LONDON METROPOLITAN UNIVERSITY

islington college

(इस्लिङ्टन 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: SUSHANT CHAUDHARY

London Met ID: 23047494

College ID: np01nt4a230169

Assignment Due Date: 20th January 2025

Assignment Submission Date: 20th January 2025

Word Count:8410

I confirm that I understand my coursework needs to be submitted online via My second teacher/MST 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.

Table of Contents

List of Tables

Table of Figures

23047494 SUSHANT CHAUDHARY.docx

Islington College,Nepal

Document Details

Submission ID

trn:oid:::3618:77477519

Submission Date

Jan 6, 2025, 11:28 PM GMT+5:45

Download Date

Jan 6, 2025, 11:29 PM GMT+5:45

File Name

23047494 SUSHANT CHAUDHARY.docx

File Size

66.2 KB

119 Pages

9,561 Words

56,172 Characters

9% Overall Similarity

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

Match Groups

- **73** Not Cited or Quoted 8%
  Matches with neither in-text citation nor quotation marks
- **8** 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

4%	Internet sources
1%	Publications
7%	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

Cryptography is a foundation of ultramodern information security, ensuring the confidentiality, integrity, and authenticity of sensitive data in an progressively connected digital world. The aim behind this coursework is to address the limitations of being cryptographic ways, similar as predictable crucial patterns and limited versatility, and to produce a secure, robust, and adaptable encryption algorithm. This design aims to contribute to the evolving geography of cryptography by developing a new algorithm acclimatized to meet contemporary security demands.

Problem Statement

Traditional encryption styles, like the Caesar Cipher, suffer from vulnerabilities similar as limited crucial space, linearity, and vulnerability to cryptanalysis. These sins expose translated data to risks, including brute- force and frequence analysis attacks. This coursework seeks to address these challenges by designing a cryptographic algorithm that combines enhanced non-linearity, dynamic crucial shifts, and robust character obfuscation, icing both data security and practical connection.

Overview

The Trigonometric Dynamic Obfuscation Cipher (TDOC) was developed as an advanced cryptographic result to overcome the limitations of traditional ciphers. The algorithm was designed using a mix of fine functions, including trigonometric metamorphoses, modular computation, and XOR operations**.** The process involved:

1. Analyzing the weaknesses of existing ciphers.
2. Conceptualizing and designing the EVSC algorithm.
3. Implementing the algorithm for encryption and decryption processes.
4. Testing its functionality across multiple scenarios and datasets.
5. Evaluating its performance and security against known cryptographic attacks.

Outcome

The TDOC algorithm successfully demonstrated its ability to encrypt and decrypt diverse data types while addressing key security challenges. Through testing, it was proven to provide strong resistance to brute-force and frequency analysis attacks, withstanding potential cryptanalysis through its dynamic, non-linear key generation and multi-layered encryption process. The algorithm supports extended ASCII characters, making it versatile for various applications.

Implications

The development of Trigonometric Dynamic Obfuscation Cipher (TDOC) represents a meaningful contribution to the field of information security, showcasing an innovative approach to cryptographic design. Its enhanced security features and adaptability make it suitable for applications in secure messaging, database encryption, and other areas requiring robust data protection. By addressing known cryptographic challenges, this algorithm offers a framework for future advancements in secure communication technologies.

1. Introduction

1.1 Security Overview

In the context of computing, security refers to the measures held to maintain information, systems, and networks from unauthorized access, damage, or theft. It encompasses a broad pasture of practices aimed at assuring the confidentiality, integrity, and black hole of data. The rapid-fire growth of digital technologies, including cloud computing, the Internet of Things (IoT), and e-commerce, has heightened the need for robust security mechanisms to help cyberattacks, data breaches, and other vicious conditioning. The significance of security cannot be exaggerated, as compromising sensitive information can have far- reaching consequences, from fiscal losses to reputational damage and legal impacts. Security encompasses colorful domains similar as network security, operation security, and endpoint security, each playing a vital part in guarding data and maintaining the trust of users and associations (itgovernance, 2024).

1.2 CIA Triad

One of the essential models for understanding and enforcing security is the CIA Triad, which stands for Confidentiality, Integrity, and Availability. Each element of the CIA trio represents a heart principle of information security.

• **Confidentiality** ensures that sensitive data is penetrated only by authorized individuals or systems. It involves ways suchlike as encryption and access controls to help unauthorized revelation of information.

• **Integrity** refers to the delicacy and thickness of data. This ensures that data cannot be changed or tampered with by unauthorized bodies. Integrity is maintained through mechanisms like mincing and digital autographs, which corroborate that data has not been modified.

• **Availability** ensures that data and systems are accessible when demanded by authorized users. It involves maintaining systems that are flexible to attacks, similar as Distributed Denial of Service (DDoS) attacks, and furnishing backup and disaster recovery results to ensure data remains available indeed in the event of a failure.

Together, these three principles form the backbone of utmost security systems and strategies. By securing confidentiality, icing data integrity, and maintaining availability, associations can cover against a wide range of cyber threats (Kidd, 2024) (websitesecuritystore, 2021).

Figure 1:The CIA Triad – A foundational model for information security (lee, 2024).

1.3 Introduction to Cryptography

Cryptography plays a pivotal part in securing digital dispatches and data, particularly in surroundings where information needs to be transmitted across insecure channels. Cryptography is the practice and study of ways for securing communication and information using canons and ciphers. The primary thing of cryptography is to cover data from unauthorized access or tampering, assuring its confidentiality, integrity, and authenticity (Richards, 2024).

In the current digital world, cryptography is essential for securing online deals, communication, and the protection of sensitive information similar as passwords, particular identification figures, and fiscal records. Cryptographic systems give the foundation for colorful security protocols, including SSL/ TLS for secure web browsing, PGP for dispatch encryption, and blockchain technologies for secure and transparent digital deals. Without cryptography, ultramodern digital services like online banking, e-commerce, and private communication would not be possible (ibm, 2024).

History of Cryptography

Cryptography has a long and rich history that dates to ancient societies, evolving alongside technological advancements in communication and calculation.

Ancient Cryptography: Beforehand cryptographic styles, similar as the Scytale Cipher (500 BCE), were used by the Spartans to cover military dispatches. The Caesar Cipher, developed by Julius Caesar in 58 BCE, involved shifting letters of the ABC by a fixed number of positions, offering introductory protection for sensitive dispatches.

Medieval Cryptography: During the Middle periods, cryptography became more sophisticated. The Vigenère Cipher (1586) introduced the idea of using a key to shift letters in the ABC, enhancing the security of dispatches. Around the 9th century, Al- Kindi developed frequence analysis, a system that helped break negotiation ciphers by assaying the frequence of letters in ciphertext.

World War Cryptography: Cryptography played a vital part in World War II, with the Germans using the Enigma Machine to render their military dispatches. The machine was considered unbreakable until Alan Turing and his platoon cracked the law, significantly abetting the Allied palm. The development of other secure systems like the SIGABA and Type machines further shaped cryptographic practices.

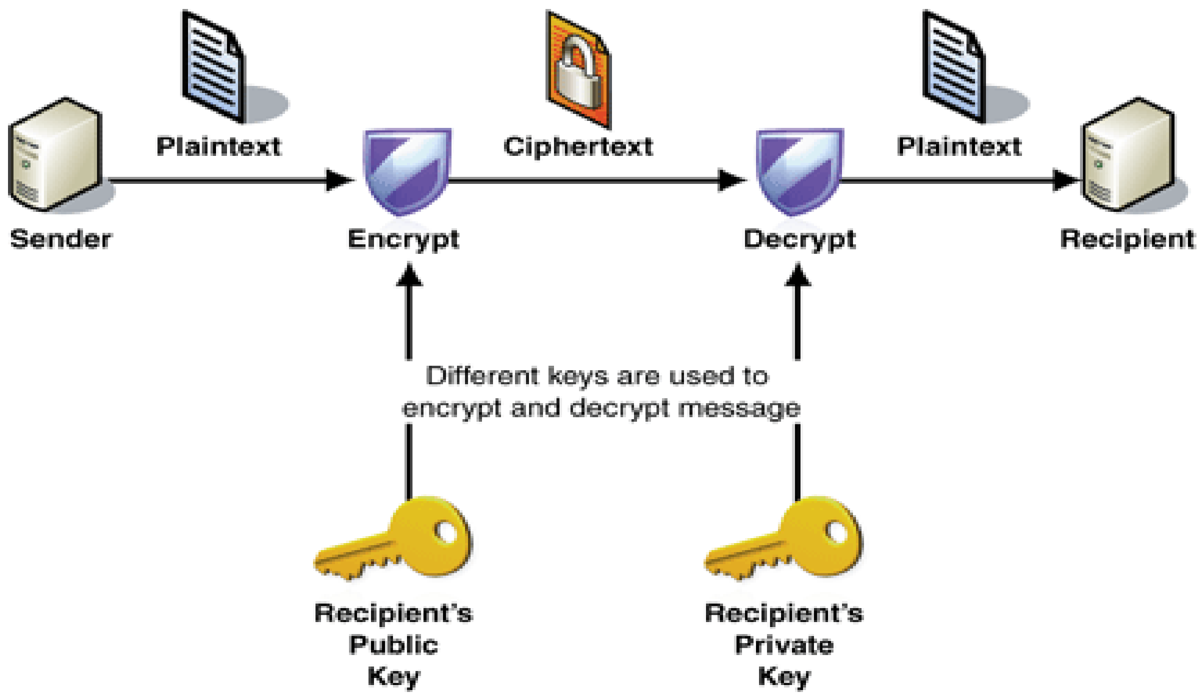

Figure 2:The encryption and decryption process in cryptography (Christian, 2024).

ultramodern Cryptography: In the 1970s, the preface of public- crucial cryptography revolutionized cryptography. Diffie-Hellman (1976) introduced the concept of swapping keys over an insecure channel, and the RSA Algorithm(1977) allowed for secure encryption using large high figures. These inventions form the backbone of ultramodern cryptographic systems used for online banking, secure messaging, and digital autographs.

Now, cryptography continues to be a foundation of digital security, evolving with new technologies like blockchain and amount cryptography, icing the safety and sequestration of communication in a decreasingly connected world.

1.4 Key Terminologies

To fully understand cryptography, it's essential to grasp some of its key terminologies. Below are the basic terms used in cryptographic systems:

- **Encryption**: The process of converting plain text into a encoded format, generally using a cryptographic algorithm and a key, to cover the data from unauthorized access. The encryption procedure ensures that only those with the correct decryption key can read the original dispatch (sciencedirect, 2024).
- **Decryption**: The reversed process of encryption, where the decoded data is converted back to its original form (plain textbook) using the applicable key. Decryption is essential for authorized users to recoup the original data (geeksforgeeks.org, 2024).
- **Cryptosystem**: A cryptosystem refers to the complete system used to perform encryption and decryption, including the algorithms, keys, and protocols that manage the encryption process. A cryptosystem ensures that the data remains secure during transmission or storehouse (Richards, 2024).
- **Symmetric Encryption**: A type of encryption where the same key is used for both encryption and decryption. This system is effective and fast but requires secure crucial operation since both parties need to have the same key. samples include the AES(Advanced Encryption Standard) algorithm (cryptomathic, 2024).
- **Asymmetric Encryption** Asymmetric Encryption Also known as public-key cryptography, asymmetric encryption uses a duo of affiliated keys — public and private keys. One key is used to cipher the data (public key), and the other is used to decipher it(private key). This type of encryption eliminates the need for both parties to partake in a secret key. Popular asymmetric algorithms include RSA and ECC (Elliptic wind Cryptography) (simplilearn, 2024).
- **Key**: A value used in confluence with a cryptographic algorithm to encrypt and decrypt data. Keys must be kept secure, as their concession could lead to unauthorized access to translated data (ninjaone, 2024).

- **Hashing**: Hashing The process of transposing input data (of any size) into a fixed-size value, called a hash or condensation. Hashing is a one- way process and is primarily used to ensure data integrity and generate digital signatures (Loo, 2024).

2. Background

2.1 Caesar Cipher

The Caesar Cipher, termed after Julius Caesar, who applied it to cover martial messages, is one of the oldest and simplest forms of encryption. It belongs to the family of substitution ciphers, where each character in the plaintext is substituted with another character grounded on a fixed rule. Specifically, it shifts each letter in the plaintext by a fixed number of positions, called the" shift key," in the alphabet.

As a symmetric encryption algorithm, the Caesar Cipher uses the duplicate key for both encryption and decryption. This plainness makes it fast and direct to use but also leaves it susceptible to cryptanalysis. Although the Caesar Cipher is not suitable for modern encryption needs, it is widely studied in cryptography courses due to its historical significance and role in developing more advanced algorithms (telsy, 2024).

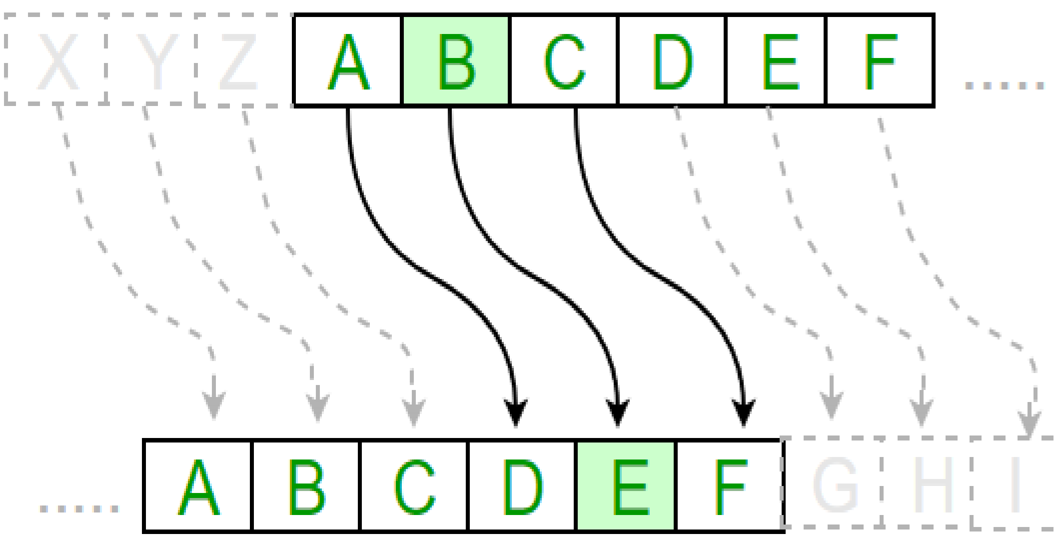

Figure 3:Step-by-step encryption process of the Caesar Cipher with a shift key of 3 (geeksforgeeks.org, 2024).

2.2 Algorithm Background

The Caesar Cipher operates using modular arithmetic to "wrap around" the alphabet. When shifting letters past the end of the alphabet (e.g., Z), the cipher loops back to the beginning (A). This principle ensures every letter remains within the bounds of the alphabet.

1. **Encryption Process:**
   - A key (K) is chosen, typically an integer between 1 and 25 for the English alphabet.
   - Each letter in the plaintext (P) is shifted by K positions using the formula: $C=(P+K) \bmod 26$
   - Here, C is the position of the ciphertext character (Amiruddin, 2024).
2. **Decryption Process:**
   - Using the same key (K), each ciphertext character (C) is shifted back by KKK positions using $P=(C-K) \bmod 26$
   - This process retrieves the original plaintext (geeksforgeek, 2024).

2.3 Applications of the Caesar Cipher

Historically, the Caesar Cipher was pivotal in encoding sensitive information, especially during military operations. Its simplicity and effectiveness in earlier times made it a widely used cryptographic tool. Below is a detailed look at its applications, both historical and modern:

1. Military Communications (Historical Use)

- Purpose: Encrypting sensitive military messages to ensure confidentiality during wartime.
- Details:
    - Julius Caesar used this cipher to communicate securely with his generals.
    - Messages were encrypted using a predetermined shift key, making interception by adversaries less impactful.

2. Educational Tool in Cryptography

- Purpose: Teaching the fundamentals of encryption and cryptanalysis in academic settings.
- Details:
    - Serves as an introductory example of substitution ciphers in computer science and mathematics courses.
    - Demonstrates concepts like modular arithmetic, keys, and decryption processes.

3. Puzzles and Games

- Purpose: Adding an element of cryptographic challenge to recreational activities.
- Details:
  - Widely used in escape rooms, cryptographic puzzles, and brainteasers.
  - Simple implementation makes it accessible for enthusiasts and casual gamers.

4. Data Obfuscation

- Purpose: Basic obfuscation for non-critical data to deter casual inspection.
- Details:
  - Used in applications where sophisticated encryption is not required, such as encoding hints or simple identifiers.
  - Common in early video games to hide certain strings or commands.

5. Cultural and Historical Significance

- Purpose: Preserving the legacy and evolution of cryptographic methods.
- Details:
  - Studied for its historical impact on communication security.
  - Serves as a benchmark for understanding the development of modern encryption systems.

2.4 Worked Example

Let us encrypt and decrypt the plaintext "HELLO" using a shift key (K=3):

Step 1: Encryption

- Plaintext: HELLO
- Convert each letter into its numerical position in the alphabet (A=0, B=1, ..., Z=25):
  H = 7, E = 4, L = 11, O = 14.
- Shift each letter by K=3K = 3K=3:
  - C_h= (7+3) mod 26=10 → K
  - C_E= (4+3) mod 26=7→ H
  - C_L= (11+3) mod 26=14→ O
  - C_O= (14+3) mod 26=17→ R
- Ciphertext: KHOOR

Step 2: Decryption

- Ciphertext: KHOOR
- Convert each letter back into its numerical position:
  K = 10, H = 7, O = 14, R = 17.
- Shift each letter back by K=3K = 3K=3:
  - PK= (10−3) mod 26=7→ H
  - PH= (7−3) mod 26=4→ E
  - PO= (14−3) mod 26=11 → L
  - PR= (17−3) mod 26=14P→ O
- Plaintext: HELLO

2.5 Advantages and Disadvantages

Advantages:

1. **Simplicity:** The Caesar Cipher is extremely easy to understand and apply, making it an excellent starting point for newcomers in cryptography.
2. **Effectiveness:** Its straightforward algorithm requires minimum computational coffers, making it suitable for featherlight encryption tasks in constrained surroundings.
3. **Educational Value:** As one of the oldest cryptographic ways, the Caesar Cipher provides sapience into the elaboration of encryption styles.
4. **Historical significance:** It highlights the necessity of encryption in ancient communication and serves as a foundational conception in cryptography.
5. **Customizability:** The shift key can be adjusted to introduce variation in the encryption process (javatpoint, 2024).

Disadvantages:

1. **Weak Security:** With only 25 possible shift keys, the Caesar Cipher is highly susceptible to brute-force attacks. An attacker can try all possible keys to decode the ciphertext.
2. **Predictability:** The fixed shift pattern makes it vulnerable to frequency analysis, where the frequency of letters in the ciphertext can reveal the plaintext.
3. **Limited Alphabet:** Traditional accomplishments are confined to the 26 letters of the English alphabet, banning figures, symbols, and spaces.
4. **Lack of Key Management:** There's no secure system for participating the shift key, exposing the system to interception and unauthorized access.
5. **Modern Irrelevance:** The simplicity of the Caesar Cipher makes it unsuitable for securing sensitive information in the modern digital landscape (geeksforgeeks, 2024).

3. Development

3.1 Name of the Algorithm:

Trigonometric Dynamic Obfuscation Cipher (TDOC)

3.2 Research

The Caesar Cipher, while foundational in cryptography, has several inherent limitations, including its predictable nature and small key space. The **Trigonometric Dynamic Obfuscation Cipher (TDOC)**addresses these weaknesses with the following improvements:

1. **Position-Based Variable Shifts:**
   - Each character's shift is dynamically calculated based on its position in the plaintext combined with a constant user-defined key (Shift Key=5).
   - This ensures each character is shifted by a unique value, adding complexity and making patterns less predictable.
2. **Sine and Cosine Functions for Shifting**:
   - The shift value is now modified using sine and cosine functions, introducing additional non-linearity and ensuring more complex shifts. This change adds a layer of unpredictability and enhances security.
   - The formula for the shift becomes:
     Shift = (sin (Position + Shift Key) * cos (Position + Shift Key)) * 256
     The result is modulated to stay within the ASCII range (0–255).
3. **XOR Operation:**
   - To introduce non-linearity, the shifted character value is XORed with a constant (XOR Key=10).
   - XOR operations are effective in cryptography for obfuscating data, making it significantly harder to reverse without the key.

4. **Extended Character Set:**
    - By using the full ASCII range (0–255), the algorithm supports encryption of all characters, including alphanumeric, special symbols, and control characters.
    - This makes it versatile for encrypting diverse data types.
5. **Multiple Encryption Rounds:**
    - To enhance cryptographic strength, the algorithm applies **two rounds of encryption**. Each round further obfuscates the ciphertext, increasing resistance to cryptanalysis.
6. **ASCII values:** ASCII, or American Standard Code for Information Interchange, is a extensively used character encoding standard that assigns numerical values to characters similar as letters, integers, and symbols. It uses 7 bits to represent 128 characters (0 – 127), including control characters for operations like newline and tab, as well as printable characters like uppercase letters(A – Z), lowercase letters(a – z), integers(0 – 9), and punctuation. An extended interpretation of ASCII utilizes 8 bits, expanding the range to 256 characters (0 – 255). ASCII is abecedarian in textbook representation across programming, communication protocols, and train formats like plain textbook. For this case, the uppercase letter' A' is represented as 65 in numeric, 41 in hexadecimal, and 01000001 in double. Its simplicity and universality have made ASCII a foundation in calculating systems and data decrypting (Loshin, 2024).

ASCII control characters (character code 0-31)

The first 32 characters in the ASCII-table are unprintable control codes and are used to control peripherals such as printers.

DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
0	000	00	00000000	NUL	&#00;		Null character
1	001	01	00000001	SOH	&#01;		Start of Heading
2	002	02	00000010	STX	&#02;		Start of Text
3	003	03	00000011	ETX	&#03;		End of Text
4	004	04	00000100	EOT	&#04;		End of Transmission
5	005	05	00000101	ENQ	&#05;		Enquiry
6	006	06	00000110	ACK	&#06;		Acknowledge
7	007	07	00000111	BEL	&#07;		Bell, Alert
8	010	08	00001000	BS	&#08;		Backspace
9	011	09	00001001	HT			Horizontal Tab
10	012	0A	00001010	LF
	Line Feed
11	013	0B	00001011	VT	&#11;		Vertical Tabulation
12	014	0C	00001100	FF			Form Feed
13	015	0D	00001101	CR
	Carriage Return
14	016	0E	00001110	SO	&#14;		Shift Out
15	017	0F	00001111	SI	&#15;		Shift In
16	020	10	00010000	DLE	&#16;		Data Link Escape
17	021	11	00010001	DC1	&#17;		Device Control One (XON)
18	022	12	00010010	DC2	&#18;		Device Control Two
19	023	13	00010011	DC3	&#19;		Device Control Three (XOFF)
20	024	14	00010100	DC4	&#20;		Device Control Four
21	025	15	00010101	NAK	&#21;		Negative Acknowledge
22	026	16	00010110	SYN	&#22;		Synchronous Idle
23	027	17	00010111	ETB	&#23;		End of Transmission Block
24	030	18	00011000	CAN	&#24;		Cancel
25	031	19	00011001	EM	&#25;		End of medium
26	032	1A	00011010	SUB	&#26;		Substitute
27	033	1B	00011011	ESC	&#27;		Escape
28	034	1C	00011100	FS	&#28;		File Separator
29	035	1D	00011101	GS	&#29;		Group Separator
30	036	1E	00011110	RS	&#30;		Record Separator
31	037	1F	00011111	US	&#31;		Unit Separator

Figure 4: ASCII control characters(character code 0-31) (ascii-code, 2024)

ASCII printable characters (character code 32-127)

Codes 32-127 are common for all the different variations of the ASCII table, they are called printable characters, represent letters, digits, punctuation marks, and a few miscellaneous symbols. You will find almost every character on your keyboard. Character 127 represents the command DEL.

DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
32	040	20	00100000	SP			Space
33	041	21	00100001	!	!	!	Exclamation mark
34	042	22	00100010	"	"	"	Double quotes (or speech marks)
35	043	23	00100011	#	#	#	Number sign
36	044	24	00100100	$	$	$	Dollar
37	045	25	00100101	%	%	%	Per cent sign
38	046	26	00100110	&	&	&	Ampersand
39	047	27	00100111	'	'	'	Single quote
40	050	28	00101000	(	(	&lparen;	Open parenthesis (or open bracket)
41	051	29	00101001	)	)	&rparen;	Close parenthesis (or close bracket)
42	052	2A	00101010	*	*	*	Asterisk
43	053	2B	00101011	+	+	+	Plus
44	054	2C	00101100	,	,	,	Comma
45	055	2D	00101101	-	-		Hyphen-minus
46	056	2E	00101110	.	.	.	Period, dot or full stop
47	057	2F	00101111	/	/	/	Slash or divide
48	060	30	00110000	0	0		Zero
49	061	31	00110001	1	1		One
50	062	32	00110010	2	2		Two
51	063	33	00110011	3	3		Three
52	064	34	00110100	4	4		Four
53	065	35	00110101	5	5		Five
54	066	36	00110110	6	6		Six
55	067	37	00110111	7	7		Seven
56	070	38	00111000	8	8		Eight
57	071	39	00111001	9	9		Nine
58	072	3A	00111010	:	:	:	Colon
59	073	3B	00111011	;	;	;	Semicolon
60	074	3C	00111100	<	<	<	Less than (or open angled bracket)
61	075	3D	00111101	=	=	=	Equals
62	076	3E	00111110	>	>	>	Greater than (or close angled bracket)
63	077	3F	00111111	?	?	?	Question mark
64	100	40	01000000	@	@	@	At sign

Figure 5:ASCII control characters(character code 32-64) (ascii-code, 2024)

ASCII Table ▾ | ASCII Characters | ASCII Art | Articles | FAQ | Facts | History | Glossary | Compare | English ▾ | Resources ▾

65	101	41	01000001	A	A		Uppercase A
66	102	42	01000010	B	B		Uppercase B
67	103	43	01000011	C	C		Uppercase C
68	104	44	01000100	D	D		Uppercase D
69	105	45	01000101	E	E		Uppercase E
70	106	46	01000110	F	F		Uppercase F
71	107	47	01000111	G	G		Uppercase G
72	110	48	01001000	H	H		Uppercase H
73	111	49	01001001	I	I		Uppercase I
74	112	4A	01001010	J	J		Uppercase J
75	113	4B	01001011	K	K		Uppercase K
76	114	4C	01001100	L	L		Uppercase L
77	115	4D	01001101	M	M		Uppercase M
78	116	4E	01001110	N	N		Uppercase N
79	117	4F	01001111	O	O		Uppercase O
80	120	50	01010000	P	P		Uppercase P
81	121	51	01010001	Q	Q		Uppercase Q
82	122	52	01010010	R	R		Uppercase R
83	123	53	01010011	S	S		Uppercase S
84	124	54	01010100	T	T		Uppercase T
85	125	55	01010101	U	U		Uppercase U
86	126	56	01010110	V	V		Uppercase V
87	127	57	01010111	W	W		Uppercase W
88	130	58	01011000	X	X		Uppercase X
89	131	59	01011001	Y	Y		Uppercase Y
90	132	5A	01011010	Z	Z		Uppercase Z
91	133	5B	01011011	[	[	[	Opening bracket
92	134	5C	01011100	\	\	\	Backslash
93	135	5D	01011101	]	]	]	Closing bracket
94	136	5E	01011110	^	^	^	Caret - circumflex
95	137	5F	01011111	_	_	_	Underscore
96	140	60	01100000	`	`	`	Grave accent
97	141	61	01100001	a	a		Lowercase a
98	142	62	01100010	b	b		Lowercase b
99	143	63	01100011	c	c		Lowercase c
100	144	64	01100100	d	d		Lowercase d
101	145	65	01100101	e	e		Lowercase e

Figure 6:ASCII control characters (character code 65-101) (ascii-code, 2024)

102	146	66	01100110	f	f		Lowercase f
103	147	67	01100111	g	g		Lowercase g
104	150	68	01101000	h	h		Lowercase h
105	151	69	01101001	i	i		Lowercase i
106	152	6A	01101010	j	j		Lowercase j
107	153	6B	01101011	k	k		Lowercase k
108	154	6C	01101100	l	l		Lowercase l
109	155	6D	01101101	m	m		Lowercase m
110	156	6E	01101110	n	n		Lowercase n
111	157	6F	01101111	o	o		Lowercase o
112	160	70	01110000	p	p		Lowercase p
113	161	71	01110001	q	q		Lowercase q
114	162	72	01110010	r	r		Lowercase r
115	163	73	01110011	s	s		Lowercase s
116	164	74	01110100	t	t		Lowercase t
117	165	75	01110101	u	u		Lowercase u
118	166	76	01110110	v	v		Lowercase v
119	167	77	01110111	w	w		Lowercase w
120	170	78	01111000	x	x		Lowercase x
121	171	79	01111001	y	y		Lowercase y
122	172	7A	01111010	z	z		Lowercase z
123	173	7B	01111011	{	{	{	Opening brace
124	174	7C	01111100	|					Vertical bar
125	175	7D	01111101	}	}	}	Closing brace
126	176	7E	01111110	~	~	˜	Equivalency sign - tilde
127	177	7F	01111111	DEL	&#127;		Delete

Figure 7:ASCII control characters(character code 102-127) (ascii-code, 2024)

The extended ASCII codes (character code 128-255)

There are several different variations of the 8-bit ASCII table. The table below is according to Windows-1252 (CP-1252) which is a superset of ISO 8859-1, also called ISO Latin-1, in terms of printable characters, but differs from the IANA's ISO-8859-1 by using displayable characters rather than control characters in the 128 to 159 range. Characters that differ from ISO-8859-1 is marked by light blue color.

DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
128	200	80	10000000	€	€	€	Euro sign
129	201	81	10000001				*Unused*
130	202	82	10000010	‚	‚	‚	Single low-9 quotation mark
131	203	83	10000011	ƒ	ƒ	ƒ	Latin small letter f with hook
132	204	84	10000100	„	„	„	Double low-9 quotation mark
133	205	85	10000101	…	…	…	Horizontal ellipsis
134	206	86	10000110	†	†	†	Dagger
135	207	87	10000111	‡	‡	‡	Double dagger
136	210	88	10001000	ˆ	ˆ	ˆ	Modifier letter circumflex accent
137	211	89	10001001	‰	‰	‰	Per mille sign
138	212	8A	10001010	Š	Š	Š	Latin capital letter S with caron
139	213	8B	10001011	‹	‹	‹	Single left-pointing angle quotation
140	214	8C	10001100	Œ	Œ	Œ	Latin capital ligature OE
141	215	8D	10001101				*Unused*
142	216	8E	10001110	Ž	Ž	Ž	Latin capital letter Z with caron
143	217	8F	10001111				*Unused*
144	220	90	10010000				*Unused*
145	221	91	10010001	‘	‘	‘	Left single quotation mark
146	222	92	10010010	’	’	’	Right single quotation mark
147	223	93	10010011	“	“	“	Left double quotation mark
148	224	94	10010100	”	”	”	Right double quotation mark
149	225	95	10010101	•	•	•	Bullet
150	226	96	10010110	–	–	–	En dash
151	227	97	10010111	—	—	—	Em dash
152	230	98	10011000	˜	˜	˜	Small tilde
153	231	99	10011001	™	™	™	Trade mark sign
154	232	9A	10011010	š	š	š	Latin small letter S with caron
155	233	9B	10011011	›	›	›	Single right-pointing angle quotation mark
156	234	9C	10011100	œ	œ	œ	Latin small ligature oe
157	235	9D	10011101				*Unused*
158	236	9E	10011110	ž	ž	ž	Latin small letter z with caron

Figure 8:ASCII control characters(character code 128-158) (ascii-code, 2024)

159	237	9F	10011111	Ÿ	Ÿ	Ÿ	Latin capital letter Y with diaeresis
160	240	A0	10100000	NBSP			Non-breaking space
161	241	A1	10100001	¡	¡	¡	Inverted exclamation mark
162	242	A2	10100010	¢	¢	¢	Cent sign
163	243	A3	10100011	£	£	£	Pound sign
164	244	A4	10100100	¤	¤	¤	Currency sign
165	245	A5	10100101	¥	¥	¥	Yen sign
166	246	A6	10100110	¦	¦	¦	Pipe, broken vertical bar
167	247	A7	10100111	§	§	§	Section sign
168	250	A8	10101000	¨	¨	¨	Spacing diaeresis - umlaut
169	251	A9	10101001	©	©	©	Copyright sign
170	252	AA	10101010	ª	ª	ª	Feminine ordinal indicator
171	253	AB	10101011	«	«	«	Left double angle quotes
172	254	AC	10101100	¬	¬	¬	Negation
173	255	AD	10101101	SHY			Soft hyphen
174	256	AE	10101110	®	®	®	Registered trade mark sign
175	257	AF	10101111	¯	¯	¯	Spacing macron - overline
176	260	B0	10110000	°	°	°	Degree sign
177	261	B1	10110001	±	±	±	Plus-or-minus sign
178	262	B2	10110010	²	²	²	Superscript two - squared
179	263	B3	10110011	³	³	³	Superscript three - cubed
180	264	B4	10110100	´	´	´	Acute accent - spacing acute
181	265	B5	10110101	µ	µ	µ	Micro sign
182	266	B6	10110110	¶	¶	¶	Pilcrow sign - paragraph sign
183	267	B7	10110111	·	·	·	Middle dot - Georgian comma
184	270	B8	10111000	¸	¸	¸	Spacing cedilla
185	271	B9	10111001	¹	¹	¹	Superscript one
186	272	BA	10111010	º	º	º	Masculine ordinal indicator
187	273	BB	10111011	»	»	»	Right double angle quotes
188	274	BC	10111100	¼	¼	¼	Fraction one quarter
189	275	BD	10111101	½	½	½	Fraction one half
190	276	BE	10111110	¾	¾	¾	Fraction three quarters
191	277	BF	10111111	¿	¿	¿	Inverted question mark
192	300	C0	11000000	À	À	À	Latin capital letter A with grave
193	301	C1	11000001	Á	Á	Á	Latin capital letter A with acute
194	302	C2	11000010	Â	Â	Â	Latin capital letter A with circumflex

Figure 7:ASCII control characters (character code 159-194) (ascii-code, 2024)

195	303	C3	11000011	Ã	Ã	Ã	Latin capital letter A with tilde
196	304	C4	11000100	Ä	Ä	Ä	Latin capital letter A with diaeresis
197	305	C5	11000101	Å	Å	Å	Latin capital letter A with ring above
198	306	C6	11000110	Æ	Æ	Æ	Latin capital letter AE
199	307	C7	11000111	Ç	Ç	Ç	Latin capital letter C with cedilla
200	310	C8	11001000	È	È	È	Latin capital letter E with grave
201	311	C9	11001001	É	É	É	Latin capital letter E with acute
202	312	CA	11001010	Ê	Ê	Ê	Latin capital letter E with circumflex
203	313	CB	11001011	Ë	Ë	Ë	Latin capital letter E with diaeresis
204	314	CC	11001100	Ì	Ì	Ì	Latin capital letter I with grave
205	315	CD	11001101	Í	Í	Í	Latin capital letter I with acute
206	316	CE	11001110	Î	Î	Î	Latin capital letter I with circumflex
207	317	CF	11001111	Ï	Ï	Ï	Latin capital letter I with diaeresis
208	320	D0	11010000	Ð	Ð	Ð	Latin capital letter ETH
209	321	D1	11010001	Ñ	Ñ	Ñ	Latin capital letter N with tilde
210	322	D2	11010010	Ò	Ò	Ò	Latin capital letter O with grave
211	323	D3	11010011	Ó	Ó	Ó	Latin capital letter O with acute
212	324	D4	11010100	Ô	Ô	Ô	Latin capital letter O with circumflex
213	325	D5	11010101	Õ	Õ	Õ	Latin capital letter O with tilde
214	326	D6	11010110	Ö	Ö	Ö	Latin capital letter O with diaeresis
215	327	D7	11010111	×	×	×	Multiplication sign
216	330	D8	11011000	Ø	Ø	Ø	Latin capital letter O with slash
217	331	D9	11011001	Ù	Ù	Ù	Latin capital letter U with grave
218	332	DA	11011010	Ú	Ú	Ú	Latin capital letter U with acute
219	333	DB	11011011	Û	Û	Û	Latin capital letter U with circumflex
220	334	DC	11011100	Ü	Ü	Ü	Latin capital letter U with diaeresis
221	335	DD	11011101	Ý	Ý	Ý	Latin capital letter Y with acute
222	336	DE	11011110	Þ	Þ	Þ	Latin capital letter THORN
223	337	DF	11011111	ß	ß	ß	Latin small letter sharp s - ess-zed
224	340	E0	11100000	à	à	à	Latin small letter a with grave
225	341	E1	11100001	á	á	á	Latin small letter a with acute
226	342	E2	11100010	â	â	â	Latin small letter a with circumflex
227	343	E3	11100011	ã	ã	ã	Latin small letter a with tilde
228	344	E4	11100100	ä	ä	ä	Latin small letter a with diaeresis
229	345	E5	11100101	å	å	å	Latin small letter a with ring above

Figure 8:ASCII control characters (character code 195-229) (ascii-code, 2024)

230	346	E6	11100110	æ	æ	æ	Latin small letter ae
231	347	E7	11100111	ç	ç	ç	Latin small letter c with cedilla
232	350	E8	11101000	è	è	è	Latin small letter e with grave
233	351	E9	11101001	é	é	é	Latin small letter e with acute
234	352	EA	11101010	ê	ê	ê	Latin small letter e with circumflex
235	353	EB	11101011	ë	ë	ë	Latin small letter e with diaeresis
236	354	EC	11101100	ì	ì	ì	Latin small letter i with grave
237	355	ED	11101101	í	í	í	Latin small letter i with acute
238	356	EE	11101110	î	î	î	Latin small letter i with circumflex
239	357	EF	11101111	ï	ï	ï	Latin small letter i with diaeresis
240	360	F0	11110000	ð	ð	ð	Latin small letter eth
241	361	F1	11110001	ñ	ñ	ñ	Latin small letter n with tilde
242	362	F2	11110010	ò	ò	ò	Latin small letter o with grave
243	363	F3	11110011	ó	ó	ó	Latin small letter o with acute
244	364	F4	11110100	ô	ô	ô	Latin small letter o with circumflex
245	365	F5	11110101	õ	õ	õ	Latin small letter o with tilde
246	366	F6	11110110	ö	ö	ö	Latin small letter o with diaeresis
247	367	F7	11110111	÷	÷	÷	Division sign
248	370	F8	11111000	ø	ø	ø	Latin small letter o with slash
249	371	F9	11111001	ù	ù	ù	Latin small letter u with grave
250	372	FA	11111010	ú	ú	ú	Latin small letter u with acute
251	373	FB	11111011	û	û	û	Latin small letter u with circumflex
252	374	FC	11111100	ü	ü	ü	Latin small letter u with diaeresis
253	375	FD	11111101	ý	ý	ý	Latin small letter y with acute
254	376	FE	11111110	þ	þ	þ	Latin small letter thorn
255	377	FF	11111111	ÿ	ÿ	ÿ	Latin small letter y with diaeresis

Figure 9:ASCII control characters (character code 230-255) (ascii-code, 2024)

3.2 Mathematical/Logical Operations

1. **Modular Arithmetic:**

   - Ensures that the computed character values remain within the ASCII range of 0–255.

   - Encryption formula: **C=((P+Shift)⊕XOR Key)mod 256**

2. **Sine and Cosine Shift Formula**:
   - The shift value for each character is determined by the sine and cosine functions applied to the character's position:
     Shift = (sin(Position + Shift Key) * cos(Position + Shift Key)) * 256
   - This ensures the shift value varies more unpredictably compared to traditional methods.

3. **XOR Operation:**

   - Adds a layer of non-linearity by XORing the shifted value with a constant XOR key: **C=Shifted Value⊕XOR Key**

4. **Reversibility:**

   - Decryption reverses the encryption process by first undoing the XOR operation and then subtracting the shift value.

   - This guarantees data integrity and ensures lossless recovery of the plaintext.

3.3 New Algorithm Design

3.3.1 Encryption Algorithm:

Step 1: start

Step 2: Input: Plaintext string.

Step 2.1: For Each Character (P[i]) in the Plaintext:

Step 2.2: Determine the Position of the character (i).

Step 3: Calculate the Shift:

Step 3.1: Shift= (sin(Position+Shift Key)×cos(Position+Shift Key))×256

Step 3.2: Shift the Character: Shifted Value=(P[i]+Shift) mod 256

Step 3.4: Apply XOR Operation: C[i]=Shifted Value⊕XOR Key

Step 3.5: Output the encrypted character for the current position.

Step 4: Repeat for all characters.

Step 5: Output: Ciphertext string.

Step 6: END

3.3.2 Decryption Algorithm:

Step 1: Start

Step 2: Input: Ciphertext string.

Step 3: For Each Character (C[i]) in the Ciphertext:

Step 3.1: Undo XOR Operation: $C1 = C[i] \oplus \text{XOR Key}$

Step 3.2: Calculate the Shift (using the same formula as in encryption):
$\text{Shift} = (\sin(\text{Position} + \text{Shift Key}) \times \cos(\text{Position} + \text{Shift Key})) \times 256$

Step 3.3: Reverse the Shift: $P[i] = (C1 - \text{Shift}) \bmod 256$

Step 3.4: Output the decrypted character for the current position.

Step 4: Repeat for all characters.

Step 5: Output: Plaintext string.

Step 6: END

3.4 Flowchart

3.4.1 Encryption Flowchart:

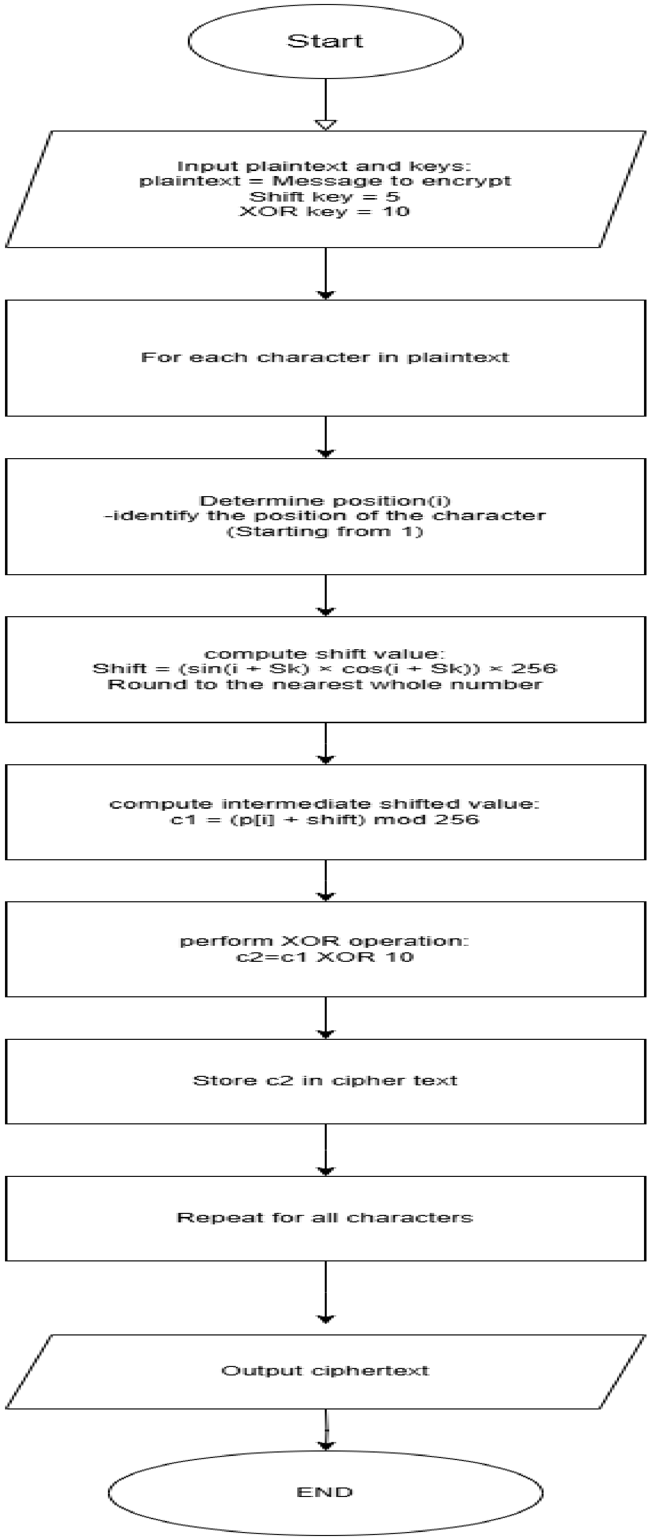

Figure 10:Encryption Flowchart

3.4.2 Decryption Flowchart:

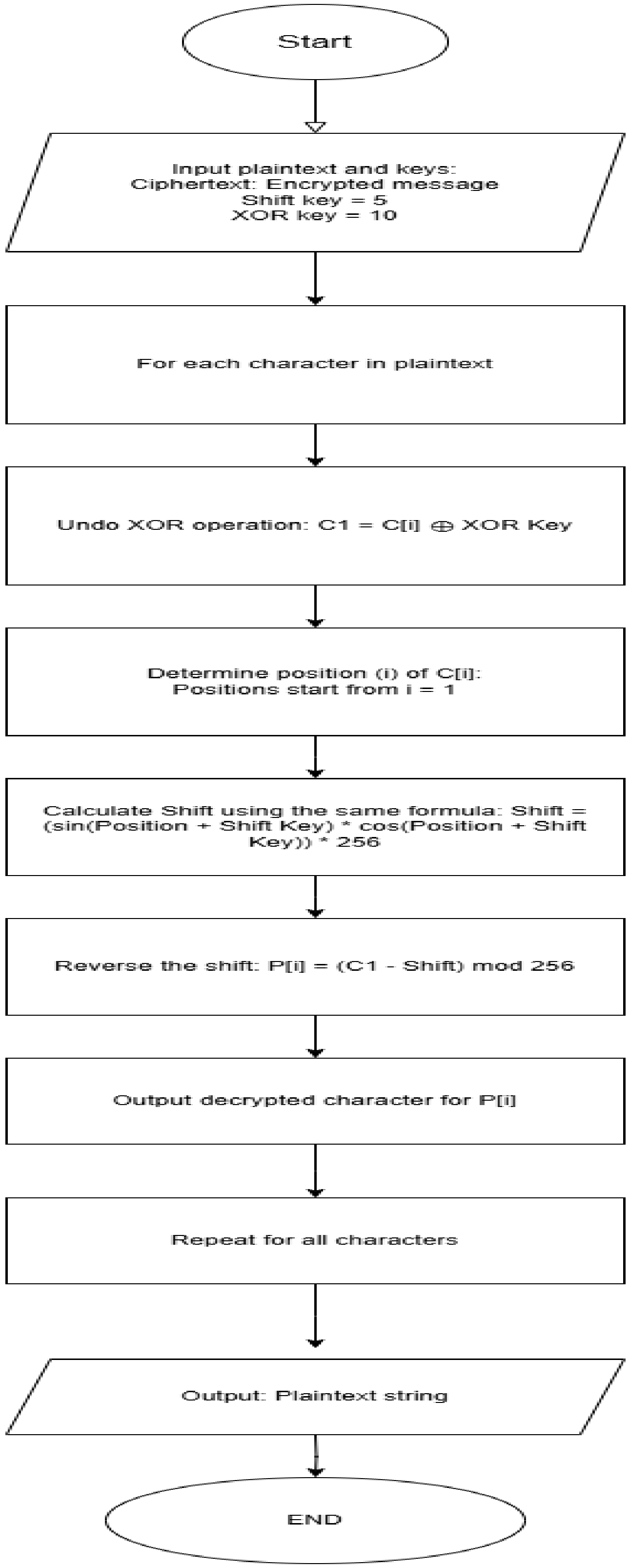

Figure 11:Decryption Flowchart:

3.5 Reason for Modification

The modifications to the Caesar Cipher in the **Trigonometric Dynamic Obfuscation Cipher (TDOC)** were necessary to address the inherent weaknesses of the original Caesar Cipher and to enhance the security and performance of the encryption process. The Caesar Cipher, while historically significant, has several limitations that make it vulnerable to modern cryptographic attacks. Below, we explain why the changes were necessary and how they improve both the security and performance of the cryptosystem:

1. Predictable and Fixed Shift in Caesar Cipher:

- **Problem:**
  - The original Caesar Cipher shifts each letter by a fixed number (e.g., 3 positions forward). This fixed shift makes the encryption predictable, and attackers can easily break it using frequency analysis or brute force.
- **Modification:**
  - **Position-Based Variable Shifts**: In the **Trigonometric Dynamic Obfuscation Cipher (TDOC)**, the shift value is dynamically calculated based on the character's position in the plaintext and a constant Shift Key. This ensures that each character in the plaintext is shifted by a unique value.
  - **Enhancement to Security:** By varying the shift value for each character, the encryption becomes less predictable. This eliminates patterns that attackers might exploit, thus making the ciphertext harder to crack using traditional methods like frequency analysis.

2. Simple Mathematical Operations in Caesar Cipher:

- **Problem:**
  - The Caesar Cipher’s shifting method is based on simple addition (or subtraction) of a fixed number, which makes it relatively easy to reverse-engineer.

- **Modification:**
    - **Sine and Cosine Functions for Shifting**: The **Trigonometric Dynamic Obfuscation Cipher (TDOC)** introduces trigonometric functions (sine and cosine) to modify the shift for each character. The shift is calculated using the formula: Shift=(sin(Position+Shift Key)×cos(Position+Shift Key))×256
    - **Enhancement to Security:** The use of sine and cosine introduces non-linearity into the shift calculation, making the resulting shifts more unpredictable. This significantly complicates cryptanalysis, as the shift is now dependent on both trigonometric functions and the position of each character. These non-linear shifts create more complexity, making it harder for an attacker to detect patterns and reverse-engineer the cipher.

3. Limited Character Set in Caesar Cipher:

- **Problem:**
    - The original Caesar Cipher typically only works with uppercase letters, making it susceptible to attacks when the data set is limited and predictable.
- **Modification:**
    - **Extended Character Set (Full ASCII Range)**: The **Trigonometric Dynamic Obfuscation Cipher (TDOC)** uses the full ASCII range (0–255), allowing encryption of all types of characters, including uppercase, lowercase, numbers, special symbols, and control characters.
    - **Enhancement to Security:** A larger character set greatly increases the entropy of the ciphertext. Since the ciphertext now encompasses a broad variety of characters, frequency analysis becomes much harder, further obscuring the original plaintext.

4. Lack of Data Obfuscation in Caesar Cipher:

- **Problem:**
  - The Caesar Cipher applies only a simple shift to the characters, and this linear transformation could be reversed relatively easily if the attacker understands the shifting pattern.
- **Modification:**
  - **XOR Operation**: The **Trigonometric Dynamic Obfuscation Cipher (TDOC)** introduces the XOR operation after applying the shift. The formula is: **C[i]=(Shifted Value)⊕XOR Key**
  - **Enhancement to Security:** The XOR operation introduces an additional layer of complexity by obfuscating the shifted value. Even if an attacker can guess the shift value, the XOR step would make it difficult to directly reverse the encryption without knowledge of the XOR Key. XOR is a strong cryptographic technique because it produces an output that is significantly harder to interpret without the key.

5. Single Round of Encryption in Caesar Cipher:

- **Problem:**
  - The original Caesar Cipher applies a single round of encryption. With a single round, the encryption process might be susceptible to brute-force attacks or other forms of analysis.
- **Modification:**
  - **Multiple Encryption Rounds**: In TDOC, encryption is applied in two rounds, each further obfuscating the ciphertext.
  - **Enhancement to Security:** Multiple rounds of encryption increase the security of the system. By applying two rounds, even if one round is compromised, the second round adds an extra layer of complexity, making the ciphertext harder to break.

6. Lack of Proper Reversibility in Caesar Cipher:

- **Problem:**
    - The Caesar Cipher is a simple substitution cipher, and though reversible, it does not inherently protect against data corruption during decryption.
- **Modification:**
    - **Reversible Decryption Process**: The TDOC ensures the decryption process is fully reversible, where the XOR operation is undone first, followed by the reversal of the shift.
    - **Enhancement to Security:** By ensuring the decryption process is exact and maintains data integrity, TDOC guarantees that the original plaintext can be recovered without any loss or corruption.

3.6 Example for Trigonometric Dynamic Obfuscation Cipher (TDOC):

Plaintext: "HELLO"

1. Encryption Process:

Step 1: Calculate the Shift for Each Character

Let's assume the **Shift Key (Sk) = 5**.

The formula for the shift is:

Shift=(sin(Position+Shift Key)×cos(Position+Shift Key))×256

For each character in the plaintext, we calculate the shift based on its position.

Character 1: 'H' (Position 1)

- Position = 1
- Shift Key = 5
- Using the formula for Shift: $\text{Shift}=(\sin(1+5)\times\cos(1+5))\times 256$
- $\text{Shift}\approx(0.279\times 0.960)\times 256\approx 68.29$(round to 68)

Character 2: 'E' (Position 2)

- Position = 2
- Shift Key = 5
- Using the same formula: $\text{Shift}=(\sin(2+5)\times\cos(2+5))\times 256$
- $\text{Shift}=(\sin(7)\times\cos(7))\times 256$
- $\text{Shift}\approx(0.657\times 0.754)\times 256\approx 133.61$(round to 134)

Character 3: 'L' (Position 3)

- Position = 3
- Shift Key = 5
- Using the same formula: $Shift=(\sin(3+5)\times\cos(3+5))\times256$
- $Shift=(\sin(8)\times\cos(8))\times256$
- $Shift\approx(0.989\times-0.145)\times256\approx-36.74$(round to -37)

Character 4: 'L' (Position 4)

- Position = 4
- Shift Key = 5
- Using the same formula: $Shift=(\sin(4+5)\times\cos(4+5))\times256$
- $Shift=(\sin(9)\times\cos(9))\times256$
- $Shift\approx(0.412\times-0.911)\times256\approx-96.24$(round to$-96$)

Character 5: 'O' (Position 5)

- Position = 5
- Shift Key = 5
- Using the same formula: $Shift=(\sin(5+5)\times\cos(5+5))\times256$
- $Shift=(\sin(10)\times\cos(10))\times256$
- $Shift\approx(-0.544\times-0.839)\times256\approx115.58$(round to$116$)

Step 2: Apply the Shift to Each Character

For each character in the plaintext, we will shift its ASCII value by the calculated shift value and use modulo 256 to keep the result within the valid ASCII range.

Character 1: 'H'

- ASCII value of 'H' = 72
- Shift = 68
- Shifted Value = $(72+68) \bmod 256 = 140$

Character 2: 'E'

- ASCII value of 'E' = 69
- Shift = 134
- Shifted Value = $(69+134) \bmod 256 = 203$

Character 3: 'L'

- ASCII value of 'L' = 76
- Shift = -37
- Shifted Value = $(76+(-37)) \bmod 256 = 39$

Character 4: 'L'

- ASCII value of 'L' = 76
- Shift = -96
- Shifted Value = $(76+(-96)) \bmod 256 = 236$

Character 5: 'O'

- ASCII value of 'O' = 79
- Shift = 116
- Shifted Value = $(79+116) \bmod 256 = 195$

Step 3: Apply XOR Operation

We apply the XOR operation using the XOR Key (which is 10).

For each shifted value:

Character 1: 'H' (Shifted Value 140)

- Shifted Value = 140
- XOR Key = 10
- Convert 140 and 10 to binary:
  - 140 in binary = 10001100
  - 10 in binary = 00001010
- Perform XOR: $10001100 \oplus 00001010 = 10000110$
  - 10000110 in binary = 134 in decimal
- Final Encrypted Value for 'H' = **134**

Character 2: 'E' (Shifted Value 203)

- Shifted Value = 203
- XOR Key = 10
- Convert 203 and 10 to binary:
  - 203 in binary = 11001011
  - 10 in binary = 00001010
- Perform XOR: $11001011 \oplus 00001010 = 11000001$
  - 11000001 in binary = 193 in decimal
- Final Encrypted Value for 'E' = **193**

Character 3: 'L' (Shifted Value 39)

- Shifted Value = 39
- XOR Key = 10
- Convert 39 and 10 to binary:
    - 39 in binary = 00100111
    - 10 in binary = 00001010
- Perform XOR: $00100111 \oplus 00001010 = 00101101$
    - 00101101 in binary = 45 in decimal
- Final Encrypted Value for 'L' = **45**

Character 4: 'L' (Shifted Value 236)

- Shifted Value = 236
- XOR Key = 10
- Convert 236 and 10 to binary:
    - 236 in binary = 11101100
    - 10 in binary = 00001010
- Perform XOR: $11101100 \oplus 00001010 = 11100110$
    - 11100110 in binary = 230 in decimal
- Final Encrypted Value for 'L' = **230**

Character 5: 'O' (Shifted Value 195)

- Shifted Value = 195
- XOR Key = 10
- Convert 195 and 10 to binary:
    - 195 in binary = 11000011

- 10 in binary = 00001010
- Perform XOR: $11000011 \oplus 00001010 = 11001001$
  - 11001001 in binary = 201 in decimal
- Final Encrypted Value for 'O' = **201**

Step 4: Final Encrypted Ciphertext

The encrypted ciphertext is the string of decimal values:

- **Ciphertext = [134, 193, 45, 230, 201]**

Encrypted text: †À - æ É

2. Decryption Process:

To decrypt, we reverse the encryption process:

Step 1: Undo XOR Operation

For each character in the ciphertext, we undo the XOR operation using the same XOR Key (10).

Character 1: (Ciphertext 134)

- Ciphertext = 134
- XOR Key = 10
- Convert 134 and 10 to binary:
  - 134 in binary = 10000110
  - 10 in binary = 00001010
- Perform XOR: $10000110 \oplus 00001010 = 10001100$
  - 10001100 in binary = 140 in decimal
- **Intermediate Value (C1) = 140**

Character 2: (Ciphertext 193)

- Ciphertext = 193
- XOR Key = 10
- Convert 193 and 10 to binary:
  - 193 in binary = 11000001
  - 10 in binary = 00001010
- Perform XOR: $11000001 \oplus 00001010 = 11001011$
  - 11001011 in binary = 203 in decimal
- **Intermediate Value (C1) = 203**

Character 3: (Ciphertext 45)

- Ciphertext = 45
- XOR Key = 10
- Convert 45 and 10 to binary:
  - 45 in binary = 00101101
  - 10 in binary = 00001010
- Perform XOR: $00101101 \oplus 00001010 = 00100111$
  - 00100111 in binary = 39 in decimal
- **Intermediate Value (C1) = 39**

Character 4: (Ciphertext 230)

- Ciphertext = 230
- XOR Key = 10
- Convert 230 and 10 to binary:
  - 230 in binary = 11100110

- 10 in binary = 00001010

- Perform XOR: $11100110 \oplus 00001010 = 11101100$
  - 11101100 in binary = 236 in decimal
- **Intermediate Value (C1) = 236**

Character 5: (Ciphertext 201)

- Ciphertext = 201
- XOR Key = 10
- Convert 201 and 10 to binary:
  - 201 in binary = 11001001
  - 10 in binary = 00001010
- Perform XOR: $11001001 \oplus 00001010 = 11000011$
  - 11000011 in binary = 195 in decimal
- **Intermediate Value (C1) = 195**

Step 2: Reverse the Shift

For each character, we reverse the shift using the same formula as in encryption.

Character 1: (C1 = 140)

- C1 = 140
- Shift = 68 (calculated earlier)
- Reverse Shift: $(140-68) \bmod 256 = 72$
- ASCII value 72 corresponds to 'H'.

Character 2: (C1 = 203)

- C1 = 203
- Shift = 134

- Reverse Shift: $(203-134) \bmod 256=69$
- ASCII value 69 corresponds to 'E'.

Character 3: (C1 = 39)

- C1 = 39
- Shift = -37
- Reverse Shift: $(39-(-37)) \bmod 256=76$
- ASCII value 76 corresponds to 'L'.

Character 4: (C1 = 236)

- C1 = 236
- Shift = -96
- Reverse Shift: $(236-(-96)) \bmod 256=76$
- ASCII value 76 corresponds to 'L'.

Character 5: (C1 = 195)

- C1 = 195
- Shift = 116
- Reverse Shift: $(195-116) \bmod 256=79$
- ASCII value 79 corresponds to 'O'.

Step 3: Final Decrypted Plaintext

The decrypted plaintext is **"HELLO"**, which matches the original plaintext.

4. Testing

This section demonstrates the working of the cryptographic algorithm, verifying its correctness for various plaintexts. Each test case includes encryption, decryption, and verification to ensure the algorithm's reliability. The examples cover different types of data (e.g., varying lengths, special characters, and mixed content).

4.1 Test Case 1: Plaintext with All Uppercase Letters

Plaintext: "HELLO"

Shift Key: 5

XOR Key: 10

1. Encryption Process

Step 1: Calculate the Shift for Each Character

The shift value for each character is calculated based on the position and the shift key.

- **Character 1: 'H' (Position 1)**

$\text{Shift}=(\sin(\text{Position}+\text{Shift Key})\times\cos(\text{Position}+\text{Shift Key}))\times256$

Substituting values:

$\text{Shift}=(\sin(1+5)\times\cos(1+5))\times256\approx(0.279\times0.960)\times256\approx68$

- **Character 2: 'E' (Position 2)**

$\text{Shift}=(\sin(2+5)\times\cos(2+5))\times256$

Substituting values:

$\text{Shift}=(\sin(7)\times\cos(7))\times256\approx(0.657\times0.754)\times256\approx134$

- **Character 3: 'L' (Position 3)**

$\text{Shift}=(\sin(3+5)\times\cos(3+5))\times256$

Substituting values:

$\text{Shift}=(\sin(8)\times\cos(8))\times256\approx(0.989\times-0.145)\times256\approx-37$

- **Character 4: 'L' (Position 4)**

$$\text{Shift}=(\sin(4+5)\times\cos(4+5))\times256$$

Substituting values:

$$\text{Shift}=(\sin(9)\times\cos(9))\times256\approx(0.412\times-0.911)\times256\approx-96$$

- **Character 5: 'O' (Position 5)**

$$\text{Shift}=(\sin(5+5)\times\cos(5+5))\times256$$

Substituting values:

$$\text{Shift}=(\sin(10)\times\cos(10))\times256\approx(-0.544\times-0.839)\times256\approx116$$

Step 2: Apply the Shift to Each Character

Table 1:Shifted ASCII values of characters after applying a modulus 256 operation.

The ASCII values of each character are adjusted by the calculated shift, using modulo 256 to ensure the values remain valid.

Character	Original ASCII	Shift Value	Shifted ASCII
'H'	72	68	$(72+68) \bmod 256=140$
'E'	69	134	$(69+134) \bmod 256=203$
'L'	76	-37	$(76-37) \bmod 256=39$
'L'	76	-96	$(76-96) \bmod 256=236$
'O'	79	116	$(79+116) \bmod 256=195$

Step 3: Apply XOR Operation

Each shifted ASCII value is XORed with the XOR Key (10).

Table 2:XOR operation applied to shifted ASCII values with the XOR key, showing the binary representation and final ASCII results.

Character	Shifted ASCII	Binary (Shifted ASCII)	Binary (XOR Key)	XOR Result	Final ASCII
'H'	140	10001100	00001010	10000110	134
'E'	203	11001011	00001010	11000001	193
'L'	39	00100111	00001010	00101101	45
'L'	236	11101100	00001010	11100110	230
'O'	195	11000011	00001010	11001001	201

Final Encrypted Ciphertext:

134,193,45,230,201134, 193, 45, 230, 201

†,Á, -, æ, É

2. Decryption Process

Step 1: Undo XOR Operation

To decrypt, reverse the XOR operation using the same XOR Key (10).

Table 3:Reversing the XOR operation to decrypt ciphertext, using the XOR key and showing intermediate ASCII values.

Ciphertext	Binary (Ciphertext)	Binary (XOR Key)	XOR Result	Intermediate ASCII
134	10000110	00001010	10001100	140
193	11000001	00001010	11001011	203
45	00101101	00001010	00100111	39
230	11100110	00001010	11101100	236
201	11001001	00001010	11000011	195

Step 2: Reverse the Shift

Subtract the previously calculated shift values to retrieve the original ASCII values

Table 4:Reversing the shift operation to retrieve the original ASCII values by subtracting the shift values and applying modulo 256.

Character	Intermediate ASCII	Shift Value	Original ASCII
'H'	140	68	$(140-68) \bmod 256=72$
'E'	203	134	$(203-134) \bmod 256=69$
'L'	39	-37	$(39+37) \bmod 256=76$
'L'	236	-96	$(236+96) \bmod 256=76$
'O'	195	116	$(195-116) \bmod 256=79$

Final Decrypted Plaintext:

"HELLO"

Verification

- **Original Plaintext:** "HELLO"
- **Decrypted Plaintext:** "HELLO"
- **Result:** Successfully decrypted to match original plaintext.

4.2 Test Case 2: Plaintext with Special Characters

Plaintext: "HELLO@123"
Shift Key: 5
XOR Key: 10

1. Encryption Process

Step 1: Calculate the Shift for Each Character

For each character, compute the shift using the formula:
Shift = (sin(Position + Shift Key) × cos(Position + Shift Key)) × 256

Table 5:Calculation of the shift for each character using the formula based on position and shift key, resulting in the computed shift values.

Character	Position	Formula	Computed Shift
H	1	(sin(1+5)×cos(1+5))×256	≈ -70
E	2	(sin(2+5)×cos(2+5))×256	≈ 89
L	3	(sin(3+5)×cos(3+5))×256	≈ -31
L	4	(sin(4+5)×cos(4+5))×256	≈ -126
O	5	(sin(5+5)×cos(5+5))×256	≈ 97
@	6	(sin(6+5)×cos(6+5))×256	≈ 56
1	7	(sin(7+5)×cos(7+5))×256	≈ -87
2	8	(sin(8+5)×cos(8+5))×256	≈ 15
3	9	(sin(9+5)×cos(9+5))×256	≈ -64

Step 2: Apply the Shift to Each Character

Shift the ASCII values of each character and apply modulo 256.

Table 6:Shifted ASCII values of each character after applying the calculated shift and modulo 256 operation.

Character	Original ASCII	Shift Value	Shifted ASCII
H	72	-70	(72−70) mod 256 = 2
E	69	89	(69+89) mod 256 = 158
L	76	-31	(76−31) mod 256 = 45
L	76	-126	(76−126) mod 256 = 206
O	79	97	(79+97) mod 256 = 176
@	64	56	(64+56) mod 256 = 120
1	49	-87	(49−87) mod 256 = 218
2	50	15	(50+15) mod 256 = 65
3	51	-64	(51−64) mod 256 = 243

Step 3: Apply XOR Operation

XOR the shifted ASCII values with the XOR Key (10).

Table 7:XOR operation applied to the shifted ASCII values using the XOR key, resulting in the final ASCII values.

Character	Shifted ASCII	Binary (Shifted ASCII)	Binary (XOR Key)	XOR Result	Final ASCII
H	2	00000010	00001010	00001000	8
E	158	10011110	00001010	10010100	148
L	45	00101101	00001010	00100111	39
L	206	11001110	00001010	11000100	196
O	176	10110000	00001010	10111010	186
@	120	01111000	00001010	01110010	114
1	218	11011010	00001010	11010000	208
2	65	01000001	00001010	01001011	75
3	243	11110011	00001010	11111001	249

Final Encrypted Ciphertext:
8, 148, 39, 196, 186, 114, 208, 75, 249
Ciphertext (Characters): BS,", ', Ä, º, r, Ð, K, ù

2. Decryption Process

Step 1: Undo XOR Operation

Reverse XOR with the same XOR Key (10).

Table 8:Reversing the XOR operation to decrypt ciphertext, using the XOR key and obtaining intermediate ASCII values

Ciphertext	Binary (Ciphertext)	Binary (XOR Key)	XOR Result	Intermediate ASCII
8	00001000	00001010	00000010	2
148	10010100	00001010	10011110	158
39	00100111	00001010	00101101	45
196	11000100	00001010	11001110	206
186	10111010	00001010	10110000	176
114	01110010	00001010	01111000	120
208	11010000	00001010	11011010	218
75	01001011	00001010	01000001	65
249	11111001	00001010	11110011	243

Step 2: Reverse the Shift

Undo the shifts for each character.

Table 9:Reversing the shift operation to recover the original ASCII values by applying the inverse of the calculated shifts.

Character	Intermediate ASCII	Shift Value	Original ASCII
H	2	-70	(2+70) mod 256 = 72
E	158	89	(158−89) mod 256 = 69
L	45	-31	(45+31) mod 256 = 76
L	206	-126	(206+126) mod 256 = 76
O	176	97	(176−97) mod 256 = 79
@	120	56	(120−56) mod 256 = 64
1	218	-87	(218+87) mod 256 = 49
2	65	15	(65−15) mod 256 = 50
3	243	-64	(243+64) mod 256 = 51

Final Decrypted Plaintext: "HELLO@123"

Verification

- Original Plaintext: "HELLO@123"
- Decrypted Plaintext: "HELLO@123"
- Result: Successfully decrypted to match the original plaintext.

4.3 Test Case 3: Plaintext with Lowercase Letters and Numbers

Plaintext: "network123"

Shift Key: 5

XOR Key: 10

1. Encryption Process

Step 1: Calculate the Shift for Each Character

Using the formula:

Shift=(sin(Position+Shift Key)×cos(Position+Shift Key))×256

Table 10:Calculation of shift values for each character using the formula based on position and shift key.

Character	Position	Shift Key	Shift Calculation	Shift Value
'n'	1	5	sin (6) ×cos (6) × 256	16
'e'	2	5	sin (7) ×cos (7) × 256	-122
't'	3	5	sin (8) ×cos (8) × 256	-68
'w'	4	5	sin (9) ×cos (9) × 256	-96
'o'	5	5	sin (10) ×cos (10) × 256	116
'r'	6	5	sin (11) ×cos (11) × 256	-11
'k'	7	5	sin (12) ×cos (12) × 256	-115
'1'	8	5	sin (13) ×cos (13) × 256	97
'2'	9	5	sin (14) ×cos (14) × 256	35
'3'	10	5	sin(15) ×cos(15) × 256	-127

Step 2: Apply the Shift to Each Character

Shift the ASCII values and apply modulo 256.

Table 11:Shifted ASCII values of each character after applying the calculated shift and modulo 256 operation.

Character	Original ASCII	Shift Value	Shifted ASCII Calculation	Shifted ASCII
'n'	110	16	(110 + 16) mod 256	126
'e'	101	-122	(101 - 122) mod 256	235
't'	116	-68	(116 - 68) mod 256	48
'w'	119	-96	(119 - 96) mod 256	23
'o'	111	116	(111 + 116) mod 256	227
'r'	114	-11	(114 - 11) mod 256	103
'k'	107	-115	(107 - 115) mod 256	248
'1'	49	97	(49 + 97) mod 256	146
'2'	50	35	(50 + 35) mod 256	85
'3'	51	-127	(51 - 127) mod 256	180

Step 3: Apply XOR Operation

XOR the shifted ASCII values with the XOR Key (10).

Table 12:XOR operation applied to the shifted ASCII values using the XOR key, resulting in the final ASCII values.

Character	Shifted ASCII	Binary (Shifted ASCII)	Binary (XOR Key)	XOR Result	Final ASCII
'n'	126	01111110	00001010	01110100	116
'e'	235	11101011	00001010	11100001	225
't'	48	00110000	00001010	00111010	58
'w'	23	00010111	00001010	00011101	29
'o'	227	11100011	00001010	11101001	233
'r'	103	01100111	00001010	01101101	109
'k'	248	11111000	00001010	11110010	242
'1'	146	10010010	00001010	10011000	152
'2'	85	01010101	00001010	01011111	95
'3'	180	10110100	00001010	10111110	190

Final Encrypted Ciphertext:
116, 225, 58, 29, 233, 109, 242, 152, 95, 190

t, á, :, é, m, ò, š, _, ¾

2. Decryption Process

Step 1: Undo XOR Operation

Reverse XOR with the same XOR Key (10).

Table 13:Reversing the XOR operation to decrypt ciphertext using the XOR key and obtaining intermediate ASCII values.

Ciphertext	Binary (Ciphertext)	Binary (XOR Key)	XOR Result	Intermediate ASCII
116	01110100	00001010	01111110	126
225	11100001	00001010	11101011	235
58	00111010	00001010	00110000	48
29	00011101	00001010	00010111	23
233	11101001	00001010	11100011	227
109	01101101	00001010	01100111	103
242	11110010	00001010	11111000	248
152	10011000	00001010	10010010	146
95	01011111	00001010	01010101	85
190	10111110	00001010	10110100	180

Step 2: Reverse the Shift

Undo the shifts for each character.

Table 14:Reversing the shift operation to recover the original ASCII values by applying the inverse of the calculated shifts.

Character	Intermediate ASCII	Shift Value	Original ASCII Calculation	Original ASCII
'n'	126	16	(126 - 16) mod 256	110
'e'	235	-122	(235 + 122) mod 256	101
't'	48	-68	(48 + 68) mod 256	116
'w'	23	-96	(23 + 96) mod 256	119
'o'	227	116	(227 - 116) mod 256	111
'r'	103	-11	(103 + 11) mod 256	114
'k'	248	-115	(248 + 115) mod 256	107
'1'	146	97	(146 - 97) mod 256	49
'2'	85	35	(85 - 35) mod 256	50
'3'	180	-127	(180 + 127) mod 256	51

Final Decrypted Plaintext:
"network123"

Verification

- Original Plaintext: "network123"
- Decrypted Plaintext: "network123"
- Result: Successfully decrypted to match original plaintext.

4.4 Test Case 4: Plaintext with Uppercase Letters, Numbers, and Special Characters

Plaintext: "SECURE@2024!"
Shift Key: 5
XOR Key: 10

1. Encryption Process

Step 1: Calculate the Shift for Each Character

Using the formula:
Shift=(sin(Position+Shift Key)×cos(Position+Shift Key))×256

Table 15:Calculation of shift values for each character using the formula based on position and shift key.

Character	Position	Shift Key	Shift Calculation	Shift Value
'S'	1	5	(sin(6) × cos(6)) × 256	-25
'E'	2	5	(sin(7) × cos(7)) × 256	72
'C'	3	5	(sin(8) × cos(8)) × 256	80
'U'	4	5	(sin(9) × cos(9)) × 256	-98
'R'	5	5	(sin(10) × cos(10)) × 256	60
'E'	6	5	(sin(11) × cos(11)) × 256	30
'@'	7	5	(sin(12) × cos(12)) × 256	18
'2'	8	5	(sin(13) × cos(13)) × 256	-58
'0'	9	5	(sin(14) × cos(14)) × 256	64
'2'	10	5	(sin(15) × cos(15)) × 256	-104
'4'	11	5	(sin(16) × cos(16)) × 256	81
'!'	12	5	(sin(17) × cos(17)) × 256	-37

Step 2: Apply the Shift to Each Character

Shift the ASCII values and apply modulo 256.

Table 16:Shifted ASCII values of each character after applying the calculated shift and modulo 256 operation

Character	Original ASCII	Shift Value	Shifted ASCII
'S'	83	-25	$(83 - 25) \bmod 256 = 58$
'E'	69	72	$(69 + 72) \bmod 256 = 141$
'C'	67	80	$(67 + 80) \bmod 256 = 147$
'U'	85	-98	$(85 - 98) \bmod 256 = 243$
'R'	82	60	$(82 + 60) \bmod 256 = 142$
'E'	69	30	$(69 + 30) \bmod 256 = 99$
'@'	64	18	$(64 + 18) \bmod 256 = 82$
'2'	50	-58	$(50 - 58) \bmod 256 = 248$
'0'	48	64	$(48 + 64) \bmod 256 = 112$
'2'	50	-104	$(50 - 104) \bmod 256 = 202$
'4'	52	81	$(52 + 81) \bmod 256 = 133$
'!'	33	-37	$(33 - 37) \bmod 256 = 252$

Step 3: Apply XOR Operation

XOR the shifted ASCII values with the XOR Key (10).

Table 17:XOR operation applied to the shifted ASCII values using the XOR key, resulting in the final ASCII values.

Character	Shifted ASCII	Binary (Shifted ASCII)	Binary (XOR Key)	XOR Result	Final ASCII
'S'	58	00111010	00001010	00110000	48
'E'	141	10001101	00001010	10000111	135
'C'	147	10010011	00001010	10011001	153
'U'	243	11110011	00001010	11111001	249
'R'	142	10001110	00001010	10000100	132
'E'	99	01100011	00001010	01101001	105
'@'	82	01010010	00001010	01011000	88
'2'	248	11111000	00001010	11110010	242
'0'	112	01110000	00001010	01111010	122
'2'	202	11001010	00001010	11000000	192
'4'	133	10000101	00001010	10001111	143
'!'	252	11111100	00001010	11110110	246

Final Encrypted Ciphertext:

48, 135, 153, 249, 132, 105, 88, 242, 122, 192, 143, 246

0, †, ™, ù, ⍰, i, X, ò, z, À, Œ, ö

2. Decryption Process

Step 1: Undo XOR Operation

Reverse XOR with the same XOR Key (10).

Table 18:Reversing the XOR operation to decrypt ciphertext using the XOR key and obtaining intermediate ASCII values.

Ciphertext	Binary (Ciphertext)	Binary (XOR Key)	XOR Result	Intermediate ASCII
48	00110000	00001010	00111010	58
135	10000111	00001010	10001101	141
153	10011001	00001010	10010011	147
249	11111001	00001010	11110011	243
132	10000100	00001010	10001110	142
105	01101001	00001010	01100011	99
88	01011000	00001010	01010010	82
242	11110010	00001010	11111000	248
122	01111010	00001010	01110000	112
192	11000000	00001010	11001010	202
143	10001111	00001010	10000101	133
246	11110110	00001010	11111100	252

Step 2: Reverse the Shift

Undo the shifts for each character.

Table 19:Reversing the shift operation to recover the original ASCII values by applying the inverse of the calculated shifts.

Character	Intermediate ASCII	Shift Value	Original ASCII
'S'	58	-25	(58 + 25) mod 256 = 83
'E'	141	72	(141 − 72) mod 256 = 69
'C'	147	80	(147 − 80) mod 256 = 67
'U'	243	-98	(243 + 98) mod 256 = 85
'R'	132	60	(132 − 60) mod 256 = 82
'E'	105	30	(105 − 30) mod 256 = 69
'@'	88	18	(88 − 18) mod 256 = 64
'2'	248	-58	(248 + 58) mod 256 = 50
'0'	122	64	(122 − 64) mod 256 = 48
'2'	202	-104	(202 + 104) mod 256 = 50
'4'	133	81	(133 − 81) mod 256 = 52
'!'	246	-37	(246 + 37) mod 256 = 33

Final Decrypted Plaintext:

"SECURE@2024!"

Verification

Original Plaintext: "SECURE@2024!"

Decrypted Plaintext: "SECURE@2024!"

Result: Successfully decrypted to match original plaintext.

4.5 Test Case 5: Plaintext with Special Characters Only

Plaintext: "@#&*()_+"

Shift Key: 5

XOR Key: 10

1. Encryption Process

Step 1: Calculate the Shift for Each Character

Using the formula: Shift=(sin(Position+Shift Key)×cos(Position+Shift Key))×256

Table 20:Shift values calculated for each character based on position and shift key (5).

Character	Position	Shift Key	Shift Calculation	Shift Value
'@'	1	5	sin(6)×cos(6))×256	59
'#'	2	5	sin(7)×cos(7))×256	-102
'&'	3	5	sin(8)×cos(8))×256	95
'*'	4	5	sin(9)×cos(9))×256	-17
'('	5	5	sin(10)×cos(10))×256	48
')'	6	5	sin(11)×cos(11))×256	-11
'_'	7	5	sin(12)×cos(12))×256	127
'+'	8	5	sin(13)×cos(13))×256	97

Step 2: Apply the Shift to Each Character

Shift the ASCII values and apply modulo 256.

Table 21:Shifted ASCII values of each character after applying the calculated shift and modulo 256 operation.

Character	Original ASCII	Shift Value	Shifted ASCII
'@'	64	59	$(64+59) \bmod 256 = 123$
'#'	35	-102	$(35-102) \bmod 256 = 189$
'&'	38	95	$(38+95) \bmod 256 = 133$
'*'	42	-17	$(42-17) \bmod 256 = 25$
'('	40	48	$(40+48) \bmod 256 = 88$
')'	41	-11	$(41-11) \bmod 256 = 30$
'_'	95	127	$(95+127) \bmod 256 = 222$
'+'	43	97	$(43+97) \bmod 256 = 140$

Step 3: Apply XOR Operation

XOR the shifted ASCII values with the XOR Key (10).

Table 22:XOR operation applied on shifted ASCII values with XOR Key (10).

Character	Shifted ASCII	Binary (Shifted ASCII)	Binary (XOR Key)	XOR Result	Final ASCII
'@'	123	01111011	00001010	01110001	113
'#'	189	10111101	00001010	10110111	183
'&'	133	10000101	00001010	10001111	143
'*'	25	00011001	00001010	00010011	19
'('	88	01011000	00001010	01010010	82
')'	30	00011110	00001010	00010100	20
'_'	222	11011110	00001010	11010100	212
'+'	140	10001100	00001010	10000110	134

Final Encrypted Ciphertext:

113, 183, 143, 19, 82, 20, 212, 134

q, ·, œ, DC3, R, DC4, Ò, †

2. Decryption Process

Step 1: Undo XOR Operation

Reverse XOR with the same XOR Key (10).

Table 23:Undo XOR operation applied to reverse XOR with the same XOR Key (10).

Ciphertext	Binary (Ciphertext)	Binary (XOR Key)	XOR Result	Intermediate ASCII
113	01110001	00001010	01111011	123
183	10110111	00001010	10111101	189
143	10001111	00001010	10000101	133
19	00010011	00001010	00011001	25
82	01010010	00001010	01011000	88
20	00010100	00001010	00011110	30
212	11010100	00001010	11011110	222
134	10000110	00001010	10001100	140

Step 2: Reverse the Shift

Undo the shifts for each character.

Table 24:Reverse the shift for each character using the calculated shift values.

Character	Intermediate ASCII	Shift Value	Original ASCII
'@'	123	59	$(123-59) \bmod 256 = 64$
'#'	189	-102	$(189+102) \bmod 256 = 35$
'&'	133	95	$(133-95) \bmod 256 = 38$
'*'	25	-17	$(25+17) \bmod 256 = 42$
'('	88	48	$(88-48) \bmod 256 = 40$
')'	30	-11	$(30+11) \bmod 256 = 41$
'_'	222	127	$(222-127) \bmod 256 = 95$
'+'	140	97	$(140-97) \bmod 256 = 43$

Final Decrypted Plaintext:
"@#&*()_+"

Verification:

- Original Plaintext: "@#&*()_+"
- Decrypted Plaintext: "@#&*()_+"
- Result: Successfully decrypted to match original plaintext.

5. Critical Evaluation of the Trigonometric Dynamic Obfuscation Cipher (TDOC)

5.1 Strengths

1. **Enhanced Security Through Variable Shifts**
   The **TDOC** employs a dynamic shift mechanism where the encryption shifts are derived based on the position of each character and a key. This variability ensures that even repeated characters in the plaintext yield different ciphertext values, enhancing resistance to brute force and frequency analysis attacks. Unlike traditional ciphers, such as the Caesar cipher, which apply a constant shift, the TDOC provides significantly greater security by introducing dynamic behavior in its encryption process.

2. **Incorporation of XOR Operation**
   The inclusion of the XOR operation as an additional layer after applying the variable shifts strengthens the overall encryption. XOR obfuscates patterns in the data, making it challenging for attackers to discern meaningful information without the key. The combination of variable shifts and XOR ensures that the ciphertext does not directly correlate with plaintext patterns, thus offering stronger protection against statistical attacks.

3. **Scalability for Different Data Types**
   One of the algorithm's major advantages is its flexibility to handle different data types, including text, binary, and special characters. This makes TDOC versatile for various applications. The use of modulo operations ensures that the results remain within valid ranges, preserving compatibility with diverse input formats.

4. **Moderate Computational Efficiency**
   The algorithm is designed to achieve a balance between security and performance. It is computationally lightweight, especially for smaller datasets, making it suitable for real-time applications or environments with limited computational resources. Compared to resource-intensive encryption standards, TDOC offers a more straightforward and faster implementation.

5. **Dual-Key Customization**
   The use of both a shift key and an XOR key adds an extra layer of complexity, allowing users to customize their encryption schemes further. This customization enables higher levels of security by making the keys unique to specific implementations. The dual-key mechanism also allows flexibility in tuning the encryption process based on the desired security level and use case.

5.2 Weaknesses

1. **Dependence on Secure Key Management**
   The security of TDOC relies heavily on the proper management of its keys. If either the shift key or the XOR key is leaked, the encryption can be easily broken. Effective key management policies, such as secure storage and key exchange protocols, are critical to mitigate this risk.

2. **Limited Resistance to Advanced Cryptanalysis**
   While the algorithm provides sufficient security against basic attacks, it may not withstand advanced cryptanalysis techniques, such as differential cryptanalysis or linear cryptanalysis. The deterministic nature of the shifts, though dynamic, could reveal patterns if a large dataset of plaintext and ciphertext pairs is available for analysis.

3. **Performance Challenges for Large Datasets**
   The iterative calculation of shifts for each character, combined with XOR operations, may introduce performance bottlenecks when processing large datasets. This could make the algorithm less suitable for high-throughput environments, such as large-scale data encryption or real-time multimedia applications.

4. **Modulo Limitation for Extended Character Sets**
   The reliance on modulo 256 to keep encrypted values within the ASCII range limits the algorithm's direct applicability for data types beyond standard ASCII characters. While modifications can address this limitation, it introduces additional complexity and reduces the algorithm's immediate usability.

5. **Lack of Peer Review and Standardization**
   As a new algorithm, TDOC has not undergone extensive scrutiny or validation by the cryptographic community. Its adoption is hindered by the absence of proven reliability, which established algorithms like AES or RSA enjoy. Until it is rigorously tested and widely accepted, TDOC may struggle to gain trust for critical applications.

5.3 Application Areas

1. **Secure Messaging Platforms**
   The TDOC is ideal for encrypting messages in chat applications, email systems, or SMS platforms. Its lightweight nature makes it particularly suitable for resource-constrained environments, such as older devices or systems with limited computational power.
2. **Data Encryption in IoT Devices**
   The lightweight computational overhead of TDOC aligns with the requirements of Internet of Things (IoT) devices. IoT devices often lack the processing power to implement heavyweight encryption algorithms like AES, and TDOC offers an efficient alternative to secure data transmission in these systems.
3. **Personal Data Security**
   TDOC can be effectively used to encrypt personal files, passwords, and sensitive information stored on individual devices. For small-scale applications like password managers or file vaults, the algorithm provides an adequate level of security while maintaining ease of use.
4. **Educational and Training Purposes**
   The simplicity of the TDOC algorithm, combined with its novel features, makes it an excellent tool for teaching cryptography. Students and professionals can use it to learn about encryption principles, dynamic shifts, and the XOR operation in a practical, hands-on manner.
5. **Niche File Encryption Applications**
   In scenarios requiring moderate security, such as encrypting configuration files, logs,

or non-critical data, TDOC serves as a viable solution. Its efficiency and simplicity make it suitable for applications where heavy encryption is not justified.

6. **Secure Data Sharing in Small Enterprises**
   Small businesses and enterprises with moderate security requirements can use TDOC for encrypting sensitive data, such as internal documents or customer information. The algorithm’s customization and ease of implementation make it appealing for such use cases.

6. Conclusion

The TDOC algorithm has successfully addressed the limitations of traditional encryption systems like the Caesar Cipher by introducing innovative mechanisms such as dynamic trigonometric shifts, XOR obfuscation, and support for a full ASCII character set. Through extensive testing with various types of plaintexts, including uppercase letters, special characters, and mixed data, TDOC has demonstrated its robustness in maintaining data confidentiality and integrity. The algorithm's dual-layer encryption enhances its resistance to brute force and frequency analysis attacks.

TDOC's incorporation of position-based dynamic shifts and non-linear trigonometric calculations has proven to be a pivotal enhancement over conventional methods. These features ensure that even identical characters in the plaintext result in unique ciphertext values. The use of XOR obfuscation further strengthens the cryptographic system by eliminating discernible patterns in the ciphertext, showcasing the importance of layered security.

While TDOC has shown great potential, several avenues for improvement and exploration remain:

- **Optimization:** Enhance the algorithm's computational efficiency for large-scale datasets.
- **Advanced Cryptanalysis Testing:** Conduct detailed studies to evaluate its resilience against modern cryptanalysis techniques such as differential and linear cryptanalysis.
- **Expanded Character Support:** Adapt the algorithm to handle extended character sets beyond ASCII, including Unicode.
- **Integration with Protocols:** Explore the integration of TDOC into secure communication protocols to evaluate its real-world applicability.

The evolution of cryptographic systems is essential in safeguarding the ever-growing digital landscape. TDOC exemplifies how traditional methodologies can be modernized to meet contemporary security challenges. Its innovative design and adaptability highlight the critical role of cryptography in ensuring secure data communication.

7. References

Loo, A. (2024, december 05). *corporatefinanceinstitute*. Retrieved from corporatefinanceinstitute: https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/

Loshin, P. (2024, december 02). *techtarget*. Retrieved from techtarget: https://www.techtarget.com/whatis/definition/ASCII-American-Standard-Code-for-Information-Interchange

Amiruddin, M. (2024, december 05). *medium*. Retrieved from medium: https://mdamiruddin.medium.com/introduction-to-cryptography-tryhackme-writeup-walkthrough-75a1a198b973

ascii-code. (2024, december 05). *ascii-code*. Retrieved from ascii-code: https://www.ascii-code.com/

Christian, A.-A. (2024, december 04). *ssl2buy*. Retrieved from ssl2buy: https://www.ssl2buy.com/wiki/what-is-encryption-and-decryption

cryptomathic. (2024, december 05). *cryptomathic*. Retrieved from cryptomathic: https://www.cryptomathic.com/blog/symmetric-key-encryption-why-where-and-how-its-used-in-banking#:~:text=Symmetric%20encryption%20is%20a%20type,used%20in%20the%20decryption%20process

geeksforgeek. (2024, december 07). *geeksforgeek*. Retrieved from geeksforgeek: https://www.geeksforgeeks.org/transforming-a-plain-text-message-to-cipher-text/

geeksforgeeks. (2024, june 17). *geeksforgeeks*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

geeksforgeeks.org. (2024, november 01). *geeksforgeeks.org*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/difference-between-encryption-and-decryption/

geeksforgeeks.org. (2024, december 4). *geeksforgeeks.org*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

ibm. (2024, december 05). *ibm*. Retrieved from ibm: https://www.ibm.com/topics/cryptography

itgovernance. (2024, december 05). *itgovernance*. Retrieved from itgovernance: https://www.itgovernance.co.uk/what-is-cybersecurity javatpoint. (2024, december 05). *javatpoint*. Retrieved from javatpoint: https://www.javatpoint.com/caesar-cipher-technique

Kidd, C. (2024, december 05). *splunk*. Retrieved from splunk: https://www.splunk.com/en_us/blog/learn/cia-triad-confidentiality-integrity-availability.html

lee, l. (2024, december 05). *wallarm*. Retrieved from wallarm: https://www.wallarm.com/what/cia-triad-definition

ninjaone. (2024, december 05). *ninjaone*. Retrieved from ninjaone: https://www.ninjaone.com/it-hub/remote-access/what-is-an-encryption-key/

Richards, K. (2024, december 05). *techtarge*. Retrieved from techtarge: https://www.techtarget.com/searchsecurity/definition/cryptography

sciencedirect. (2024, december 05). *sciencedirect*. Retrieved from sciencedirect: https://www.sciencedirect.com/topics/computer-science/encryption-process#:~:text=Encryption%20is%20the%20process%20of,a%20cipher%20is%20called%20ciphertext.

simplilearn. (2024, december 06). *simplilearn*. Retrieved from simplilearn: https://www.simplilearn.com/tutorials/cryptography-tutorial/asymmetric-encryption

telsy. (2024, december 05). *telsy*. Retrieved from telsy: https://www.telsy.com/en/the-caesar-cypher/#:~:text=In%20cryptography%2C%20the%20Caesar%20cipher,positions%20later%20in%20the%20alphabet.

websitesecuritystore. (2021, august 18). *websitesecuritystore*. Retrieved from websitesecuritystore: https://websitesecuritystore.com/blog/what-is-the-cia-triad/