NUMBER THEORY & CRYPTOGRAPHY ASSIGNMENT - 02

CHACHA20 ALGORITHM (STREAM CIPHER)

SAURAV S
B210573CS

Introduction

The Chacha20 Algorithm:

- ChaCha20 is a symmetric encryption algorithm that uses a 256-bit key for both encryption and decryption.
- It was developed by Daniel J. Bernstein, a renowned cryptographer, in 2008 as a stream cipher.
- The ChaCha20 encryption algorithm is designed to provide a combination of speed and security.
- ChaCha20 is designed to provide a high level of security. Its resistance against known cryptanalytic attacks, such as differential and linear cryptanalysis, contributes to its reputation as a secure symmetric encryption algorithm.
- It has 20 rounds of key-diffusion, resulting in a keystream of 64(512) completely random bytes(bits).
- ChaCha20 is used in various security protocols and applications, including the Transport Layer Security (TLS) protocol, where it is often employed as a stream cipher in combination with the Poly1305 authenticator.

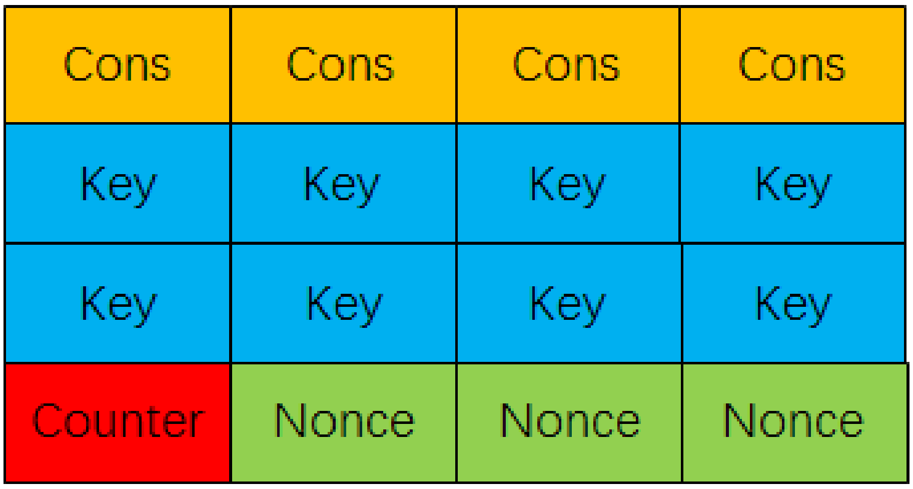
ChaCha20, along with its predecessor Salsa20, has gained widespread adoption and is considered a reliable and secure choice for symmetric encryption in modern cryptographic applications. It is often chosen for its simplicity, performance, and strong security properties.

Components of Chacha20

Initialization:

The ChaCha20 state is initialized as follows:

- The first four words (0-3) are constants: 0x61707865, 0x3320646e, 0x79622d32, 0x6b206574.
- The next eight words (4-11) are taken from the 256-bit key by reading the bytes in little-endian order, in 4-byte chunks.
- Word 12 is a block counter. Since each block is 64-byte, a 32-bit word is enough for 256 gigabytes of data.
- Words 13-15 are a nonce, which should not be repeated for the same key.



Cons	Cons	Cons	Cons
Key	Key	Key	Key
Key	Key	Key	Key
Counter	Nonce	Nonce	Nonce

Chacha Quarter Round:

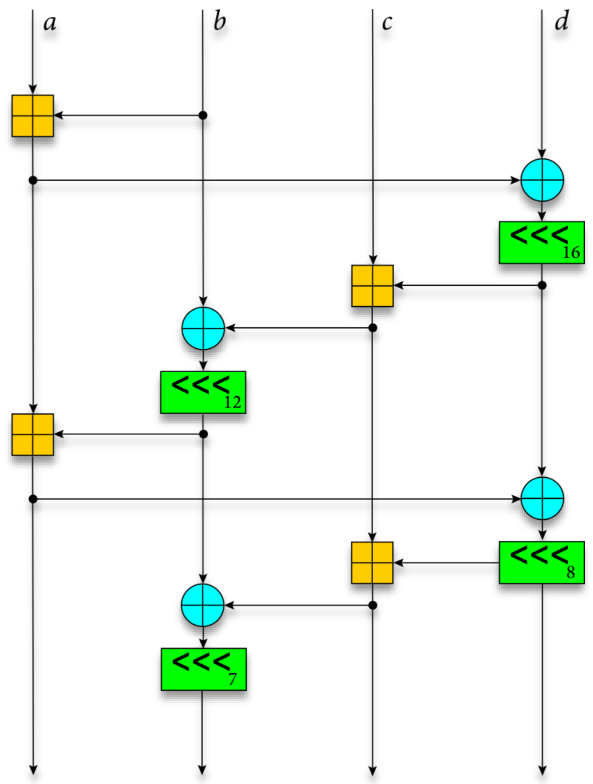
The basic operation of the Chacha algorithm is the quarter round. It operates on four 32-bit unsigned integers, denoted a, b, c, and d.

The operation is as follows:

```
QUARTERROUND (a,b,c,d):
    a += b; d ^= a; d <<<= 16;
    c += d; b ^= c; b <<<= 12;
    a += b; d ^= a; d <<<= 8;
    c += d; b ^= c; b <<<= 7;
```

Where "+" denotes integer addition modulo 2^32, "^" denotes a bitwise Exclusive OR (XOR), and "<<< n" denotes an n-bit left rotation (towards the high bits).

Quarter-Round Visual Representation:



ChaCha20 Block Function:

The Chacha block function transforms a Chacha state by running multiple quarter rounds. The output is 64 random-looking bytes.

The function first initializes the state, then runs 20 rounds, alternating between "column rounds" and “diagonal rounds".

Each round consists of four quarter-rounds, and they are run as follows. Quarter rounds 1-4 are part of a "column" round, while 5-8 are part of a "diagonal" round:

1. QUARTERROUND (0, 4, 8,12)
2. QUARTERROUND (1, 5, 9,13)
3. QUARTERROUND (2, 6,10,14)
4. QUARTERROUND (3, 7,11,15)
5. QUARTERROUND (0, 5,10,15)
6. QUARTERROUND (1, 6,11,12)
7. QUARTERROUND (2, 7, 8,13)
8. QUARTERROUND (3, 4, 9,14)

Keystream Generation:

At the end of 20 rounds (or 10 iterations of the above list), we add the original input words to the output words, and serialize the result by sequencing the words one-by-one in little-endian order.

Pseudocode:

```
chacha20_block(key, counter, nonce):
    state = Initialization(Constants, key, counter, nonce)
    working_state = state
    for i=1 upto 10
        QUARTERROUND(working_state, 0, 4, 8,12)
        QUARTERROUND(working_state, 1, 5, 9,13)
        QUARTERROUND(working_state, 2, 6,10,14)
        QUARTERROUND(working_state, 3, 7,11,15)
        QUARTERROUND(working_state, 0, 5,10,15)
        QUARTERROUND(working_state, 1, 6,11,12)
        QUARTERROUND(working_state, 2, 7, 8,13)
        QUARTERROUND(working_state, 3, 4, 9,14)
    end
    state += working_state
    return serialize(state)
end
```

ChaCha20 Encryption Algorithm:

ChaCha20 successively calls the block function, with the same key and nonce, and with successively increasing block counter parameters. ChaCha20 then serializes the resulting state by writing the numbers in little-endian order, creating a keystream block.

Concatenating the keystream blocks from the successive blocks forms a keystream. The ChaCha20 function then performs an XOR of this keystream with the plaintext. Alternatively, each keystream block can be XORed with a plaintext block before proceeding to create the next block, saving some memory. There is no requirement for the plaintext to be an integral multiple of 512 bits. If there is extra keystream from the last block, it is discarded.

The inputs to ChaCha20 are:

- A 256-bit key.
- A 96-bit nonce. In some protocols, this is known as the Initialization Vector (IV).
- An arbitrary-length plaintext.

The output is an encrypted message, or "ciphertext", of the same length.

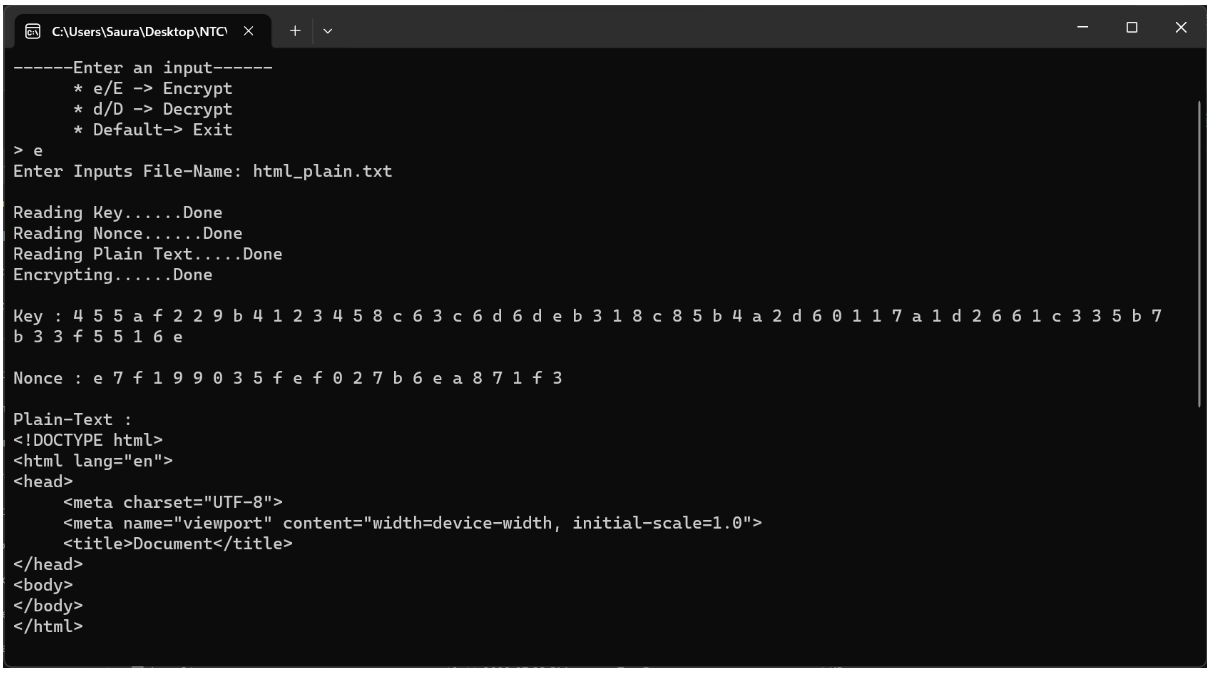
Decryption is done in the same way. The ChaCha20 block function is used to expand the key into a keystream, which is XORed with the ciphertext giving back the plaintext.

Pseudocode:

```
chacha20_encrypt(key, nonce, plaintext)
    counter = 1
    for j = 0 upto floor(len(plaintext)/64)-1
        key_stream = chacha20_block(key, counter+j, nonce)
        block = plaintext[(j*64)..(j*64+63)]
        encrypted_message +=  block ^ key_stream
    end
    if ((len(plaintext) % 64) != 0)
           j = floor(len(plaintext)/64)
           key_stream = chacha20_block(key, counter+j, nonce)
           block = plaintext[(j*64)..len(plaintext)-1]
           encrypted_message += (block^key_stream)[0..len(plaintext)%64]
    end
    return encrypted_message
end
```

Implementation:

Encryption:



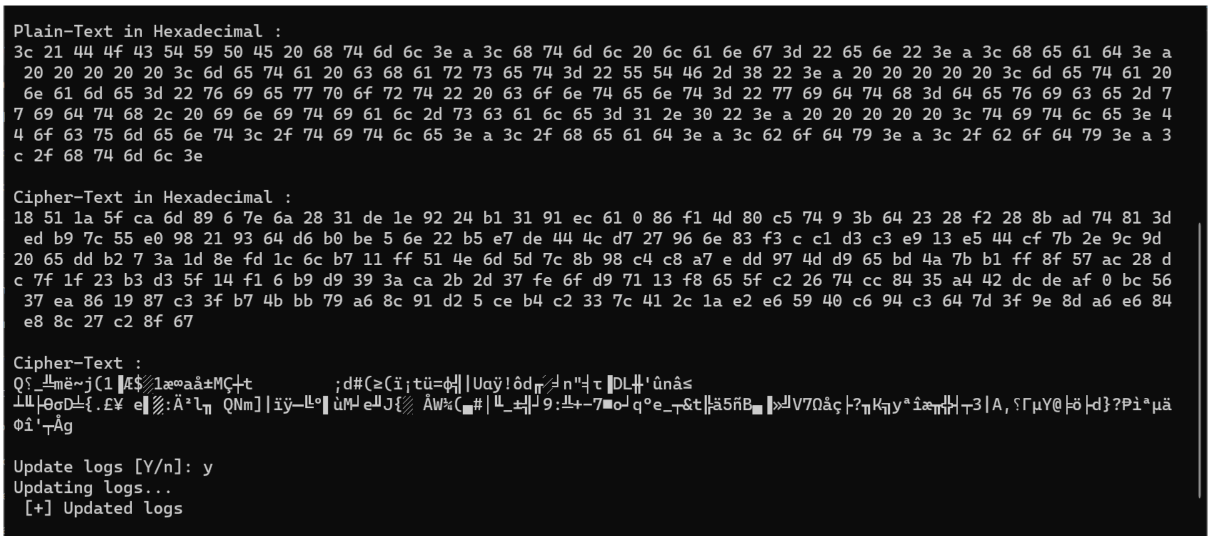
```
------Enter an input------
      * e/E -> Encrypt
      * d/D -> Decrypt
      * Default-> Exit
> e
Enter Inputs File-Name: html_plain.txt

Reading Key......Done
Reading Nonce......Done
Reading Plain Text.....Done
Encrypting......Done

Key : 4 5 5 a f 2 2 9 b 4 1 2 3 4 5 8 c 6 3 c 6 d 6 d e b 3 1 8 c 8 5 b 4 a 2 d 6 0 1 1 7 a 1 d 2 6 6 1 c 3 3 5 b 7
b 3 3 f 5 5 1 6 e

Nonce : e 7 f 1 9 9 0 3 5 f e f 0 2 7 b 6 e a 8 7 1 f 3

Plain-Text :
<!DOCTYPE html>
<html lang="en">
<head>
     <meta charset="UTF-8">
     <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <title>Document</title>
</head>
<body>
</body>
</html>
```



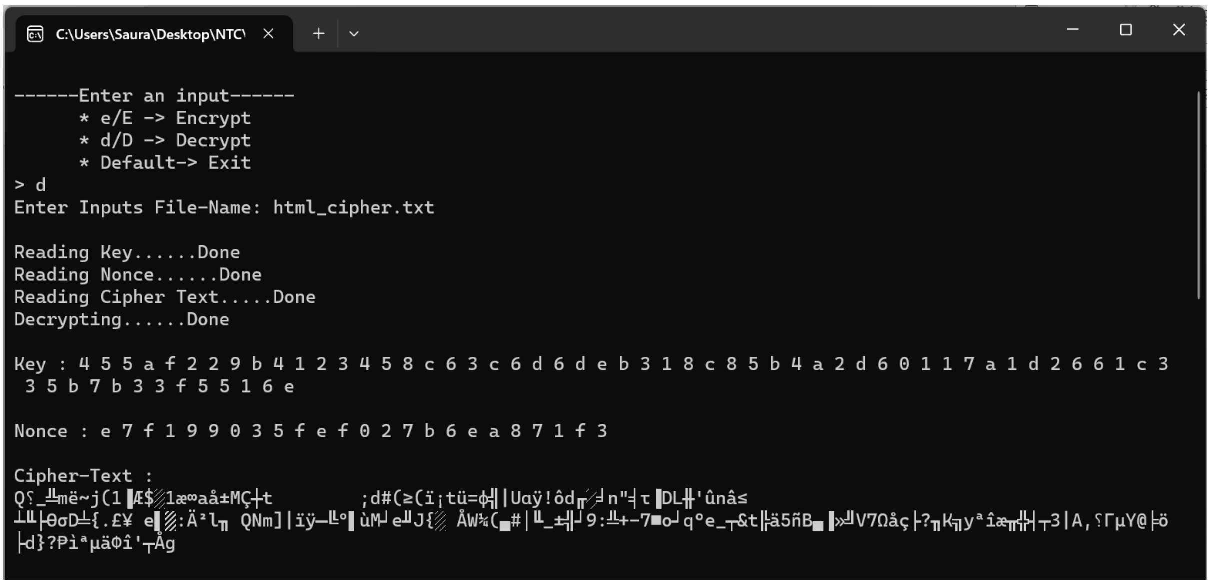
```
Plain-Text in Hexadecimal :
3c 21 44 4f 43 54 59 50 45 20 68 74 6d 6c 3e a 3c 68 74 6d 6c 20 6c 61 6e 67 3d 22 65 6e 22 3e a 3c 68 65 61 64 3e a
 20 20 20 20 20 3c 6d 65 74 61 20 63 68 61 72 73 65 74 3d 22 55 54 46 2d 38 22 3e a 20 20 20 20 20 3c 6d 65 74 61 20
 6e 61 6d 65 3d 22 76 69 65 77 70 6f 72 74 22 20 63 6f 6e 74 65 6e 74 3d 22 77 69 64 74 68 3d 64 65 76 69 63 65 2d 7
7 69 64 74 68 2c 20 69 6e 69 74 69 61 6c 2d 73 63 61 6c 65 3d 31 2e 30 22 3e a 20 20 20 20 20 3c 74 69 74 6c 65 3e 4
4 6f 63 75 6d 65 6e 74 3c 2f 74 69 74 6c 65 3e a 3c 2f 68 65 61 64 3e a 3c 62 6f 64 79 3e a 3c 2f 62 6f 64 79 3e a 3
c 2f 68 74 6d 6c 3e

Cipher-Text in Hexadecimal :
18 51 1a 5f ca 6d 89 6 7e 6a 28 31 de 1e 92 24 b1 31 91 ec 61 0 86 f1 4d 80 c5 74 9 3b 64 23 28 f2 28 8b ad 74 81 3d
 ed b9 7c 55 e0 98 21 93 64 d6 b0 be 5 6e 22 b5 e7 de 44 4c d7 27 96 6e 83 f3 c c1 d3 c3 e9 13 e5 44 cf 7b 2e 9c 9d
20 65 dd b2 7 3a 1d 8e fd 1c 6c b7 11 ff 51 4e 6d 5d 7c 8b 98 c4 c8 a7 e dd 97 4d d9 65 bd 4a 7b b1 ff 8f 57 ac 28 d
c 7f 1f 23 b3 d3 5f 14 f1 6 b9 d9 39 3a ca 2b 2d 37 fe 6f d9 71 13 f8 65 5f c2 26 74 cc 84 35 a4 42 dc de af 0 bc 56
 37 ea 86 19 87 c3 3f b7 4b bb 79 a6 8c 91 d2 5 ce b4 c2 33 7c 41 2c 1a e2 e6 59 40 c6 94 c3 64 7d 3f 9e 8d a6 e6 84
 e8 8c 27 c2 8f 67

Cipher-Text :
Q∟_╩më~j(1▐Æ$▒1æ∞aå±MÇ┼t         ;d#(≥(ï¡tü=φ╣|Uαÿ!ôd╓▒╛n"╡τ▐DL╫'ûnâ≤
┴╨├ΘσD╧{.£¥ e▌▒:Ä²l╖ QNm]|ïÿ─╙°▌ùM┘e╝J{▒ ÅW¾(▄#|╙_±╣┘9:╩+-7■o┘q°e_┬&t╠ä5ñB▄▐»╝V7Ωåç├?╖K╗yªîæ╦╬┤┬3|A,∟ΓµY@╞ö├d}?₧ìªµä
Φî'┬Åg

Update logs [Y/n]: y
Updating logs...
 [+] Updated logs
```

Decryption:



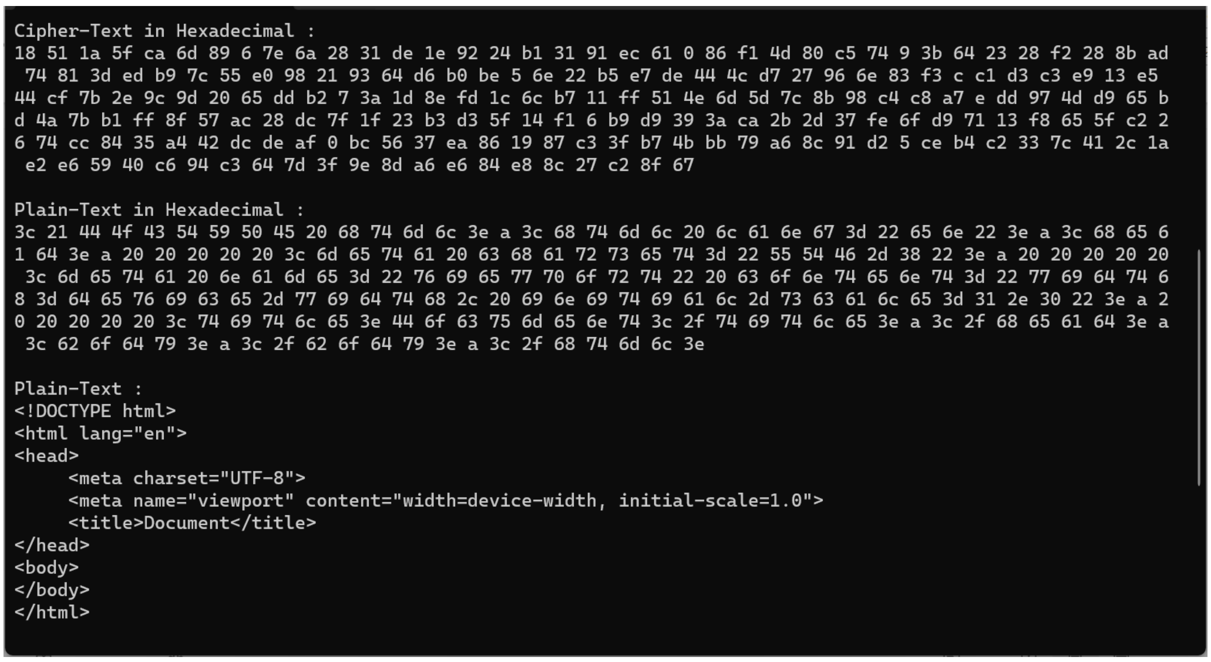
```
------Enter an input------
      * e/E -> Encrypt
      * d/D -> Decrypt
      * Default-> Exit
> d
Enter Inputs File-Name: html_cipher.txt

Reading Key......Done
Reading Nonce......Done
Reading Cipher Text.....Done
Decrypting......Done

Key : 4 5 5 a f 2 2 9 b 4 1 2 3 4 5 8 c 6 3 c 6 d 6 d e b 3 1 8 c 8 5 b 4 a 2 d 6 0 1 1 7 a 1 d 2 6 6 1 c 3
 3 5 b 7 b 3 3 f 5 5 1 6 e

Nonce : e 7 f 1 9 9 0 3 5 f e f 0 2 7 b 6 e a 8 7 1 f 3

Cipher-Text :
Q؟_╨më~j(1▐Æ$▒1æ∞aå±MÇ┼t        ;d#(≥(ï¡tü=φ╣|Uαÿ!ôd╓▒╛n"╡τ▐DL╫'ûnâ≤
┴╙├ΘσD╧{.£¥ e▌▒:Ä²l╥ QNm]|ïÿ─╙º▌ùM┘e╜J{▒ ÅW¾(▄#|╙_±╢┘9:╨+-7■o┘q°e_┬&t╠ä5ñB▄▐»╜V7Ωåç├?╖K╗yªîæ╥╬┤┬3|A,؟ΓµY@╞ö
├d}?₧ìªµäΦî'┬Åg
```



```
Cipher-Text in Hexadecimal :
18 51 1a 5f ca 6d 89 6 7e 6a 28 31 de 1e 92 24 b1 31 91 ec 61 0 86 f1 4d 80 c5 74 9 3b 64 23 28 f2 28 8b ad
 74 81 3d ed b9 7c 55 e0 98 21 93 64 d6 b0 be 5 6e 22 b5 e7 de 44 4c d7 27 96 6e 83 f3 c c1 d3 c3 e9 13 e5
44 cf 7b 2e 9c 9d 20 65 dd b2 7 3a 1d 8e fd 1c 6c b7 11 ff 51 4e 6d 5d 7c 8b 98 c4 c8 a7 e dd 97 4d d9 65 b
d 4a 7b b1 ff 8f 57 ac 28 dc 7f 1f 23 b3 d3 5f 14 f1 6 b9 d9 39 3a ca 2b 2d 37 fe 6f d9 71 13 f8 65 5f c2 2
6 74 cc 84 35 a4 42 dc de af 0 bc 56 37 ea 86 19 87 c3 3f b7 4b bb 79 a6 8c 91 d2 5 ce b4 c2 33 7c 41 2c 1a
 e2 e6 59 40 c6 94 c3 64 7d 3f 9e 8d a6 e6 84 e8 8c 27 c2 8f 67

Plain-Text in Hexadecimal :
3c 21 44 4f 43 54 59 50 45 20 68 74 6d 6c 3e a 3c 68 74 6d 6c 20 6c 61 6e 67 3d 22 65 6e 22 3e a 3c 68 65 6
1 64 3e a 20 20 20 20 20 3c 6d 65 74 61 20 63 68 61 72 73 65 74 3d 22 55 54 46 2d 38 22 3e a 20 20 20 20 20
 3c 6d 65 74 61 20 6e 61 6d 65 3d 22 76 69 65 77 70 6f 72 74 22 20 63 6f 6e 74 65 6e 74 3d 22 77 69 64 74 6
8 3d 64 65 76 69 63 65 2d 77 69 64 74 68 2c 20 69 6e 69 74 69 61 6c 2d 73 63 61 6c 65 3d 31 2e 30 22 3e a 2
0 20 20 20 20 3c 74 69 74 6c 65 3e 44 6f 63 75 6d 65 6e 74 3c 2f 74 69 74 6c 65 3e a 3c 2f 68 65 61 64 3e a
 3c 62 6f 64 79 3e a 3c 2f 62 6f 64 79 3e a 3c 2f 68 74 6d 6c 3e

Plain-Text :
<!DOCTYPE html>
<html lang="en">
<head>
     <meta charset="UTF-8">
     <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <title>Document</title>
</head>
<body>
</body>
</html>
```

References:

- https://datatracker.ietf.org/doc/html/rfc7539#
- https://xilinx.github.io/Vitis_Libraries/security/2019.2/guide_L1/internals/chacha20.html
- https://en.wikipedia.org/wiki/ChaCha20-Poly1305
- https://youtu.be/UeIpq-C-GSA?si=MQvJ5H7OACpg0y3g