**MINOR PROJECT**

**Title:** Learning Cryptography with Python programming.

**Mentor**

Dr. Priya Sajan

Sr. Project Engineer,

CDAC, TVM

**Team Members**

Gopi Aghav

Harshit Upadhyaya

Himanshu Shekhar

Jyothilaxmi P.

Kailas Kalyane

Vipul Bhirud

1. Abstract

Cryptography is one of the important tools in any hackers toolbox. Learning about encryption, decryption and implementing the code in python is long way to go. The objective of this project is to be useful for networking professionals as well as hackers who want to implement new frameworks instead of following a traditional approach.

Keywords: Cryptography, encryption, decryption, hackers, python.

1. Introduction

Cryptography is the one used widely among computer science projects to secure the data messages. There are hundreds of cryptographic algorithms available now a days, but here our idea behind bringing this tools to understand the basics of cryptography with basic and common algorithms.

1. Methodology

[Cryptography](https://www.geeksforgeeks.org/cryptography-introduction-to-crypto-terminologies/) is a technique of securing information and communications through the 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 “crypt” means “hidden” and the suffix “graphy” means “writing”.

In Cryptography, the techniques that 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 them.

These algorithms are used for cryptographic key generation, digital signing, and verification to protect data privacy, web browsing on the internet and to protect confidential transactions such as credit card and debit card transactions.

**Techniques Used for Cryptography:** In today’s age of computers cryptography is often associated with the process where an ordinary plain text is converted to cipher text which is the text made such that the intended receiver of the text can only decode it and hence this process is known as encryption. The process of conversion of cipher text to plain text is known as decryption.

**Features Of Cryptography are as follows:**

1. **Confidentiality:** Information can only be accessed by the person for whom it is intended and no other person except him can access it.
2. **Integrity:** Information cannot be modified in storage or transition between sender and intended receiver without any addition to information being detected.
3. **Non-repudiation:** The creator/sender of information cannot deny his intention to send information at a later stage.
4. **Authentication:** The identities of the sender and receiver are confirmed. As well as destination/origin of information is confirmed.

**Types of Cryptography:**

There are several types of cryptography, each with its own unique features and applications. Some of the most common types of cryptography include:

**1. Symmetric-key cryptography:** This type of cryptography involves the use of a single key to encrypt and decrypt data. Both the sender and receiver use the same key, which must be kept secret to maintain the security of the communication.

**2. Asymmetric-key cryptography:** Asymmetric-key cryptography, also known as public-key cryptography, uses a pair of keys – a public key and a private key – to encrypt and decrypt data. The public key is available to anyone, while the private key is kept secret by the owner.

**Hash functions:** A hash function is a mathematical algorithm that converts data of any size into a fixed-size output. Hash functions are often used to verify the integrity of data and ensure that it has not been tampered with.

**Applications Of Cryptography:**

1. **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.
2. **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.
3. **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.
4. **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.
5. **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.
6. **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.
7. **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.

**Advantages:**

1. **Access Control:** Cryptography can be used for access control to ensure that only parties with the proper permissions have access to a resource. Only those with the correct decryption key can access the resource thanks to encryption.
2. **Secure Communication:** For secure online communication, cryptography is crucial. It offers secure mechanisms for transmitting private information like passwords, bank account numbers, and other sensitive data over the Internet.
3. **Protection against attacks:** Cryptography aids in the defence against various types of assaults, including replay and man-in-the-middle attacks. It offers strategies for spotting and stopping these assaults.
4. **Compliance with legal requirements:** Cryptography can assist firms in meeting a variety of legal requirements, including data protection and privacy legislation.

[Cryptography](https://www.geeksforgeeks.org/cryptography-introduction-to-crypto-terminologies/) is the science of secret writing with the intention of keeping the data secret. Cryptography is classified into symmetric cryptography, asymmetric cryptography, and [hashing](https://www.geeksforgeeks.org/what-is-hashing/).

### **Private Key:**

In the Private key, the same key (secret key) is used for encryption and decryption. In this key is symmetric because the only key is copied or shared by another party to decrypt the cipher text. It is faster than public-key cryptography.

### **Public Key:**

In a [Public key](https://www.geeksforgeeks.org/public-key-encryption/), two keys are used one key is used for encryption and another key is used for decryption. One key (public key) is used to encrypt the plain text to convert it into cipher text and another key (private key) is used by the receiver to decrypt the cipher text to read the message.

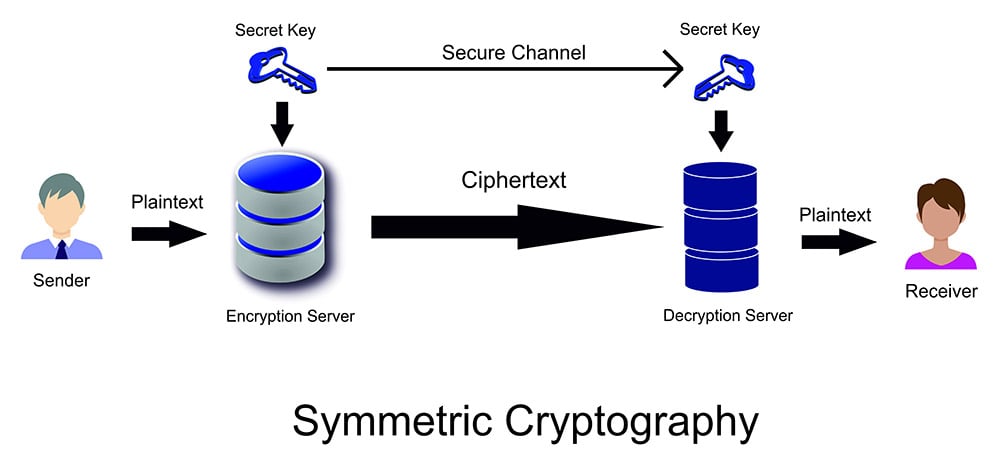
### **What is symmetric encryption?**

Symmetric encryption is a widely used data encryption technique whereby data is encrypted and decrypted using a single, secret cryptographic key.

Specifically, the key is used to encrypt plaintext - the data’s pre-encryption or post-decryption state - and decrypt ciphertext - the data’s post-encryption or pre-decryption state.

Symmetric encryption is one of the most widely used encryption techniques and also one of the oldest, dating back to the days of the Roman Empire. Caesar’s cipher, named after none other than Julius Caesar, who used it to encrypt his military correspondence, is a famous historical example of symmetric encryption in action.

The goal of symmetric encryption is to [secure sensitive information](https://www.trentonsystems.com/cybersecurity?hsLang=en-us). It’s used daily in many major industries, including defense, aerospace, banking, health care, and other industries in which securing a person’s, business’, or organization’s sensitive data is of the utmost importance.



Graphic: This is an illustration of the symmetric encryption process.

### **How does symmetric encryption work?**

Symmetric encryption works by using either a stream cipher or block cipher to encrypt and decrypt data. A stream cipher converts plaintext into ciphertext one byte at a time, and a block cipher converts entire units, or blocks, of plaintext using a predetermined key length, such as 128, 192, or 256 bits.

Senders and recipients using symmetric encryption to transfer data to each other must know the secret key to, in the case of senders, encrypt the data they intend to share with recipients, and in the case of recipients, decrypt and read the encrypted data the senders share with them, as well as encrypt any necessary responses.

Here’s a simplified example of symmetric encryption: if Claire, the sender, wants to send Jacqueline, the recipient, a confidential document, Claire would use the secret key to encrypt the file and send it to Jacqueline, who would be unable to read its contents until she entered the same key that Claire just used to encrypt the file. Conversely, if Jacqueline makes changes to the document and wishes to share them with Claire, she’d use the same key to re-encrypt the file and send it back to Claire, who will use the same key to decrypt the file and access its contents, and the process repeats itself.

Note that this is just an example used to simplify how symmetric encryption works. Symmetric encryption may be carried out manually or automatically.

Symmetric encryption is not limited to the sharing of data between one sender and one recipient, however. Symmetrically encrypted information can be accessed by anyone – Claire, Jacqueline, their co-worker Frank, their boss, Jennifer, et al. – who knows the secret key. Therein lies the reason why concealing the shared cryptographic key from unauthorized parties is vital to the success of symmetric encryption and the integrity of symmetrically encrypted data.

### **What is asymmetric encryption?**

Unlike symmetric encryption, which uses the same secret key to encrypt and decrypt sensitive information, asymmetric encryption, also known as public-key cryptography or public-key encryption, uses mathematically linked public- and private-key pairs to encrypt and decrypt senders’ and recipients’ sensitive data.

As with symmetric encryption, plaintext is still converted into ciphertext and vice versa during encryption and decryption, respectively. The main difference is that two unique key pairs are used to encrypt data asymmetrically.



Graphic: This is an illustration of the asymmetric encryption process.

### **How does asymmetric encryption work?**

Here’s a simplified example of asymmetric encryption: if Claire, the sender, and Jacqueline, the recipient, want to continually send a confidential file back and forth to each other, Claire and Jacqueline will give their unique and respective public keys to each other. Claire will then use Jacqueline’s public key to encrypt the file, since it’s intended for Jacqueline only, and send the file to Jacqueline. Upon receipt of the file, Jacqueline will use her private key – keyword, “private,” meaning no one else other than Jacqueline knows it - to decrypt the file and access its contents. No one other than Jacqueline, not even Claire, can decrypt this file, because no one other than Jacqueline knows Jacqueline’s private key. The same process applies when Jacqueline wants to send the file back to Claire. Jacqueline ties it to Claire’s public key, and Claire uses her private key to decrypt the file.

Note that this is a simplification of asymmetric encryption. Like symmetric encryption, asymmetric encryption may be carried out manually or automatically.

Now, do you see how asymmetric encryption could be seen as more secure than symmetric encryption? While this is an interesting inquiry, it’s not the right question to ask, really, because, technically, whether symmetric or asymmetric encryption is more secure [depends largely on key size](https://www.thesslstore.com/blog/asymmetric-vs-symmetric-encryption/) and the security of the media that stores or transmits cryptographic keys.

One reason asymmetric encryption is often regarded as more secure than symmetric encryption is that asymmetric encryption, unlike its counterpart, does not require the exchange of the same encrypt-decrypt key between two or more parties. Yes, public keys are exchanged, but users sharing data in an asymmetric cryptosystem have unique public and private key pairs, and their public keys, because they’re used for encryption only, pose no risk of unauthorized decryption by hackers should they become known, because the hackers, assuming private keys are kept private, don’t know the users’ private keys and thus cannot decrypt the encrypted data.

Asymmetric encryption also allows for digital signature authentication, unlike symmetric encryption. Basically, this involves using private keys to digitally sign messages or files, and their corresponding public keys are used to confirm that these messages originated from the correct, verified sender.

**Cryptographic models explained**

1. Caesar cipher
2. ROT13
3. Vignere Cypher
4. XOR Encoding
5. Base64 Encoding
6. RSA Algorithm

**2. ROT 13**

ROT13, short for "rotate by 13 places," is a simple letter substitution cipher that replaces each letter in a text with the letter 13 positions ahead or behind it in the alphabet. It's a type of Caesar cipher, which is a basic form of encryption where each letter in the plaintext is shifted by a fixed number of positions in the alphabet.

In ROT13, the letters are shifted by 13 positions. The interesting property of ROT13 is that applying it twice (rotating by 13 positions twice) results in the original text. In other words, ROT13 is its own inverse.

Here's the correspondence between the letters before and after applying ROT13:

Original: 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

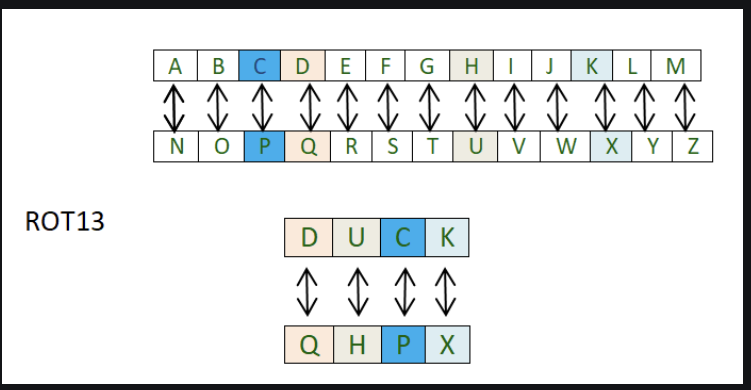
ROT13: N O P Q R S T U V W X Y Z A B C D E F G H I J K L M

So, if you want to encrypt a message using ROT13, you would replace each letter with its corresponding letter from the ROT13 alphabet. Non-alphabetic characters like numbers and symbols are not affected by ROT13 and remain unchanged.

For example, the message "HELLO, WORLD!" would be encrypted using ROT13 as "URYYB, JBEYQ!"

In many contexts, ROT13 is used as a simple and fun way to obfuscate text, often for spoilers or for hiding content that users can choose to reveal if they want. Since ROT13 is symmetric (applying it twice restores the original text), it's not considered a secure form of encryption for protecting sensitive information.

ROT13 cipher(read as – “rotate by 13 places”) is a special case of the Ceaser cipher in which the shift is always 13.

So every letter is shifted 13 places to encrypt or to decrypt the message.

You must think that it is just another caesar cipher so what’s different this time? Well the difference is in its implementation. The approach is to use two separate python dictionaries.

First one to lookup the various letters according to their place in the English alphabets to get the shifted number

Second one to get the letters which correspond to those shifted numbers.

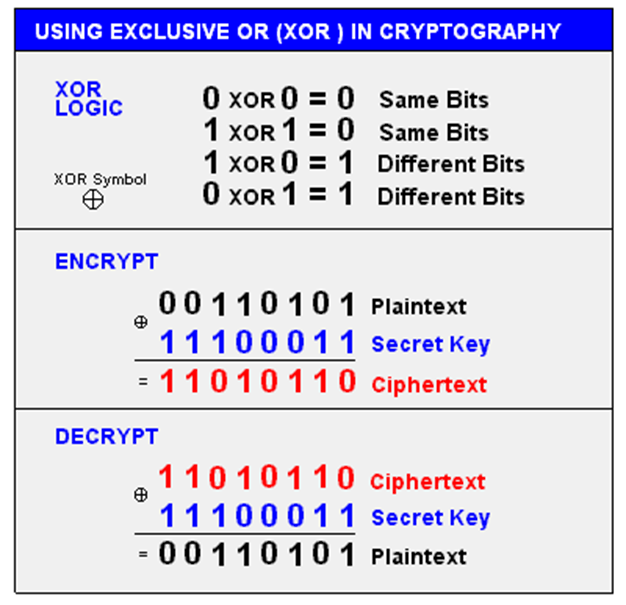
3.XOR PROCESS

XOR is a bitwise operator, and it stands for "exclusive or." It performs logical operation. If input bits are the same, then the output will be false(0) else true(1).

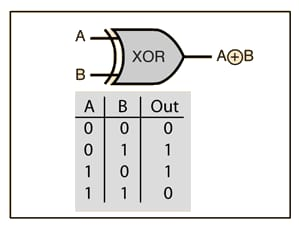
(e**X**clusive **OR**) A Boolean logic operation that is widely used in cryptography as well as in generating parity bits for error checking and fault tolerance. XOR compares two input bits and generates one output bit. The logic is simple. If the bits are the same, the result is 0. If the bits are different, the result is 1.

**Several Symbols**

Various symbols are used to designate the XOR operation including a + sign inside a circle, an underlined "V" and the caret (^). See [cryptography](https://www.pcmag.com/encyclopedia/term/cryptography), [RAID](https://www.pcmag.com/encyclopedia/term/raid) and [or](https://www.pcmag.com/encyclopedia/term/or).



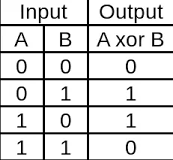
**XOR Symbol:-**

****

**XOR in Cryptography**

This example uses XOR and the same "secret key" to encrypt and decrypt. Although XOR logic may be used, secret key algorithms are a lot more sophisticated than this. See [encryption algorithm](https://www.pcmag.com/encyclopedia/term/encryption-algorithm).

XOR encoding work:-



XOR, or “exclusive or” operates on binary data. It returns true if both of its inputs are opposites (one false and one true), otherwise, it returns false. The XOR cipher isn't used in production because it is impractical to use keys that are the same length as the message body.

# Xor cipher:-

# **XOR bitwise operation**

**The ultimate shift cipher**

If you’ve seen the lesson on the [one-time pad](https://www.khanacademy.org/math/applied-math/cryptography/crypt/v/one-time-pad), you know that it is the **ultimate shift cipher**. It involves the application of a random list of shifts equal to the length of the message. It’s important to understand exactly how and why the one-time pad is unbreakable, or, [perfectly secret](https://www.khanacademy.org/math/applied-math/cryptography/crypt/v/perfect-secrecy).

To understand why, we need to first introduce the **AND**, **OR** and **XOR** bitwise operations. Specifically why XOR must be used when performing the one-time pad on computers. **Bitwise** simply means that we are dealing with individual bits, or **binary numbers**. In any modern/computerized encryption scheme we represent our symbols using binary digits.

In [cryptography](https://en.wikipedia.org/wiki/Cryptography), the **simple XOR cipher** is a type of *additive*[*cipher*](https://en.wikipedia.org/wiki/Cipher), an [encryption algorithm](https://en.wikipedia.org/wiki/Encryption_algorithm) that operates according to the principles:

A ⊕ 0 = A,

A ⊕ A = 0,

A ⊕ B = B ⊕ A,

(A ⊕ B) ⊕ C = A ⊕ (B ⊕ C),

(B ⊕ A) ⊕ A = B ⊕ 0 = B,

For example where ⊕ denotes the [exclusive disjunction](https://en.wikipedia.org/wiki/Exclusive_disjunction) (XOR) operation. This operation is sometimes called modulus 2 addition (or subtraction, which is identical). With this logic, a string of text can be encrypted by applying the bitwise XOR operator to every character using a given key. To decrypt the output, merely reapplying the XOR function with the key will remove the cipher.

Example:-

The string "Wiki" (01010111 01101001 01101011 01101001 in 8-bit [ASCII](https://en.wikipedia.org/wiki/ASCII)) can be encrypted with the repeating key 11110011 as follows:

|  |  |
| --- | --- |
|  | 01010111 01101001 01101011 01101001 |
| ⊕ | 11110011 11110011 11110011 11110011 |
| = | 10100100 10011010 10011000 10011010 |

And conversely, for decryption:

|  |  |
| --- | --- |
|  | 10100100 10011010 10011000 10011010 |
| ⊕ | 11110011 11110011 11110011 11110011 |
| = | 01010111 01101001 01101011 01101001 |
|  |  |

Use and security:-

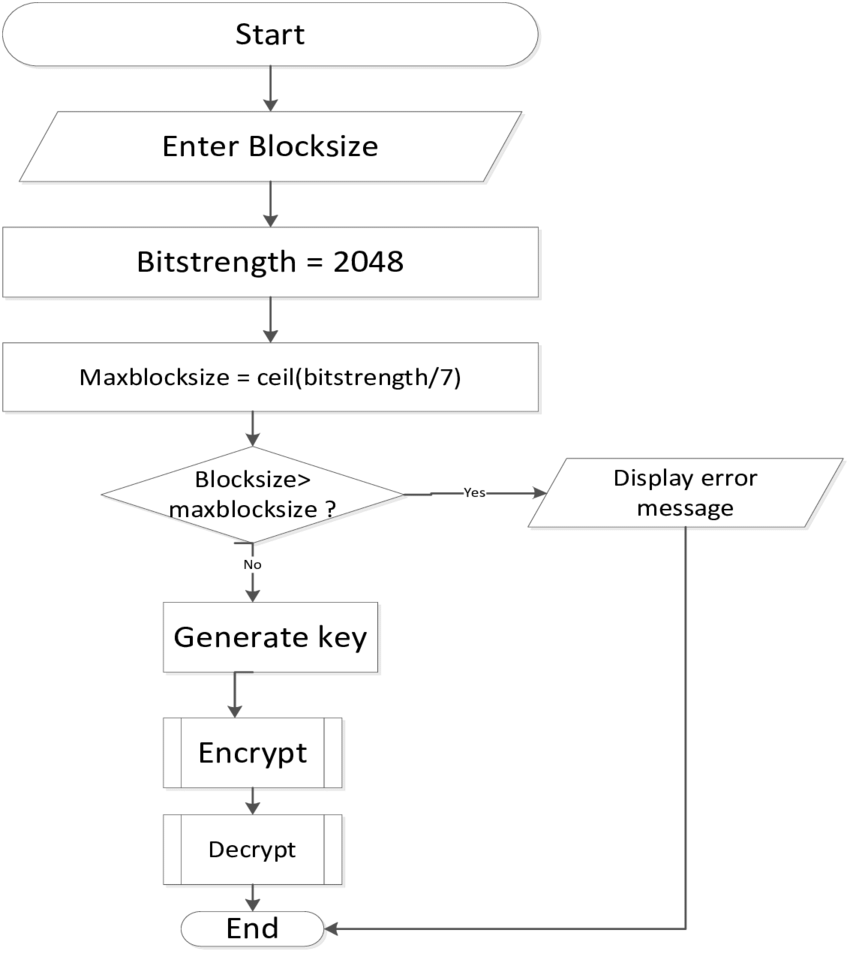
The XOR operator is extremely common as a component in more complex ciphers. By itself, using a constant repeating key, a simple XOR cipher can trivially be broken using [frequency analysis](https://en.wikipedia.org/wiki/Frequency_analysis). If the content of any message can be guessed or otherwise known then the key can be revealed. Its primary merit is that it is simple to implement, and that the XOR operation is computationally inexpensive. A simple repeating XOR (i.e. using the same key for xor operation on the whole data) cipher is therefore sometimes used for hiding information in cases where no particular security is required. The XOR cipher is often used in computer [malware](https://en.wikipedia.org/wiki/Malware) to make reverse engineering more difficult.

If the key is random and is at least as long as the message, the XOR cipher is much more secure than when there is key repetition within a message. When the keystream is generated by a [pseudo-random number generator](https://en.wikipedia.org/wiki/Pseudo-random_number_generator), the result is a [stream cipher](https://en.wikipedia.org/wiki/Stream_cipher). With a key that is [truly random](https://en.wikipedia.org/wiki/Hardware_random_number_generator), the result is a [one-time pad](https://en.wikipedia.org/wiki/One-time_pad), which is [unbreakable in theory](https://en.wikipedia.org/wiki/Information-theoretic_security).

Algorithm:-

XOR algorithm of encryption and decryption converts the plain text in the format ASCII bytes and uses XOR procedure to convert it to a specified byte. It offers the following advantages to its users −

* Fast computation
* No difference marked in left and right side
* Easy to understand and analyze



**4.Base64 Encryption and Decryption**

Base64 is not encryption; rather, it's a method of encoding binary data into an ASCII text format. It's often used to represent binary data, such as images or files, as text strings that can be safely transmitted over text-based protocols like email or included in URLs.

Here's how Base64 encoding works:

**1. Binary Data:**

Any type of binary data, like images or files, is a sequence of bytes (8-bit chunks).

**2. Encoding:**

Base64 takes these bytes and divides them into groups of 3 bytes. Each group (24 bits) is then split into four 6-bit chunks.

**3. Character Mapping:**

Each 6-bit chunk is represented by a character from a predefined set of 64 characters. These characters include uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), and two additional characters (usually '+', '/'). Padding characters ('=') might be added at the end if the length isn't a multiple of 3 bytes.

**4. Textual Representation:**

The resulting sequence of characters forms a textual representation of the binary data. This encoded text is safe to transmit over text-based protocols.

**5. Decoding:**

To retrieve the original binary data, the recipient decodes the Base64-encoded text. Each character is converted back into its 6-bit representation, and the original binary data is reconstructed.

It's important to note that Base64 is not a method of encryption and does not provide any security against unauthorized access. Base64-encoded data can be easily decoded by anyone who knows the encoding scheme. Its primary use is for encoding binary data into a format that text-based systems can handle.

For example, when you see images embedded in email signatures or when you come across long strings of characters in URLs, those are often Base64-encoded representations of the original binary data.

The Base64 encoding process converts binary data into a text-based format using a set of 64 distinct characters. This encoding is commonly used to represent binary data, such as images or files, as text strings that can be safely transmitted over text-based protocols. Here's a step-by-step explanation of the Base64 encoding process:

**1. Binary Data Input:**

Start with the binary data you want to encode. This data consists of a sequence of bytes (8-bit chunks).

**2. Dividing into 6-Bit Groups:**

Divide the binary data into groups of 3 bytes (24 bits). Each group contains 3 bytes (3 \* 8 bits = 24 bits), which can be represented as four 6-bit chunks.

**3. Padding:**

If the length of the binary data is not divisible by 3, padding is added to ensure that the last group has 3 bytes. Padding is done using the '=' character.

**4. Character Mapping:**

Convert each 6-bit chunk into its corresponding character from the Base64 character set. The character set typically includes 64 characters: uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), and two additional characters (usually '+', '/'). Each character corresponds to a specific 6-bit pattern.

**5. Concatenation:**

Concatenate the Base64 characters from all the groups to form the Base64-encoded string.

**6. Encoded Output:**

The resulting string is the Base64-encoded representation of the original binary data.

Here's a visual example of the Base64 encoding process for a short piece of binary data:

**Binary Data:** 10110100 11101011 00100100 (in binary) = B4 EB 24 (in hexadecimal)

**Dividing into 6-Bit Groups:**

* Group 1: 101101 00 (B4)
* Group 2: 111010 11 (EB)
* Group 3: 001001 00 (24)

**Character Mapping:**

* Group 1: Base64 characters: 17 (decimal) => R (Base64 character)
* Group 2: Base64 characters: 29 (decimal) => F (Base64 character)
* Group 3: Base64 characters: 10 (decimal) => K (Base64 character)

**Concatenation:**

The Base64-encoded string for the given binary data is "RFK”.

It's important to note that Base64 encoding is not a secure form of encryption. It's a reversible transformation, meaning anyone with access to the encoded data can easily decode it back to its original binary form. The primary purpose of Base64 encoding is to represent binary data in a format that's compatible with text-based protocols and systems.

The Base64 decoding process is the reverse of the encoding process. It converts a Base64-encoded string back into its original binary data. Here's a step-by-step explanation of the Base64 decoding process:

**1. Base 64 Encoded Input:**

Begin with the Base64-encoded string that you want to decode.

**2. Character Mapping:**

For each character in the Base64-encoded string, convert it back to its 6-bit representation. Each character corresponds to a specific 6-bit pattern.

**3. 6 Bit Groups:**

Group the 6-bit representations of characters into sets of four, forming a complete 24-bit group.

**4. Binary Conversion:**

Convert the 24-bit group into three 8-bit bytes (bytes are 8 bits each).

**5. Binary Output:**

The resulting sequence of bytes is the original binary data that was encoded using Base64.

**6. Padding Handling:**

If the Base64-encoded string includes padding ('=' characters), these padding characters are ignored during the decoding process.

Here's a visual example of the Base64 decoding process using the "RFK" example from the previous response:

**Base64 Encoded String: “RFK”**

**Character Mapping:**

* 'R' corresponds to 17 (decimal) => 10001 (binary)
* 'F' corresponds to 29 (decimal) => 11101 (binary)
* 'K' corresponds to 10 (decimal) => 01010 (binary)

**Concatenation of 6-Bit Groups:**

* 100011101010010101010 (concatenation of the 6-bit groups)

**Binary Conversion:**

* Group 1: 10001110 (decimal) => 142 (decimal)
* Group 2: 10100101 (decimal) => 165 (decimal)
* Group 3: 01010010 (decimal) => 82 (decimal)

**Binary Output:**

The original binary data that was encoded as "RFK" in Base64 is 142 165 82 (in decimal), which can be represented in hexadecimal as 8E A5 52.

It's important to remember that Base64 decoding is not a form of encryption, and it's a reversible process. The original binary data can be easily recovered from the Base64-encoded string. Base64 is primarily used to represent binary data in a text-based format, making it suitable for transmission over protocols that only support text, such as email or URLs.

Here are a few additional points about Base64:

**1. Padding:**

Base64 encoding may include padding characters ('=') at the end of the encoded string. These padding characters are used to ensure that the length of the encoded string is a multiple of 4 characters. They help indicate how many bytes were in the original binary data, especially when the data's length is not a multiple of 3 bytes.

**2. Use Cases:**

Base64 encoding is widely used in various applications:

* + Embedding binary data in XML, JSON, or other text-based formats.
  + Sending binary data over email attachments.
  + Representing images in HTML or CSS code.
  + Encoding data for URL parameters.
  + Storing binary data in configuration files.

**3. Efficiency:**

Base64 encoding expands the size of the data by approximately 33%. This is because every 3 bytes (24 bits) of data are represented by 4 Base64 characters (6 bits each). While the encoded data is larger, it's still much more efficient than representing binary data in its raw binary form when dealing with text-based protocols.

**4. Security:**

Base64 encoding is not a secure method of protecting sensitive information. It's not a form of encryption, and anyone with access to the encoded data can easily decode it. If security is a concern, encryption should be used instead of Base64 encoding.

**5. Character Set:**

The specific set of 64 characters used in Base64 encoding can vary slightly depending on the context. The standard character set includes uppercase letters, lowercase letters, numbers, and the characters '+', '/', and '=' for padding. Some variations replace the '+' and '/' characters with different characters to make the encoded data more URL-friendly.

**6. Decoding Libraries:**

Most programming languages provide libraries or functions to easily encode and decode Base64 data. These libraries handle the conversion process, making it simple for developers to work with Base64-encoded data.

**7. Non-Text Data:**

Base64-encoded data is treated as text, which means it can be safely transmitted over text-based protocols. However, keep in mind that the resulting encoded string can still contain characters that might need to be URL-encoded when used in URLs or other contexts.

**8. Human Readable:**

Base64-encoded data is human-readable but not human-understandable. While the encoded data appears as a series of characters, its original meaning is not apparent without decoding.

**Base 64 Encode Algorithm**

The Base64 encode algorithm converts any data into plain text. Technically, it can be said that it converts eight-bit bytes into six-bit bytes. To understand how the encoding algorithm works, check the example below that describes step by step how to manually encode strings to Base64.

For example, you have the “ABC” string and want to convert it to Base64:

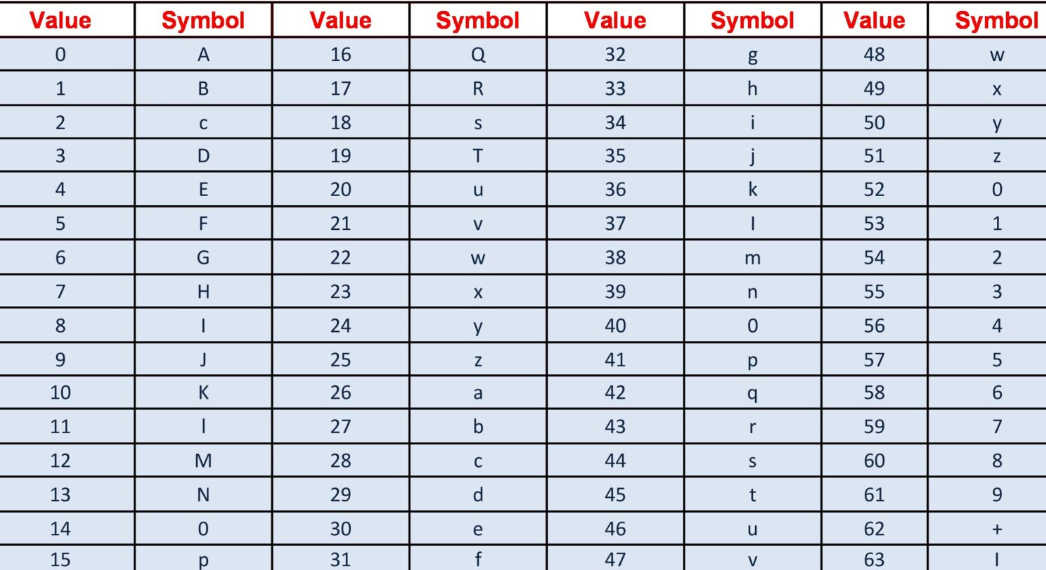
1. First, you need to split the string letter by letter. Thus, you got 3 groups:
   * A
   * B
   * C
2. Next you need to convert each group to binary. To do this, for each letter you need to find the corresponding **binary** value in the ASCII table. Thus, now you have 3 groups of ones and zeros:
   * 01000001
   * 01000010
   * 01000011
3. Now concatenate all the binary values together (that is, glue all the groups along and make sure you get a total of 24 characters):  
   010000010100001001000011
4. Then, divide the resulting string into groups so that each one has 6 characters (if the last group has less than 6 characters, you need to fill it with zeros until it reaches the desired length). Well and good, now you have 4 groups:
   * 010000
   * 010100
   * 001001
   * 000011
5. At this step you have to *convert* six-bit bytes into eight-bit bytes. To do this, prepend the prefix “00” (two zeros) in front of each group:
   * 00010000
   * 00010100
   * 00001001
   * 00000011
6. There you have to convert each group from binary to decimal by finding its corresponding **decimal** value in the ASCII table. If you did everything right, each group will be transformed into its integer number as follows:
   * 16
   * 20
   * 9
   * 3
7. Integer numbers obtained in the previous step are called “Base64 indices”. They are easy to remember because it is a zero-based numbering, where each index corresponds to a Latin letter. It starts with the letter “A” in alphabetical order (i.e., A=0, B=1, C=2, D=3, and so on):
   * Q
   * U
   * J
   * D
8. The final chord, concatenate all letters to get the Base64 string:  
   QUJD

**Base64 Decode Algorithm**

The Base64 decode algorithm converts plain text into original data. Technically, it can be said that it converts six-bit bytes into eight-bit bytes. To understand how the decoding algorithm works, check the example below that describes step by step how to manually decode strings from Base64.

For example, you have the “QUJD” string and want to decode it from Base64:

1. First, you need to split the string letter by letter. Thus, you got 4 groups:
   * Q
   * U
   * J
   * D
2. Each group (letter) is a Base64 character that has its own index, and now your task is to convert groups to indices. To do this, by mapping values from the Base 64 Characters Table replace each character with its index (if you cannot find an index for a specific group, just discard it). All in all, you should get the following indices:
   * 16
   * 20
   * 9
   * 3
3. At this step you should convert each group from decimal to binary. So, find corresponding decimal values in the ASCII table and make sure you get the following binary values:
   * 00010000
   * 00010100
   * 00001001
   * 00000011
4. Now remove the prefix “00” (two zeros) in front of each group:
   * 010000
   * 010100
   * 001001
   * 000011
5. There you have a simple concatenation of previous groups (that is, glue all the binary values together and get a 24-character string):  
   010000010100001001000011
6. Then, divide the resulting string into groups so that each one has 8 characters (if the last group has less than 8 characters, you must discard it). Now you have 3 groups of eight-bit bytes:
   * 01000001
   * 01000010
   * 01000011
7. Once again using the ASCII table, convert all binary values into their ASCII characters:
   * A
   * B
   * C
8. The final chord, concatenate all ASCII characters to get the result string:  
   ABC



**5.RSA ALGORITHM**

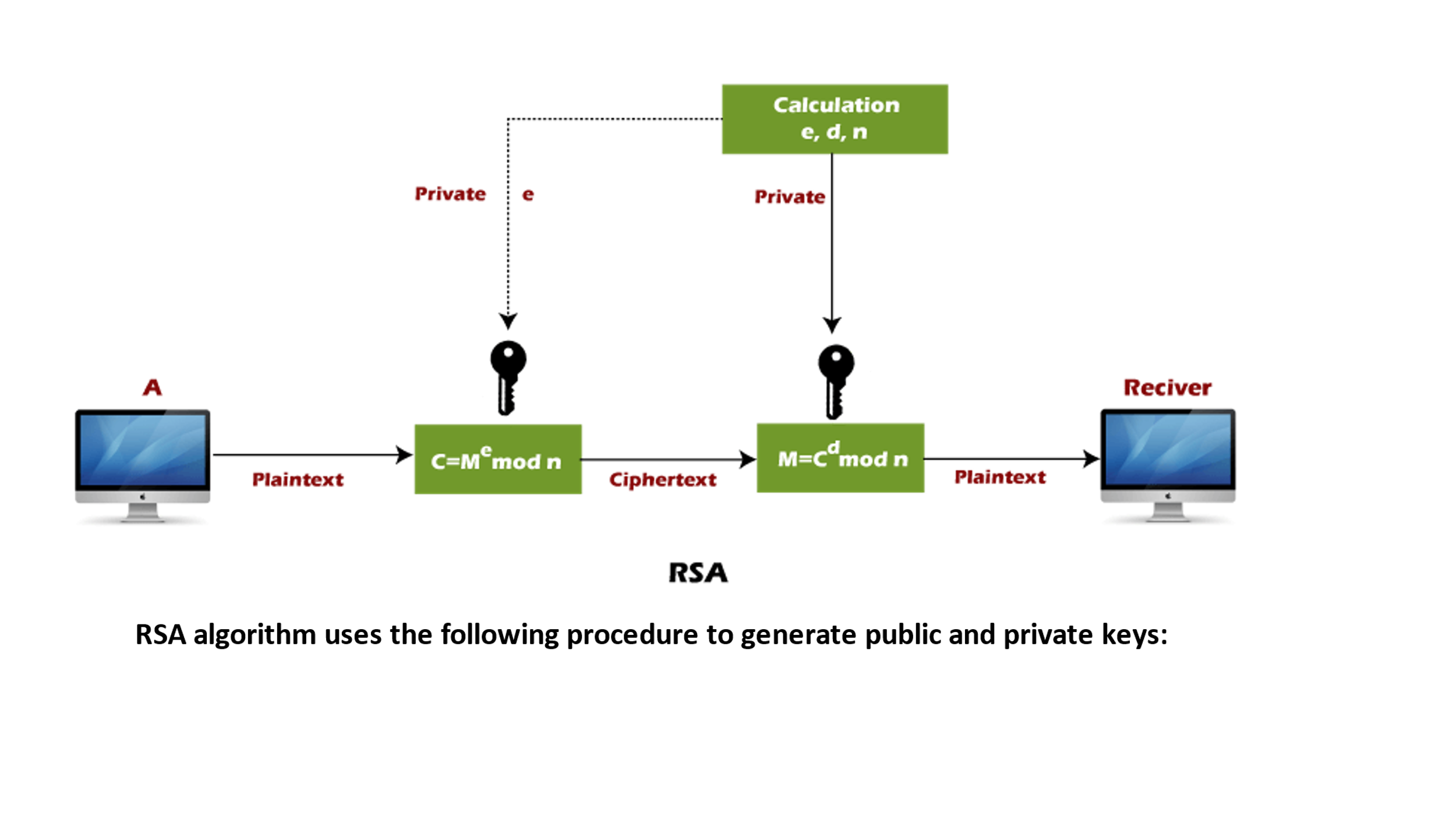
**RSA algorithm** is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and the Private key is kept private.

**The idea!** The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is a multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task.

**An example of asymmetric cryptography:**

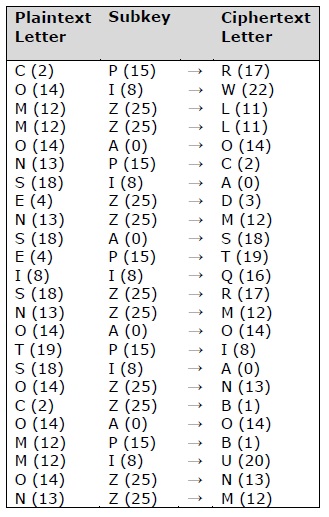
1. A client (for example browser) sends its public key to the server and requests some data.
2. The server encrypts the data using the client’s public key and sends the encrypted data.
3. The client receives this data and decrypts it.

Since this is asymmetric, nobody else except the browser can decrypt the data even if a third party has the public key of the browser.

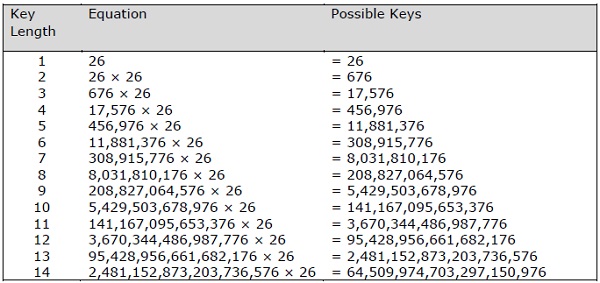


Vignere Cipher

Vignere cipher uses more than one set of substitutions, and hence it is also referred as **polyalphabetic cipher**. Vignere Cipher will use a letter key instead of a numeric key representation: Letter A will be used for key 0, letter B for key 1 and so on. Numbers of the letters before and after encryption process is shown below −



The possible combination of number of possible keys based on Vignere key length is given as follows, which gives the result of how secure is Vignere Cipher Algorithm −

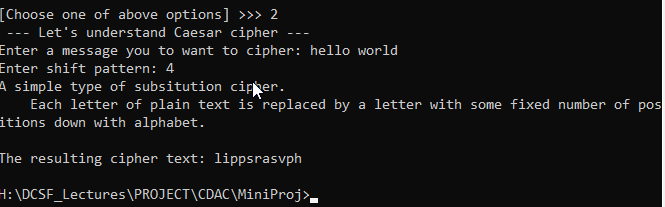
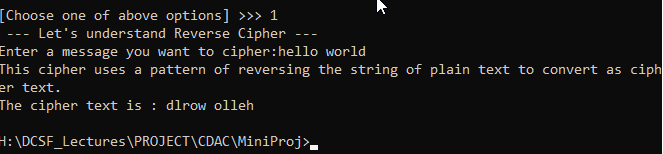
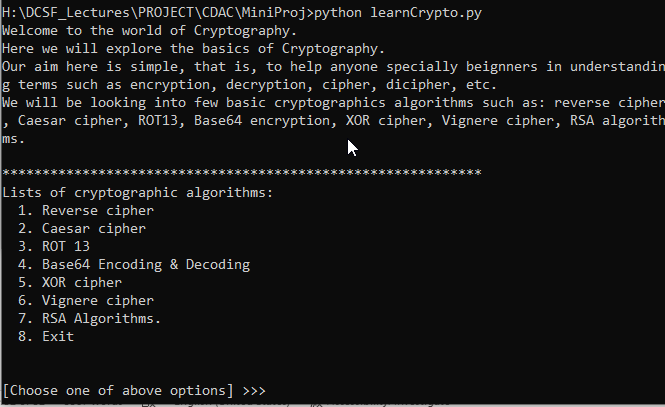
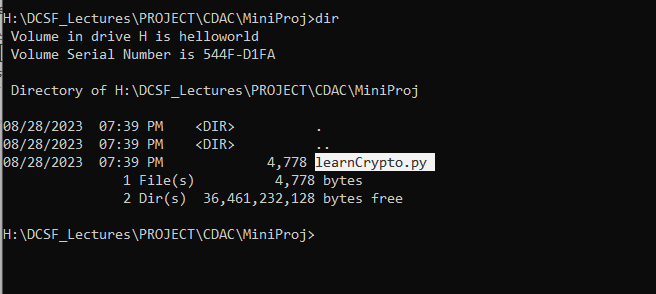


**Vignere Tableau**

The tableau used for Vignere cipher is as shown below −



1. Figures



1. References

8.1 <https://www.tutorialspoint.com/cryptography_with_python/index.htm>

8.2 <https://www.python.org/>

8.3 <https://pypi.org/project/fernet-encrypt/>

8.4 <https://pypi.org/project/cryptography/>

8.5 <https://docs.python.org/3/library/crypto.html>

8.6 <https://blog.logrocket.com/implementing-cryptography-python/>

8.7 <https://en.wikipedia.org/wiki/XOR_cipher>