Project Report

on

Shrive Crypt
Dynamic Website Using Web Development and Cryptography

Submitted in partial fulfillment of the

requirement for the award of the degree of

B. Tech CSE (CNCS)

GALGOTIAS
UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Dr. A. Suresh Kumar
Assistant Professor

Submitted By

Gurushabad Singh Saluja
18021140056
Adil Abdul Rahman
18021011336

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING /
DEPARTMENT OF COMPUTERAPPLICATION
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
DECEMBER, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"SHRIVE CRYPT"** in partial fulfillment of the requirements for the award of the B tech CSE (CNCS) submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of July 2021 to December 2021, under the supervision of Dr. A. Suresh Kumar, Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

Gurushabad Singh Saluja 18021140056

Adil Abdul Rahman 18021011336

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. A. Suresh Kumar

Assistant Professor

CERTIFICATE

The Final Project Viva-Voce examination of Gurushabad Singh Saluja (18021140056), Adil Abdul Rahman (18021011336) has been held on ________________ and his/her work is recommended for the award of BTECH CSE (CNCS).

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021

Place: Greater Noida

Abstract

In this modern era, the use of cryptography and web dev everywhere like, E-commerce is perfect example of how cryptography and web dev go hand in hand whether it is about purchasing something or about online transactions, we use it to securely send passwords over vast networks for online purchases on various websites. Bank servers and e-mail clients save your passwords using cryptography as well. Cryptography is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents. Here, data is encrypted using a secret key, and then both the encoded message and secret key are sent to the recipient for decryption. In this project we are merging these two fields cryptography and web development and as product we get a dynamic website through which a user can choose various encryption/decryption techniques to encrypt/decrypt various inputs. It will be helpful in field in such as education, secret services, messaging applications, defense sector etc.

In today's world everything is stored digitally including sensitive information like our passwords, addresses, phone no. etc. People are unaware of the security threats that they can face when the data gets leaked or gets into the wrong hands. Most apps these days require access to storage, contacts and location and without it won't open. So, it becomes a necessity as a user to encrypt as much data as possible. Most of us are usually not aware about the security threat and most of the existing website that provide user to manually encrypt some plain text given by user are very confusing and doesn't provide information regarding technique used and they usually provide max one to two techniques and lacks variety and flexibility which user needs.

Shrive Crypt ensures that it will contain all the major encryption and decryption technique all in one website. The user can use our website to encrypt his or her message for example while sending bank information to family members or passwords or any sensitive information which might be risky to send as apps can access our data and the receiving user can decrypt data using our website so that he can get the original message.

Our website provides flexibility, by doing multiple tasks at one place. Provides ease of access to user and provides sense of security that their data is not being used illegally. In future we aim to implement the feature which will help us to perform steganography and Morse code. Morse code will have audio output also.

Table of Contents

Title **Page No.**

List of Figures

Acronyms

B.Tech.	Bachelor of Technology
M.Tech.	Master of Technology
BCA	Bachelor of Computer Applications
MCA	Master of Computer Applications
B.Sc. (CS)	Bachelor of Science in Computer Science
M.Sc. (CS)	Master of Science in Computer Science
SCSE	School of Computing Science and Engineering

CHAPTER-1

Introduction

Does increased security provide comfort to paranoid people? Or does security provide some very basic protections that we are naive to believe that we don't need? During this time when the Internet provides essential communication between literally billions of people and is used as a tool for commerce, social interaction, and the exchange of an increasing amount of personal information, security has become a tremendously important issue for every user to deal with.

There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting health care information. One essential aspect for secure communications is that of cryptography. But it is important to note that while cryptography is necessary for secure communications, it is not by itself sufficient. People are unaware of the security threats that they can face when the data gets leaked or gets into the wrong hands. Most apps these days require access to storage, contacts and location and without it won't open. So, it becomes a necessity as a user to encrypt as much data as possible. Most of us are usually not aware about the security threat and most of the existing website that provide user to manually encrypt some plain text given by user are very confusing and doesn't provide information regarding technique used and they usually provide max one to two techniques and lacks variety and flexibility which user needs.

Shrive Crypt ensures to provide the variety and flexibility which users crave for along with an attractive user interface and seamless user experience, we are trying to provide multiple techniques at one place to facilitate user with different variety so that one doesn't have to seek any other website.

To ensure these functionalities we will be using different tools such as HTML and CSS for frontend, JS and DOM for handling and handle inputs and carry out the process of encryption and decryption, and GitHub for hosting our website and for the cryptographic part we will using cryptographic Algorithm such as Creaser cipher, IDEA Algorithm, SHA-512, RSA, etc.

1.2 Problem Formulation

These days everything is put away carefully including delicate data like our passwords, addresses, telephone no. and so forth Individuals are unaware of the security dangers that they can confront when the information gets spilled or gets into some unacceptable hands. Most applications these days expect admittance to capacity, contacts and area and without it will not open. So, it turns into a need as a client to scramble however much information as could reasonably be expected. A large portion of us is normally not mindful with regards to the security danger and the vast majority of the current site that give client to physically encode some plain text given by client are exceptionally befuddling and doesn't give data in regards to strategy utilized and they as a rule give max one to two procedures and needs assortment and adaptability which client needs. There are numerous angles to security and numerous applications, going from secure business and installments to private interchanges and ensuring medical care data. One fundamental perspective for secure interchanges is that of cryptography. In any case, note that while cryptography is vital for secure interchanges, it isn't without anyone else adequate. Individuals are unaware of the security dangers that they can confront when the information gets spilled or gets into some unacceptable hands. Most applications these days expect admittance to capacity, contacts and area and without it will not open. So, it turns into a need as a client to encode however much information as could reasonably be expected.

Shrive Crypt ensures to provide the variety and flexibility which users crave for along with an attractive user interface and seamless user experience, we are trying to provide multiple techniques at one place to facilitate user with different variety so that one doesn’t have to seek any other website.

1.2.1 Tool and Technology Used

since we are making a cryptographic website, we will be using different type of tools such as HTML and CSS for frontend, JS and DOM for handling and handle inputs and carry out the process of encryption and decryption, and GitHub for hosting our website and for the cryptographic part we will using cryptographic Algorithm such as Creaser cipher, IDEA Algorithm, SHA-512, RSA, etc.

1. Visual Studio Code- features a lightning-fast source code editor, perfect for day-to-day use. With support for hundreds of languages, VS Code helps you be instantly productive with syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets, and more. Intuitive keyboard shortcuts, easy customization and community-contributed keyboard shortcut mappings let you navigate your code with ease.

For serious coding, you'll often benefit from tools with more code understanding than just blocks of text. Visual Studio Code includes built-in support for IntelliSense code completion, rich semantic code understanding and navigation, and code refactoring.

2. HTML/CSS- HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, CSS the (visual and aural) layout, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications.

3. Encryption/Decryption techniques-

Encryption is the process of translating plain text data (plaintext) into something that appears to be random and meaningless (ciphertext). Decryption is the process of converting ciphertext back to plaintext.

The goal of every encryption algorithm is to make it as difficult as possible to decrypt the generated ciphertext without using the key. If a really good encryption algorithm is used, there is no technique significantly better than methodically trying every possible key. For such an algorithm, the longer the key, the more difficult it is to decrypt a piece of ciphertext without possessing the key.

CHAPTER-2

Literature Survey/Project Design

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 fueled the natural need of people to communicate secretly with selective recipient which in turn ensured the continuous evolution of cryptography as well.

Currently many websites similar to Shrive Crypt exists but the problem with them is that they lack variety and flexibility along with poor user interface which confuses user about the current technique being used and one has to hassle to much for a simple encryption/decryption.

These types of websites usually provide max of one to two crypto Techniques to perform and the outcome is also not completely reliable but the most tiring thing which user suffers is user experience the lag in the website turns off enthusiasm of any user who wants to learn about cryptography.

On the other hand, Shrive Crypt tries to provide most possible variety and flexibility along with attractive user interface which peeks the interest of the user and clearly explains about the current running technologies, it maintains the task of various websites at one place and the outcome is completely reliable with a the most exhilarating user experience which is so fluid user doesn’t want to exit it.

Project Design

We are making a website which contains many encryption/decryption algorithms working separately and independently such that outcome of one doesn’t affect other below figure1.0 shows overall working of website, in which user is free to give any type of input and he/she shall receive the output accordingly.

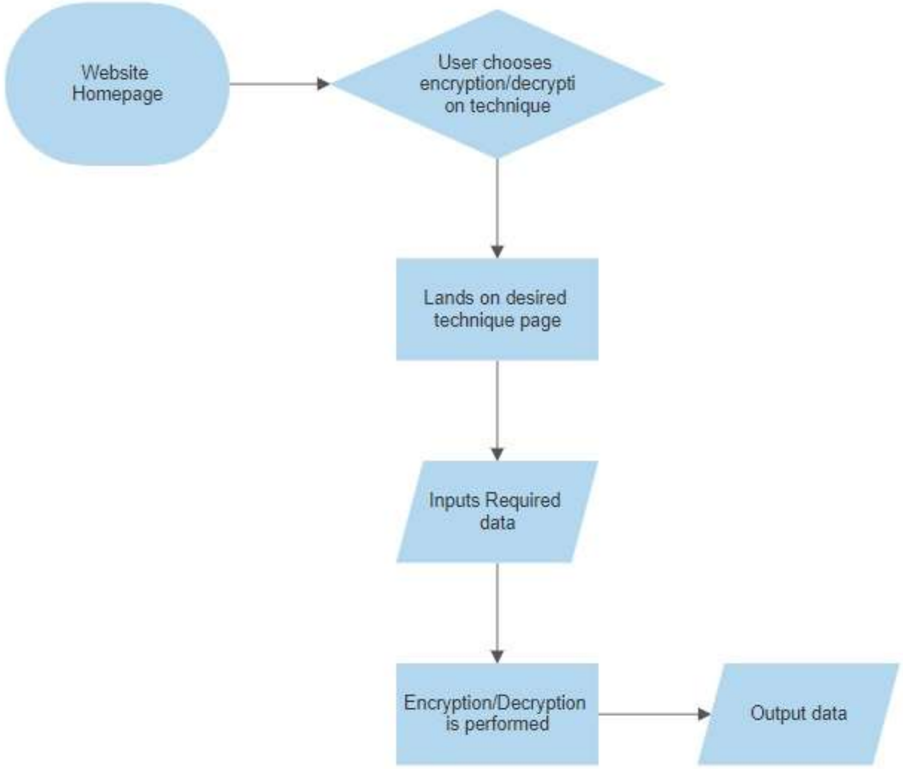

Website Homepage
User chooses encryption/decryption technique
Lands on desired technique page
Inputs Required data
Encryption/Decryption is performed
Output data

Fig.1.0 Work Flow/Project Design

and for the cryptographic part we will using cryptographic Algorithm such as Creaser cipher, IDEA Algorithm, SHA-512, RSA

CHAPTER-3

Functionality/Working of Project

we are using different tools such as HTML and CSS for frontend, JS and DOM for handling and handle inputs and carry out the process of encryption and decryption, and GitHub for hosting our website and for the cryptographic part we will using cryptographic Algorithm such as Caeser cipher, SHA-512, RSA, base 64, vigenere cipher, rail fence cipher, play fair cipher etc.

Base64

Base64 is a binary to ASCII encoding scheme. It is designed as a way to transfer binary data reliably across channels that have limited support for different content types.
A base64 encoded string looks like this:

V2hhdCBoYXBwZW5zIHdoZW4geW91IGJhc2U2NCgpPw==

Base64 characters only use the same 64 characters that are present in most character sets. They are:

- Upper case alphabet characters A-Z.
- Lower case alphabet characters a-z.
- Number characters 0–9.
- And finally, characters + and /.
- The = character is used for padding.

These characters are generally implemented by most character sets and are not often used as controlled characters in Internet protocols. So when you encode content with base64, you can be fairly confident that your data is going to arrive uncorrupted.

Whereas when you transfer your data in their original, “bits and bytes” state, the data might be screwed up due to protocols misinterpreting special characters.

What is it used for?

The original use case for base64 was simply as a safe way to transmit data across machines. Overtime, base64 has been integrated into the implementation of certain core Internet technologies such as encryption and file embedding.

Data transmission: Base64 can simply be used as a way to transfer and store data without the risk of data corruption. It is often used to transmit JSON data and cookie information for a user.

File embedding: Base64 can be used to embed files within scripts and webpages, so as to avoid depending on external files. Email attachments are also often sent this way.

Data obfuscation: Base64 can be used to obfuscate data since the resulting text is not human readable. However, this should not be used as a security mechanism as the encoding is easily reversible.

Data hashing: Data hashing schemes such as SHA and MD5 often produce results that are not readable or transmittable. Therefore, hashes are almost always base64 encoded so that they could be easily displayed and used for file integrity checks.

Cryptography: Similarly, encrypted data often contain sequences of bytes that are not easily transmitted or stored. When encrypted data needs to be stored in a database or sent over the Internet, base64 is often used. In addition to cyphertext, public key certificates and other encryption keys are also commonly stored in base64 format.

Other transfer-safe encoding schemes

But why is base64 so widely used? Aren’t there other transfer-safe encoding schemes, like Hex and Decimal? This is because base64 is very compact compared to other transfer-safe encoding schemes.

For example, Hex encoding encodes each byte as a pair of Hex characters. This means that each byte of data would become two bytes after encoding.

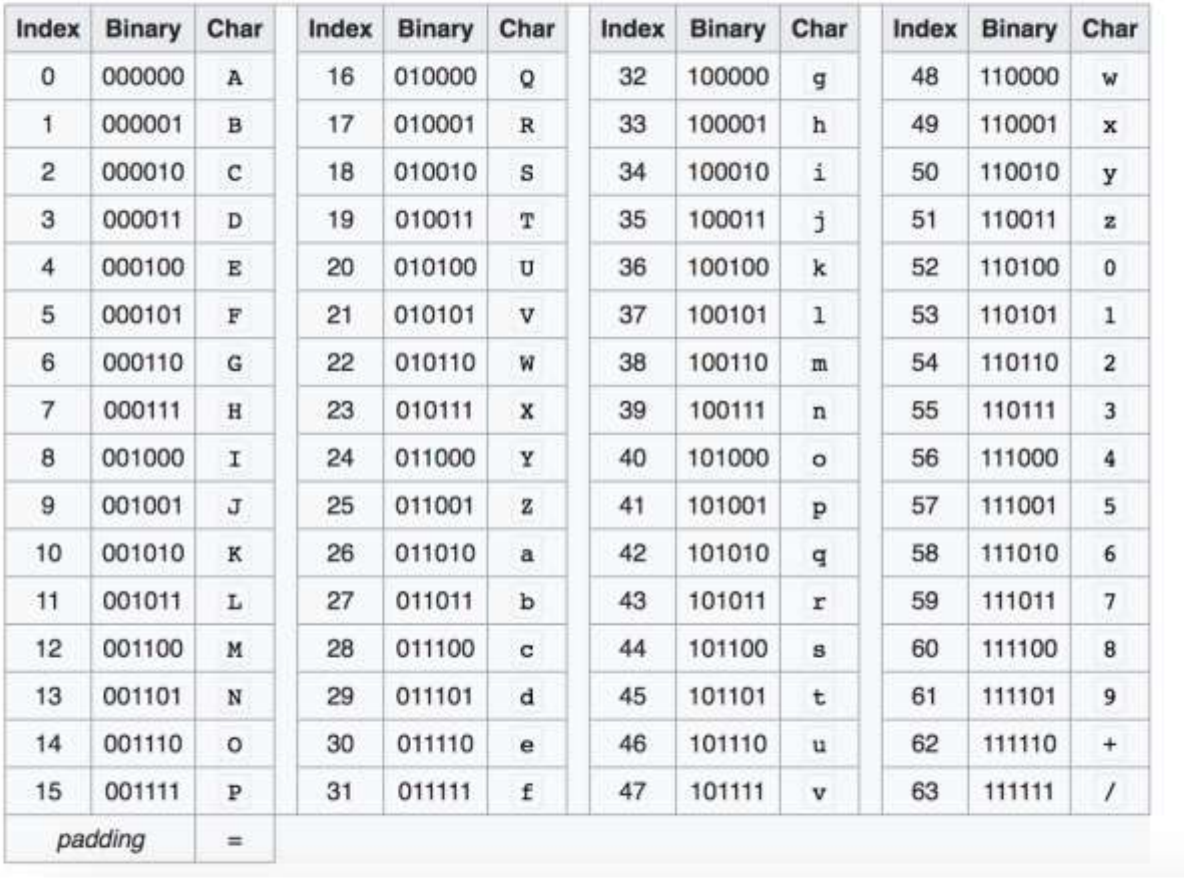
Whereas Decimal is even less efficient: each byte of data would be represented as three numbers, which means that each byte of unencoded data would take up three bytes as encoded data.

Base64 maps every three bytes of data into four bytes of encoded data. This means that the data would only bloat 4/3 times once it’s base64 encoded.

How does base64 work?

Base64 encoding converts every three bytes of data (three bytes is 3*8=24 bits) into four base64 characters.

Each six-bit sequence is uniquely mapped to one of the 64 characters used:



Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
padding		=									

For example, the text **Hi!** has the binary representation of

```
01001000 01101001 00100001
```

Which makes up a total of three bytes (24 bits). Base64 encoding will divide the binary data up to six-bit chunks and map them to a base64 character according to the table above.

```
010010 | 000110 | 100100 | 100001
S        G        k        h
```

Therefore, the base64 encoding of **Hi!** is **SGkh**.

Padding

When the number of characters to be encoded does not come with a multiple of six bits, zeros will be used to complete the last six-bit sequence.

For example, the text **Hi** has the binary representation of:

```
01001000 01101001
```

The binary representation only contains 16 bits, which is not divisible by six.

```
010010 | 000110 | 1001
```

To encode the text properly, base64 will add zeros to the end of the bit sequence.

```
010010 | 000110 | 100100 |
S        G        k        =
```

So the base64 representation of **Hi** is **SGk=.** (The = padding character is added so that the last encoded block will have four base64 characters.)

Decoding base64

To decode base64, you simply have to reverse the above operation:

First, you remove any padding characters from the end of the encoded string.

Then, you translate each base64 character back to their six-bit binary representation.

Finally, you divide the bits into byte-sized (eight-bit) chunks and translate the data back to its original format.

Other base64 implementations

Apart from the standard base64 encoding scheme that was mentioned above, there are many different implementations of base64 for specific use cases. For example:

Base64 for filenames uses **“-”** in place of **“/”.** This is to work around the fact that Unix and Windows filenames cannot contain the character “/” since it’s used in file paths.

Base64 for URLs uses **“-” and “_”** in the place of **“+” and “/”** and omits padding the encoded string with **“=”**. This is because URLs require special characters like +, / and = to be URL encoded into **%2b, %2f and %3d**, which makes the encoded string unnecessarily long.

SHA-512:

This is intended to give you a basic understanding about what actually happens during the execution of a hashing algorithm. I’ve used the SHA-512 algorithm in order to help explain the inner working of a hash function.

SHA-512 is a hashing algorithm that performs a hashing function on some data given to it. Hashing algorithms are used in many things such as internet security, digital certificates and even blockchains. Since hashing algorithms play such a vital role in digital security and cryptography, this is an easy-to-understand walkthrough, with some basic and simple maths along with some diagrams, for a hashing algorithm called SHA-512. It’s part of a group of hashing algorithms called SHA-2 which includes SHA-256 as well which is used in the bitcoin blockchain for hashing.

Before starting with an explanation of SHA-512, I think it would be useful to have a basic idea of what a hashing function’s features are.

Hashing Functions

Hashing functions take some data as input and produce an output (called hash digest) of fixed length for that input data. This output should, however, satisfy some conditions to be useful.

- Uniform distribution: Since the length of the output hash digest is of a fixed length and the input size may vary, it is apparent that there are going to be some output values that can be obtained for different input values. Even though this is the case, the hash function should be such that for any input value, each possible output value should be equally likely. That is to say that every possible output has the same likelihood to be produced for any given input value.

- Fixed Length: This is should be quite self-explanatory. The output values should all be of a fixed length. So, for example, a hashing function could have an output size of 20 characters or 12 characters, etc. SHA-512 has an output size of 512 bits.
- Collision resistance: Simply speaking, this means that there aren't any or rather it is not feasible to find two distinct inputs to the hash function that result in the same output (hash digest).

SHA-512 does its work in a few stages. These stages go as follows:

1. Input formatting
2. Hash buffer initialization
3. Message Processing
4. Output

1.Input Formatting:

SHA-512 can't actually hash a message input of any size, i.e. it has an input size limit. This limit is imposed by its very structure as you may see further on. The entire formatted mesage has basically three parts: the original message, padding bits, size of original message. And this should all have a combined size of a whole multiple of 1024 bits. This is because the formatted message will be processed as blocks of 1024 bits each, so each bock should have 1024 bits to work with.

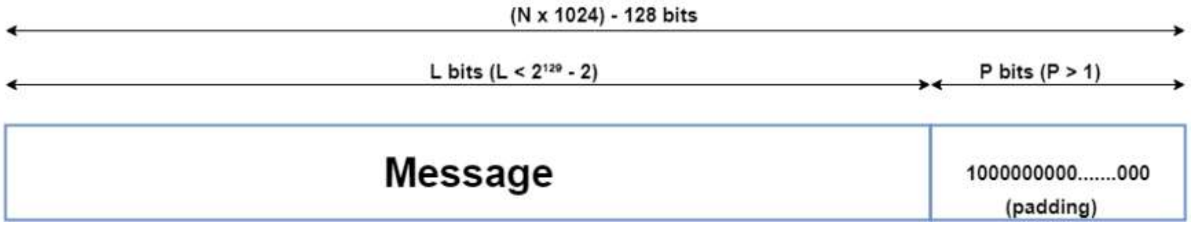
L bits
Message

Original message

Padding bits

The input message is taken and some padding bits are appended to it in order to get it to the desired length. The bits that are used for padding are simply ‘0’ bits with a leading ‘1’ (100000…000). Also, according to the algorithm, padding needs to be done, even if it is by one bit. So a single padding bit would only be a ‘1’.

The total size should be equal to 128 bits short of a multiple of 1024 since the goal is to have the formatted message size as a multiple of 1024 bits (N x 1024).

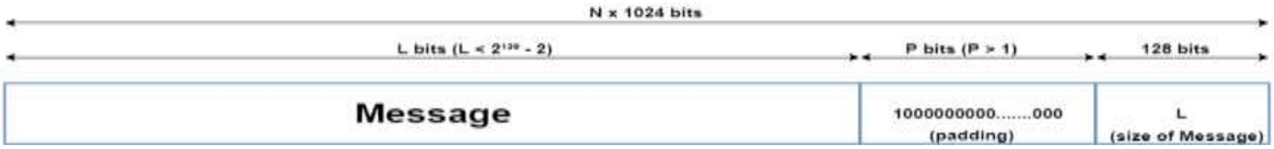


Message with padding

Padding size

After this, the size of the original message given to the algorithm is appended. This size value needs to be represented in 128 bits and is the only reason that the SHA-512 has a limitation for its input message.

Since the size of the original message needs to be represented in 128 bits and the largest number that can be represented using 128 bits is (2^{128}-1), the message size can be at most (2^{128}-1) bits; and also taking into consideration the necessary single padding bit, the maximum size for the original message would then be (2^{128}-2). Even though this limit exists, it doesn’t actually cause a problem since the actual limit is so high (2^{128}-2 = 340,282,366,920,938,463,463,374,607,431,768,211,454 bits).



Message with padding and size

After padding bits and the size of the message have been appended, we are left with the completely formatted input for the SHA-512 algorithm.

N x 1024 bits
Formatted Input

Formatted Message

2. Hash buffer initialization:

The algorithm works in a way where it processes each block of 1024 bits from the message using the result from the previous block. Now, this poses a problem for the first 1024 bit block which can’t use the result from any previous processing. This problem can be solved by using a default value to be used for the first block in order to start off the process. (Have a look at the second-last diagram).

Since each intermediate result needs to be used in processing the next block, it needs to be stored somewhere for later use. This would be done by the hash buffer, this would also then hold the final hash digest of the entire processing phase of SHA-512 as the last of these ‘intermediate’ results.

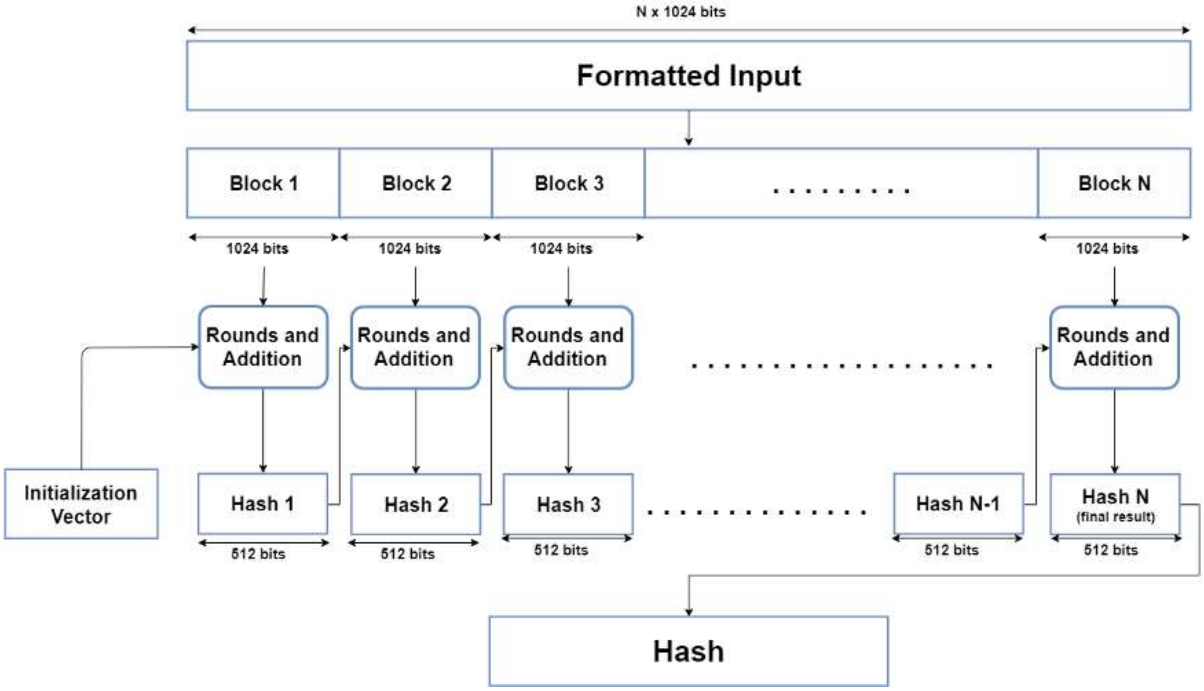
So, the default values used for starting off the chain processing of each 1024 bit block are also stored into the hash buffer at the start of processing. The actual value used is of little consequence, but for those interested, the values used are obtained by taking the first 64 bits of the fractional parts of the square roots of the first 8 prime numbers (2,3,5,7,11,13,17,19). These values are called the Initial Vectors (IV).

Why 8 prime numbers instead of 9? Because the hash buffer actually consists of 8 subparts (registers) for storing them.

3. Message Processing:

Message processing is done upon the formatted input by taking one block of 1024 bits at a time. The actual processing takes place by using two things: The 1024 bit block, and the result from the previous processing.

This part of the SHA-512 algorithm consists of several ‘Rounds’ and an addition operation.



William Stallings, Cryptography and Network Security — Principles and Practise (Seventh Edition) referred for diagram

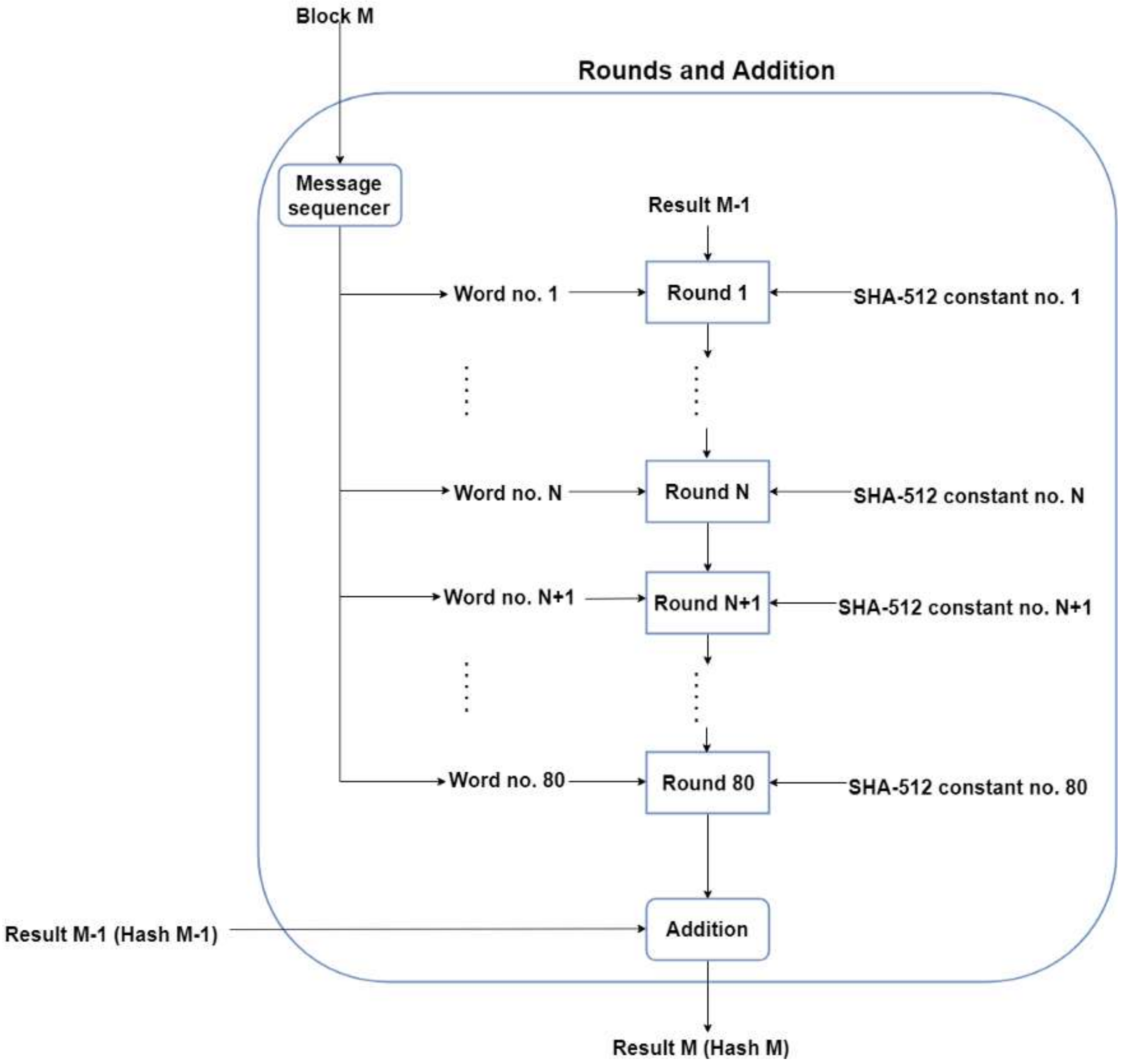
So, the Message block (1024 bit) is expanded out into ‘Words’ using a ‘message sequencer’. Eighty Words to be precise, each of them having a size of 64 bits.

Rounds

The main part of the message processing phase may be considered to be the Rounds. Each round takes 3 things: one Word, the output of the previous Round, and a SHA-512 constant. The first Round doesn’t have a previous Round whose output it can use, so it uses the final output from the previous message processing phase for the previous block of 1024 bits. For the first Round of the first block (1024 bits) of the formatted input, the Initial Vector (IV) is used.

SHA-512 constants are predetermined values, each of whom is used for each Round in the message processing phase. Again, these aren’t very important, but for those interested, they are the first 64 bits from the fractional part of the cube roots of the first 80 prime numbers.Why 80? Because there are 80 Rounds and each of them needs one of these constants.

Once the Round function takes these 3 things, it processes them and gives an output of 512 bits. This is repeated for 80 Rounds. After the 80th Round, its output is simply added to the result of the previous message processing phase to get the final result for this iteration of message processing.



4. Output:

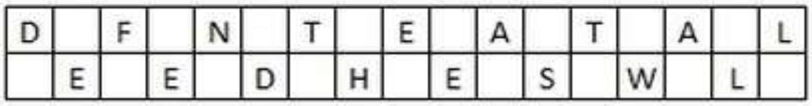
After every block of 1024 bits goes through the message processing phase, i.e. the last iteration of the phase, we get the final 512 bit Hash value of our original message. So, the intermediate results are all used from each block for processing the next block. And when the final 1024 bit block has finished being processed, we have with us the final result of the SHA-512 algorithm for our original message.

Thus, we obtain the final hash value from our original message. The SHA-512 is part of a group of hashing algorithms that are very similar in how they work, called SHA-2. Algorithms such as SHA-256 and SHA-384 are a part of this group alongside SHA-512. SHA-256 is also used in the Bitcoin blockchain as the designated hash function.

Rail Fence

The rail fence cipher is an easy to apply transposition cipher that jumbles up the order of the letters of a message in a quick convenient way. It also has the security of a key to make it a little bit harder to break.

The Rail Fence cipher works by writing your message on alternate lines across the page, and then reading off each line in turn. For example, the plaintext "defend the east wall" is written as shown below, with all spaces removed.



D		F		N		T		E		A		T		A		L
	E		E		D		H		E		S		W		L	

The simplest Rail Fence Cipher, where each letter is written in a zigzag pattern across the page.

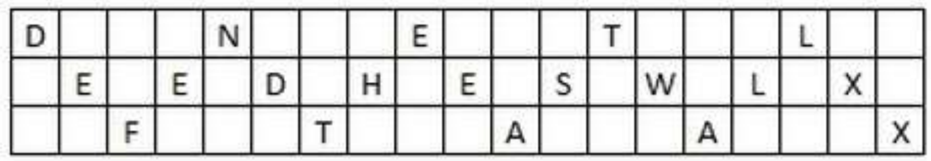
The simplest Rail Fence Cipher, where each letter is written in a zigzag pattern across the page.

The ciphertext is then read off by writing the top row first, followed by the bottom row, to get "DFNTEATALEEDHESWL".

Encryption

To encrypt a message using the Rail Fence Cipher, you have to write your message in zigzag lines across the page, and then read off each row. Firstly, you need to have a key, which for this cipher is the number of rows you are going to have. You then start writing the letters of the plaintext diagonally down to the right until you reach the number of rows specified by the key. You then bounce back up diagonally until you hit the first row again. This continues until the end of the plaintext.

For the plaintext we used above, "defend the east wall", with a key of 3, we get the encryption process shown below.



D				N				E				T				L		
	E		E		D		H		E		S		W		L		X	
		F				T				A				A				X

The Rail Fence Cipher with a key of 3. Notice the nulls added at the end of the message to make it the right length.

The Rail Fence Cipher with a key of 3. Notice the nulls added at the end of the message to make it the right length.

Note that at the end of the message we have inserted two "X"s. These are called nulls, and act as placeholders. We do this to make the message fit neatly in to the grid (so that there are the same number of letters on the top row, as on the bottom row. Although not necessary, it makes the decryption process a lot easier if the message has this layout.

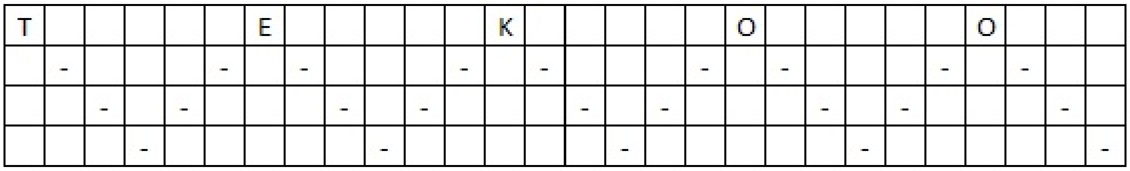
The ciphertext is read off row by row to get "DNETLEEDHESWLXFTAAX".

Decryption

The decryption process for the Rail Fence Cipher involves reconstructing the diagonal grid used to encrypt the message. We start writing the message, but leaving a dash in place of the spaces yet to be occupied. Gradually, you can replace all the dashes with the corresponding letters, and read off the plaintext from the table.

We start by making a grid with as many rows as the key is, and as many columns as the length of the ciphertext. We then place the first letter in the top left square, and dashes diagonally downwards where the letters will be. When we get back to the top row, we place the next letter in the ciphertext. Continue like this across the row, and start the next row when you reach the end.

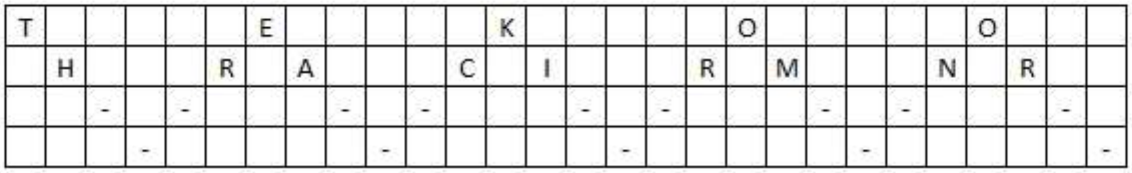
For example, if you receive the ciphertext "TEKOOHRACIRMNREATANFTETYTGHH", encrypted with a key of 4, you start by placing the "T" in the first square. You then dash the diagonal down spaces until you get back to the top row, and place the "E" here. Continuing to fill the top row you get the pattern below.



T						E						K						O						O			
	-				-		-				-		-				-		-				-		-		
		-		-				-		-				-		-				-		-				-	
			-						-						-						-						-

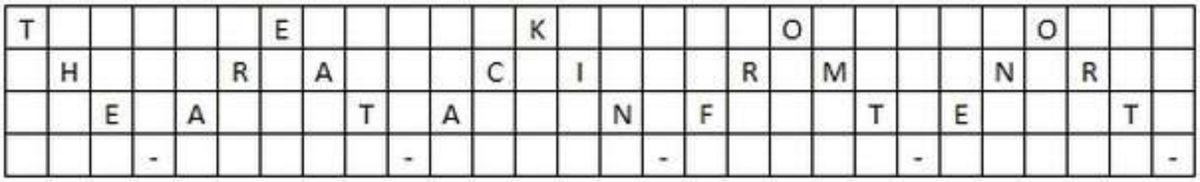
The first row of the decryption process for the Rail Fence Cipher. We have a table with 4 rows because the key is 4, and 28 columns as the ciphertext has length 28.

Continuing this row-by-row, we get the successive stages shown below.



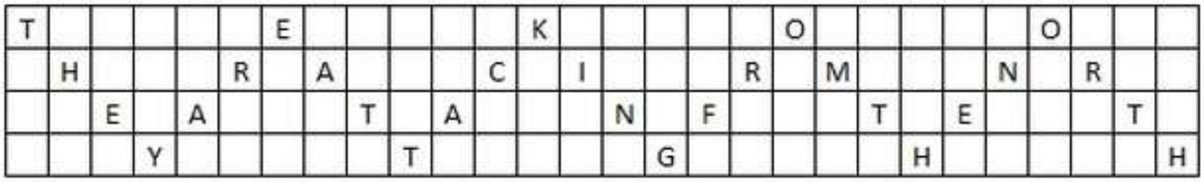
T						E						K						O						O			
	H				R		A				C		I				R		M				N		R		
		-		-				-		-				-		-				-		-				-	
			-						-						-						-						-

The second stage in the decryption process.



T						E						K						O						O			
	H				R		A				C		I				R		M				N		R		
		E		A				T		A				N		F				T		E				T	
			-						-						-						-						-

The third stage in the decryption process.



T						E						K						O						O			
	H				R		A				C		I				R		M				N		R		
		E		A				T		A				N		F				T		E				T	
			Y						T						G						H						H

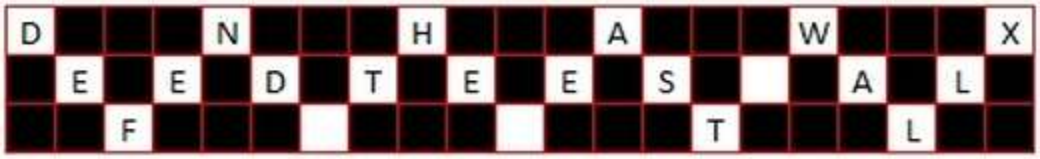
The fourth and final stage in the decryption process.

From this we can now read the plaintext off following the diagonals to get "they are attacking from the north".

Discussion

The Rail Fence Cipher is a very easy to apply transposition cipher. However, it is not particularly secure, since there are a limited number of usable keys, especially for short messages (for there to be enough movement of letters, the length of the message needs to be at lease twice the key, but preferably 3 times the key). You could process these quite quickly by hand, and even more quickly with a computer.

The use of nulls can also have a detrimental effect on the security of the cipher, as an interceptor can use them to identify where the end of the line is, and so have a sensible guess at the key. This can be averted by using a more common letter, such as "E", to fill the null spaces, as it will still be clear to the recipient that these are not part of the message as they will appear at the end of the plaintext. The Rail Fence Cipher can also be utilised without the use of nulls.
One way to also make the encryption a little bit more secure, is to keep the spaces as characters, and include them in the encryption table. They are treated in exactly the same way as any other letter. For example, using the plaintext "defend the east wall" with a key of 3 again, but this time including spaces we get the table below.



D	■	■	■	N	■	■	■	H	■	■	■	A	■	■	■	W	■	■	■	X
■	E	■	E	■	D	■	T	■	E	■	E	■	S	■		■	A	■	L	■
■	■	F	■	■	■		■	■	■		■	■	■	T	■	■	■	L	■	■

The Rail Fence Cipher with spaces left in the message. Colour is used to emphasise where spaces are, against the blank squares of the table.
So the ciphertext would read "DNHAWXEEDTEES ALF TL".

Caesar Cipher:

Cryptography is the science of secrets. Literally meaning 'hidden writing,' cryptography is a method of hiding and protecting information by using a code, or cipher, only decipherable by its intended recipient. Today, modern cryptography is essential to the secure Internet, corporate cybersecurity, and blockchain technology. However, the earliest use of ciphers dates back to around 100 BC.

In this three-part series, we will explore the history of cryptography before the 20th century, in the 20th century, and in the modern day.

Historical Cryptography

Caesar Box

The "Caesar Box," or "Caesar Cipher," is one of the earliest known ciphers. Developed around 100 BC, it was used by Julius Caesar to send secret messages to his generals in the field. In the event that one of his messages got intercepted, his opponent could not read them. This obviously gave him a great strategic advantage. So, what was the code

Caesar shifted each letter of his message three letters to the right to produce what could be called the ciphertext. The ciphertext is what the enemy would see instead of the true message. So, for example, if Caesar’s messages were written in the English alphabet, each letter “A” in the message would become a "D," the "B’s" would become "E’s," and the "X's" become "A’s." This type of cipher is appropriately called a “shift cipher.”

Caesar’s magic number was three, however, a modern day use of the Caesar Cipher is a code called "ROT13." ROT13, short for "rotate by 13 places," shifts each letter of the English alphabet 13 spaces. It is often used in online forums to hide information such as movie and tv show spoilers, solutions to puzzles or games, or offensive material. The code is easily crackable, however, it hides the information from the quick glance.

Cryptanalysis: Breaking a Caesar Box

The hardest part of breaking a Caesar Box is figuring out the language of the message that it encodes. Once the code cracker figures this out, two scenarios are considered. Either the “attacker” utilizes a technique such as frequency analysis, or they use what is referred to as a brute force attack.

In the first instance, the attacker knows that certain letters are used more frequently than others. For example, A,E,O, and T are the most commonly used letters, while Q, X, and Z are the least. The relative frequencies of each letter in the English language are shown in the graph below.
A Caesar cipher is a simple method of encoding messages. Caesar ciphers use a substitution method where letters in the alphabet are shifted by some fixed number of spaces to yield an encoding alphabet. A Caesar cipher with a shift of 11 would encode an A as a B, an M as an N, and a Z as an A, and so on. The method is named after Roman leader Julius Caesar, who used it in his private correspondence.
Steps for designing and using a Caesar cipher:

- Choose a value to shift the alphabet by.
- Make a table where the top row contains letters in standard alphabetical order, and the bottom row is the new shifted alphabet.
- Encode the message by exchanging each letter in the message with the equivalent shifted letter.
- Make sure that the message’s intended recipient knows the shifting scheme you used to encode the message so they can decode it.
- To decrypt a message encoded with a Caesar cipher, simply take the value of 26 minus the shift value, and apply that new value to shift the encoded message back to its original form.

Example 1:

Using a Caesar cipher described by the table below, encode the following message: “I like chemistry”.

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
k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j

This is a shift of 10.

A shift of 10 encodes “I like chemistry” to “S vsuo mrowscdbi”.

Example 2:

Using a Caesar cipher with a shift of 14, encode the following message: I send secret messages.

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
o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n

Go through the table and match each letter in the phrase "I send secret messages" to the corresponding encoded letter in the table. For example, the letter "s" is encoded as "g" and the letter "c" is encoded as "q".

Using this process, "I send secret messages" can be encoded as "W gsbr gsqfsh asggousg".

Pros and Cons of a Caesar Cipher

A Caesar cipher is very easy to design, but also very easy to decode. To crack a Caesar code, a decoder could simply go through every possible shift of the alphabet (all 2626 of them) and see if any sensible message appears. This is a relatively small number of combinations to check, for perspective, a message encoded with a Vigenère cipher has over 11 million possible combinations (and that is if the key is only five letters long — the longer the key, the more combinations), and the Enigma code has 158,962,555,217,826,360,000158,962,555,217,826,360,000 possible combinations.

Vigenère cipher

History:

First described by Giovan Battista Bellaso in 1553, the cipher is easy to understand and implement, but it resisted all attempts to break it until 1863, three centuries later. This earned it the description le chiffrage indéchiffrable (French for 'the indecipherable cipher'). Many people have tried to implement encryption schemes that are essentially Vigenère ciphers. In 1863, Friedrich Kasiski was the first to publish a general method of deciphering Vigenère ciphers. In the 19th century the scheme was misattributed to Blaise de Vigenère (1523–1596), and so acquired its present name.

The very first well-documented description of a polyalphabetic cipher was by Leon Battista Alberti around 1467 and used a metal cipher disk to switch between cipher alphabets. Alberti's system only switched alphabets after several words, and switches were indicated by writing the letter of the corresponding alphabet in the ciphertext. Later, Johannes Trithemius, in his work Polygraphiae (which was completed in manuscript form in 1508 but first published in 1518), invented the tabula recta, a critical component of the Vigenère cipher. The Trithemius cipher, however, provided a progressive, rather rigid and predictable system for switching between cipher alphabets.

In 1586 Blaise de Vigenère published a type of polyalphabetic cipher called an autokey cipher – because its key is based on the original plaintext – before the court of Henry III of France. The cipher now known as the Vigenère cipher, however, is that originally described by Giovan Battista Bellaso in his 1553 book La cifra del Sig. Giovan Battista Bellaso.He built upon the tabula recta of Trithemius but added a repeating "countersign" (a key) to switch cipher alphabets every letter. Whereas Alberti and Trithemius used a fixed pattern of substitutions, Bellaso's scheme meant the pattern of substitutions could be easily changed, simply by selecting a new key. Keys were typically single words or short phrases, known to both parties in advance, or transmitted "out of band" along with the message. Bellaso's method thus required strong security for only the key. As it is relatively easy to secure a short key phrase, such as by a previous private conversation, Bellaso's system was considerably more secure.[citation needed]

In the 19th century, the invention of Bellaso's cipher was misattributed to Vigenère. David Kahn, in his book, The Codebreakers lamented this misattribution, saying that history had "ignored this important contribution and instead named a regressive and elementary cipher for him [Vigenère] though he had nothing to do with it".

The Vigenère cipher gained a reputation for being exceptionally strong. Noted author and mathematician Charles Lutwidge Dodgson (Lewis Carroll) called the Vigenère cipher unbreakable in his 1868 piece "The Alphabet Cipher" in a children's magazine. In 1917, Scientific American described the Vigenère cipher as "impossible of translation". That reputation was not deserved. Charles Babbage is known to have broken a variant of the cipher as early as 1854 but did not publish his work. Kasiski entirely broke the cipher and published the technique in the 19th century, but even in the 16th century, some skilled cryptanalysts could occasionally break the cipher.

The Vigenère cipher is simple enough to be a field cipher if it is used in conjunction with cipher disks. The Confederate States of America, for example, used a brass cipher disk to implement the Vigenère cipher during the American Civil War. The Confederacy's messages were far from secret, and the Union regularly cracked its messages. Throughout the war, the Confederate leadership primarily relied upon three key phrases: "Manchester Bluff", "Complete Victory" and, as the war came to a close, "Come Retribution".

A Vigenère cipher with a completely random (and non-reusable) key which is as long as the message becomes a one-time pad, a theoretically unbreakable cipher. Gilbert Vernam tried to repair the broken cipher (creating the Vernam–Vigenère cipher in 1918), but the technology he used was so cumbersome as to be impracticable.

Description:

In a Caesar cipher, each letter of the alphabet is shifted along some number of places. For example, in a Caesar cipher of shift 3, a would become D, b would become E, y would become B and so on. The Vigenère cipher has several Caesar ciphers in sequence with different shift values.

To encrypt, a table of alphabets can be used, termed a tabula recta, Vigenère square or Vigenère table. It has the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows. The alphabet used at each point depends on a repeating keyword.[citation needed]

For example, suppose that the plaintext to be encrypted is
"attackatdawn".
The person sending the message chooses a keyword and repeats it until it matches the length of the plaintext, for example, the keyword "LEMON":

"LEMONLEMONLE"
Each row starts with a key letter. The rest of the row holds the letters A to Z (in shifted order). Although there are 26 key rows shown, a code will use only as many keys (different alphabets) as there are unique letters in the key string, here just 5 keys: {L, E, M, O, N}. For successive letters of the message, successive letters of the key string will be taken and each message letter enciphered by using its corresponding key row. The next letter of the key is chosen, and that row is gone along to find the column heading that matches the message character. The letter at the intersection of [key-row, msg-col] is the enciphered letter.

For example, the first letter of the plaintext, a, is paired with L, the first letter of the key. Therefore, row L and column A of the Vigenère square are used, namely L. Similarly, for the second letter of the plaintext, the second letter of the key is used. The letter at row E and column T is X. The rest of the plaintext is enciphered in a similar fashion:

Plaintext: attackatdawn

Key: LEMONLEMONLE

Ciphertext: LXFOPVEFRNHR

Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in that row and then using the column's label as the plaintext. For example, in row L (from LEMON), the ciphertext L appears in column A, so a is the first plaintext letter. Next, in row E (from LEMON), the ciphertext X is located in column T. Thus t is the second plaintext letter.

Cryptanalysis:

The idea behind the Vigenère cipher, like all other polyalphabetic ciphers, is to disguise the plaintext letter frequency to interfere with a straightforward application of frequency analysis. For instance, if P is the most frequent letter in a ciphertext whose plaintext is in English, one might suspect that P corresponds to e since e is the most frequently used letter in English. However, by using the Vigenère cipher, e can be enciphered as different ciphertext letters at different points in the message, which defeats simple frequency analysis.

The primary weakness of the Vigenère cipher is the repeating nature of its key. If a cryptanalyst correctly guesses the key's length n, the cipher text can be treated as n interleaved Caesar ciphers, which can easily be broken individually. The key length may be discovered by brute force testing each possible value of n, or Kasiski examination and the Friedman test can help to determine the key length (see below: § Kasiski examination and § Friedman test).

Example :

The running key variant of the Vigenère cipher was also considered unbreakable at one time. For the key, this version uses a block of text as long as the plaintext. Since the key is as long as the message, the Friedman and Kasiski tests no longer work, as the key is not repeated.

If multiple keys are used, the effective key length is the least common multiple of the lengths of the individual keys. For example, using the two keys `GO` and `CAT`, whose lengths are 2 and 3, one obtains an effective key length of 6 (the least common multiple of 2 and 3). This can be understood as the point where both keys line up.

Plaintext:	`attackatdawn`
Key 1:	`GOGOGO`**GOGOGO**
Key 2:	`CATCAT`**CATCAT**
Ciphertext:	`IHSQIRIHCQCU`

Encrypting twice, first with the key `GO` and then with the key `CAT` is the same as encrypting once with a key produced by encrypting one key with the other.

Plaintext:	`gogogo`
Key:	`CATCAT`
Ciphertext:	`IOZQGH`

This is demonstrated by encrypting `attackatdawn` with `IOZQGH`, to produce the same ciphertext as in the original example.

Plaintext:	`attackatdawn`
Key:	`IOZQGHIOZQGH`
Ciphertext:	`IHSQIRIHCQCU`

If key lengths are relatively prime, the effective key length grows exponentially as the individual key lengths are increased. For example, while the effective length of keys 10, 12, and 15 characters is only 60, that of keys of 8, 11, and 15 characters is 1320. If this effective key length is longer than the ciphertext, it achieves the same immunity to the Friedman and Kasiski tests as the running key variant.

If one uses a key that is truly random, is at least as long as the encrypted message, and is used only once, the Vigenère cipher is theoretically unbreakable. However, in that case, the key, not the cipher, provides cryptographic strength, and such systems are properly referred to collectively as one-time pad systems, irrespective of the ciphers employed.

Playfair Cipher:

The Playfair cipher or Playfair square or Wheatstone–Playfair cipher is a manual symmetric encryption technique and was the first literal digram substitution cipher. The scheme was invented in 1854 by Charles Wheatstone, but bears the name of Lord Playfair for promoting its use.

The technique encrypts pairs of letters (bigrams or digrams), instead of single letters as in the simple substitution cipher and rather more complex Vigenère cipher systems then in use. The Playfair is thus significantly harder to break since the frequency analysis used for simple substitution ciphers does not work with it. The frequency analysis of bigrams is possible, but considerably more difficult. With 600 possible bigrams rather than the 26 possible monograms (single symbols, usually letters in this context), a considerably larger cipher text is required in order to be useful.

History:

The Playfair cipher was the first cipher to encrypt pairs of letters in cryptologic history. Wheatstone invented the cipher for secrecy in telegraphy, but it carries the name of his friend Lord Playfair, first Baron Playfair of St. Andrews, who promoted its use. The first recorded description of the Playfair cipher was in a document signed by Wheatstone on 26 March 1854.

It was initially rejected by the British Foreign Office when it was developed because of its perceived complexity. Wheatstone offered to demonstrate that three out of four boys in a nearby school could learn to use it in 15 minutes, but the Under Secretary of the Foreign Office responded, "That is very possible, but you could never teach it to attachés."

It was however later used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the British and Australians during World War II. This was because Playfair is reasonably fast to use and requires no special equipment - just a pencil and some paper. A typical scenario for Playfair use was to protect important but non-critical secrets during actual combat e.g. the fact that an artillery barrage of smoke shells would commence within 30 minutes to cover soldiers' advance towards the next objective. By the time enemy cryptanalysts could decode such messages hours later, such information would be useless to them because it was no longer relevant.

During World War II, the Government of New Zealand used it for communication among New Zealand, the Chatham Islands, and the coastwatchers in the Pacific Islands. Coastwatchers established by Royal Australian Navy Intelligence also used this cipher.

Cryptanlysis:

Like most classical ciphers, the Playfair cipher can be easily cracked if there is enough text. Obtaining the key is relatively straightforward if both plaintext and ciphertext are known. When only the ciphertext is known, brute force cryptanalysis of the cipher involves searching through the key space for matches between the frequency of occurrence of diagrams (pairs of letters) and the known frequency of occurrence of diagrams in the assumed language of the original message.

Cryptanalysis of Playfair is similar to that of four-square and two-square ciphers, though the relative simplicity of the Playfair system makes identifying candidate plaintext strings easier. Most notably, a Playfair digraph and its reverse (e.g. AB and BA) will decrypt to the same letter pattern in the plaintext (e.g. RE and ER). In English, there are many words which contain these reversed digraphs such as Receiver and Departed. Identifying nearby reversed digraphs in the ciphertext and matching the pattern to a list of known plaintext words containing the pattern is an easy way to generate possible plaintext strings with which to begin constructing the key.

A different approach to tackling a Playfair cipher is the shotgun hill climbing method. This starts with a random square of letters. Then minor changes are introduced (i.e. switching letters, rows, or reflecting the entire square) to see if the candidate plaintext is more like standard plaintext than before the change (perhaps by comparing the digrams to a known frequency chart). If the new square is deemed to be an improvement, then it is adopted and then further mutated to find an even better candidate. Eventually, the plaintext or something very close is found to achieve a maximal score by whatever grading method is chosen. This is obviously beyond the range of typical human patience, but computers can adopt this algorithm to crack Playfair ciphers with a relatively small amount of text.

Another aspect of Playfair that separates it from four-square and two-square ciphers is the fact that it will never contain a double-letter digram, e.g. EE. If there are no double letter digrams in the ciphertext and the length of the message is long enough to make this statistically significant, it is very likely that the method of encryption is Playfair.

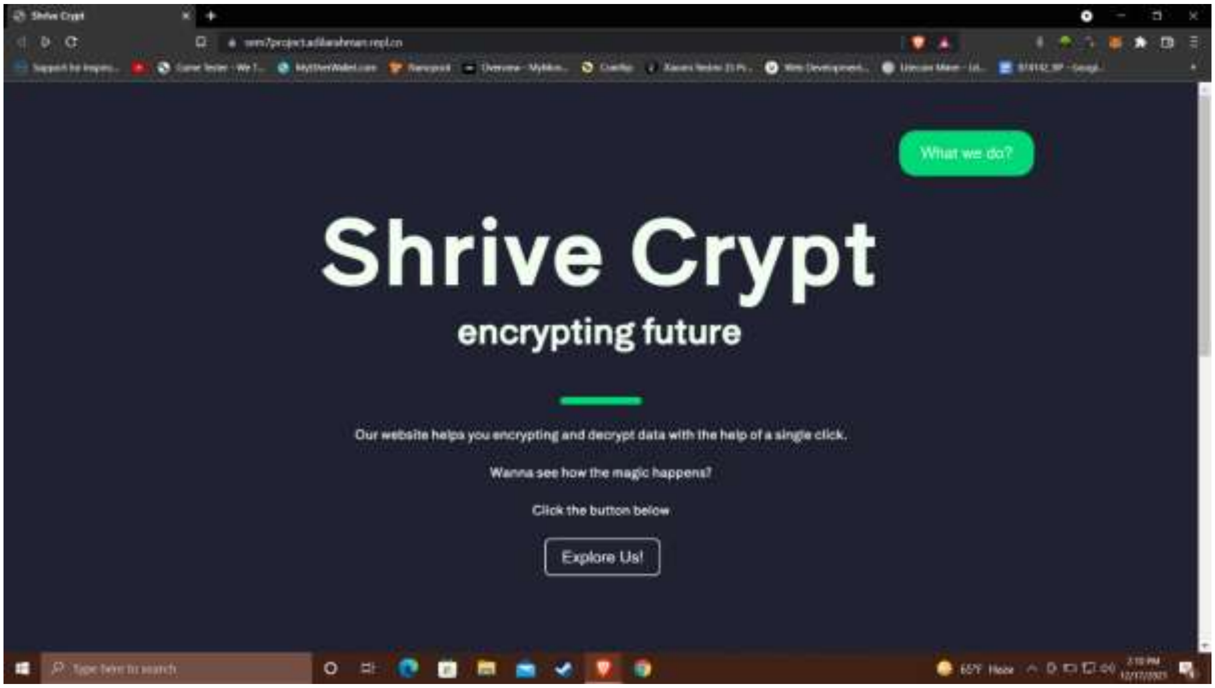
A good tutorial on reconstructing the key for a Playfair cipher can be found in chapter 7, "Solution to Polygraphic Substitution Systems," of Field Manual 34-40-2, produced by the United States Army. Another cryptanalysis of a Playfair cipher can be found in Chapter XXI of Helen Fouché Gaines, Cryptanalysis / a study of ciphers and their solutions.

A detailed cryptanalysis of Playfair is undertaken in chapter 28 of Dorothy L. Sayers' mystery novel Have His Carcase. In this story, a Playfair message is demonstrated to be cryptographically weak, as the detective is able to solve for the entire key making only a few guesses as to the formatting of the message (in this case, that the message starts with the name of a city and then a date). Sayers' book includes a detailed description of the mechanics of Playfair encryption, as well as a step-by-step account of manual cryptanalysis.

The German Army, Air Force and Police used the Double Playfair cipher as a medium-grade cipher in WWII, based on the British Playfair cipher they had broken early in WWI. They adapted it by introducing a second square from which the second letter of each bigram was selected, and dispensed with the keyword, placing the letters in random order. But with the German fondness for pro forma messages, they were broken at Bletchley Park. Messages were preceded by a sequential number, and numbers were spelled out. As the German numbers 1 (eins) to twelve (zwölf) contain all but eight of the letters in the Double Playfair squares, pro forma traffic was relatively easy to break

CHAPTER-4

Results and Discussion



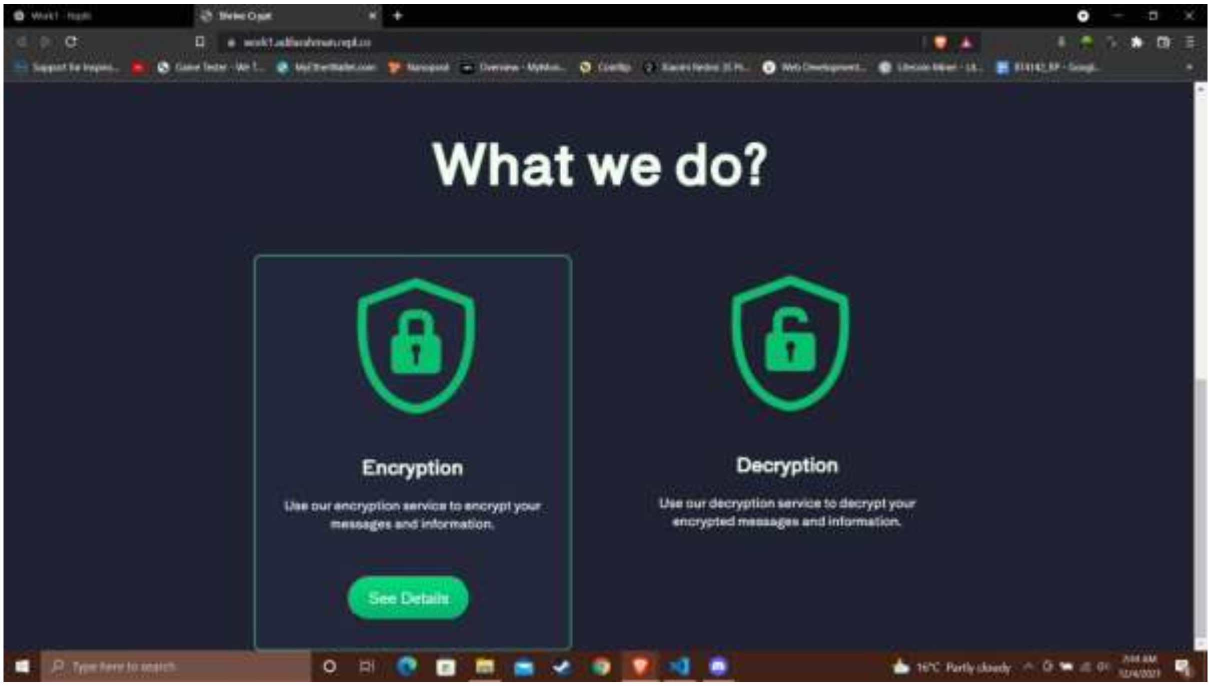
Shrive Crypt

encrypting future

Our website helps you encrypting and decrypt data with the help of a single click.

Wanna see how the magic happens?

Click the button below



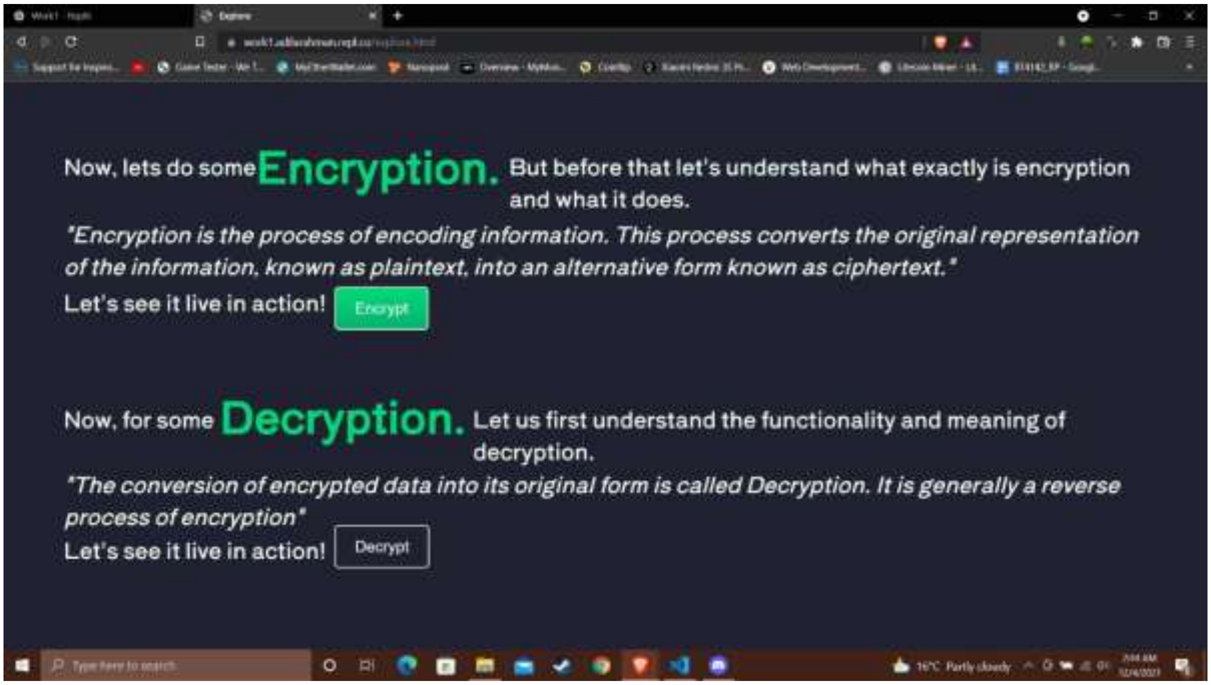
What we do?

Encryption

Use our encryption service to encrypt your messages and information.

Decryption

Use our decryption service to decrypt your encrypted messages and information.



Now, lets do some **Encryption.** But before that let's understand what exactly is encryption and what it does.

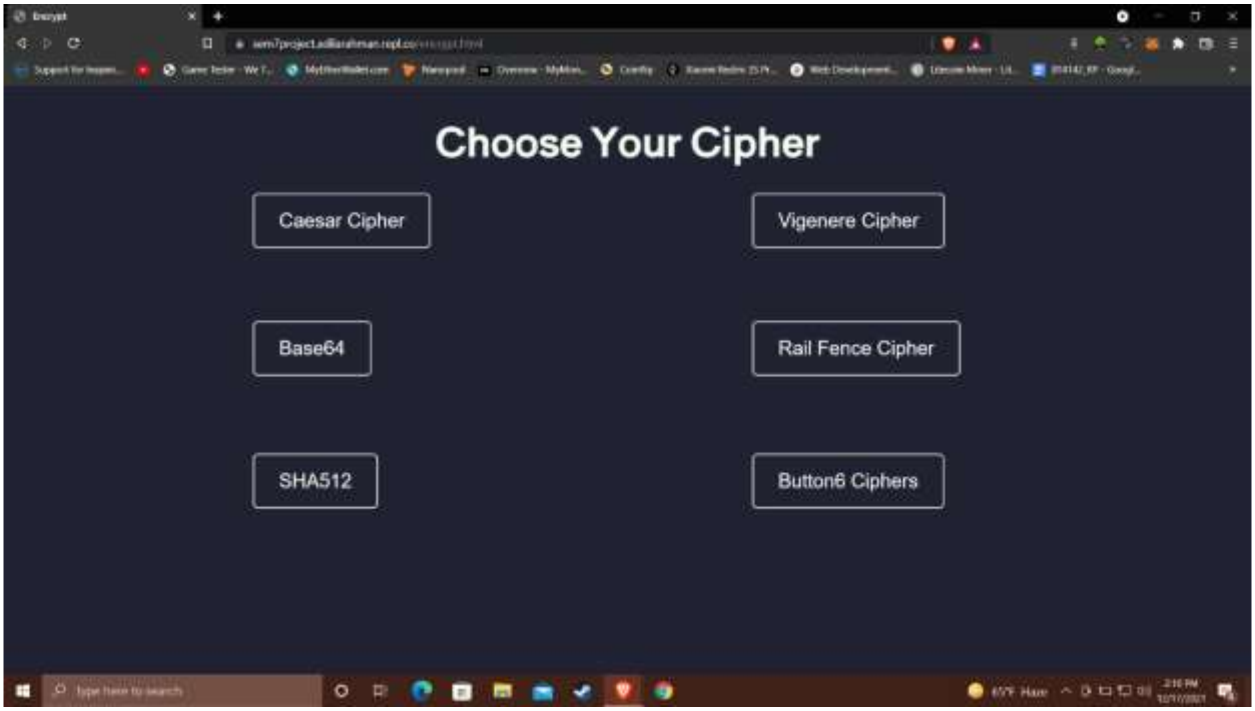
"Encryption is the process of encoding information. This process converts the original representation of the information, known as plaintext, into an alternative form known as ciphertext."

Let's see it live in action!

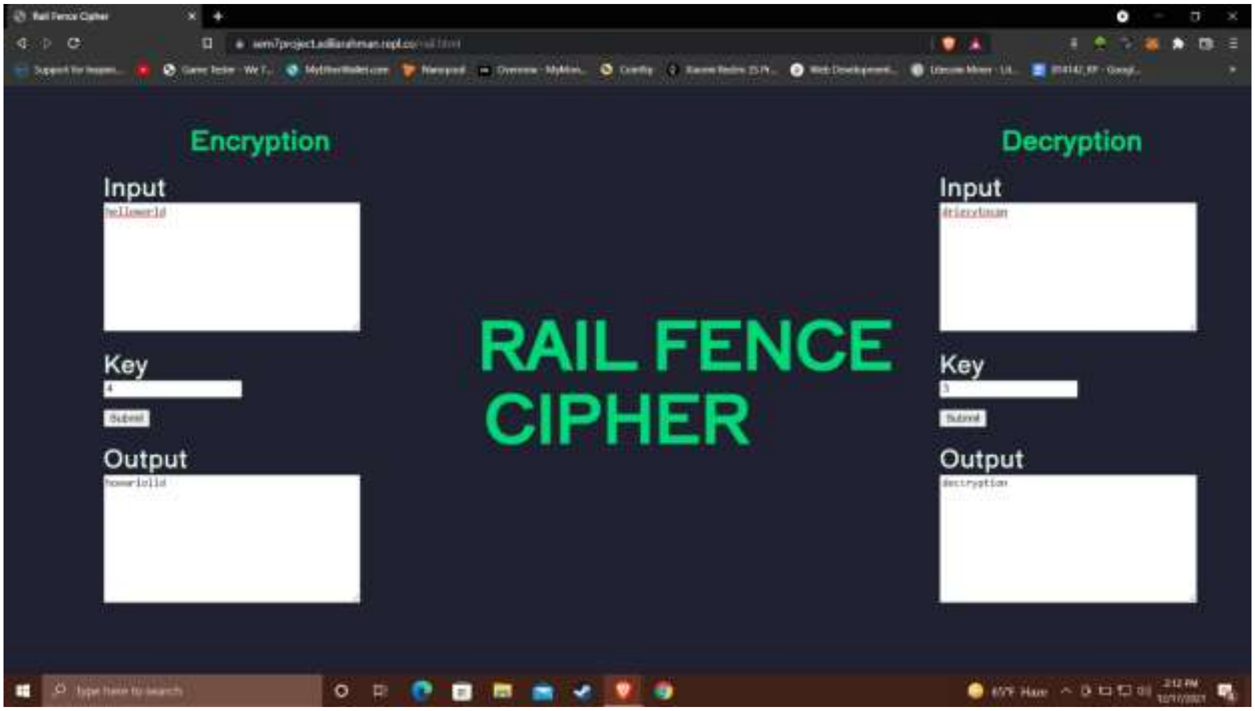
Now, for some **Decryption.** Let us first understand the functionality and meaning of decryption.

"The conversion of encrypted data into its original form is called Decryption. It is generally a reverse process of encryption"

Let's see it live in action!



Choose Your Cipher



Encryption

Input

helloworld

Key

4

Output

hoewrlolld

RAIL FENCE CIPHER

Decryption

Input

[illegible]

Key

3

Output

decryption

CHAPTER-5

Conclusion and Future Scope

As we toward a society where automated information resources are increased and cryptography will continue to increase in importance as a security mechanism. Electronic networks for banking, shopping, inventory control, benefit and service delivery, information storage and retrieval, distributed processing, and government applications will need improved methods for access control and data security. The information security can be easily achieved by using Cryptography technique.

Our Future Scope is to include morse-code functionality in our website. Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes, or dits and dahs. Morse code is named after Samuel Morse, one of the inventors of the telegraph.

Morse code can be memorized and sent in a form perceptible to the human senses, e.g., via sound waves or visible light, such that it can be directly interpreted by persons trained in the skill. Morse code is usually transmitted by on-off keying of an information-carrying medium such as electric current, radio waves, visible light, or sound waves. The current or wave is present during the time period of the dit or dah and absent during the time between dits and dahs.