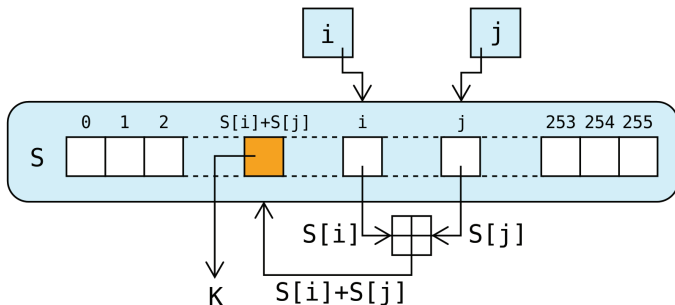


# Advanced Stream Ciphers

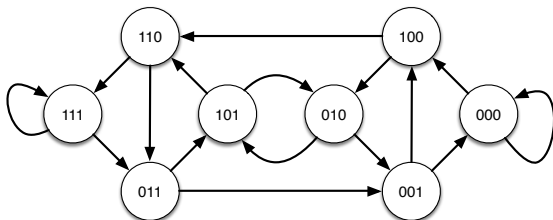


# Nonlinear Feedback Shift Registers

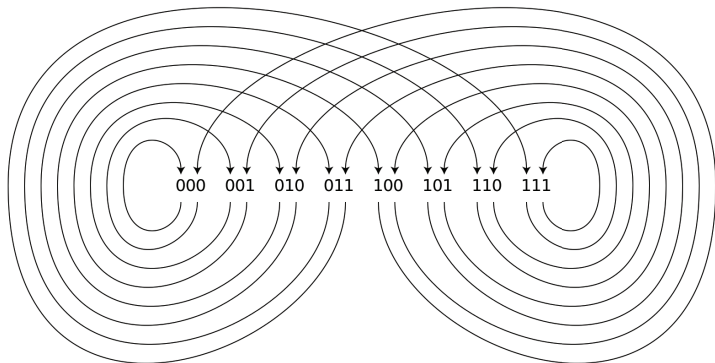
- LFSRs have very desirable statistical properties and excellent theory to build stream ciphers with large (maximal) periods
- However, due to linearity and the Berlekamp-Massey algorithm), they provide no security against known text attacks
- One way to overcome the cryptographic weakness is to use a nonlinear feedback function
- As in the case for LFSRs, the sequences of maximal period are of special interest; furthermore, the all-zero state is not excluded
- There are several mathematical tools for studying the properties of nonlinear shift register sequences

# de Bruijn Graphs

- A general shift register sequence can be represented using a graph of  $2^n$  vertices, labeled by the words of  $\{0, 1\}^n$
- The edges show possible transitions of a shift register, for example, we would have a transition from  $(s_3, s_2, s_2, s_1, s_0)$  to  $(0, s_3, s_2, s_2, s_1)$  and to  $(1, s_3, s_2, s_2, s_1)$
- These graphs are named de Bruijn graphs, and have applications in other fields as well, such as genome research



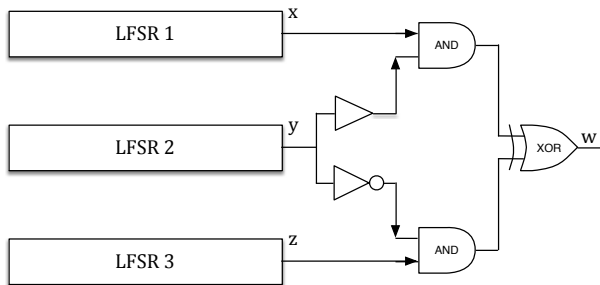
# A pretty de Bruijn Graph :)



# Nonlinear Combination of LFSRs

- While many mathematical properties of nonlinear shift register sequences can be studied using de Bruijn graphs (and associated de Bruijn sequences), there are no fast algorithms for generating such sequences
- An approach that makes use of the LFSRs (particularly, statistical properties, simple design, fast generation, maximality) but removing the cryptographic weakness is the *nonlinear combination* of LFSRs
- A very simple generator is proposed by Geffe: 3 LFSRs are used, and the output of one them decides which of the output of the other 2 LFSRs is to be used

# Geffe Generator



- If  $x, y, z$  are the output bits of the LFSRs at the  $i$ th step, the combined generator is

$$w = (x \wedge \bar{y}) \oplus (z \wedge y)$$

# Properties of Geffe Generator

- The LFSRs are not identical, each one having a length  $n_i$ , a particular connection polynomial  $c_i(x)$  for  $i = 1, 2, 3$
- Also, the lengths  $n_i$  are selected to be pairwise relative prime:  
 $\gcd(n_1, n_2) = \gcd(n_1, n_3) = \gcd(n_2, n_3) = 1$
- When the connection polynomials are primitive, each LFSR will be maximal, having the period of  $2^{n_i} - 1$  for  $i = 1, 2, 3$
- The Geffe generator has the period of

$$(2^{n_1} - 1)(2^{n_2} - 1)(2^{n_3} - 1)$$

- Unfortunately, this simple generator, while it has a long period, is still not secure due to correlation properties

# Properties of Geffe Generator

- The truth table of the Geffe generator output is given as

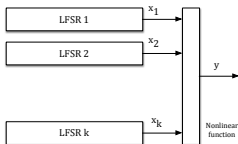
x	y	z	w
0	0	0	0 ←
0	0	1	1 ←
1	0	0	0 ←
1	0	1	1 ←
0	1	0	0 ←
0	1	1	0
1	1	0	1
1	1	1	1 ←

- The output the Geffe generator ( $w$ ) matches the output of the LFSR3 ( $z$ ) in 6 out of 8 times, correlating 75%
- If the known text attack gives us a set of  $w$  bits, we can apply the Berlekamp-Massey and exhaustive search to reconstruct LFSR3

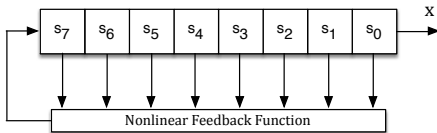


# General Nonlinear Generators

- The Geffe generator can be generalized to include other nonlinear functions that combine several LFSRs



- Or, more generally, we can use nonlinear feedback shift registers



- There are a myriad of other design options

# RC4

- RC4 is a software-suitable stream cipher, designed by Ron Rivest at RSA Labs (research arm of RSA Inc.)
- RC stands for Ron's Cipher, since he designed several ciphers for the company RSA Inc. (where I also worked between 1990-1995)
- RC4 became very important because of its use in TLS suites, providing confidentiality for e-commerce communication
- RC4 was designed for 8-bit processors, and requires a small state table, but it has very large period
- The key (seed) size is flexible, and can be any multiple of 8 bits
- The mathematics of RC4 is difficult, making it a hard cipher to cryptanalyze

# RC4 Key Table

- The RC4 Key Table is a linear array of  $k$  cells, where each cell is a byte

$T[0]$	$T[1]$	$T[2]$	$\dots$	$T[k-2]$	$T[k-1]$
--------	--------	--------	---------	----------	----------

- The number of bits in an RC4 key is a multiple of 8, and the value of  $k$  is between 1 and 256
- In other words, the minimum size key is 8 bits, while the maximum key size 2048 bits!
- During the times of export restrictions (until late 90s), the most common RC4 key was 48 bits ( $k = 6$ )
- Many US-based implementations used 128-bit ( $k = 16$ ) keys

# RC4 State Table

- The RC4 State Table is a linear array of 256 cells, where each cell is a byte

$S[0]$	$S[1]$	$S[2]$	$\dots$	$S[254]$	$S[255]$
--------	--------	--------	---------	----------	----------

- RC4 algorithm has 3 phases: Initialization, key scheduling, and pseudorandom byte generation
- First the State Table is initialized as

```
for i in range(256):  
    S[i] = i
```

- Thus, it becomes

0	1	2	$\dots$	254	255
---	---	---	---------	-----	-----

which is a permutation of 256 integer values from 0 to 255

# RC4 Key Scheduling

- The  $k$ -byte Key Table  $T$  is used to permute the elements of the State Table using the following algorithm

```
j = 0
for i in range(256):
    j = j + S[i] + T[i % k] % 256
    temp = S[i]
    S[i] = S[j]
    S[j] = temp
```

- As the index  $i$  progresses from 0 to 255, a new value of  $j$  is computed, and the State Table cells  $S[i]$  and  $S[j]$  are swapped
- At the end of the Key Scheduling process, the State Table is still a permutation of 256 integer values from 0 to 255

# RC4 Pseudorandom Byte Generation

- The Key Scheduling algorithm runs once, and the Key Table is not needed again
- The cipher starts producing the running key bytes, however, at each step, two cells in the the State Table is swapped, mixing up the State Table as it proceeds

```
i = 0
```

```
j = 0
```

```
while GenerateOutput == True
```

```
    i = i + 1 % 256
```

```
    j = j + S[i] % 256
```

```
    temp = S[i]
```

```
    S[i] = S[j]
```

```
    S[j] = temp
```

```
    R = S[ S[i] + S[j] % 256 ]
```

```
    print(R)
```

# Properties of RC4

- The State Table runs through all possible permutations of 256 values, and there are 256! distinct state tables
- This is indeed a huge number, which is about  $2^{1684}$
- However, note that the same key will always produce the same output sequence; RC4 does not have a nonce variable alongside with the key
- If a nonce is to be used, there must be a way to incorporate with the single long-term key, and thus, an incorrect use of the nonce and key may weaken the key scheduling algorithm
- A protocol involving RC4 that does not discard the beginning part of the output stream or that uses nonrandom (or related) keys will be vulnerable to attack, such as WEP

# Using Stream Cipher Modes of Block Ciphers

- An efficient way to generate a stream of deterministic random numbers is to use block ciphers, turning a block cipher box into a stream cipher
- There are 3 basic methods: OFB (output feedback), CFB (cipher feedback), and CTR (counter)
- In block cipher context, these methods are called “modes of operation”
- There are other modes of operation for block ciphers, each one of which is serving a different purpose



# Setup

- For all three modes, we assume the following:
- A block cipher encryption function is available, which produces an  $m$ -bit ciphertext  $C$  from an  $m$ -bit plaintext  $M$  using a  $n$ -bit key  $K$ :

$$C = E_k(M) \text{ such that } |C| = |M| = m \text{ and } |K| = n$$

- Also assume an initial value is available, which is generally called *initializing variable* and written as  $IV$

# The Output Feedback Mode

- The OFB produces a key stream  $r_i$  of  $s$  bits at each step, for  $s = 0, 1, 2, \dots$ , and computes the ciphertext  $c_i = r_i \oplus m_i$
- We have  $s \leq m$ , and generally  $s$  is a small number, such as 1, 2, or 8
- The algorithm performs for  $i = 0, 1, 2, \dots$ , starting with  $S_0 = IV$

$$T_i = E_K(S_i)$$

$$r_i = TR_s(T_i)$$

$$c_i = r_i \oplus m_i$$

$$S_{i+1} = r_i || RS_s(S_i)$$

- $TR_s(T_i)$  is the function that truncates the  $m$ -bit number  $T_i$  to  $s$  bits, either by taking the leftmost  $s$  bits
- Then,  $S_i$  is shifted  $s$  bits to right using  $RS_s(S_i)$  function, and the  $s$ -bit  $r_i$  is left-appended to get the new  $m$ -bit  $S_{i+1}$

# The Cipher Feedback Mode

- The CFB produces a key stream  $r_i$  of  $s$  bits at each step, for  $s = 0, 1, 2, \dots$ , and computes the ciphertext  $c_i = r_i \oplus m_i$
- We have  $s \leq m$ , and generally  $s$  is a small number, such as 1, 2, or 8
- The algorithm performs for  $i = 0, 1, 2, \dots$ , starting with  $S_0 = IV$

$$T_i = E_K(S_i)$$

$$r_i = TR_s(T_i)$$

$$c_i = r_i \oplus m_i$$

$$S_{i+1} = c_i || RS_s(S_i)$$

- $TR_s(T_i)$  is the function that truncates the  $m$ -bit number  $T_i$  to  $s$  bits, either by taking the leftmost  $s$  bits
- Then,  $S_i$  is shifted  $s$  bits to right using  $RS_s(S_i)$  function, and the  $s$ -bit  $c_i$  is left-appended to get the new  $m$ -bit  $S_{i+1}$

# The Counter Mode

- In the counter mode the  $m$ -bit initial state  $S_0$  consists of two parts: The  $u$ -bit count value  $l$  on the right and a  $(m - u)$ -bit nonce value  $N$  on the left:  $S_0 = N||l$
- The initial value of  $l = 1$ , and  $u$  is selected appropriately
- The algorithm performs for  $i = 0, 1, 2, \dots$ , starting with  $S_0 = N||1$

$$T_i = E_K(S_i)$$

$$r_i = TR_s(T_i)$$

$$c_i = r_i \oplus m_i$$

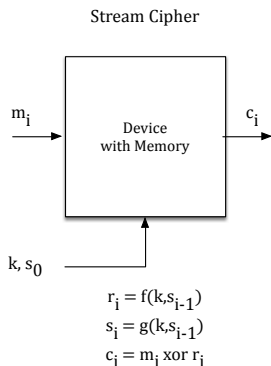
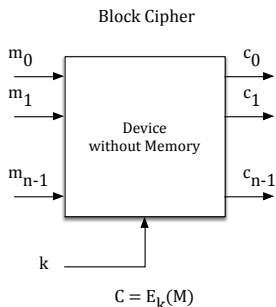
$$l = l + 1$$

$$S_{i+1} = N||l$$

- The new value  $S_{i+1}$  is obtained by incrementing the counter value  $l$  and keeping the nonce  $N$  as the same

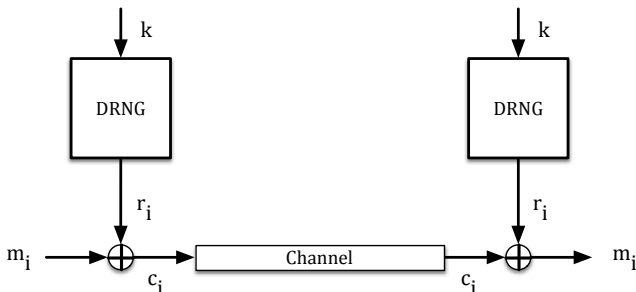
# Block Ciphers vs Stream Ciphers

- The fundamental difference between block ciphers and stream ciphers is the memory



# Synchronous Stream Ciphers

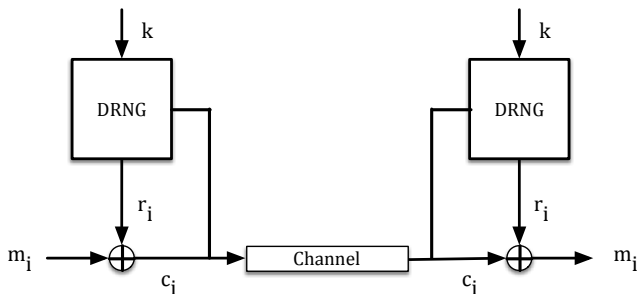
- The synchronous stream ciphers rely on the communication protocol to stay synchronous



- If the synchronization is lost, the cipher needs to be initialized, and to restart

# Self-synchronizing Stream Ciphers

- The self-synchronizing stream ciphers keep the synchrony



- Whenever a correct set of  $r$  consecutive ciphertext bits are transmitted, the cipher function will produce the same  $r_i$  value

$$r_i = f(k, c_j, c_{j+1}, \dots, c_{j+r-1})$$

# When to Use Stream Ciphers?

- Block ciphers are better understood than stream ciphers: If there are no special requirements, a block cipher used in one of the stream cipher modes (OFB or CFB) is a good choice
- There are some platforms where block cipher modes may not be suitable, for example, embedded devices where we try to save chip area (code space) and energy in embedded devices
- A shift-register based stream cipher needs fewer gates by several magnitudes than even a simple CPU, and therefore, much more suitable small, mobile, low-energy embedded systems



# When to Use Stream Ciphers?

- Also, stream ciphers can reach higher speeds than block ciphers; some stream ciphers can produce 64 or 128 bits per clock cycle which is significantly higher than any block cipher
- Currently hard disk space is growing faster than CPU speed; this might imply that in future we will have a greater need for high-speed ciphers, making stream ciphers good choices
- Furthermore, stream ciphers may also be a lot more useful for RFID devices, which is an important application
- If low energy consumption is important, stream ciphers will win out over block ciphers