



 slington college  
(इरिलिङ्टन कलेज)

**Module Code & Module Title**

**CC5009NI Cyber Security in Computing**

**Assessment Weightage & Type**

**40% Individual Coursework 01**

**Year and Semester**

**2024 - 25 Autumn Semester**

**Student Name: Navraj Rajak**

**London Met ID: 23047346**

**College ID: NP01NT4A230040**

**Assignment Due Date: Monday, January 20, 2025**

**Assignment Submission Date: Monday, January 20, 2025**





**Word Count: 10449**

*I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*




## 7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

### Match Groups

-  **49 Not Cited or Quoted 6%**  
Matches with neither in-text citation nor quotation marks
-  **7 Missing Quotations 1%**  
Matches that are still very similar to source material
-  **0 Missing Citation 0%**  
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**  
Matches with in-text citation present, but no quotation marks

### Top Sources

- 3%  Internet sources
- 1%  Publications
- 6%  Submitted works (Student Papers)

### Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

## Abstract

The increasing reliance on secure communication has emphasized the need for robust cryptographic techniques. The traditional encryption methods, such as the Caesar Cipher, has many vulnerabilities. These vulnerabilities make them ineffective against modern attacks like brute force, frequency analysis. The motivation behind this development is to develop a modified version of Caesar Cipher addressing its vulnerability for better security.

The aim of this report is to research, develop and test a new cryptographic system addressing the traditional Caesar Cipher's vulnerabilities and strengthen them through the introduction of dynamic shifting, an expanded character set, noise insertion, and Base64 encoding and XOR operation. This tends to improves its resistance to the brute force, frequency analysis. The development involves analyzing, modifying and testing the cipher to improve its security.

The newly developed algorithm, *CipherNova-47*, was tested to evaluate its effectiveness. The result demonstrates that the developed algorithm has better and improved security while reducing susceptibility to decryption without proper authorization.

## Table of Contents

Abstract.....	ii
Table of figures.....	v
Table of tables.....	vi
1. Introduction .....	1
1.1. Security.....	1
1.2. CIA.....	1
1.3. Cryptography and its History.....	2
1.4. Key Terminologies.....	4
1.5. Symmetric and Asymmetric Encryption Systems .....	5
2. Background.....	7
2.1. Caesar Cipher and its History .....	7
2.2. Working Mechanisms.....	7
2.3. Advantages.....	10
2.4. Disadvantages .....	11
3. Development.....	12
3.1. Introduction .....	12
3.2. Reasons for Modification.....	12
3.3. Background and Explanation for Modifications .....	13
3.4. Encryption Algorithm .....	16
3.5. Decryption Algorithm.....	23
3.6. Naming the Cryptographic Algorithm Developed .....	29
3.7. Flowchart for Encryption and Decryption algorithm.....	30
4. Testing.....	32
4.1. Test 1 .....	32

4.1.1.	Encryption.....	32
4.1.2.	Decryption.....	36
4.2.	Test 2 .....	39
4.2.1.	Encryption.....	39
4.2.2.	Decryption.....	42
4.3.	Test 3 .....	45
4.3.1.	Encryption.....	45
4.3.2.	Decryption.....	52
4.4.	Test 4 .....	58
4.4.1.	Encryption.....	58
4.4.2.	Decryption.....	62
4.5.	Test 5 .....	65
4.5.1.	Encryption.....	65
4.5.2.	Decryption.....	69
5.	Critical Evaluation of CipherNova-47 .....	74
5.1.	Strengths.....	74
5.2.	Weakness .....	75
5.3.	Application Areas .....	75
6.	Conclusion .....	77
	References.....	78

## Table of figures

Figure 1: CIA Triad .....	1
Figure 2: Egyptian Hieroglyphs .....	3
Figure 3: Enigma Machine .....	4
Figure 4: Symmetric Encryption .....	6
Figure 5: Asymmetric Encryption .....	6
Figure 6: Caesar Cipher .....	7
Figure 7: Flowchart for Encryption Algorithm .....	30
Figure 8: Flowchart for Decryption Algorithm .....	31

## Table of tables

Table 1: Caesar Cipher Table .....	8
Table 2: Extended Caesar Cipher Table (i) .....	13
Table 3: Extended Caesar Cipher Table (ii) .....	13
Table 4: Extended Caesar Cipher Table (iii) .....	14
Table 5: Extended Caesar Cipher Table .....	14

## 1. Introduction

### 1.1. Security

Security is the protection of information, systems, and physical assets from any possible threats, unauthorized access, damage or destruction (IBM, n.d.). In simple terms, it is the practice of keeping documents confidential, maintaining its integrity and availability so that the document can be only accessible by authorized individual, system or user. Security is vital in the realm of information technology. It refers to adopting and maintaining resilient measures against any sort of cyber threats and balancing it all together for efficient workflow. Its main goal is to establishing and maintaining a secure environment where data, information and system have adequate resilience against any sort of threats or disruption.

### 1.2. CIA

It is one of the core principles of information security. It stands for **confidentiality**, **integrity** and **availability**. It has laid foundational grounds for information security. It is a framework providing guidelines in order to create and maintain resilient information security posture within any organization. (Steinberg, 2022)



*Figure 1: CIA Triad*

Confidentiality ensures that the data is only accessible only by authorized users on need-to-know basis, no other way around. This can be maintained by the of mechanisms of



encryption, access control, authentication. For example: sensitive information such as account details and password protected through encryption and multi-factor authentications to ensure that only authorized individual can access the data or system.

Integrity ensures that the information is accurate and has not been altered throughout its state (transit or at-rest). It can be maintained by the adoption of mechanism like hash function, checksums, and so on. For example: information in transit or at rest remain accurate and unaltered uses mechanisms like hashing to verify the integrity of the information.

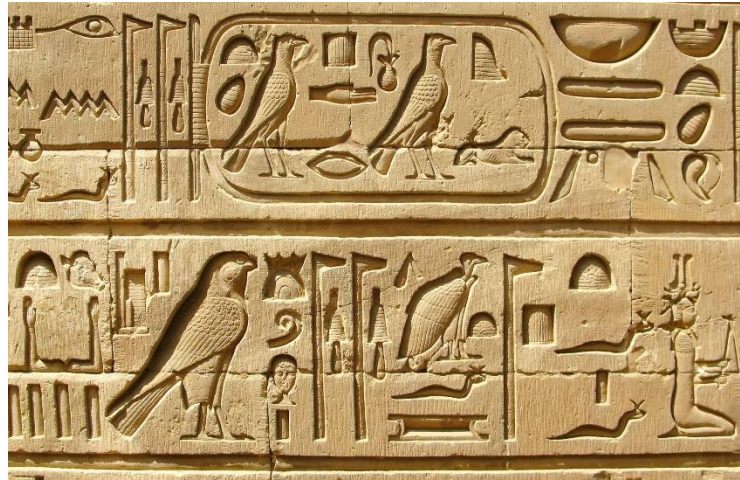
Availability refers that the data and the system can be accessed whenever needed by authorized personnel. It can be achieved by redundancy, backup and business continuity plan. For example, banking system ensure customers can access their accounts anytime through redundant servers, regular backups, and a business continuity plan. (Steinberg, 2022)

### 1.3. Cryptography and its History

It is the science and practice to secure information by transforming it into an unreadable format in order to ensure that only intended individuals or systems can read it (PGP Corporation, 2002). It is also the practice of transforming plaintext into ciphertext. It is achieved by the use of mathematical algorithms and protocols. Its primary goal is to achieve confidentiality (ensuring data is secret), integrity (ensuring data is unaltered) and authentication (ensuring only the authorize can read it) and non-repudiation (ensuring the action cannot be denied). The development of cryptographic methods has advanced over the years. The need for secure communication has been the driving force since early period of its development. This has proved to be viral in the digital age, where all the information is exchanged over an open communication channel, the internet. (Stallings, 2017)

The history of cryptography dates back to thousands of years. It has vastly evolved from simple ciphers to advanced encryption methods used today in modern communication. **Egyptian Hieroglyphs** in 2000 BCE was used in ancient Egypt where symbols and

secret codes were used for communication, more like encoded meaning in the writings. It is believed to be earliest form of cryptography (Murray, 1920). In 1<sup>st</sup> century BCE, Julius Caesar is believed to use simple cipher, called **Caesar Cipher**, where each letter of alphabet was shifted by three position to perform military communication securely. For example, A becomes D, B becomes E, and so on.



*Figure 2: Egyptian Hieroglyphs*

In the 5<sup>th</sup> century BCE, the Spartans used **Scytale Cipher**. It was used through a device called a scytale which could be represented by a rod around which a strip of parchment was wound and on unwound, the message was revealed. Leon Battista Alberti, in 1467, developed **Alberti Cipher**, a cipher disk tool for encrypting messages using shifting alphabet and thus laid foundation for more complex encryption methods to be developed in the future. Blaise de Vigenère in 1553 invented Vigenère Cipher, a polyalphabetic cipher that used a key to shift letters in alphabets which made it much more difficult to decode. (David Kahn, 1967)

An Arab scholar al-Kindi, around 9<sup>th</sup> century, developed frequency analysis which applied a method to crack substitution ciphers by analyzing the frequency of letters in the ciphertext. By 17<sup>th</sup> century Mathematicians began developing methods to break ciphers which marked the beginning of cryptanalysis as an organized field. During World War II, the Germans were using **Enigma Machine**, a complex system mechanism that required rotating rotors to encrypt message, but eventually cracked by Alan Turing and his team at Bletchley Park.



*Figure 3: Enigma Machine*

In 1970s, Whitfield Diffie and Martin Hellman developed **Public Key Cryptography** allowing two parties to share keys securely over insecure channels (Diffie & Hellman, 1976). By 1977, **RSA** Algorithm was co-developed by Ron Rivest, Adi Shamir, and Leonard Adelman, introducing public and private key pairs from encryption and decryption and thus this laid the foundation for modern secure communication (Rivest, 1978). **Elliptic Curve Cryptography** was developed in 1985 as an upgrade to RSA that offered higher security with relatively shorter keys (Miller, 1985). Finally, in the 1990s, there was emergence of **Quantum Cryptography**, utilizing the principles of quantum mechanics to provide unbreakable encryption methods (theoretically) ensuring secure communication in an increasingly digital world (Bennett & Brassard, 1984).

#### 1.4. Key Terminologies

There are some terminologies one must be familiar with in order to understand cryptography. These are as follows:

- a) **Plaintext**: This is the clear data or information existing in its ordinary form and without the process of decryption. It might refer to any kind of messages, documents, or any files which are straightforwardly accessible to the eyes of human beings. Plaintext is an input data before an encryption occurs. Examples of plaintext are simple text files, an unencrypted message, or password.

- b) **Ciphertext:** This is the encrypted version of the plaintext. It is hardly readable to human beings. It is the output derived by the mechanism of an encryption algorithm to the plaintext. That means, only authorized people can read it, and unauthorized people cannot read it.
- c) **Encryption:** It is the action of transformation of a readable plain text into a cipher text through the operation of applying given encryption algorithm and the key. Making this transformation provides confidentiality in the data, whereby it cannot be read by any individual without having a decryption key.
- d) **Decryption:** It is the process of decoding the ciphertext to its original text using a key; from the human-readable format to the earlier one, it restores encrypted data.
- e) **Key:** It is a sequence of characters used in a cryptographic algorithm to encrypt or decrypt data. In symmetric encryption, the same key is used to encrypt and decrypt. In asymmetric encryption, two keys are used: public key is used for encryption, while private key is used for decryption.

## 1.5. Symmetric and Asymmetric Encryption Systems

- a) **Symmetric Encryption:** In this, the same secret key is used for encryption and decryption. The sender and the receiver use the same key. For an example, the sender encrypts a brief message into ciphertext. While decrypting that message, the same key used in encryption should be used. Symmetric encryption is fast. It requires less computation power than asymmetric encryption. It shares the same key for encryption and decryption; thus, the distribution of the key must be shared securely. Advanced Encryption Standard (AES) and Data Encryption Standard are some common symmetric encryption algorithms.

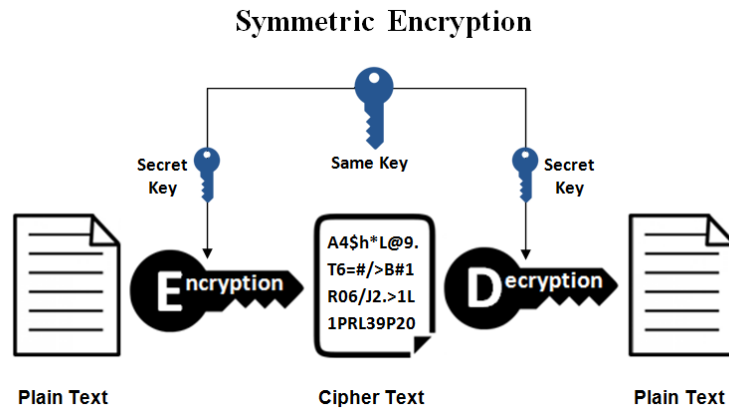


Figure 4: Symmetric Encryption

- b) Asymmetric Encryption:** Here, two different keys are used, a public and a private key. Public key is shared without any privacy, while private key is kept secure. The plaintext is encrypted using the public key while only private key can decrypt that ciphertext into plain readable format. It tends to be more secure for communication over unsecured networks. It is slower than symmetric encryption because of its complex computing. However, it is more secure than symmetric encryption. RSA and ECC are some widely known asymmetric algorithm. (Panhwar, et al., 2019)

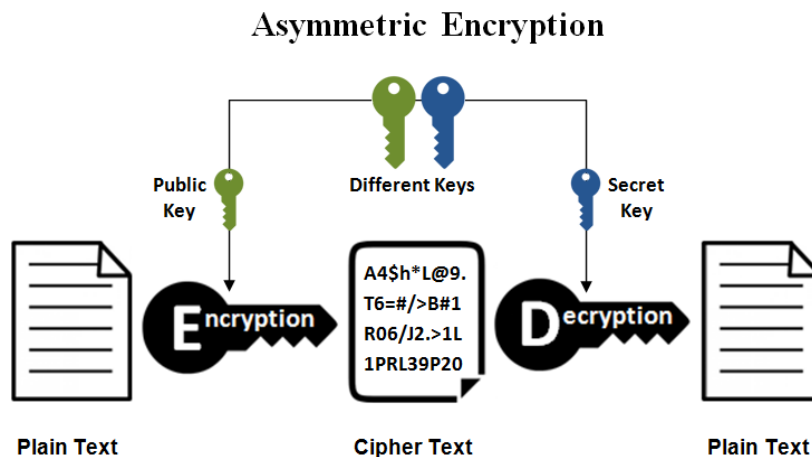


Figure 5: Asymmetric Encryption

Both methods are often used in hybrid systems. Asymmetric encryption is used to securely exchange the symmetric key, which is then employed to encrypt larger amounts of data efficiently.

## 2. Background

### 2.1. Caesar Cipher and its History

The Caesar Cipher is one of the very earliest encryption methods. It was developed during the time of Julius Caesar to securely communicate military messages. It is one of the most basic encryption methods categorized as a substitution cipher. Each letter in the plaintext is shifted to a fixed number of positions, forward or backwards, depending on encryption or decryption in the alphabets. For encryption, the letters in the plaintext are shifted forward to a fix number, while during decryption it is shifted backwards. It was employed to keep message secure and those to be understood by only intended individuals.

It is believed that Julius Caesar used a shift value of three position in the alphabet. For an example, the letter **A** would be **D**, **B** would become **E** and so on. This technique allowed him to securely communicate with his generals on military matters.

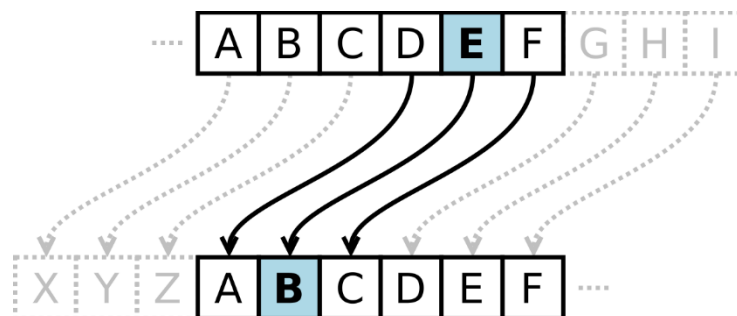


Figure 6: Caesar Cipher

Caesar Cipher is recognized for its simplicity and severely susceptible to vulnerability. It could be easily cracked through the methods of brute-force. However, it serves as an important part in the cryptographic history.

### 2.2. Working Mechanisms

Caesar Cipher is based on a simple process. It involves the shifting the letters of plaintexts by a fixed number of positions in the alphabet. Both encryption and decryption rely on shift value, the key. (Wickramasinghe, 2024)

To facilitate the shifting mechanism, the alphabets are numerically represented as in the table below.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Table 1: Caesar Cipher Table

**Encryption:** During encryption process, every letter in the plaintext is shifted forward by the key value. Taking the scenario of a shift value of 5, the letter at position  $x$  is replaced by the letter at position  $(x+5) \bmod 26$ .

Example with a Shift of 5:

- E => J

The letter E corresponds to position 4. Shifting 5 position forwards, it maps to 9, which is J.

- K => P

The letter K corresponds to position 10. Shifting 5 position forward, it maps to 15, which is P.

Mathematically,

$$E(x) = (x + k) \bmod 26$$

Where,

- $E(x)$  is the encrypted letter position,
- $x$  is the plaintext letter position,
- $k$  is the shift value, the key.

**Decryption:** Decryption is reversing the encryption process. This is done by shifting each letter backwards by the key value. Suppose, a letter is at position  $y$ , the original plaintext letter will be found at position  $(y-5) \bmod 26$ .

Example with a Shift of 5:

- J => E

The letter **J** corresponds to position 9. Shifting 5 position backwards, it maps to 4, which is **E**.

- P => K

The letter **P** corresponds to position 15. Shifting 5 position backwards, it maps to 10, which is **D**.

Mathematically,

$$D(y) = (y - k) \bmod 26$$

Where,

- $D(y)$  is the decrypted letter position,
- $y$  is the encrypted letter position,
- $K$  is the shift value, the key.

## Examples

To demonstrate the encryption and decryption process, the following example can be considered.

- Plaintext: "**NIGHT**"
- Key (Shift Value): 5

Encryption Process:

- N (Position 13) => S (position 18)
- I (position 8) => N (position 13)
- G (position 6) => L (position 11)
- H (position 7) => M (position 12)



- T (position 19)  $\Rightarrow$  Y (position 24)

Ciphertext: “**SNLMY**”

**Decryption Process:**

- S (Position 18)  $\rightarrow$  N (Position 13)
- N (Position 13)  $\rightarrow$  I (Position 8)
- L (Position 11)  $\rightarrow$  G (Position 6)
- M (Position 12)  $\rightarrow$  H (Position 7)
- Y (Position 24)  $\rightarrow$  T (Position 19)

Decipher text: “**NIGHT**”

### 2.3. Advantages

The advantages of Caesar Cipher are as follows:

- a) **Simple and Easy to Use:** Ceaser cipher's working mechanism is simple to understand and implement. This is due to its basic design.
- b) **Fast Encryption and Decryption:** Ceaser Cipher relies on shifting of letters within the alphabet. This makes the encryption and decryption process extremely fast and efficient.
- c) **Minimal Resource Requirements:** The algorithm can operate without the need of any complex computational resources. This makes it suitable for low computational power.
- d) **Easy Key Management:** The shift value (the key) is small and easy to manage. There are only 25 possible shifts in the alphabets.

## 2.4. Disadvantages

The disadvantages of Caesar Cipher are as follows:

- a) **Weak Security:** It has limited key space. There are only 25 possible shifts, making it critically vulnerable to brute-force attack.
- b) **Vulnerable to Frequency Analysis:** It does not alter letter frequencies. The attacker can easily recover the plaintext by identifying the shift value (the key) by analyzing the most common letters in the ciphertext.
- c) **Fixed Shift Pattern:** The same shift is used for all letters. This makes it simple to figure out the entire pattern of the ciphertext.
- d) **Limited Key Space:** There are only 25 keys (shift values). It works with only alphabetic characters and does not include numbers, symbols, or special character. This makes it not secure option for modern encryption.
- e) **Easily Broken with known Plaintext:** If any part of the original message is known by the attackers, the shift value can be easily figured out, making the encryption completely unsecure.

### 3. Development

#### 3.1. Introduction

In this chapter, a new cryptographic algorithm is developed based on the Caesar Cipher with the modification to enhance security. The modification involves using dynamic shifting with a key sequence and then apply Base64 encoding to it to output ciphertext. This has aimed to improve to the algorithm's robustness against common attacks (brute-force and frequency analysis) by the addition of complexity and making the ciphertext harder to decipher.

#### 3.2. Reasons for Modification

The Caesar cipher is one of the earliest and simplest encryption techniques. It is categorized as substitution cipher. It is developed during the time of Julius Caesar to securely communicate military messages with his generals. It is operated by a shifting each letter of a plaintext by a fixed number of portions, forward for encryption while backward for decryption, of the English alphabets. It has a significant contribution in the advancement of cryptology. However, it is highly susceptible to brute-force attacks and frequency analysis due to limited number of possible shifts (i.e. 25). With the modern computation power, one can very easily try all possible shifts in order to decipher the ciphertext.

The following modification introduced this the development of the algorithm to address the weakness of the Caesar Cipher.

- a) **Expanded Character Set:** It supports numbers, space, special and alphabetic characters. This increases the algorithm versatility and adds complexity.
- b) **Dynamic Key Shifting:** It incorporates a key sequence and character position for encryption. This increases the resistance to brute-force and frequency analysis attack.

- c) **Noise Insertion:** Some random noise characters are introduced to obscure the structure of the ciphertext. This thus makes it difficult in identifying patterns that hackers can exploit.
- d) **Base64 Encoding:** It is introduced to further obscure the ciphertext to add another additional layer of complexity in order to harden the ciphertext interpretation directly.
- e) **XOR Operation:** It is introduced to further scramble the ciphertext in order to add further complexity.

These modifications significantly improve the strength and functionality of the Caesar Cipher. These improve both the security and versatility of the cipher, making it more resilient to attacks.

### 3.3. Background and Explanation for Modifications

- 1) **Expanded Characters:** In order to support numbers, spaces, and special characters along with alphabets, a new table is implemented. Here, each character is identified with a unique index value. This table will expand the total character set to 47 characters.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Table 2: Extended Caesar Cipher Table (i)

0	1	2	3	4	5	6	7	8	9
26	27	28	29	30	31	32	33	34	35

Table 3: Extended Caesar Cipher Table (ii)

(SPACE)	.	,	!	?	@	#	\$	%	&	*
36	37	38	39	40	41	42	43	44	45	46

Table 4: Extended Caesar Cipher Table (iii)

Category	Characters	Position
Alphabets	A – Z	0 – 25
Numbers	0 – 9	26 – 35
Space	(space)	36
Special Character	.,?@#\$\$%&*	37-46

Table 5: Extended Caesar Cipher Table

2) **Dynamic Shifting:** Ceaser Cipher uses a fixed single shift value. However, in this algorithm the shift value is be determined dynamically based on a key sequence. This will increase the complexity of the cipher and make is comparatively less predictable. The shift value is calculated based on the key sequence and the position of the character in the plaintext.

Mathematically,

$$C_i = (I_i + K_{(i \bmod n)} + i) \bmod 47$$

Where:

- $I_i$ : Position of character  $P_i$
- $K_j$ : Key value at position  $j$ , where  $j = i \bmod n$
- $i$ : Position of the character in the plaintext (starting from 0)
- 47: Total number of characters in the expanded character set
- $C_i$ : Encrypted index of the character

- 3) **Noise Characters:** The ciphertext are then obscured using random noise character. This is to harden the frequencies analysis. These are inserted into the encrypted text at position that is determined by the key sequence instead. In themselves, they do not carry any meaning but included to increase the complexity at decryption process.

Logically,

Noise character Position = Derived from  $K (i \bmod n)$

Mathematically

$$P_{\text{noise}} = (k + i) \bmod L$$

where  $k$  is the Key sequence,

$i$  is the character (starts from 0)

$L$  is the total number of the character in the text

- 4) **Base64 Encoded:** The Caesar Cipher produces plaintext-like output. This makes them easy to analyze. Base64 encoding is applied, after noise insertion, to further obscure the content. This adds an extra layer of complexity making the ciphertext harder to interpret. Base64 operates on binary data by converting it into a 64-character representation. To encode, first of all combine the encrypted text and the noise characters, then apply Base64 encoding to get final ciphertext.

- 5) **XOR Operation:** For the final encryption, another key, where its value is **3** i.e.  $K_2$  is used. The key is XOR with the last three characters from the output obtained. The XOR operation can be represented mathematically for encryption as follows:

Let,  $K_2$  (the encryption key)

$C_1, C_2, C_3$  be the ASCII values of the last three characters from the output.

$E_1, E_2, E_3$  be the ASCII values after applying the XOR operation.

The mathematical form for the encryption process:

$$C_1 \oplus K_2 = E_1$$

$$C_2 \oplus K_2 = E_2$$

$$C_3 \oplus K_2 = E_3$$

Here,  $\oplus$  denotes the **XOR** operation.

### 3.4. Encryption Algorithm

#### Inputs:

Plaintext: P

Key: A key sequence,  $K = [k_1, k_2, \dots, k_n]$

Let's assume a plain text: "**STRIKE 1205!**" with key sequence **[5,4,3,2,1]**

#### Steps:

1. Map each character in the plaintext into its corresponding position using the character set table.

For that identify the position of each character in the plain text.

Character	Position
S	18
T	19
R	17
I	8
K	10
E	4
[ S p a c e ]	36
1	27
2	28
0	26
5	31
!	39

2. Calculate the dynamic shift key for each character  $P_i$  at position  $i$ :

$$C_i = (l_i + K_{(i \bmod n)} + i) \bmod 47$$

Where:

$l_i$ : Position of character  $P_i$

$K_j$ : Key value at position  $j$ , where  $j = i \bmod n$

$i$ : Position of the character in the plaintext (starting from 0)

Position $i$	Character $P$	Index of character in Plaintext $l_i$	Key, $K_j = n - (i \bmod n)$	Shift Calculation  $(l_i + K_j + i) \bmod 47$	Result $C_i$
0	S	18	5 – (0 mod 5) = 5	(18 + 5 + 0) mod 47	23
1	T	19	5 – (0 mod 4) = 4	(19 + 4 + 1) mod 47	24
2	R	17	5 – (0 mod 3) = 3	(17 + 3 + 2) mod 47	22
3	I	8	5 – (0 mod 2) = 2	(8 + 2 + 3) mod 47	13
4	K	10	5 – (0 mod 1) = 1	(10 + 1 + 4) mod 47	15



5	E	4	$5 - (0 \bmod 5) = 5$	$(4 + 5 + 5) \bmod 47$	14
6	[ S p a c e ]	36	$5 - (0 \bmod 4) = 4$	$(36 + 4 + 6) \bmod 47$	46
7	1	27	$5 - (0 \bmod 3) = 3$	$(27 + 3 + 7) \bmod 47$	37
8	2	28	$5 - (0 \bmod 2) = 2$	$(28 + 2 + 8) \bmod 47$	38
9	0	26	$5 - (0 \bmod 1) = 1$	$(26 + 1 + 9) \bmod 47$	36
10	5	31	$5 - (0 \bmod 5) = 5$	$(31 + 5 + 10) \bmod 47$	46
11	!	39	$5 - (0 \bmod 4) = 4$	$(39 + 4 + 11) \bmod 47$	7

3. Convert the modified position  $C_i$  back to its corresponding characters.

Output	Characters
23	X
24	Y
22	W
13	N

15	P
14	O
46	*
37	.
38	!
36	[s p a c e]
46	*
7	H

Output: **XYWNPO\*.\*H**

4. Add some random noise characters at positions derived from the key. For that identify the position where noise character could be inserted.

$$P_{noise} = (k + i) \bmod L$$

Where:

$P_{noise}$ : Position of the noise character in the ciphertext.

K: Key value from the key sequence, indexed by j.

i: Position of the plaintext character (0-indexed).

L: Length of the plaintext (before noise insertion).

Position i	Key k	Calculation $P_{noise} = (k + i) \bmod L$	Noise Position $P_{noise}$
0	5	$(5 + 0) \bmod 12$	5
1	4	$(4 + 1) \bmod 12$	5

2	3	$(3 + 2) \bmod 12$	5
3	2	$(2 + 3) \bmod 12$	5
4	1	$(1 + 4) \bmod 12$	5
5	5	$(5 + 5) \bmod 12$	10
6	4	$(4 + 6) \bmod 12$	10
7	3	$(3 + 7) \bmod 12$	10
8	2	$(2 + 8) \bmod 12$	10
9	1	$(1 + 9) \bmod 12$	10
10	5	$(5 + 10) \bmod 12$	3
11	4	$(4 + 11) \bmod 12$	3

Since,  $\mathbf{P}_{\text{noise}} = [3, 5, 10]$ , noise characters can be inserted in the position 3, 5 and 10. Thus, output = *XYW#N\$PO.! &H*

5. Encode the output from step 4 to Base64 to get the ciphertext.

5.1. Write the values of the characters into ASCII and convert them to binary.

Character	ASCII	8-bit Binary
X	88	01011000
Y	89	01011001
W	87	01010111
#	35	00100011
N	78	01001110

\$	36	00100100
P	80	01010000
O	79	01001111
*	42	00101010
.	46	00101110
!	33	00100001
[space]	32	00100000
&	38	00100110
H	72	01001000

## 5.2. Combine all those binaries

⇒ 0101100001011001010101110010001101001110  
 0010010001010000010011110010101000101110  
 00100001001000000010011001001000

## 5.3. Group those binary into 6 each and convert them to decimal form and map them to its base character:

⇒ 01011000 01011001 01010111 00100011 01001110  
 00100100 01010000 01001111 00101010 00101110  
 00100001 00100000 00100110 01001000

## 5.4. Base64 characters consist of the following in order:

*ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-*

6-bit Block	Decimal Equivalent	Base64 Character
-------------	--------------------	------------------

010110	22	W
000101	5	F
100101	37	I
010111	23	X
001000	16	Q
110100	52	0
000100	4	E
100100	36	k
001010	10	K
100010	34	i
001011	11	L
100001	33	h
001000	8	l
000010	2	C
011001	25	Z
001000	8	l

Thus, the encrypted text is **“WFIXQ0EkKiLhICZI”**.

6. Now for the final encryption we use another key,  $K_2 = 5$ . XOR the key with the last three characters, i.e CZI.

- $C \text{ XOR } K_2 = 67 \text{ XOR } 5 = 70 = F$
- $Z \text{ XOR } K_2 = 90 \text{ XOR } 5 = 95 = \text{'\_'}'$
- $I \text{ XOR } K_2 = 73 \text{ XOR } 5 = 76 = L$

So, the final cipher text is **WFIXQ0EkKiLhIF\_L**.

### 3.5. Decryption Algorithm

#### Inputs:

Ciphertext: C

Key: A key sequence,  $K = [K_1, K_2, \dots, K_n]$

We have an encrypted text “**WFIXQ0EkKiLhICZI**” with a key sequence [5, 4, 3, 2, 1], Key,  $K_2 = 5$

#### Steps:

1. We xor the last three characters with key,  $k_2 = 5$

- $F \text{ xor } K_2 = 70 \text{ xor } 5 = 67 = C$
- $\_ \text{ or } K_2 = \_ = 95 \text{ xor } 5 = 90 = Z$
- $L \text{ xor } K_2 = L = 76 \text{ xor } 5 = 73 = I$

Thus, the output obtained is “**WFIXQ0EkKiLhICZI**”

2. Decode the ciphertext from Base64 format to get the noisy encoded data.

Convert each base64 characters to a 6-bit binary.

Base64 Character	Decimal Value	6-bit Binary
W	22	010110
F	5	000101
I	37	100101
X	23	010111
Q	16	001000
0	52	110100

E	4	000100
k	36	100100
K	10	001010
i	34	100010
L	11	001011
h	33	100001
l	8	001000
C	2	000010
Z	25	011001
l	8	001000

Then concatenate all binary values and then split them into 8-bit each:

010110 000101 100101 010111 001000 110100 000100 100100 001010 100010  
001011 100001 001000 000010 011001 001000

Now,

8-bit Binary	ASCII Decimal	Character
01011000	88	X
01011001	89	Y
01010111	87	W
00100011	35	#
01001110	78	N
00100100	36	\$
01010000	80	P

01001111	79	O
00101010	42	*
00101110	46	.
00100001	33	!
00100000	32	(space)
00100110	38	&
01001000	72	H

We get output “**XYW#N\$PO\*! &H**”.

3. Use positions that were derived from the key sequence K to identify the position of noise characters and then remove them.

Here, identify the noise character position and remove them.

$$P_{noise} = (K_i + i) \bmod L$$

Position (i)	Key (k)	P <sub>noise</sub>
0	5	(5+0) mod 12=5
1	4	(4+1) mod 12 = 5
2	3	(3+2) mod 12 = 5
3	2	(2+3) mod 12 = 5
4	1	(1+4) mod 12 =5
5	5	(5+5) mod 12 = 10
6	4	(4+6) mod 12 =10
7	3	(3+7) mod 12 = 10



8	2	$(2+8) \bmod 12 = 10$
9	1	$(1+9) \bmod 12 = 10$
10	5	$(5+10) \bmod 12 = 3$
11	4	$(4+11) \bmod 12 = 3$

From the above table, the positions where noise characters were inserted are **3, 5, and 10** are removed. Output: **XYW#N\$PO.! &H**

- Reverse the dynamic shift key for each character  $C_i$  at position  $i$  through the below formula.

$$P_i = (C_i - K_{(i \bmod n)} - i + 47) \bmod 47$$

Where:

- $C_i$ : Position of character  $C_i$  in the character table
- $K_j$ : Key value at position  $j$ , where  $j = i \bmod n$
- $i$ : Position of the character in the plaintext (starting from 0)

Map the characters to their corresponding position through the predefined character table.

Characters	Output
X	23
Y	24
W	22
N	13
P	15
O	14

*	46
.	37
!	38
[s p a c e]	36
*	46
H	7

For each character, we apply the decryption formula for Reverse Dynamic Shift Key to get the following value.

Position (i)	Ciphertext (C)	Index	Key (K)	Calculation	Decrypted Index
0	X	23	5	$(23-5-0+47)\text{mod } 47=18$	18
1	Y	24	4	$(24-4-1+47)\text{mod } 47=19$	19
2	W	22	3	$(22-3-2+47)\text{mod } 47=17$	17
3	N	13	2	$(13-2-3+47)\text{mod } 47=8$	8
4	P	15	1	$(15-1-4+47)\text{mod } 47=10$	10
5	O	14	5	$(14-5-5+47)\text{mod } 47=4$	4
6	*	46	4	$(46-4-6+47)\text{mod } 47=27$	27
7	.	37	3	$(37-3-7+47)\text{mod } 47=28$	28
8	!	38	2	$(38-2-8+47)\text{mod } 47=26$	26
9	[space]	36	1	$(36-1-9+47)\text{mod } 47=31$	31
10	*	46	5	$(46-5-10+47)\text{mod } 47=31$	31

11	H	7	4	$(7-4-11+47)\bmod 47=39$	39
----	---	---	---	--------------------------	----

5. Convert the resulting position to their corresponding character using the character set table. Map the resulting position to the characters from the extended character table.

Resulting Position (P)	Decrypted Character (P)
18	S
19	T
17	R
8	I
10	K
4	E
27	(space)
28	1
26	2
31	0
31	5
39	!

6. Combine all the characters obtained in step 4 to get the original plaintext P.

We get the decrypted text is “**STRIKE 1205!**”.

### 3.6. Naming the Cryptographic Algorithm Developed

This cryptographic algorithm is named “**CipherNova - 47**”.

### 3.7. Flowchart for Encryption and Decryption algorithm

a) For Encryption:

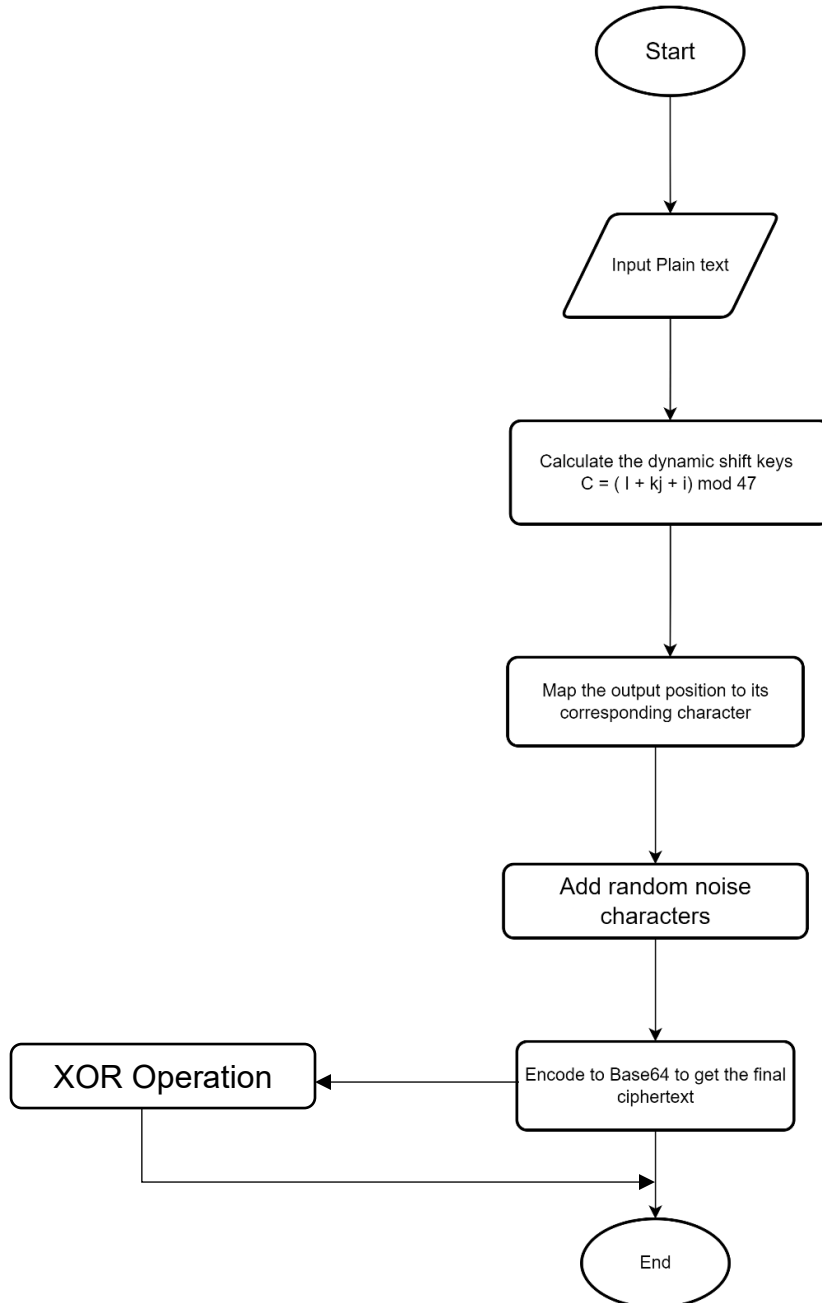
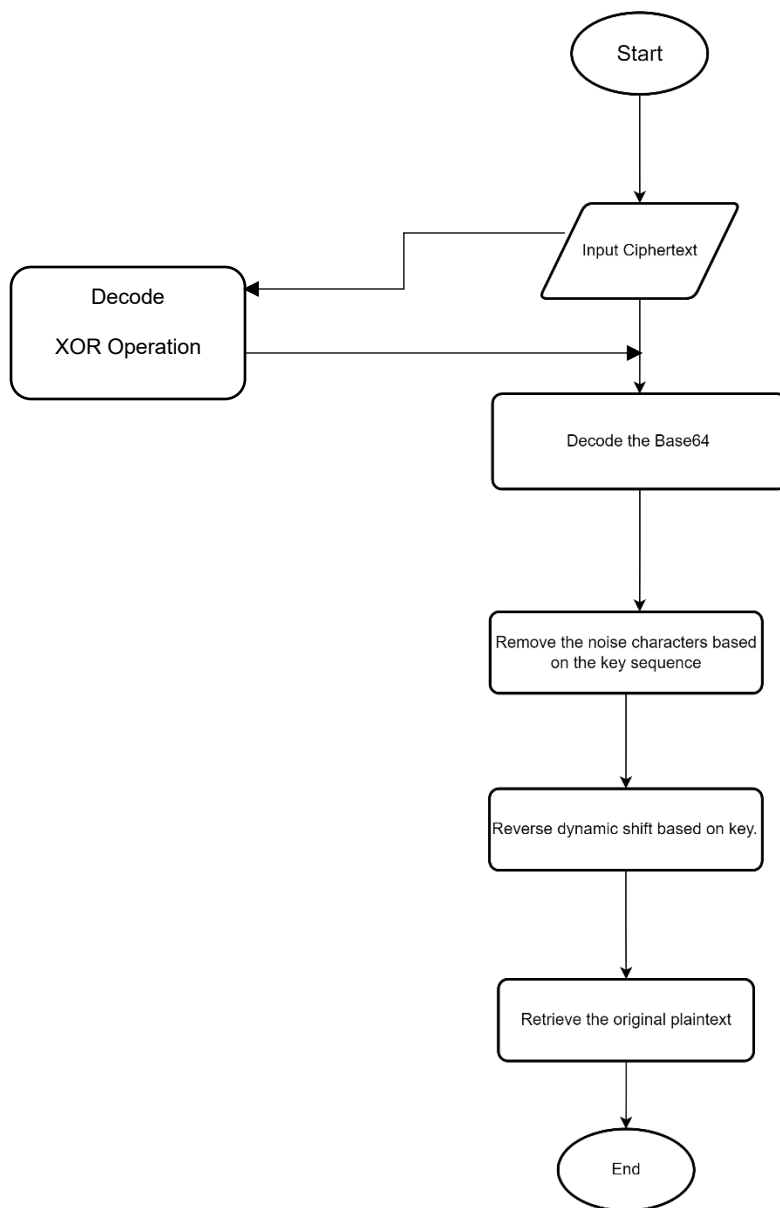


Figure 7: Flowchart for Encryption Algorithm

## b) For Decryption

*Figure 8: Flowchart for Decryption Algorithm*

## 4. Testing

### 4.1. Test 1

#### 4.1.1. Encryption

Plaintext: "Touch"

Key Sequence: [3, 1, 4, 2, 5]

**Step 1:** Map each character in the plaintext into its corresponding position using the custom character set table.

Character	Position
T	19
o	15
u	21
c	12
h	7

**Step 2:** Calculate the dynamic shift key for each character ( $C_i$ ).

Using the formula:  $C_i = (I_i + K_{(i \bmod n)} + i) \bmod 47$

Position $i$	Character $P$	Index of character in Plaintext $I_i$	Key, $K_j = n - (i \bmod n)$  Shift Calculation	$(I_i + K_{(i \bmod n)} + i) \bmod 47$	Result $C_i$
0	T	19	3	$(19 + 3 + 0) \bmod 47 = 22$	22

1	o	15	1	$(15 + 1 + 1)$ $\text{mod } 47 = 17$	17
2	u	21	4	$(21 + 4 + 2)$ $\text{mod } 47 = 27$	27
3	c	12	2	$(12 + 2 + 3)$ $\text{mod } 47 = 17$	17
4	h	7	5	$(7 + 5 + 4)$ $\text{mod } 47 = 16$	16

**Step 3:** Convert the modified positions ( $C_i$ ) back to corresponding characters.

Position ( $C_i$ )	Character
22	W
17	R
27	A
17	R
16	P

We get the output "**WRARP**".

**Step 4:** Add some random noise characters to the output from step 3 at positions derived from the key. For that identify the position where noise character could be inserted using the formula

$$P_{\text{noise}} = (k + i) \text{ mod } L$$

Where:

- K is the key value.
- i is the position of the plaintext character.



- o L is the length of the plaintext.

Position (i)	Key (k)	$P_{\text{noise}} = (k + i) \bmod L$	Noise Position
0	3	$(3 + 0) \bmod 5 = 3$	3
1	1	$(1 + 1) \bmod 5 = 2$	2
2	4	$(4 + 2) \bmod 5 = 1$	1
3	2	$(2 + 3) \bmod 5 = 0$	0
4	5	$(5 + 4) \bmod 5 = 4$	4

Noise Characters: @, #, \$, +, %

Insert Noise at Derived Positions:

Output from step 4: **"WRARP"**

- i Insert + at position 0: "+WRARP"
- ii Insert \$ at position 1: "+\$WRARP"
- iii Insert # at position 2: "+\$#WRARP"
- iv Insert @ at position 3: "+\$#@WRARP"
- v Insert % at position 4: "+\$#@%WRARP"

Insert the random noise characters at these positions. The final output becomes **"+\$#@%WRARP"**.

**Step 5:** Now we encode the output obtained from step 4 into base 64. For that we use the following calculation as shown in the tables below.

First convert characters all the characters to ASCII and binary.

Character	ASCII	8-bit Binary
+	43	00101011
\$	36	00100100

#	35	00100011
@	64	01000000
%	37	00100101
W	87	01010111
R	82	01010010
A	65	01000001
R	82	01010010
P	80	01010000

Then combine all those binaries into one as below.

⇒ 00101011001001000010001101000000001001010101011101010010010000010  
101001001010000

Now, group those into 6-bit blocks and encode as shown in the table below.

6-bit Block	Decimal Equivalent	Base64 Character
001010	10	K
110010	50	y
010000	16	Q
100011	35	j
010000	16	Q
001001	9	J
010101	21	V
110101	53	1

001001	9	J
000101	5	F

Thus, we get the final encrypted text **KyQjQJV1JF**".

**Step 6:** For the final encryption, we use another key, where its value is **2** i.e.  $K_2 = 5$ . Now, XOR the key with the last three characters from the output obtained from step 4, i.e. **1JF**.

- $1 \text{ XOR } K_2 = 49 \text{ XOR } 5 = 52 = 4$
- $J \text{ XOR } K_2 = 74 \text{ XOR } 5 = 79 = O$
- $F \text{ XOR } K_2 = 70 \text{ XOR } 5 = 67 = C$

So, the final cipher text is **KyQjQJV4OC**.

#### 4.1.2. Decryption

Encrypted text: **KyQjQJV4OC**

Key Sequence,  $K_j = [3, 1, 4, 2, 5]$

$K_2 = 5$

**Step 1:** For the initial phase of the decryption, we use key, where  $K_2 = 5$ . Now, XOR the key with the last three characters from the encrypted text i.e. **4OC**.

- $4 \text{ XOR } K_2 = 52 \text{ XOR } 5 = 49 = 1$
- $O \text{ XOR } K_2 = 79 \text{ XOR } 5 = 74 = J$
- $C \text{ XOR } K_2 = 67 \text{ XOR } 5 = 70 = F$

So, the final cipher text is **KyQjQJV1JF**. Now, let's decrypt the Base64 encoded ciphertext "**KyQjQJV1JF**" step-by-step.

**Step 2:** Decode the output obtained from Base64 into binary.

Base64 Character	Decimal Value	6-bit Binary
K	10	001010
y	50	110010
Q	16	010000
j	35	100011
Q	16	010000
J	9	001001
V	21	010101
1	53	110101
J	9	001001
F	5	000101

**Step 3:** Split into 8-bit blocks.

8-bit Binary	ASCII Decimal	Character
00101011	43	+
00100100	36	\$
00100011	35	#
01000000	64	@
00100101	37	%
01010111	87	W
01010010	82	R

01000001	65	A
01010010	82	R
01010000	80	P

Base64 decoded output: "+\$#@%WRARP"

**Step 4:** To remove the noise character, use the following calculations to identify their position and then remove them accordingly.

Position (i)	Key (k)	$P_{\text{noise}} = (k + i) \bmod L$	Noise Position
0	3	$(3 + 0) \bmod 5 = 3$	3
1	1	$(1 + 1) \bmod 5 = 2$	2
2	4	$(4 + 2) \bmod 5 = 1$	1
3	2	$(2 + 3) \bmod 5 = 0$	0
4	5	$(5 + 4) \bmod 5 = 4$	4

Noise Positions: [0, 1, 2, 3, 4]. After removing the noise characters, the output becomes "WRARP".

**Step 5:** Reverse the Shift Key and Map Back.

Position	Character (C <sub>i</sub> )	Decrypted Position (P <sub>i</sub> )	Decrypted Character
0	W	19	T
1	R	15	O
2	A	21	U
3	R	12	C
4	P	7	H

Thus, the final decrypted text is "Touch".

## 4.2. Test 2

### 4.2.1. Encryption

Plaintext: "Go"

Key Sequence,  $K_j$ : [4, 2]

Key,  $K_2 = 3$

**Step 1:** Map each character in the plaintext to its position in the character set

Character	Position ( $l_i$ )
G	7
O	15

**Step 2:** Calculate the dynamic shift key for each character ( $C_i$ ).

Using the formula:  $C_i = (l_i + K_{(i \bmod n)} + i) \bmod 47$

Position $i$	Character $P$	Index of character in Plaintext $l_i$	Key, $K_j = n - (i \bmod n)$ Shift Calculation	$(l_i + K_j = (i \bmod n) + i) \bmod 47$	Result $C_i$
0	G	7	4	$(7+4+0) \bmod 47$	11
1	O	15	2	$(15+2+1) \bmod 47$	18

**Step 3:** Convert the modified positions ( $C_i$ ) back to corresponding characters.

Position ( $C_i$ )	Character
11	L

18	S
----	---

We get the output "**LS**"

**Step 4:** Add some random noise characters to the output from step 3 at positions derived from the key. For that identify the position where noise character could be inserted using the formula

$$P_{\text{noise}} = (k + i) \bmod L$$

Where:

- K is the key value.
- i is the position of the plaintext character.
- L is the length of the plaintext.

Position (i)	Key (k)	$P_{\text{noise}} = (k + i) \bmod L$	Noise Position
0	4	2	$(4+0) \bmod 2 = 0$
1	2	2	$(2+1) \bmod 2 = 1$

Noise Position: [0, 1]. Insert the random noise characters at these positions. The final output becomes "**#@LS**".

**Step 5:** Now we encode the output obtained from step 4 into base 64. For that we use the following calculation as shown in the tables below.

First convert characters all the characters to ASCII and binary.

Character	ASCII	8-bit Binary
#	35	00100011
@	64	01000000
L	76	01001100

S	83	01010011
---	----	----------

Then combine all those binary as shown below.

⇒ 00100011010000000100110001010011

Now, group them into 6-bit blocks as

⇒ 001000 110100 000001 001100 010100 11

⇒ 001000 110100 000001 001100 010100 110000

Since, all there is only 2's 1 bit, we add six zeros to it from the right side. The conversion is done as shown in the table below.

6-bit Block	Decimal	Base64 Character
001000	8	I
110100	52	0
000001	1	B
001100	12	M
010100	20	U
110000	48	w

Thus, we get the encrypted text "**I0BMUw**".

**Step 6:** For the final encryption, we use another key, where its value is **3** i.e.  $K_2 = 3$ . Now, XOR the key with the last three characters from the output obtained from step 4, i.e **MUw**.

We convert the last three character to its ascii value, and then xor with  $K_2$ , the output obtained is again converted to its ascii value.

○  $M \text{ XOR } K_2 = 77 \text{ XOR } 3 = 78 = N$



- $U \text{ XOR } K_2 = 85 \text{ XOR } 3 = 86 = V$
- $w \text{ XOR } K_2 = 119 \text{ XOR } 3 = 116 = t$

So, the final cipher text is **I0BNVt**.

#### 4.2.2. Decryption

Ciphertext: **I0BNVt**

Key Sequence,  $K_j = [4, 2]$

Key,  $K_2 = 3$

**Step 1:** For the initial phase of the decryption, we use key, where  $K_2 = 3$ .

Now, XOR the key with the last three characters from the encrypted text i.e. **NVt**.

- $N \text{ XOR } K_2 = 78 \text{ XOR } 3 = 77 = M$
- $V \text{ XOR } K_2 = 86 \text{ XOR } 3 = 85 = U$
- $t \text{ XOR } K_2 = 116 \text{ XOR } 3 = 119 = w$

So, the final cipher text is **I0BMUw**.

**Step 2:** Let's decrypt the Base64 encoded ciphertext "**I0BMWw**" step-by-step. Decode the output obtained from Base64 into binary.

Base64 Character	Decimal	6-bit Block
I	8	001000
0	52	110100
B	1	000001
M	12	001100
U	20	010100
w	48	110000

Now concatenate all those binaries into one as

⇒ 110100000001001100010100110000

**Step 2:** Split these binaries into 8-bit block each and do the conversion as shown in the table.

8-bit Binary	Decimal	Character
00100011	35	#
01000000	64	@
01001100	76	L
01010011	83	S

Thus, the output obtained is “#@LS”

**Step 3:** To remove the noise character, use the following calculations to identify their position and then remove them accordingly.

Position (i)	Key (k)	$P_{\text{noise}} = (k + i) \bmod L$	Noise Position
0	4	2	$(4+0) \bmod 2 = 0$
1	2	2	$(2+1) \bmod 2 = 1$

Noise Positions: [0, 1]

After removing the noise characters, the output becomes "LS".

**Step 4:** Reverse the Shift Key and Map Back each character to their position through the custom table.

$$P_i = (C_i - K_{(i \bmod n)} - i + 47) \bmod 47$$

Position (i)	Character (C <sub>i</sub> )	Key (K)	Shift Calculation	Decrypted Position (P <sub>i</sub> )
0	L	4	$(11-4-0+47) \bmod 47 = 7$	7
1	S	2	$(18-2-1+47) \bmod 47 = 15$	15

**Step 5:** Now, convert decrypted positions (P<sub>i</sub>) back to characters using the custom table.

Position (P <sub>i</sub> )	Character
7	G
15	O

Thus, the final decrypted text is "**GO**".

### 4.3. Test 3

#### 4.3.1. Encryption

Plaintext: **Arrive 0001**

Key Sequence,  $K_j$ : **[3, 5, 1, 4, 2, 6]**

$K_2 = 8$

**Step 1:** Map each character in the plaintext to its position in the custom character set.

Character	Position
A	0
r	17
r	17
i	8
v	21
e	4
space	36
0	26
0	26
0	26
1	27

**Step 2:** Calculate the dynamic shift key for each character ( $C_i$ ).

Using the formula:

$$C_i = (I_i + K_{(i \bmod n)} + i) \bmod 47$$

Position i	Character P	Index character Plaintext $I_i$	of in $K_j = n - (i \bmod n)$ Shift Calculation	$(I_i + K_j = (i \bmod n) + i) \bmod 47$	Result $C_i$
0	A	0	3	$(0 + 3 + 0) \bmod 47 = 3$	3
1	r	17	5	$(17 + 5 + 1) \bmod 47 = 23$	23
2	r	17	1	$(17 + 1 + 2) \bmod 47 = 20$	20
3	i	8	4	$(8 + 4 + 3) \bmod 47 = 15$	15
4	v	21	2	$(21 + 2 + 4) \bmod 47 = 27$	27
5	e	4	6	$(4 + 6 + 5) \bmod 47 = 15$	15
6	space	36	5	$(36 + 5 + 6) \bmod 47 = 0$	0
7	0	26	1	$(26 + 1 + 7) \bmod 47 = 34$	34
8	0	26	4	$(26 + 4 + 8) \bmod 47 = 38$	38
9	0	26	3	$(26 + 3 + 9) \bmod 47 = 38$	38

10	1	27	5	$(27 + 5 + 10) \bmod 47 = 42$	42
----	---	----	---	-------------------------------	----

**Step 3:** Convert the modified positions ( $C_i$ ) back to its corresponding characters.

Result ( $C_i$ )	Character
3	D
23	X
20	U
15	P
27	4
15	P
0	A
34	8
38	\$
38	\$
42	^

We get the output **"DXUP4PA8\$\$^"**.

**Step 4:** Add some random noise characters to the output from step 3 at positions derived from the key. For that identify the position where noise character could be inserted using the formula

$$P_{\text{noise}} = (k + i) \bmod L$$

Where:

- K is the key value.

- $i$  is the position of the plaintext character.
- $L$  is the length of the plaintext.

Position ( $i$ )	Key ( $k$ )	$P_{noise} = (k + i) \bmod L$	Noise Position
0	3	$(3 + 0) \bmod 11 = 3$	3
1	5	$(5 + 1) \bmod 11 = 6$	6
2	1	$(1 + 2) \bmod 11 = 3$	3
3	4	$(4 + 3) \bmod 11 = 7$	7
4	2	$(2 + 4) \bmod 11 = 6$	6
5	6	$(6 + 5) \bmod 11 = 10$	10
6	5	$(5 + 6) \bmod 11 = 0$	0
7	1	$(1 + 7) \bmod 11 = 8$	8
8	4	$(4 + 8) \bmod 11 = 1$	1
9	3	$(3 + 9) \bmod 11 = 1$	1
10	5	$(5 + 10) \bmod 11 = 4$	4

Noise positions: [0, 1, 3, 6, 7, 10]. Insert random noise characters at these positions. The output after noise insertion becomes: “!@D#XU\$%P4\*PA8\$\$^”

**Step 5:** Encode the Output into Base64. Now, we encode the string “!@D#XU\$%P4\*PA8\$\$^” into Base64.

**Step 5.1:** ASCII to Binary

Character	ASCII	Binary
!	33	00100001

@	64	01000000
D	68	01000100
#	35	00100011
X	88	01011000
U	85	01010101
\$	36	00100100
%	37	00100101
P	80	01010000
4	52	00110100
*	42	00101010
P	80	01010000
A	65	01000001
8	56	00111000
\$	36	00100100
\$	36	00100100
^	94	01011110

**Step 5.2: Combine the binaries.**

⇒ 00100001 01000000 01000100 00100011 01011000 01010101 00100100  
 00100101 01010000 00110100 00101010 01010000 01000001 00111000  
 00100100 00100100 01011110



⇒ 0010000101000000010001000010001101011000010101010010010000100101  
 01010000001101000010101001010000010000010011100000100100001001000  
 1011110

**Step 5.3: Group into 6-bit blocks and convert to decimal.**

⇒ 001000 010100 000001 000100 001000 110101 100001 010101 001001 000010  
 010101 010000 001101 000010 101001 010000 010000 010011 100000 100100  
 001001 000101 111000

6-bit Block	Decimal	Base64 Character
001000	8	I
010100	20	U
000001	1	B
000100	4	E
001000	8	I
110101	53	1
100001	33	h
010101	21	V
001001	9	J
000010	2	C
010101	21	V
010000	16	Q
001101	13	N

000010	2	C
101001	41	p
010000	16	Q
010000	16	Q
010011	19	T
100000	32	g
100100	36	k
001001	9	J
000101	5	F
111000	56	4

Thus, base64 encoded ciphertext is "IUBEI1hVJCVQNCpQQTgkJF4".

**Step 6:** For the final encryption, we use another key, where its value is **8** i.e.  $K_2 = 8$ . Now, XOR the key with the last three characters from the output obtained from step 4, i.e. "JF4".

We convert the last three character to its ascii value, and then xor with  $K_2$ , the output obtained is again converted to its ascii value.

- $J \text{ XOR } K_2 = 74 \text{ XOR } 8 = 66 = B$
- $F \text{ XOR } K_2 = 70 \text{ XOR } 8 = 78 = N$
- $4 \text{ XOR } K_2 = 52 \text{ XOR } 8 = 60 = <$

So, the final cipher text is "IUBEI1hVJCVQNCpQQTgkBN<".

### 4.3.2. Decryption

Ciphertext: **IUBEI1hVJCVQNCpQQTgkBN<**

Key Sequence,  $K_j = [3, 5, 1, 4, 2, 6]$

Key,  $K_2 = 8$

**Step 1:** For the initial phase of the decryption, we use key, where  $K_2 = 8$ . Now, XOR the key with the last three characters from the encrypted text i.e. **BN<**.

- $B \text{ XOR } K_2 = 66 \text{ XOR } 8 = 74 = J$
- $N \text{ XOR } K_2 = 78 \text{ XOR } 8 = 70 = F$
- $< \text{ XOR } K_2 = 60 \text{ XOR } 8 = 52 = 4$

So, the output is **"IUBEI1hVJCVQNCpQQTgkJF4"**.

**Step 2:** Decode the Base64 Ciphertext. We start by decoding the Base64 ciphertext **"IUBEI1hVJCVQNCpQQTgkJF4"**

**Step 2.1:** Base64 Characters to Decimal and Binary

Character	Decimal	Binary (6 bits)
I	8	001000
U	20	010100
B	1	000001
E	4	000100
I	8	001000
1	53	110101
h	33	100001
V	21	010101
J	9	001001

C	2	000010
V	21	010101
Q	16	010000
N	13	001101
C	2	000010
p	41	101001
Q	16	010000
Q	16	010000
T	19	010011
g	32	100000
k	36	100100
J	9	001001
F	5	000101
4	56	111000

### Step 2.2: Combine the Binaries

⇒ 001000 010100 000001 000100 001000 110101 100001 010101 001001 000010  
 010101 010000 001101 000010 101001 010000 010000 010011 100000 100100  
 001001 000101 111000

⇒ 0010000101000000010001000010001101011000010101010010010000100101  
 01010000001101000010101001010000010000010011100000100100001001000  
 1011110

### Step 2.3: Group these into 8-bit block as

⇒ 00100001 01000000 01000100 00100011 01011000 01010101 00100100  
 00100101 01010000 00110100 00101010 01010000 01000001 00111000  
 00100100 00100100 01011110

**Step 2.4:** Convert to ASCII

Binary (8 bits)	Decimal	ASCII Character
00100001	33	!
01000000	64	@
01000100	68	D
00100011	35	#
01011000	88	X
01010101	85	U
00100100	36	\$
00100101	37	%
01010000	80	P
00110100	52	4
00101010	42	*
01010000	80	P
01000001	65	A
00111000	56	8
00100100	36	\$
00100100	36	\$
01011110	94	^

Decoded Output: **!@D#XU\$%P4\*PA8\$\$\$^**

**Step 3:** Remove Noise Characters

Output with Noise: **!@D#XU\$%P4\*PA8\$\$\$^**

Noise Positions: [0, 1, 3, 6, 7, 10]

Remove characters at noise positions as shown in the table.

Position	Character Removed	Remaining Output
0	!	@D#XU\$%P4*PA8\$\$\$^
1	@	D#XU\$%P4*PA8\$\$\$^
3	#	DXU\$%P4*PA8\$\$\$^
6	\$	DXUP4*PA8\$\$\$^
7	%	DXUP4PA8\$\$\$^
10	*	DXUP4PA8\$\$\$^

Noise Removed Output: **DXUP4PA8\$\$\$^**

**Step 4:** Reverse the dynamic shift key for each character  $C_i$  at position  $i$  through the below formula.

$$P_i = (C_i - K_{(i \bmod n)} - i + 47) \bmod 47$$

Where:

$C_i$ : Position of character  $C_i$  in the character table

$K_j$ : Key value at position  $j$ , where  $j = i \bmod n$

$i$ : Position of the character in the plaintext (starting from 0)

Position (i)	Character (P)	Ciphertext Position (C <sub>i</sub> )	Key (K <sub>i</sub> mod n)	Reverse Shift Calculation: (C <sub>i</sub> - K <sub>i</sub> mod n) - i) mod 47	Resulting Position (I <sub>i</sub> )
0	D	3	3	$(3 - 3 - 0) \bmod 47 = 0$	0
1	X	23	5	$(23 - 5 - 1) \bmod 47 = 17$	17
2	U	20	1	$(20 - 1 - 2) \bmod 47 = 17$	17
3	P	15	4	$(15 - 4 - 3) \bmod 47 = 8$	8
4	4	27	2	$(27 - 2 - 4) \bmod 47 = 21$	21
5	P	15	6	$(15 - 6 - 5) \bmod 47 = 4$	4
6	A	0	5	$(0 - 5 - 6) \bmod 47 = 36$	36
7	8	34	1	$(34 - 1 - 7) \bmod 47 = 26$	26
8	\$	38	4	$(38 - 4 - 8) \bmod 47 = 26$	26
9	\$	38	3	$(38 - 3 - 9) \bmod 47 = 26$	26
10	^	42	5	$(42 - 5 - 10) \bmod 47 = 27$	27

**Step 5:** Map the Resulting Positions (I<sub>i</sub>) Back to Characters.

Position ( $I_i$ )	Character
0	A
17	r
17	r
8	i
21	v
4	e
36	space
26	0
26	0
26	0
27	1

Thus, the final decrypted plaintext is “**Arrive 0001**”



#### 4.4. Test 4

##### 4.4.1. Encryption

Plaintext: **7023**

Key Sequence,  $K_j = [3, 4, 6, 8]$

Key,  $K_2 = 7$

**Step 1:** Map each character to its position in the custom character set.

Character	Position
7	33
0	26
2	28
3	29

**Step 2:** Calculate the dynamic shift key for each character ( $C_i$ ).

Using the formula:  $C_i = (I_i + K_{(i \bmod n)} + i) \bmod 47$

Position $i$	Character $P$	Index of character in Plaintext $I_i$	$K_j = n - (i \bmod n)$ Shift Calculation	$(I_i + K_{(i \bmod n)} + i) \bmod 47$	Result $C_i$
0	7	33	3	$(33 + 3 + 0) \bmod 47 = 36$	36
1	0	26	4	$(26 + 4 + 1) \bmod 47 = 31$	31

2	2	28	6	$(28 + 6 + 2) \bmod 47 = 36$	36
3	3	29	8	$(29 + 8 + 3) \bmod 47 = 40$	40

**Step 3:** Map modified positions ( $C_i$ ) to characters.

Result ( $C_i$ )	Character
36	[Space]
31	V
36	[Space]
40	Z

Encrypted text so far: "[space]V[space]Z"

**Step 4:** Add Noise Characters.

$$P_{\text{noise}} = (k + i) \bmod L$$

Where:

- K is the key value.
- i is the position of the plaintext character.
- L is the length of the plaintext.

Position (i)	Key (K)	$P_{\text{noise}}$	Noise Position
0	3	$(3 + 0) \bmod 4 = 3$	3
1	4	$(4 + 1) \bmod 4 = 1$	1

2	6	$(6 + 2) \bmod 4 = 0$	0
3	8	$(8 + 3) \bmod 4 = 3$	3

**Noise Positions:** [0, 1, 3]

Add noise at these positions

Random noise characters: \*, @, #

So, the output with noise “\*@[space]#V[space]Z”.

**Step 5:** Encode the Output into Base64

Convert each character to its ASCII value and then to binary

Character	ASCII	Binary
*	42	00101010
@	64	01000000
[space]	32	00100000
#	35	00100011
V	86	01010110
[space]	32	00100000
Z	90	01011010

Combine the binary values

⇒ 00101010010000000010000000100011010101100010000001011010

Group the combined binary string into 6-bit blocks

⇒ 001010 100100 000000 100000 001000 110101 011000 100000 010110 100000

We add 4 0's to the end block to make it 6-bit block.

6-bit Block	Decimal Equivalent	Base64 Character
001010	10	K
100100	36	k
000000	0	A
100000	32	g
001000	8	I
110101	53	1
011000	24	Y
100000	32	g
010110	22	W
100000	32	g

Thus, the output obtained is **"KkAgI1YgWg"**.

**Step 6:** For the final encryption, we use another key, where its value is **7** i.e.  $K_2 = 7$ . Now, XOR the key with the last three characters from the output obtained from step 4, i.e. **"gWg"**. We convert the last three character to its ascii value, and then xor with  $K_2$ , the output obtained is again converted to its ascii value.

- $g \text{ XOR } K_2 = 103 \text{ XOR } 7 = 96 = `$
- $W \text{ XOR } K_2 = 87 \text{ XOR } 7 = 80 = P$
- $g \text{ XOR } K_2 = 103 \text{ XOR } 7 = 78 = `$

So, the final cipher text is **"KkAgI1Y`P`"**.

#### 4.4.2. Decryption

Ciphertext: **KkAgI1Y`P`**

Key Sequence,  $K_j = [3, 4, 6, 8, 9, 7]$

Key,  $K_2 = 7$

**Step 1:** For the initial phase of the decryption, we use key, where  $K_2 = 7$ .

Now, XOR the key with the last three characters from the encrypted text i.e. **`P`**.

- **`** XOR  $K_2 = 96 \text{ XOR } 7 = 103 = \text{g}$
- **P** XOR  $K_2 = 80 \text{ XOR } 7 = 87 = \text{W}$
- **`** XOR  $K_2 = 96 \text{ XOR } 7 = 103 = \text{h}$

So, the output becomes " **KkAgI1YgWg**".

**Step 2:** Decode the Base64 Ciphertext.

**Step 2.1:** Convert each Base64 character to its binary form

Base64 Character	Decimal Equivalent	6-bit Block
K	10	001010
k	36	100100
A	0	000000
g	32	100000
I	8	001000
1	53	110101
Y	24	011000
g	32	100000
W	22	010110

g	32	100000
---	----	--------

Combine the binary values

⇒ 001010100100000000100000001000110101011000100000010110100000

Group the combined binary string into 8-bit blocks as shown in the table

Binary	ASCII	Character
00101010	42	*
01000000	64	@
00100000	32	(space)
00100011	35	#
01010110	86	V
00100000	32	(space)
01011010	90	Z

Thus, the output obtained is “\* @ # V Z”.

**Step 4:** Add noise characters. For that find the noise character position as shown above in the encryption process. The output would be

Noise positions: [0, 1, 3]

Random noise characters: \* , @ , #

The output without noise is " V Z".

**Step 4:** Apply Reverse Shift to Each Character.

Position (i)	Character (P)	Ciphertext Position (C <sub>i</sub> )	Key ( $K_{(i \bmod n)}$ )	Reverse Shift Calculation	Resulting Position (I <sub>i</sub> )
0	[space]	36	3	$(36 - 3 - 0) \bmod 47 = 33$	33
1	V	21	4	$(21 - 4 - 1) \bmod 47 = 16$	26
2	[space]	36	6	$(36 - 3 - 0) \bmod 47 = 33$	28
3	Z	25	8	$(25 - 8 - 3) \bmod 47 = 14$	14

**Step 5:** Map the Resulting Positions (I<sub>i</sub>) Back to Characters.

Position (I <sub>i</sub> )	Character
33	7
26	0
28	2
29	3

Thus, the final decrypted plaintext is **"7023"**.

## 4.5. Test 5

### 4.5.1. Encryption

Plaintext: **5tr1k3**

Key Sequence: [2, 4, 3, 1, 5, 2]

Key,  $K_2 = 9$

**Step 1:** Map each character in the plaintext to its position in the character set

Character	Position ( $I_i$ )
5	31
t	19
r	17
1	27
k	10
3	29

**Step 2:** Calculate the dynamic shift key for each character ( $C_i$ ).

Using the formula:  $C_i = (I_i + K_{(i \bmod n)} + i) \bmod 47$

Position (i)	Character (P)	Index ( $I_i$ )	Key ( $K_{(i \bmod n)}$ )	Shift Calculation	Resulting Position ( $C_i$ )
0	5	31	2	$(31 + 2 + 0) \bmod 47 = 33$	33
1	t	19	4	$(19 + 4 + 1) \bmod 47 = 24$	24



2	r	17	3	$(17 + 3 + 2) \bmod 47 = 22$	22
3	1	27	1	$(27 + 1 + 3) \bmod 47 = 31$	31
4	k	10	5	$(10 + 5 + 4) \bmod 47 = 19$	19
5	3	29	2	$(29 + 2 + 5) \bmod 47 = 36$	36

**Step 3:** Convert the modified positions ( $C_i$ ) back to corresponding characters

Position ( $C_i$ )	Character
33	7
24	y
22	w
31	5
19	t
36	[space]

The output obtained is "7yw5t "

**Step 4:** Add noise characters at calculated positions as shown in the table.

$$P_{\text{noise}} = (k + i) \bmod 6$$

Position (i)	Key (k)	Noise Position Calculation	Noise Position
0	2	$(2 + 0) \bmod 6 = 2$	2

1	4	$(4 + 1) \bmod 6 = 5$	5
2	3	$(3 + 2) \bmod 6 = 5$	5
3	1	$(1 + 3) \bmod 6 = 4$	4
4	5	$(5 + 4) \bmod 6 = 3$	3
5	2	$(2 + 5) \bmod 6 = 1$	1

**Noise positions:** [1, 2, 3, 4, 5]

Now, insert random noise characters at these positions, giving us the output with noise:

The output obtained after inserting noise character is “!@#\$\$7yw5t[space]”.

#### **Step 5: Encode the output into Base64**

We now encode the string “!@#\$\$7yw5t[space]” into Base64.

##### **Step 5.1: Convert characters to ASCII and binary**

Character	Decimal	8-bit Binary
!	33	00100001
@	64	01000000
#	35	00100011
\$	36	00100100
%	37	00100101
7	55	00110111
y	121	01111001
w	119	01110111
5	53	00110101

T	84	01010100
(space)	32	00100000

**Step 5.2: Combine the binaries**

⇒ 00100001 01000000 00100011 00100100 00100101 00110111 01111001  
01110111 00110101 01010100 00100000

⇒ 001000010100000000100011001001000010010100110111011110010111011100  
1101010101010000100000

**Step 5.3: Group into 6-bit blocks and convert to decimal**

⇒ 001000 010100 000000 100011 001001 000010 010100 110101 011101 000111  
001000 110001 011010 110011 001100 100000

**Step 5.4:** Convert each 6-bit block to decimal and then base64 character as shown in the table.

6-bit Binary	Decimal	Base64 Character
001000	8	I
010100	20	U
000000	0	A
100011	35	j
001001	9	J
000010	2	C
010100	20	U
110101	53	1
011101	29	d

000111	7	H
001000	8	I
110001	49	x
011010	26	a
110011	51	z
001100	12	M
100000	32	g

The output obtained after base64 encoded is **"IUAjJCUI1dHlxazMg"**

**Step 6:** For the final encryption, we use another key, where its value is **8** i.e.  $K_2 = 9$ . Now, XOR the key with the last three characters from the output obtained from step 4, i.e. **"zMg"**. We convert the last three character to its ascii value, and then xor with  $K_2$ , the output obtained is again converted to its ASCII value.

- $z \text{ XOR } K_2 = 122 \text{ XOR } 9 = 115 = s$
- $M \text{ XOR } K_2 = 77 \text{ XOR } 9 = 68 = D$
- $g \text{ XOR } K_2 = 103 \text{ XOR } 9 = 110 = n$

So, the final cipher text, encrypted text, is **"IUAjJCUI1dHlxasDn"**.

#### 4.5.2. Decryption

Ciphertext: **IUAjJCUI1dHlxasDn**

Key Sequence,  $K_j = [2, 4, 3, 1, 5]$

Key,  $K_2 = 9$

**Step 1:** For the initial phase of the decryption, we use key, where  $K_2 = 9$ . Now, XOR the key with the last three characters from the encrypted text i.e. **sDn**.

- $s \text{ XOR } K_2 = 115 \text{ XOR } 9 = 122 = o$
- $D \text{ XOR } K_2 = 68 \text{ XOR } 9 = 77 = M$
- $n \text{ XOR } K_2 = 110 \text{ XOR } 9 = 103 = g$

So, the output is "IUAjJCUI1dHlxazMg".

**Step 2: Decode the Base64 ciphertext.**

**Step 2.1: Convert Base64 to Binary**

Base64 Character	Decimal	6-bit Binary
I	8	001000
U	20	010100
A	0	000000
j	35	100011
J	9	001001
C	2	000010
U	20	010100
1	53	110101
d	29	011101
H	7	000111
I	8	001000
x	49	110001
a	26	011010

z	51	110011
M	12	001100
g	32	100000

**Step 2.2:** Combine the binaries as

⇒ 00100001010000000010001100100100001001010011010101110100011100100  
0110001011010110011001100100000

**Step 2.3:** Group the binary into 8-bit chunks and converted them to decimal and then ASCII character.

⇒ 00100001 01000000 00100011 00100100 00100101 00110111 01111001  
01110111 00110101 01010100 00100000

8-bit Binary	Decimal	ASCII Character
00100001	33	!
01000000	64	@
00100011	35	#
00100100	36	\$
00100101	37	%
00110111	55	7
01111001	121	y
01110111	119	w
00110101	53	5
01010100	84	T

00100000	32	(space)
----------	----	---------

Decoded String: "!@#%7yw5t[space]"

**Step 3:** To remove the noise character, use the following calculations to identify their position and then remove them accordingly.

Position (i)	Key (k)	$P_{noise} = (k + i) \bmod L$	Noise Position
0	4	2	$(4+0) \bmod 2 = 0$
1	2	2	$(2+1) \bmod 2 = 1$

Noise Positions: [0, 1]

After removing the noise characters, the output becomes "7yw5t[space]".

**Step 4:** Reverse the dynamic shift key for each character  $C_i$  at position  $i$  through the below formula.

$$P_i = (C_i - K_{(i \bmod n)} - i + 47) \bmod 47$$

Where:

$C_i$ : Position of character  $C_i$  in the character table

$K_j$ : Key value at position  $j$ , where  $j = i \bmod n$

$i$ : Position of the character in the plaintext (starting from 0)

Position (i)	Ciphertext Character (C)	Position ( $C_i$ )	Key ( $K_j$ )	Reverse Shift Calculation	Original Position ( $I_i$ )
0	7	33	2	$(33 - 2 - 0) \bmod 47 = 31$	31
1	y	24	4	$(24 - 4 - 1) \bmod 47 = 19$	19

2	w	22	3	$(22 - 3 - 2) \bmod 47$ = 17	17
3	5	5	1	$(5 - 1 - 3) \bmod 47$ = 27	27
4	t	19	5	$(19 - 5 - 4) \bmod 47$ = 10	10
5	space	36	2	$(36 - 2 - 5) \bmod 47$ = 29	29

**Step 5:** Now, convert decrypted positions ( $P_i$ ) back to characters using the custom table.

Position ( $P_i$ )	Character
31	5
19	t
17	r
27	1
10	k
29	3

Thus, the decrypted plaintext is “**5tr1k3**”.



## 5. Critical Evaluation of CipherNova-47

*CipherNova – 47* is a developed algorithm for encryption and decryption system. *Caesar cipher* is the base system that used to develop this cryptographic algorithm. It is based on dynamic shift keys, noise insertion, base64 encoding and XOR operation. It has strengthened the traditional Ceaser cipher. However, like all others, it has also some weakness.

### 5.1. Strengths

This newly developed cryptographic algorithm has enhanced the security of traditional Caesar cipher by the addition of multiple layers of security. The strengths of the algorithm are as follows.

- i **Improved Security Against Brute-Force Attacks:** The traditional Caesar Cipher has only 25 possible shift value. This employs dynamic shifting based on a key sequence. This makes brute-force attack computationally much more complex, as each character is shifted differently. This eliminates the predictable patterns.
- ii **Expanded Character Support:** It includes alphabets, number, spaces, and special characters using custom created expanded character table set of 47. This allows the encryption of a wide variety of data, including passwords, codes and alphanumeric data.
- iii **Multiple Layered Encryption:** Multiple layers of scrambling the text are included. Noise character insertion, base64 encoding, XOR operation are each designed to scramble the text in a complex and unpredictable form to minimize predictability. The XOR operation further obscure the ciphertext, ensuring if most of the layers are breach, the final output still remains protected.
- iv **Scalability and Adaptability:** The inclusion of a key sequence allows for flexible configurations based on different keys. This makes the algorithm scalable for varying levels of security requirements.

## 5.2. Weakness

Despite being the improved version of Caesar Cipher, it has certain limitations. These are discussed below.

- i **Increased computational overhead:** The inclusion of dynamic shifts, noise insertion, Base64 encoding and XOR operations have increased the computational complexity. Thus, the encryption and decryption process are slower than the traditional Caesar Cipher algorithm.
- ii **Key management complexity:** The key sequence and XOR key need to be securely stored and shared.
- iii **Longer ciphertext Size:** Noise insertion and Base64 encoding increases the size of the cipher text, resulting in increase of storage requirement and transmission cost.

## 5.3. Application Areas

- i **CipherNova – 47** is an improved version of the traditional Caesar Cipher. It can be applied in various domain for securing communication and data storage.
- ii **Secure Messaging System:** It can be used to encrypt text messages and emails.
- iii **Password Protection:** It is ideal for encrypting passwords and sensitive data stored.
- iv **Digital Authentication Systems:** It can be applied for two-factor authentication and session encryption for user authentication processes.
- v **Cloud Data Storage:** It can be used to encrypt the data before it is uploaded to the cloud storage systems.

- vi ***File Encryption***: It can be employed to encrypt files and documents to secure storage and transmission, preventing unauthorized access.

## 6. Conclusion

CipherNova-47 is an improved and advanced version of the traditional Caesar Cipher, addressing its weakness and introducing mechanisms to improve its encryption mechanism. It employs the mechanism of *dynamic shifting*, *noise insertion*, *Base64 encoding*, and *XOR operations*, adding multiple layers of security. This makes brute force attacks and frequency analysis attack highly impractical.

The algorithm includes alphabets, number, spaces, and special characters using custom created expanded character table, a total of set of 47 characters, allowing the encryption of a wide variety of data, including passwords, codes and alphanumeric data.

## References

Bennett, C. H. & Brassard, G., 1984. Quantum Cryptography: Public Key Distribution and Coin Tossing. pp. 175-179.

David Kahn, S. S., 1967. *he Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. s.l.:Macmillan.

Diffie, W. & Hellman, M., 1976. New Directions in Cryptography. 22(6), pp. 644-654.

Miller, V. S., 1985. Use of Elliptic Curves in Cryptography. Volume 218, pp. 417-426.

Murray, J., 1920. *An Egyptian Hieroglyphic Dictionary*. London: Harrison and Sons.

Panhwar, M. A., Khuhro, S. A., Panhwar, G. & Memon, K. A., 2019. A Study of Symmetric and Asymmetric Cryptographic. Volume 19, pp. 48-53.

PGP Corporation, 2002. *An Introduction to Cryptography*. s.l.:PGP Corporation.

Rivest, R. L. S. A. a. A. L., 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. 21(2), pp. 120-126.

Stallings, W., 2017. *Cryptography and Network Security*. 7th ed. s.l.:Pearson.

Steinberg, J., 2022. *Cybersecurity for Dummies*. 2nd ed. s.l.:John Wiley & Sons, Inc.

Wickramasinghe, S., 2024. *Splunk Blogs*. [Online]

Available at: [https://www.splunk.com/en\\_us/blog/learn/caesar-cipher.html](https://www.splunk.com/en_us/blog/learn/caesar-cipher.html)