EE 720

Assignment 1

Trivium (cipher)

Group Members- T Pavan Kalyan (190020124)
Puranjay Datta (19D070048)
Badal Varshney (19D070015)
Chirag Meena (18D070008)

Introduction :-

Trivium is a hardware-oriented synchronous stream cipher. It was designed as an exercise in exploring how far a stream cipher can be simplified without sacrificing its security, speed or flexibility it designed to generate up to 2^{64} bits of key stream from an 80-bit secret key and an 80-bit initial value IV. As for most stream ciphers, this technique consists of two phases: first the internal state of the cipher is initialized using the key and the IV, after it state is repeatedly updated and used to generate key stream bits.

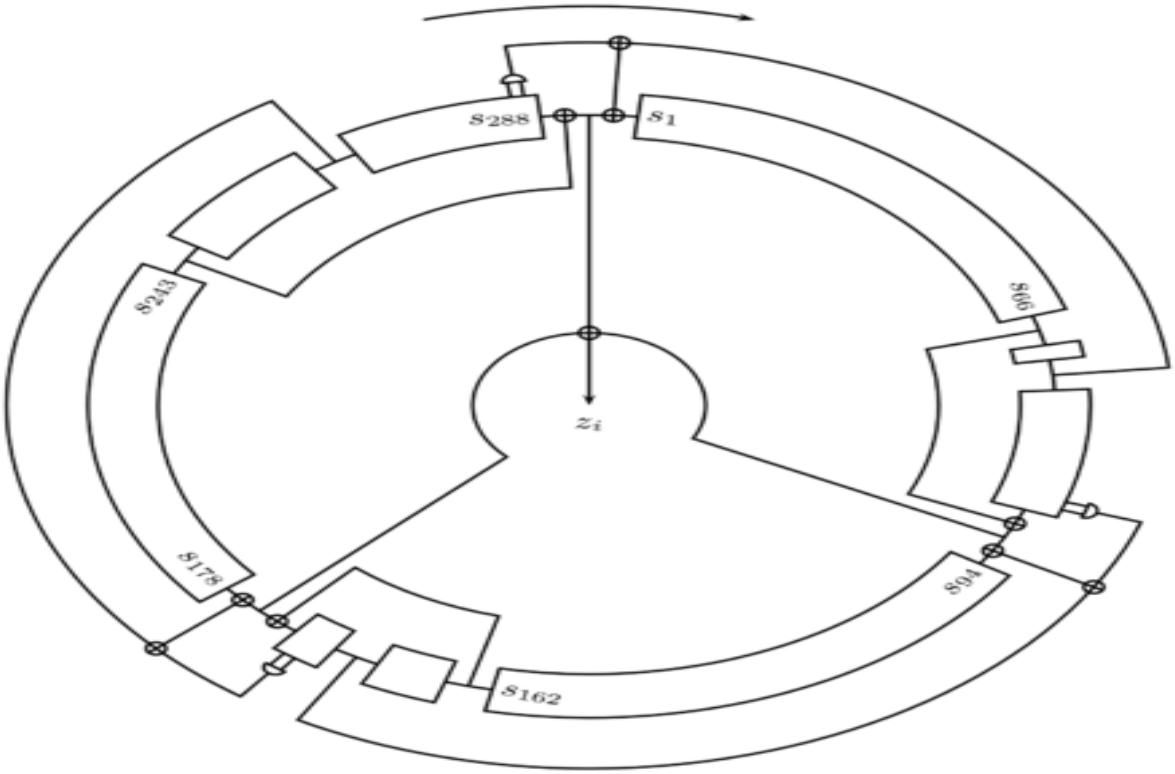
Key and IV setup :-

The algorithm is initialized by loading an 80-bit key and an 80-bit IV into the 288-bit initial state, and setting all remaining bits to 0, except forS_{286}, S_{287}, and S_{288}. Then, the state is rotated over 4 full cycles, in the same way as explained in introduction, but without generating key stream bits

Key stream generation : -

The proposed design contains a 288-bit internal state denoted by (S_1 , . . . , S_{288}). The key stream generation consists of an iterative process which extracts the values of 15 specific state bits and uses them both to update 3 bits of the state and to compute 1 bit of key stream zi . The state bits are then rotated and the process repeats itself until the requested N ≤ 2^64 bits of the key stream have been generated.

A graphical representation of the key stream generation process can be found in Fig. below.



Description:-

Trivium's 288-bit internal state consists of three shift registers of different lengths. At each round, a bit is shifted into each of the three shift registers using a non-linear combination of taps from that and one other register; one bit of output is produced. To initialize the cipher, the key and IV are written into two of the shift registers, with the remaining bits starting in a fixed pattern; the cipher state is then updated 4 × 288 = 1152 times, so that every bit of the internal state depends on every bit of the key and of the IV in a complex nonlinear way.

No taps appear on the first 65 bits of each shift register, so each novel state bit is not used until at least 65 rounds after it is generated. This is the key to Trivium's software performance and flexibility in hardware.

Trivium may be specified very concisely using three recursive equations. Each variable is an element of GF(2); they can be represented as bits, with "+" being XOR and "•" being AND.

- $a_i = c_{i-66} + c_{i-111} + c_{i-110} \cdot c_{i-109} + a_{i-69}$

- $b_i = a_{i-66} + a_{i-93} + a_{i-92} \cdot a_{i-91} + b_{i-78}$
- $c_i = b_{i-69} + b_{i-84} + b_{i-83} \cdot b_{i-82} + c_{i-87}$

The output bits $r_0 \ldots r_{2^{64}-1}$ are then generated by

- $r_i = c_{i-66} + c_{i-111} + a_{i-66} + a_{i-93} + b_{i-69} + b_{i-84}$

Given an 80-bit key $k_0 \ldots k_{79}$ and an l-bit IV $v_0 \ldots v_{l-1}$ (where $0 \le l \le 80$), Trivium is initialized as follows:

- $(a_{-1245} \ldots a_{-1153}) = (0, 0 \ldots 0, k_0 \ldots k_{79})$
- $(b_{-1236} \ldots b_{-1153}) = (0, 0 \ldots 0, v_0 \ldots v_{l-1})$
- $(c_{-1263} \ldots c_{-1153}) = (1, 1, 1, 0, 0 \ldots 0)$

The large negative indices on the initial values reflect the 1152 steps that must take place before output is produced.

To map a stream of bits r to a stream of bytes R, we use the little-endian mapping $R_i = \Sigma_{j=0 \ldots 7} 2^j r_{8i+j}$

New Trivium cipher Construction (Reduced States)

K 15 bits

IV 15 bits

S 68 bits

(s0, s2, . . . , s22) ← (K0, . . . , K14, 0, . . . , 0)

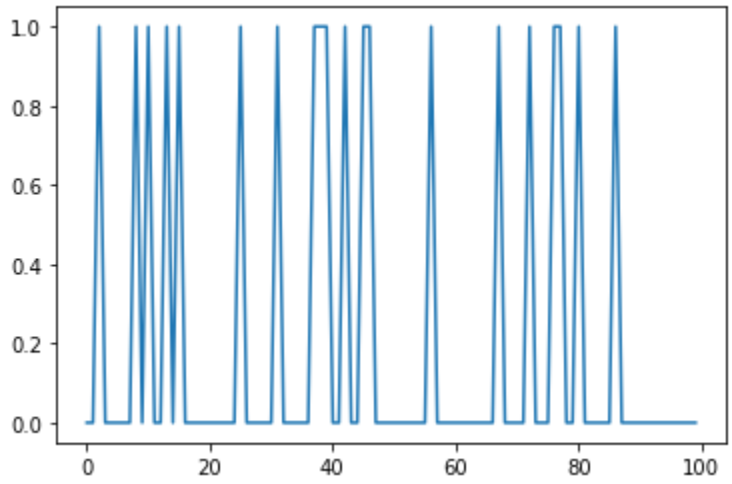
(s23, s25, . . . , s44) ← (IV0, . . . , IV14, 0, . . . , 0)

(s45, s39, . . . , 67) ← (0, . . . , 0, 1, 1, 1)

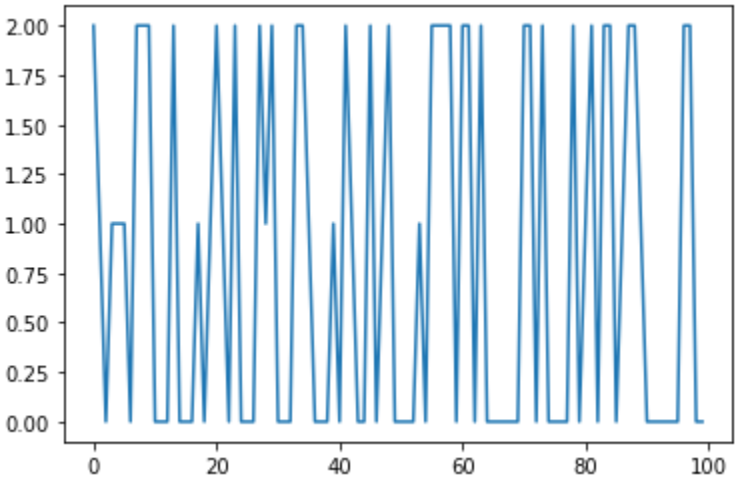
Conclusion-As the states(N) keep on increasing, the linear complexity fluctuations decrease and it saturates at N/2 for the berlekamp massey function, which is shown below using graphs.

Results

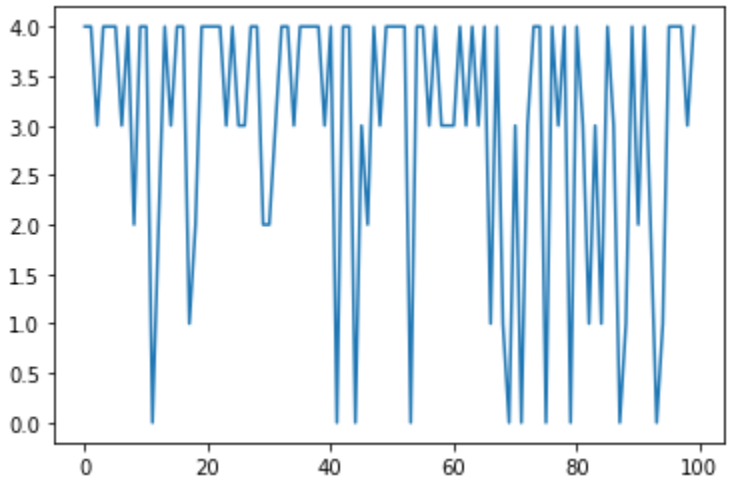
N=2



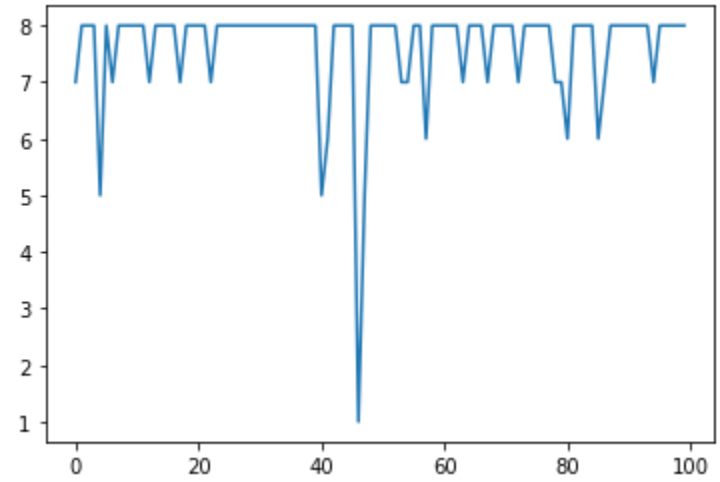
N=4



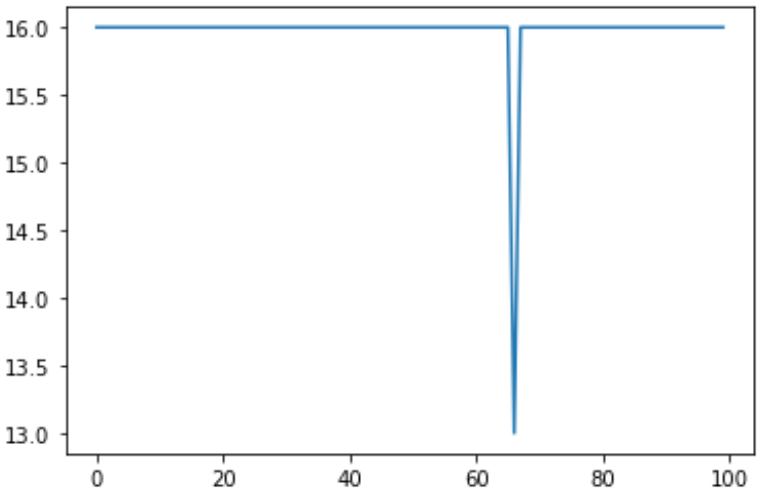
N=8


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100

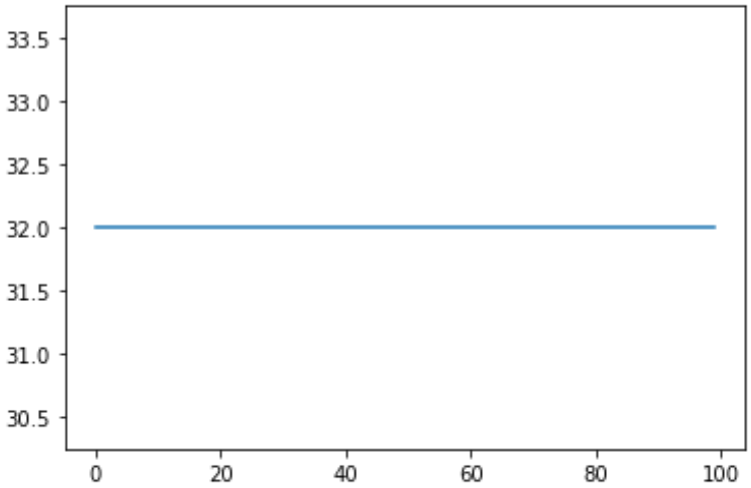
N=16


8
7
6
5
4
3
2
1
0
20
40
60
80
100

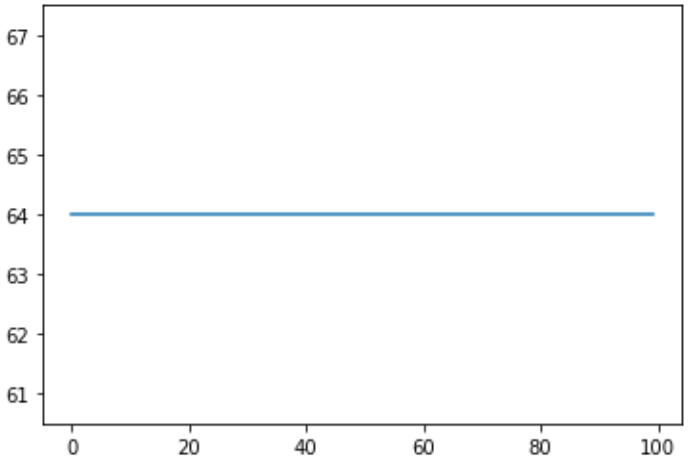
N=32


16.0
15.5
15.0
14.5
14.0
13.5
13.0
0
20
40
60
80
100

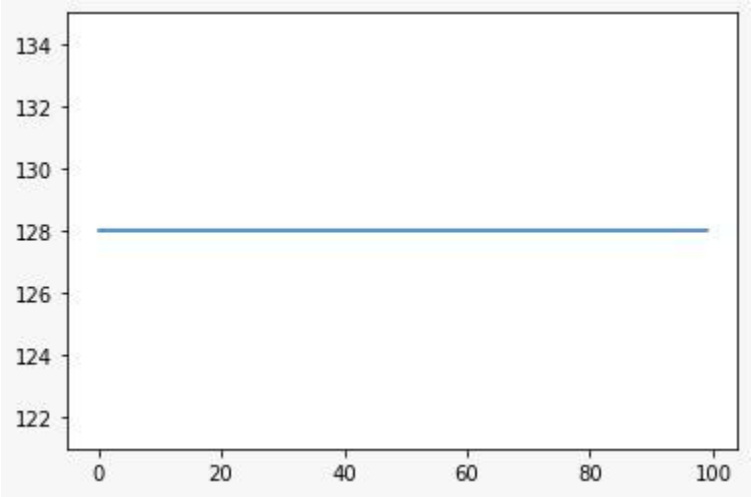
N=64


33.5
33.0
32.5
32.0
31.5
31.0
30.5
0
20
40
60
80
100

N=128


67
66
65
64
63
62
61
0
20
40
60
80
100

N=256


134
132
130
128
126
124
122
0
20
40
60
80
100