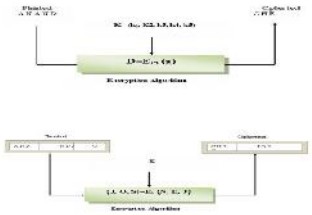
Cryptography

Stream cipher

Anand Ballabh Joshi
Department of Mathematics
University of Lucknow, Lucknow, India

Stream and block cipher



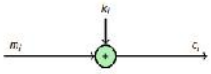
One Time Pad

The One Time Pad is the only theoretically secure cipher

- **Theorem:** The one time pad has perfect secrecy
- You must generate a truly random key sequence equally long as the message, and then find a secure channel for transportation of that key to the intended message recipient
- Your problem is now instead to transfer large quantities of secure key

One Time Pad

- Useful for sending secret data without preserved data integrity
- Uses a long random bitsequence as key
- Adds this to the plaintext to form the next cryptogram
- No error propagation



Stream Cipher

A more practical alternative is a stream cipher

- We generate a pseudorandom "key stream" from a seed, a "real key" much shorter than the full "key stream" added to the message
- We try to make the set of possible seeds, the real keys, so large that exhaustive search is impossible in practice
- We try to eliminate any shortcuts to finding this key from the "key stream"

Stream cipher

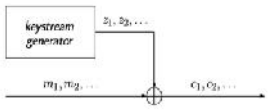
- Design goal is to efficiently produce random-looking sequences that are as "indistinguishable" as possible from truly random sequences.
- Recall the unbreakable Vernam cipher.
- For a synchronous stream cipher, a known-plaintext attack (or chosen-plaintext or chosen-ciphertext) is equivalent to having access to the keystream $z = z_1, z_2, \ldots, z_N$.
- *We assume that an output sequence z of length N from the keystream generator is known to Eve.*

Stream cipher

- Symmetric encryption algorithms are divided into two main categories, *block ciphers and stream ciphers*.
- Block ciphers tend to encrypt a block of characters of a plaintext message using a fixed encryption transformation
- A stream cipher encrypt individual characters of the plaintext using an encryption transformation that varies with time.

A stream cipher built around LFSRs and producing one bit output on each clock = *classic stream cipher design*.

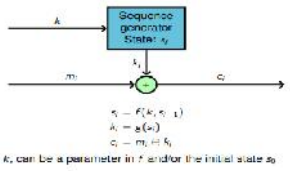
Stream cipher: key generator



- $z = z_1, z_2, \ldots$ *keystream*
- *key* K

Stream Cipher

Additive stream ciphers



The key, k, can be a parameter in f and/or the initial state s_0

Stream Cipher

Pseudorandom generator

- Sender and receiver must generate exactly the same key stream from the seed, so the generator is deterministic
- Real-world generators must be finite, having a fixed total number of states
- Sooner or later the generator is in a state it has been in before
- From then on, the internal states and the output repeats, exactly as the previous time the state was used
- Therefore, the output will be periodic, directly or after some initial output part

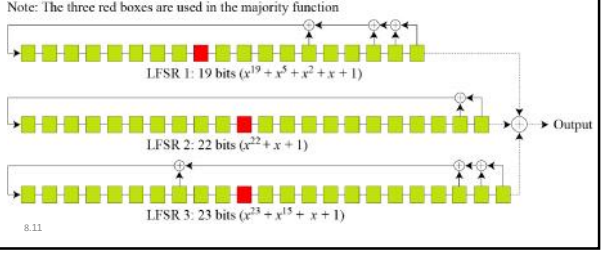
LFSR based stream cipher A5/1

Key Generator

A5/1 uses three LFSRs with 19, 22, and 23 bits.
Key size is 64.

Three LFSR's in A5/1

Note: The three red boxes are used in the majority function

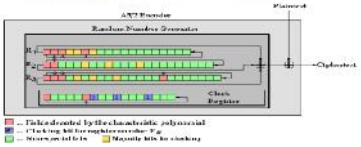


8.11

A5/2

A5/2 (Weaker GSM)

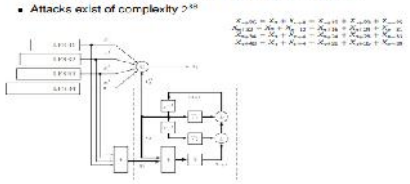
- GSM A5/2 uses four LFSRs, only clock output LFSRs where the majority orange bits agree with the clock LFSR
- Broken the same month 1999 it was published
- Prohibited from implementation in mobiles since 2007



E0(Bluetooth)

E0 (Bluetooth)

- Bluetooth E0 uses four LFSRs and two 2 bit internal states
- Attacks exist of complexity 2^{38}



RC4

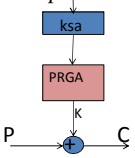
Developed by RSA Labs, RC4 is a symmetric, byte-oriented stream cipher with a variable length key size, in which a byte (8 bits) of a plaintext is exclusive-ored with a byte of key to produce a byte of a ciphertext.

KEY

RC4 HAS two main parts:

KSA (Key Scheduling Algorithm)

PRGA (Pseudo Random Generation Algorithm)



State

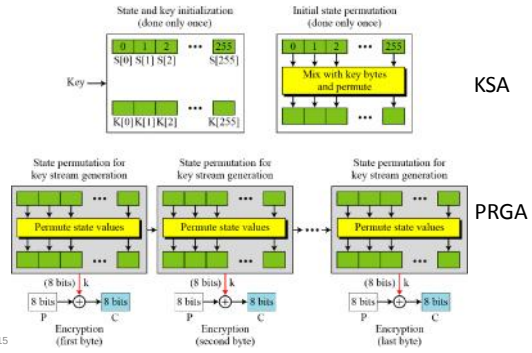
RC4 is based on the concept of a state.

S[0] S[1] S[2] ··· S[255]

8.14

Continued

Figure 8.10 *The idea of RC4 stream cipher*



8.15

RC4 Key Schedule KSA

Starts with an array S of numbers: 0..255

Use key to truly shuffle S

S forms internal state of the cipher

Given a key k of length L bytes

Scrambling Pseudocode :

for i = 0 to 255 do

S[i] = i

j = 0

for i = 0 to 255 do

j = (j + S[i] + k[i]) (mod 256)

swap (S[i], S[j])

8.16

RC4 PRGA and Encryption

Encryption involves XORing data bytes with output of the PRGA

The PRGA initializes i and j to 0 and then loops over 4 basic operations: increase j, increase j using s[i], swap and output s[i]+s[j]

PRGA Pseudocode is:

i = j = 0

for each message byte Mi

i = (i + 1) (mod 256)

j = (j + S[i]) (mod 256)

swap(S[i], S[j])

t = (S[i] + S[j]) (mod 256) ; Ki = S[t]

Encryption : Ci = Mi XOR S[t]

8.17