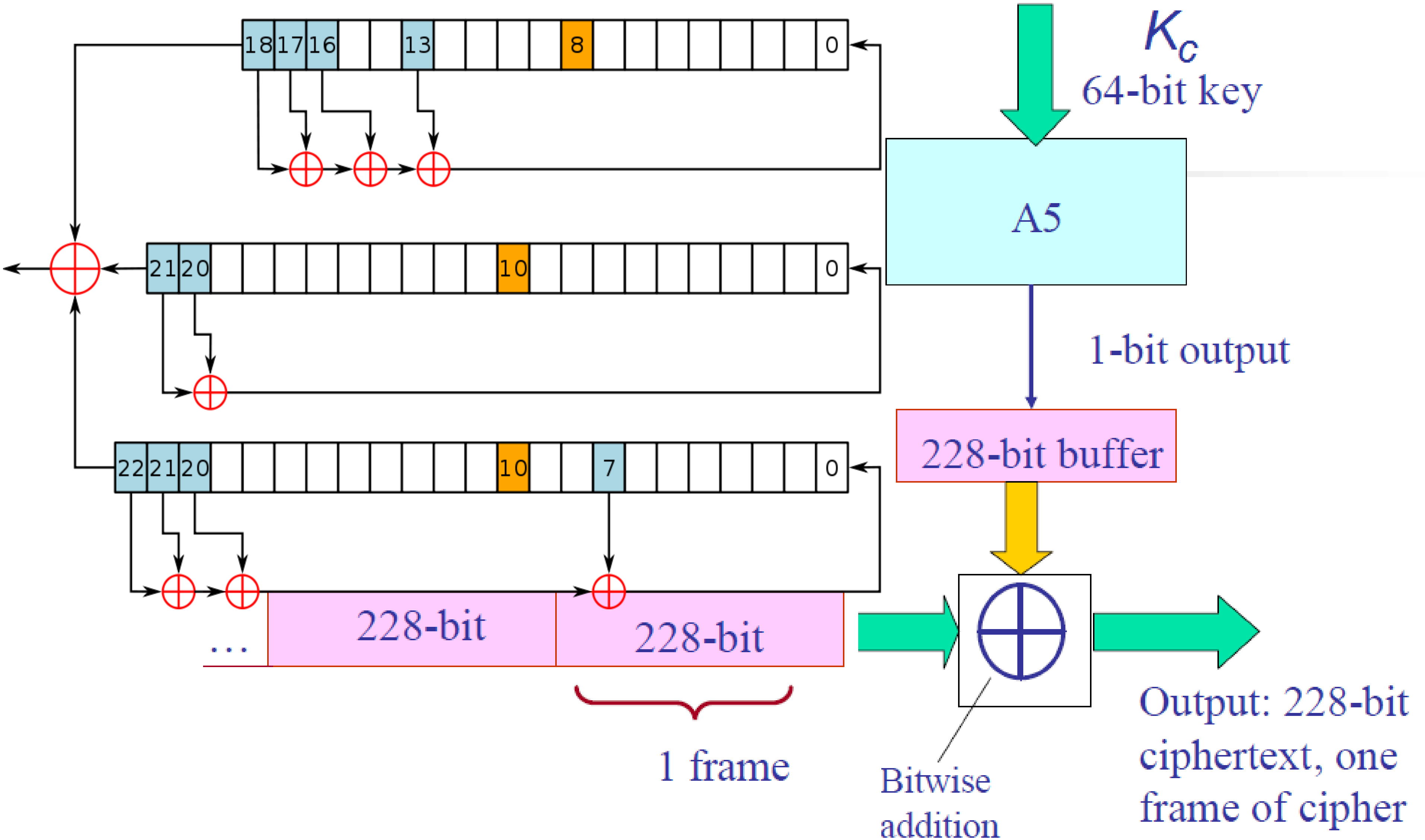
CS557: Cryptography

Stream Cipher-II

S. Tripathy
IIT Patna

LFSR-Based Ciphers (A5/1)



A5/1 Initialization

- Registers set to all 0's
- Incorporate the key and frame number:
  - For 64 cycles, the key is mixed in by XORing the i^{th} key bit with the least significant bit of each register
  - For 22 cycles, the 22 bit frame value is mixed in – same as with key value
  - Normal clocking used
- 100 cycles are run using the majority clocking, the output is discarded
- End result is the initial state

RC4 Basics

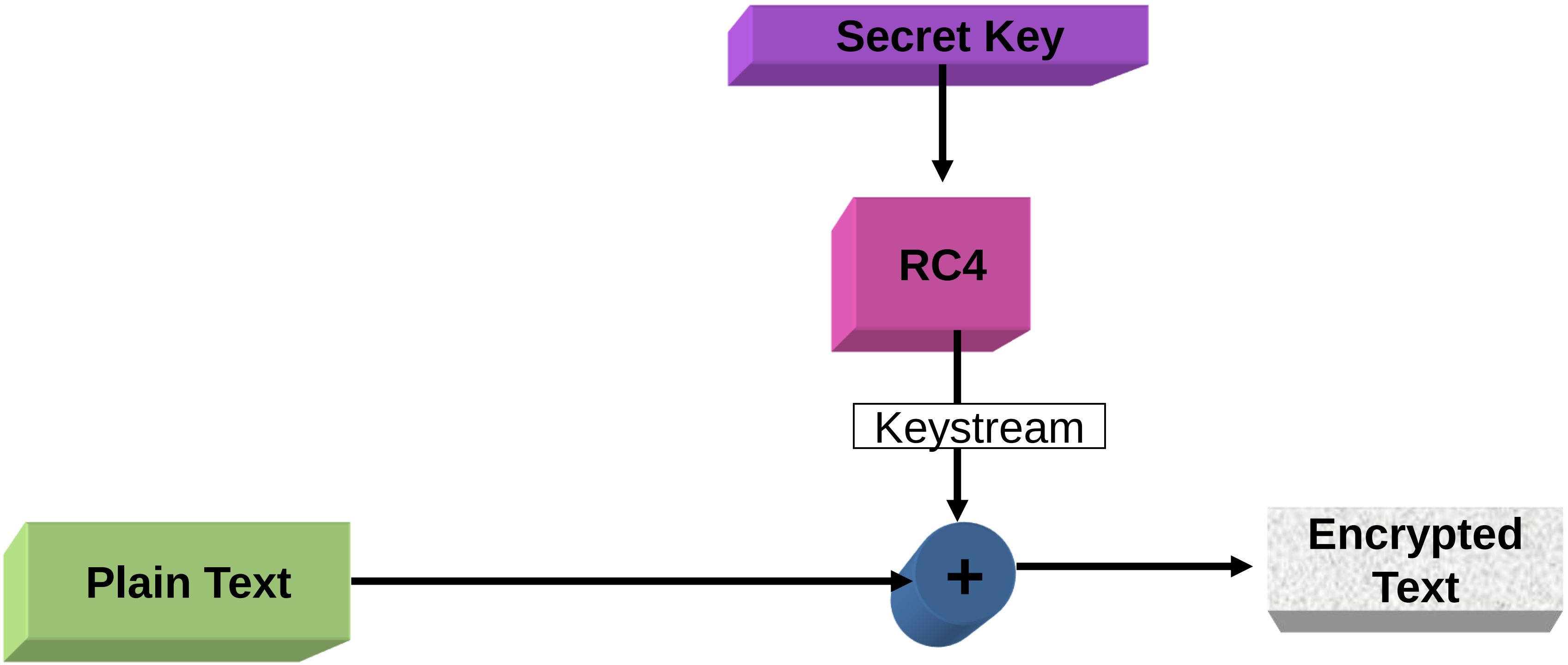
- A symmetric key encryption algorithm invented by Ron Rivest
  - A proprietary cipher owned by RSA, kept secret

- **Variable key size**, **byte-oriented** stream cipher
  - Normally uses 64 bit and 128 bit key sizes.
- Used in
  - SSL/TLS (Secure socket, transport layer security) between web browsers and servers,
  - IEEE 802.11 wirelss LAN std: WEP (Wired Equivalent Privacy), WPA (WiFi Protocol Access) protocol

Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
  - large linear complexity
  - correlation immunity
  - confusion
  - diffusion
  - use of highly non-linear Boolean functions

RC4 Block Diagram



Cryptographically very strong and easy to implement

RC4 …Inside

- Consists of 2 parts:
  - Key Scheduling Algorithm (KSA)
  - Pseudo-Random Generation Algorithm (PRGA)
- KSA
  - Generate State array
- PRGA on the KSA
  - Generate keystream
  - XOR keystream with the data to generated encrypted stream

The KSA

- Use the secret key to initialize and permutation of state vector S, done in two steps

1

```
for i = 0 to 255 do
   S[i] = i;
   T[i] = K[i mod(|K|)]);
```

[S], S is set equal to the values from 0 to 255
S[0]=0, S[1]=1,…, S[255]=255
[T], A temporary vector
[K], Array of bytes of secret key
|K| = Keylen, Length of (K)

2

```
j = 0;
for i = 0 to 255 do
   j = (j+S[i]+T[i])(mod 256)
swap (S[i], S[j])
```

- Use T to produce initial permutation of S
- The only operation on S is a swap;
  S still contains number from 0 to 255

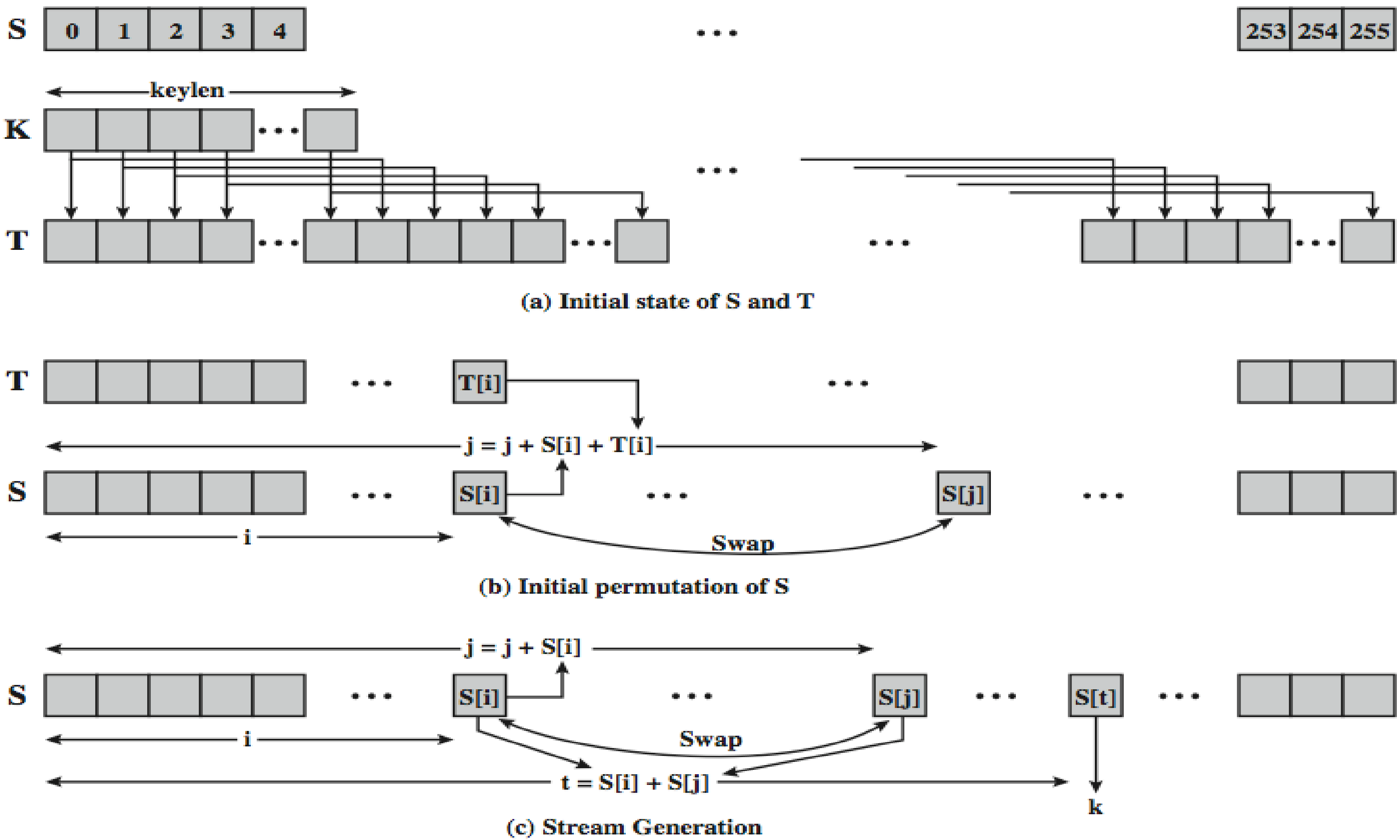
After KSA, the input key and the temporary vector T will be no longer used

The PRGA

- Generate key stream *k* , one by one
- XOR S[k] with next byte of message to encrypt/decrypt

```
i = j = 0;
While (more_byte_to_encrypt)
    i = (i + 1) (mod 256);
    j = (j + S[i]) (mod 256);
    swap(S[i], S[j]);
    k = (S[i] + S[j]) (mod 256);
    C_i = M_i XOR S[k];
```

RC4 Lookup Stage



Decryption using RC4

- Use the same secret key as during the encryption phase.
- Generate keystream by running the KSA and PRGA.
- XOR keystream with the encrypted text to generate the plain text.
- Logic is simple :

(A xor B) xor B = A

A = Plain Text or Data
B = KeyStream

Security of RC4

- Bit-flipping attack
  - A **bit-flipping attack** is an attack on a cryptographic cipher in which the attacker can change the ciphertext in such a way as to result in a predictable change of the plaintext, although the attacker is not able to learn the plaintext itself.
    - The attack is especially dangerous when the attacker knows the format of the message. In such a situation, the attacker can turn it into a similar message but one in which some important information is altered.
      - For example, a change in the destination address might alter the message route in a way that will force re-encryption with a weaker cipher, thus possibly making it easier for an attacker to decipher the message.
      - The attacker might be able to change a a note "I owe you $10.00" into one stating "I owe you $10000".
- In 1995, Andrew Roos experimentally observed that the first byte of the keystream is correlated to the first three bytes of the key.
  - the first few bytes of the permutation after the KSA are correlated to some linear combination of the key bytes

RC4 Security

- claimed secure against known attacks
  - have some analyses, none practical
- result is very non-linear
- since RC4 is a stream cipher, must *never reuse a key*
- have a concern with WEP, but due to key handling rather than RC4 itself

Performance comparison

Speed comparisons:
(from Crypto++ 5.1 benchmarks, on a 2.1 GHz P4):

Algorithm	Speed (MByte/s.)
DES	22
AES	62
RC5-32/12	79
RC4	111
SEAL	920
MD5	205

RC4 and WEP

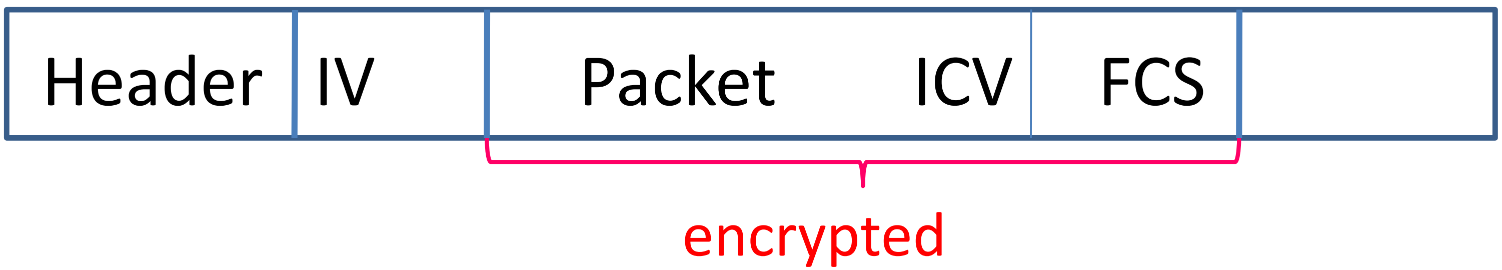
- WEP is a protocol using RC4 to encrypt packets for transmission over IEEE 802.11 wireless LAN.
  - WEP requires each packet to be encrypted with a separate RC4 key.
- The RC4 key for each packet is a concatenation of a 24-bit IV (initialization vector) and a 40 or 104-bit long-term key.

RC4 key:	IV (24)	Long-term key (40 or 104 bits)

The IV is only 24 bits, so eventually it will wrap around
1500-byte packets, 5Mbps, IV wraps in less than 12 hours

With random IVs, the birthday effect says we expect a repeat within 5000 packets (a few mins in the scenario above)

802.11 frames using WEP



- ICV: integrity check value (for data integrity)
- FCS: frame check sequence (for error detection)
- Both use CRC32

WEP Vulnerability

- WEP protocol has several flaws but not the RC4 itself
  - **Short IV length**
    - 24 bits IV not sufficient
  - **Clear text IV as part of the key**
    - 24 bits of every key in cleartext
    - Collect and analyze IVs to extract the WEP key
  - **Weak IVs**
    - Some generated IVs do not provide enough randomness
    - Can be used to extract the key

- Pseudo random number generator

Random Numbers in Cryptography

- The keystream in the one-time pad
- The secret key in the SKE encryption
- The prime numbers p, q in the RSA encryption
- The private key in DSA
- The initialization vectors (IVs) used in ciphers

Pseudo-random Number Generator

- Pseudo-random number generator:
  - A polynomial-time computable function f (x) that expands a short random string x into a long string f(x) that appears random
- Not truly random in that:
  - Deterministic algorithm
  - Dependent on initial values
- Objectives
  - Fast
  - Secure

Pseudo-random Number Generator

- Classical PRNGs
  - Linear Congruential Generator

- Cryptographically Secure PRNGs
  - RSA Generator
  - Blum-Micali Generator
  - Blum-Blum-Shub Generator

- Standardized PRNGs
  - ANSI X9.17 Generator
  - FIPS 186 Generator

Linear Congruential Generator -

- Algorithm :

  Based on the linear recurrence:

  $x_i = a\, x_{i-1} + b \mod m \qquad i \geq 1$

  Where

  x_0 is the seed or start value

  a is the multiplier

  b is the increment

  m is the modulus

  Output

  $(x_1, x_2, \ldots, x_k)$

  $y_i = x_i \mod 2$

  $Y = (y_1 y_2 \ldots y_k)$ ← pseudo-random sequence of K bits

Linear Congruential Generator - Example

- Let $x_n = 3\ x_{n-1} + 5 \bmod 31\ \ n \geq 1$, and $x_0 = 2$
  - 3 and 31 are relatively prime, one-to-one (affine cipher)
  - 31 is prime, order is 30
- Then we have the 30 residues in a cycle:
  - 11,
  - 7,
  - 26, 21, 6, 23, 12, 10, 4, 17, 25, 18, 28, 27, 24, 15, 19, 0, 5, 20, 3, 14, 16, 22, 9, 1, 8, 29, 30,2

- Pseudo-random sequences of 10 bits
  - when $x_0 = 2$
    1101010001
  - When $x_0 = 3$
  - …………………………………