Dilnoza Saidova
TCSS481: Computer Security
October 13, 2022

Stream Ciphers

1) Chacha20 Algorithm

Chacha20 is a cipher stream the inputs of which include 256-bit key, 32-bit counter, 96-bit nonce, and plain text. The initial state of this cipher stream is a 4x4 matrix, which invertedly transforms the matrix *r* rounds, adding the result to the original matrix with a resulting 16-word (or 64-byte) output block. The first row of such matrix is a constant string, the second and third rows are filled with 256-bit key, the first word in the last row is a 32-bit counter, and the rest of the matrix contents are 96-bit nonce. The cipher stream generates 512-bit keystream for each iteration to encrypt a 512-bit plain text block.

Constant	Constant	Constant	Constant
Key	Key	Key	Key
Key	Key	Key	Key
Counter (input)	Nonce (input)	Nonce (input)	Nonce (input)

In the first round of Chacha20, the keys added to constants are **xor-**ed into inputs, then adding the results into keys, which is then repeated. In other words, encryption involves generation of a new 512-bit key **xor-**ed with 512-bit plain text, outputting a cipher block after each iteration. Chacha20 uses 4 additions, 4 **xor**s, and 4 rotations to invertedly update 4 32-bit state words, updating each word twice. The quarter-rounds of Chacha20 diffuse changes through bits quickly, compared to its predecessor Salsa20 which is much slower. Keystream of Chacha20 is formed through concatenation of the keystream blocks from the previous blocks. Then, the produced keystream is **xor**-ed with plaintext, with any extra keystream from previous block discarded.

Given below is the Chacha20 block function and encryption algorithms in Pseudo-Code:

Block Function

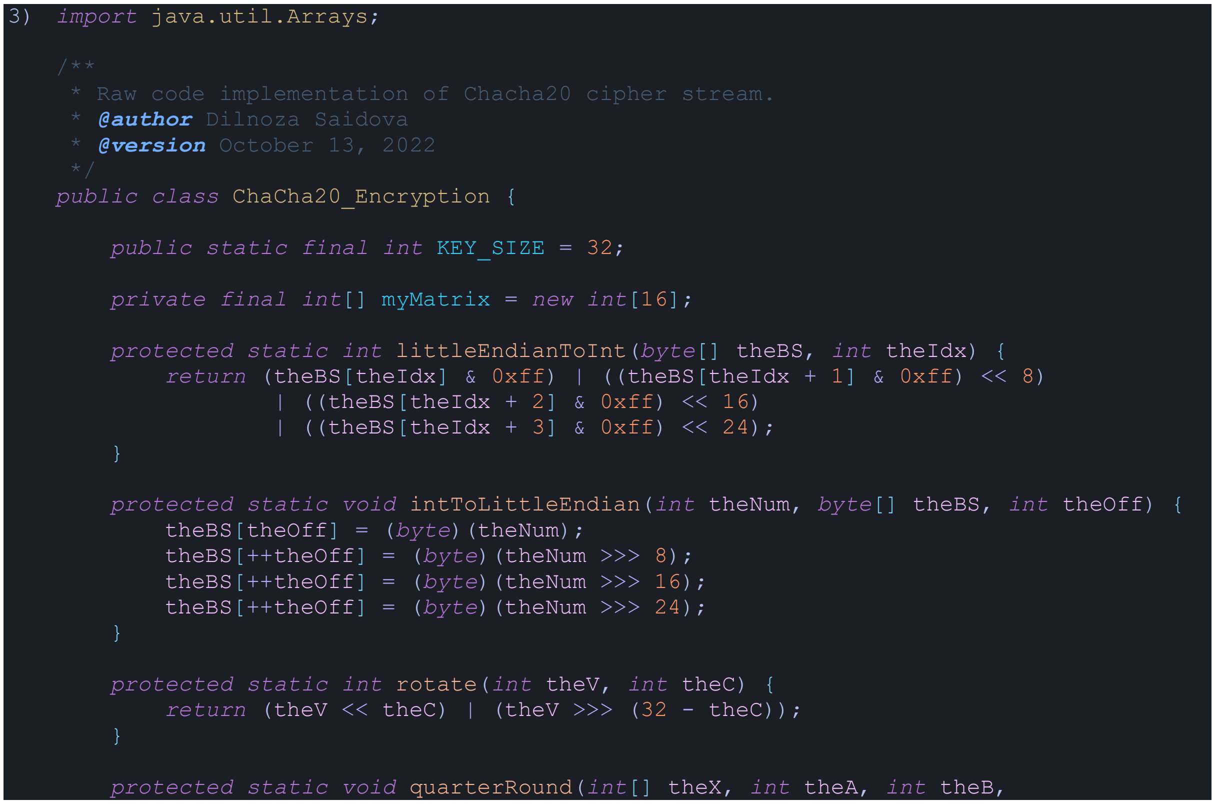
```
inner_block (state):
         Qround(state, 0, 4, 8,12)
         Qround(state, 1, 5, 9,13)
         Qround(state, 2, 6,10,14)
         Qround(state, 3, 7,11,15)
         Qround(state, 0, 5,10,15)
         Qround(state, 1, 6,11,12)
         Qround(state, 2, 7, 8,13)
         Qround(state, 3, 4, 9,14)
         end

      chacha20_block(key, counter, nonce):
         state = constants | key | counter | nonce
         working_state = state
         for i=1 upto 10
            inner_block(working_state)
            end
         state += working_state
         return serialize(state)
         end
```

Encryption

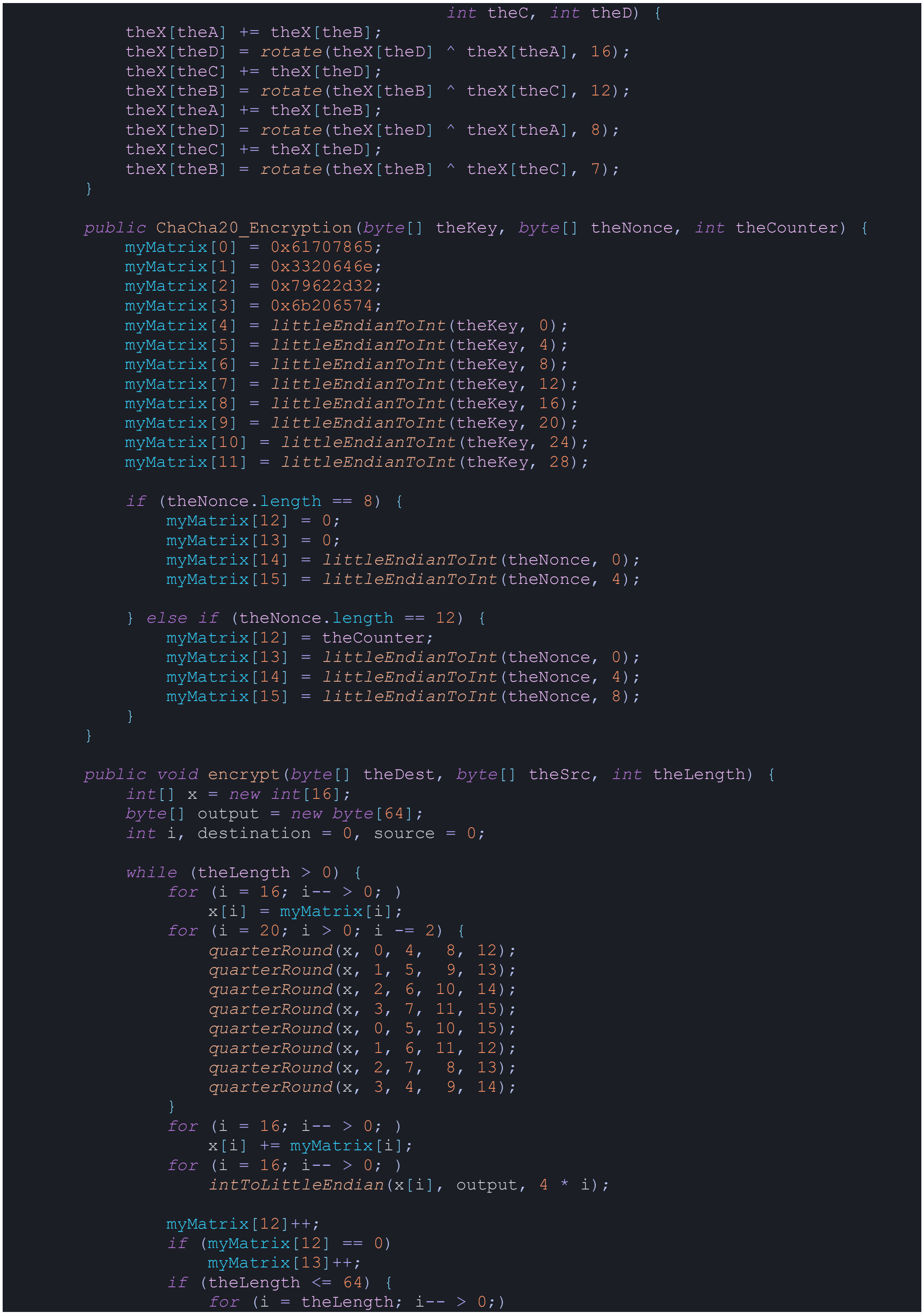
```
chacha20_encrypt(key, counter, nonce, plaintext):
        for counter=1 upto ceil(len(plaintext) / 64)
           key_stream = chacha20_block(key, counter, nonce)
           block = plaintext[((counter-1)*64)..(counter*64-1)]
           encrypted_message +=  block ^ key_stream
           end
   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

In the given Chacha, the inputs include 256-bit key, 32-bit initial counter, 96-bit nonce (IV), and arbitrary-length plaintext. The output is a ciphertext – encrypted message, which is the same length as the original plaintext.

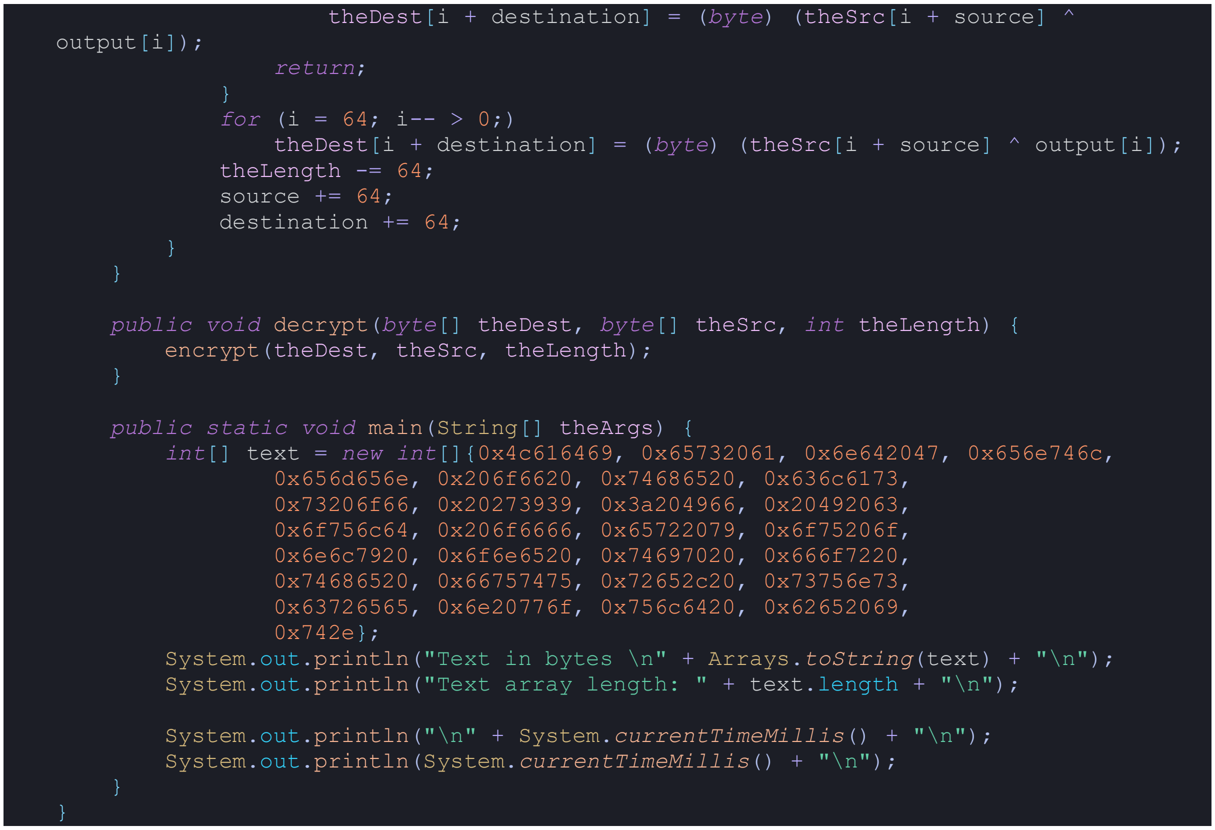
2) Code Implementing of Chacha20



```
3)  import java.util.Arrays;

    /**
     * Raw code implementation of Chacha20 cipher stream.
     * @author Dilnoza Saidova
     * @version October 13, 2022
     */
    public class ChaCha20_Encryption {

        public static final int KEY_SIZE = 32;

        private final int[] myMatrix = new int[16];

        protected static int littleEndianToInt(byte[] theBS, int theIdx) {
            return (theBS[theIdx] & 0xff) | ((theBS[theIdx + 1] & 0xff) << 8)
                    | ((theBS[theIdx + 2] & 0xff) << 16)
                    | ((theBS[theIdx + 3] & 0xff) << 24);
        }

        protected static void intToLittleEndian(int theNum, byte[] theBS, int theOff) {
            theBS[theOff] = (byte)(theNum);
            theBS[++theOff] = (byte)(theNum >>> 8);
            theBS[++theOff] = (byte)(theNum >>> 16);
            theBS[++theOff] = (byte)(theNum >>> 24);
        }

        protected static int rotate(int theV, int theC) {
            return (theV << theC) | (theV >>> (32 - theC));
        }

        protected static void quarterRound(int[] theX, int theA, int theB,
```

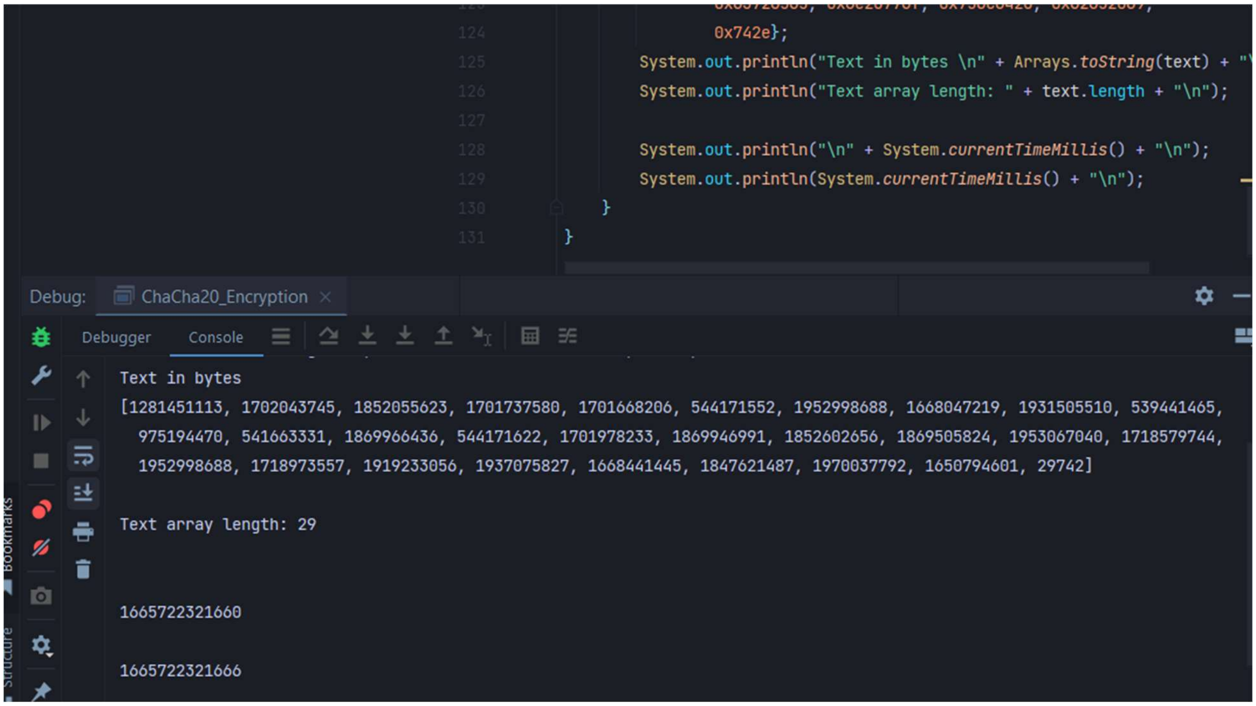


```
                                       int theC, int theD) {
        theX[theA] += theX[theB];
        theX[theD] = rotate(theX[theD] ^ theX[theA], 16);
        theX[theC] += theX[theD];
        theX[theB] = rotate(theX[theB] ^ theX[theC], 12);
        theX[theA] += theX[theB];
        theX[theD] = rotate(theX[theD] ^ theX[theA], 8);
        theX[theC] += theX[theD];
        theX[theB] = rotate(theX[theB] ^ theX[theC], 7);
    }

    public ChaCha20_Encryption(byte[] theKey, byte[] theNonce, int theCounter) {
        myMatrix[0] = 0x61707865;
        myMatrix[1] = 0x3320646e;
        myMatrix[2] = 0x79622d32;
        myMatrix[3] = 0x6b206574;
        myMatrix[4] = littleEndianToInt(theKey, 0);
        myMatrix[5] = littleEndianToInt(theKey, 4);
        myMatrix[6] = littleEndianToInt(theKey, 8);
        myMatrix[7] = littleEndianToInt(theKey, 12);
        myMatrix[8] = littleEndianToInt(theKey, 16);
        myMatrix[9] = littleEndianToInt(theKey, 20);
        myMatrix[10] = littleEndianToInt(theKey, 24);
        myMatrix[11] = littleEndianToInt(theKey, 28);

        if (theNonce.length == 8) {
            myMatrix[12] = 0;
            myMatrix[13] = 0;
            myMatrix[14] = littleEndianToInt(theNonce, 0);
            myMatrix[15] = littleEndianToInt(theNonce, 4);

        } else if (theNonce.length == 12) {
            myMatrix[12] = theCounter;
            myMatrix[13] = littleEndianToInt(theNonce, 0);
            myMatrix[14] = littleEndianToInt(theNonce, 4);
            myMatrix[15] = littleEndianToInt(theNonce, 8);
        }
    }

    public void encrypt(byte[] theDest, byte[] theSrc, int theLength) {
        int[] x = new int[16];
        byte[] output = new byte[64];
        int i, destination = 0, source = 0;

        while (theLength > 0) {
            for (i = 16; i-- > 0; )
                x[i] = myMatrix[i];
            for (i = 20; i > 0; i -= 2) {
                quarterRound(x, 0, 4,  8, 12);
                quarterRound(x, 1, 5,  9, 13);
                quarterRound(x, 2, 6, 10, 14);
                quarterRound(x, 3, 7, 11, 15);
                quarterRound(x, 0, 5, 10, 15);
                quarterRound(x, 1, 6, 11, 12);
                quarterRound(x, 2, 7,  8, 13);
                quarterRound(x, 3, 4,  9, 14);
            }
            for (i = 16; i-- > 0; )
                x[i] += myMatrix[i];
            for (i = 16; i-- > 0; )
                intToLittleEndian(x[i], output, 4 * i);

            myMatrix[12]++;
            if (myMatrix[12] == 0)
                myMatrix[13]++;
            if (theLength <= 64) {
                for (i = theLength; i-- > 0;)
```



```
                    theDest[i + destination] = (byte) (theSrc[i + source] ^
output[i]);
                    return;
                }
                for (i = 64; i-- > 0;)
                    theDest[i + destination] = (byte) (theSrc[i + source] ^ output[i]);
                theLength -= 64;
                source += 64;
                destination += 64;
            }
        }

        public void decrypt(byte[] theDest, byte[] theSrc, int theLength) {
            encrypt(theDest, theSrc, theLength);
        }

        public static void main(String[] theArgs) {
            int[] text = new int[]{0x4c616469, 0x65732061, 0x6e642047, 0x656e746c,
                    0x656d656e, 0x206f6620, 0x74686520, 0x636c6173,
                    0x73206f66, 0x20273939, 0x3a204966, 0x20492063,
                    0x6f756c64, 0x206f6666, 0x65722079, 0x6f75206f,
                    0x6e6c7920, 0x6f6e6520, 0x74697020, 0x666f7220,
                    0x74686520, 0x66757475, 0x72652c20, 0x73756e73,
                    0x63726565, 0x6e20776f, 0x756c6420, 0x62652069,
                    0x742e};
            System.out.println("Text in bytes \n" + Arrays.toString(text) + "\n");
            System.out.println("Text array length: " + text.length + "\n");

            System.out.println("\n" + System.currentTimeMillis() + "\n");
            System.out.println(System.currentTimeMillis() + "\n");
        }
    }
```

4) Screenshots of running Chacha20 and generating key stream.



```
                0x742e};
            System.out.println("Text in bytes \n" + Arrays.toString(text) + "\
            System.out.println("Text array length: " + text.length + "\n");

            System.out.println("\n" + System.currentTimeMillis() + "\n");
            System.out.println(System.currentTimeMillis() + "\n");
        }
    }
```

```
Text in bytes
[1281451113, 1702043745, 1852055623, 1701737580, 1701668206, 544171552, 1952998688, 1668047219, 1931505510, 539441465,
 975194470, 541663331, 1869966436, 544171622, 1701978233, 1869946991, 1852602656, 1869505824, 1953067040, 1718579744,
 1952998688, 1718973557, 1919233056, 1937075827, 1668441445, 1847621487, 1970037792, 1650794601, 29742]

Text array length: 29


1665722321660

1665722321666
```

5) Measurement of how many bits of key stream material Chacha20 generates per second.
247 bit/ms or 247000 bit/sec