing. Informatica\CUARTO\DAT510_SVN\A1>
```

***task1.py - Main program***

## 2.2. Evaluate the Avalanche Effect

In this experiment, we evaluated the **Avalanche Effect** of a cryptographic system by introducing a minor change in the input text and re-encrypting the ciphertext for multiple rounds. The Avalanche Effect is the phenomenon where a small change in the input (such as flipping a single bit) produces a significant and unpredictable change in the output, particularly in cryptographic algorithms. The goal is to observe how even a small modification in the input affects the ciphertext across multiple encryption rounds.

### Step 1: Initial Avalanche Effect Analysis

We are going to use the same text as before and we are going to change the letter S for security:

```
Enter the same text changing only one character
Introduce the original text:  Daniel Linfon Security and Vulnerability in networks
Introduce the modified text:  Daniel Linfon Cecurity and Vulnerability in networks
Introduce the last five digits of your phone number: 34036

Original test:  Daniel Linfon Security and Vulnerability in networks
Modified test:  Daniel Linfon Cecurity and Vulnerability in networks

Caesar cipher original text: JGTOKRROTLUTYKIAXOZEGTJBARTKXGHOROZEOTTKZCUXQY
Caesar cipher modified text: JGTOKRROTLUTIKIAXOZEGTJBARTKXGHOROZEOTTKZCUXQY

Encrypted original message: TOYOJKRTUJRUAGRHEZYOTKZBXOTXGRTXTTOOCKLIEAGZKQ
Encrypted modified message: TOIOJKRTUJRUAGRHEZYOTKZBXOTXGRTXTTOOCKLIEAGZKQ

The percentage of differing bits is: 2.174%
```

***task2.py - Comparison***

A value of **2.714%** is relatively low, suggesting that the Caesar cipher - Transposition does not produce a significant avalanche effect in this case. This means that a small change does not produce a major change in the ciphertext, when the ideal would be a much larger avalanche effect to ensure the confidentiality of the information.

According to the Strict Avalanche Criterion (SAC), even a small change in the plaintext is expected to cause a change in about **50%** of the ciphertext bits. The Caesar cipher is one of the simplest and least secure ciphers due to its low avalanche effect and vulnerability to attacks.

## Step 2: Repeated Avalanche Effect Analysis

In order to be able to perform the avalanche effect multiple times, I have designed the `evaluate_avalanche_effect` function:

```
results = []
previous_text = original_text
current_text = modified_text

# Track the start time
start_time = time.time()

for round_num in range(1, rounds + 1):
    # Encrypt the text Caesar - Transposition cipher
    encrypted_prev_text = encryption(previous_text, transposition_key, caesar_key)
    encrypted_current_text = encryption(current_text, transposition_key, caesar_key)

    # Calculate avalanche effect
    avalanche_effect = calculate_bit_difference(encrypted_prev_text, encrypted_current_text)

    # Store results
    elapsed_time = time.time() - start_time
    results.append((round_num, avalanche_effect, elapsed_time))

    # Prepare for next round
    previous_text = encrypted_prev_text
    current_text = encrypted_current_text

print("\nRepeated Avalanche Effect Analysis Results:")
for round_num, avalanche_effect, elapsed_time in results:
    print(f"Round {round_num}: Avalanche Effect = {avalanche_effect:.3f}% | Time Elapsed = {elapsed_time:.4f} seconds")
```

***task2.py - `evaluate_avalanche_effect` function***

I have repeated the encryption up to 16 times and by evaluating the avalanche effect between the previous and the current one I have obtained the following results:

```
*** Task 2: Analyze the Avalanche effect ***
Introduce the original text: Daniel Linfon Security and Vulnerability in Networks
Introduce the modified text: Daniel Linfon Cecurity and Vulnerability in Networks
Introduce the last five digits of your phone number: 34036

Repeated Avalanche Effect Analysis Results:
Round 1: Avalanche Effect = 2.174% | Time Elapsed = 0.0000 seconds
Round 2: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 3: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 4: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 5: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 6: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 7: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 8: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 9: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 10: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 11: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 12: Avalanche Effect = 2.174% | Time Elapsed = 0.0010 seconds
Round 13: Avalanche Effect = 2.174% | Time Elapsed = 0.0020 seconds
Round 14: Avalanche Effect = 2.174% | Time Elapsed = 0.0020 seconds
Round 15: Avalanche Effect = 2.174% | Time Elapsed = 0.0020 seconds
Round 16: Avalanche Effect = 2.174% | Time Elapsed = 0.0020 seconds
```

***task2.py - Main Program***

## 2.3. Analyze the Avalanche effect

The results of the avalanche analysis show that the percentage of difference between the ciphertexts is **constant** in all rounds (**2.174%**). We can interpret that:

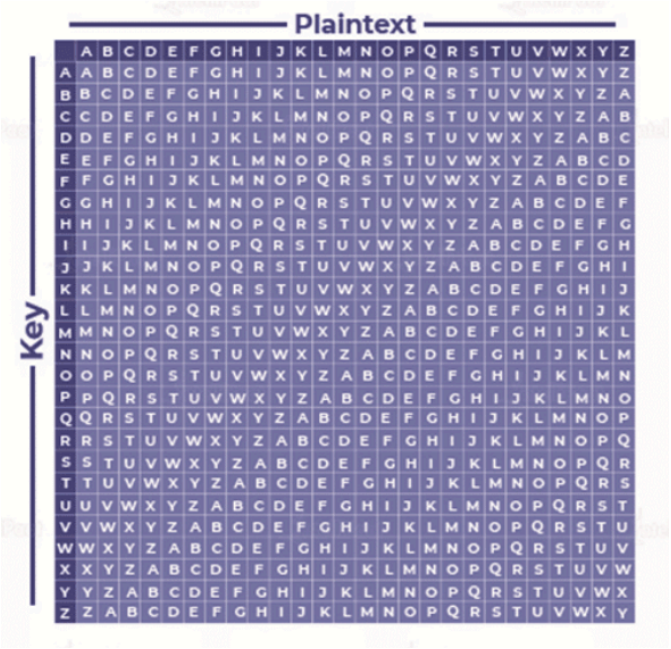
- The Caesar cipher followed by transposition is relatively **simple** compared to modern ciphers. The Caesar cipher is linear and predictable, and the transposition simply rearranges the characters without changing the amount of information being represented. Both mechanisms operate independently and do not reinforce each other to create a stronger effect. This can lead to a less variable avalanche effect.
- Applying the same cipher repeatedly on the ciphertext may not significantly change the content of the ciphertext in each round. They follow the same pattern.
- The computation time per round was stable, with minor variations as the rounds progressed. This reflects the efficiency of the scheme in terms of speed, but does not compensate for the lack of diffusion of change.

The low and consistent Avalanche Effect observed shows that the Caesar Cipher-transposition encryption scheme lacks the ability to adequately diffuse small changes in the input. More robust encryption would require more complex algorithms that guarantee a higher degree of randomness and diffusion. We can deduce that the current scheme is vulnerable to cryptographic attacks, due to its low sensitivity to small changes in the original text.

## 2.4. Optimize Avalanche Effect with Reasonable Computation

After analyzing the results of the initial encryption using a combination of Caesar Cipher and transposition, it was observed that the avalanche effect was not strong enough. That is why I have introduced the **Vigenère Autokey System.**

Unlike the Caesar Cipher, where the key is static and the character offset is fixed, the Vigenère Autokey System uses an initial key that expands dynamically with the text itself, causing a small change in one part of the text to propagate throughout the entire encryption process. This results in a greater dispersion of changes and therefore a more robust encryption.




*Vigenere Cipher Table*

I have introduced the `vigenere_autokey_encrypt` function in my program:

```python
def vigenere_autokey_encrypt(plaintext, key):

    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    key = key.upper()
    plaintext = plaintext.upper()

    # Extend the key with the original text
    extended_key = key + plaintext
    extended_key = extended_key[:len(plaintext)]  # The length of the extended key is equal to that of the text

    ciphertext = []

    for i, char in enumerate(plaintext):
        # Get the index of the current letter in the plaintext and the key
        text_index = alphabet.index(char)
        key_index = alphabet.index(extended_key[i])

        # Update alphabet according to key_index
        alphabet_i = alphabet[key_index:] + alphabet

        # Find the new cipher char
        ciphertext.append(alphabet_i[text_index])

    return ''.join(ciphertext)
```

***task4.py - Vigenere Autokey System code***

Therefore, encryption now consists of:

1. Caesar cipher.
2. Row Transposition Cipher.
3. Vigenère Autokey System.

Evaluating the avalanche effect with this system, we obtain the following results:

Plain text: **Daniel Linfon Security and Vulnerability in Networks**
Modified text: **Daniel Linfon Cecurity and Vulnerability in Networks**
Phone number: **34036**
Key for Caesar cipher: **6**
Key for Transposition: **24135**
Key for Autokey system: **avalanche**

```
Task 4: Optimize Avalanche Effect
Introduce the original text: Daniel Linfon Security and Vulnerability in Networks
Introduce the modified text: Daniel Linfon Cecurity and Vulnerability in Networks
Introduce the last five digits of your phone number: 34036
Introduce the key for Vigenère Autokey System: avalanche

Repeated Avalanche Effect Analysis Results:
Round 1: Avalanche Effect = 4.348% | Time Elapsed = 0.0000 seconds
Round 2: Avalanche Effect = 8.696% | Time Elapsed = 0.0010 seconds
Round 3: Avalanche Effect = 13.043% | Time Elapsed = 0.0010 seconds
Round 4: Avalanche Effect = 19.565% | Time Elapsed = 0.0010 seconds
Round 5: Avalanche Effect = 39.130% | Time Elapsed = 0.0010 seconds
Round 6: Avalanche Effect = 60.870% | Time Elapsed = 0.0010 seconds
Round 8: Avalanche Effect = 89.130% | Time Elapsed = 0.0020 seconds
Round 9: Avalanche Effect = 97.826% | Time Elapsed = 0.0020 seconds
Round 10: Avalanche Effect = 93.478% | Time Elapsed = 0.0020 seconds
Round 11: Avalanche Effect = 93.478% | Time Elapsed = 0.0020 seconds
Round 12: Avalanche Effect = 89.130% | Time Elapsed = 0.0020 seconds
Round 13: Avalanche Effect = 89.130% | Time Elapsed = 0.0030 seconds
Round 14: Avalanche Effect = 95.652% | Time Elapsed = 0.0030 seconds
Round 15: Avalanche Effect = 100.000% | Time Elapsed = 0.0030 seconds
Round 16: Avalanche Effect = 93.478% | Time Elapsed = 0.0030 seconds
```

***task4.py - Main Program***

### Avalanche Effect Growth:

- In the first rounds (1 to 4), the avalanche effect grows **moderately**, reaching 19.565% in the fourth round.
- Beginning with the fifth round (39.130%), the avalanche effect experiences a **more pronounced increase**, reaching 60.870% in the sixth round and 97.826% in the ninth round. This rapid increase indicates that the encryption is causing a greater diffusion of changes in the original text.
- Beginning with the ninth round, the avalanche effect stabilizes around **97.826%**, indicating that most bits have been altered in response to the initial modification in the plaintext.

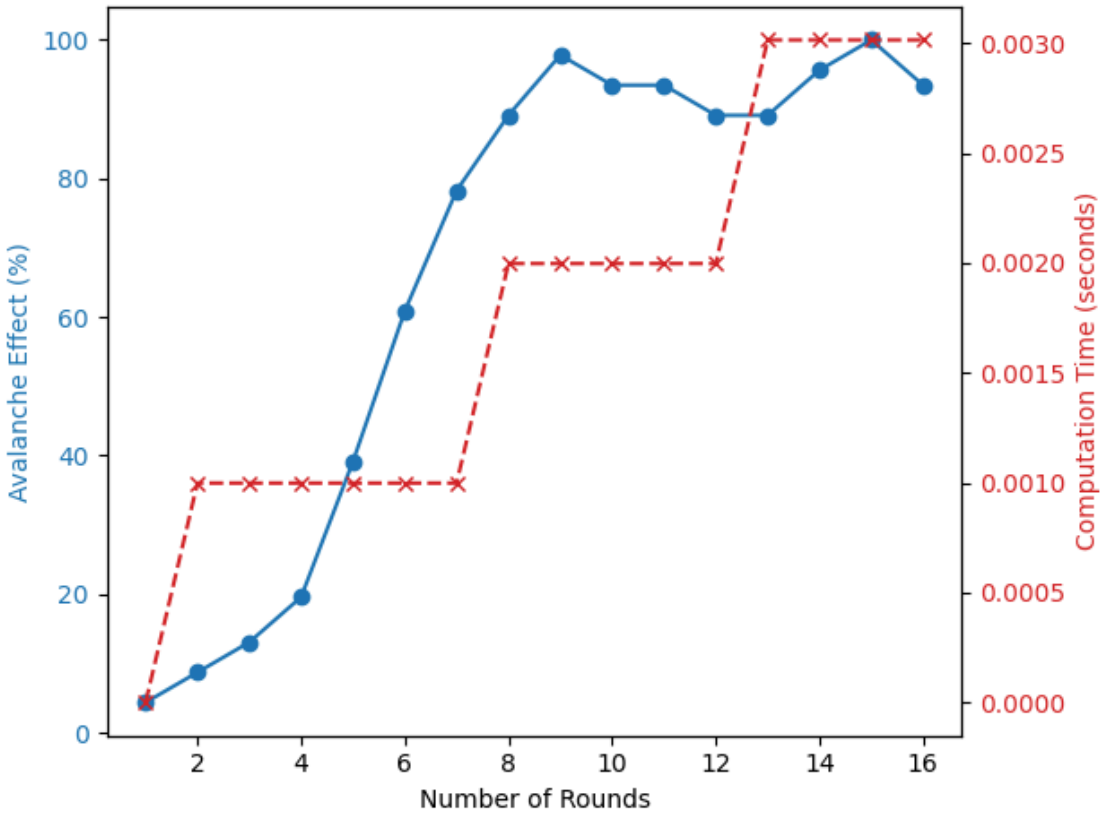
### Computational Efficiency:

The computation time remains fairly stable, starting with 0.0010 seconds in the early rounds and increasing slightly to 0.0020 - 0.0030 seconds in the later rounds. Despite this slight increase in time, the computational efficiency remains reasonable. The variations in time are not drastic, suggesting that the algorithm is efficient enough to handle multiple rounds of encryption without significant computational cost.

### Optimal Point:

The avalanche effect reaches a near-maximum level of **97.826% in the ninth round**, and this is the point where the best balance between a high avalanche effect and a low computational cost is observed.

*Avalanche Effect and Computation Time vs. Number of Rounds*




Between rounds 10 and 16, some fluctuation in the avalanche effect is observed. Although in round 15 it reaches 100%, in other rounds such as 10 and 11, the effect decreases to 93.478%, suggesting that some bits stabilize or return to similar values.

These fluctuations may be due to the structure of the Vigenère Autokey System, where the key is extended with the plaintext itself. Since each round generates a new ciphertext, the self-generated key introduces variations that may cause the effect to stabilize at some points.

Despite these fluctuations, the overall stability indicates that the encryption system has reached a high degree of complexity and security, where a small change in the original text causes a significant alteration in the encrypted output.

## 2.5. Enhance Security with Block Ciphers

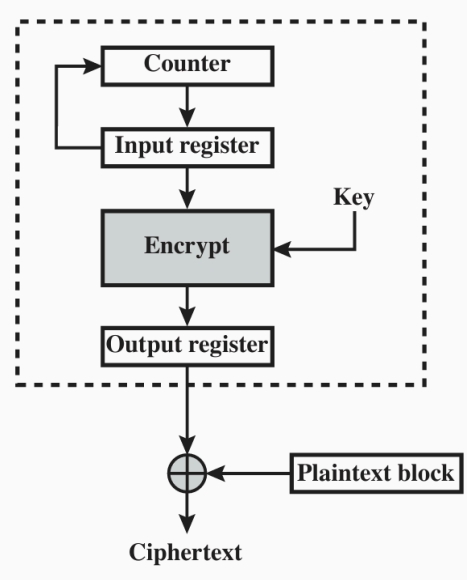
****NOTE: In this assignment I used the AES algorithm, in addition to the Caesar and transposition combination (and Vigenere) since when researching the CTR mode, it was the most generally used. I understood that the plaintext would be to use our combination and then apply the block operation, even if it involved using another encryption algorithm**

**However, today Friday 09/13/2024 I went to class for the review and found out that it was not necessary to use said algorithm. Since the delivery date is today, I have not had enough time to change the project. Therefore, the analysis of this section has been using AES. Sorry for the misunderstanding.****

As we have previously analyzed, both transposition and substitution ciphers, while simple and easy to implement, suffer from weaknesses such as susceptibility to pattern recognition, lack of diffusion, and vulnerability to frequency analysis. Therefore, an encryption combining both algorithms does not provide good results since they do not enrich each other.

To address these limitations and improve the security of the combination of transposition and substitution ciphers, we will implement a block cipher mode of operation. These modes will add an additional level of complexity and security to the encryption process, helping to mitigate problems such as pattern repetition and lack of diffusion.

I have opted for **CTR (Counter Mode)**. Instead of encrypting data blocks directly, it uses a counter that is combined with a unique seed value (nonce) to generate a key stream, similar to stream ciphers.



The process starts with a nonce (number used once) and a counter. The nonce is a random or unique value that ensures that even if the same message is encrypted twice with the same key, the ciphertext will be different. The counter is initialized to a value and incremented for each encrypted block.

For each block of data to be encrypted, the nonce and counter are combined and encrypted using the symmetric encryption algorithm **(AES)**. This produces a sequence of bits (keystream) that is used to encrypt the data block. The counter is incremented with each block, ensuring that each block is encrypted with a different key.

Finally, the plaintext, which has been previously encrypted with our combination of substitution and transposition, is encrypted using a bitwise XOR operation between the generated keystream and the plaintext block.

To implement it in our code, the following function has been created:

```python
def encrypt_CTR(plaintext, key):
    # Generates a random 16-byte nonce
    nonce = os.urandom(16)

    # Creates an encryption object using the AES algorithm in CTR mode
    cipher = Cipher(algorithms.AES(key), modes.CTR(nonce), backend=default_backend())
    encryptor = cipher.encryptor()

    # Encrypt the plaintext and get the ciphertext
    ciphertext = encryptor.update(plaintext.encode('utf-8')) + encryptor.finalize()

    return nonce, ciphertext
```

***task5.py - CTR encryption***

The **cryptography** library has been used to implement the CTR.

## Avalanche effect

To measure the avalanche effect with the CTR, we have used two combinations for encryption:

- Caesar - Transposition - AES in CTR
- Caesar - Transposition - Vigenere Autokey System - AES in CTR

## Caesar - Transposition - AES in CTR mode

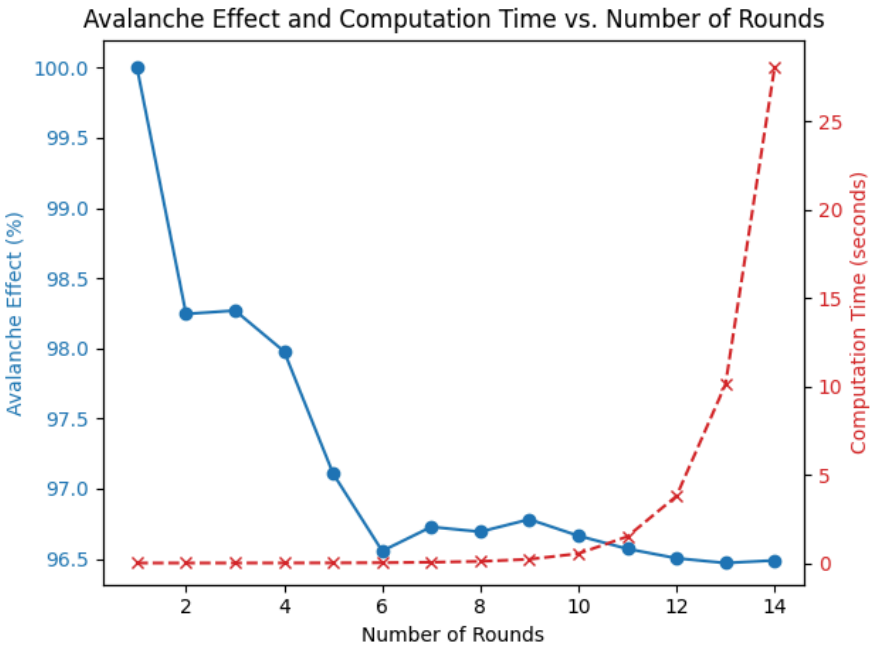
```
*** Task 5: Enhance Security with Block Ciphers ***
Introduce the original text: Daniel Linfon Security and Vulnerability in Networks
Introduce the modified text: Daniel Linfon Cecurity and Vulnerability in Networks
Introduce the last five digits of your phone number: 34036
Do you want to add vigenere autokey system to encryption? (y/n): n

Repeated Avalanche Effect Analysis Results:
Round 1: Avalanche Effect = 100.000% | Time Elapsed = 0.0395 seconds
Round 2: Avalanche Effect = 98.246% | Time Elapsed = 0.0395 seconds
Round 3: Avalanche Effect = 98.270% | Time Elapsed = 0.0405 seconds
Round 4: Avalanche Effect = 97.978% | Time Elapsed = 0.0425 seconds
Round 5: Avalanche Effect = 97.107% | Time Elapsed = 0.0465 seconds
Round 6: Avalanche Effect = 96.557% | Time Elapsed = 0.0574 seconds
Round 7: Avalanche Effect = 96.728% | Time Elapsed = 0.0824 seconds
Round 8: Avalanche Effect = 96.694% | Time Elapsed = 0.1312 seconds
Round 9: Avalanche Effect = 96.782% | Time Elapsed = 0.2528 seconds
Round 10: Avalanche Effect = 96.665% | Time Elapsed = 0.5688 seconds
Round 11: Avalanche Effect = 96.572% | Time Elapsed = 1.5364 seconds
Round 12: Avalanche Effect = 96.505% | Time Elapsed = 3.8247 seconds
Round 13: Avalanche Effect = 96.471% | Time Elapsed = 10.1464 seconds
Round 14: Avalanche Effect = 96.490% | Time Elapsed = 28.0476 seconds
```

***task5.py - Avalanche effect without Vigenere***

CTR mode used in conjunction with substitution and transposition ciphers has proven to be highly effective in achieving a strong avalanche effect. The 100% achieved in the first round indicates that even a small change in the plaintext immediately affects all bits in the ciphertext.

As the rounds progress, there is a slight decrease in the avalanche effect. This may be because after multiple iterations, the initial differences have already been completely absorbed and further rounds generate only minor variations in the bit structure. However, staying above 96% across all rounds is still an indicator of robust encryption.




Regarding processing time, it has been affected. The times recorded for each round show a notable growth in processing duration, especially from round 10 onwards. This increase is a direct consequence of the iterative nature of multi-round encryption, where successive ciphertexts become progressively more complex. Transposition and substitution operations, together with CTR mode encryption, contribute to the time increase in each round, especially as the amount of data and its complexity grow.

## Caesar - Transposition - Vigenere Autokey System - AES in CTR mode

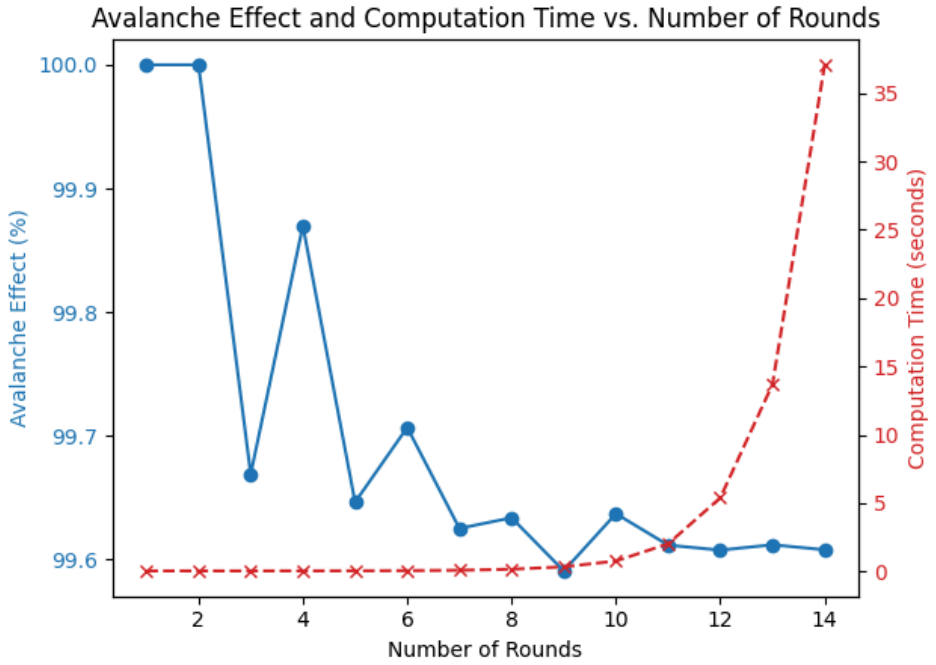
```
*** Task 5: Enhance Security with Block Ciphers ***
Introduce the original text: Daniel Linfon Security and Vulnerability in Networks
Introduce the modified text: Daniel Linfon Cecurity and Vulnerability in Networks
Introduce the last five digits of your phone number: 34036
Do you want to add vigenere autokey system to encryption? (y/n): y
Introduce the key for Vigenère Autokey System: avalanche

Repeated Avalanche Effect Analysis Results:
Round 1: Avalanche Effect = 100.000% | Time Elapsed = 0.0399 seconds
Round 2: Avalanche Effect = 100.000% | Time Elapsed = 0.0399 seconds
Round 3: Avalanche Effect = 99.669% | Time Elapsed = 0.0409 seconds
Round 4: Avalanche Effect = 99.870% | Time Elapsed = 0.0429 seconds
Round 5: Avalanche Effect = 99.646% | Time Elapsed = 0.0469 seconds
Round 6: Avalanche Effect = 99.707% | Time Elapsed = 0.0589 seconds
Round 7: Avalanche Effect = 99.625% | Time Elapsed = 0.0879 seconds
Round 8: Avalanche Effect = 99.634% | Time Elapsed = 0.1639 seconds
Round 9: Avalanche Effect = 99.591% | Time Elapsed = 0.3393 seconds
Round 10: Avalanche Effect = 99.637% | Time Elapsed = 0.7640 seconds
Round 11: Avalanche Effect = 99.612% | Time Elapsed = 2.0200 seconds
Round 12: Avalanche Effect = 99.607% | Time Elapsed = 5.3874 seconds
Round 13: Avalanche Effect = 99.612% | Time Elapsed = 13.6779 seconds
Round 14: Avalanche Effect = 99.608% | Time Elapsed = 37.0615 seconds
```

***task5.py - Avalanche effect with Vigenere***

The avalanche effect was then evaluated using a combination of Caesar, Transposition, Vigenère ciphers with Autokey and the CTR mode of operation (with AES). As we can see, the avalanche effect is complete, reaching almost 100%. This means that the encryption system responds extremely sensitive to small modifications in the text, which is ideal from a security perspective.

The addition of the Vigenère Autokey cipher seems to help maintain a high avalanche effect, with very stable results in almost all rounds. This encryption method introduces a key dependent on the original text itself, making it more difficult to predict the structure of the cipher.




Processing time shows a similar incremental trend to that observed in previous analyses, although with a slightly larger increase due to the incorporation of Vigenère Autokey encryption, which adds an extra layer of calculation each round.

## Comparisons and conclusions

Having evaluated both combinations, we can draw some conclusions:

- **Avalanche Effect:** Both schemes show an excellent avalanche effect, with values remaining high in all rounds, although there is a slight decrease in later rounds. The inclusion of Vigenère Autokey improves the avalanche effect compared to using only Caesar and transposition cipher, making it almost perfect.

- **Processing Time:** The processing time increases significantly in both cases as the rounds progress, but the scheme including Vigenère Autokey shows a more pronounced increase. This may be due to the additional complexity of the Vigenère cipher, which requires additional calculations in each round.
- **General Security:** The combination of Caesar - Transposition - Vigenère Autokey - AES in CTR offers superior security to the system without Vigenère encryption, thanks to the additional variability in the encryption key. However, the cost in terms of processing time is higher, which must be considered in practical applications.

In summary, Caesar - Transposition - Vigenère Autokey - AES in CTR provides an improvement in overall security by introducing more complex encryption, but at the expense of increased processing time. On the other hand, Caesar - Transposition - AES in CTR offers a good balance between security and efficiency, being more suitable in situations where performance is crucial.

CTR block cipher mode helps mitigate the inherent weaknesses of transposition and substitution ciphers by providing increased confusion, diffusion, and resistance to statistical attacks. By transforming the plaintext into a pseudo-random cipher stream and combining it with the plaintext using an XOR operation, CTR mode significantly improves the overall security of the cipher and addresses many of the weaknesses associated with older encryption methods.

# 3. Discussion

After testing several combinations of encryption techniques, we have obtained a series of results that reflect the impact of each approach on the avalanche effect and processing time. A detailed analysis of each combination and its implications is presented below.

## 1. Caesar - Transposition

- **Avalanche Effect:** The avalanche effect remained low, with a constant value of 2.174% in each round. This result indicates that a small change in the plaintext does not produce a significant alteration in the ciphertext, which limits the ability of the cipher to effectively spread changes.
- **Processing Time:** The processing time was very low, ranging from 0.0010 to 0.0020 seconds per round. This suggests that the cipher is very time-efficient, but at the expense of lower security.

**Conclusion**: Although this combination is time-efficient, its poor avalanche effect indicates a limited ability to hide changes in the plaintext, which may make it less secure.

## 2. Caesar - Transposition - Vigenère Autokey

- **Avalanche Effect:** In the first rounds, the avalanche effect grew moderately, reaching 19.565% in the fourth round and 60.870% in the eighth. From the fifth round onwards, the avalanche effect experienced a more pronounced increase, reaching up to 97.826% in the ninth round. This suggests that encryption with Vigenère Autokey significantly improves the diffusion of changes in the ciphertext.
- **Processing Time:** The processing time remained stable, starting at 0.0010 seconds and increasing slightly to 0.0020 - 0.0030 seconds in later rounds. The increase in time is minimal compared to the avalanche effect.

**Conclusion**: The addition of Vigenère Autokey to the Caesar - Transposition combination considerably improves the avalanche effect, making the encryption more secure. The increase in processing time is slight, suggesting a good balance between security and efficiency.

## 3. Caesar - Transposition - AES in CTR

- **Avalanche Effect:** A high and consistent avalanche effect was observed, with values ranging from around 96% to 100,000%. This indicates that CTR mode provides strong diffusion of changes in the plaintext.
- **Processing Time:** The processing time increased significantly with each round, starting at 0.0395 seconds and reaching up to 28.0476 seconds in round 14. This increase is notable and reflects the impact of encryption on efficiency.

**Conclusion**: The Caesar - Transposition - CTR combination provides a very high avalanche effect, improving security compared to previous combinations. However, the processing time is considerably increased, which can be a concern in high-throughput applications.

## 4. Caesar - Transposition - Vigenère Autokey - AES in CTR

- **Avalanche Effect:** The avalanche effect remained very high and stable, starting at 100.000% in the first few rounds and stabilizing around 99.6% from the ninth round onwards. This indicates a robust ability to spread changes in the plaintext.
- **Processing Time:** The processing time started at 0.0399 seconds and progressively increased to reach 37.0615 seconds in round 14. This increase is significant and reflects the additional complexity of the encryption.

**Conclusion**: The Caesar - Transposition - Vigenère Autokey - AES in CTR combination provides outstanding security with an excellent avalanche effect. However, as with the previous combination, the increase in processing time is considerable, which may limit its applicability in systems where encryption time is critical.

# 4. Conclusion

After analyzing the four cipher combinations, the goal was to evaluate the avalanche effect and computational efficiency of each scheme, in search of increased security. The results show that **using block ciphers** like CTR, with or without Vigenère Autokey, **provides a much more pronounced avalanche effect** from the first rounds compared to simpler methods, such as Caesar and Transposition, which show a significantly lower avalanche effect.

In terms of processing times, the combinations with CTR showed a gradual increase in each round, suggesting increased computational complexity, but also increased security.
If we had to choose one, in terms of balance between security and computational efficiency, it seems to be **Caesar - Transposition - AES in CTR**. This combination achieves a high avalanche effect (over 99% from the first round) and although processing times increase gradually in each round, it is still reasonably efficient compared to other more complex combinations

It should be noted that Caesar - Transposition - Vigenère achieves a good avalanche effect in satisfactory processing time, however, to reach that point several rounds are needed.

For future improvements, one could consider optimizing key usage and exploring other block cipher operation modes. It would also be useful to experiment with longer text lengths and different data types to assess the robustness of the schemes under different usage scenarios.

# References


- Substitution and Transposition Ciphers: Lecture 03 (**”03 Classical”**)
- Avalanche Effect: Lecture 04 (**”04 Block Des”**).
- Avalanche effect - Wikipedia
- Vigenere Autokey System
- Block Cipher Modes of Operation: Lecture 07 (**”07 Block Ops”**)
- CTR:
    - https://xilinx.github.io/Vitis_Libraries/security/2020.1/guide_L1/internals/ctr.html
    - https://keepcoding.io/blog/modo-de-cifrado-ctr/ (in spanish)