Applied Science Private University

Faculty of Information Technology

Department of Cyber Security

THE THEORY OF CRYPTOGRAPHY AND DATA AUTHENTICATION

DOCUMENTATION

Topic:

HYBRID CIPHER

Prepared by:

Omar Hijah

Zeyad Altaslaq

Asya Abdel-All

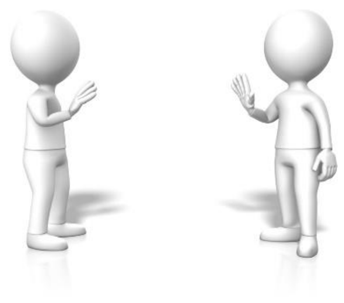
Omar Abulfeilat

Danah Abdal-Hafeath

Doctor: Ahmad Otoom

TABLE OF CONTENT

1. Introduction



1.1 Overview

What is Hybrid Cipher?

Hybrid cipher is a cryptographic project that aims to blend the functionalities of classic and advanced ciphers into a novel and robust encryption system. By leveraging the strengths of these historical ciphers and introducing innovative modifications and utilizing a combination of classic and advanced encryption techniques such as substitution, transposition, and polyalphabetic techniques.

1.2 Symmetric Key Encryption

- This cipher is built on the idea of symmetric key encryption, Symmetric-key [1] algorithms are algorithms for cryptography that use the same cryptographic keys for both the encryption of plaintext and the decryption of cipher text. The keys may be identical, or there may be a simple transformation to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link.

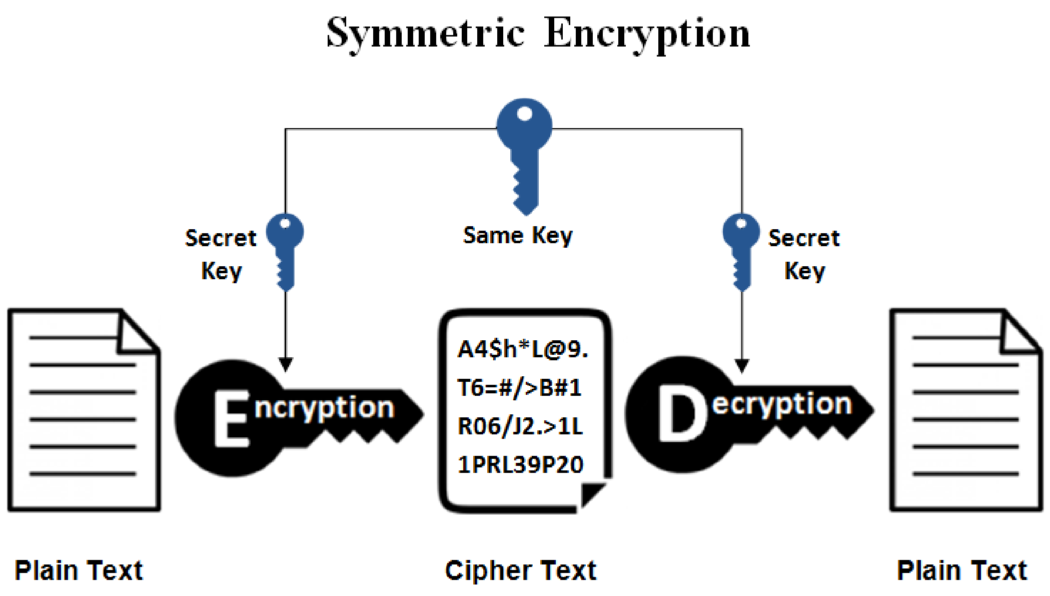


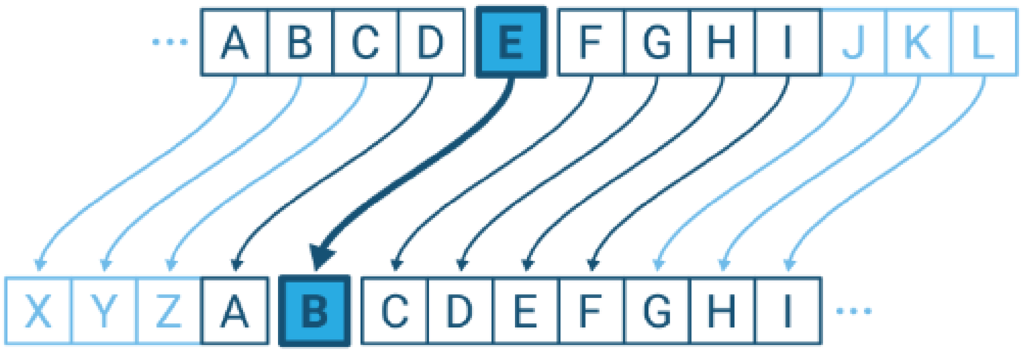
Figure 1: Symmetric Key Encryption

1.3 Stream Cipher

- This is also a stream cipher [2]; a stream cipher is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream (keystream). In a stream cipher, each plaintext digit is encrypted one at a time with the corresponding digit of the keystream, to give a digit of the cipher text stream.

- The Python code provided implements a cryptographic algorithm based on the Caesar cipher technique [3]. This cipher operates by shifting each letter in the message by a variable number of positions in the alphabet based on the key input of the user.

Figure 3: Caser Cipher



- The script begins by importing necessary libraries, including 'base64'[4] for key encoding and " itertools “for cycling through keys longer than the message.

2. Key Generation

2.1 Input Key: The user inputs a key for encryption or decryption.

2.2 Encoding Key with Base64 [4]: The script encodes the input key using Base64 encoding [4]. This step obscures the original plaintext key, enhancing the security of the encryption process. The encoded key is then used for encryption or decryption.

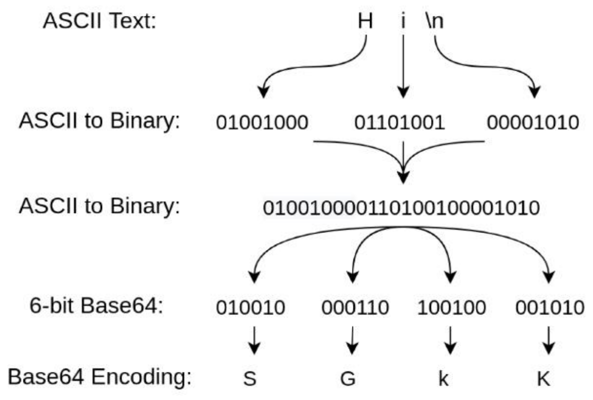
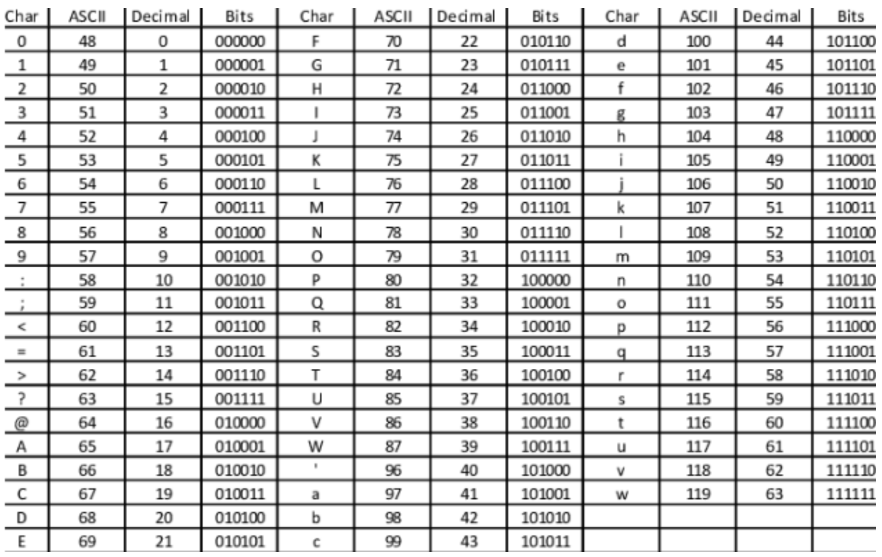


Figure 4: Base64 encoding

2.3 Transforming Encoded Key into ASCII Representation

2.3 Transforming Encoded Key into ASCII Representation [5]: The script converts the input key into its ASCII representation. This transformation ensures compatibility with the Caesar cipher algorithm, which operates on numerical values rather than characters directly.



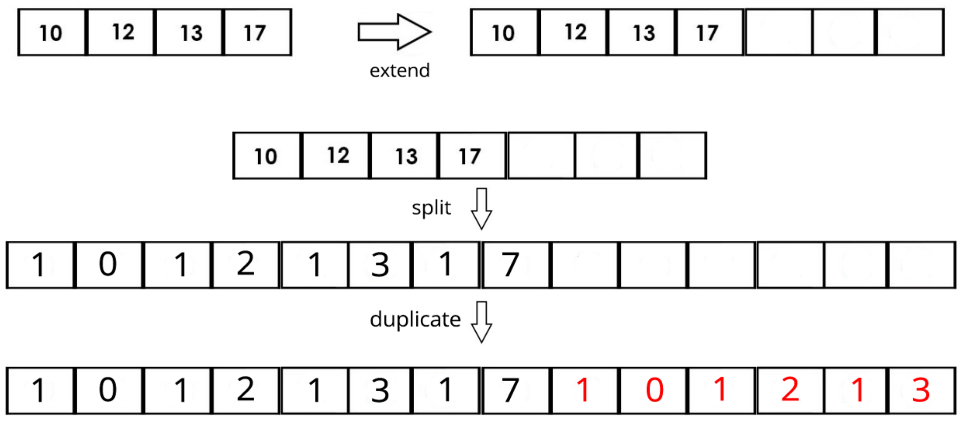
Char	ASCII	Decimal	Bits	Char	ASCII	Decimal	Bits	Char	ASCII	Decimal	Bits
0	48	0	000000	F	70	22	010110	d	100	44	101100
1	49	1	000001	G	71	23	010111	e	101	45	101101
2	50	2	000010	H	72	24	011000	f	102	46	101110
3	51	3	000011	I	73	25	011001	g	103	47	101111
4	52	4	000100	J	74	26	011010	h	104	48	110000
5	53	5	000101	K	75	27	011011	i	105	49	110001
6	54	6	000110	L	76	28	011100	j	106	50	110010
7	55	7	000111	M	77	29	011101	k	107	51	110011
8	56	8	001000	N	78	30	011110	l	108	52	110100
9	57	9	001001	O	79	31	011111	m	109	53	110101
:	58	10	001010	P	80	32	100000	n	110	54	110110
;	59	11	001011	Q	81	33	100001	o	111	55	110111
<	60	12	001100	R	82	34	100010	p	112	56	111000
=	61	13	001101	S	83	35	100011	q	113	57	111001
>	62	14	001110	T	84	36	100100	r	114	58	111010
?	63	15	001111	U	85	37	100101	s	115	59	111011
@	64	16	010000	V	86	38	100110	t	116	60	111100
A	65	17	010001	W	87	39	100111	u	117	61	111101
B	66	18	010010	'	96	40	101000	v	118	62	111110
C	67	19	010011	a	97	41	101001	w	119	63	111111
D	68	20	010100	b	98	42	101010				
E	69	21	010101	c	99	43	101011				

Figure 5: ASCII Table

2.4 Duplicating Key

If the key is shorter than the message, the script duplicates it intelligently using “itertools.cycle” to ensure that each character in the message receives a corresponding ASCII value.

Figure 6: Duplicating Key to match plaintext length



3. Encryption

3.1 User Input

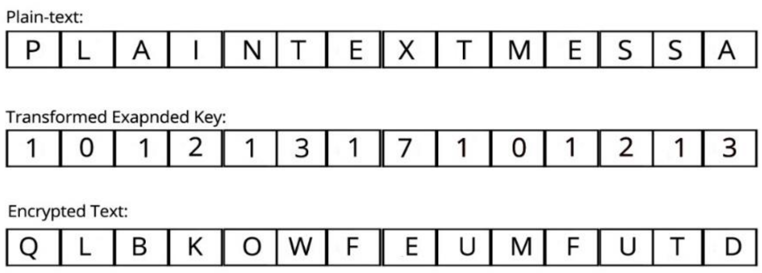
Upon execution, the user is prompted to input a message and a key for encryption or decryption. The entered message is converted to uppercase for consistency, ensuring proper handling of both uppercase and lowercase characters. The algorithm iterates over each character in the message, encrypting or decrypting it based on the provided key using the Caesar cipher technique.

3.2 Encrypting the Message

Encrypting the Message: The script iterates over each character in the input message. For each character, it applies the Caesar cipher encryption technique (character rotation). This involves shifting the character by the given number of positions in the alphabet, determined by the **corresponding ASCII value of the key**. The result of this encryption process is stored in a variable ‘var’.

3.3 Output Encrypted Message

The script outputs the encrypted message to the user.



Plain-text:

P	L	A	I	N	T	E	X	T	M	E	S	S	A

Transformed Exapnded Key:

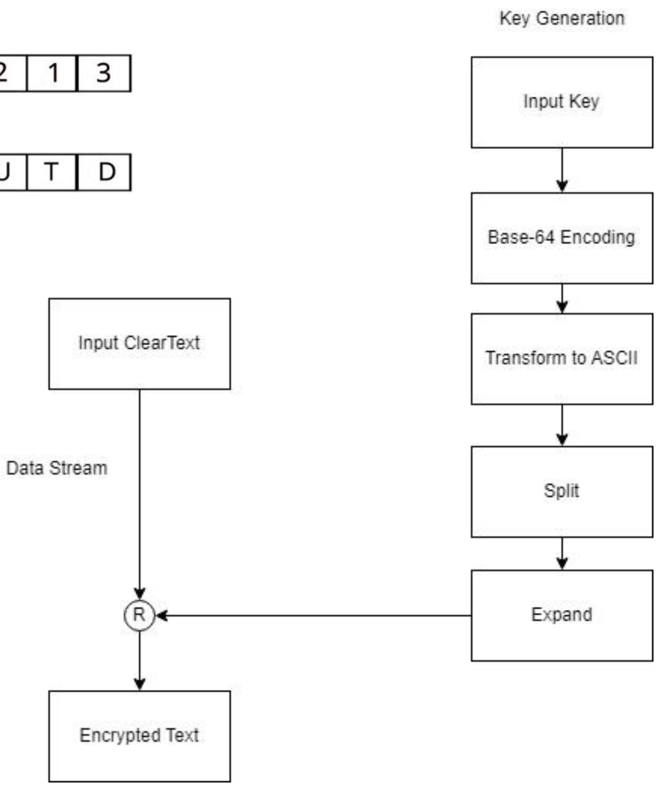
1	0	1	2	1	3	1	7	1	0	1	2	1	3

Encrypted Text:

Q	L	B	K	O	W	F	E	U	M	F	U	T	D

Figure7: Example Encrypted Text

The Process of Encryption



4. Decryption

4.1 User Input

Input Encrypted Message and Key: Similar to encryption, the user is prompted to input the encrypted message they want to decrypt and the decryption key.

4.2 Transforming Key into ASCII Representation

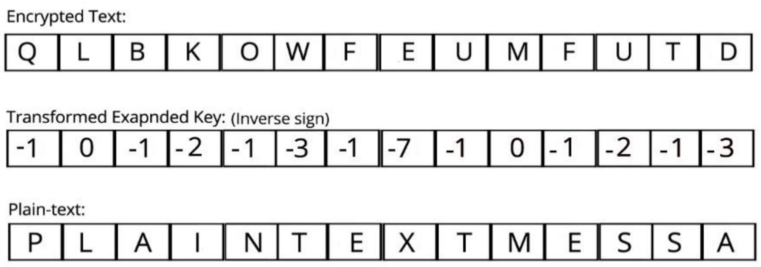
Transforming Key into ASCII Representation: The script converts the input key into its ASCII representation, ensuring compatibility with the decryption process.

4.3 Decrypting the Message

Decrypting the Message: The script iterates over each character in the encrypted message. For each character, it applies the Caesar cipher decryption technique. This involves shifting the character in the opposite direction by the given number of positions in the alphabet, determined by the corresponding ASCII value of the key. The result of this decryption process is stored in a variable 'prev'.

4.4 Output Decrypted Message

Output Decrypted Message: finally, the script outputs the decrypted message to the user.



Encrypted Text:

Q	L	B	K	O	W	F	E	U	M	F	U	T	D

Transformed Exapnded Key: (Inverse sign)

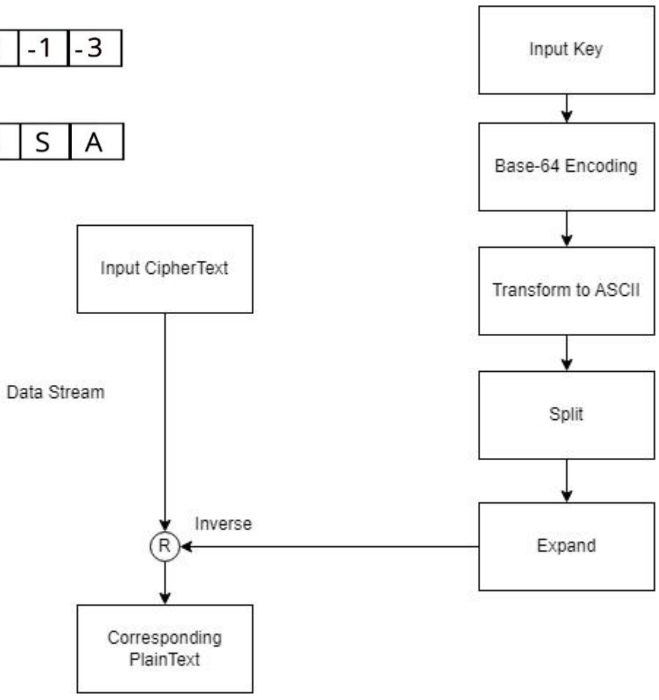
-1	0	-1	-2	-1	-3	-1	-7	-1	0	-1	-2	-1	-3

Plain-text:

P	L	A	I	N	T	E	X	T	M	E	S	S	A

Figure 8: Example Decrypted Message

The Process of Decryption



5. Security Advantages

5.1 Enhanced Security

The Hybrid Cipher encryption algorithm surpasses the Caesar Cipher due to its polyalphabetic nature, which significantly enhances its security. Unlike the monoalphabetic Caesar Cipher, which shifts all characters by a fixed amount, making it vulnerable to frequency analysis. This variation ensures that the same letter can be encoded differently depending on its position in the text, thereby thwarting frequency analysis. As a result, patterns in the encrypted message are obscured, making it exceedingly difficult for attackers to deduce the original plaintext through analysis methods. This complexity makes the Hybrid Cipher a far more robust and secure choice for encryption.

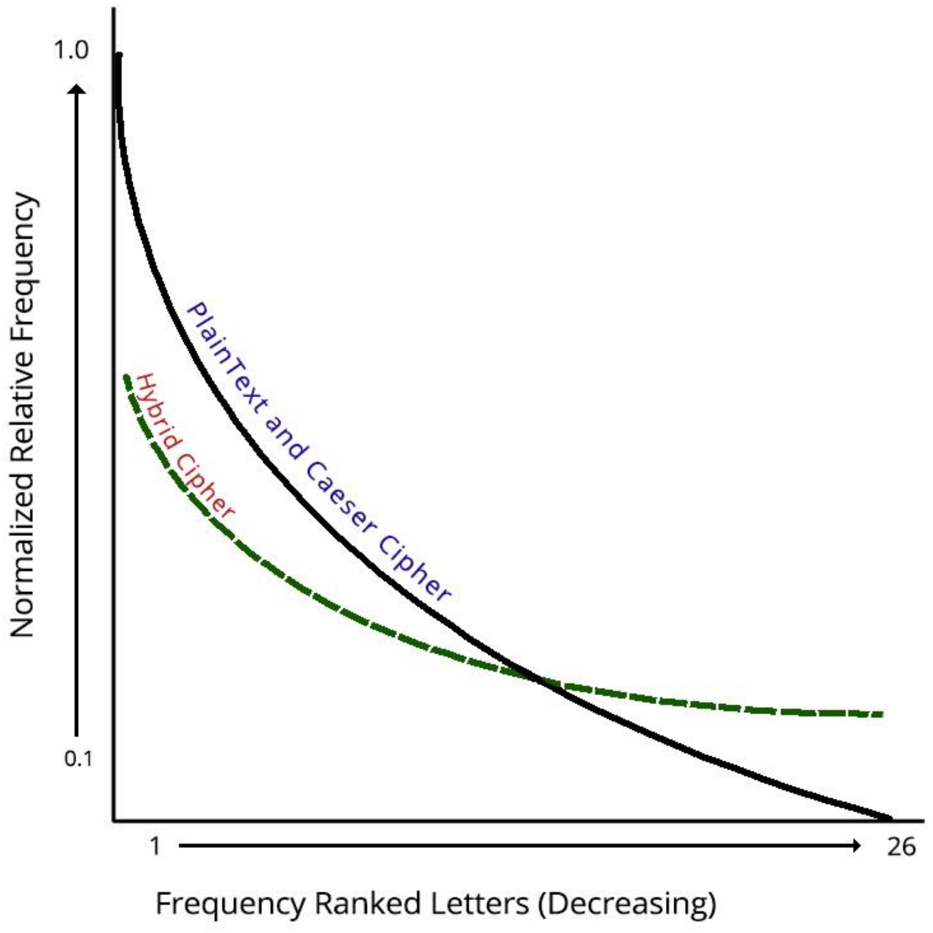
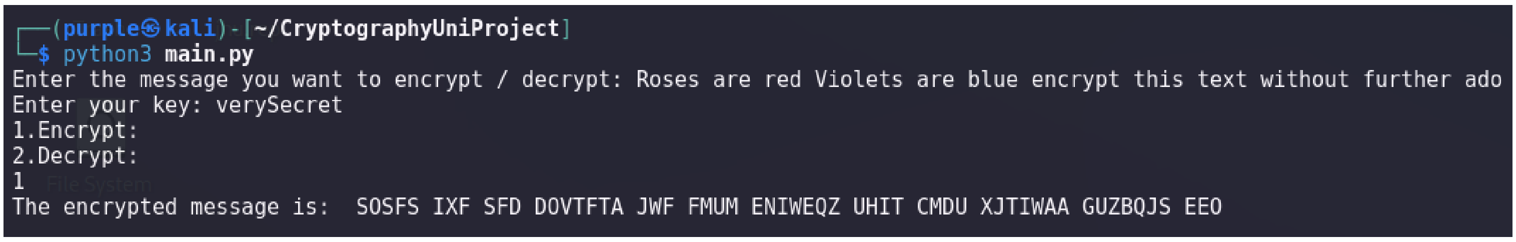


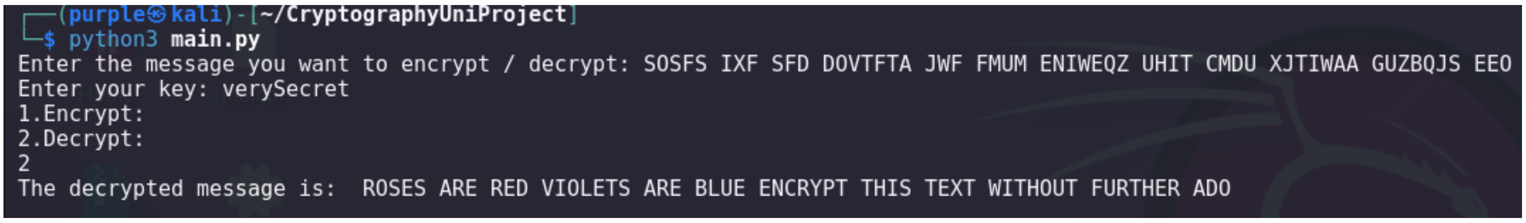
Figure 9: Caesar vs Hybrid Frequency

Encryption Example:



```
┌──(purple㉿kali)-[~/CryptographyUniProject]
└─$ python3 main.py
Enter the message you want to encrypt / decrypt: Roses are red Violets are blue encrypt this text without further ado
Enter your key: verySecret
1.Encrypt:
2.Decrypt:
1
The encrypted message is:  SOSFS IXF SFD DOVTFTA JWF FMUM ENIWEQZ UHIT CMDU XJTIWAA GUZBQJS EEO
```

Decryption Example:



```
┌──(purple㉿kali)-[~/CryptographyUniProject]
└─$ python3 main.py
Enter the message you want to encrypt / decrypt: SOSFS IXF SFD DOVTFTA JWF FMUM ENIWEQZ UHIT CMDU XJTIWAA GUZBQJS EEO
Enter your key: verySecret
1.Encrypt:
2.Decrypt:
2
The decrypted message is:  ROSES ARE RED VIOLETS ARE BLUE ENCRYPT THIS TEXT WITHOUT FURTHER ADO
```

6. Script Download

➢ **Download the script:**

Git clone https://github.com/FireBotato/CryptographyUniProject

7. Usefulness

7.1 Simplicity and Versatility

The cryptographic algorithm implemented in the provided Python code offers several advantages that contribute to its effectiveness and suitability for secure communication.

7.1.1 Leveraging the Caesar Cipher

Firstly, the algorithm leverages the robustness of the Caesar cipher, a classical encryption technique renowned for its simplicity and versatility. By employing this time-tested cipher, the algorithm ensures a straightforward yet reliable method for encrypting and decrypting messages. This simplicity not only facilitates ease of implementation but also enables efficient computation, making it suitable for resource-constrained environments.

7.1.2 Benefits of Python

This happens because we are using the programming language Python, which is available on every kind of operating system. Python’s availability across various platforms ensures broad compatibility, making it easier to deploy the algorithm in diverse environments. Additionally, Python’s readability and simplicity make the code easy to understand, maintain, and modify. This is particularly beneficial for educational purposes, allowing students and developers to quickly grasp the concepts and mechanics behind the algorithm. Furthermore, the ease of editing in Python supports rapid prototyping and testing of new features or improvements, ensuring the algorithm can evolve and adapt over time.

7.2 Enhanced Security with Base64 Encoding

Moreover, the integration of Base64 encoding for key representation significantly enhances the security of the algorithm. By encoding the key, the algorithm obscures the original plaintext key, mitigating the risk of key exposure and unauthorized access. This additional layer of security strengthens the confidentiality of encrypted messages, safeguarding sensitive information from prying eyes and malicious actors. Base64 encoding transforms the key into a format that is not immediately recognizable, adding a crucial layer of protection. Even if an attacker intercepts the encoded key, they would still need to decode it before using it, which complicates unauthorized access.

7.3 Dynamic Key Transformation

Furthermore, the algorithm incorporates dynamic key transformation mechanisms to accommodate keys of varying lengths relative to the message. Through the intelligent duplication of shorter keys using `itertools.cycle`, the algorithm ensures consistency in the encryption process, regardless of key length disparities. This adaptive approach not only enhances the algorithm's versatility but also promotes robustness and resilience against potential cryptographic attacks. By ensuring that every character in the message has a corresponding key character, the algorithm maintains its integrity and effectiveness across different scenarios.

7.4 User-Friendly Interface

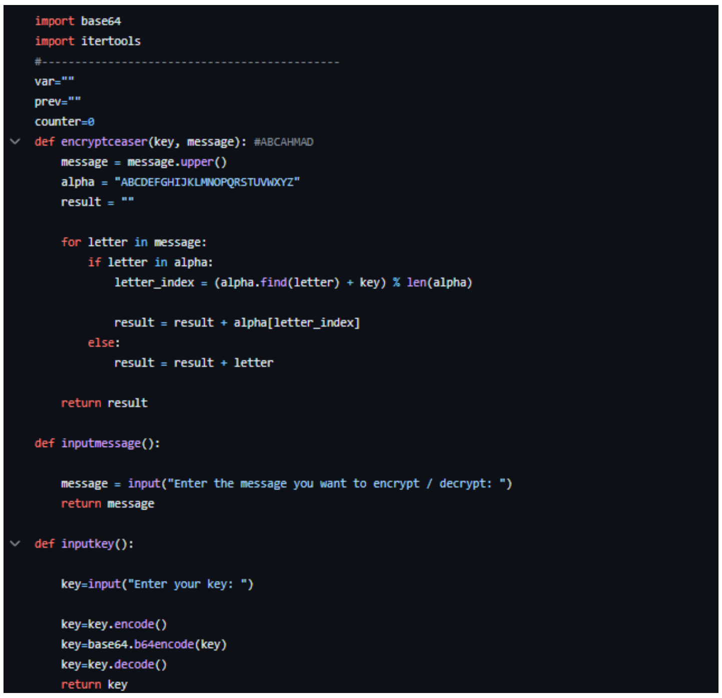
Additionally, the algorithm provides a user-friendly interface, enabling seamless interaction with the encryption and decryption processes. By prompting users to input messages and keys interactively, the algorithm simplifies the encryption and decryption workflow, minimizing user errors and streamlining cryptographic operations. This intuitive design fosters accessibility and usability, making the algorithm accessible to users with diverse levels of technical expertise. Whether someone is a beginner or an experienced cryptographer, the straightforward user interface allows them to efficiently secure their communications.

7.5 Overall Effectiveness

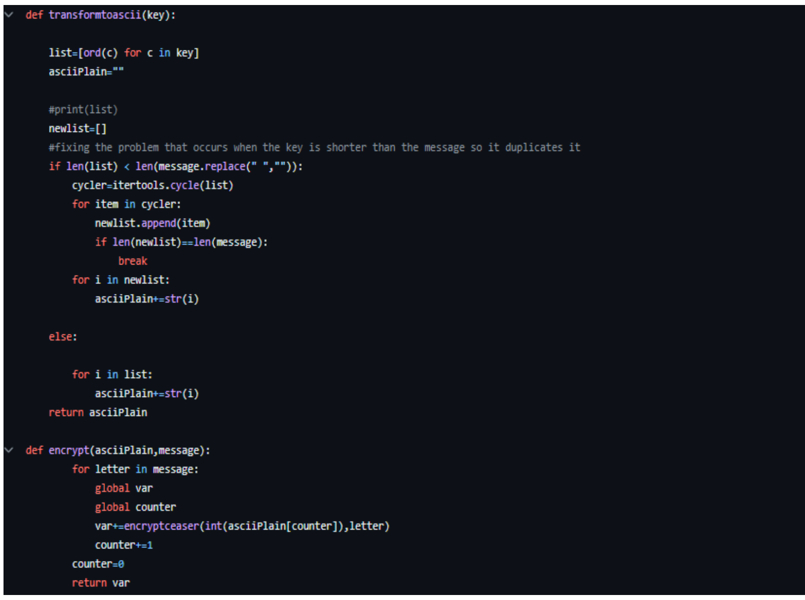
Overall, the cryptographic algorithm implemented in the provided Python code embodies a balance of simplicity, security, and usability. By leveraging established encryption techniques, incorporating additional security measures, and prioritizing user experience, the algorithm offers a compelling solution for securing sensitive communication channels in various contexts. Whether used for personal communication, data protection, or secure information exchange, the algorithm stands as a testament to the principles of effective cryptography and the pursuit of digital security. Its design ensures that users can confidently encrypt and decrypt their messages, knowing that their information is well-protected against unauthorized access and potential attacks.

8. Source Code

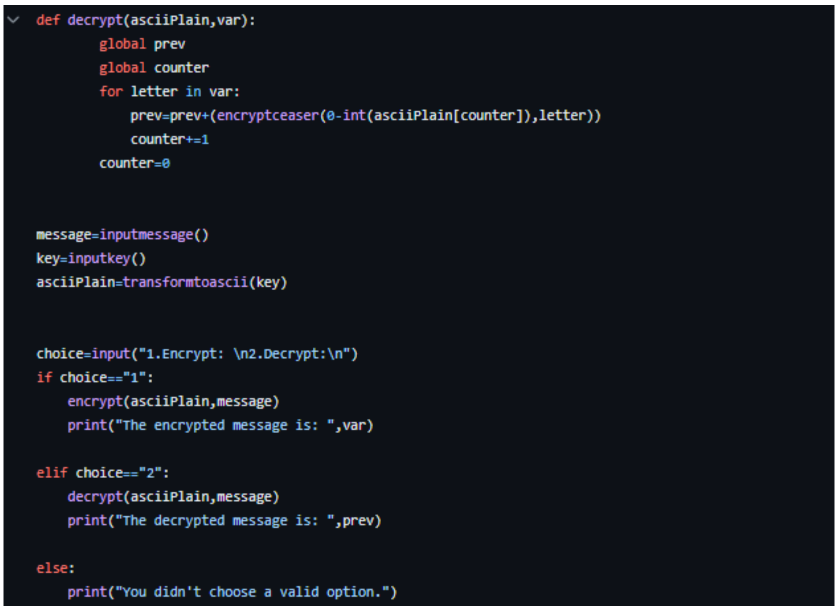
# Python code for the Hybrid Cipher



```python
import base64
import itertools
#------------------------------------------
var=""
prev=""
counter=0
def encryptceaser(key, message): #ABCAHMAD
    message = message.upper()
    alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    result = ""

    for letter in message:
        if letter in alpha:
            letter_index = (alpha.find(letter) + key) % len(alpha)

            result = result + alpha[letter_index]
        else:
            result = result + letter

    return result

def inputmessage():

    message = input("Enter the message you want to encrypt / decrypt: ")
    return message

def inputkey():

    key=input("Enter your key: ")

    key=key.encode()
    key=base64.b64encode(key)
    key=key.decode()
    return key
```



```python
def transformtoascii(key):

    list=[ord(c) for c in key]
    asciiPlain=""

    #print(list)
    newlist=[]
    #fixing the problem that occurs when the key is shorter than the message so it duplicates it
    if len(list) < len(message.replace(" ","")):
        cycler=itertools.cycle(list)
        for item in cycler:
            newlist.append(item)
            if len(newlist)==len(message):
                break
        for i in newlist:
            asciiPlain+=str(i)

    else:

        for i in list:
            asciiPlain+=str(i)
    return asciiPlain

def encrypt(asciiPlain,message):
        for letter in message:
            global var
            global counter
            var+=encryptceaser(int(asciiPlain[counter]),letter)
            counter+=1
        counter=0
        return var
```



```python
def decrypt(asciiPlain,var):
        global prev
        global counter
        for letter in var:
            prev=prev+(encryptceaser(0-int(asciiPlain[counter]),letter))
            counter+=1
        counter=0


message=inputmessage()
key=inputkey()
asciiPlain=transformtoascii(key)


choice=input("1.Encrypt: \n2.Decrypt:\n")
if choice=="1":
    encrypt(asciiPlain,message)
    print("The encrypted message is: ",var)

elif choice=="2":
    decrypt(asciiPlain,message)
    print("The decrypted message is: ",prev)

else:
    print("You didn't choose a valid option.")
```

9. References

1. Symmetric Key Algorithms: https://en.wikipedia.org/wiki/Symmetric-key_algorithm
2. Stream ciphers: https://en.wikipedia.org/wiki/Stream_cipher
3. Caser Cipher : https://en.wikipedia.org/wiki/Caesar_cipher
4. Base-64 encoding: https://en.wikipedia.org/wiki/Base64
5. ASCII (American Standard Code for Information Interchange): https://en.wikipedia.org/wiki/ASCII