

A
Project Report
on
ROUND KEY-BASED HYBRID CRYPTOGRAPHY SYSTEM

A Report submitted in partial fulfillment of the
requirements for the award of degree of
Bachelor Of Technology

by

Megha Parate
(20EG105425)

Are Jagannath Rao
(20EG105456)

Bharath Kandimalla
(20EG105457)



Under the guidance of
Mr. B. Ravinder Reddy
Assistant Professor
Department of CSE,
Anurag University

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ANURAG UNIVERSITY
VENTAKAPUR-500088
Year 2023-2024**

DECLARATION

We hereby declare that the Report entitled “**Round key-based hybrid cryptography system**” submitted for the award of Bachelor of technology Degree is our original work and the Report has not formed the basis for the award of any degree, diploma, associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

Megha Parate
(20EG105425)

Are Jagannath Rao
(20EG105456)

Bharath Kandimalla
(20eg105457)



CERTIFICATE

This is to certify that the Report entitled **“Round key-based hybrid Cryptography System”** that is being submitted by **Megha Parate** bearing the roll number **20EG105425**, **Are Jagannath Rao** bearing the roll number **20EG105456**, **Bharath Kandimalla** bearing the roll number **20EG105457** in partial fulfillment for the award of B. Tech in **Computer Science and Engineering** to the **Anurag University** is a record of bonafide work carried out by them under my guidance and supervision.

The result embodied in this Report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature Of Supervisor
B. Ravinder Reddy
Assistant Professor, Dept. of CSE
Anurag University

Head of Department

External Examiner

ACKNOWLEDGMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **B. RAVINDER REDDY**, Assistant Professor, Department of Computer Science and Engineering, Anurag University, for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. His patience, guidance and encouragement made this project possible.

We would like to acknowledge our sincere gratitude for the support extended by **DR. G. VISHNU MURTHY**, Dean, Department of Computer Science and Engineering, Anurag University. We also express our deep sense of gratitude to **Dr. V. V. S. S. S. BALARAM**, Academic co-ordinator. **Dr. PALLAM RAVI**, Project Co-ordinator and project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stages our project work.

We would like to express our special thanks to **Dr. V. VIJAYA KUMAR**, Dean School of Engineering, Anurag University, for their encouragement and timely support in our B. Tech program.

Megha Parate
(20EG105425)

Are Jagannath Rao
(20EG105456)

Bharath Kandimalla
(20EG105457)

ABSTRACT

Hybrid cryptographic encryption techniques combine multiple encryption algorithms to enhance the security of sensitive data. The work presents a hybrid encryption scheme based on the Vigenère and Polybius ciphers. Although these classical ciphers provide basic encryption capabilities, they are vulnerable to few attacks like kasiski attack, frequency analysis attacks and index of coincidence. To overcome this vulnerability, the round key is introduced as an additional layer of security. The round key is generated using a secure random number generator and serves as a dynamic component in the encryption process. By incorporating the round key, the encryption algorithm becomes more resistant to frequency analysis attacks, kasiski attack and index of coincidence as letter frequencies are masked and vary across different rounds. The effectiveness of the proposed hybrid encryption scheme and demonstrated its resilience against known attacks. The results indicate that the round key significantly enhances the security of the encryption process and reduces the susceptibility to known attacks. The project contributes to the advancement of hybrid encryption techniques and provides a practical solution for securing sensitive data in communication channels and storage systems.

Keywords: Vigenère, Polybius, Kasiski attack, Frequency analysis attack, index of coincidence.

List of Figures

Figure No.	Figure Name	Page No.
1.1.1.	Vigenère Square	2
1.1.2.	Polybius Square	3
1.2.1.	Hybrid Cryptography System (Existing Method)	4
1.2.2.	Illustration of Existing Method	5
3.1.	Flowchart of Round key-based hybrid cryptography system.	10
3.2.	Extended Polybius Square	11
4.4.1.	Plaintext file	22
4.4.2.	Output	22
4.4.3.	Ciphertext file	22
5.1.	Environment Screenshot	25
6.1.1.	Comparison of Index of Coincidence	27
6.3.1.	Kasiski Attack on Existing Method	30
6.3.2.	Kasiski Attack on New Method	30

List of Tables

Table No.	Table Name	Page No.
2.1.	Comparison of Existing Methods	9
3.2.	XOR Operation	12
3.1.1.	Encryption using proposed method	13
3.2.1.	Decryption using proposed method	15
6.2.1.	Frequency analysis for existing method	28
6.2.2.	Frequency analysis for proposed method	29

List of Abbreviations

Abbreviations	Full Form
PRNG	Pseudo-random number generator
RSA	Rivest-Shamir-Adleman
XOR	Exclusive OR
IDLE	Integrated Development and Learning Environment
IC/IOC	Index of Coincidence

TABLE OF CONTENT

S. No.	CONTENT	Page No.
1.	Introduction	1
	1.1. Classical Ciphers	1
	1.2. Hybrid Cryptography System Based on Vigenère Cipher and Polybius Cipher	4
	1.3. Motivation	5
	1.4. Problem Definition	6
	1.5. Objective of the Project	7
2.	Literature Document	8
3.	Round Key-based Hybrid Cryptography System	10
	3.1. Proposed Scheme – Encryption Phase	12
	3.2. Proposed Scheme – Decryption Phase	14
4.	Implementation	16
	4.1. Functionalities	16
	4.2. Attributes	17
	4.3. Sample Code	18
	4.4. Experiment Screenshots	22
5.	Experimental Setup	23
	5.1. Installation of Python	23
	5.2. Selecting Files	24
	5.3. Packages	24
	5.4. Parameters	25
6.	Discussion of Results	27
	6.1. Index of coincidence	27
	6.2. Frequency Analysis Attack	28
	6.3. Kasiski Attack	29
7.	Summary, Conclusion and Recommendation	31
8.	Future Enhancements	32
9.	References	33

1. INTRODUCTION

Cryptography plays a vital role in ensuring data security through various aspects like authentication, confidentiality, non-repudiation, data integrity, etc. It is undeniable that the importance of network security has increased with the rapid advancement of technology. This change has ushered in a new era in which almost every aspect of human lives, from personal communication to critical processes, is largely based on digital technology. Classical ciphers are encryption methods used historically to protect the confidentiality of messages before the advent of modern computer-based encryption methods. Although they have historical significance and were once considered secure, classical ciphers are now widely used due to their flaws and are not considered secure enough for modern encryption needs. The vulnerabilities are well documented, and attackers can easily exploit them thanks to the computing power available today.

1.1. CLASSICAL CIPHERS

1.1.1. Vigenère cipher

The technique was invented by Blaise de Vigenère in the 16th century, is a classical encryption technique that offers a significant improvement over its predecessor, the Caesar cipher. The Vigenère cipher employs a keyword to determine the amount of shift for each character in the plaintext (Fig 1). However, despite its historical significance, the Vigenère cipher is no longer considered secure for modern cryptographic applications, as it can be cracked with computational tools and methods, such as index of coincidence, Kasiski examination and frequency analysis, when used with short or easily guessable keywords.

	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
A	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
B	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	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

Figure 1.1.1. Vigenère Square

Encryption in the Vigenère cipher involves using a keyword to shift the letters of the plaintext to create the ciphertext, for the keyword “KEY” and want to encrypt the message “HELLO”.

Firstly, the keyword is repeated to match the length of the plaintext:

Plaintext: H E L L O

Keyword: K E Y K E

Next, convert the letters to their numerical equivalents (A=0, B=1, C=2 ...):

Plaintext: 7 4 11 11 14

Keyword: 10 4 24 10 4

Now, for plaintext(P) and key(K):

$$\text{Ciphertext (Ei)} = (\text{Pi} + \text{Ki}) \bmod 26 \quad (1.1.1.)$$

Ciphertext: 17 8 9 21 18

Finally, convert these numbers back to letters using the same numerical mapping (0=A, 1=B, 2=C ...):

Ciphertext: R I J V S

So, “HELLO” encrypted with the keyword “KEY” becomes “RIJVS.”

Decryption in the Vigenère cipher follows a similar process but in reverse. The same keyword is used to determine the shifts and obtain the original plaintext from the ciphertext.

$$\text{Plaintext (Pi)} = (\text{Ei} - \text{Ki}) \bmod 26 \quad (1.1.2.)$$

The key step is subtracting the numerical values of the keyword from the ciphertext to recover the original numerical values of the plaintext, and then converting them back to letters.

1.1.2. Polybius cipher

Polybius Square is a simple substitution cipher that replaces each letter in the plaintext with a pair of coordinates representing its position in a 5x5 grid. This grid typically contains the letters of the alphabet (excluding J, which is often combined with I) arranged in rows and columns.

Encryption in the Polybius cipher involves replacing each letter in the plaintext with a corresponding pair of numbers based on its position in a grid. The grid typically consists of rows and columns labelled with numbers or letters. For example, if alphabets are arranged in 5x5 grid with the (excluding ‘J’), ‘A’ could be represented as “11,” ‘B’ as “12,” ‘C’ as “13,” and so on figure 2. Spaces or other non-alphabetic characters can also be represented using unique symbols. To encrypt a word like “HELLO,” convert it to its numeric representation, resulting in “23 15 31 31 24.”

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Figure 1.1.2. Polybius Square

Decryption in the Polybius cipher involves reversing the process by mapping the pairs of numbers back to their corresponding letters. Using the same 5x5 grid, “23 15 31 31 24” would be decoded as “HELLO” by finding the letters at the positions

specified by each pair of numbers. In this example, “23” corresponds to ‘H,’ “15” to ‘E,’ “31” to ‘L,’ and “24” to ‘O.’ The spaces or special symbols, if used, can be replaced as necessary. This straightforward mapping makes the Polybius cipher relatively easy to encrypt and decrypt, but it lacks the security needed for modern cryptographic purposes.

1.2. HYBRID CRYPTOGRAPHY SYSTEM BASED ON VIGENERE CIPHER AND POLYBIUS CIPHER

The existing hybrid cryptography system combines two classical ciphers, the Vigenère cipher and the Polybius cipher, in an attempt to enhance the security of data transmission. In this system as we can see figure 1.1, plaintext input undergoes a two-step encryption process. Initially, it is encrypted using the Vigenère cipher, and the output of this encryption is further encrypted using the Polybius cipher. The resulting ciphertext is then sent to the user along with the encryption keys. To decrypt the message, the user applies the reverse process, first using the Polybius cipher and then the Vigenère cipher.

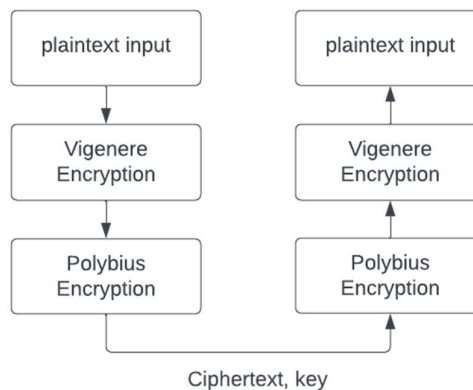


Figure 1.2.1. Hybrid Cryptography System (Existing Method)

However, this system has notable vulnerabilities that compromise its overall security. The primary weakness lies in the Polybius cipher’s usage, which is simply a mapping of the alphabet to numbers using a predefined Polybius square (Figure 1.2). Unlike the Vigenère cipher, there is no key involved in the Polybius encryption process. The lack of a key means that every user employs the same Polybius square for

encryption and decryption. As a result, this part of the system does not provide any substantial security enhancement.

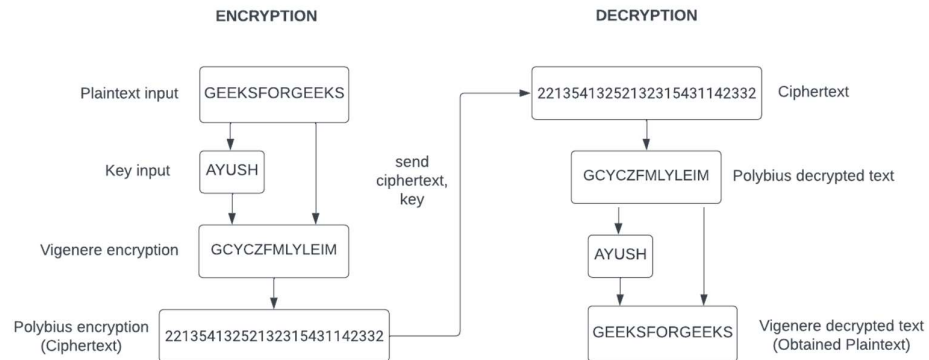


Figure 1.2.2. Illustration of existing method

Additionally, the Vigenère cipher, despite being using a second layer of encryption, still retains its well-known vulnerabilities. It is susceptible to attacks such as the Kasiski attack, index of coincidence analysis, and frequency analysis. These attacks can be used to recover the encryption key and, subsequently, the original message. Hence, even with the added step of Polybius encryption, the system is still vulnerable to classic cryptographic attacks, and its overall security remains compromised.

1.3. MOTIVATION

The motivation behind developing a hybrid cryptography system with a round key lies in the persistent quest to fortify the security of sensitive data. Hybrid cryptographic encryption techniques, by combining multiple algorithms, aim to create a robust defense against potential vulnerabilities in individual methods. The chosen combination of the Vigenère and Polybius ciphers forms a foundation, acknowledging their basic encryption capabilities while recognizing their susceptibility to certain attacks such as kasiski attack, frequency analysis attacks, and index of coincidence.

To address these vulnerabilities head-on, the introduction of a round key serves as a pivotal enhancement. This additional layer of security is not merely a static element but is dynamically generated using a secure random number generator. By incorporating the round key into the encryption process, the aim is to mitigate the risks posed by frequency analysis attacks, kasiski attack, and index of coincidence. The

round key acts as a dynamic component, masking letter frequencies and introducing variability across different rounds, thereby bolstering the resilience of the encryption algorithm.

The overarching goal is to elevate the security posture of the proposed hybrid encryption scheme, ensuring its effectiveness in the face of known attacks. Through meticulous evaluation, the project demonstrates that the inclusion of the round key significantly augments the security of the encryption process. By doing so, the susceptibility to well-known attacks is measurably reduced, affirming the practical value of this hybrid cryptography system in safeguarding sensitive data within communication channels and storage systems. This project, in essence, contributes to the advancement of hybrid encryption techniques, offering a tangible and effective solution for securing data in an evolving landscape of cybersecurity challenges.

1.4. PROBLEM DEFINITION

The integration of the Vigenère and Polybius ciphers within a hybrid cryptographic system forms a robust foundation for securing data communication. This combination harnesses the unique strengths of both ciphers to create a comprehensive encryption approach. Despite these advantages, the hybrid system faces vulnerabilities, specifically to frequency analysis attacks, kasiski attacks, and the index of coincidence.

The frequency analysis attack poses a risk by capitalizing on patterns within the ciphertext to unveil the underlying plaintext. Given the periodic nature of the Vigenère cipher, an assailant may analyze the distribution of characters in the ciphertext, identifying recurring patterns that correspond to the periodicity induced by the Vigenère key. This analysis can expose segments of the original message, compromising the confidentiality of the communication.

Kasiski attacks target the repetition introduced by the Vigenère cipher. By identifying repeated sequences in the ciphertext and calculating the distances between them, an attacker can discern the length of the key, facilitating the decryption process.

This exploits the inherent patterns in the encryption, posing a threat to the security of the hybrid cryptographic system.

The index of coincidence is a measure of how likely two randomly selected characters from a piece of text are to be the same. In the context of the hybrid system, if the index of coincidence is higher than expected, it may indicate patterns that attackers could exploit.

1.5. OBJECTIVE OF THE PROJECT

The objective of this project is to enhance the security of sensitive data through the development and implementation of a hybrid cryptography system. This system employs a combination of the Vigenère and Polybius ciphers, acknowledging their foundational encryption capabilities while addressing vulnerabilities to attacks such as kasiski attack, frequency analysis attacks, and index of coincidence. The key innovation involves the integration of a round key as an additional layer of security. Generated dynamically using a secure random number generator, the round key introduces variability into the encryption process, making the algorithm more resilient against known attacks. The project aims to evaluate and demonstrate the effectiveness of this hybrid encryption scheme, emphasizing its practical utility in securing sensitive data within communication channels and storage systems. Ultimately, the objective is to contribute to the advancement of hybrid encryption techniques, providing a tangible solution to evolving cybersecurity challenges.

2. LITERATURE SURVEY

Classical ciphers, vulnerable to modern decryption techniques, have prompted the development of hybrid cryptography systems for enhanced security. In the work by **Shivam Vatshayan et al. [1]**, a hybrid cryptography system is introduced, amalgamating the Vigenère and Polybius ciphers. The process involves initially encrypting the plaintext with Vigenère and then applying further encryption using Polybius. Decryption follows the reverse steps. This approach offers heightened security by leveraging the strengths of both ciphers, presenting a more robust defence than relying on Vigenère or Polybius alone. **Aditi Saraswat et al. [2]** contribute to this field by creating a hybrid cipher that combines Vigenère and Caesar ciphers with an extended alphabet. This modification complicates frequency analysis, enhancing security. The inclusion of numbers, punctuation, and symbols in encryption makes attacks more challenging. **Hamza Touil et al. [3]** enhance security by combining Vigenère and Hill ciphers. Their method involves breaking the plaintext into equal-sized blocks, encrypting each block with Vigenère, and then encrypting these blocks again with Hill cipher. Decryption involves reversing these steps, adding an extra layer of complexity. **John Paul et al. [4]** secure the Vigenère cipher further by employing a pseudo-random number generator (PRNG) to create a random key. Expanding the alphabet to include numbers, punctuation, and symbols contributes to improved security. The modified Vigenère cipher encrypts the message with this key and extended alphabet, providing a more secure alternative to the standard Vigenère cipher. **Sazzad Hossain Saju et al. [5]** introduce a hybrid cryptosystem that merges a modified Vigenère cipher with asymmetric encryption, such as RSA. Creating two subkeys from the input key and rearranging the ciphertext for added complexity, the system enhances security by encrypting the symmetric key using asymmetric encryption. **O. E. Omolara et al. [6]** present a method involving a random number key and a random letter key. Initially using the Caesar cipher to shift the letter key based on the number key, they then apply the Vigenère cipher to the plaintext with the modified letter key as the keyword. Decryption involves reversing these steps. **Jan Carlo et al. [7]** propose a method where plaintext is encrypted with the Caesar cipher, followed by the Polybius square cipher. The ciphertext is then XORed with a random key, and decryption involves reversing these steps, offering a unique approach to hybrid cryptography.

Table 2.1. Comparison of Existing Methods

Sl.no	Author (s)	Method	Advantages	Disadvantages
1	Shivam Vatshayan, Raza Abbas Haidri, Jitendra Kumar Verma	Design of hybrid cryptography system based on Vigenère cipher and Polybius cipher	More secure than classical ciphers	Vulnerable to Frequency analysis attack, Kasiski attack
2	Miss. Pradnya Patil, Prof. S. S. Redekar	Hybrid Vigenère Polybius cipher with XOR operation for enhanced Cryptography	Security against Kasiski attack	Vulnerable to Frequency analysis attack, men in the middle attack, fault analysis attack
3	Aditi Saraswata, Chahat Khatria, Sudhakara, Prateek Thakrala, Prantik Biswasa	An Extended Hybridization of Vigenère and Caesar Cipher Techniques for Secure Communication	Extended the Vigenère table by including the digits in the table so that numerical data can also be encrypted using the new proposed table.	Vulnerable to Frequency analysis attack
4	Sazzad Hossain Saju, Sayed Mahmudul Haque, Liakot Hossain Lingcon	A Hybrid Cryptographic Scheme of Modified Vigenère Cipher using Randomized Approach for Enhancing Data Security	Security against Kasiski attack	Vulnerable to Frequency analysis attack if the key size is small
5	Ayman Al-ahwal, Sameh Farid	The Effect of Varying Key Length on A Vigenère Cipher	Security against Frequency analysis attack with large key length	Performance is compromised when the key size is larger

3. ROUND KEY-BASED HYBRID CRYPTOGRAPHY SYSTEM

Proposed hybrid cryptography system represents a remarkable advancement in data security by effectively mitigating the vulnerabilities of traditional ciphers to attacks like Kasiski, frequency analysis, and the index of coincidence.

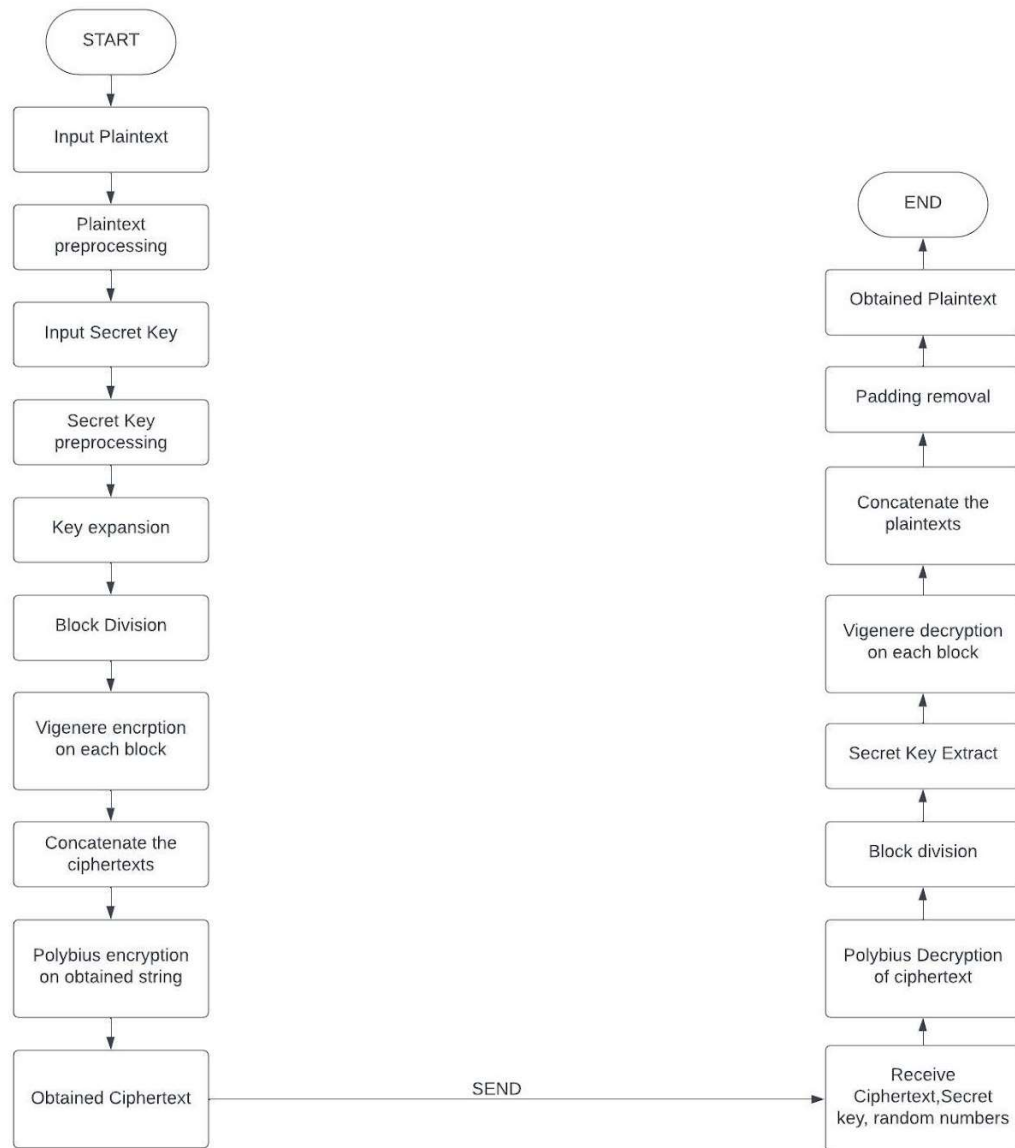


Figure 3.1. Flowchart of Round key-based hybrid cryptography system.

This novel approach incorporates three key elements to enhance encryption robustness:

i. Extended Polybius Square

Expanding from the traditional 5x5 grid to an 8x8 grid, introduces more complexity and versatility into the cipher. In addition to accommodating the 26 letters of the English alphabet (excluding 'J'), the 8x8 grid can incorporate numbers and symbols. This allows for a wider range of characters to be encoded. For instance, 'A' might be represented as "11" just as before, but now, numbers like '0' can be included, along with various symbols like '@' or '#,' each assigned a unique numeric code within the 8x8 grid [7]. This expanded grid enhances the encryption capabilities of the Polybius cipher, enabling the secure encoding of a broader set of characters, numbers, and symbols while still retaining the fundamental principle of mapping characters to specific grid coordinates for encryption and decryption.

	1	2	3	4	5	6	7	8
1	A	B	C	D	E	F	G	H
2	I	J	K	L	M	N	O	P
3	Q	R	S	T	U	V	W	X
4	Y	Z		0	1	2	3	4
5	5	6	7	8	9	!	"	#
6	\$	%	&	'	()	*	+
7	,	-	.	/	:	;	<	=
8	>	?	@	[\]	^	_

Figure 3.2. Extended Polybius Square

ii. Round keys

The Vigenère cipher, in its traditional form, is susceptible to Kasiski and frequency analysis attacks due to its key repetition during the encryption and decryption process. To bolster its security, a round key concept has been introduced. In this advanced approach, the plaintext is divided into 64-bit blocks, each undergoing encryption using round keys generated from random numbers. This strategic division and encryption not only add complexity to the cipher but also eliminate the key repetition pattern that attackers could exploit.

For example, the Plaintext is: INFORMATIONSYSTEM

Padded Plaintext: INFORMATIONSYSTEMGGGGGGG

Key input: SATURDAY

As the plaintext size is 24 repeat the key until its length becomes 24

Processed Key: SATURDAYSATURDAYSATURDAY

As the plaintext is of size 24 it will be divided into 3 blocks of 64-bit, hence 3 round keys must be generated.

Hence 3 Random numbers generated are: 19, 6, 12.

The round key is extracted by accessing the key in a circular manner based on the value of the random number.

Round keys generated:

Round key 1: TURDAYSA

Round key 2: DAYSATUR

Round key 3: URDAYSAT

iii. XOR Operation

The Polybius cipher, while straightforward in its ciphertext production by mapping plaintext characters onto the Polybius square, is susceptible to analysis due to its predictable nature. To fortify its security, an additional layer of protection is introduced. In this enhanced scheme, a random number is XORed with the Polybius ciphertext. This added step significantly complicates the analysis for potential attackers, as the randomness introduced through XOR operations makes it exceptionally challenging to deduce patterns or decipher the message through simple frequency analysis or other traditional cryptanalysis methods.

Table 3.2. XOR Operation

S.No.	Level of Processing	Results
1.	Polybius Ciphertext	423116252225342617172624 332237151517421138221715
2.	Random number for XOR	165052825952296358225255 871536308879445437005708
3.	Ciphertext	496415446875339543665134 424367152618810030105887

3.1 PROPOSED SCHEME – ENCRYPTION PHASE

The random package is employed to autonomously generate random numbers, eliminating the need for user intervention in selecting round keys. These randomly generated numbers serve as the basis for generating the round key, enhancing the security and unpredictability of the cryptographic process.

Proposed Encryption Algorithm

Step 1: Input the plaintext message.

Step 2: Convert the plaintext into uppercase and pad it to a multiple of the block size.

Step 3: Input the secret key.

Step 4: Convert the secret key into uppercase.

Step 5: Repeat the secret key until it matches the length of the padded plaintext.

Step 6: Divide the padded plaintext into fixed-size blocks.

Step 7: Generate a random number in the range of 0 to the length of the repeated secret key for each block.

Step 8: Extract a 64-bit secret key (round-keys) starting from the generated random number for each block, accessing the secret key in a circular manner.

Step 9: Encrypt each block of plaintext with its respective secret key using the Vigenère cipher.

Step 10: Concatenate the encrypted blocks.

Step 11: Encrypt the concatenated message using the Polybius cipher.

Step 12: Perform XOR operation on Polybius Ciphertext with a random number generated to obtain the ciphertext for transmission.

Table 3.1.1. Encryption using proposed method

S. No.	Level of Processing	Results
1.	Plaintext	INFORMATIONSYSTEM
2.	Padded Plaintext	INFORMATIONSYSTEMGGGG GGG
3.	Key	SATURDAY
4.	Processed key	SATURDAYSATURDAYSATUR DAY
5.	Plaintext 64-bit Blocks	INFORMAT
		IONSYSSTE
		MGGGGGGG
6.	Random Numbers Generated	19
		6
		12
7.	Round Keys Extracted	TURDAYSA

		DAYSATUR
		URDAYSAT
8.	Vigenère Encryption and merge the results	BHWRRKSTLOLKYNV GXJGEYGZ
9.	Polybius Encryption	12235242422543443134312554313 3512253242215542255
10.	Polybius Ciphertext	12235242422543443134312554313 3512253242215542255
11.	Random Number for XOR	74589612589655338459869682184 6691210516515406205
12.	Final Ciphertext	86654801982602819078417828869 7567962880702344338

Once the encryption process is complete, the resulting ciphertext is transmitted to the user, accompanied by both the encryption key and the random numbers generated during the process.

3.2 PROPOSED SCHEME – DECRYPTION PHASE

In the decryption process, the sender provides the recipient with the decryption key and the random numbers that were initially used in the encryption process, alongside the ciphertext.

Proposed Decryption Algorithm

Step 1: Receive the ciphertext, secret key, and random numbers.

Step 2: Decrypt the ciphertext by initially performing XOR operation with received random number.

Step 3: Decrypt the ciphertext using the Polybius cipher.

Step 4: Divide the decrypted Polybius decrypted text into fixed-size blocks.

Step 5: Extract a 64-bit secret key (round-keys) starting from the received random number generated for each block of the Polybius decrypted text, accessing the secret key in a circular manner.

Step 6: Decrypt each block of the Polybius decrypted text with its respective secret key using the Vigenère cipher.

Step 7: Concatenate the decrypted blocks.

Step 8: Remove padding from the decrypted message to obtain the plaintext.

Table 3.2.1. Decryption using proposed method

S. No.	Level of Processing	Results
1.	Ciphertext Received	86654801982602819078417 82886975679628807023443 38
2.	Received number for XOR	74589612589655338459869 68218466912105165154062 05
3.	After XOR operation	12235242422543443134312 55431335122532422155422 55
4.	Polybius Decrypted	BHWRRKSTLOLKYLNV GXJGEYGZ
5.	Split Text into blocks of 64-bits	BHWRRKST
		LOLKYLNV
		GXJGEYGZ
6.	Vigenère Decryption using received round keys	INFORMAT
		IONSYSTE
		MGGGGGGG
7.	Padded Plaintext	INFORMATIONSYSTE MGGGGGGG
8.	After removing Padding	INFORMATIONSYSTEM

4. IMPLEMENTATION

Program file is hybrid cryptography system.py

Input: Plaintext file, key

Output: Ciphertext file

List of what all are sent from sender to receiver: ciphertext, random numbers, key for XOR operation, secret key.

4.1. FUNCTIONALITY:

1. Input Handling:

- Accepts a plaintext message as input.
- Converts the plaintext to uppercase and pads it to a multiple of the block size.

2. Key Input:

- Takes a secret key as input.
- Converts the secret key to uppercase.
- Repeats the secret key to match the length of the padded plaintext.

3. Block Processing:

- Divides the padded plaintext into fixed-size blocks.
- Generates a random number for each block to determine the starting point for key extraction.

4. Key Generation:

- Extracts a 64-bit secret key (round-keys) for each block, starting from the generated random number.
- Accesses the secret key in a circular manner.

5. Encryption (Vigenère Cipher):

- Encrypts each block of plaintext with its respective secret key using the Vigenère cipher.
- Concatenates the encrypted blocks.

6. Secondary Encryption (Polybius Cipher):

- Encrypts the concatenated message using the Polybius cipher.
- Performs XOR operation on Polybius ciphertext with a generated random number for transmission.

7. Decryption:

- Receives ciphertext, secret key, and random numbers for decryption.

8. Secondary Decryption (Polybius Cipher):

- Performs XOR operation on the received random number to obtain the Polybius ciphertext.
- Decrypts the Polybius ciphertext.

9. Key Extraction (Decryption):

- Divides the decrypted Polybius text into fixed-size blocks.
- Extracts a 64-bit secret key (round-keys) for each block, starting from the received random number.
- Accesses the secret key in a circular manner.

10. Decryption (Vigenère Cipher):

- Decrypts each block of the Polybius decrypted text with its respective secret key using the Vigenère cipher.
- Concatenates the decrypted blocks.

11. Output Handling:

- Removes padding from the decrypted message to obtain the plaintext.

4.2. ATTRIBUTES:

1. Uppercase Conversion:

- Converts both plaintext and secret key to uppercase.

2. Padding:

- Pads the plaintext to ensure it is a multiple of the block size.

3. Block Size:

- Defines the size of the blocks into which the padded plaintext is divided.

4. Random Number Generation:

- Generates random numbers for determining starting points in key extraction and XOR operations.

5. Cipher Algorithms:

- Utilizes Vigenère cipher for block-level encryption/decryption.
- Uses Polybius cipher for secondary encryption.

6. Circular Key Access:

- Accesses the secret key in a circular manner during both encryption and decryption.

7. XOR Operation:

- Performs XOR operations during both encryption and decryption with randomly generated numbers.

8. Concatenation:

- Concatenates blocks during encryption and decryption processes.

9. Data Transmission:

- Applies XOR operation for additional security during data transmission.

10. Message Formatting:

- Ensures proper formatting of the message during encryption and decryption processes.

4.3. SAMPLE CODE

```
#pkcs5 padding
def pkcs5_padding(plaintext, block_size):
    if len(plaintext) % block_size != 0:
        padding_length = block_size - (len(plaintext) % block_size)
        padbit=chr(ord('A')+padding_length-1)
        padbit=bytes(padbit,'utf-8')
        plaintext += padbit * padding_length
    else:
        padbit='H'
        padbit=bytes(padbit,'utf-8')
        plaintext += padbit * blocksize
    return plaintext

#Extracting keys for respective plaintexts
def round_key_generator(string, start, size):
    accessed_string = ""
    length = len(string)
    start %= length
    end = (start + size) % length
    if start <= end:
        accessed_string = string[start:end]
    else:
        accessed_string = string[start:] + string[:end]
    return accessed_string

#ENCRYPTION
#Encrypting plaintext using vigenere cipher
def vigenere_encrypt(plaintext, key):
    ciphertext = ""
    key_index = 0
    for char in plaintext:
```

```

if not char.isalpha(): # Ignore non-alphabetic characters
    ciphertext += char
    continue
char_num = ord(char) - ord("A") # Convert letter to a number (A=0, B=1, ...)
key_num = ord(key[key_index % len(key)]) - ord("A") # Get corresponding
key number
encrypted_num = (char_num + key_num) % 26 # Perform Vigenère
encryption
encrypted_char = chr(encrypted_num + ord("A")) # Convert number back to
letter
ciphertext += encrypted_char
key_index += 1
return ciphertext

```

#Encrypting obtained vigenere ciphertext using polybius cipher

```

def polybius_encrypt(plain_text):
    polybius_grid = {
        'A': '11', 'B': '12', 'C': '13', 'D': '14', 'E': '15', 'F': '16', 'G': '17', 'H': '18',
        'I': '21', 'J': '22', 'K': '23', 'L': '24', 'M': '25', 'N': '26', 'O': '27', 'P': '28',
        'Q': '31', 'R': '32', 'S': '33', 'T': '34', 'U': '35', 'V': '36', 'W': '37', 'X': '38',
        'Y': '41', 'Z': '42', ' ': '43', '0': '44', '1': '45', '2': '46', '3': '47', '4': '48',
        '5': '51', '6': '52', '7': '53', '8': '54', '9': '55', '!' : '56', '"': '57', '#': '58',
        '$': '61', '%': '62', '&': '63', "'": '64', '(': '65', ')': '66', '*': '67', '+': '68',
        ',': '71', '-': '72', '.': '73', '/': '74', ':': '75', ';': '76', '<': '77', '=': '78',
        '>': '81', '?': '82', '@': '83', '[': '84', '\\': '85', ']' : '86', '^': '87', '\n': '88'
    }

    encrypted_text = ""
    for char in plain_text:
        if char in polybius_grid:
            encrypted_text += polybius_grid[char]
    return encrypted_text

```

```
#XOR operation with Random number
def xor_operation(Ciphertext):
    polykey=random.randint(pow(10,len(Ciphertext)-1),pow(10,len(Ciphertext))-1)
    Ciphertext=int(Ciphertext)^polykey
    return Ciphertext, polykey
```

```
#DECRYPTION
```

```
#decrypting ciphertext using polybius
```

```
def polybius_decrypt(encrypted_text):
    polybius_grid = {
        '11': 'A', '12': 'B', '13': 'C', '14': 'D', '15': 'E', '16': 'F', '17': 'G', '18': 'H',
        '21': 'I', '22': 'J', '23': 'K', '24': 'L', '25': 'M', '26': 'N', '27': 'O', '28': 'P',
        '31': 'Q', '32': 'R', '33': 'S', '34': 'T', '35': 'U', '36': 'V', '37': 'W', '38': 'X',
        '41': 'Y', '42': 'Z', '43': ' ', '44': '0', '45': '1', '46': '2', '47': '3', '48': '4',
        '51': '5', '52': '6', '53': '7', '54': '8', '55': '9', '56': '!', '57': '"', '58': '#',
        '61': '$', '62': '%', '63': '&', '64': "'", '65': '(', '66': ')', '67': '*', '68': '+',
        '71': ',', '72': '-', '73': '.', '74': '/', '75': ':', '76': ';', '77': '<', '78': '=',
        '81': '>', '82': '?', '83': '@', '84': '[', '85': '\\', '86': ']', '87': '^', '88': '\n'
    }
    
```

```
    decrypted_text = ""
    digits = []
    for char in encrypted_text:
        digits.append(char)
        if len(digits) == 2:
            pair = "".join(digits)
            if pair in polybius_grid:
                decrypted_text += polybius_grid[pair]
            digits = []
    return decrypted_text
```

```
#decrypting the ciphertext using vigenere cipher
```

```
def vigenere_decrypt(ciphertext, key):
    ciphertext = ciphertext.upper()
```

```

key = key.upper()
plaintext = ""
key_index = 0

for char in ciphertext:
    if not char.isalpha(): # Ignore non-alphabetic characters
        plaintext += char
        continue
    char_num = ord(char) - ord("A") # Convert letter to a number (A=0, B=1, ...)
    key_num = ord(key[key_index % len(key)]) - ord("A") # Get corresponding
    key number
    decrypted_num = (char_num - key_num) % 26 # Perform Vigenère decryption
    decrypted_char = chr(decrypted_num + ord("A")) # Convert number back to
    letter
    plaintext += decrypted_char
    key_index += 1

return plaintext

#remove padding to obtain Decrypted text
def unpad(text):
    plaintext=""
    lastbit=text[-1]
    unpadding_length=ord(lastbit)-ord('A')+1
    plaintext = text[:-unpadding_length]
    return plaintext

```

4.4. EXPERIMENTAL SCREENSHOTS

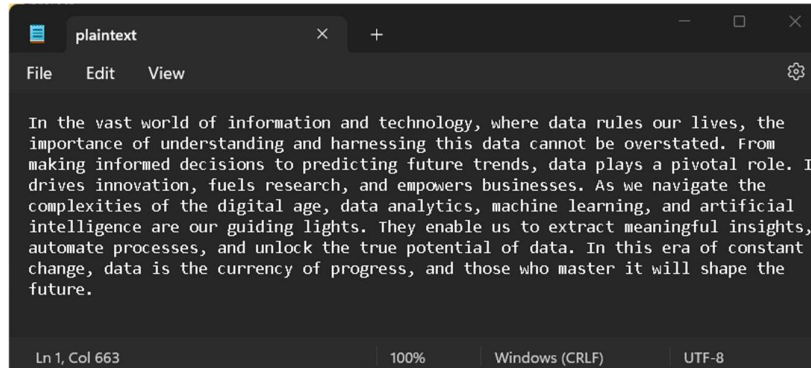


Figure 4.4.1. Plaintext file

```
===== RESTART: C:\Users\Megha Parate\OneDrive\Desktop\project\hybrid cryptography system.py =====
Enter key:cryptography

Vigenere encrypted: ZN IOC XTGE NOGSB DY WTWHSRTXVL TBJ KERWGCRCFGN, PVKIE SAIH PWCCZ MWI JXKVQ, IAS BAVFRIHLEV MU NPUOGLHGEFZLV TBJ AOXEEHZZLV MVOJ KYVR
APGBUX BT QMOGLHGRVD, UYMO DYZBEM ZLUIHFSVD KCEZQXQBY KO EYSJZCIPLI DJMIXV HGVNSZ, RGA ESVAJ Y EBJPTFS PQCT, BH JIBJKJ ICIUMAIMPM, LLEAZ PCHXONTH, ACK CO
GOCVRI ISQXGSYJEH, YH PS CTJOKAIL ZYE RVKVCMPKRVZ MH KFT UIVPRCC PNC, FRZR ACHJAMWIJ, MPAJZLT ESPKBOEG, PGR GITXLCZKHJ KLVVJABUKNRL YTV MJK UAZDXUE NZEW
ZJ. TWLW XBGSLT AJ TD LVAPCTR EXHLKEEUNZ ZNHPEJUG, GLTDTRTT WPQTGGJTL, OLS NBRFCQ KHT APJL NQKCCORL DM BPMO. OE TWQF GIC FD RHBQVRLI VVACNC, FRMO OJ TWX
QAIRTUEP MU IFCMIEHZ, CEB IACYV UWQ DYQVVE XM FWRC SWHNT MVK WBRWICQU

Polybius Encrypted text: 42264321271343383417424326271733124314414337343718163332343836244334122243231532371713321617267143283623211543331121184328371313
424325372143223838363171432111334312113616322118241536432535432628351317241817151642243643341222431127381515184242243643253627224323413632431128171235234
312344331251317241817233614734335412527431441421212254342243518163336144323131542313831124143232743154133224213212824214314222521383643183836263342714332
172311431533411122434143151222163428334328311334734312184322211222322423211335352511212825287143242415114243281318382738341871431113234313271713133632184
32133313817341221518734341843283343134222738112124434241154332362336131525283223364243251843231634433521362832131343282613714316324324311131822112537
21227143252811224224343153328231227151771432817324317213438244213381822432343636221112352326322443413436432522243351142143835154364215374222754334372
43743381217332434431122433414432436112813343243123818242315153526424342261828152223177143172434143432343443372831341722313424714327433432612321613314323
18344311282224432631231318273224431425431228252773432715433437283143171211343161443321812313632242143363611132613714316322527432722433437384331112132343
515284325343211613252115184271431315124321111341364335373143144113136362843382543283732134333371826344325362343371232372113733135

Final Ciphertext is in ciphertext.txt

TOTAL ENCRYPTION TIME: 0.0781188011694336

Polybius Decrypted: ZN IOC XTGE NOGSB DY WTWHSRTXVL TBJ KERWGCRCFGN, PVKIE SAIH PWCCZ MWI JXKVQ, IAS BAVFRIHLEV MU NPUOGLHGEFZLV TBJ AOXEEHZZLV MVOJ KYV
R APGBUX BT QMOGLHGRVD, UYMO DYZBEM ZLUIHFSVD KCEZQXQBY KO EYSJZCIPLI DJMIXV HGVNSZ, RGA ESVAJ Y EBJPTFS PQCT, BH JIBJKJ ICIUMAIMPM, LLEAZ PCHXONTH, ACK
GOCVRI ISQXGSYJEH, YH PS CTJOKAIL ZYE RVKVCMPKRVZ MH KFT UIVPRCC PNC, FRZR ACHJAMWIJ, MPAJZLT ESPKBOEG, PGR GITXLCZKHJ KLVVJABUKNRL YTV MJK UAZDXUE NZ
EWZJ. TWLW XBGSLT AJ TD LVAPCTR EXHLKEEUNZ ZNHPEJUG, GLTDTRTT WPQTGGJTL, OLS NBRFCQ KHT APJL NQKCCORL DM BPMO. OE TWQF GIC FD RHBQVRLI VVACNC, FRMO OJ T
WX QAIRTUEP MU IFCMIEHZ, CEB IACYV UWQ DYQVVE XM FWRC SWHNT MVK WBRWICQU

Vigenere Decrypted: IN THE VAST WORLD OF INFORMATION AND TECHNOLOGY, WHERE DATA RULES OUR LIVES, THE IMPORTANCE OF UNDERSTANDING AND HARNESSING THIS DATA
CANNOT BE OVERSTATED. FROM MAKING INFORMED DECISIONS TO PREDICTING FUTURE TRENDS, DATA PLAYS A PIVOTAL ROLE. IT DRIVES INNOVATION, FUELS RESEARCH, AND EM
POWERS BUSINESSES. AS WE NAVIGATE THE COMPLEXITIES OF THE DIGITAL AGE, DATA ANALYTICS, MACHINE LEARNING, AND ARTIFICIAL INTELLIGENCE ARE OUR GUIDING LIGH
TS. THEY ENABLE US TO EXTRACT MEANINGFUL INSIGHTS, AUTOMATE PROCESSES, AND UNLOCK THE TRUE POTENTIAL OF DATA. IN THIS ERA OF CONSTANT CHANGE, DATA IS THE
CURRENCY OF PROGRESS, AND THOSE WHO MASTER IT WILL SHAPE THE FUTURE.BB

TOTAL DECRYPTION TIME: 0.06253576278686523
```

Figure 4.4.2. Output

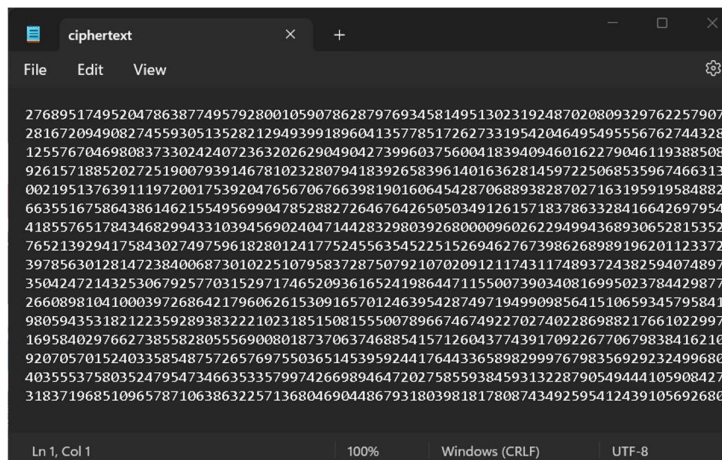


Figure 4.4.3. Ciphertext file

5. EXPERIMENTAL SETUP

Used **python IDLE** to develop this hybrid cryptography system.

5.1.PYTHON INSTALLATION

Here are the steps to install Python IDLE:

1. Download Python:

- Visit the official Python website: [<https://www.python.org/>].
- Navigate to the "Downloads" section.

2. Choose Python Version:

- Select the version of Python you want to install. It's recommended to choose the latest stable version.
- Click on the download link for the installer that corresponds to your operating system (Windows, macOS, or Linux).

3. Run the Installer:

- For Windows: Double-click the downloaded executable file (usually ending with `.exe`) to run the installer.
- For macOS: Open the downloaded `.pkg` file and follow the installation instructions.
- For Linux: Run the appropriate commands in the terminal as per your distribution's package management system.

4. Customize Installation (Optional):

- During the installation process, you may be given the option to customize the installation. Ensure that the checkbox for "IDLE" or "Tcl/Tk and IDLE" is selected.

5. Complete the Installation:

- Follow the on-screen instructions to complete the installation. Ensure that you check the box that says "Add Python to PATH" during the installation on Windows for easier command-line access.

6. Verify Installation:

- After the installation is complete, you can verify it by opening a command prompt or terminal and typing `python --version` or `python3 --version` to check that Python is installed.

7. Launch IDLE:

- IDLE is Python's integrated development environment, and it is typically installed along with Python. To launch IDLE:
 - For Windows: Look for "IDLE" in the Start menu.
 - For macOS: Open a terminal and type `idle` or look for IDLE in the Applications.
 - For Linux: Open a terminal and type `idle` or `idle3` depending on your Python version.

8. Start Coding:

- Once IDLE is launched, you can start writing and executing Python code.

5.2. SELECTING FILES:

Steps to select the plaintext file or ciphertext file:

Step 1: Open the location of the file

Step 2: Copy the path of the file

Step 3: paste the path in the following line numbers:

- i.) For reading plaintext file: 132
- ii.) For writing into ciphertext file: 176
- iii.) For reading the ciphertext file: 185

5.3.PACKAGES

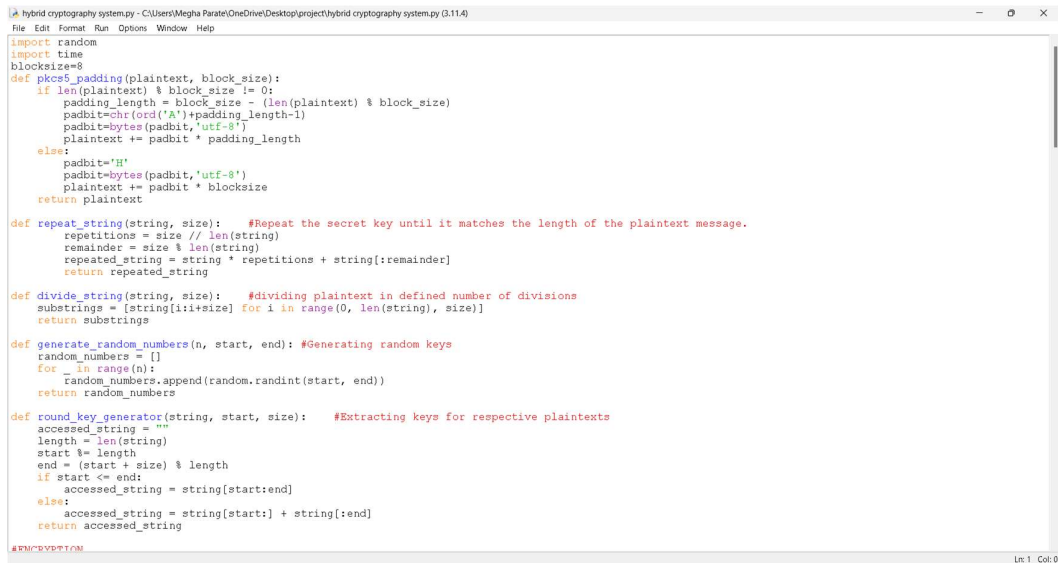
5.3.1. Time package

The "time" package in Python is a built-in module that provides various functions and methods for working with time-related operations. It allows you to measure time intervals, work with time in both human-readable and machine-readable formats, and perform tasks like creating delays in your programs. The "time" module can be used to access the current time, calculate execution times, format time as strings, and manipulate time in different ways. It's a valuable tool for tasks such as benchmarking code performance, scheduling actions based on time, and handling date and time information in various applications, making it an essential component for developers working on time-sensitive or time-dependent projects in Python. It has been used to determine encryption and decryption time.

5.3.2. Random package

The random package in Python is a versatile and essential module for generating random numbers, making decisions based on probability, and creating randomized elements in various applications. It provides functions for generating random integers, floating-point numbers, and sequences, as well as for shuffling lists, selecting random elements, and simulating random events. With its use, Python programmers can implement randomness in games, simulations, statistical analysis, and cryptographic applications, among others, contributing to the

unpredictability and variety needed for a wide range of programming tasks. Used to generate random number for round key generation.



```

hybrid cryptography system.py - C:\Users\Megha Parate\OneDrive\Desktop\project\hybrid cryptography system.py (3.11.4)
File Edit Format Run Options Window Help
import random
import time
blocksize=8
def pkcs5_padding(plaintext, block_size):
    if len(plaintext) % block_size != 0:
        padding_length = block_size - (len(plaintext) % block_size)
        padbit=chr(ord("A")+padding_length-1)
        padbit=bytes(padbit,'utf-8')
        plaintext += padbit * padding_length
    else:
        padbit="H"
        padbit=bytes(padbit,'utf-8')
        plaintext += padbit * blocksize
    return plaintext

def repeat_string(string, size): #Repeat the secret key until it matches the length of the plaintext message.
    repetitions = size // len(string)
    remainder = size % len(string)
    repeated_string = string * repetitions + string[:remainder]
    return repeated_string

def divide_string(string, size): #dividing plaintext in defined number of divisions
    substrings = [string[i:i+size] for i in range(0, len(string), size)]
    return substrings

def generate_random_numbers(n, start, end): #Generating random keys
    random_numbers = []
    for _ in range(n):
        random_numbers.append(random.randint(start, end))
    return random_numbers

def round_key_generator(string, start, size): #Extracting keys for respective plaintexts
    accessed_string = ""
    length = len(string)
    start %= length
    end = (start + size) % length
    if start <= end:
        accessed_string = string[start:end]
    else:
        accessed_string = string[start:] + string[:end]
    return accessed_string

#ENCIPHERMENT

```

Figure. 5.1. Environment screenshot

5.4. PARAMETERS

5.4.1. Index of Coincidence

It is a statistical measure used in cryptography and cryptanalysis to assess the likelihood that two characters, randomly selected from a given text, will be the same. It provides insights into the level of repetition or pattern within a piece of text, which can be valuable in analysing and breaking cryptographic ciphers, particularly when dealing with classical ciphers.

The formula to calculate the Index of Coincidence is typically as follows:

$$I.C. = \sum_{i=A}^{i=Z} \frac{f_i(f_i-1)}{N(N-1)} \quad (5.4.1.)$$

Where:

- f_i is the frequency of the i^{th} letter of the alphabet in the text.
- N is the total number of characters in the text.

In this method, a coincidence value greater than 0.068 is considered indicative of a ciphertext's dependency and can aid in its decryption.

5.4.2. Frequency Analysis Attack

This attack is a cryptanalysis technique used to break or decipher encrypted text, particularly when dealing with classical or simple substitution ciphers. This attack relies on the fact that in any language, certain letters or symbols appear more frequently than others. By analysing the frequency of characters or symbols in the ciphertext, an attacker can make educated guesses about the corresponding plaintext characters and potentially decrypt the message.

$$\text{Frequency of a character} = \frac{\text{Number of occurrences of that character in the ciphertext}}{\text{Total number of characters in the ciphertext}} \quad (5.4.2.)$$

5.4.3. Kasiski Attack

The Kasiski attack is a historical cryptanalysis technique used to break monoalphabetic substitution ciphers, such as the Vigenère cipher. Named after its originator, Friedrich Kasiski, this attack relies on the identification of repeating patterns in the ciphertext, exploiting the fact that if the same plaintext word or sequence is encrypted at multiple positions, it will produce the same ciphertext substring each time. By finding these repeating patterns and measuring the distances between them, an attacker can deduce the likely length of the keyword used in the Vigenère cipher, greatly simplifying the process of decryption. The Kasiski attack exemplifies the vulnerability of simple polyalphabetic ciphers when key management and encryption techniques are not appropriately robust.

$$\text{Key Length} = \text{GCD}(d1, d2, d3, \dots) \quad (5.4.3.)$$

- GCD stands for the Greatest common Divisor.
- $d1, d2, d3, \dots$ represents the difference in position between repeating substrings in the ciphertext.

6. DISCUSSION OF RESULT

6.1. INDEX OF COINCIDENCE

The below chart represents the results of the Index of Coincidence (IOC) for both the existing and new methods of encryption analysis. In this method, a coincidence value greater than 0.068 is considered indicative of a ciphertext's dependency and can aid in its decryption. Upon examination, it becomes evident that the values generated by the new method consistently fall below the threshold of 0.068, in stark contrast to the higher values associated with the existing method. This significant difference can be attributed to the deliberate introduction of key randomization in the new method. The deliberate variation in the encryption key has evidently led to lower IOC values, making the ciphertext less predictable and more secure against decryption attempts.

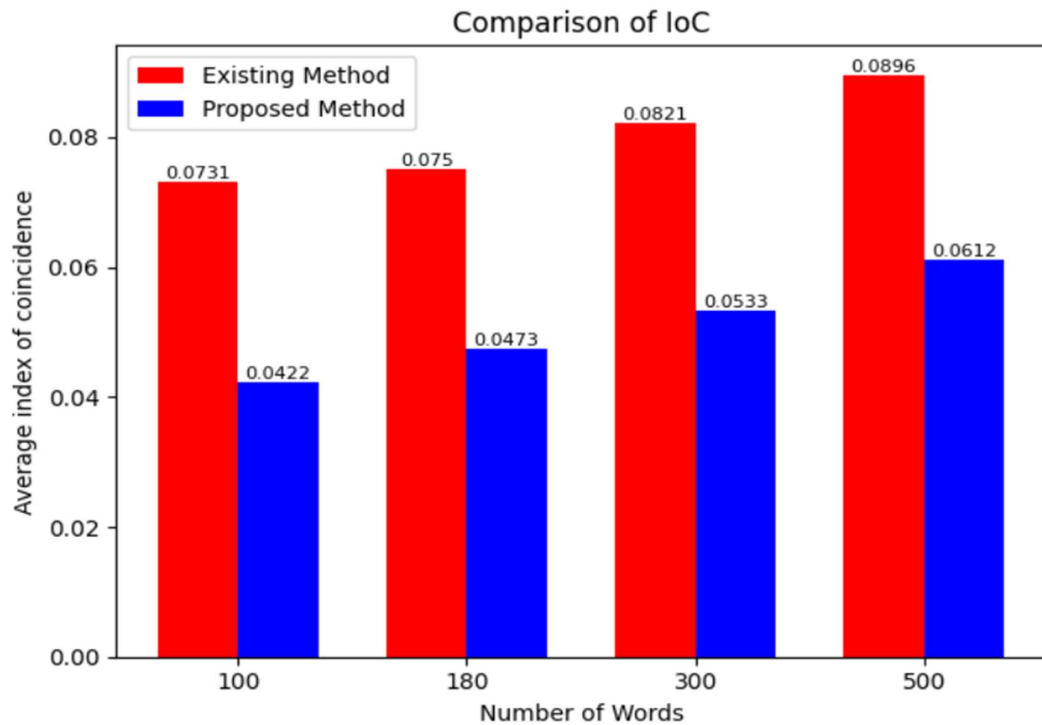


Figure 6.1.1. Comparison of Index of Coincidence

6.2. FREQUENCY ANALYSIS ATTACK

In the below tables, it is evident that the frequency analysis attack conducted on the existing encryption method yields ciphertext with patterns that can be somewhat guessed, especially when examining key lengths. However, when comparing this to the results of the new encryption method, a remarkable difference becomes apparent. In the new method, the frequencies of characters or symbols are distributed evenly throughout the ciphertext. This deliberate randomization of character occurrences makes traditional cryptanalysis techniques, such as the Kasiski test, considerably more challenging to perform effectively.

Table 6.2.1. Frequency analysis for existing method

Alphabets	Frequency	Alphabets	Frequency
G	7.16%	I	3.67%
L	6.42%	X	3.3%
T	6.06%	S	3.12%
R	5.87%	D	2.94%
W	5.69%	C	2.94%
V	5.32%	M	2.57%
F	5.32%	P	2.39%
U	4.59%	K	2.39%
A	4.59%	B	2.39%
H	4.22%	O	2.2%
N	4.22%	J	2.02%
E	3.85%	Z	1.65%
Y	3.85%	Q	1.28%

Table 6.2.2. Frequency analysis for proposed method

Alphabets	Frequency	Alphabets	Frequency
G	5.81%	E	3.65%
V	5.73%	H	3.61%
A	5.07%	I	3.54%
B	4.92%	X	3.24%
M	4.88%	C	3.38%
S	4.58%	P	3.27%
T	4.38%	Z	3.15%
L	4.27%	O	3%
R	4.15%	Q	3%
F	3.88%	D	2.92%
W	3.88%	Y	2.88%
U	3.73%	N	2.65%
K	3.69%	J	2.54%

6.3. KASISKI ATTACK

In the below images (Figure 6.3.1. and figure 6.3.2.), the outcomes of the Kasiski attack on the current encryption method unfold a compelling narrative. The attack efficiently discerned the precise length of the encryption key utilized in the existing method, offering valuable intelligence into the decryption process. This success implies a potential vulnerability in the original encryption technique, as the Kasiski attack managed to unravel key details crucial for decryption. However, the intriguing shift occurs when the same Kasiski attack is applied to the newly devised encryption method. Here, a stark contrast emerges, illustrating the formidable challenges encountered by the attack in accurately analyzing the key length. This stark disparity signifies a noteworthy improvement in the security posture of the new encryption method. The evident struggle faced by the Kasiski attack underscores the effectiveness of the heightened security measures integrated into the updated method,

presenting a formidable obstacle for adversaries attempting to exploit vulnerabilities in the encryption process. This disparity in outcomes serves as a compelling testament to the success of the implemented security enhancements, affirming the robustness of the new encryption method against Kasiski attacks.

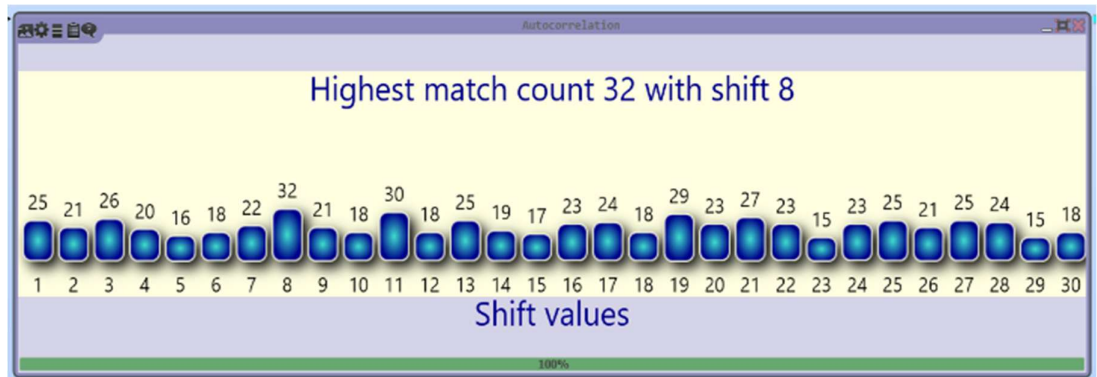


Figure 6.3.1. Kasiski Attack on Existing Method

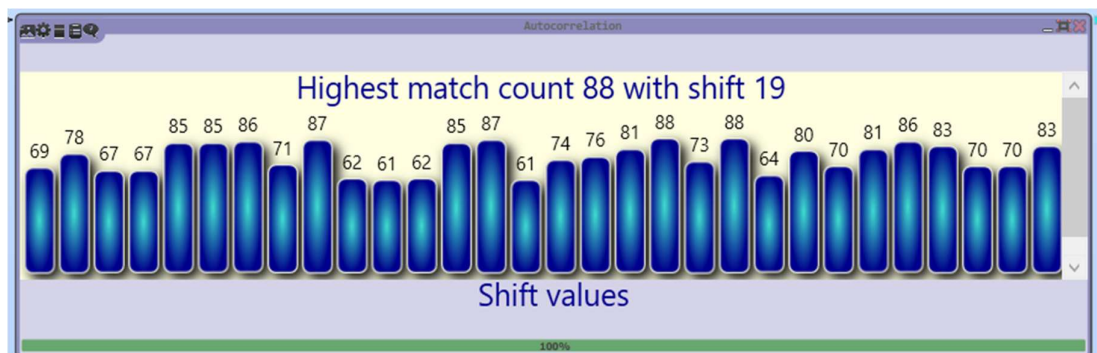


Figure 6.3.2. Kasiski Attack on New Method

7. SUMMARY, CONCLUSION AND RECOMMENDATION

Our hybrid cryptography system has proven to be exceptionally robust against a range of cryptanalysis techniques, including the formidable frequency analysis, Kasiski attack, and index of coincidence. The introduction of the round key concept stands out as a pivotal defence mechanism, injecting a crucial element of randomness into the encryption process. This strategic addition significantly elevates the difficulty for potential attackers seeking to exploit known patterns in the ciphertext. Moreover, the expansion of the Polybius square from a 5x5 grid to a more intricate 8x8 grid serves as a substantial enhancement to the system's complexity. This modification effectively raises the bar, making it exceedingly challenging for adversaries to deduce the original plaintext from the encrypted output.

A noteworthy aspect contributing to the system's security is the application of the XOR operation to the ciphertext derived from Polybius encryption. This additional layer of security serves as a formidable barrier against cryptanalysis attempts, further complicating the process of unravelling the encrypted information. While it is acknowledged that the incorporation of these advanced security measures may result in increased encryption and decryption times, the benefits derived from countering vulnerabilities in both the Vigenère and Polybius ciphers far outweigh the minor performance trade-offs. The systematic integration of these defence mechanisms demonstrates a comprehensive and well-considered approach to hybrid cryptography, where the focus on security measures ensures the confidentiality and integrity of the encrypted communication, even in the face of sophisticated cryptographic attacks.

8. FUTURE ENHANCEMENTS

In future enhancements of this hybrid cryptography project, a targeted focus could be directed towards optimizing the time efficiency of the encryption and decryption processes. Implementing algorithmic improvements and leveraging parallel computing techniques could potentially reduce the computational load, ensuring quicker cryptographic operations without compromising on security. Additionally, addressing the constraint of the secret key containing only alphabets presents an opportunity for expansion. Introducing support for a broader set of characters, including numbers, punctuation, and symbols, would enhance the system's versatility and compatibility with diverse data types. This adjustment would not only overcome current limitations but also contribute to the adaptability and applicability of the hybrid cryptography system in real-world scenarios where a richer character set may be encountered.

9. REFERENCE

- [1] S. Vatshayan, R. A. Haidri and J. Kumar Verma, "Design of Hybrid Cryptography System based on Vigenère Cipher and Polybius Cipher," 2020 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 2020, pp. 848-852, doi: 10.1109/ComPE49325.2020.9199997.
- [2] Aditi Saraswat, Chahat Khatri, Prateek Thakral Sudhakar and Prantik Biswas, "An Extended Hybridization of Vigenère and Caesar Cipher Techniques for Secure Communication", 2nd International Conference on Intelligent Computing Communication & Convergence Procedia Computer Science, vol. 92, pp. 355-360, 2016.
- [3] H. TOUIL, N. E. AKKAD and K. SATORI, "Text Encryption: Hybrid cryptographic method using Vigenère and Hill Ciphers," 2020 International Conference on Intelligent Systems and Computer Vision (ISCV), Fez, Morocco, 2020, pp. 1-6, doi:10.1109/ISCV49265.2020.9204095.
- [4] J. P. G. Perez et al., "A Modified Key Generation Scheme of Vigenère Cipher Algorithm using Pseudo-Random Number and Alphabet Extension," 2021 7th International Conference on Computer and Communications (ICCC), Chengdu, China, 2021, pp. 565-569, doi: 10.1109/ICCC54389.2021.9674565.
- [5] S. Saju, S. Haque, and L. Lingcon, "A Hybrid Cryptographic Scheme of Modified Vigenère Cipher using Randomized Approach for Enhancing Data Security," International Journal of Computer Applications, vol. 183, pp. 1-8, 2021, doi: 10.5120/ijca2021921290.
- [6] Omolara, O. E., A. I. Oludare, and S. E. Abdulahi. "Developing a Modified Hybrid Caesar Cipher and Vigenère Cipher for Secure Data Communication." Computer Engineering and Intelligent Systems 5.5 (2014): 34-46,2014

[7] J. C. Arroyo and A. J. Delima, "A Hybrid Caesar-Polybius Cipher with XOR Operation for Enhanced Cryptography," International Journal of Advanced Trends in Computer Science and Engineering, vol. 9, pp. 2961-2967, 2020. doi: 10.30534/ijatcse/2020/72932020.

[8] G. Swain, Object-Oriented Analysis and Design Through Unified Modelling Language. Laxmi Publication, Ltd., 2010.

[9] G. Booch, D. L. Bryan, and C. G. Peterson, Software engineering with Ada. Redwood city, Calif.: Benjamin/Cummings, 1994.

[10] J. M. Zelle, Python Programming: an introduction to computer science. Portland, Oregon: Franklin, Beedle & Associates Inc, 2017.