**PANDIT DEENDAYAL ENERGY UNIVERSITY**

Raisan, Gandhinagar – 382 426, Gujarat, India

**B.TECH- CSE**

**Laboratory Manual**

****

**Name- Jaivik Jariwala**

**Roll no.- 21BCP004**

**Subject-** Information Security Laboratory **Semester- 5th**

Submitted to

Dr. Hargeet Kaur

**Department of Computer Science&Engineering**

School of Energy Technology, Pandit Deendayal Energy University

**Experiment - 1**

**Aim :**

Hands-on Cryptool and Cryptography method/Toolkits

**Introduction :**

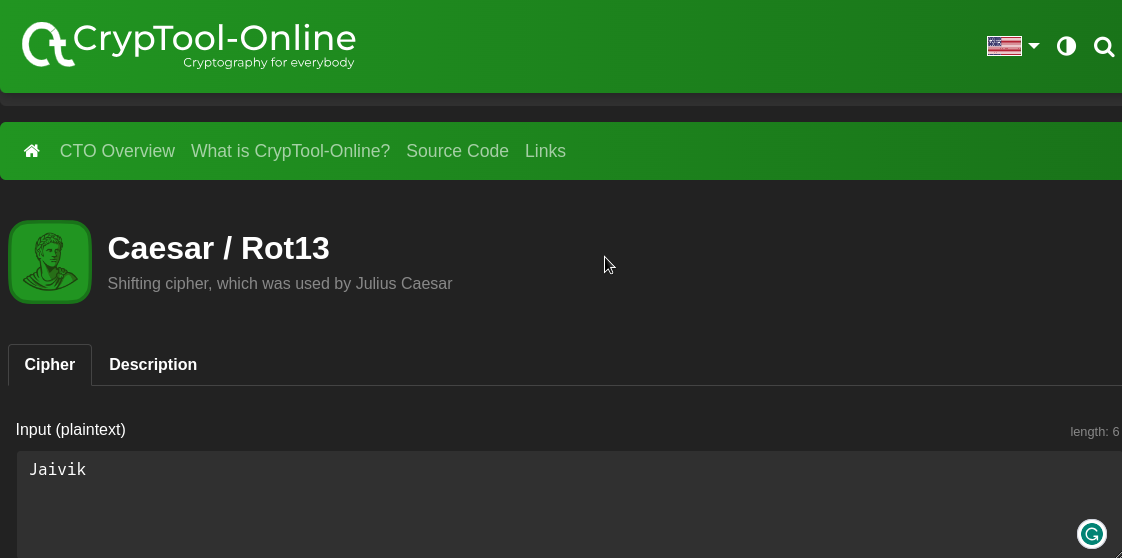
CryptoTool is a comprehensive software application created to serve as a hub for cryptographic exploration and analysis. By offering an intuitive interface and an array of features, CryptoTool empowers users to dive into the realm of cryptography, offering them opportunities to learn about encryption, decryption, hashing, and other vital cryptographic concepts.

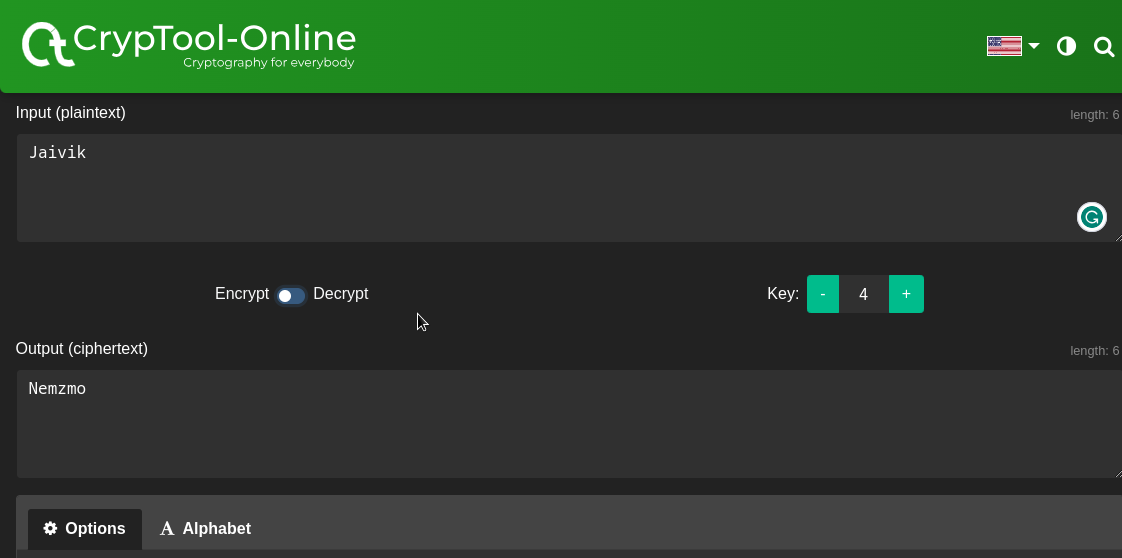
**Key Characteristics of CryptoTool:**

1. **Cryptographic Algorithms:** CryptoTool boasts support for an extensive selection of cryptographic algorithms, encompassing both symmetric and asymmetric encryption, cryptographic hashing, digital signatures, and more.
2. **User-Friendly Interface:** The tool's graphical user interface (GUI) is designed with ease of use in mind, allowing users to perform cryptographic operations without having to deal with complex coding.
3. **Algorithm Exploration:** Users are able to experiment with diverse cryptographic algorithms, input data, and parameters, enabling them to observe and comprehend the outcomes of different cryptographic processes.
4. **Encryption and Decryption:** CryptoTool enables users to encrypt and decrypt messages through both symmetric and asymmetric algorithms. Users can input plaintext and select encryption methods, keys, and other pertinent parameters.
5. **Hashing:** The application permits users to generate cryptographic hash values for input data, contributing to the verification of data integrity and the secure storage of passwords.
6. **Digital Signatures:** Users can create and validate digital signatures, ensuring the authenticity and integrity of messages.
7. **Random Number Generation:** CryptoTool provides tools for generating secure random numbers, which are indispensable for various cryptographic operations.
8. **Educational Utility:** CryptoTool is a valuable educational resource, aiding students and cryptography enthusiasts in comprehending cryptographic principles through hands-on experimentation.
9. **Algorithm Comparison:** Users can compare the strengths and weaknesses of different algorithms and encryption methods, further enhancing their understanding of real-world applications.

**Using CryptoTool:**

1. Installation: Download and install CryptoTool on your computer. The installation process is typically straightforward and involves running the installer file.
2. Launching the Application: After installation, launch CryptoTool. The interface will present a variety of cryptographic tools and options.
3. Selecting an Operation: Choose the desired cryptographic operation from the available options, such as encryption, decryption, hashing, or digital signatures.
4. Input Data: Input the data or plaintext message you intend to work with. Depending on the operation, you may also need to provide keys or specific parameters.
5. Customization: Customize the cryptographic algorithm, encryption method, keys, and other necessary parameters based on your requirements.
6. Executing the Operation: Initiate the chosen cryptographic operation by clicking the relevant button. The application will process the data according to your selections.
7. Observation: Observe the outcomes of the cryptographic operation. This may include encrypted or decrypted output, hash values, or digital signatures.
8. Exploration: Experiment with various algorithms, data, and parameters to understand the mechanics of different cryptographic techniques and their implications.
9. Saving and Analysis: Save the results of your experiments for further analysis or documentation purposes.





**Conclusion:**

CryptoTool serves as a valuable resource for individuals interested in cryptography, catering to individuals of varying expertise levels. Its user-friendly interface and assortment of features make it an optimal platform for learning, experimenting, and gaining practical insights into cryptographic principles. Whether you're a student, researcher, or enthusiast, CryptoTool offers an interactive approach to comprehending the intricate world of cryptography.

**Reference**

*www.cryptool.org/en/documentation/*

*www.cryptool.org/en/cto/caesar*

**Experiment - 2**

**Aim :**

Study and Implement program for Ceaser Cipher with Encryption and Decryption Brute Force Attack and Frequency Analysis Attack

**Introduction :**

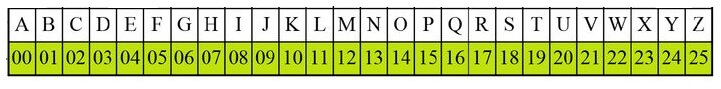
The Caesar cipher is the simplest and oldest method of cryptography. The Caesar cipher method is based on a mono-alphabetic cipher and is also called a shift cipher or additive cipher. Julius Caesar used the shift cipher (additive cipher) technique to communicate with his officers. For this reason, the shift cipher technique is called the Caesar cipher. The Caesar cipher is a kind of replacement (substitution) cipher, where all letter of plain text is replaced by another letter.

**Methodology :**

Let's take an example to understand the Caesar cipher, suppose we are shifting with 1, then A will be replaced by B, B will be replaced by C, C will be replaced by D, D will be replaced by C, and this process continues until the entire plain text is finished.

Formula for

1. Encryption : En (x) = (x + n) mod 26
2. Decryption : Dn (x) = (xi - n) mod 26



**Example : Use the Caesar cipher to encrypt and decrypt the message "JAIVIK," and the key (shift) value of this message is 3.**

**Encryption**

**We apply encryption formulas by character, based on alphabetical order.**

**The formula of encryption is: En (x) = (x + n) mod 26**

|  |  |  |
| --- | --- | --- |
| **Plaintext J -> 09** | **En (09+04)mod26** | **Ciphertext 13 ->N** |
| **Plaintext A -> 00** | **En (00+04)mod26** | **Ciphertext 4 ->E** |
| **Plaintext I -> 08** | **En (08+04)mod26** | **Ciphertext 12 ->M** |
| **Plaintext V -> 21** | **En (21+04)mod26** | **Ciphertext 25 ->Z** |
| **Plaintext I -> 08** | **En (08+04)mod26** | **Ciphertext 12 ->M** |
| **Plaintext K -> 10** | **En (10+04)mod26** | **Ciphertext 14 -> O** |

**The encrypted message is "NEMZMO". Note that the Caesar cipher is monoalphabetic, so the same plaintext letters are encrypted as the same letters.**

**Decryption**

**We apply decryption formulas by character, based on alphabetical order.**

**The formula of decryption is: Dn (x) = (xi - n) mod 26**

|  |  |  |
| --- | --- | --- |
| **Ciphertext N -> 13** | **En (13-04)mod26** | **Plaintext J -> 09** |
| **Ciphertext E -> 4** | **En (04-+04)mod26** | **Plaintext A -> 00** |
| **Ciphertext M -> 12** | **En (12-04)mod26** | **Plaintext I -> 08** |
| **Ciphertext Z -> 25** | **En (25-04)mod26** | **Plaintext V -> 21** |
| **Ciphertext M ->12** | **En (12-04)mod26** | **Plaintext I -> 08** |
| **Ciphertext O -> 14** | **En (14-04)mod26** | **Plaintext K -> 10** |

**If any case (Dn) value becomes negative (-ve), in this case, we will add 26 in the negative value.**

**Advantage**

1. **t is very easy to implement.**
2. **This method is the simplest method of cryptography.**
3. **Only one short key is used in its entire process.**
4. **If a system does not use complex coding techniques, it is the best method for it.**
5. **It requires only a few computing resources.**

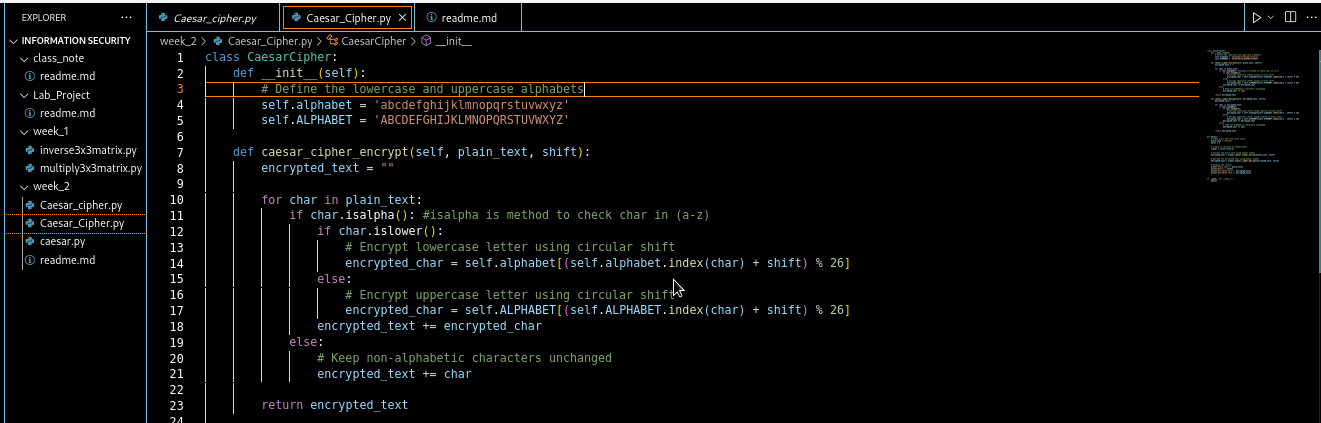
**Disadvantage**

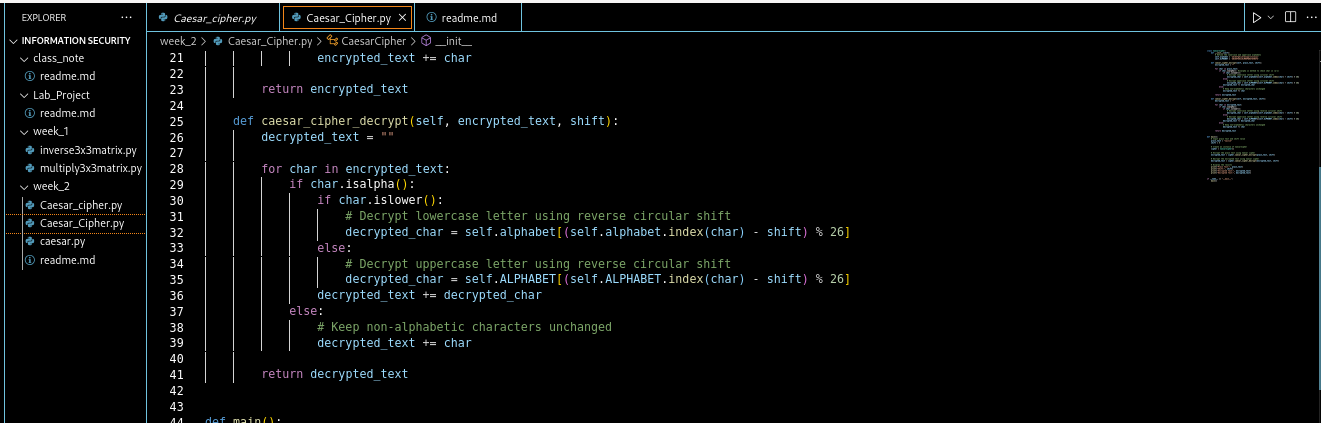
1. **It can be easily hacked. It means the message encrypted by this method can be easily decrypted.**
2. **It provides very little security.**
3. **By looking at the pattern of letters in it, the entire message can be decrypted.**

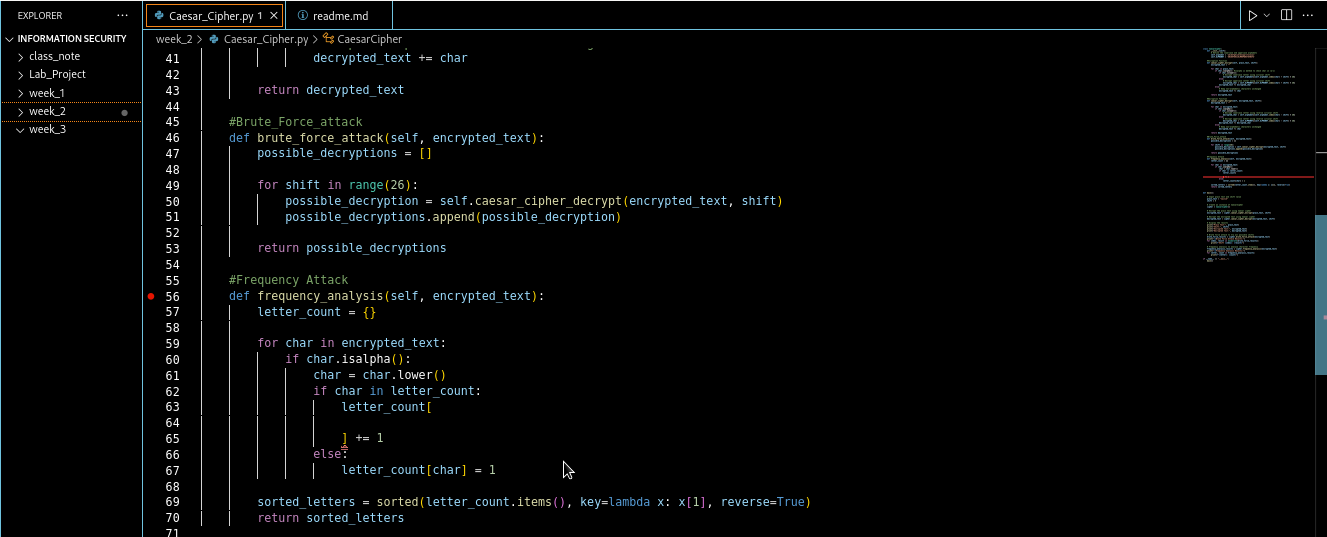
**Algorithm**

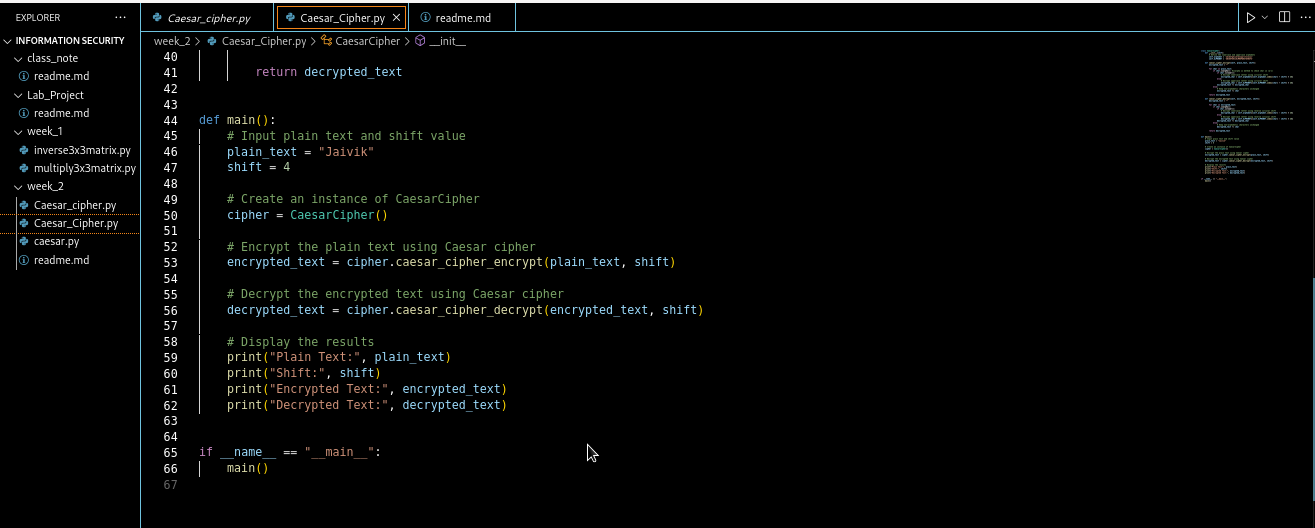
1. **Choose a shift value between 1 and 25.**
2. **Write down the alphabet in order from A to Z.**
3. **Create a new alphabet by shifting each letter of the original alphabet by the shift value. For example, if the shift value is 3, the new alphabet would be:**
4. **A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 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**
5. **Replace each letter of the message with the corresponding letter from the new alphabet. For example, if the shift value is 3, the word “hello” would become “khoor”.**
6. **To decrypt the message, shift each letter back by the same amount. For example, if the shift value is 3, the encrypted message “khoor” would become “hello”.**

**Code implementation**

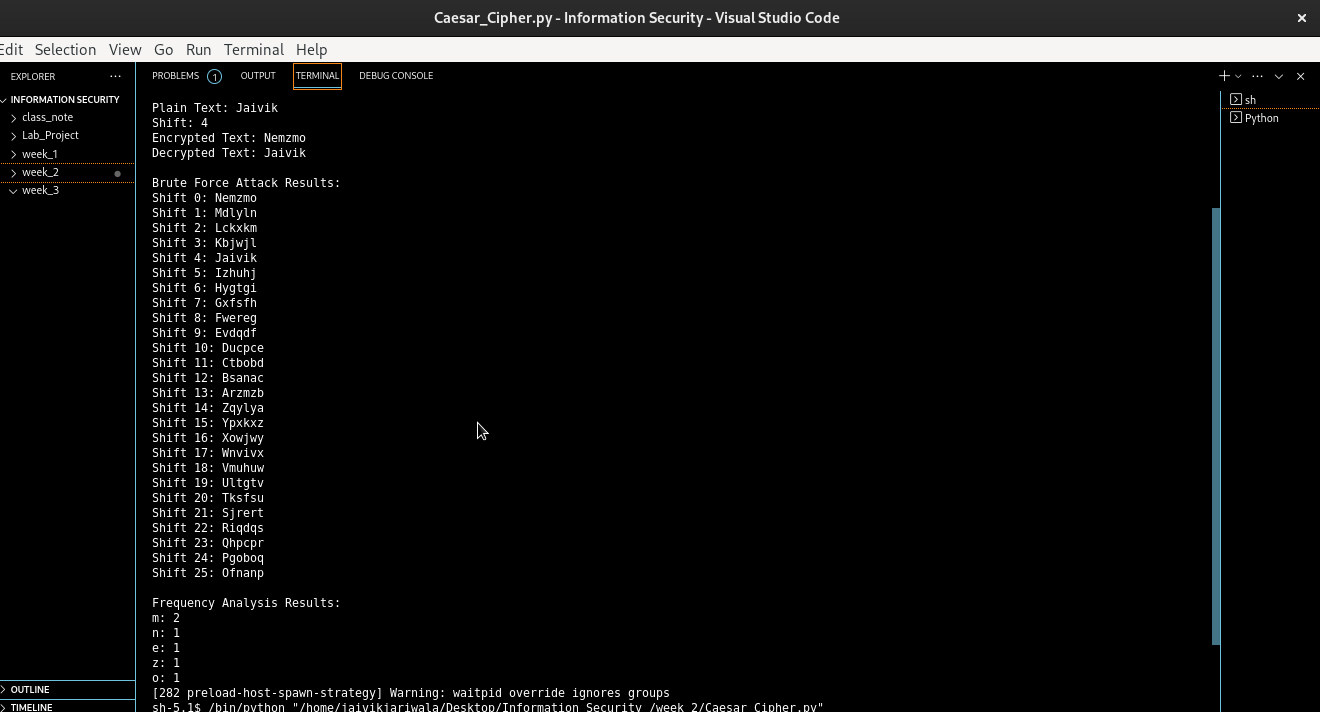
****

****

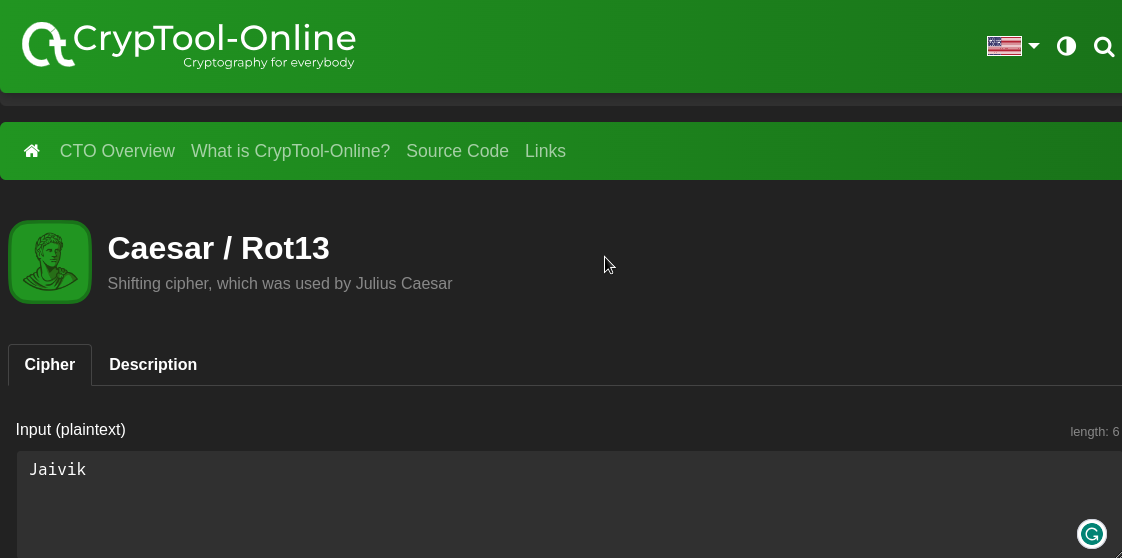
****

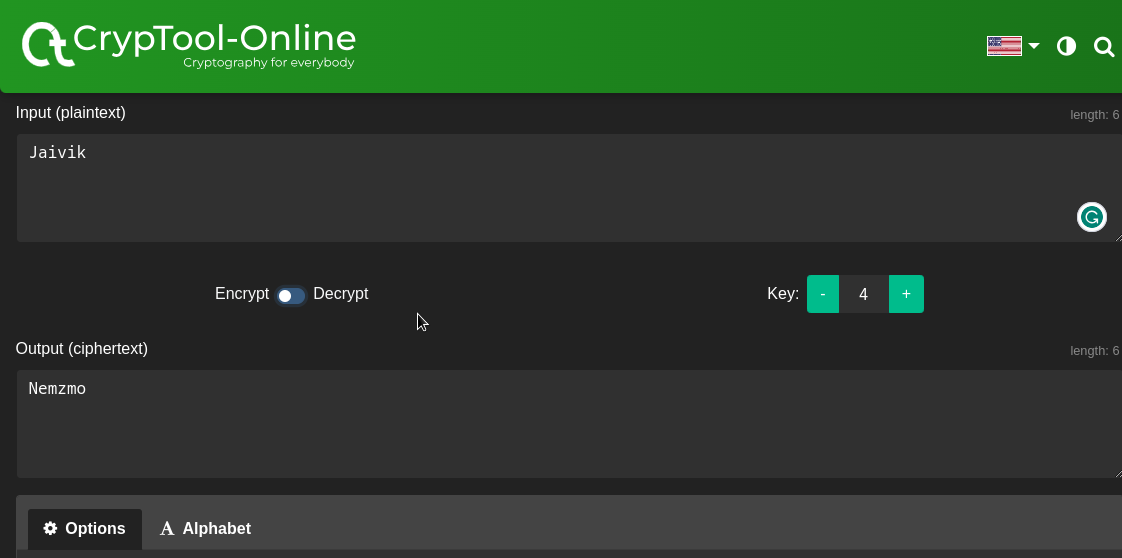
****

**Output**

****

Output Cryptool :





**CryptAnalysis**

Cryptanalysis of the Caesar cipher involves using various techniques and methods to break the encryption and reveal the original message without knowing the key (shift value) used for encryption. Here are some common cryptanalysis techniques used for breaking the Caesar cipher:

1. Brute Force Attack: In a brute force attack, all possible shift values (keys) are tried one by one to decrypt the ciphertext. This results in 25 different attempts (excluding the key 0, which is the same as the original text). The decrypted message that makes sense in English is most likely the original message.
2. 2.Frequency Analysis: In a Caesar cipher, if you analyze the frequency of letters in the ciphertext, you can look for patterns that match the frequency distribution of English letters. For example, if the most common letter in the ciphertext is 'X', then it is likely that the shift value is such that 'X' corresponds to 'E' or 'T' in English. By comparing frequencies, you can deduce the key.

**Applications**

While the Caesar cipher is no longer used for serious cryptographic purposes due to its vulnerability to attacks so:

1. Introduction to Cryptography: The Caesar cipher is often used as a beginner's example in cryptography courses and workshops. It introduces students to the fundamental concepts of encryption, decryption, and key-based transformations.
2. Puzzle and Escape Rooms: Escape rooms and puzzle games sometimes incorporate the Caesar cipher as a challenge for participants to decipher hidden clues. Players need to apply basic cryptographic skills to solve puzzles and progress through the game.
3. Children's Educational Tools: Educational apps, games, and books for children use the Caesar cipher to teach them about secret codes and messages. It's a fun way to introduce basic cryptographic concepts and engage young learners.
4. Historical and Cultural References: The Caesar cipher's historical association with Julius Caesar and ancient Rome makes it a popular element in historical fiction, movies, and cultural references. It adds an authentic touch to portrayals of historical contexts.
5. Personal Entertainment: People might use the Caesar cipher for personal amusement. They 0can encode messages to friends or family using a simple shift, creating a sense of mystery and intrigue. Recipients then have the challenge of decoding the message.

**References:**

[*www.cryptool.org/en/cto/caesar*](http://www.cryptool.org/en/cto/caesar)

[*www.javatpoint.com/caesar-cipher-technique*](http://www.javatpoint.com/caesar-cipher-technique)

[*www.geeksforgeeks.org/caesar-cipher-in-cryptography*](http://www.geeksforgeeks.org/caesar-cipher-in-cryptography)

*Enhancement Caesar Cipher for Better Security - isojournal*

**Experiment - 3**

**Aim :**

Study and Implement program for transposition (columnar) cipher with Encryption and Decryption

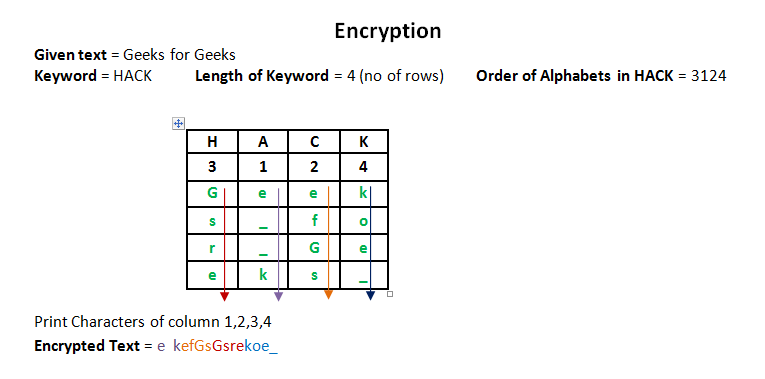
**Introduction :**

The Transposition Columnar Cipher is a cryptographic technique that operates by rearranging the characters of a plaintext message according to a defined columnar arrangement. Unlike substitution ciphers, it doesn't replace characters with other characters; instead, it reorders them. This method of encryption enhances security by introducing complexity. The key aspect of this cipher is the columnar arrangement, where the characters are written in columns and read out row by row to create the ciphertext. It's important to note that the security of this cipher lies in the secrecy of the columnar arrangement, making it a valuable example of historical cryptographic methods.

**Methodology :**

1. Key Generation:
   1. Choose a keyword that will serve as the basis for the encryption. Remove any duplicate letters from the keyword to create the key.
2. Creating the Columnar Arrangement:
   1. Write the plaintext message in rows, filling each row with as many characters as the length of the key. If the last row is not completely filled, pad it with dummy characters.
3. Rearranging Columns:
   1. Arrange the columns of the matrix based on the alphabetical order of the letters in the key. The columns represent the arrangement of the characters in the ciphertext.
4. Reading Out the Ciphertext:
   1. Read the matrix row by row, extracting characters column by column. The extracted characters form the ciphertext.
5. Decryption (Reverse Process):
   1. Start with the ciphertext and the same key used for encryption. Create the matrix by arranging the columns based on the alphabetical order of the key.
   2. Read the matrix column by column, extracting characters row by row. The extracted characters form the original plaintext message.

**Encryption :**

1. In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword.
4. In this case, the order would be “3 1 2 4”. Any spare spaces are filled with nulls or left blank or placed by a character (Example: \_).
5. Finally, the message is read off in columns, in the order specified by the keyword.

**Decryption :**

To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length. Then, write the message out in columns again, then re-order the columns by reforming the key word.

**Advantage :**

* Offers an additional layer of security by reordering characters.
* Requires knowledge of the key arrangement for decryption.
* Suitable for small messages and situations where complex encryption is not required.

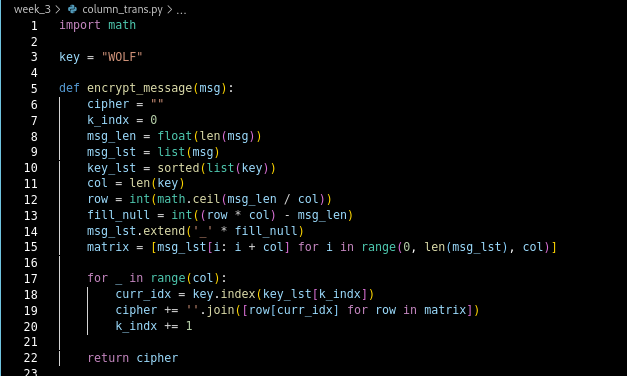
**Advantage :**

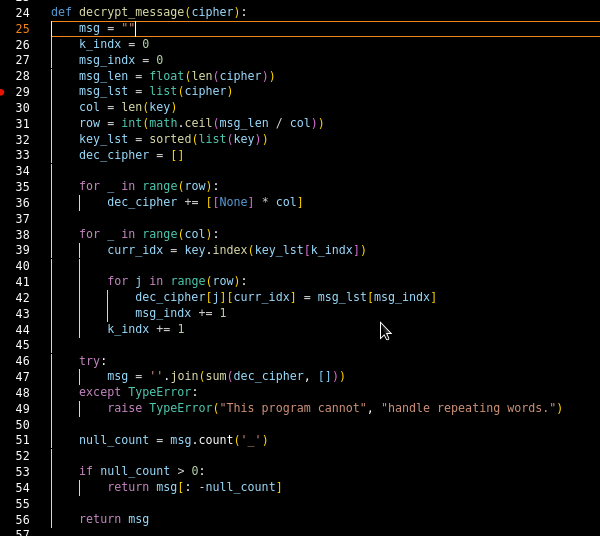
* Vulnerable to frequency analysis attacks if the message is long.
* Security depends on the secrecy of the key arrangement.

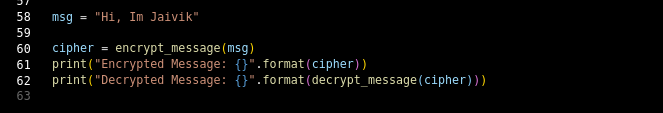
**Algorithm**

1. Write the plain text message row-by-row in grids of rectangle of a pre-defined size.
2. Read the message column by column. However, it need to be in the order of column 1,2,3,4,5,6 etc. It can be any random order such as 2, 6, 4, 3, 5, and 1.
3. After this operation message obtain is cipher text message. Simple columnar transposition technique simple arranges the plain text as a sequence of row of rectangle that is read in column randomly.
4. Simple columnar transposition technique is also quite simple to break after trying some permutation combinations of column orders to get original plain text.
5. To make this encryption more complex we can perform more than one round of transposition using same technique.
6. Let’s see an example. Suppose we have plain text HELLO YOU ARE READING THIS ON MY BSC IT. If we perform Simple columnar transposition technique operation on this text in column order of 2, 6, 4, 3, 5, and 1 then encrypted text will be EUAHYYRGNILRISSLADIBOENOCHOETMT .

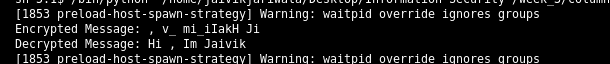
**Code implementation**

****

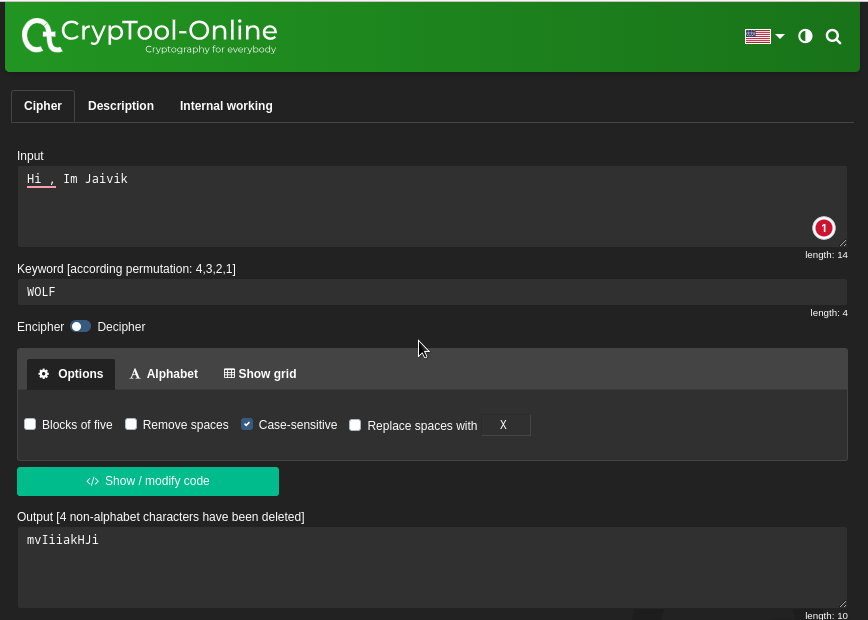
****

****

**Output**

****

Output Cryptool :



**ryptanalysis of Transposition (Columnar) Cipher:**

**The transposition cipher, particularly the columnar variant, is a type of permutation cipher where characters are rearranged according to a specific pattern. While it can provide some level of security, it has vulnerabilities that can be exploited through various techniques:**

**Brute Force Attack:**

**One of the most common cryptanalysis techniques is brute force, where all possible keys (permutations of column orders) are tried until the correct key is found.**

**The complexity of the brute force attack increases with the length of the key, making longer keys more secure.**

**However, the number of possible keys can grow rapidly, and the attack may still be feasible for shorter keys.**

**Frequency Analysis:**

**If the plaintext is in a specific language (e.g., English), certain letters and lettezr combinations will have a higher frequency of occurrence.**

**An attacker might analyze the frequency of letters or pairs of letters in the ciphertext to deduce the key length or column order.**

**For example, columns corresponding to more frequent letters might stand out in the frequency analysis.**

**Applications**

Historical Communication:

In the past, transposition columnar ciphers were used for secure communication in military and diplomatic circles.The reordering of characters provided an additional layer of security to protect sensitive information.

Basic Data Obfuscation:

The cipher can be used for basic data obfuscation in scenarios where strong encryption is not required.It can deter casual observers from understanding the content of the message.

Educational Purposes:

Transposition columnar ciphers are often used in educational settings to teach students about classical encryption techniques.They provide a practical understanding of encryption principles and historical cipher methods.

Puzzle and Games:

Transposition columnar ciphers can be employed in puzzles, escape rooms, or interactive games.Participants must decipher the message by following the specific key-based rearrangement.

Historical and Cultural Preservation:

The use of classical ciphers like the transposition columnar cipher helps preserve historical encryption methods and the evolution of cryptography.

**References:**

[*https://www.geeksforgeeks.org/columnar-transposition-cipher/*](https://www.geeksforgeeks.org/columnar-transposition-cipher/)

[*https://crypto.interactive-maths.com/columnar-transposition-cipher.html#:~:text=We%20first%20pick%20a%20keyword,the%20columns%20in%20this%20order*](https://crypto.interactive-maths.com/columnar-transposition-cipher.html#:~:text=We%20first%20pick%20a%20keyword,the%20columns%20in%20this%20order)*.*

Experiment - 4

**Aim:**

Study and Implement a program for Rail Fence Cipher with Encryption and Decryption

**Introduction:**

The Rail Fence Cipher is a transposition cipher that rearranges the characters of a message to create the ciphertext. This method is based on a simple zigzag pattern. It is named "Rail Fence" because the arrangement of letters resembles a fence made of parallel lines. The Rail Fence Cipher is a basic form of transposition cipher and provides a simple way to obscure the original message.







**Methodology:**

1. The Rail Fence Cipher arranges plaintext diagonally in a zigzag pattern on a specified number of "rails" or lines.
2. Ciphertext is read off from the rails row by row, following the zigzag pattern.
3. The number of rails chosen dictates the encryption level.
4. The process includes two key steps: encryption and decryption.

**Encryption:**

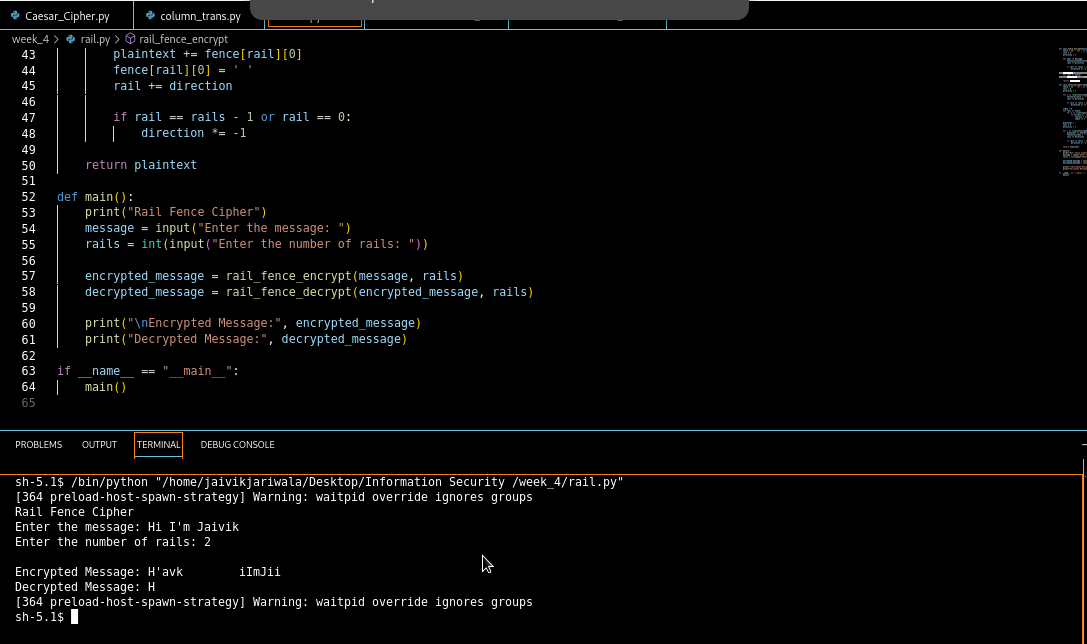
1. Initialize a 2D list named fence to represent the rail pattern, with each cell initialized to a space character.
2. Initialize variables rail and direction to keep track of the rail position and movement direction, respectively.
3. Iterate through each character (char) in the input message:
4. Find the index of the first empty space in the current rail of the fence using fence[rail].index(' ').
5. Place the current character in the found empty space.
6. Update the rail position based on the direction.
7. If rail reaches the top or bottom rail, reverse the direction.
8. Construct the encrypted ciphertext by joining the characters in each rail of fence.

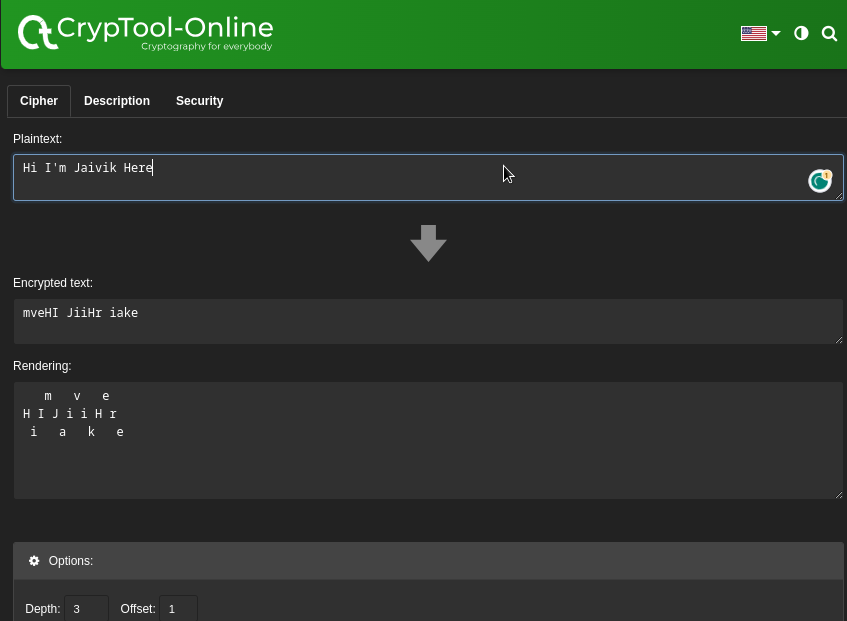
**Decryption:**

1. Initialize a 2D list named fence to represent the rail pattern, with each cell initialized to a space character.
2. Initialize variables rail and direction to keep track of the rail position and movement direction, respectively.
3. Iterate through indices (i) from 0 to the length of the ciphertext:
4. Mark the current position in the fence with an asterisk (\*) to indicate a character placement.
5. Update the rail position based on the direction.
6. If rail reaches the top or bottom rail, reverse the direction.
7. Iterate through each rail in fence and place characters from the ciphertext into positions marked with asterisks (\*).
8. Initialize an index variable index to 0 to keep track of the current character being placed.
9. Iterate through each rail in fence again:
10. Extract the character in the first position of the current rail and append it to the decrypted plaintext.
11. Set the first position of the current rail to a space character.
12. Update the rail position based on the direction.
13. If rail reaches the top or bottom rail, reverse the direction.
14. Return the decrypted plaintext obtained by joining the characters extracted from the rails.

****

****

****

****

**Advantages of Rail Fence Cipher:**

1. Simplicity: The Rail Fence Cipher is easy to understand and implement, making it suitable for educational purposes and basic encryption needs.
2. Quick Encryption: The encryption process is straightforward and requires minimal computational resources, making it efficient for quick encryption.
3. Visual Representation: The rail pattern creates a visual representation that can be appealing and engaging, especially for beginners learning about encryption.

**Disadvantages of Rail Fence Cipher:**

1. Low Security: The Rail Fence Cipher offers minimal security since it is susceptible to brute-force attacks and frequency analysis due to its simple nature.
2. Limited Key Space: The key space is limited to the number of rails, which reduces the number of possible keys and makes the cipher easier to crack.
3. Lack of Modern Applications: The Rail Fence Cipher is not commonly used for serious encryption due to its vulnerability, limiting its application in modern cryptography.

**Cryptanalysis :**

1. Brute Force Attack:

* The Rail Fence Cipher has a limited number of possible keys (number of rails), making it susceptible to a brute force attack.
* An attacker can try all possible rail configurations and decrypt the ciphertext with each configuration.
* The correct decryption will reveal itself when the plaintext becomes coherent.

1. Frequency Analysis:

* Since the Rail Fence Cipher rearranges characters in a predictable pattern, the frequency analysis technique can be applied.
* In English, certain letters have higher frequencies of occurrence (e.g., E, T, A), while others are less frequent (e.g., Q, Z).
* An attacker can analyze the frequency of letters in the ciphertext and compare it to the expected frequency distribution of English letters.
* The most common letters in the ciphertext may correspond to the most common letters in English, revealing potential rail configurations.

**Application:**

Educational Tool:

The Rail Fence Cipher is often used as a teaching tool to introduce students to the basic concepts of encryption, transposition ciphers, and programming. It provides a hands-on way to learn about cryptography and helps beginners understand how encryption techniques work.

Puzzle and Games:

Escape rooms, puzzles, and board games sometimes incorporate the Rail Fence Cipher as a challenge for participants to decode hidden messages. Solving the cipher adds an interactive and engaging element to gameplay.

Children's Activities:

Educational apps, games, and books for children use the Rail Fence Cipher to teach them about secret codes and messages. It's a fun way to introduce basic cryptographic concepts and engage young learners in problem-solving.

Historical and Cultural References:

The Rail Fence Cipher's simple yet intriguing nature makes it suitable for historical fiction, movies, and cultural references. Its association with codes and secret messages adds authenticity to portrayals of historical contexts.

Simple Communication:

In informal settings, individuals might use the Rail Fence Cipher for personal amusement. They can encode messages to friends or family using a simple rail configuration, adding an element of mystery and intrigue to the communication.

Reference :

<https://www.cryptool.org/en/cto/railfence>

<https://www.101computing.net/the-rail-fence-cipher/>

‑

Experiment - 5

Aim :

Study and implement a program for Vigenere cipher to encrypt and decrypt the message.

Introduction :

The Vigenère cipher, a Renaissance masterpiece, transcends basic encryption. By weaving shifting alphabets using a keyword, it defies traditional decryption tactics. This complex innovation baffles cryptanalysts, adding a layer of security that captivates cryptography enthusiasts. In this exploration, we unveil the cipher's mechanics, its resistance to attacks, and its historical prominence, showcasing the marriage of ingenuity and security in cryptography.

Methodology :

1. Choose a Keyword: Select a keyword that will be used for encryption and decryption. For example, let's use the keyword "KEY" for this example.
2. Convert to Numerical Values: Assign numerical values to the letters of the keyword based on their position in the alphabet (A=0, B=1, ..., Z=25). In our case, "K" becomes 10, "E" becomes 4, and "Y" becomes 24.
3. Repeat the Keyword: Repeat the keyword to match the length of the plaintext message. If the message is "HELLO", repeat the keyword "KEY" to get "KEYKE".
4. Shift and Encrypt: For each letter in the plaintext message, shift its numerical value by the corresponding value from the keyword using modular arithmetic. Add the numerical value of the keyword letter to the numerical value of the plaintext letter, and then take the result modulo 26 (since there are 26 letters in the alphabet).

H (7) + K (10) = 17 % 26 = 17 (R)

E (4) + E (4) = 8 % 26 = 8 (I)

L (11) + Y (24) = 35 % 26 = 9 (J)

L (11) + K (10) = 21 % 26 = 21 (V)

O (14) + E (4) = 18 % 26 = 18 (S)

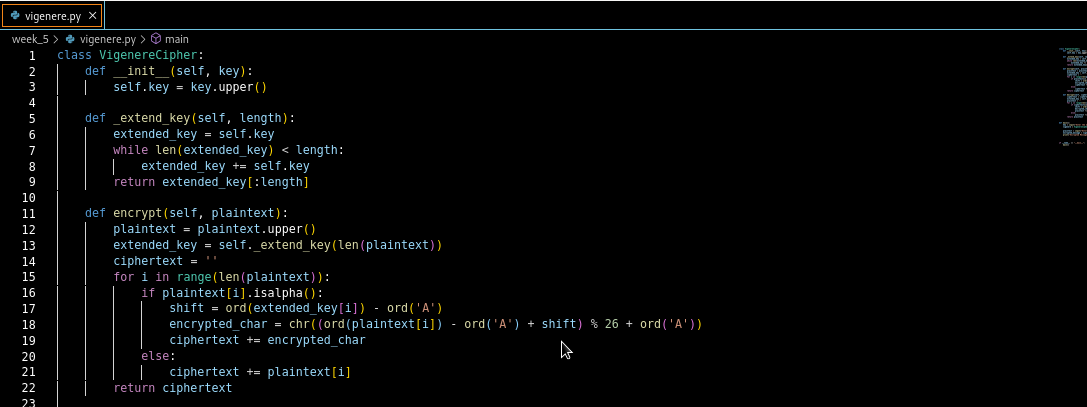
1. Ciphertext Formation: The encrypted message is formed by converting the numerical values back to letters. So, "HELLO" encrypts to "RIJSV".

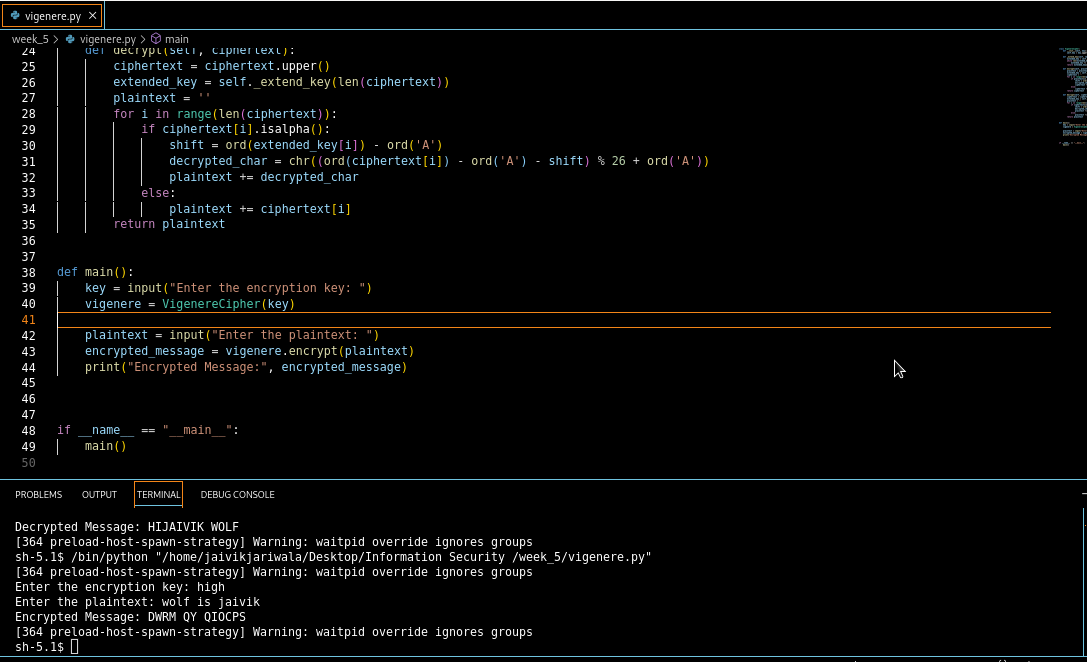
Encryption :

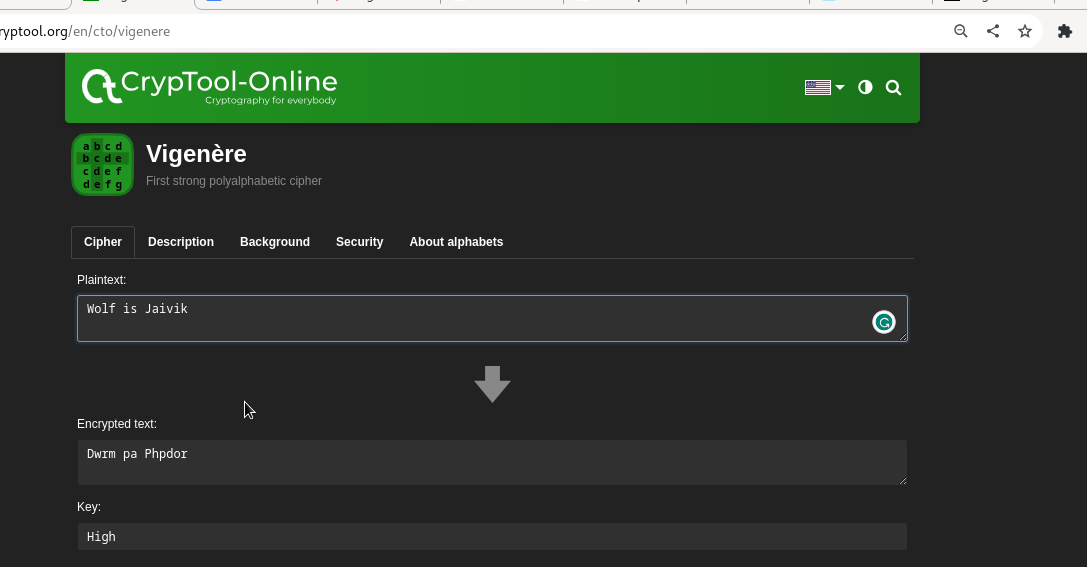
1. Choose a Keyword: Select a keyword consisting of letters. This keyword will determine the shifts applied to each letter of the plaintext.
2. Repeat Keyword: Repeat the keyword to match the length of the plaintext. If the plaintext is longer than the keyword, continue repeating the keyword.
3. Convert to Numerical Values: Assign numerical values to each letter of the plaintext and the repeated keyword, typically using A=0, B=1, ..., Z=25.
4. Encrypt Each Letter: For each letter in the plaintext, add the corresponding letter's numerical value from the keyword. Perform this addition modulo 26 to ensure the result remains within the range of letters.
5. Convert Back to Letters: Convert the numerical results back to letters using A=0, B=1, ..., Z=25. These letters form the ciphertext.

Decryption :

1. Choose the Same Keyword: To decrypt, use the same keyword that was used for encryption.
2. Repeat Keyword: Repeat the keyword to match the length of the ciphertext.
3. Convert to Numerical Values: Assign numerical values to each letter of the ciphertext and the repeated keyword, using the same mapping as before (A=0, B=1, ..., Z=25).
4. Decrypt Each Letter: For each letter in the ciphertext, subtract the corresponding letter's numerical value from the keyword. Perform this subtraction modulo 26.
5. Convert Back to Letters: Convert the numerical results back to letters using A=0, B=1, ..., Z=25. These letters form the decrypted plaintext.







Advantage :

* Polyalphabetic Complexity: The Vigenère cipher employs multiple shifting alphabets, enhancing security by thwarting frequency analysis and letter pattern recognition.
* Keyword-Based: The security of the cipher lies in the keyword, allowing for relatively simple encryption and decryption processes once the keyword is known.
* Historical Significance: The Vigenère cipher played a pivotal role in the history of cryptography, serving as an advancement beyond monoalphabetic ciphers.

Disadvantage :

* Key Management: Longer keywords are more secure, but they can become challenging to manage and remember.
* Kasiski Examination: Although resistant to simple attacks, the Vigenère cipher can still be broken using the Kasiski examination, which focuses on repeating patterns in the ciphertext.

Cryptanalysis :

Brute Force Analysis :

* The χ2 method is used to break simple substitution ciphers like Caesar's cipher or a Vigenère cipher with a one-letter keyword. It compares the letter frequencies of the encrypted text with the expected letter frequencies in the language (e.g., English). By trying all possible shifts for the single-letter keyword, the shift that gives the smallest χ2 value likely represents the correct decryption key. This method helps find the original message by identifying the best-fitting decryption based on letter frequency patterns.

Frequency analysis :

* letter x is encrypted to a ciphertext letter (ax+b) mod 26. If a is equal to 1, this is Caesar's cipher. In fact, if we choose a keyword of length 1 in a Vigenère cipher, it becomes Caesar's cipher. Because of this, the encryption and decryption algorithms discussed in Algebraic Nature have all the ingredients for encryption and decryption. The χ2 method is an effective way to decrypt a ciphertext encrypted using a simple substitution method, because there is only letter in the keyword and hence only one coset. Therefore, we just choose the keyword length to be 1 and use the χ2 method to find the best fit.
* The χ2 method is a goodness-of-fit measure that works on two sequences of values. In our case, the two sequences of values are: the English letter frequency and the letter frequency of a particular shift of the only coset. The shift of a coset that yields the smallest χ2 is likely to be encrypted by the corresponding letter of that shift. Hence, shifting the coset 26 times and finding the letter that has the smallest χ2 value will likely find the single-letter keyword.

Application :

1. Educational Purposes:

The Vigenère cipher is often used as an introduction to cryptography for educational purposes. It helps students understand basic concepts of encryption, substitution ciphers, and the importance of key management.

1. Historical Demonstrations:

Demonstrating the Vigenère cipher can help people understand historical encryption techniques. Reenactments or presentations can provide insights into how ciphers were used in history.

1. Puzzle and Challenges:

The Vigenère cipher can be used in puzzles and challenges, especially in escape room scenarios or puzzle hunts. Participants might need to decrypt a Vigenère-encrypted message to proceed to the next stage.

1. Classic Literature and Movies:

The Vigenère cipher can make appearances in literature, movies, or games set in historical or spy-themed contexts. It can add an element of authenticity and intrigue.

1. Code and Puzzle Design:

In the realm of recreational cryptography, designers might use the Vigenère cipher to create puzzles, treasure hunts, or cryptic codes for fun events.

References :

<https://inventwithpython.com/hacking/chapter21.html>

<https://www.cryptool.org/en/cto/vigenere>

Experiment – 6

**Aim**

Study and implement a program for Playfair cipher to encrypt and decrypt the message

**Introduction**

The Playfair cipher is a classical symmetric encryption technique that was developed in the 19th century and was used for secure communication during both World War I and World War II. This cipher is known for its unique approach to encryption, involving the use of a 5x5 matrix of letters and a set of simple rules for encrypting and decrypting messages. Unlike traditional substitution ciphers, the Playfair cipher operates on pairs of letters, making it resistant to many common cryptographic attacks.

Methodology

Encryption:

1. Key Setup:
   1. Choose a secret keyword (e.g., "KEYWORD") to generate the key matrix.
   2. Remove any duplicate letters from the keyword and append the remaining letters of the alphabet in order, excluding 'J' (which is treated as 'I').
   3. Create a 5x5 key matrix using the modified keyword. Fill the matrix row by row, left to right.
2. Preparing the Plaintext:
3. Remove any spaces and convert the plaintext message to uppercase.
4. Replace any 'J' with 'I' in the plaintext.
5. Creating Letter Pairs:
6. Divide the plaintext into pairs of letters. If there's an odd number of letters, add an 'X' to the end to create a pair.
7. Encrypting Pairs:
   1. For each letter pair:
   2. Find the positions (row and column) of the two letters in the key matrix.
   3. Apply the following rules to determine the ciphertext letters:
   4. If both letters are in the same row, replace each letter with the letter to its right in the same row. If at the end of the row, wrap to the beginning.
   5. If both letters are in the same column, replace each letter with the letter below it in the same column. If at the bottom, wrap to the top.
   6. If the letters are in different rows and columns, create a rectangle using the two letters and replace each letter with the letter at the opposite corner of the rectangle.
8. Forming the Ciphertext:
   1. Combine the encrypted letter pairs to create the ciphertext.

Decryption:

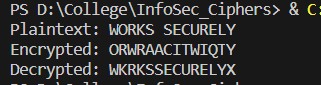
1. Key Setup:
   1. Use the same keyword to generate the key matrix as used in encryption.
2. Preparing the Ciphertext:
   1. Remove any spaces and convert the ciphertext to uppercase.
3. Creating Letter Pairs:
   1. Divide the ciphertext into pairs of letters.
4. Decrypting Pairs:
   1. For each letter pair:
   2. Find the positions (row and column) of the two letters in the key matrix.
   3. Apply the inverse of the encryption rules to determine the plaintext letters:
   4. If both letters are in the same row, replace each letter with the letter to its left in the same row. If at the beginning, wrap to the end.
   5. If both letters are in the same column, replace each letter with the letter above it in the same column. If at the top, wrap to the bottom.
   6. If the letters are in different rows and columns, create a rectangle using the two letters and replace each letter with the letter at the opposite corner of the rectangle.
5. Forming the Plaintext:
   1. Combine the decrypted letter pairs to obtain the original plaintext.

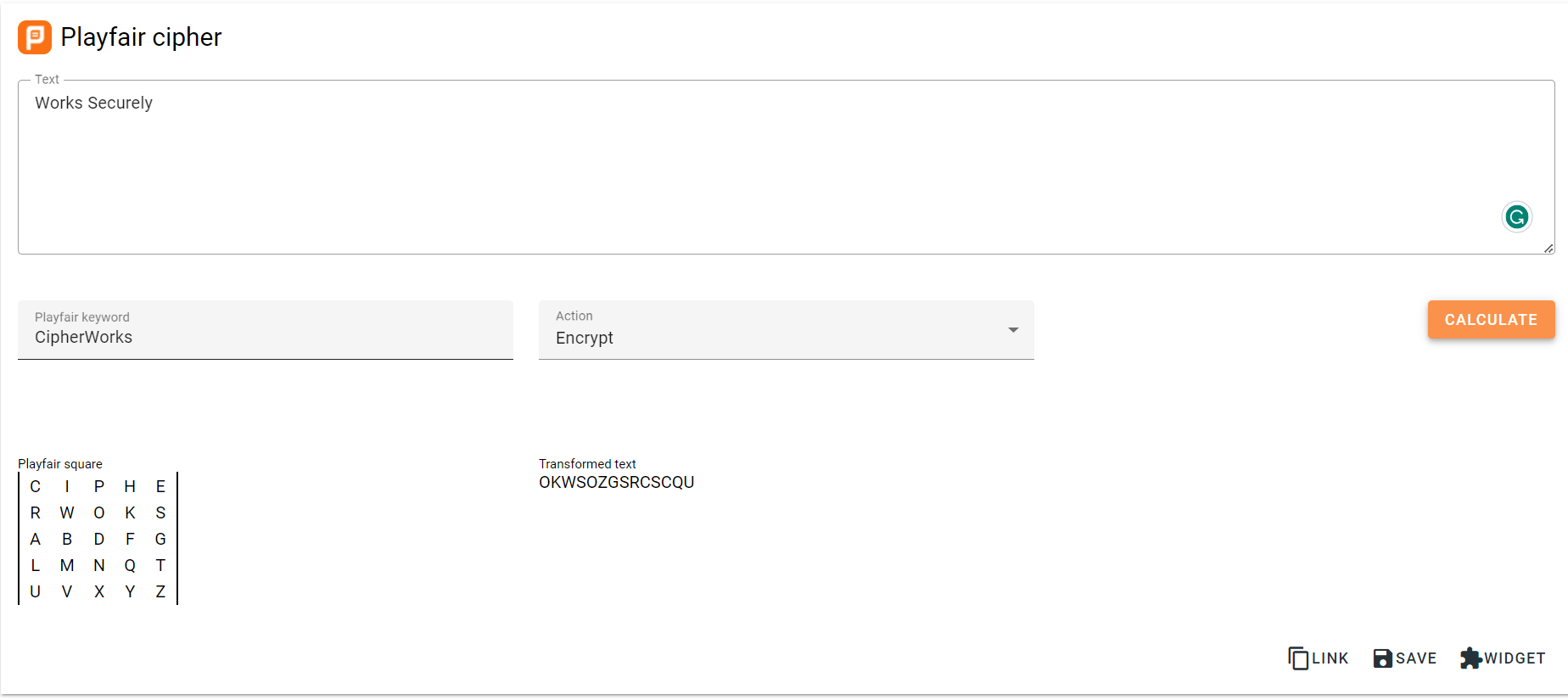
Implementation

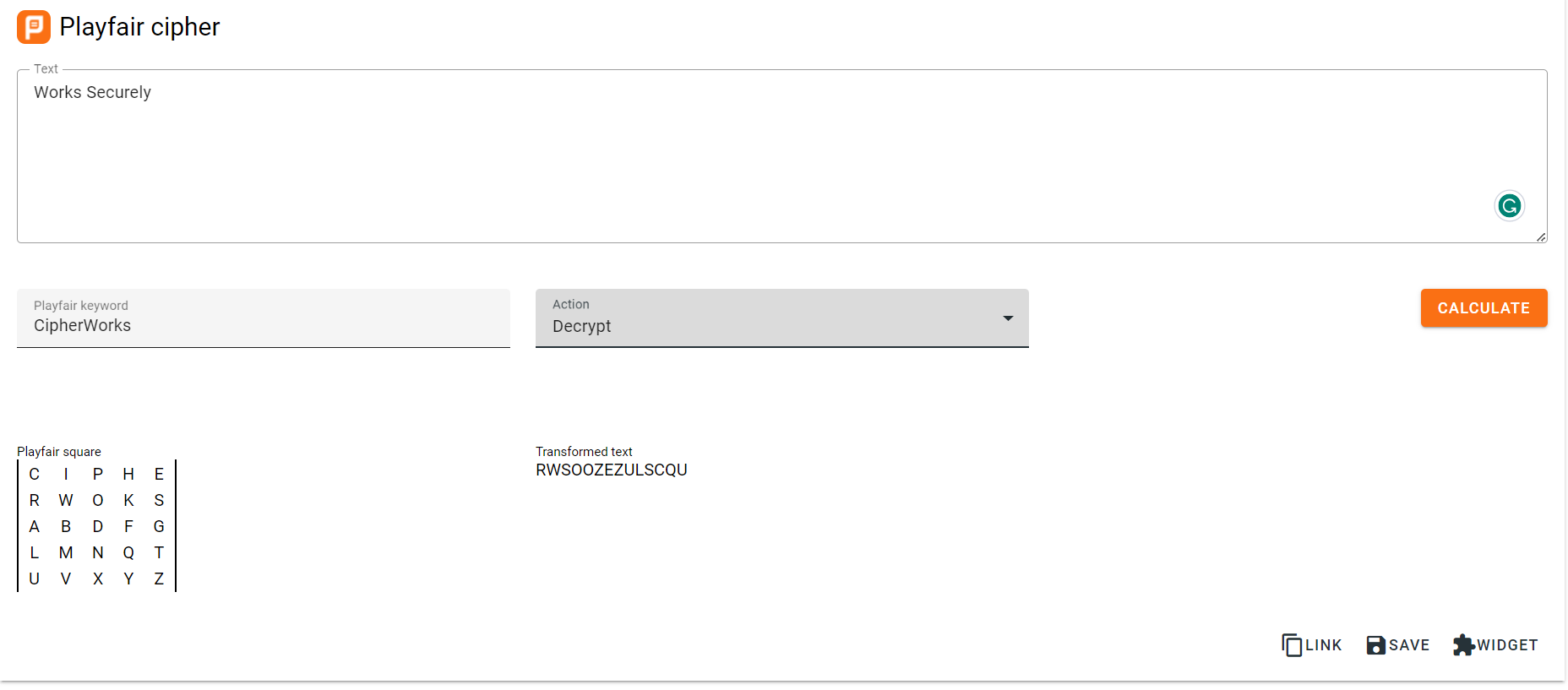




Output







Advantages :

1. Polyalphabetic Complexity: The Playfair cipher uses multiple shifting alphabets based on the key, making it more complex than simple substitution ciphers. This complexity enhances security by thwarting frequency analysis and letter pattern recognition.
2. Keyword-Based: The security of the Playfair cipher relies on the keyword used for encryption and decryption. Once the keyword is known, the encryption and decryption processes become relatively simple.
3. Historical Significance: The Playfair cipher played a pivotal role in the history of cryptography. It represents an advancement beyond monoalphabetic ciphers, demonstrating the evolution of cryptographic techniques over time.
4. Balanced Security: It offers a good balance between security and usability, making it suitable for various encryption needs, especially during its historical usage.

Disadvantages:

1. Key Management: Longer keywords are more secure, but they can become challenging to manage and remember. If a weak keyword is used, the security of the cipher is compromised.
2. Limited Key Space: The Playfair cipher has a limited key space due to the 5x5 key matrix. This limitation makes it susceptible to brute-force attacks when a weak keyword is used.
3. Complexity for Manual Encryption: The Playfair cipher's encryption and decryption processes can be complex and error-prone when done manually, especially for longer messages. It requires careful attention to the key matrix and rules.
4. Not Suitable for Modern Cryptography: While historically significant, the Playfair cipher is not suitable for modern cryptographic applications due to its limited key space and vulnerability to advanced cryptanalysis methods.

Cryptanalysis

Frequency Analysis:

1. Frequency analysis is less effective against the Playfair cipher compared to simpler ciphers.
2. The cipher uses digram substitution, making it more resistant to individual letter frequency analysis.
3. Frequency analysis can still provide clues if common digrams in the ciphertext correspond to known digrams in the plaintext language.
4. The Playfair cipher's complexity makes frequency analysis less straightforward.

Brute Force Attack:

1. A brute force attack against the Playfair cipher involves trying all possible key matrices.
2. The number of possible key matrices is determined by permutations and combinations of 25 unique characters (excluding "J").
3. The security of the Playfair cipher depends on the strength and randomness of the keyword.
4. Brute force attacks become more feasible with shorter or weaker keywords.
5. Modern computational power can potentially break the cipher through brute force, especially if the keyword is weak.

Application

1. Education and Learning: The Playfair cipher is often used as a teaching tool in cryptography courses. It helps students understand fundamental encryption concepts, such as substitution ciphers and the importance of key management.
2. Recreational Cryptography: In recreational or puzzle-solving contexts, the Playfair cipher can be employed to create cryptic messages or puzzles for fun events, puzzle hunts, escape rooms,tp or treasure hunts.
3. Historical Demonstrations: When reenacting historical scenarios or giving presentations on cryptography history, the Playfair cipher can be used to showcase how ciphers were applied in the past.
4. Classic Literature and Movies: Writers and filmmakers may include the Playfair cipher in their works, especially those set in historical or spy-themed contexts, to add an element of authenticity and intrigue.
5. Code and Puzzle Design: Designers of games, puzzles, and challenges might use the Playfair cipher to create cryptic codes and clues for players to decipher.

References :

[*https://github.com/AtriSaxena/PLAYFAIR-CIPHER-PYTHON-*](https://github.com/AtriSaxena/PLAYFAIR-CIPHER-PYTHON-)

[*https://www.geeksforgeeks.org/playfair-cipher-with-examples/*](https://www.geeksforgeeks.org/playfair-cipher-with-examples/)

[*https://people.eecs.berkeley.edu/~bh/pdf/v1ch12.pdf*](https://people.eecs.berkeley.edu/~bh/pdf/v1ch12.pdf)

Experiment – 7

**Aim**

Study and implement a program for Hill cipher to encrypt and decrypt the message

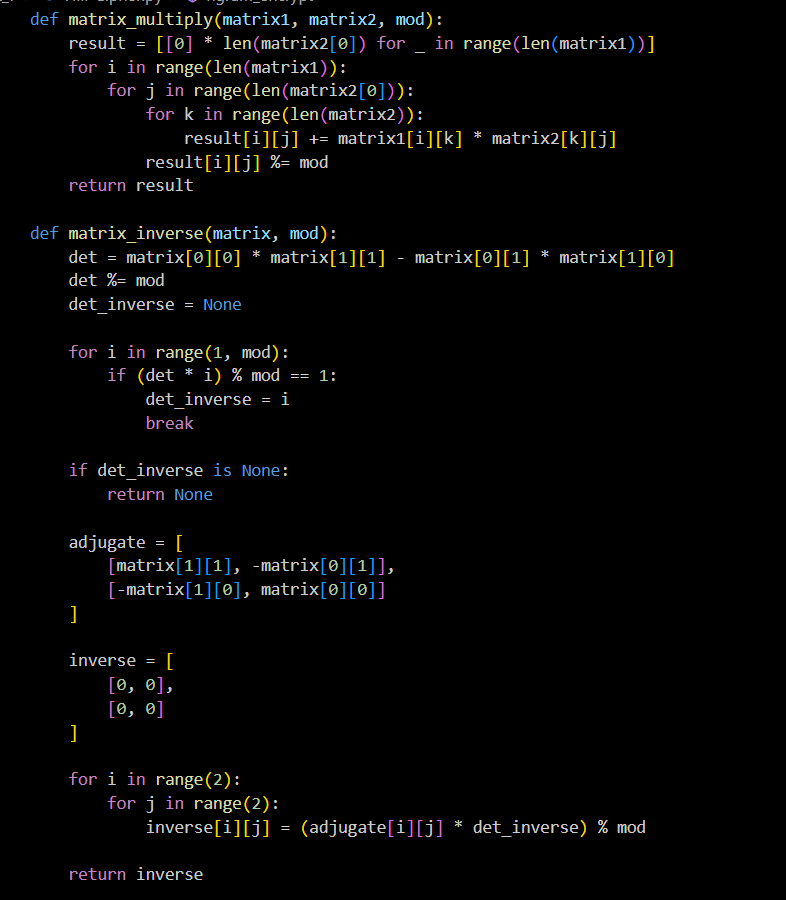
**Introduction**

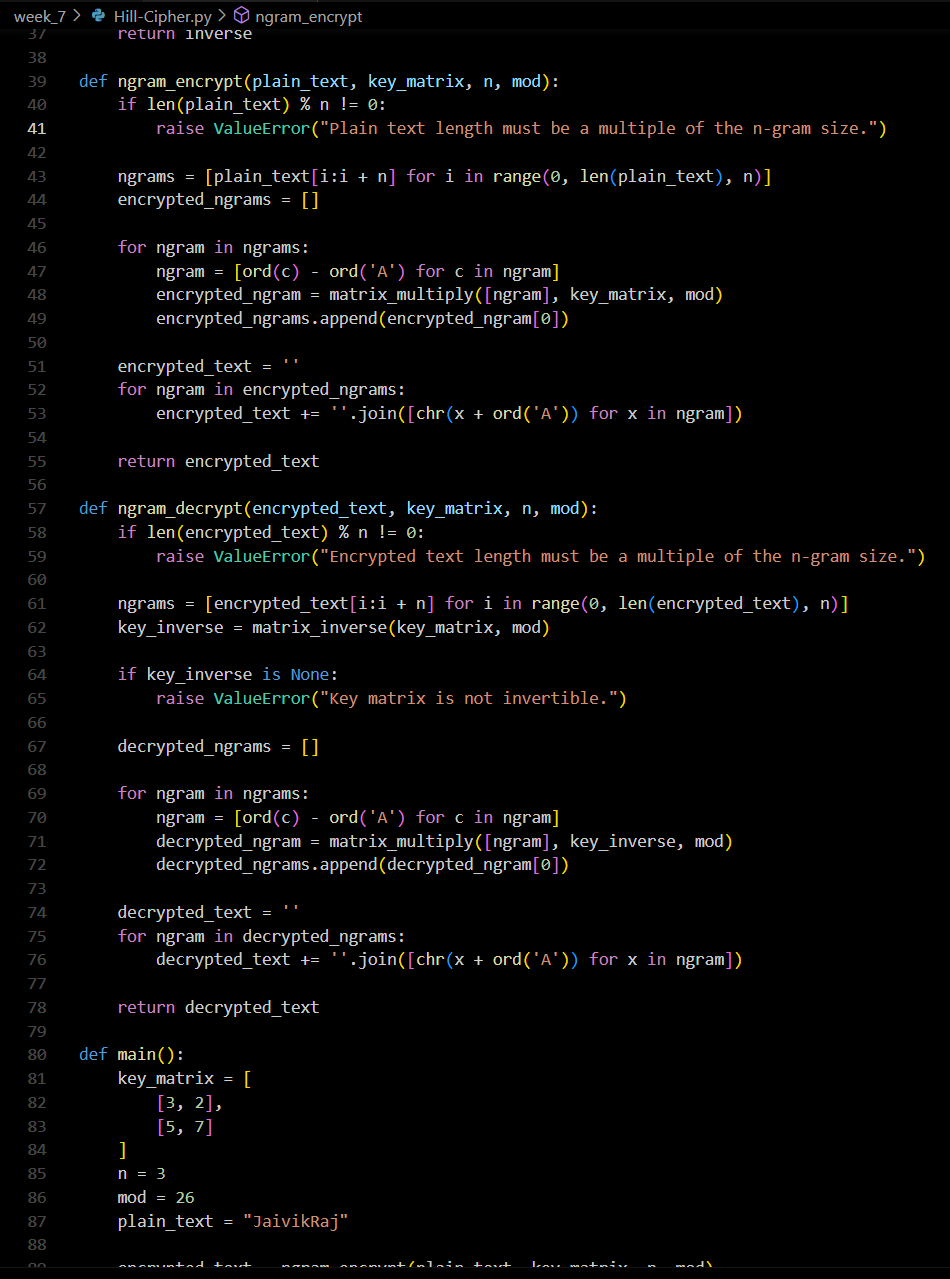
Hill Cipher is a cryptographic algorithm used for encrypting and decrypting text messages. It's a type of substitution cipher that operates on blocks of characters, typically pairs (bigrams) or triplets (trigrams). Unlike traditional ciphers like the Caesar cipher, Hill Cipher provides a more secure method of encryption.

**Methodology**

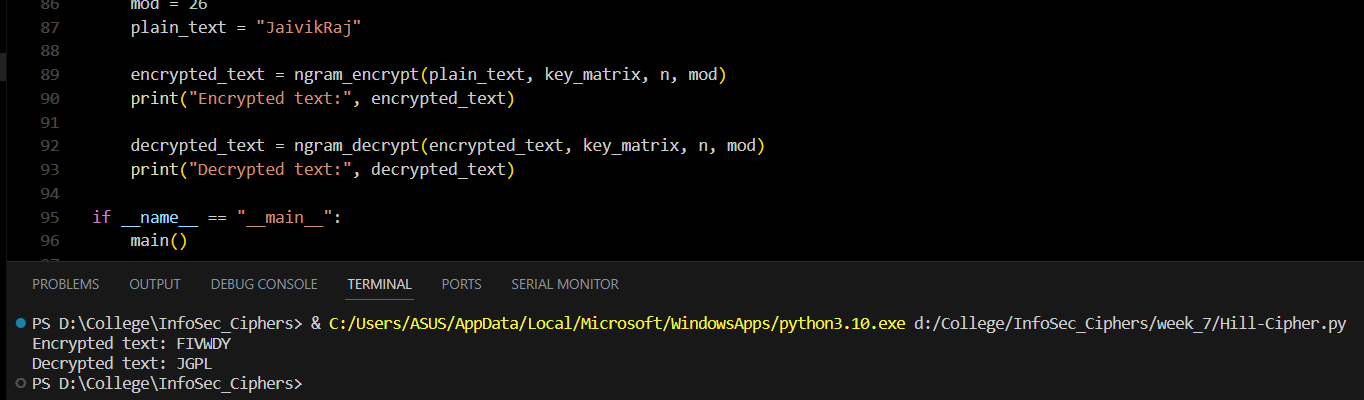
1. Key Generation:
   1. Select a square key matrix. The size of the matrix determines the block size for encryption (e.g., 2x2 for bigrams, 3x3 for trigrams).
   2. Ensure the key matrix is invertible, which is crucial for decryption.\
2. Encryption:
   1. Divide the plaintext into blocks based on the matrix size.
   2. Represent each block as a numerical vector, typically by mapping characters to numbers (e.g., A=0, B=1, ..., Z=25).
   3. Multiply each numerical vector by the key matrix.
   4. Take the result modulo a fixed value (usually 26 for letters).
   5. Convert the resulting numerical vector back to characters to obtain the ciphertext block.
3. Decryption:
   1. Divide the ciphertext into blocks matching the encryption block size.
   2. Apply the inverse of the key matrix to each numerical vector in the ciphertext blocks.
   3. Take the result modulo the same fixed value (e.g., 26).
   4. Convert the numerical vector back to characters to obtain the plaintext block.
4. Key Management:
   1. Safeguard the key matrix, as knowledge of the key is essential for decryption.
   2. Ensure the key matrix is kept secret and distributed securely to authorized parties.
5. Error Handling:
   1. Check that the plaintext and ciphertext block sizes are compatible with the key matrix size.
   2. Verify the key matrix's invertibility before decryption.
6. Security Considerations:
   1. Recognize that the security of the Hill Cipher depends on the key matrix. A poorly chosen key matrix can lead to vulnerabilities.
   2. Implement additional security measures, such as key rotation and authentication, if used in practical applications.

**Implementation**





**Output**



**Advantages:**

1. Security: Hill Cipher is more secure than simple substitution ciphers like the Caesar Cipher or Atbash Cipher. It offers a higher level of security due to its mathematical foundation.
2. Polygraphic Nature: Unlike monoalphabetic ciphers, Hill Cipher operates on groups of characters (bigrams or trigrams), making it resistant to frequency analysis attacks that work well on single-letter ciphers.
3. Variable Block Size: Hill Cipher can work with various block sizes (2x2, 3x3, etc.), allowing for flexibility in encryption and accommodating different types of messages.
4. Mathematical Rigor: Hill Cipher introduces the concept of matrix multiplication and modular arithmetic, making it an excellent educational tool for understanding cryptographic principles and mathematical techniques used in encryption.
5. Key Space: The number of possible key matrices grows exponentially with the matrix size, increasing the complexity of brute-force attacks.

**Disadvantages:**

1. Key Management: One of the major disadvantages is the challenge of managing and securely distributing the key matrices. A compromised key matrix can lead to decryption of sensitive data.
2. Matrix Inversion: The Hill Cipher relies on matrix inversion to decrypt messages. Matrix inversion can be computationally intensive, especially for larger matrix sizes, and may not always be possible if the determinant is not coprime with the modulus.
3. Block Alignment: Plaintext and ciphertext must align perfectly with the chosen block size. If the message length is not a multiple of the block size, padding or truncation is required.
4. Vulnerability to Known Plaintext Attacks: Hill Cipher can be vulnerable to known plaintext attacks, where an attacker knows or can guess parts of the plaintext and ciphertext pairs.
5. Lack of Perfect Secrecy: While it is more secure than simple substitution ciphers, Hill Cipher does not provide perfect secrecy, as patterns and repetitions in the plaintext can still be discerned.
6. Limited Applicability: Hill Cipher is not commonly used in modern cryptography due to its limitations and vulnerabilities when compared to more advanced encryption techniques like AES (Advanced Encryption Standard) or RSA (Rivest-Shamir-Adleman).

**Cryptanalysis**

Frequency Analysis:

1. Bigram and Trigram Frequencies: In Hill Cipher, blocks of characters (bigrams or trigrams) are encrypted. An attacker can analyze the frequency of these blocks in the ciphertext. Common bigrams or trigrams may correspond to common letter combinations or words in the plaintext.
2. Pattern Recognition: If certain blocks frequently appear in the ciphertext, an attacker might deduce the corresponding plaintext blocks. For example, in English text, the bigram "TH" is very common, so if a certain block frequently decrypts to "TH," it might provide a clue.
3. Attack Limitations: Frequency analysis is less effective against polyalphabetic ciphers because each block is encrypted using diffferent matrix transformations, making it harder to detect consistent patterrns.

Brute Force Attack:

1. Key Space Size: The size of the key space in Hill Cipher depends on the matrix size. For a 2x2 matrix with a modulo of 26 (typical for letters), there are 26^4 (about 456,976) possible matrices. For larger matrices, the key space grows exponentially.
2. Computational Complexity: Brute forcing the Hill Cipher can be computationally intensive, especially for larger matrices. It becomes impractical as the matrix size and modulo increase.
3. Determinant Constraints: To decrypt using a key matrix, its determinant must be invertible modulo the modulus (typically 26). Not all matrices meet this criterion, further limiting the brute force approach.
4. Known Plaintext Attack: In some cases, if an attacker has access to both plaintext and ciphertext pairs, they can narrow down the key search by using a known plaintext attack. This reduces the effective key space.

**Application**

1. Education and Learning: Hill Cipher is often used as an educational tool to introduce students to fundamental concepts of cryptography and linear algebra. It provides hands-on experience with matrix operations and modular arithmetic.
2. Historical Encryption: Hill Cipher has historical significance as one of the earliest examples of a polygraphic substitution cipher. It was used during World War II by various military forces and intelligence agencies, such as the United States and Germany.
3. Puzzle and Recreational Use: Enthusiasts and puzzle solvers sometimes use Hill Cipher as a recreational challenge. Cryptogram puzzles based on Hill Cipher can be found in puzzle books and games.
4. Cryptography Challenges: In cryptographic competitions and challenges, Hill Cipher might be employed as a part of a more complex cipher system. Competitors may need to decipher messages encrypted using the Hill Cipher to advance in the challenge.
5. Teaching Cryptanalysis: Hill Cipher is also used to teach cryptanalysis—the science of breaking codes and ciphers. Students can practice analyzing ciphertexts encrypted with Hill Cipher to improve their skills in cryptanalysis.
6. Steganography: In some cases, Hill Cipher is used within steganography, the art of concealing messages within other data. It can serve as one layer of encryption within a steganographic technique.

References :

<https://www.cryptool.org/en/cto/hill>

<https://www.geeksforgeeks.org/hill-cipher/>

<https://www.youtube.com/watch?v=-EQ8UomTrAQ>

Experiment – 8

**Aim**

Use Crypto++ library to implement encryption and decryption functions

for different block ciphers.

**Introduction**

The Crypto++ library is a powerful and versatile tool for implementing encryption and decryption functions using various block ciphers. It offers a comprehensive set of cryptographic algorithms and utilities that can be easily integrated into your applications, making it a go-to choice for developers seeking robust security features. Whether you need to protect sensitive data, secure communications, or meet regulatory compliance, Crypto++ simplifies the implementation of encryption and decryption using a range of block ciphers, ensuring the confidentiality and integrity of your digital assets.

**Methodology**

1. Library Integration: Download and integrate the Crypto++ library into your development environment.

2. Selecting a Block Cipher: Choose the specific block cipher you want to use. Crypto++ supports a variety of block ciphers, including AES, DES, Triple DES, and others.

3. Key Management: Generate and securely manage encryption keys. Use Crypto++'s key generation and management functions to create and store keys.

4. Data Preprocessing: Prepare the data you want to encrypt. Data preprocessing can include formatting, padding, or serialization, depending on the specific cipher and mode of operation.

5. Encryption: Use the chosen block cipher and the associated encryption mode (e.g., ECB, CBC, GCM) to encrypt your data. Crypto++ provides easy-to-use classes and functions for encryption.

6. Decryption: Implement decryption functions using the same block cipher and mode of operation. Crypto++ facilitates decryption with dedicated classes and functions.

**Implementation**

AES.c++

#include <iostream>

#include <cryptopp/aes.h>

#include <cryptopp/modes.h>

#include <cryptopp/filters.h>

using namespace CryptoPP;

int main() {

    byte key[AES::DEFAULT\_KEYLENGTH] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};

    byte iv[AES::BLOCKSIZE] = {0};

    std::string plainText = "This is a secret message.";

    std::string cipherText;

    std::string decryptedText;

    AES::Encryption aesEncryption(key, AES::DEFAULT\_KEYLENGTH);

    CBC\_Mode\_ExternalCipher::Encryption cbcEncryption(aesEncryption, iv);

    // Encrypt

    StringSource(plainText, true,

        new StreamTransformationFilter(cbcEncryption, new StringSink(cipherText)));

    // Decrypt

    AES::Decryption aesDecryption(key, AES::DEFAULT\_KEYLENGTH);

    CBC\_Mode\_ExternalCipher::Decryption cbcDecryption(aesDecryption, iv);

    StringSource(cipherText, true,

        new StreamTransformationFilter(cbcDecryption, new StringSink(decryptedText)));

    std::cout << "Original Text: " << plainText << std::endl;

    std::cout << "Encrypted Text: " << cipherText << std::endl;

    std::cout << "Decrypted Text: " << decryptedText << std::endl;

    return 0;

}

DES.c++

#include <iostream>

#include <cryptopp/des.h>

#include <cryptopp/modes.h>

#include <cryptopp/filters.h>

using namespace CryptoPP;

int main() {

    byte key[DES::DEFAULT\_KEYLENGTH] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};

    byte iv[DES::BLOCKSIZE] = {0};

    std::string plainText = "This is a secret message.";

    std::string cipherText;

    std::string decryptedText;

    DES::Encryption desEncryption(key, DES::DEFAULT\_KEYLENGTH);

    CBC\_Mode\_ExternalCipher::Encryption cbcEncryption(desEncryption, iv);

    // Encrypt

    StringSource(plainText, true,

        new StreamTransformationFilter(cbcEncryption, new StringSink(cipherText)));

    // Decrypt

    DES::Decryption desDecryption(key, DES::DEFAULT\_KEYLENGTH);

    CBC\_Mode\_ExternalCipher::Decryption cbcDecryption(desDecryption, iv);

    StringSource(cipherText, true,

        new StreamTransformationFilter(cbcDecryption, new StringSink(decryptedText)));

    std::cout << "Original Text: " << plainText << std::endl;

    std::cout << "Encrypted Text: " << cipherText << std::endl;

    std::cout << "Decrypted Text: " << decryptedText << std::endl;

    return 0;

}

**Output**

**Advantages :**

* Security: Crypto++ is a trusted and well-vetted cryptographic library that provides strong security features. It ensures the confidentiality and integrity of data, making it suitable for secure applications.
* Variety of Ciphers: Crypto++ supports a wide range of block ciphers, including AES, DES, Triple DES, Blowfish, and many more. This versatility allows you to choose the cipher that best fits your specific security requirements.
* Encryption Modes: The library supports various encryption modes, such as ECB, CBC, GCM, and CFB. You can choose the mode that aligns with your application's security needs.
* Ease of Use: Crypto++ provides a straightforward and consistent API for encryption and decryption operations, making it accessible to developers, including those with limited cryptographic expertise.
* Open Source: Crypto++ is open-source and released under a permissive license (Boost Software License). This allows you to use it freely, modify the code, and incorporate it into your projects without licensing concerns.

**Disadvantages:**

* Complexity: Crypto++ provides a wide range of cryptographic algorithms and features, which can be overwhelming for users who are not well-versed in cryptography. Implementing encryption and decryption correctly requires a good understanding of the library's capabilities.
* Steep Learning Curve: For developers new to cryptography, Crypto++ may have a steep learning curve. Proper use of the library often requires in-depth knowledge of cryptographic principles and best practices.
* Documentation: While Crypto++ has extensive documentation, some users find it challenging to navigate and locate the specific information they need. It may not always provide clear examples or use cases for certain features.
* Performance Overhead: Although Crypto++ is optimized for performance, cryptographic operations inherently introduce computational overhead. In high-performance or real-time systems, this can be a concern, especially for large datasets.
* Compatibility Issues: Crypto++ is a C++ library, and integrating it into applications built with other programming languages can be challenging. Interoperability with non-C++ applications may require additional effort.
* Lack of Cryptanalysis: The library does not come with built-in cryptanalysis tools. You need to rely on external tools and expertise to assess the security of your cryptographic implementations.
* Frequent Updates: Security is an ever-evolving field, and cryptographic libraries need to be updated to address vulnerabilities and weaknesses. Users must stay vigilant about library updates and incorporate them into their applications.

**Cryptanalysis**

* Brute Force Attack: A brute force attack involves trying every possible key until the correct one is found. For block ciphers used in modern encryption, the keyspace is extremely large, making a brute force attack impractical.

Crypto++ provides strong encryption algorithms like AES with key lengths of 128, 192, or 256 bits. The number of possible keys is 2^128, 2^192, or 2^256, which are astronomically large numbers. Brute force attacks on such keys are computationally infeasible and would take longer than the age of the universe to complete with current technology.

* Frequency Analysis: Frequency analysis is a technique used to break classical ciphers, which substitute one symbol for another (e.g., letter substitutions). It relies on the frequency of symbols in a language to deduce the key. Modern cryptographic systems, such as those implemented in Crypto++, do not operate at the level of individual letters or symbols. Instead, they manipulate binary data using complex mathematical algorithms.

Block ciphers like AES create ciphertext with no recognizable patterns, making frequency analysis irrelevant. The ciphertext produced is essentially a stream of random bits, so frequency analysis doesn't apply.

**Application**

* Data Encryption: Crypto++ is used to encrypt sensitive data, protecting it from unauthorized access. This is applicable in databases, file storage, and secure communication.
* Secure Communication: Crypto++ is often used to secure network communication in applications like VPNs, secure chat apps, and email encryption.
* Digital Signatures: It is used to generate and verify digital signatures for authentication and data integrity. This is essential in secure email, software updates, and secure document signing.
* Secure File Storage: Crypto++ helps encrypt files and data stored on devices or in the cloud to prevent unauthorized access.
* Password Hashing: It is used to securely hash and verify passwords, protecting user credentials in authentication systems.
* Secure Protocols: Crypto++ is employed in secure communication protocols such as SSL/TLS, SSH, and IPsec.

**References:**

Experiment – 9

**Aim**

Study and implement RSA encryption and decryption functions.

**Introduction**

The implementation of RSA is a multi-step process, involving key generation, encryption, and decryption. To begin, a pair of keys is generated – a public key and a private key. The public key is used for encryption, while the private key is used for decryption. The security of RSA relies on the fact that while it's computationally easy to multiply two large prime numbers, factoring the product of two large primes is significantly more difficult, even for modern computers.

**Methodology**

implementation of the RSA algorithm involves several key steps, including key generation, encryption, and decryption. Here's a high-level methodology for implementing RSA:

1. Key Generation:

Step 1: Choose Two Large Prime Numbers: Select two distinct, large prime numbers, typically denoted as p and q.

Step 2: Compute N and Φ(N): Calculate N = p \* q, and Φ(N) = (p - 1) \* (q - 1), where Φ(N) is Euler's totient function.

Step 3: Choose the Public Exponent (e): Select a public exponent e that is relatively prime to Φ(N). Common choices include 3 or 65537.

Step 4: Calculate the Private Exponent (d): Calculate the private exponent d, which is the modular multiplicative inverse of e modulo Φ(N). This can be done using the Extended Euclidean Algorithm.

Step 5: Public and Private Keys: The public key consists of (N, e), and the private key is (N, d). Keep the private key secure.

2.. Encryption:

To encrypt a message (plaintext M), the sender uses the recipient's public key (N, e).

Convert the plaintext to an integer value.

Calculate the ciphertext C as C ≡ M^e (mod N).

3. Decryption:

To decrypt the ciphertext C, the recipient uses their private key (N, d).

Calculate the plaintext M as M ≡ C^d (mod N).

**Implementation**

The Mathematics behind RSA

In RSA, we have two large primes p and q, a modulus N = pq, an encryption exponent e and a decryption exponent d that satisfy ed = 1 mod (p - 1)(q - 1). The public key is the pair (N,e) and the private key is d.

To encrypt a message M, compute  
      C = Me mod N.

We want to show  
  
      M = Cd mod N,  
  
i.e., that we can decrypt by raising the ciphertext C to the d power and reducing the result modulo N. But first we must take a slight mathematical detour.

Two positive integers m and n are said to be *relatively prime* if they have no common factors other than 1. For example, though both 10 and 9 are composite numbers, they are relatively prime, since they have no factor (other than 1) in common.

For a positive integer n, define φ(n) to be the number of integers less than n that are relatively prime with n. For example, φ(12) = 4, since only 11, 7, 5 and 1 are less than 12 and relatively prime to 12, while φ(7) = 6. In fact, for any prime number p we have φ(p) = p - 1.

Suppose the prime factorization of n is given by  
  
      n = p1k1 p2k2 ... prkr  
  
Then it can be shown that  
  
      φ(n) = n (1 - 1/p1) (1 - 1/p2) ... (1 - 1/pr)  
  
Note that for the RSA modulus N = pq this result implies  
  
      φ(N) = (p - 1)(q - 1)

The final mathematical result we need is Fermat's Little Theorem. This theorem is usually stated as

**Fermat's Little Theorem**: If p is prime and p does not divide x, then xp - 1 = 1 mod p

However, a generalization of Fermat's Little Theorem (sometimes known as Euler's Theorem) is more directly applicable to RSA. This theorem states that

**Euler's Theorem**: If x is relatively prime to n then xφ(n) = 1 mod n

Now back to RSA decryption. We want to show that  
  
      M = Cd = (Me)d = Med mod N.

Recall that ed = 1 mod (p - 1)(q - 1). Also, since N = pq, as noted above, we have  
  
      φ(N) = (p - 1)(q - 1)  
  
and it follows that  
  
      ed = 1 mod φ(N).  
  
Then by the definition of "mod", there is some k such that ed - 1 = kφ(N). We now have  
  
      Med = M(ed - 1) + 1 = M Med - 1 = M Mkφ(N) mod N

Finally, Fermat's Little Theorem (in the form of Euler's Theorem) can be applied to yield the desired result  
  
      Med = M (Mk)φ(N) = M mod N = M.

#include <iostream>

#include <cmath>

using namespace std;

// Function to compute the greatest common divisor (GCD)

int gcd(int a, int h) {

    int temp;

    while (1) {

        temp = a % h;

        if (temp == 0)

            return h;

        a = h;

        h = temp;

    }

}

int main() {

    // Two random prime numbers

    double p = 3;

    double q = 7;

    // First part of public key: n

    double n = p \* q;

    // Finding the other part of the public key (e stands for encrypt)

    double e = 2;

    double phi = (p - 1) \* (q - 1);

    while (e < phi) {

        // e must be co-prime to phi and smaller than phi.

        if (gcd(e, phi) == 1)

            break;

        else

            e++;

    }

    // Private key (d stands for decrypt), choosing d such that it satisfies d\*e = 1 + k \* totient

    int k = 2; // A constant value

    double d = (1 + (k \* phi)) / e;

    // Message to be encrypted

    double msg = 12;

    cout << "Message data = " << msg << endl;

    // Encryption c = (msg ^ e) % n

    double c = pow(msg, e);

    c = fmod(c, n);

    cout << "Encrypted data = " << c << endl;

    // Decryption m = (c ^ d) % n

    double m = pow(c, d);

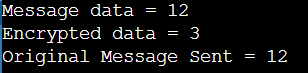
    m = fmod(m, n);

    cout << "Original Message Sent = " << m << endl;

    return 0;

}

**Output**



**Advantages:**

* Security: RSA is based on the difficulty of factoring large composite numbers, which is a computationally intensive problem. As long as the key length is sufficient, RSA provides a high level of security, and its resistance to attacks remains strong.
* Public and Private Keys: RSA uses a pair of keys, one for encryption (public key) and the other for decryption (private key). This separation of keys provides a robust method for securing data transmission and digital signatures.
* Data Confidentiality: RSA is excellent for securing data during transmission. Messages encrypted with the recipient's public key can only be decrypted by the recipient who holds the corresponding private key.
* Data Integrity: RSA can be used for digital signatures, which ensure that the data has not been tampered with during transmission. The sender signs the data with their private key, and the recipient can verify it using the sender's public key.
* Authentication: RSA can be used for user or entity authentication. By verifying digital signatures, users can be certain of the origin and integrity of received data.
* Key Exchange: RSA is used in key exchange protocols, such as Diffie-Hellman key exchange. It enables two parties to securely exchange a shared secret key over an untrusted communication channel.

**Disadvantages:**

* Key Length: To maintain adequate security, RSA key lengths must be relatively long, which can result in slower encryption and decryption processes. As computing power advances, longer key lengths are required, further impacting performance.
* Computational Intensity: RSA is computationally intensive, particularly when using long key lengths. This can be a drawback in resource-constrained environments or when quick processing times are essential.
* Key Management: RSA requires secure key management. Protecting private keys is of paramount importance, as their compromise could lead to the exposure of sensitive data or the ability to forge digital signatures.
* Vulnerability to Quantum Computers: RSA relies on the difficulty of factoring large composite numbers, which can potentially be broken by quantum computers using Shor's algorithm. This poses a future risk to RSA's security, as quantum computing technology advances.
* Performance Impact with Long Messages: Encrypting long messages with RSA can result in performance issues because RSA is designed for encrypting relatively short blocks of data. Often, RSA is used to encrypt a symmetric encryption key that is used to encrypt the actual message.

**Cryptanalysis**

Brute Force Attack:

RSA encryption relies on the mathematical difficulty of factoring large composite numbers, which is a complex problem.

The security of RSA is based on the principle that even with the most advanced computing technology available today, it would take an impractical amount of time to factor the product of two large prime numbers (the modulus N) used in the RSA keys.

Therefore, attempting a brute force attack by trying all possible combinations to find the private key in an RSA system is considered infeasible with current technology. The time required for such an attack would be astronomical, often beyond the age of the universe.

Frequency Analysis:

Frequency analysis is a method typically used in cryptanalysis to analyze patterns in ciphertext. It is most relevant for classical ciphers like the Caesar cipher, where specific patterns can reveal the key.

RSA encryption doesn't rely on substitution or simple patterns in the plaintext. It uses modular exponentiation with very large numbers, and the ciphertext doesn't exhibit easily detectable patterns.

Frequency analysis is not applicable to RSA because it operates at the mathematical level and doesn't involve direct letter or symbol substitutions.

**Application**

* Secure Data Transmission: RSA is often used for securing data transmission over insecure networks. It ensures that data sent from one party to another can only be decrypted by the intended recipient.
* Digital Signatures: RSA is employed for creating digital signatures. By signing a document or message with their private key, a sender can prove their identity and the integrity of the data to the recipient who can verify it using the sender's public key.
* Secure Email: RSA is commonly used in email encryption and digital signatures to protect the confidentiality and integrity of email messages. PGP (Pretty Good Privacy) and S/MIME (Secure/Multipurpose Internet Mail Extensions) are popular email encryption protocols that use RSA.
* SSL/TLS Encryption: RSA is a fundamental component of the SSL (Secure Sockets Layer) and its successor, TLS (Transport Layer Security) protocols. It secures online communication, such as web browsing, by encrypting data transferred between a web server and a client (e.g., a web browser).
* Secure File Transfer: RSA is employed in secure file transfer protocols and applications, ensuring that files transferred between parties are encrypted and remain confidential.
* VPN Encryption: Virtual Private Networks (VPNs) use RSA to establish secure connections over the internet, allowing users to access remote networks and resources securely.
* Secure Chat and Messaging Apps: Many secure chat and messaging applications use RSA for end-to-end encryption, preventing unauthorized access to the content of messages.
* Secure VoIP Calls: Voice over Internet Protocol (VoIP) services often utilize RSA encryption to secure voice calls, ensuring that the call content remains confidential.
* Secure Remote Access: RSA is used to establish secure remote access to systems, servers, or cloud resources, ensuring that only authorized users can access them.
* Secure Digital Certificates: RSA is used in digital certificates (e.g., X.509 certificates) to verify the identity of websites, devices, or individuals on the internet.

**References:**

[**https://www.thecrazyprogrammer.com/2017/03/rsa-algorithm.html**](https://www.thecrazyprogrammer.com/2017/03/rsa-algorithm.html)

[**https://www.geeksforgeeks.org/rsa-algorithm-cryptography/**](https://www.geeksforgeeks.org/rsa-algorithm-cryptography/)

[**https://www.cs.sjsu.edu/~stamp/CS265/SecurityEngineering/chapter5\_SE/RSAmath.html#:~:text=The%20Mathematics%20behind%20RSA&text=In%20RSA%2C%20we%20have%20two,the%20private%20key%20is%20d.&text=C%20%3D%20Me%20mod%20N**](https://www.cs.sjsu.edu/~stamp/CS265/SecurityEngineering/chapter5_SE/RSAmath.html#:~:text=The%20Mathematics%20behind%20RSA&text=In%20RSA%2C%20we%20have%20two,the%20private%20key%20is%20d.&text=C%20%3D%20Me%20mod%20N)**.**

Experiment – 10

**Aim**

Use RSA for generation and Verification of digital signature on file

**Introduction**

RSA, or Rivest-Shamir-Adleman, is a widely-used cryptographic algorithm that is vital for generating and verifying digital signatures on electronic files. It provides a way to validate the source and integrity of digital data. In the process, RSA uses a pair of keys, public and private, to create a unique digital signature for a file's content, ensuring that it hasn't been altered and comes from the claimed sender. This makes RSA indispensable for securing electronic communication and document management, especially in legal, financial, and governmental contexts.

**Methodology**

To generate and verify digital signatures on files using the RSA algorithm, you can follow this methodology:

Digital Signature Generation (Sender):

* File Preparation: The sender prepares the file to be signed. Ensure that the file is in a consistent and unaltered state.
* Key Generation: If not already done, the sender generates an RSA key pair. This involves selecting appropriate key lengths (e.g., 2048 bits or 4096 bits) and generating the public and private keys.
* File Hashing: The sender calculates a cryptographic hash (e.g., SHA-256) of the file's content. The hash should be a fixed length and unique to the file.
* Signing the Hash:
* The sender signs the calculated hash using their private key and the RSA digital signature algorithm. The resulting signature is unique to both the file and the sender.
* Attach Signature: The sender attaches the generated digital signature to the file. This signature is often stored in a separate file or appended to the original file.

Digital Signature Verification (Recipient):

* File and Signature Reception: The recipient receives both the file and the attached digital signature.
* Key Acquisition: The recipient obtains the sender's public key. This can be through a public key server, a certificate authority, or directly from the sender.
* File Hashing: The recipient calculates the cryptographic hash of the received file using the same hash function used by the sender.
* Signature Verification: The recipient uses the sender's public key and the RSA digital signature verification algorithm to check the authenticity of the attached signature. If the signature is valid, it confirms that the file hasn't been tampered with and is from the claimed sender.
* Comparison: The recipient compares the calculated hash of the received file with the hash value that was signed by the sender. If they match, the file is considered authentic and unaltered.

**Implementation**

import random

import hashlib

# Function to check if a number is prime

def is\_prime(num):

    if num <= 1:

        return False

    for i in range(2, int(num\*\*0.5) + 1):

        if num % i == 0:

            return False

    return True

# Function to find the greatest common divisor (GCD) of two numbers

def gcd(a, b):

    while b:

        a, b = b, a % b

    return a

# Function to find the modular multiplicative inverse

def mod\_inverse(a, m):

    m0, x0, x1 = m, 0, 1

    while a > 1:

        q = a // m

        m, a = a % m, m

        x0, x1 = x1 - q \* x0, x0

    return x1 + m0 if x1 < 0 else x1

# Function to create a cryptographic hash of a message

def hash\_message(message):

    sha256 = hashlib.sha256()

    sha256.update(str(message).encode())

    return int(sha256.hexdigest(), 16)

# Generate two random prime numbers (p and q)

while True:

    p = random.randint(100, 200)

    if is\_prime(p):

        break

while True:

    q = random.randint(200, 300)

    if is\_prime(q):

        break

n = p \* q

phi = (p - 1) \* (q - 1)

# Find an encryption key (e) such that 1 < e < phi and gcd(e, phi) = 1

while True:

    e = random.randint(2, phi - 1)

    if gcd(e, phi) == 1:

        break

# Calculate the decryption key (d) using the modular multiplicative inverse of e

d = mod\_inverse(e, phi)

# Message to be signed

message = 42

# Hash the message

hashed\_message = hash\_message(message)

# Sign the hashed message

signature = pow(hashed\_message, d, n)

# Verify the signature

if pow(signature, e, n) == hashed\_message:

    print("Signature verified: The message is authentic.")

else:

    print("Signature verification failed: The message may be tampered with.")

import random

import math

import hashlib

# Function to check if a number is prime

def is\_prime(num):

    if num <= 1:

        return False

    if num <= 3:

        return True

    if num % 2 == 0 or num % 3 == 0:

        return False

    i = 5

    while i \* i <= num:

        if num % i == 0 or num % (i + 2) == 0:

            return False

        i += 6

    return True

# Function to find the greatest common divisor (GCD) of two numbers

def gcd(a, b):

    while b:

        a, b = b, a % b

    return a

# Function to find the modular multiplicative inverse

def mod\_inverse(a, m):

    m0, x0, x1 = m, 0, 1

    while a > 1:

        q = a // m

        m, a = a % m, m

        x0, x1 = x1 - q \* x0, x0

    return x1 + m0 if x1 < 0 else x1

# Function to create a cryptographic hash of a message

def hash\_message(message):

    sha256 = hashlib.sha256()

    sha256.update(str(message).encode())

    return int(sha256.hexdigest(), 16)

# Function to generate a key pair

def generate\_keypair(bits):

    p = generate\_prime(bits)

    q = generate\_prime(bits)

    n = p \* q

    phi = (p - 1) \* (q - 1)

    while True:

        e = random.randrange(2, phi)

        if math.gcd(e, phi) == 1:

            break

    d = mod\_inverse(e, phi)

    public\_key = (n, e)

    private\_key = (n, d)

    return public\_key, private\_key

# Function to generate a random prime number

def generate\_prime(bits):

    while True:

        num = random.getrandbits(bits)

        if is\_prime(num):

            return num

# Function to sign a message

def sign\_message(private\_key, message):

    n, d = private\_key

    hashed\_message = hash\_message(message)

    signature = pow(hashed\_message, d, n)

    return signature

def verify\_signature(public\_key, private\_key\_receiver, encrypted\_value):

    n\_receiver, d\_receiver = private\_key\_receiver

    n\_sender, e\_sender = public\_key

    # Decrypt the encrypted value with the sender's public key

    decrypted\_signature = pow(encrypted\_value, e\_sender, n\_sender)

    # Calculate the hashed message from the decrypted signature

    hashed\_message = hash\_message(decrypted\_signature)

    # Compare the hashed message with the decrypted signature

    return hashed\_message == decrypted\_signature

# Generate the key pairs with larger bit lengths for better security.

public\_key, private\_key = generate\_keypair(2048)

public\_key\_receiver, private\_key\_receiver = generate\_keypair(2048)

# Message to be signed

message = "Wolfing here"

# Sign the message

signature = sign\_message(private\_key, message)

# Verify the signature

if verify\_signature(public\_key, private\_key\_receiver, signature):

    print("Signature is valid.")

else:

    print("Signature is invalid.")

**Output**

**Advantages:**

* Data Integrity: Digital signatures provide a means to ensure the integrity of electronic files. If even a single bit of the file is altered, the digital signature verification will fail, indicating potential tampering.
* Authentication: Digital signatures authenticate the source of the file. Recipients can be confident that the file came from the claimed sender, as only the sender's private key could have created the corresponding signature.
* Non-Repudiation: Digital signatures provide non-repudiation, meaning the sender cannot deny the authenticity of the file or the act of signing it. This is particularly important in legal and financial contexts.
* Security: RSA, as a public-key cryptosystem, ensures that the private key used for signing remains confidential. Even if the public key is widely distributed, it cannot be used to deduce the private key.
* Flexibility: Digital signatures work with any type of electronic file, whether it's a text document, image, software, or any other digital content.

**Disadvantages:**

* Key Management Complexity: Managing the secure storage and distribution of private keys is a critical challenge. If private keys are compromised, it can lead to the compromise of all associated digital signatures.
* Scalability: As the volume of digital signatures increases, the management and storage of key pairs and signed documents can become complex and resource-intensive.
* Performance Impact: RSA signature generation and verification can be computationally intensive, particularly with long key lengths. This can slow down processes in applications with high throughput requirements.
* Key Length Requirements: To maintain adequate security in the face of advancing computing power, RSA key lengths must be increased over time. Longer key lengths mean increased processing requirements.
* Security Concerns with Quantum Computing: The emergence of quantum computing threatens the security of RSA-based encryption and digital signatures. Quantum computers may eventually factor large numbers more efficiently, making existing RSA signatures vulnerable.

**Cryptanalysis**

* Brute Force Attack:

RSA's security relies on the difficulty of factoring the product of two large prime numbers, which is known as the RSA modulus (N).

Brute force attacks involve trying every possible key to break an encryption scheme. In RSA, this means attempting to factor N by systematically testing all possible prime number combinations. This is computationally infeasible for large N values used in RSA (e.g., 2048 or 4096 bits). Even with the most powerful computers and resources available today, factoring such large numbers would take an impractical amount of time.

* Frequency Analysis:

Frequency analysis is a technique used for breaking classical ciphers by analyzing patterns in ciphertext, typically based on the frequency of letters or symbols in a language.

RSA doesn't operate at the level of individual letters or symbols in a language. It is based on mathematical operations using large numbers (modular arithmetic), making it immune to frequency analysis. RSA digital signatures do not exhibit easily detectable patterns in the way classical ciphers do.

**Application**

* Secure Email Communication: Digital signatures are used in email communication to verify the authenticity of the sender and ensure the integrity of the email's content. PGP (Pretty Good Privacy) and S/MIME (Secure/Multipurpose Internet Mail Extensions) are widely used email encryption standards that employ RSA for digital signatures.
* E-commerce Transactions: In online shopping and financial transactions, digital signatures help confirm the authenticity of payment requests, ensuring that they come from legitimate sources. This is crucial in preventing fraud and unauthorized transactions.
* Software and Firmware Updates: Software vendors and manufacturers of electronic devices use digital signatures to sign their software and firmware updates. This guarantees that the updates have not been tampered with and come from legitimate sources.
* Government and Legal Documents: Digital signatures are used to authenticate and ensure the integrity of government documents, contracts, and legal agreements. This is especially important in jurisdictions where digital signatures carry legal weight.
* Document and Data Integrity: Industries like healthcare and finance use digital signatures to verify the integrity of patient records, financial transactions, and sensitive data, ensuring that it has not been altered or tampered with.

**References:**