

CRYPTOGRAPHY AND HASHING TECHNIQUES

REPORT OF PROJECT SUBMITTED FOR PARTIAL FULFILLMENT OF THE
REQUIREMENT FOR THE DEGREE OF
BACHELOR OF COMPUTER APPLICATION

By

SURANJANA BANERJEE

REGISTRATION NO – 211641001210001 OF 2021-22

UNIVERSITY ROLL NO – 16401221003

UNDER THE SUPERVISION OF

Mr. SRIKANTA SEN

[Assistant Professor, George College of Management and Science]



AT

GEORGE COLLEGE OF MANAGEMENT AND SCIENCE

[Affiliated to Maulana Abul Kalam Azad University of Technology]

B.B.T. ROAD, KOLKATA – 141

JUNE – 2024

GEORGE COLLEGE OF MANAGEMENT AND SCIENCE

KOLKATA – 700141, INDIA



CERTIFICATE

The report of the Project titled **Cryptography and Hashing Techniques** submitted by SURANJANA BANERJEE (Roll No.: 16401221003 of BCA 6TH Semester Of 2024) has been prepared under our supervision for the partial fulfillment of the Requirements for BCA degree awarded by MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

The report is hereby forwarded.

Mr. SRIKANTA SEN

Dept. of BCA

(Internal Supervisor)

ACKNOWLEDGEMENT

I express my sincere gratitude to Mr. Srikanta Sen of Department of BCA, GCMS and for extending their valuable times for me to take up this problem as a Project. I am also indebted to Mr. Akash Agasti, Mr. Prasun Banerjee for their unconditional help and inspiration.

Date: 2024

SURANJANA BANERJEE

Reg. No.: 211641001210001 OF 2021-22

Roll No.: 16401221003

BCA 6th Semester 2024,

George College of Management and Science

GEORGE COLLEGE OF MANAGEMENT AND SCIENCE

[Affiliated to Maulana Abul Kalam Azad University of Technology]

B.B.T. ROAD, KOLKATA – 141



CERTIFICATE of ACCEPTANCE

The report of the Project titled Cryptography and Hashing Techniques submitted by SURANJANA BANERJEE (Roll No.: 16401221003 of BCA 6th Semester of 2024) is hereby recommended to be accepted for the partial fulfillment of the requirements for BCA degree in MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY.

Name of the Examiner

Signature with Date

1.

.....

2.

.....

3.

.....

4.

.....

Project Proposal: Cryptography and Hashing Techniques

Introduction:

As a final year BCA student, I am eager to delve into the fascinating world of cryptography and hashing techniques. Cryptography plays a crucial role in securing data and communication in today's digital age. With the increasing reliance on digital platforms for sensitive transactions and communication, understanding cryptography and hashing techniques is paramount. This project aims to explore various cryptographic algorithms, their applications, and the significance of hashing techniques in data security.

Objectives:

- To understand the fundamentals of cryptography and hashing.
- To explore different cryptographic algorithms such as RSA, AES, and DES.
- To analyze the applications of cryptography in data encryption, digital signatures, and secure communication.
- To investigate the role of hashing techniques in data integrity and password storage.
- To implement selected cryptographic algorithms and hashing techniques in practical scenarios.
- To evaluate the performance and security aspects of implemented algorithms.

Methodology:

- **Literature Review:** Conduct an extensive review of academic papers, textbooks, and online resources to understand the theoretical foundations of cryptography and hashing.
- **Algorithm Exploration:** Study various cryptographic algorithms such as RSA, AES, DES, and hashing techniques like SHA-256 and MD5. Analyze their working principles, strengths, weaknesses, and real-world applications.
- **Implementation:** Select a few cryptographic algorithms and hashing techniques for implementation using programming languages such as Python. Develop programs for encryption, decryption and hashing functions.
- **Practical Applications:** Design and execute experiments to demonstrate the practical applications of implemented algorithms. This could include encrypting and decrypting messages, generating digital signatures, and verifying data integrity using hashing.
- **Security Analysis:** Assess the security aspects of implemented algorithms by analyzing their resistance to common cryptographic attacks such as brute force, chosen plaintext, and chosen ciphertext attacks.

Deliverables:

- Research paper documenting the theoretical foundations, implementation details, and experimental results.
- Software prototypes demonstrating the practical applications of cryptographic algorithms and hashing techniques.
- Detailed report evaluating the performance and security aspects of implemented algorithms.

Timeline:

- **Week 1-2:** Conduct literature review and familiarize with cryptographic concepts.
- **Week 3-4:** Explore different cryptographic algorithms and hashing techniques.
- **Week 5-8:** Implement selected algorithms and hashing techniques.
- **Week 9-11:** Design and execute practical experiments.
- **Week 12-14:** Analyze experimental results, evaluate performance, and assess security.
- **Week 15-16:** Prepare research paper and project report.

Budget:

The project budget will primarily cover expenses related to software development tools, research materials, and any miscellaneous costs. As a student, I intend to utilize open-source software and freely available resources to minimize the budgetary requirements.

Conclusion:

This project will provide valuable insights into the principles, applications, and practical implementations of cryptography and hashing techniques. By gaining hands-on experience with cryptographic algorithms, I aim to enhance my understanding of data security and cryptography, which will be beneficial for my academic and professional growth. Additionally, the findings of this project could contribute to the advancement of knowledge in the field of cybersecurity.

Project Supervisor:

Mr. SRIKANTA SEN

[Assistant Professor, George College of Management and Science]

We appreciate your consideration of this project proposal and look forward to the opportunity to enhance the cybersecurity skills of our students.

Sincerely,

Suranjana Banerjee

[Student of BCA 6Th Sem]

System Used in this project:

OS Used: Windows 11 Home

System RAM: 8 GB DDR4 RAM.

System Processor: INTEL i3 10th generation processor.

System Disk space: 121 GB available disk space.

Software Used in this project

Spyder: Spyder is a free and open-source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.



Google Chrome: Google Chrome browser is a free web browser used for accessing the internet and running web-based applications. The Google Chrome browser is based on the open source Chromium web browser project. Google released Chrome in 2008 and issues several updates a year. Google Chrome is available for Microsoft Windows, Apple macOS and Linux desktop operating systems (OSes), as well as the Android and iOS mobile operating systems.

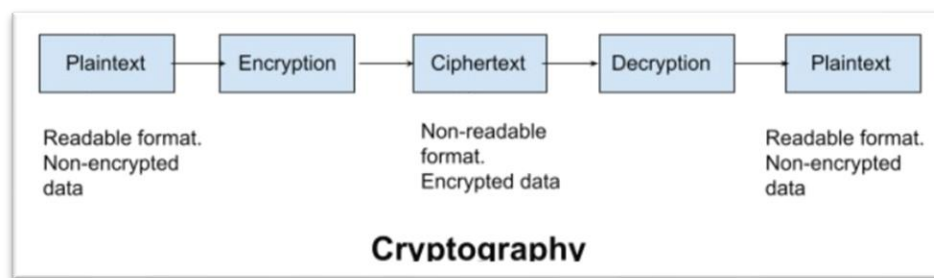


Index

Sl No.	Topics	Page No.
1.	Cryptography Introduction <ul style="list-style-type: none"> • History of cryptography 11-12 • Some terminology related to cryptography (Encryption, key, Decryption, Cipher) 13 • Applications of Cryptography 14 • Features of Cryptography 15 • GAK 15 • Cipher <ul style="list-style-type: none"> — Classical Ciphers 17-21 <ul style="list-style-type: none"> — Substitution Cipher <ul style="list-style-type: none"> — Monoalphabetic Cipher (Ceaser Cipher) 18-20 — Polyalphabetic Cipher (Vigenère Cipher) 20-21 — Transpositional Cipher 22-23 — Modern Cipher 24-31 <ul style="list-style-type: none"> — Symmetric Cipher <ul style="list-style-type: none"> — Stream Cipher (RC4) 25-28 — Block Cipher (DES) 29-31 — Asymmetric Cipher (RSA) 32-35 — Online Encryption Tools 36 	
2.	Hashing Introduction <ul style="list-style-type: none"> • MD5 39-40 • SHA1 41-42 	
3.	Conclusion (Hashing VS Encryption)	43
4.	Limitations	44
5.	Bibliography	45

Cryptography

Cryptography is technique of securing information and communications through use of codes so that only those persons for whom the information is intended can understand it and process it. Thus, preventing unauthorized access to information. The prefix "Crypto" indicates "hidden," and "graphy" indicates "writing," respectively. In Cryptography the techniques which are used to protect information are obtained from mathematical concepts and a set of rule-based calculations known as algorithms to convert messages in ways that make it hard to decode it. These algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on internet and to protect confidential transactions such as credit card and debit card transactions.



➤ How Does Cryptography Work:

Cryptographic algorithms are central to how cryptography works. Cryptographic algorithms, also called ciphers, are mathematical functions that encrypt text by combining them with keys such as phrases, digits, words, and so on. You can define the effectiveness by the strength of the cryptographic algorithms and the level of key secrecy.

➤ Examples of Cryptography:

- i. **End-to-end encryption in WhatsApp** is a prominent example of cryptography encryption these days. This feature is available in WhatsApp via the asymmetry model or public key methods. Only the intended recipient is aware of the actual message. After the WhatsApp installation, the server registers the public keys, and messages are transmitted.
- ii. **Digital signatures** are the next real-time application of cryptography. When two clients must sign documents for a business transaction. However, if two clients never meet, they may not believe each other. The use of encryption in digital signatures then ensures improved authentication and security.

History of Cryptography

The art of cryptography is considered to be born along with the art of writing. As civilizations evolved, human beings got organized in tribes, groups, and kingdoms. This led to the emergence of ideas such as power, battles, supremacy, and politics. These ideas further fuelled the natural need of people to communicate secretly with selective recipient which in turn ensured the continuous evolution of cryptography as well.

A. Ancient Origins: Cryptography traces back to ancient civilizations like Egypt and Mesopotamia, where hieroglyphs and pictograms were used to convey hidden messages. The roots of cryptography are found in Roman and Egyptian civilizations.

Hieroglyph – The Oldest Cryptographic Technique

The first known evidence of cryptography can be traced to the use of 'hieroglyph'. Some 4000 years ago, the Egyptians used to communicate by messages written in hieroglyph. This code was the secret known only to the scribes who used to transmit messages on behalf of the kings. One such hieroglyph is shown below.

Later, the scholars moved on to using simple mono-alphabetic substitution ciphers during 500 to 600 BC. This involved replacing alphabets of message with other alphabets with some secret rule. This rule became a key to retrieve the message back from the garbled message.



The earliest known use of cryptography is found in non-standard hieroglyphs carved into the wall of a tomb from the Old Kingdom of Egypt circa 1900 BC.

B. Classical Era: The Greeks employed simple ciphers, and Julius Caesar famously used a substitution cipher (Caesar cipher) to secure military communications.

Another Greek method was developed by Polybius, it's now called the "Polybius Square". Polybius, a Greek philosopher around 180 BC, revolutionized government thinking by advocating for the separation of powers and government serving the people. His ideas influenced later philosophers, including Montesquieu and the U.S. Constitution. Due to his family's political opposition, Polybius sought secrecy in his writing. His work, "The Histories," detailed Roman history from 264 to 146 BC. He devised the Polybius square, a simple cipher system, to conceal his writings on government separation and abuses of power. While not foolproof, this method marked a significant advancement in cryptography, leading to various other uses. Cryptography, from ancient times to modern secure communication, remains a fundamental aspect of safeguarding information.

The idea about "**Polybius square**" was simple. First, create a checkerboard with numbers running across the top and along the left side. Next, populate the interior with the letters of the alphabet. Then, when writing, a letter would become its coordinates on the grid; for example, A might be written as 11, while B would be 12.

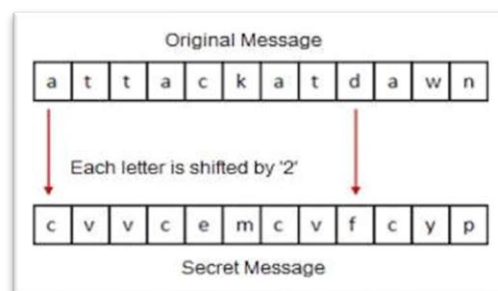
	1	2	3	4	5	6
1	A	B	C	D	E	F
2	G	H	I	J	K	L
3	M	N	O	P	Q	R
4	S	T	U	V	W	X
5	Y	Z	0	1	2	3
6	4	5	6	7	8	9



The first recorded use of cryptography for correspondence was by the Spartans, who as early as 400 BC employed a **cipher device** called the **scytale** for secret communication between military commanders. The scytale consisted of a tapered baton, around which was spirally wrapped a strip of parchment or leather on which the message was then written. When unwrapped, the letters were scrambled in order and formed the cipher; however, when the strip was wrapped around another baton of identical proportions to the original, the plaintext reappeared.

The Romans used monoalphabetic substitution with a simple cyclic displacement of the alphabet. The earlier Roman method of cryptography, popularly known as the "Caesar Shift Cipher", relies on shifting the letters of a message by an agreed number

(three was a common choice), the recipient of this message would then shift the letters back by the same number and obtain the original message. Julius Caesar employed a shift of three positions so that plaintext A was encrypted as D, while Augustus Caesar used a shift of one position so that plaintext A was enciphered as B. As many moviegoers noticed, HAL, the computer in 2001: A Space Odyssey (1968), encrypts to IBM using Augustus's cipher.



C. Middle Ages: European cryptology developed during this period, particularly in the Papal States and Italian city-states. Manuals on cryptography started emerging during this time.



The first European manual on cryptography (c. 1379) was a compilation of ciphers by Gabriele de Lavinde of Parma, who served Pope Clement VII. This manual, now in the Vatican archives, contains a set of keys for 24 correspondents and embraces symbols for letters, nulls, and several two-character code equivalents for words and names. The first brief code vocabularies, called nomenclators, were gradually expanded and became the mainstay well into the 20th century for diplomatic communications of nearly all European governments. In 1470 Leon Battista Alberti published *Trattati in cifra* ("Treatise on Ciphers"), in which he described **the first cipher disk**; he prescribed that the setting of the disk should be changed after enciphering three or four words, thus conceiving of the notion of **polyalphabetic**. This same device was used almost five centuries later by the U.S. Army Signal Corps for tactical communications in World War I.

D. Modern Cryptography: The 20th century witnessed significant advancements, including the development of complex cryptographic systems such as the Enigma machine during World War II.

By the end of the World War-I some complicated cipher systems were used for high-level communications, the most famous of which was the German ADFGVX fractionation cipher. The communications need of telegraphy and radio and the maturing of mechanical and electromechanical technology came together in the 1920s to bring about a major advance in crypto devices: the development of rotor cipher machines. Although the concept of a rotor had been anticipated in the older mechanical cipher disks, American Edward H. Hebern recognized in about 1917 (and made the first patent claim) that by hardwiring a monoalphabetic substitution in the connections from contacts on one side of an electrical disk (rotor) to contacts on the other side and then cascading a collection of such rotors, polyalphabetic substitutions of almost arbitrary complexity could be realized. Hebern was generally recognized as the inventor of the rotor encryption machine. At almost the same time that Hebern was developing the rotor cipher machine in the United States, European engineers, notably Hugo A. Koch of the Netherlands and Arthur Scherbius of Germany, independently discovered the rotor concept and designed machines that became the precursors of the best-known cipher machine in history, the German Enigma used in World War II.



E. Contemporary Era: With the rise of computers, cryptography evolved rapidly. Public-key cryptography, introduced in the 1970s, revolutionized the field, leading to secure digital communication and e-commerce.

It operates on binary bit sequences. Modern cryptography requires parties interested in secure communication to possess the secret key only. It relies on publicly known mathematical algorithms for coding the information. Secrecy is obtained through a secret key which is used as the seed for the algorithms. The computational difficulty of algorithms, absence of secret key, etc., make it impossible for an attacker to obtain the original information even if he knows the algorithm used for coding.

Some Terminology Related to Cryptography

A. Encryption:

Encryption is a way of scrambling data so that only authorized parties can understand the information. In technical terms, it is the process of converting human-readable plaintext to incomprehensible text, also known as ciphertext. In simpler terms, encryption takes readable data and alters it so that it appears random. Encryption requires the use of a cryptographic key, which is a set of mathematical values that both the sender and the recipient of an encrypted message agree on. Cryptography is the field of encrypting and decrypting information. Unencrypted data is referred to as plaintext in computing, whereas encrypted data is referred to as ciphertext. Encryption algorithms, often known as ciphers, are formulae that are used to encode and decode communications.

B. Key:

A cryptographic key is the core part of cryptographic operations. In cryptography, a key is a string of characters used within an encryption algorithm for altering data so that it appears random. Like a physical key, it locks (encrypts) data so that only someone with the right key can unlock (decrypt) it. The length of the key is a factor in considering how difficult it will be to decrypt the text in a given message.

C. Decryption:

Decryption is the procedure of changing encrypted information into its original, decipherable format. The phase of decryption takes the ambiguous information that was originally received and interprets it into words and images that a human can understand. Decryption is an important component of cybersecurity processes, because encryption needed scrambling words and pictures to securely send them to a multiple user through the internet.

D. Cipher:

In cryptology, the discipline concerned with the study of cryptographic algorithms, a cipher is an algorithm for encrypting and decrypting data. The cipher analyses the original and plaintext data to generate ciphertext that seems to be random data. Ciphers are also called stream ciphers since they can encrypt or decrypt bits in a stream. They can also use block ciphers, which break up ciphertext into uniform units of a predefined number of bits. Depending on the algorithm and use case, professionals recommend that current ciphers be designed with at least 128 bits or more, even if the key length is not necessarily connected with cipher strength.

Applications of Cryptography

Cryptography is used in many real-life applications to secure communication and information. It is used in mobile messaging tools such as WhatsApp, digital signatures, and HTTPS, as well as in banking transactions, emailing, securing houses, and more.

- **Computer passwords:** Cryptography is widely utilized in computer security, particularly when creating and maintaining passwords. When a user logs in, their password is hashed and compared to the hash that was previously stored. Passwords are hashed and encrypted before being stored. In this technique, the passwords are encrypted so that even if a hacker gains access to the password database, they cannot read the passwords.
- **Digital Currencies:** To safeguard transactions and prevent fraud, digital currencies like Bitcoin also use cryptography. Complex algorithms and cryptographic keys are used to safeguard transactions, making it nearly hard to tamper with or forge the transactions.
- **Secure web browsing:** Online browsing security is provided by the use of cryptography, which shields users from eavesdropping and man-in-the-middle assaults. Public key cryptography is used by the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols to encrypt data sent between the web server and the client, establishing a secure channel for communication.
- **Electronic signatures:** Electronic signatures serve as the digital equivalent of a handwritten signature and are used to sign documents. Digital signatures are created using cryptography and can be validated using public key cryptography. In many nations, electronic signatures are enforceable by law, and their use is expanding quickly.
- **Authentication:** Cryptography is used for authentication in many different situations, such as when accessing a bank account, logging into a computer, or using a secure network. Cryptographic methods are employed by authentication protocols to confirm the user's identity and confirm that they have the required access rights to the resource.
- **Cryptocurrencies:** Cryptography is heavily used by cryptocurrencies like Bitcoin and Ethereum to safeguard transactions, thwart fraud, and maintain the network's integrity. Complex algorithms and cryptographic keys are used to safeguard transactions, making it nearly hard to tamper with or forge the transactions.
- **End-to-End Encryption:** End-to-end encryption is used to protect two-way communications like video conversations, instant messages, and email. Even if the message is encrypted, it assures that only the intended receivers can read the message. End-to-end encryption is widely used in communication apps like WhatsApp and Signal, and it provides a high level of security and privacy for users.
- **Online Banking:** Many online banking systems use cryptography to secure sensitive financial transactions and protect customers' personal and financial information.
- **Email:** Many email services use encryption to protect the privacy and confidentiality of emails in transit. For example, services like Gmail use Transport Layer Security (TLS) to encrypt emails.
- **Mobile Devices:** Mobile devices, such as smartphones and tablets, often use cryptography to secure data stored on the device and to protect communications. For example, Apple's iOS uses a hardware encryption system to secure data on iPhones and iPads.
- **Cloud Storage:** Cryptography is used to secure data stored in the cloud. For example, Amazon Web Services uses the AES encryption algorithm to secure data stored in its Simple Storage Service (S3) and the Amazon Elastic Block Store (EBS).

Features of Cryptography

These are the features of cryptography:

- **Confidentiality:** Hides the contents of a message from unauthorized parties.
- **Integrity:** Ensures that a message has not been altered during transmission.
- **Authentication:** Verifies the identity of the sender and receiver of a message.
- **Non-repudiation:** Prevents the sender from denying having sent a message.
- **Availability:** Ensures that authorized users have access to the information they need when they need it.
- **Key Management:** The process of generating, distributing, storing, and replacing cryptographic keys.
- **Algorithm:** The mathematical formula used to encrypt and decrypt messages.
- **Encryption/Decryption:** The process of converting plaintext to ciphertext and vice versa.
- **Digital Signatures:** A signature that can be used to authenticate the identity of the sender of a message and ensure the integrity of the message.

Government Access to Keys (GAK)

Government Access Keys, commonly referred to simply as GAK, is the act of giving government full or partial access to your encryption and decryption keys. The government needs to gain access in certain circumstances to any encrypted information to carry out any legal purpose authorized by law. There may be many such circumstances where the government would require accessing the encrypted communications. For example: detecting any conspiracy to commit a serious crime, investigating a crime, etc. This is what's known as a type of key escrow. This can also give access to keys in the event that a company has shut down or no longer exists. So, think of a company that is using a key to encrypt information on someone's computer. In the event that that company goes out of existence, the government may have need to access data that has been encrypted by that software. If the company is no longer there, the government has no recourse through use of a court warrant to receive that key.



Ciphers

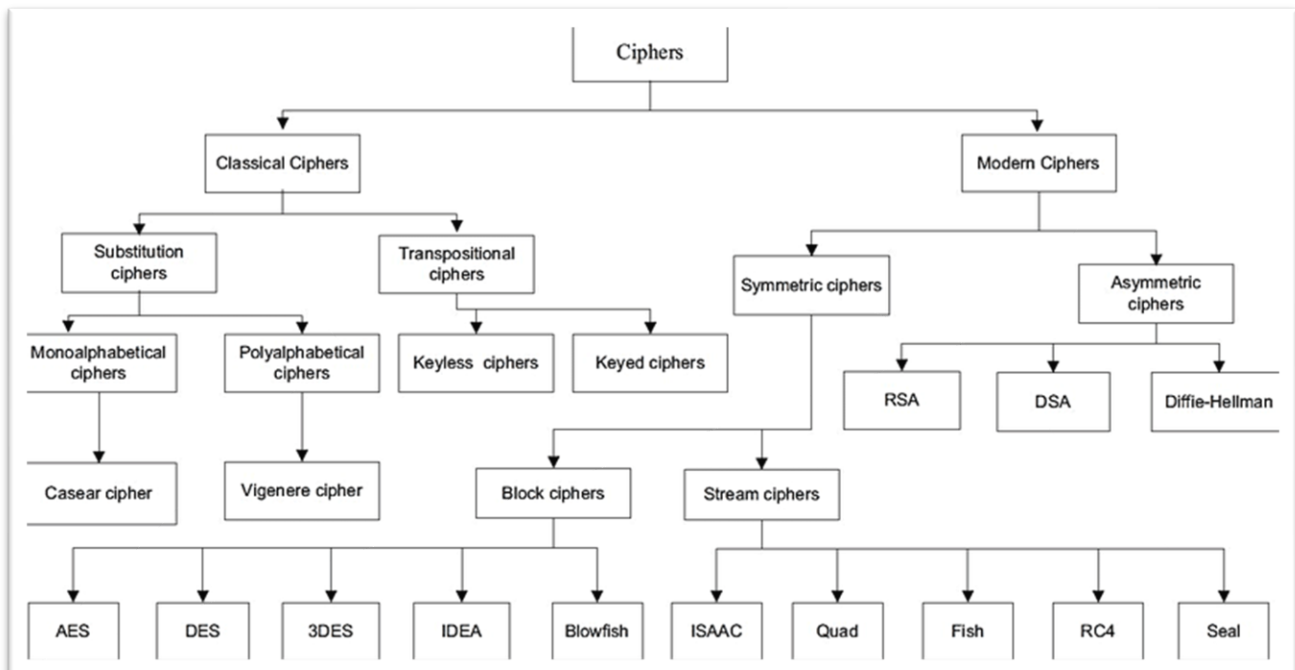
What is cipher?

Cipher is a frequently used algorithm in cryptology, a subject concerned with the study of cryptographic algorithms. It is a method of encrypting and decrypting data.

How Does a Cipher Work?

- Ciphers use an encryption method to change plaintext, a readable communication, into ciphertext, which appears to be a random string of letters.
- Ciphers are also called stream ciphers since they can encrypt or decrypt bits in a stream.
- They can also use block ciphers, which break up ciphertext into uniform units of a predefined number of bits.
- Modern cipher implementations change data as it is encrypted using the encryption method and a secret key.
- Ciphers more resist brute-force attacks when their keys are longer (measured in bits).
- Depending on the algorithm and use case, professionals recommend that current ciphers be designed with at least 128 bits or more, even if the key length is not necessarily connected with cipher strength.
- Because a key is such a vital component of an encryption process, the key is maintained as a secret rather than a procedure in real-world ciphering.
- Even if someone is familiar with the procedure, a robust encryption mechanism should make reading the ciphertext with the necessary key easier.
- As a result, a cipher requires the possession of a key or set of keys by both the sender and the recipient to work.

Types of Cipher:



Classical Ciphers

What are Classical Ciphers?

In cryptography, a classical cipher, also known as Hand Ciphers and Pen & Paper Ciphers, is a type of cipher that was used historically but for the most part, has fallen into disuse.

There are two types of Classical Ciphers:

A. Simple Substitution Cipher: In a Substitution cipher, any character of plain text from the given fixed set of characters is substituted by some other character from the same set depending on a key. For example, with a shift of 1, A would be replaced by B, B would become C, and so on.

Algorithm for Substitution Cipher:

Input:

- A String of both lower-case and upper-case letters, called Plaintext.
- An Integer denoting the required key.

Procedure:

- Create a list of all the characters.
- Copy the string, then shuffle it. It will be the key value.
- For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
- Print the new string generated.

Implementation of Simple Substitution Cipher in Python

```
1  import random
2  import string
3  char = string.punctuation + string.ascii_letters + string.digits + " "
4  char = list(char)
5  key = char.copy()
6  random.shuffle(key)
7
8  #print(f"char: {char}")
9  #print(f"key: {key}")
10
11 #Encryption
12 def encryption(plain_text, shift_key):
13     cipher_text = ""
14     for letter in plain_text:
15         index = char.index(letter)
16         cipher_text += key[index]
17
18     print(f" Original Message: {plain_text}")
19     print(f" Encrypted Message: {cipher_text}")
20
21 #Decryption
22 def decryption(cipher_text, shift_key):
23     plain_text = ""
24     for letter in cipher_text:
25         index = key.index(letter)
26         plain_text += char[index]
```

```

28     print(f" Encrypted Message: {cipher_text}")
29     print(f" Original Message: {plain_text}")
30
31     flag=False
32     while not flag:
33         user_choice= input('Type "e" for ENCRYPTION or "d" for DECRYPTION: ')
34         message= input(' Enter your Message: ')
35
36         if user_choice == 'e':
37             encryption(plain_text= message, shift_key= key)
38         elif user_choice == 'd':
39             decryption(cipher_text= message, shift_key= key)
40
41         choice= input('Type "yes" to continue or "no" to end: ')
42         if choice == 'no':
43             flag=True
44             print ('Have a Nice Day! Bye...!')

```

Output:

```

Type "e" for ENCRYPTION or "d" for DECRYPTION: e

Enter your Message: Hello World
Original Message: Hello World
Encrypted Message: O/;;*2V*9;?

Type "yes" to continue or "no" to end: yes

Type "e" for ENCRYPTION or "d" for DECRYPTION: d

Enter your Message: O/;;*2V*9;?
Encrypted Message: O/;;*2V*9;?
Original Message: Hello World

Type "yes" to continue or "no" to end: no
Have a Nice Day! Bye...!

```

There are two types of Substitution Cipher:

- **Monoalphabetic cipher:** The Greek root "**mono**" meaning "**one**". A monoalphabetic cipher is a type of cipher where each character of the plaintext is mapped to a corresponding character in the cipher text using a single alphabetic key. An example of monoalphabetic cipher is **Caesar Cipher**.

Caesar Cipher: The Caesar cipher is a simple encryption technique that was used by **Julius Caesar** to send secret messages to his allies. It works by shifting the letters in the plaintext message by a certain number of positions, known as the "shift" or "key". They mainly use '3' as the shift key.



Advantages of Caesar cipher

- It is very easy to implement.
- This method is the simplest method of cryptography.
- Only one short key is used in its entire process.

- If a system does not use complex coding techniques, it is the best method for it.
- It requires only a few computing resources.

Disadvantages of Caesar cipher

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

The formula of encryption is: Encryption = (index of the letter + key) mod 26

The formula of decryption is: Decryption = (index of the letter - key) mod 26

Implementation of Caesar Cipher in Python

```

1  alphabate= ['a','b','c','d','e','f','g','h','i','j','k','l','m',
2             'n','o','p','q','r','s','t','u','v','w','x','y','z']
3
4  def encryption(plaintext, shift_key): # hello
5      ciphertext=''
6      for char in plaintext:
7          if char in alphabate:
8              position=alphabate.index(char) # position=7
9              new_position=(position+shift_key)%26 # new_position= 7+3=10
10             ciphertext +=alphabate[new_position] # ciphertext= k
11         else:
12             ciphertext += char
13     print(f'The Encrypted message is:{ciphertext}')
14 def decryption(ciphertext, shift_key): # kloor
15     plaintext=''
16     for char in ciphertext:
17         if char in alphabate:
18             position=alphabate.index(char)
19             new_position=(position-shift_key)%26
20             plaintext +=alphabate[new_position]
21         else:
22             plaintext += char
23     print(f'The Decrypted message is:{plaintext}')
24
25     flag=False
26     while not flag:
27         user_choice= input('Type "e" for ENCRYPTION or "d" for DECRYPTION: ')
28         message= input('Type your Message:')
29         key= int(input('Enter the shift key:'))
30
31         if user_choice == 'e':
32             encryption(plaintext= message, shift_key= key)
33         elif user_choice == 'd':
34             decryption(ciphertext= message, shift_key= key)
35
36         choice= input('Type "yes" to continue or "no" to end: ')
37         if choice == 'no':
38             flag=True
39         print ('Have a Nice Day! Bye...!')

```

Output:

```
Type "e" for ENCRYPTION or "d" for DECRYPTION: e
Type your Message:hello world
Enter the shift key:3
The Encrypted message is:khoor zruog
Type "yes" to continue or "no" to end: yes
Type "e" for ENCRYPTION or "d" for DECRYPTION: d
Type your Message:khoor zruog
Enter the shift key:3
The Decrypted message is:hello world
Type "yes" to continue or "no" to end: no
Have a Nice Day! Bye...!
```

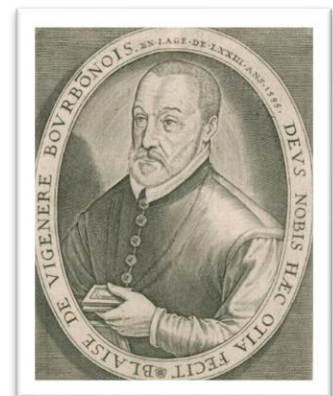
- **Polyalphabetic Cipher:** "poly" is the Greek root for "many". A polyalphabetic cipher, uses multiple substitution alphabets for encoding the plaintext. One of the well-known examples of a polyalphabetic cipher is the **Vigenère cipher**.

Vigenère cipher: Vigenère Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the **Vigenère square or Vigenère table**.

It was possibly first described in 1553 by Italian cryptographer Giovan Battista Bellaso, though it has been reinvented many times, moulding

by **Blaise de Vigenère**. It is thought to have remained unbroken

until Charles Babbage, considered to be the father of computers, broke it in the 19th century. It is called "le chiffre indéchiffrable" French for "the indecipherable cipher".



Plaintext Letter	
	ABCDEFGHIJKLMN OPQRSTUVWXYZ
A	ABCDEFGHIJKLMN OPQRSTUVWXYZ
B	BCDEFGHIJKLMN OPQRSTUVWXYZA
C	CDEFGHIJKLMN OPQRSTUVWXYZAB
D	DEFGHIJKLMN OPQRSTUVWXYZABC
E	EFGHIJKLMN OPQRSTUVWXYZABCD
F	FGHIJKLMN OPQRSTUVWXYZABCDE
G	GHIJKLMN OPQRSTUVWXYZABCDEF
H	HIKLMNOPQRSTUVWXYZABCDEFGHI
I	IJKLMN OPQRSTUVWXYZABCDEFGHIJ
J	JKLMN OPQRSTUVWXYZABCDEFGHIJK
K	KLMN OPQRSTUVWXYZABCDEFGHIJKL
L	LMN OPQRSTUVWXYZABCDEFGHIJKLM
M	MN OPQRSTUVWXYZABCDEFGHIJKLMN
N	N OPQRSTUVWXYZABCDEFGHIJKLMNO
O	OPQRSTUVWXYZABCDEFGHIJKLMNO P
P	PQRSTUVWXYZABCDEFGHIJKLMNO PQ
Q	QRSTUVWXYZABCDEFGHIJKLMNO PQR
R	RSTUVWXYZABCDEFGHIJKLMNO PQRS
S	STUVWXYZABCDEFGHIJKLMNO PQRS T
T	TUVWXYZABCDEFGHIJKLMNO PQRS TU
U	UVWXYZABCDEFGHIJKLMNO PQRS TU V
V	VWXYZABCDEFGHIJKLMNO PQRS TUV
W	WXYZABCDEFGHIJKLMNO PQRS TUV W
X	XYZABCDEFGHIJKLMNO PQRS TUVW X
Y	YZABCDEFGHIJKLMNO PQRS TUVWX
Z	ZABCDEFGHIJKLMNO PQRS TUVWXY

Vigenère square or Vigenère table:

The Vigenère square or Vigenère table, also known as the tabula recta, can be used for encryption and decryption.

Implementation Vigenère Cipher in Python

```
alphabate= 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'

#Encryption
def encryption(plaintext, shift_key):
    ciphertext=''
    temp=0
    for char in plaintext:
        if char in alphabate:
            position=alphabate.find(char) # index value of letters of plaintext
            if position!=-1:
                position= position+alphabate.find(shift_key[temp]) #index value of key added to plaintext
                new_position= position%len(alphabate) #cross checking the boundary
                ciphertext+= alphabate[new_position]
                temp+=1
                if temp==len(shift_key):
                    temp=0
            else:
                ciphertext+= char
    print(f'The Encrypted message is:{ciphertext}')

#Decryption
def decryption(ciphertext, shift_key):
    plaintext=''
    temp=0
    for char in ciphertext:
        if char in alphabate:
            position=alphabate.find(char) # index value of letters of plaintext
            if position!=-1:
                position= position-alphabate.find(shift_key[temp]) #index value of key added to plaintext
                new_position= position%len(alphabate) #cross checking the boundary
                plaintext+= alphabate[new_position]
                temp+=1
                if temp==len(shift_key):
                    temp=0
            else:
                plaintext+= char
    print(f'The Decrypted message is:{plaintext}')

flag=False
while not flag:
    user_choice= (input('Type "e" for ENCRYPTION or "d" for DECRYPTION: ').lower())
    message= input('Type your Message:')
    key= input('Enter the key:')

    if user_choice == 'e':
        encryption(plaintext= message, shift_key= key)
    elif user_choice == 'd':
        decryption(ciphertext= message, shift_key= key)

    choice= (input('Type "yes" to continue or "no" to end: ').upper())
    if choice == 'no':
        flag=True
    print ('Have a Nice Day! Bye...!')
```

Output:

```
Type "e" for ENCRYPTION or "d" for DECRYPTION: e
Type your Message:hello world
Enter the key:5
The Encrypted message is:gdkkn vnqkc
Type "yes" to continue or "no" to end: yes
Type "e" for ENCRYPTION or "d" for DECRYPTION: d
Type your Message:gdkkn vnqkc
Enter the key:5
The Decrypted message is:hello world
Type "yes" to continue or "no" to end: no
Have a Nice Day! Bye...!
```

B. Transpositional cipher: Transposition Cipher is a cryptographic algorithm where the order of alphabets in the plaintext is rearranged to form a cipher text. In this process, the actual plain text alphabets are not included.

Example: A simple example for a transposition cipher is columnar transposition cipher where each character in the plain text is written horizontally with specified alphabet width. The cipher is written vertically, which creates an entirely different cipher text.

Consider the plain text ‘**hello world**’, and let us apply the simple columnar transposition technique as shown below

The plain text characters are placed horizontally and the cipher text is created with vertical format as ‘**holewdlo lr**’. Now, the receiver has to use the same table to decrypt the cipher text to plain text.

h	e	l	l
o	w	o	r
l	d		

Here’s the algorithm for a basic columnar transposition cipher:

Choose a keyword: This keyword determines the number of columns and the order in which they are read.

Write the plaintext in rows: Arrange the plaintext message in a grid with a number of columns equal to the length of the keyword.

Fill the grid row-wise: Fill the grid with the plaintext characters row by row.

Number the columns: Number the columns based on the alphabetical order of the keyword.

Read the columns in order: Read the characters column by column in the order determined by the keyword’s alphabetical order to create the ciphertext.

Implementation of Transpositional Cipher

```
1  import math
2
3  #Encryption
4  def encryption(plaintext, shift_key):
5      t = 0
6      for r in range(row):
7          for c, ch in enumerate(message[t : t+ len_key]):
8              matrix[r][c] = ch
9              t += len_key
10
11     # print(matrix)
12     sort_order = sorted([(ch,i) for i,ch in enumerate(key)]) #to make alphabetically order o
13     # print(sort_order)
14
15     ciphertext = ''
16     for ch,c in sort_order:
17         for r in range(row):
18             ciphertext += matrix[r][c]
19     print(f'The Encrypted message is:{ciphertext}')
20
```

```

21 #Decryption
22 def decryption(ciphertext, shift_key):
23     matrix_new = [ ['X']*len_key for i in range(row) ]
24     key_order = [ key.index(ch) for ch in sorted(list(key))] #to make original key order when
25     # print(key_order)
26
27     t = 0
28     for c in key_order:
29         for r,ch in enumerate(message[t : t+ row]):
30             matrix_new[r][c] = ch
31             t += row
32     # print(matrix_new)
33
34     plaintext = ''
35     for r in range(row):
36         for c in range(len_key):
37             plaintext += matrix_new[r][c] if matrix_new[r][c] != 'X' else ''
38     print(f'The Decrypted message is:{plaintext}')
39
40 flag=False
41 while not flag:
42     user_choice= (input('Type "e" for ENCRYPTION or "d" for DECRYPTION: ').lower())
43     message= input('Type your Message:')
44     key= input('Enter the key:')
45
46     len_key = len(key)
47     len_plain = len(message)
48     row = int(math.ceil(len_plain / len_key))
49     matrix = [ ['X']*len_key for i in range(row) ]
50     # print (matrix)
51
52     if user_choice == 'e':
53         encryption(plaintext= message, shift_key= key)
54     elif user_choice == 'd':
55         decryption(ciphertext= message, shift_key= key)
56
57     choice= (input('Type "yes" to continue or "no" to end: ').lower())
58     if choice == 'no':
59         flag=True
60     print ('Have a Nice Day! Bye...!')

```

Output:

```

Type "e" for ENCRYPTION or "d" for DECRYPTION: e

Type your Message:hello world

Enter the key:hi
The Encrypted message is:hlowrdel olX

Type "yes" to continue or "no" to end: yes

Type "e" for ENCRYPTION or "d" for DECRYPTION: d

Type your Message:hlowrdel olX

Enter the key:hi
The Decrypted message is:hello world

```

Modern Ciphers

What are Modern Ciphers?

Modern ciphers enable private communication in many different networking protocols, including the Transport Layer Security (TLS) protocol and others that offer encryption of network traffic. Many communication technologies, including phones, digital television and ATMs, rely on ciphers to maintain security and privacy.

Although hybrid systems do exist (such as the SSL internet protocols), most encryption techniques fall into one of three main categories: symmetric cryptography algorithms, asymmetric cryptography algorithms, or hash functions.

There are two types of Modern Ciphers:

- A. Symmetric Cipher:** Symmetric Encryption is the most basic and old method of encryption. It uses only one key for the process of both the encryption and decryption of data. There is a need for a very strong encryption algorithm that produces cipher texts in such a way that the attacker should be unable to crack the secret key even if they have access to one or more cipher texts. There must be a secure and robust way to share the secret key between the sender and the receiver. It should be leakproof so that the attacker cannot access the secret key.

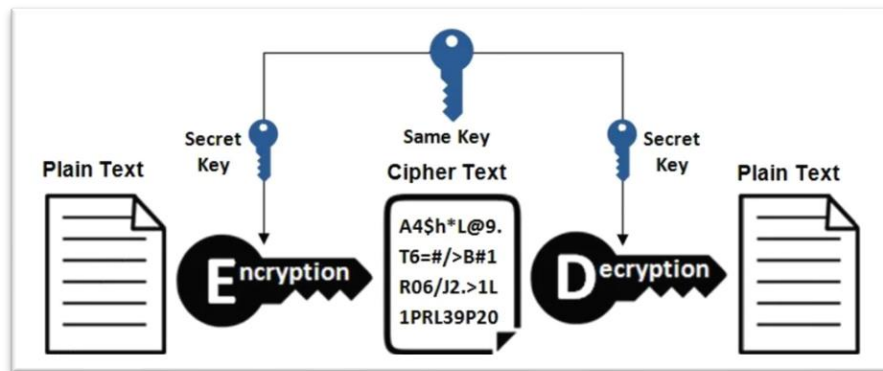
There are two types of symmetric ciphers: **Block cipher:** AES (Advanced Encryption Standard), DES (Data Encryption Standard), 3DES (Triple Data Encryption Standard), **Blowfish**, and **Stream cipher:** RC4 (Rivest Cipher 4).

Advantages of Symmetric Key Cryptography:

- **Speed:** Encryption and decryption are fast and efficient, making it suitable for large amounts of data.
- **Simplicity:** The single shared key makes it easier to implement and use compared to asymmetric key cryptography.

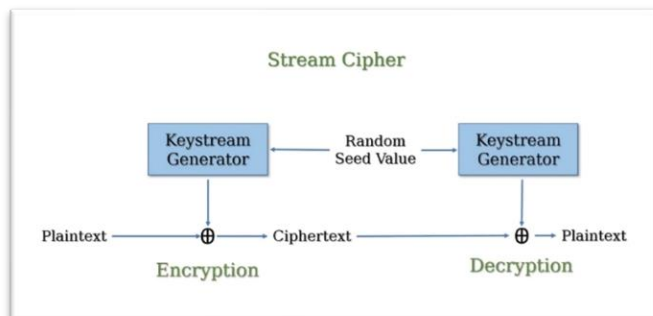
Disadvantages of Symmetric Key Cryptography:

- **Key Management:** The secure distribution of the shared key between the sender and receiver can be a challenge.
- **Scalability:** With a large number of users, the number of keys required can quickly become unmanageable.
- **Security:** If the shared key is compromised, the confidentiality and integrity of the data can be threatened.



There are two types of Symmetric Cipher:

- **Stream Cipher:** A stream cipher is an encryption technique that **works byte by byte** to transform plain text into code that's unreadable to anyone without the proper key. A Stream cipher refers to a symmetric encryption algorithm that encrypts data by combining it with a stream of pseudorandom bits, known as a keystream. Stream ciphers are linear, so the same key both encrypts and decrypts messages. stream ciphers work on each bit of data in a message rather than chunking the message into groups and encrypting them in blocks. **RC4 is the most common type of stream cipher design.** The transformation of encrypted output varies during the encryption cycle.



Where are the Stream Ciphers used?

Stream ciphers get used in applications where high-speed encryption is required, such as in wireless communication or real-time data transmission.

Here's how the stream cipher algorithm works:

Key Generation: The algorithm generates a keystream based on a secret key and an initialization vector (IV).

Keystream Generation: Using the key and IV, the algorithm produces a sequence of pseudorandom bits or bytes, which forms the keystream.

Encryption: Each bit or byte of the plaintext is combined (usually through a bitwise XOR operation) with the corresponding bit or byte of the keystream to produce the ciphertext.

Decryption: To decrypt the ciphertext, the same keystream is generated using the key and IV. Then, the ciphertext is XORed with the keystream to recover the original plaintext.

Synchronization: Both the encryption and decryption processes must remain synchronized, ensuring that the correct bits of the keystream are used for encryption and decryption.

RC4: RC4 stands for Rivest Cipher 4 invented by Ron Rivest in 1987 for RSA Security. It is a Stream Ciphers. RC4 stream cipher is one of the most widely used stream ciphers because of its simplicity and speed of operation. It is a variable key-size stream cipher with byte-oriented operations. It uses either 64 bit or 128-bit key sizes. It is generally used in applications such as Secure Socket Layer (SSL), Transport Layer Security (TLS), and also used in IEEE 802.11 wireless LAN std.

History of RC4 Encryption

RC4 was designed by Ron Rivest in 1987. He was working under RSA Security. Rivest Cipher 4 is an official name while it is also known as Ron's Code. Initially, RC4 was trade secret but once its code spread in the public domain it was no more a trade secret. While Ron did not reveal the RC4 algorithm until 2014 when he described the history of RC4 in English Wikipedia.

Applications of RC4

RC4 is used in various applications such as WEP from 1997 and WPA from 2003. We also find applications of RC4 in SSL from 1995 and it is a successor of TLS from 1999. RC4 is used in varied applications because of its simplicity, speed, and simplified implementation in both software and hardware.

RC4 Encryption Procedure

- The user inputs a plain text file and a secret key.
- The encryption engine then generates the keystream by using KSA and PRGA Algorithm.
- This keystream is now XOR with the plain text, this XORing is done byte by byte to produce the encrypted text.
- The encrypted text is then sent to the intended receiver, the intended receiver will then decrypt the text and after decryption, the receiver will get the original plain text.
- Decryption is achieved by doing the same byte-wise X-OR operation on the Ciphertext.

Example: Let A be the plain text and B be the keystream $(A \text{ XOR } B) \text{ XOR } B = A$

Implementation of RC4 Algorithm in python

```
1  #key scheduling algorithm
2  def key_scheduling(key):
3      schedule = [i for i in range(0, 256)]
4      num = 0
5      for i in range(0, 256):
6          num = (num + schedule[i] + key[i % len(key)]) % 256
7          temp = schedule[i]
8          schedule[i] = schedule[num]
9          schedule[num] = temp
10     return schedule
11
```

```

11
12 #Pseudo-Random Generation Algorithm
13 def stream_generation(schedule):
14     i = 0
15     j = 0
16     while True:
17         i = (1 + i) % 256
18         j = (schedule[i] + j) % 256
19         temp = schedule[j]
20         schedule[j] = schedule[i]
21         schedule[i] = temp
22         yield schedule[(schedule[i] + schedule[j]) % 256]
23
24 #Encryption
25 def encryption(plaintext, key):
26     plaintext = [ord(char) for char in plaintext]
27     key = [ord(char) for char in key]
28     schedule = key_scheduling(key)
29     key_stream = stream_generation(schedule)
30     ciphertext = ''
31     for char in plaintext:
32         encrypt = str(hex(char ^ next(key_stream))).upper()
33         ciphertext += encrypt
34     print(f'The Encrypted message is:{ciphertext}')
35
36 #Decryption
37 def decryption(ciphertext, key):
38     ciphertext = ciphertext.split('0X')[1:]
39     ciphertext = [int('0x' + c.lower(), 0) for c in ciphertext]
40     key = [ord(char) for char in key]
41     schedule = key_scheduling(key)
42     key_stream = stream_generation(schedule)
43     plaintext = ''
44     for char in ciphertext:
45         decrypt = str(chr(char ^ next(key_stream)))
46         plaintext += decrypt
47     print(f'The Decrypted message is:{plaintext}')
48

```

```

49
50 flag=False
51 while not flag:
52     user_choice= (input('Type "e" for ENCRYPTION or "d" for DECRYPTION: ').lower())
53     message= input('Type your Message:')
54     key= input('Enter the key:')
55
56     if user_choice == 'e':
57         encryption(plaintext= message, key = key)
58     elif user_choice == 'd':
59         decryption(ciphertext= message, key = key)
60
61     choice= (input('Type "yes" to continue or "no" to end: ').lower())
62     if choice == 'no':
63         flag=True
64         print ('Have a Nice Day! Bye...!')
65

```

Output:

```
Type "e" for ENCRYPTION or "d" for DECRYPTION: e
Type your Message:hello world
Enter the key:1234
The Encrypted message is:0X6D0X2C0X1D0XC30X9C0XCC0X700X620X7D0X770X64
Type "yes" to continue or "no" to end: yes
Type "e" for ENCRYPTION or "d" for DECRYPTION: d
Type your Message:0X6D0X2C0X1D0XC30X9C0XCC0X700X620X7D0X770X64
Enter the key:1234
The Decrypted message is:hello world
Type "yes" to continue or "no" to end: no
Have a Nice Day! Bye...!
```

Advantages of RC4

- RC4 stream ciphers are simple to use.
- The speed of operation in RC4 is fast as compared to other ciphers.
- RC4 stream ciphers are strong in coding and easy to implement.
- RC4 stream ciphers do not require more memory.
- RC4 stream ciphers are implemented on large streams of data.

Disadvantages of RC4

- If RC4 is not used with strong MAC then encryption is vulnerable to a bit-flipping attack.
- RC4 stream ciphers do not provide authentication.
- RC4 algorithm requires additional analysis before including new systems.
- RC4 stream ciphers cannot be implemented on small streams of data.
- RC4 fails to discard the beginning of output keystream or fails to use non-random or related keys for the algorithm.

- **Block Cipher:** A block cipher uses a symmetric key and algorithm to encrypt and decrypt a block of data. A block cipher requires an initialization vector (IV) that is added to the input plaintext in order to increase the key space of the cipher and make it more difficult to use brute force to break the key. The IV is derived from a random number generator, which is combined with text in the first block and the key to ensure all subsequent blocks result in ciphertext that does not match that of the first encryption block.

A preferred block size is a multiple of 8 as it is easy for implementation as most computer processor handle data in multiple of 8 bits.

Padding in Block Cipher

Block ciphers process blocks of fixed sizes (say 64 bits). The length of plaintexts is mostly not a multiple of the block size. For example, a 150-bit plaintext provides two blocks of 64 bits each with third block of balance 22 bits. The last block of bits needs to be padded up with redundant information so that the length of the final block equal to block size of the scheme. In our example, the remaining 22 bits need to have additional 42 redundant bits added to provide a complete block. The process of adding bits to the last block is referred to as padding.

Too much padding makes the system inefficient. Also, padding may render the system insecure at times, if the padding is done with same bits always.

Applications of Block Ciphers

Data Encryption: Block Ciphers are widely used for the encryption of private and sensitive data such as passwords, credit card details and other information that is transmitted or stored for a communication. This encryption process converts a plain data into non-readable and complex form. Encrypted data can be decrypted only by the authorised person with the private keys.

File and Disk Encryption: Block Ciphers are used for encryption of entire files and disks in order to protect their contents and restrict from unauthorised users. The disk encryption softwares such as BitLocker, TrueCrypt also uses block cipher to encrypt data and make it secure.

Virtual Private Networks (VPN): Virtual Private Networks (VPN) use block cipher for the encryption of data that is being transmitted between the two communicating devices over the internet. This process makes sure that data is not accessed by unauthorised person when it is being transmitted to another user.

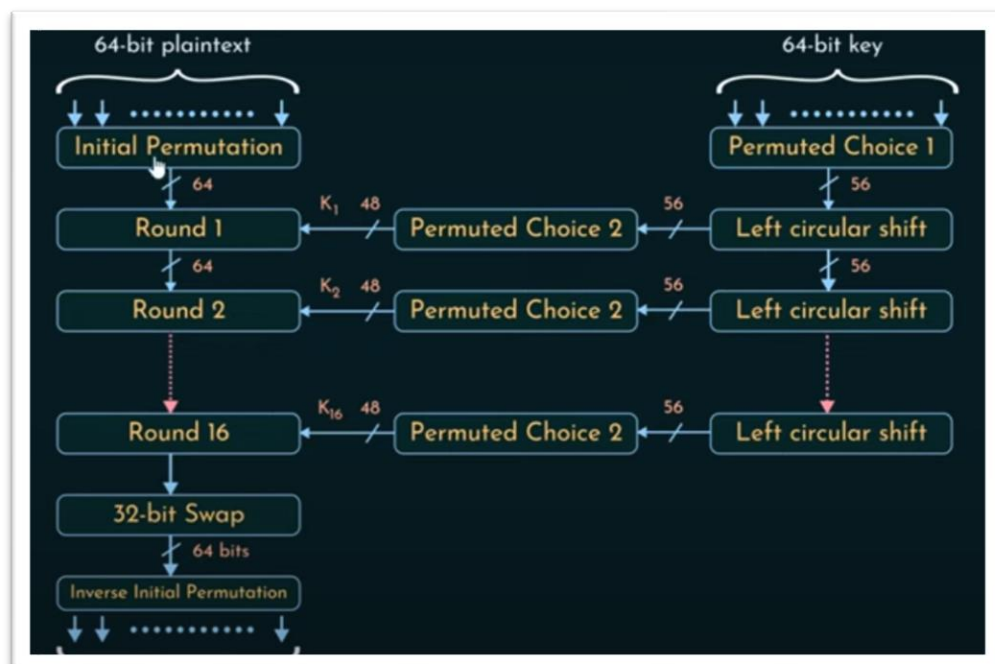
Secure Sockets Layer (SSL) and Transport Layer Security (TLS): SSL and TLS protocols use block ciphers for encryption of data that is transmitted between web browsers and servers over the internet. This encryption process provides security to confidential data such as login credentials, card information etc.

Digital Signatures: Block ciphers are used in the digital signature algorithms, to provide authenticity and integrity to the digital documents. This encryption process generates the unique signature for each document that is used for verifying the authenticity and detecting if any malicious activity is detected.

DES: Data Encryption Standard (DES) is a block cipher with a 56-bit key length that has played a significant role in data security. Data encryption standard (DES) has been found vulnerable to very powerful attacks therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.

Thus, the discarding of every 8th bit of the key produces a 56-bit key from the original 64-bit key.

- DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.
- In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.
- The initial permutation is performed on plain text.
- Next, the initial permutation (IP) produces two halves of the permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT go through 16 rounds of the encryption process.
- In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block.
- The result of this process produces 64-bit ciphertext.



Implementation of DES Cipher in Python:

```
1  from Crypto.Cipher import DES
2  from Crypto.Util.Padding import pad,unpad
3  from Crypto.Random import get_random_bytes
4
5  def des_encrypt(key,plaintext):
6      if len(key) !=8:
7          raise ValueError("key must be 8 bytes long")
8
9      cipher=DES.new(key, DES.MODE_ECB)
10     padded_text=pad(plaintext.encode('utf-8'),DES.block_size)
11     ciphertext=cipher.encrypt(padded_text)
12     return ciphertext
13
14     def des_decrypt(key,ciphertext):
15         if len(key) !=8:
16             raise ValueError("key must be 8 bytes long")
17
18         cipher=DES.new(key, DES.MODE_ECB)
19         padded_text=cipher.decrypt(ciphertext)
20         plaintext=unpad(padded_text,DES.block_size)
21         return plaintext.decode('utf-8')
22
23     key=get_random_bytes(8)
24     plaintext="hello world"
25
26     ciphertext=des_encrypt(key, plaintext)
27     print("Encrypted Text =",ciphertext)
28
29     plain_text = des_decrypt(key, ciphertext)
30     print("Decrypted Text = " ,plain_text )
```

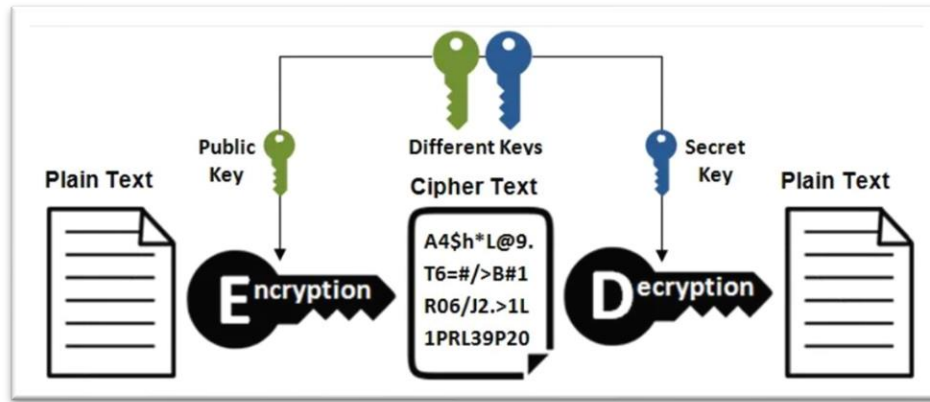
Output:

```
In [3]: runfile('F:/anaconda/DES block cipher.py', wdir='F:/anaconda')
Encrypted Text = b'\xa4gN\x05\x1e\x17P\xa04q\xfd\xd6\x9f\xb2D\xaa'
Decrypted Text =  hello world
```

In conclusion, the **Data Encryption Standard (DES)** is a block cipher with a 56-bit key length that has played a significant role in data security. However, due to vulnerabilities, its popularity has declined. DES operates through a series of rounds involving key transformation, expansion permutation, and substitution, ultimately producing ciphertext from plaintext. While DES has historical significance, it's crucial to consider more secure encryption alternatives for modern data protection needs.

B. Asymmetric cipher: As the name implies, asymmetric encryption is different on each side; the sender and the recipient use two different keys. Asymmetric encryption, also known as public key encryption, uses a public key-private key pairing: data encrypted with the public key can only be decrypted with the private key.

Some examples of asymmetrical cryptography are: **RSA** (Rivest-Shamir-Adleman), **ECC** (Elliptic Curve Cryptography), **DSA**, **PKCs**, **Diffie-Hellman**.



Advantages of Asymmetric Key Cryptography:

- **Increased security:** The use of two different keys makes it more secure than symmetric key cryptography.
- **Non-repudiation:** The digital signature created using the private key provides proof of the authenticity of the sender.
- **Scalability:** Asymmetric key cryptography can support a large number of users.
- **Public key distribution:** The public key can be freely distributed without any security risk, allowing for easy encryption of messages.

Disadvantages of Asymmetric Key Cryptography:

- **Computational overhead:** The encryption and decryption process using asymmetric key cryptography is slower and more resource-intensive compared to symmetric key cryptography.
- **Key management:** Asymmetric key cryptography requires the safekeeping and management of both private and public keys.
- **Key length:** The security of asymmetric key cryptography is directly proportional to the length of the key used. Longer keys require more processing power, making them less practical for some applications.
- **Lack of standardization:** Asymmetric key cryptography is still evolving and there is a lack of standardization in terms of algorithms and key lengths, making it difficult for interoperability between different systems

Where is asymmetric encryption used?

Asymmetric cryptography can also be applied to systems in which many users might need to encrypt and decrypt messages, including the following: Encrypted email. A public key can encrypt an email message, and a private key can decrypt it. SSL/TLS.

How are asymmetric encryption and symmetric encryption used for TLS/SSL?

TLS, historically known as SSL, is a protocol for encrypting communications over a network. TLS uses both asymmetric encryption and symmetric encryption. During a TLS handshake, the client and server agree upon new keys to use for symmetric encryption, called "session keys." Each new communication session will start with a new TLS handshake and use new session keys.

The TLS handshake itself makes use of asymmetric cryptography for security while the two sides generate the session keys, and in order to authenticate the identity of the website's origin server.

RSA: RSA means Rivest, Shamir, Adleman. These are the inventors of the popular RSA Algorithm. The RSA algorithm is a widely used method for encrypting and decrypting messages. It is named after its creators, Ron Rivest, Adi Shamir, and Leonard Adleman, who developed it in 1977. The RSA algorithm is based on the difficulty of factoring large numbers, and it is widely considered to be a secure method for encrypting data.

The Algorithm

The implementation of RSA makes heavy use of modular arithmetic, Euler's theorem, and Euler's totient function. Notice that each step of the algorithm only involves multiplication, so it is easy for a computer to perform:

1. First, the receiver chooses two large prime numbers p and q . Their product, $n = pq$, will be half of the public key.
2. The receiver calculates $\phi(pq) = (p-1)(q-1)$ and chooses a number e relatively prime to $\phi(pq)$. In practice, e is often chosen to be $216 + 1 = 65537$, though it can be as small as 3 in some cases. e will be the other half of the public key.
3. The receiver calculates the modular inverse d of e modulo $\phi(n)$. In other words, $de \equiv 1(\text{mod } \phi(n))$. d is the private key.
4. The receiver distributes both parts of the public key: n and e . d is kept secret.

Now that the public and private keys have been generated, they can be reused as often as wanted. To transmit a message, follow these steps:

1. First, the sender converts his message into a number m . One common conversion process uses the ASCII alphabet:

A	B	C	D	E	F	G	H	I	J	K	L	M
65	66	67	68	69	70	71	72	73	74	75	76	77
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
78	79	80	81	82	83	84	85	86	87	88	89	90

For example, the message "HELLO" would be encoded as **7269767679**. It is important that $m < n$, as otherwise the message will be lost when taken modulo n , so if n is smaller than the message, it will be sent in pieces.

2. The sender then calculates $c \equiv m^e \pmod{n}$. c is the ciphertext, or the encrypted message. Besides the public key, this is the only information an attacker will be able to steal.
3. The receiver computes $c^d \equiv m \pmod{n}$, thus retrieving the original number m .
4. The receiver translates m back into letters, retrieving the original message.

Implementation of RSA algorithm in python

```
1  from math import gcd
2  publickey = 0
3  t = 0
4  #Encryption
5  def encryption(p, q, plaintext):
6      global publickey, t
7      ciphertext= ''
8      n = p * q                # calculating n
9      t = (p - 1) * (q - 1)    # calculating totient, t
10     for i in range (2, t):    # selecting public key, e
11         if gcd(i, t) == 1:
12             publickey = i
13     # performing encryption
14     ascii_value = [ord(char) for char in message]
15     ciphertext = [pow(char, publickey, n) for char in ascii_value]
16     print(f"Encrypted message is {ciphertext}")
```

```

17 #Decryption
18 def decryption(p, q, ciphertext):
19     global publickey, t
20     plaintext= ''
21     n = p * q
22     j = 0
23     while True:
24         if (j * publickey) % t == 1:
25             privatekey = j
26             break
27         j += 1
28
29     # performing decryption
30     encoded_message = [pow(i, privatekey, n) for i in ciphertext]
31     plaintext = ''.join(chr(i) for i in encoded_message)
32     print(f"Decrypted message is {plaintext}")
33
34 #Select p, q (p and q both prime and p not equal to q)
35 flag=False
36 while not flag:
37     user_choice= (input('Type "e" for ENCRYPTION or "d" for DECRYPTION: ').lower())
38     message= input('Type your Message:')
39
40     if user_choice == 'e':
41         encryption(p=3, q=7, plaintext= message)
42     elif user_choice == 'd':
43         decryption(p=3, q=7, ciphertext= message)
44
45     choice= (input('Type "yes" to continue or "no" to end: ').lower())
46     if choice == 'no':
47         flag=True
48     print ('Have a Nice Day! Bye...!')

```

Output

```
Type "e" for ENCRYPTION or "d" for DECRYPTION: e
```

```
Type your Message:hello
```

```
Encrypted message is [20, 5, 12, 12, 6]
```

```
Type "yes" to continue or "no" to end: no
```

```
Have a Nice Day! Bye...!
```

Online Encryption Tools with links

- <https://www.devglan.com/online-tools/text-encryption-decryption>

Text Encryption

Enter any text to be Encrypted

All That Glitters Is Not Gold.

☒ Encrypt with a custom secret key

Enter Secret Key (Remember, the encrypted text can't be decrypted without this secret key)

5

Encrypt

Encrypted Output:

t7ILYFLchbz1+3/sGt0DAORX+CpaU35LaG8bYvNx2cA=

Text Decryption

Enter encrypted text to Decrypt

t7ILYFLchbz1+3/sGt0DAORX+CpaU35LaG8bYvNx2cA=

☒ Decryption requires a custom secret key

Enter Secret Key (The same key used during encryption)

5

Decrypt

Decrypted Text:

All That Glitters Is Not Gold.

- <https://encode-decode.com/aes-128-cbc-hmac-sha256-encrypt-online/>

aes-128-cbc-hmac-sha256 encrypt & decrypt online supported encryptions: aes-128-cbc-hmac-sha256 ▾

hello world

MqJaCDkQztollf2JdfkDg==

1234

Encrypt string →

← Decrypt string

Hashing

Hashing is a one-way mathematical function that turns data into a string of nondescript text that cannot be reversed or decoded.

In the context of cybersecurity, hashing is a way to keep sensitive information and data — including passwords, messages, and documents — secure. Once this content is converted via a hashing algorithm, the resulting value (or hash code) is unreadable to humans and extremely difficult to decrypt, even with the help of advanced technology.

Example Of Hashing

Here is a real-life example of hashing that helps you better understand the process: Consider the input string **‘Hello World’**. A hash function turns it into the following outputs, depending on the algorithm:

MD-5 hash: b10a8db164e0754105b7a99be72e3fe5

SHA-1 hash: 323032c31990d0288ace677ac8f7ad1267c9a92d

Hashing use cases in cybersecurity

Hashing plays an important role in many cybersecurity algorithms and protocols. At the most basic level, hashing is a way to encode sensitive data or text into an indecipherable value that is incredibly difficult to decode.

Below, we discuss three of the most common hashing use cases in cybersecurity:

- **Password Verification:** It is common to store user credentials of websites in a hashed format to prevent third parties from reading the passwords. Since hash functions always provide the same output for the same input, comparing password hashes is much more private.

The entire process is as follows:

- User signs up to the website with a new password.
 - It passes the password through a hash function and stores the digest on the server.
 - When a user tries to log in, they enter the password again.
 - It passes the entered password through the hash function again to generate a digest.
- **Digital signature:** A digital signature is a cryptographic technique used to verify the origin, authenticity, and integrity of a message, document, or transaction.

To create a digital signature using hashing:

- A hash function is applied to the original message to create a secure hash value
- The hash value is then encrypted using a private key that belongs to the sender; this process creates the digital signature
- The recipient uses a public key to decrypt the digital signature

- The recipient then takes the resulting hash value and applies the same hash function; if the hash values match, it proves that the message has not been altered and that it originated with the designated sender

- **File and document management:** Though digital signatures are often used to secure email and other digital communications; they can also authenticate and verify any kind of electronic transaction or document.



- Document comparisons:** During hashing, the hash function will produce a fixed-value character string that serves as a unique identifier for any type of document or file. If the document is altered in any way, even minutely, the hash value will also change. As a result, hashing provides a fast and effective way to compare files and confirm they have not been altered or compromised.

- Data integrity verification:** Some files can be checked for data corruption using hash functions. Like the above scenario, hash functions will always give the same output for similar input, irrespective of iteration parameters.

The entire process follows this order:

- A user uploads a file on the internet.
- It also uploads the hash digest along with the file.
- When a user downloads the file, they recalculate the hash digest.
- If the digest matches the original hash value, file integrity is maintained.
- Now that you have a base foundation set in hashing, you can look at the focus for this tutorial, the MD5 algorithm.

Limitations of hashing

Though hashing is a useful tool, it has its limitations. In this section, we explore a few challenges and drawbacks of using hashing in cybersecurity:

- **Collision:** A collision is when two or more inputs result in the same hash value. Large enterprises or companies that store significant amounts of data will face this challenge and must implement a solution to prevent such collisions. For example, a company might implement a chaining strategy, which is when a duplicative value is added to a linked list on the hash table.
- **Performance:** Hashing can be a bit of a balancing act. Algorithms are designed to optimize both speed and memory use; they must also be able to support the level of data input needed by the company. This creates significant complexity in terms of algorithm design and evolution. With respect to cybersecurity, a slow algorithm or one facing significant backlogs can translate into higher risk.
- **Security risks:** By definition, hashing is a one-way conversion of data into an indecipherable string of text. Generally, it is impossible to reconvert the hash value back to data. However, some sophisticated adversaries can either discover or guess the hash function, which would allow them to reverse engineer the hash values or tamper with the dataset by creating fake inputs.

MD5 Hash: MD5 (Message Digest Method 5) is a cryptographic hash algorithm used to generate a 128-bit digest from a string of any length. It represents the digests as 32 digit hexadecimal numbers.

MD5 was developed as an improvement of MD4, with advanced security purposes. The output of MD5 (Digest size) is always 128 bits. MD5 was developed in 1991 by Ronald Rivest.

Vulnerabilities of MD5: length extension attacks, Collision Vulnerabilities, brute forced attack

How does the MD5 Algorithm works (4 steps)

Padding Bits: When you receive the input string, you have to make sure the size is 64 bits short of a multiple of 512. When it comes to padding the bits, you must add one(1) first, followed by zeroes to round out the extra characters.

- **Formula:** Length (original message + padding bits) = $512 * i - 64$ where $i = 1, 2, 3 \dots$

Padding Length: You need to add a few more characters to make your final string a multiple of 512. To do so, take the length of the initial input and express it in the form of 64 bits. On combining the two, the final string is ready to be hashed.

- After adding both we will get $512 * n$ i.e. the exact multiple of 512.

Initialize MD Buffer: The entire string is converted into multiple blocks of 512 bits each. You also need to initialize four different buffers, namely A, B, C, and D. These buffers are 32 bits each.

Process Each 512-bit Block: Each 512-bit block gets broken down further into 16 sub-blocks of 32 bits each. There are four rounds of operations, with each round utilizing all the sub-blocks, the buffers, and a constant array value.

- This constant array can be denoted as $T[1] \rightarrow T[64]$.
- Each of the sub-blocks are denoted as $M[0] \rightarrow M[15]$.

Implementation of md5 hashing in python

```
1 # importing the hashlib libraries
2 import hashlib
3
4 message = input('Type your Message:')
5 # encoding the message using the library function
6 output = hashlib.md5(message.encode())
7 print(f"Hash of the input string: {output.hexdigest()}")
8
```

Output:

```
Type your Message:Hello World
Hash of the input string: b10a8db164e0754105b7a99be72e3fe5
```

Some online MD5 hash generator with links

1. <https://www.md5hashgenerator.com/>

Use this generator to create an MD5 hash of a string:

All that glitters isn't gold

Generate →

Your String	All that glitters isn't gold
MD5 Hash	8ba3df53dc24aa52c53bf025f18052be Copy

2. <https://www.miraclesalad.com/webtools/md5.php>

String(s):

All that glitters isn't gold

md5 ☐ Treat multiple lines as separate strings (blank lines are ignored)
☐ Uppercase hash(es)
☐ Blur string(s)

MD5 Hash(es):

8ba3df53dc24aa52c53bf025f18052be

SHA1 Hash: Among various hash functions available today, one of the most widely used algorithms is the Secure Hash Algorithm 1 (SHA-1). SHA-1 is a cryptographic hash function that produces a 160-bit hash value from an input message of any size, up to $2^{64} - 1$ bit. SHA-1 was designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) in 1995 as a part of the Secure Hash Standard (SHS). The main application of SHA1 is to protect communications from being intercepted by outside parties.

Vulnerabilities of SHA-1: Birthday Attack, Man-in-the-Middle Attack, Certificate Forgery.

How does the SHA1 Algorithm works (6 steps)

Padding the Message: The message is padded so that its length is congruent to 448 modulo 512. Padding involves appending a single '1' bit followed by enough '0' bits to make the total length 64 bits shy of a multiple of 512.

Appending the Original Length: After padding, the original length of the message (before padding), in bits, is appended as a 64-bit big-endian integer. This ensures the total length of the message is now a multiple of 512 bits.

Initializing the Hash Values: SHA-1 uses five initial hash values (h0, h1, h2, h3, h4) which are constants. These values are: h0 = 0x67452301, h1 = 0xEFCDAB89, h2 = 0x98BADCFE, h3 = 0x10325476, h4 = 0xC3D2E1F0

Processing the Message in 512-bit Blocks: The padded message is processed in chunks of 512 bits. Each chunk is divided into sixteen 32-bit words. These words are expanded into eighty 32-bit words using bitwise operations.

80-Step Compression Function: For each of the eighty steps, a specific function is applied based on the current step. The functions involve bitwise operations and modular additions with predefined constants. These functions mix the input data to produce the final hash values.

Producing the Final Hash Value: The hash values (h0, h1, h2, h3, h4) are updated in each step and after processing all blocks, the final hash value is produced by concatenating the updated values of h0, h1, h2, h3, and h4. This concatenated value is the 160-bit hash output.

Implementation of SHA1 Algorithm in python

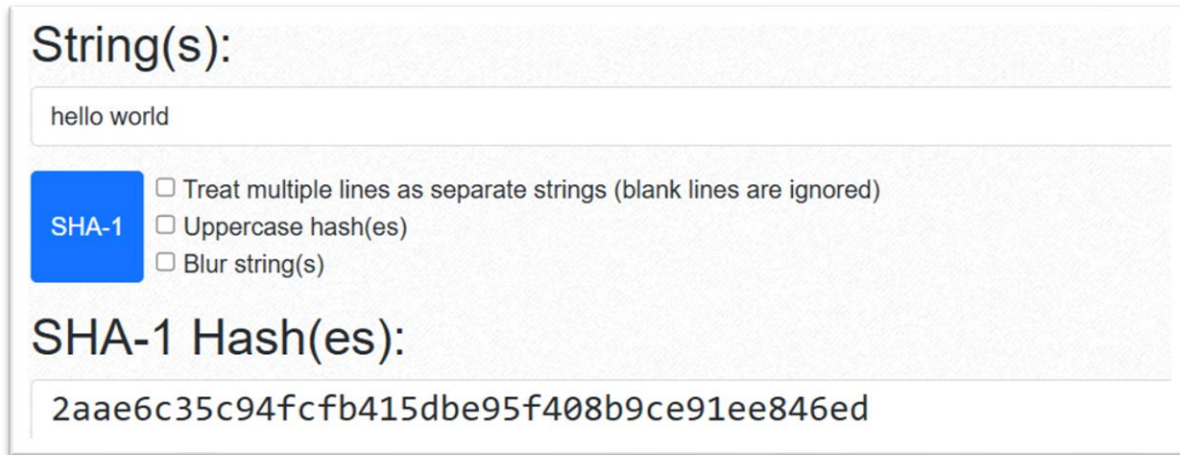
```
1  # importing the hashlib libraries
2  import hashlib
3
4  message = input('Type your Message:')
5  # encoding the message using the library function
6  output = hashlib.sh1(message.encode())
7  print(f"Hash of the input string: {output.hexdigest()}")
8
```

Output:

```
Type your Message:hello world
Hash of the input string: 2aae6c35c94fcfb415dbe95f408b9ce91ee846ed
```

Some online SHA1 hash generator with links

1. <https://www.miraclesalad.com/webtools/sha1.php>



String(s):

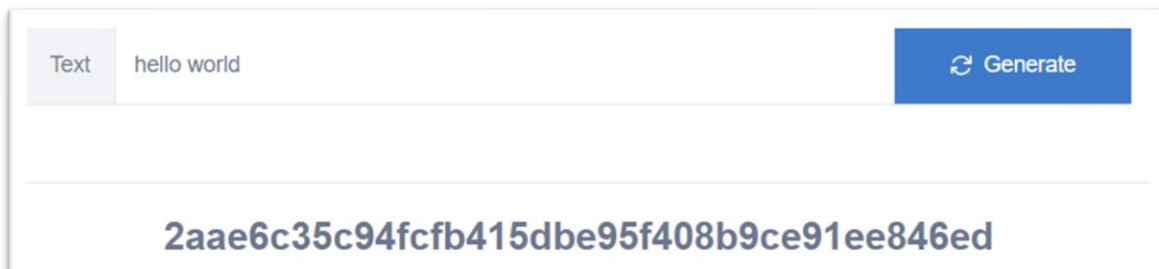
hello world

SHA-1 ☐ Treat multiple lines as separate strings (blank lines are ignored)
☐ Uppercase hash(es)
☐ Blur string(s)

SHA-1 Hash(es):

2aae6c35c94fcfb415dbe95f408b9ce91ee846ed

2. <https://codeshack.io/sha1-hash-generator/>



Text hello world **Generate**

2aae6c35c94fcfb415dbe95f408b9ce91ee846ed

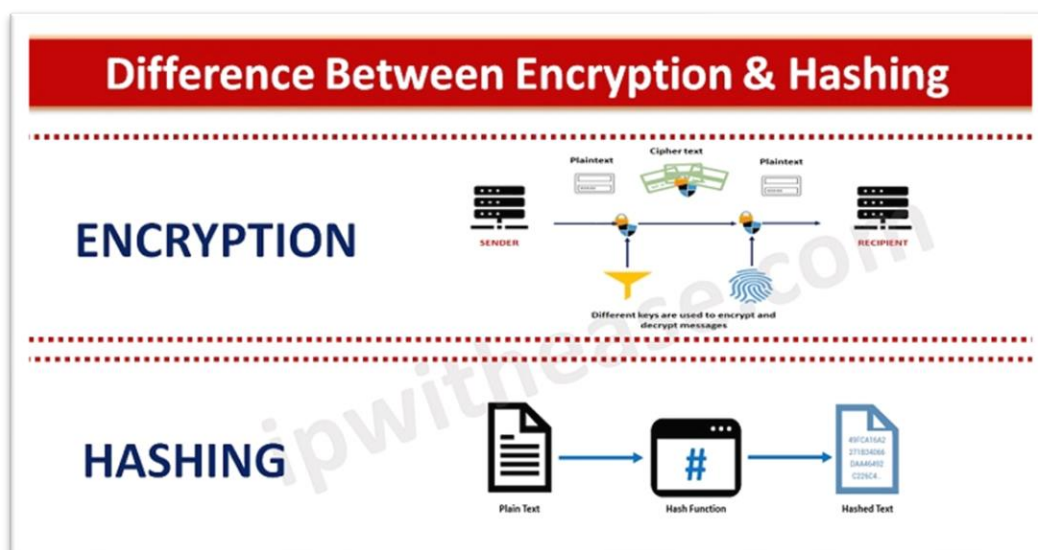
Hashing vs. Encryption

Though hashing and encryption may seem to result in the same outcome, they are actually two different functions.

The main difference is that hashing is always intended to be a one-way conversion of data. The hash value is a unique string of text that can only be decoded if the adversary is able to steal or guess the hash function and then reverse engineer the data input.

Data encryption, on the other hand, is a two-way process. Though encryption also uses cryptographic algorithms to convert plain text into an encoded format, it has a corresponding decoding key that allows users to decrypt the data.

Another key difference is that hashing provides you with the ability to authenticate data, messages, files, or other assets. Users can confirm that data sent from one user to another has not been intercepted and altered by comparing the original hash value with the one produced by the recipient. With encrypted data, on the other hand, there is no way to validate the data or tell if it has been changed, which is why hashing is preferred for authentication purposes.



Conclusion

Cryptography serves as the bedrock of secure communication in the digital era, providing the tools and techniques necessary to protect information from unauthorized access and tampering. Whether it's safeguarding financial transactions, securing sensitive communications, or preserving user privacy, the various types of cryptography play pivotal roles in ensuring the integrity and confidentiality of data. As technology continues to advance, cryptography remains a dynamic field, adapting to new challenges and evolving threat landscapes. Understanding its principles and types is essential for individuals and organizations seeking to navigate the complex landscape of digital security.

Limitations

While I was making this project, I found some algorithms are best-used for Cloud Computing, some others are for an E-Commerce website, some others for Social Media application, etc. that would lead to decision making of which are more suitable for algorithm for every project. So, I was limited in the following points –

Complexity: Implementing cryptographic algorithms and hashing techniques can be complex and require a deep understanding of mathematical concepts and programming skills.

Performance Overhead: Cryptographic operations can be resource-intensive, leading to increased processing time and power consumption, which may not be suitable for real-time applications.

Security Flaws: Potential vulnerabilities exist if the algorithms are not implemented correctly, leading to security risks such as cryptographic attacks.

Key Management: Proper management of cryptographic keys is crucial. Poor key management can compromise the entire security framework.

Compatibility Issues: Integrating cryptographic techniques with existing systems and ensuring compatibility across different platforms can be challenging.

User Education: Users may need education and training to use cryptographic systems effectively, as improper use can negate security benefits.

Regulatory Compliance: Adhering to various legal and regulatory requirements regarding data encryption and privacy can add complexity to the project.

Scalability: Ensuring that cryptographic solutions scale effectively with increasing data volumes and user numbers can be difficult.

Upgradability: As new cryptographic vulnerabilities are discovered, updating and maintaining the cryptographic systems to mitigate these issues is essential but can be cumbersome.

Resource Constraints: Limited hardware capabilities, especially in embedded systems or mobile devices, may restrict the use of advanced cryptographic techniques due to resource constraints.

Legal Considerations: The main implementation limitation of Cryptographic algorithms is patents.

Bibliography

Google Website links:

- www.geeksforgeeks.com
- www.javatpoint.com
- www.simplilearn.com
- <https://in.indeed.com/career-advice/>
- <https://www.crowdstrike.com/cybersecurity-101/data-protection/data-hashing/>
- <https://onboardbase.com/blog/rsa-encryption-decryption/>
- <https://www.cryptool.org/>
- https://www.comparitech.com/blog/information-security/rsa-encryption/#What_is_RSA_encryption
- <https://brilliant.org/wiki/rsa-encryption/>
- <https://medium.com/@gowtham180502/implementing-rsa-algorithm-using-python-836f7da2a8e0>
- <https://codereview.stackexchange.com/>

YouTube Video Links:

- https://youtu.be/D_PfV_IcUdA?si=ZoI1tG9qHQTtoOOZA
- https://youtu.be/-W8RWRab6H4?si=DFqLzbutkId_F7X9
- <https://youtu.be/MLIwtjYWJ3M?si=PVOGhlE7mtI0h7if>
- https://youtu.be/1Sy2kBQ3igM?si=u8E_l4k_7AIMwcmH
- <https://youtu.be/JEsUlX0Ps9k?si=gk0W1xb4EEMntdYp>
- <https://youtu.be/KClySt74W2I?si=ZKSaQIQHi-8K7iWJ>