Block Cipher AES, Stream Ciphers

Gerardo Pelosi

Department of Electronics, Information and Bioengineering (DEI)
Politecnico di Milano

gerardo.pelosi - at - polimi.it

Overview

Lesson contents

- Substitution-Permutation Networks
- AES
- Stream cipher design: LFSR and RC4

SPN

Description

- Round structure is explicitly split into three parts, acting on the whole state:
  - **Substitution**: a nonlinear function is applied to the state
    - As the function is expensive to compute, it is usually represented as a LUT, and applied via lookup and substitution, hence the name
  - **Permutation**: a permutation of the bits of the state
    - Instead of a bitwise permutation, it is common for SPNs to perform a **xor-linear** mixing of the bits (e.g., via a permutation matrix)
  - **Key mixing**: the key is added to the state via a XOR operation
    - In case the key is added via a nonlinear operation (f.i. addition mod 2^{32} –i.e. the common addition performed by a 32-bit ALU) the cipher is defined a *product cipher*
- Unlike Feistel-based ciphers, the encryption and decryption transformations are distinct

SPN

Main guidelines

- SPNs are built along Shannon's criteria of confusion and diffusion
- Confusion: obtained via the key mixing and substitution phase
  - The nonlinearity of the S-Boxes hinders statistical (i.e., frequency-based) attacks
- Diffusion: obtained through the permutation or linear diffusion phase
  - Spreads the effect of a small change in the ptx over the whole ctx
- Note: The diffusion speed of the whole cipher depends on the nature of the permutation

AES

AES - cipher structure

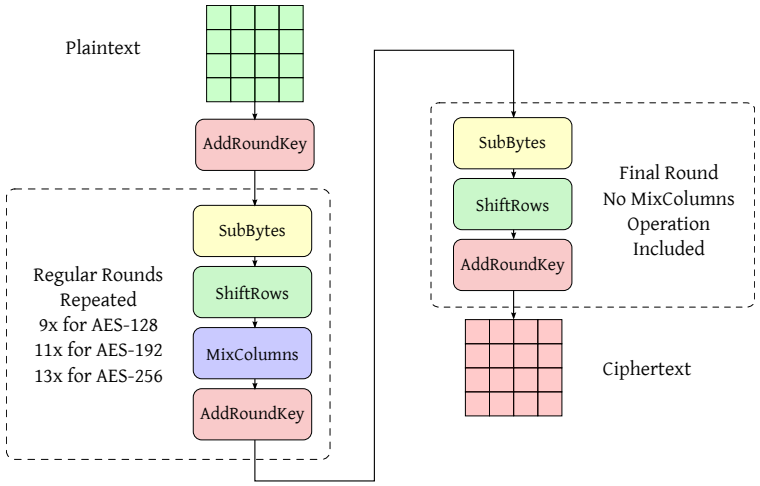
- 128-bit block size (thought as a 4×4 byte matrix), 3 key sizes (128- 192- and 256-bits)
- Extra **key addition** at the **beginning** of the cipher, **no diffusion** performed in the **last round**
- Simple and invertible key-Schedule:
  - it can be inverted employing an amount of key-material equal to the user-key
- Public, documented design criteria: winner of an international contest issued by US NIST in 2000
- Design criteria include:
  - a round structure simple to analyze
  - immune to linear and differential cryptanalyses (makes them more expensive than bruteforce)
  - Efficiently implementable in both HW and SW

SPN

AES - Round Structure

- Given a 4×4-byte matrix as input, the standard round contains:
  - A Substitution layer in the form of 16 S-boxes, 8-to-8 bit
  - A Permutation layer implemented via a bytewise rotation (ShiftRows) and a **xor-linear** operation among state bytes (MixColumn)
  - A key addition: bitwise ⊕, with 128 bits of expanded key material
- 10, 12 or 14 rounds are employed depending on the key length
- A key addition is performed as the initial operation of the cipher
- The last round does not have the Permutation layer (as it could easily be inverted)

AES - Structure

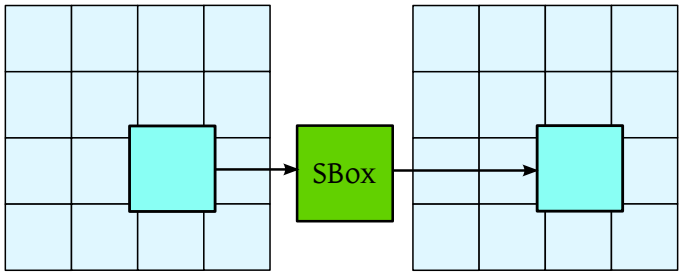


SPN

AES - Primitives - SubBytes

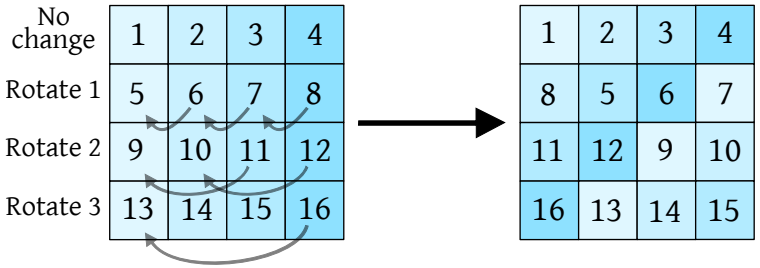
- Nonlinear function, 8-to-8 bit **bijective map**
- Structure completely described:
  1. Take input byte i, consider it as the coefficients of a polynomial over $(\mathbb{F}_{2^8}, \oplus, \odot)$: $i_7x^7 \oplus i_6x^6 \oplus \ldots \oplus i_0 \mod x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$
  2. Compute the inverse over the field (i.e. find i^{-1} such that $i \odot i^{-1} = 1$)
     - Difficult to approximate w/ linear relations: linear and differential biases are negligible
  3. Consider i^{-1} as a 8-bit vector (i.e., we are now working over vectors with values in $\mathbb{F}_2$), and compute $o = a \cdot i^{-1} \oplus b$, where a is a constant 8×8-bit matrix and b is a constant 8-bit vector
     - Difficult to solve algebraically (every output bit depends from **all** the bits of i^{-1})
  4. o is the output of the S-Box function

AES - Primitives - SubBytes



- Every byte is modified individually
- The byte substitutions can be **scheduled** in **any** order to the designer's preference

AES - Primitives - ShiftRows

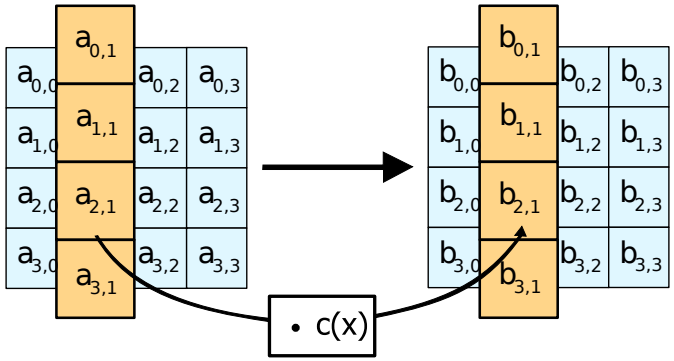


- First half of the permutation layer, provides inter-column diffusion
- Fast, easy to implement via bytewise rotate instructions
- Simple to implement in hardware, just correct cross-wiring

AES - Primitives - MixColumns

- Second half of the permutation layer, provides a **xor-linear intra-column diffusion**: **each column of the state is multiplied by a constant matrix** (with the rules of a proper algebraic structure)
- Every column $I = (i_0, i_1, i_2, i_3)$ is considered as coefficients of a polynomial over the ring $(\mathbb{F}_{2^8}[X], +, \cdot)$,
  i.e : $I(X) = i_0X^0 + i_1X^1 + i_2X^2 + i_3X^3 \mod (X^4 + 1)$
- Every coefficient i_i lies on $(\mathbb{F}_{2^8}, \oplus, \odot)$, and is expressed as a byte
- I is multiplied by a fixed pol.
  $C(X) = 0x02X^0 + 0x01X^1 + 0x01X^2 + 0x03X^3$
- The result $O(X) = I(X) \cdot C(X) \mod X^4 + 1$
  $O(X) = i_0 \odot 0x02 + (i_0 \odot 0x01 \oplus i_1 \odot 0x02)X + \ldots \mod X^4 + 1$
- As we want an invertible transformation, the $C(X)$ polynomial has an inverse $C(X) \cdot C^{-1}(X) \equiv_{X^4+1} 1$, which is
  $C^{-1}(X) = 0x14X^0 + 0x09X^1 + 0x13X^2 + 0x11X^3$

AES - Primitives - MixColumns



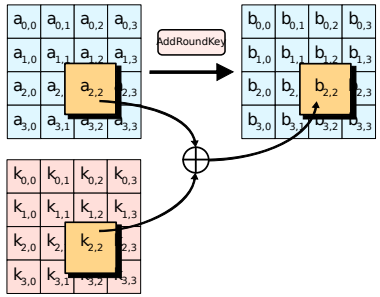
AES - Primitives - MixColumns

- It is possible to rewrite the operation in order to deal only with operations over $(\mathbb{F}_{2^8}, \oplus, \odot)$
- This transforms the operation in a vector-matrix multiplication with a constant matrix, over elements of $(\mathbb{F}_{2^8}, \oplus, \odot)$
- It is a **linear**, invertible transform, over $(\mathbb{F}_{2^8}, \oplus, \odot)$
- Acts on a column at once $I = (i_0, i_1, i_2, i_3)$ as

$$\begin{pmatrix} o_0 \\ o_1 \\ o_2 \\ o_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} i_0 \\ i_1 \\ i_2 \\ i_3 \end{pmatrix} \Rightarrow \begin{array}{l} o_0 = 2i_0 \oplus 3i_1 \oplus 1i_2 \oplus 1i_3 \\ o_1 = 1i_0 \oplus 2i_1 \oplus 3i_2 \oplus 1i_3 \\ o_2 = 1i_0 \oplus 1i_1 \oplus 2i_2 \oplus 3i_3 \\ o_3 = 3i_0 \oplus 1i_1 \oplus 1i_2 \oplus 2i_3 \end{array}$$

- Implementable uniquely with bitwise shifts and $\oplus$ operations

AES - Primitives - AddRoundKey



- The AddRoundKey is a simple bitwise $\oplus$ (i.e. an addition over $\mathbb{F}_2$) of the round key with the state
- The bitwise $\oplus$ is also an addition over $\mathbb{F}_{2^8} \Rightarrow$ MixColumns (MC) and AddRoundKey can be swapped provided $MC^{-1}(K)$ is used as the key

AES - T-Tables

- It is possible to fuse the three round primitives not involving the key (SubBytes, ShiftRows, MixColumns) into a single one
- The lookup table of the S-Box is combined with the effects of the MixColumns primitive to obtain four T-Tables

$$T_0[i] = \begin{bmatrix} S[i] \odot 02 \\ S[i] \\ S[i] \\ S[i] \odot 03 \end{bmatrix} \quad T_1[i] = \begin{bmatrix} S[i] \odot 03 \\ S[i] \odot 02 \\ S[i] \\ S[i] \end{bmatrix}$$

$$T_2[i] = \begin{bmatrix} S[i] \\ S[i] \odot 03 \\ S[i] \odot 02 \\ S[i] \end{bmatrix} \quad T_3[i] = \begin{bmatrix} S[i] \\ S[i] \\ S[i] \odot 03 \\ S[i] \odot 02 \end{bmatrix}$$

- The round is then computed column-wise (the input column is $I = [i_0, i_1, i_2, i_3]$ top to bottom in the state matrix) as

$$O = T_0[i_0] \oplus T_1[i_1] \oplus T_2[i_2] \oplus T_3[i_3] \oplus K$$

AES - Key Schedule

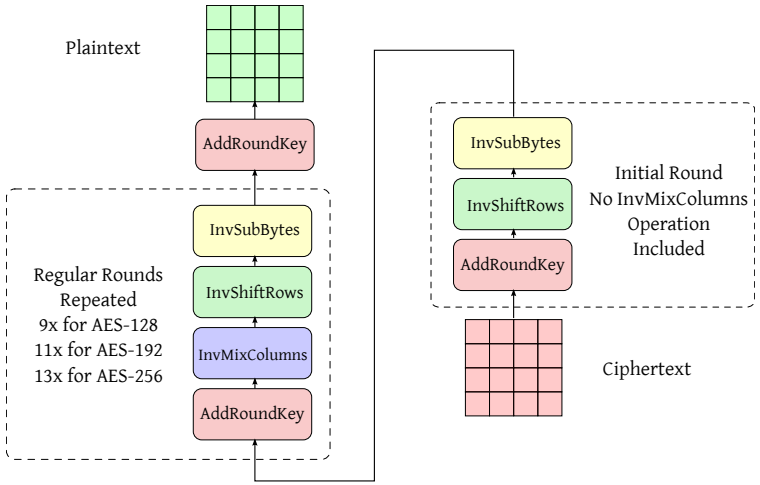
- The KEY-SCHEDULE expands the user-key into $\texttt{rounds}+1$ round keys i.e., expands $s = 4$ (AES-128), $s = 6$ (AES-192) or $s = 8$ (AES-256) 32-bit key words into $s \cdot (\texttt{rounds}+1)$ key words
- The first round key(s) are filled with the original user-key
- The rest of the key schedule KS is filled column-wise, following:

For $(i = s, \ldots, 4(r+1)-1)$
  If $(i \equiv 0 \mod s)$ $\{KS[i] = KS[i-s] \oplus S[KS[i-1] <<< 8]) \oplus RCON[i/s]\}$
  Else If $(s = 8 \wedge i \equiv 4 \mod s)$ $\{(KS[i] = KS[i-s] \oplus S[KS[i-1]])\}$
  Else $\{KS[i] = KS[i-s] \oplus KS[i-1]\}$

- $RCON$ is an array of round-dependent 32-bit constants
- **The key scheduling is invertible** (i.e., the user-key can be retrieved) provided at least s consecutive words of the key material are available

AES - Decryption

- The decryption process simply applies the inverse transformation of each cipher step in reverse order
- The structure of the decryption engine is thus different from the one employed for the encryption
- The InvSubBytes requires the inverse of the S-Box function
- The InvMixColumns employs the inverse matrix w.r.t. the Mixcolumns
- The InvShiftRows rotates the bytes to their right, instead of their left
- The key scheduling strategy is the same, but the round keys are employed in reverse order



AES - Properties

- A single bit flip in the input of the cipher is completely diffused over the state (i.e., all state bits are flipped with Pr$\approx\frac{1}{2}$ after 2 rounds)
- AES is completely immune to linear and differential cryptanalysis
- The best known cryptanalytic attacks are:
  - A known plaintext attack with complexities $\mathbf{2^{126.1}}$, $2^{189.7}$, $2^{254.4}$ for AES-128, AES-192, AES-256 respectively
  - A known plaintext, *related key* attack (i.e., correct ptx/ctx pairs encrypted with keys similar to the correct one needed) **breaks AES-192** and **AES-256** with 2^{176}, $2^{99.5}$ computations respectively (...they have more key bits than the number of cipher state bits)

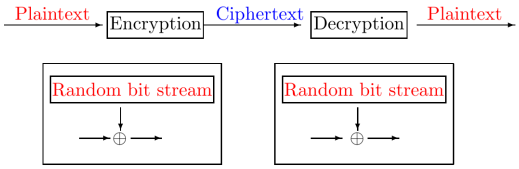
    AES-128 is immune to this attack (as the key is as large as the cipher state, thus there are no collisions of different key values onto the same intermediate state values)

Why a stream cipher?

- A stream cipher operates on individual ptx bits or digits
  - No padding needed
  - Useful for data streaming communications (e.g., voice, images, video)
- Throughput greater or equal to the fastest block ciphers
- Limited hardware/Software resources are required (f.i., in RFID)
  - can be fit into minimal HW resources ($\approx$500 gate equivalent)
  - limited memory requirements ($<$ 1 KiB)

Stream ciphers

The basic idea is to mimic the construction of the OTP cipher



The ptx messages are considered as a stream of digits $m_0, m_1, \ldots$, which is encrypted into a sequence of ctx digits $c_0, c_1, \ldots$ as follows:

- A sequence of **pseudo-random** digits called *running-key* or *keystream*, $k_0, k_1, \ldots$, is generated at both communication endpoints
- The i-th ctx digit c_i combines the i-th ptx digit with the i-th keystream one: $c_i = g(k_i, m_i)$. Usually: $c_i = k_i \oplus m_i$

Stream ciphers

In a **OTP cipher**, a **random key** must be employed for every ptx message and the key must have the **same length of the message**

- The **OTP keystream** must be **truly random** with no periodic repetitions whatsoever

In a **Stream Cipher**, we would like to

- use **short keys** $k \in \mathcal{K}$ to encrypt long messages
- use algorithmically generated **pseudo-random** values for the **keystream** instead of truly random ones
- be sure that the same **keystream** sequence is **repeated** only **after a very long sequence of messages** has been encrypted (... a practical value for infinity)

The keystream will be generated employing a number of memory elements $s_0, s_1, \ldots, s_{L-1}$ which initial state is determined by the cipher key $k \in \mathcal{K}$ (n.b. there will be at most 2^L different states)

Stream ciphers

Synchronous Stream Ciphers

The keystream is generated as a function of the **cipher key** and of the **memory elements**, independently of any previous ptx or ctx digit, i.e. $k_i = f(k, s_0, s_1, \ldots)$

- No error propagation. A bit error in the ctx affects one bit in the deciphered ptx, provided that synchronization is maintained
- Synchronization is crucial for a correct encryption and decryption
- An active adversary can use insertion or deletion or substitution (replay) of ctx digits to get predictable changes on the deciphered ptx

Stream ciphers

Asynchronous Stream Ciphers

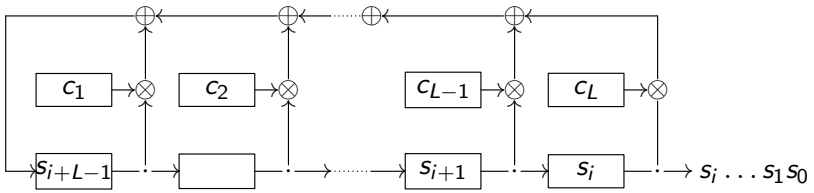
The keystream is generated as a function of the **cipher key** and a finite number of **previous ctxs**.
Given a key k and an initial state $S_0 = \langle s_{L-1}, \ldots, s_0 \rangle$ the keystream is composed as: $k_i = f(k, S_i, S_{i-1}, \ldots)$ with
$S_i = \langle c_{i+L-1}, c_{i+L-2}, \ldots, c_{i+1}, c_i \rangle$

- An erroneous ctx digit affects at most L digits of the deciphered ptx
- The decrypting endpoint synchronizes after receiving L ctx digits:
  - easier to recover if digits are dropped or added to the ctx stream (*self-synchronizing cipher*)
- An active adversary can use insertion or deletion or substitution (replay) of ctx digits to get predictable changes on the deciphered ptx

Linear Feedback Shift Registers

LFSR

A *linear feedback shift register* implements a keystream generator,



- A LFSR is a clocked circuit with some 1-bit memory cells, s_i, (the register)
- At each clock cycle, each bit value is moved to the adjacent memory cell. The output bit is the one shifted out of the register
- At each clock cycle, cells with a non-zero weight $c_i \in \{0,1\}$ are xor-ed together and the result is fed into the empty cell at the top of the register

LFSR

Properties

A LFSR has the following desirable properties:

- Easy to implement in hardware
- Produces a keystream with a *provably* (i.e., demonstrably) long period
- Produces a keystream with good statistical properties (pseudo-randomness)
- Can be readily analyzed using algebraic techniques

LFSR

Given the initial values of the register $\langle s_{L-1}, \ldots, s_1, s_0 \rangle$ and the configuration of the feedback network $\langle 1, c_1, c_2, \ldots, c_{L-1}, 1 \rangle$ (c_0=1, c_L=1, always) the **cipher key** is: $k = \{L, \langle s_{L-1}, \ldots, s_0 \rangle, \langle c_1, \ldots, c_{L-1} \rangle\}$
The algorithm for updating the contents of the leftmost memory cell (i.e., s_{L-1}) gives a recurrence relation

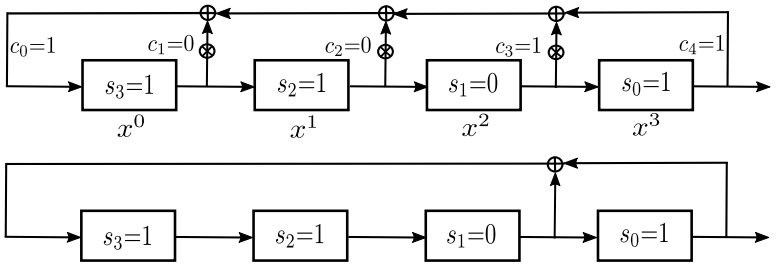
$$s_L = \sum_{i=1}^{L} c_i \cdot s_{L-i}$$

$$s_{t+L} = \sum_{i=1}^{L} c_i \cdot s_{t+L-i}, \ t \geq 0$$

- The *period* of the sequence is defined to be the smallest positive integer N such that $s_{t+N} = s_t, \ \forall\, t \geq 0$
- Note that, there are $2^L - 1$ possible non-zero states of the memory cell values $\langle s_{L-1}, \ldots, s_0 \rangle$ thus, N=$2^L - 1$ at most

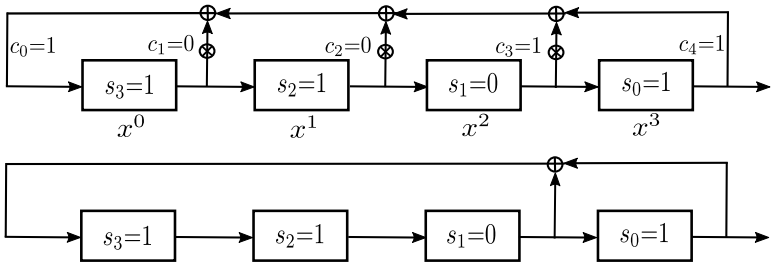
LFSR

Example. Consider a LFSR of length L=4 with feedback coefficients $\langle c_1, c_2, c_3, 1 \rangle$=$\langle 0, 0, 1, 1 \rangle$ and initial state $\langle s_3, s_2, s_1, s_0 \rangle$=$\langle 1, 1, 0, 1 \rangle$



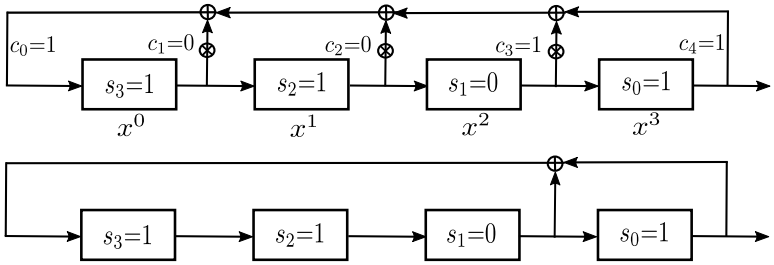
It corresponds to the equality: s_{t+4}=s_{t+1}+$s_t \mod 2, \ \forall\, t$≥0

LFSR



At each clock tick, every bit in the register is right-shifted, while a new value is computed for the leftmost bit by the feedback network.
In the example: $s_0 s_1 \ldots 1\,0\,1\,1\,1\,1\,0 \ldots$

LFSR



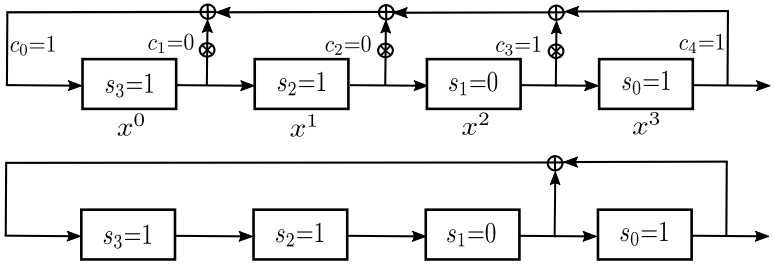
The LFSR function is described also by the values in the registers.

- A **status polynomial** with degree $L-1$ and coefficients in $\{0,1\}$ is associated with the LFSR: $s(x)=\sum_{i=0}^{L-1} s_{L-1-i} \cdot x^i$
- A **connection polynomial** with degree L and coefficients in $\{0,1\}$ is associated with the feedback network: $c(x)=\sum_{i=0}^{L} c_i \cdot x^i$

At each clock tick, the bits in the registers are equal to the coefficient of the status polynomial updated as:

$$s'(x) = s(x) \cdot x \bmod c(x)$$

LFSR



- **status polynomial**: $s(x)=1+x+x^3$
- **connection polynomial**: $c(x)=1+x^3+x^4$

At each clock tick, the bits in the registers are equal to the coefficient of the status polynomial updated as:
$s'(x) = s(x) \cdot x \bmod c(x)$
$s'(x) \equiv_{c(x)} x \cdot s(x) \equiv_{1+x^3+x^4} x + x^2 + x^4 \equiv_{1+x^3+x^4} 1 + x + x^2 + x^3$

LFSR

Feedback polynomial and characteristic polynomial

The output sequence of an LFSR is uniquely determined by its feedback coefficients and its initial state. The feedback coefficients are usually represented by the LFSR **feedback polynomial** (connection polynomial):

$$C(X) = 1 + \sum_{i=1}^{L} c_i X^i \in \mathbb{F}_2[X] \quad (\text{op.s mod } 2)$$

Alternatively, one can use the **characteristic polynomial**, which is:

$$G(X) = X^L C(X^{-1}) = X^L + \sum_{i=1}^{L} c_i X^{L-i} \in \mathbb{F}_2[X]$$

In the previous example, we had $L=4$, $(c_1, c_2, c_3, c_4)=(0,0,1,1)$, thus:

$$C(X) = 1 + X^3 + X^4, \; G(X) = X^4 + X + 1$$

LFSR

Quick note on Polynomials with coefficient in $\mathbb{F}_{2^L}$

A polynomial with degree L, $c(x) = 1 + x + \ldots, x^L \in \mathbb{F}_{2^L}$ is **irreducible**:

- if it cannot be written as the multiplication of polynomials with degree less than L

A polynomial with degree L, $c(x) = 1 + x + \ldots, x^L \in \mathbb{F}_{2^L}$ is **primitive**:

- If each of its roots (in the finite field $\mathbb{F}_{2^L}$), say r, is able to generate every element in $\mathbb{F}_{2^L} \setminus \{0\}$ as: r^n, $n = \{0, \ldots, 2^L - 1\}$

A **primitive** polynomial is also an **irreducible** polynomial

In a few weeks we will give definitions for irreducible and primitive polynomial, and methods to find them

LFSR

Theorem (Characterization of LFSR output sequences)

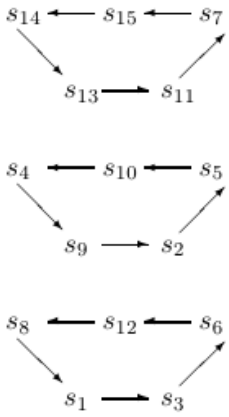
Given a LFSR with L-memory cells:

- *a non-zero initial state* $\langle s_{L-1}, \ldots, s_1, s_0 \rangle$
- *a feedback network* $\langle c_1, \ldots, c_{L-1}, c_L \rangle$, $c_L \neq 0$
- **If** $C(X)$ **is irreducible** *then the LFSR produces a keystream with period N, where N is a* **divisor of** *the maximum possible period* $2^L - 1$
  - *N is the smallest integer s.t.* $C(X)$ *is a factor of* $X^N + 1$
- **If** $C(X)$ **is primitive** *then the LFSR produces a keystream with period* $N = 2^L - 1$

LFSR

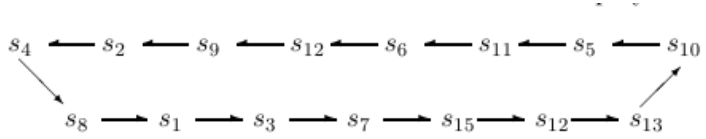
Consider a LFSR with $L = 4$, $C(X) = 1 + X + X^2 + X^3 + X^4$ which is **irreducible <u>but not</u> primitive**.
All the non-zero initial states, $S = \langle s_3, s_2, s_1, s_0 \rangle$, provide the following possible transitions with $N = 5$
(n.b., N=5 divides $2^L - 1 = 2^4 - 1 = 15$)

- the state register $S = \langle s_3, s_2, s_1, s_0 \rangle$ of the LFSR will cycle through only 5 possible values
- the cycled values are denoted on the right as: $S_{14} = S_{14_{\text{dec}}} = S_{1110_{\text{bin}}}$ $\Rightarrow \langle s_3, s_2, s_1, s_0 \rangle = \langle 1110 \rangle$



LFSR

Consider a LFSR with $L = 4, C(X) = 1 + X + X^4$ which is **irreducible <u>and</u> primitive** then the LFSR output values will have a period $N = 2^4 - 1$



LFSR

A primitive **LFSR** has **good statistical properties** and can be employed as **fair PRNG**. It is possible to prove that the following postulates for the binary keystream sequence s_t, with period N are satisfied

Golomb's postulates

A sequence of k consecutive 0's (or 1's) is called a *run* of length k

- In every N-bit sequence, the number of zeros is nearly equal to the number of ones (the disparity does not exceed 1!)
- In every N-bit sequence, $\frac{1}{2}$ the runs have length one, $\frac{1}{4}$ have length two, $\frac{1}{8}$ three etc..
- For each run length, there are equally many runs of '0s and of 1's
- Counting the number digits matching between a period of the sequence and an infinite sequence from the same LFSR yields 0 if the single period is aligned with a period of the infinite sequence, and a constant k otherwise

- Lossless data compression algorithms applied to a primitive LFSR yield negligible reduction of the bit-rate (... there is no statistical redundancy)

LFSR

Stream ciphers based on **LFSRs** are not usable on their own for cryptographic purposes, because they are **essentially linear** (as $c_i = m_i \oplus s_i$) and subject to *known plaintext attacks*

Known Plaintext Attack against a LFSR

Assume that the length L of an LFSR is known, and assume that we can obtain at least $2L$ **ptx-ctx digit pairs**: (m_0, c_0),...,(m_{2L-1}, c_{2L-1}).

The initial state of the State Register $\langle s_{L-1}, \ldots, s_1, s_0 \rangle$ is computed

- recovering the keystream values $s_{2L-1}, s_{2L-2}, \ldots, s_0$ as $s_i = m_i \oplus c_i$

The coefficients of the feedback network $\langle c_1, c_2, \ldots, c_L \rangle$ are computed through solving a set of L simultaneous linear equations

- $$s_{j+L} = \sum_{i=1}^{L} c_i \, s_{j+L-i} \mod 2 \quad j \geq 0$$

LFSR

KPA against a LFSR

$$\begin{array}{c|l} j = 0 & s_L = c_1 \, s_{L-1} + \ldots + c_L \, s_0 \mod 2 \\ j = 1 & s_{L+1} = c_1 \, s_L + \ldots + c_L \, s_1 \mod 2 \\ \ldots & \ldots \ \ldots \ \ldots \\ j = L-1 & s_{2L-1} = c_1 \, s_{2L-2} + \ldots + c_L \, s_{L-1} \mod 2 \end{array}$$

We now have L linear equations in L unknowns $c_1, \ldots, c_L$.
The c_i coefficients are fixed by the structure of the feedback network,
$\Rightarrow$ we solve for them with the previously collected $2L$ keystream values

$$\begin{pmatrix} s_{L-1} & \cdots & s_0 \\ s_L & & s_1 \\ \vdots & & \vdots \\ s_{2L-2} & \cdots & s_{L-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_L \end{pmatrix} = \begin{pmatrix} s_L \\ s_{L+1} \\ \vdots \\ s_{2L-1} \end{pmatrix}$$

LFSR

Desirable properties of keystream generators

- large period
- good randomness properties (Golomb's postulates)
- the output bits should be obtained in non-linear way

- **Combining multiple LFSR to compute a single keystream**. The combination function **increase the total period of the resulting keystream generator**
  - f.i., choosing to combine 3 LFSR outputs (having optimal lengths L_1, L_2, L_3) with $f(x_1, x_2, x_3) = x_1 \oplus x_2 x_3$ yields a LFSR with longer period... however if we observe the frequencies of the output sequence and the ones of each internal LFSR ... (from the truth table of f) we'll note that ... $\Pr(f{=}x_1){=}75\%$

LFSR

Geffe Generator

Three LFSRs:
LFSR2 (x_2) is used to choose between LFSR1 (x_1) and LFSR3 (x_3)

$$x = x_1x_2 \oplus x_3x_2 \oplus x_3$$

This means that

$$\begin{cases} x = x_1, & \text{if } x_2 = 1 \\ x = x_3, & \text{otherwise} \end{cases}$$

- The output bit x is not chosen uniformly from the two registers x_1, x_3
- A correlation attack based on the fact that $\Pr(x = x_1) = \Pr(x = x_3) = 0.75$ can be performed... (*refer to the book*)

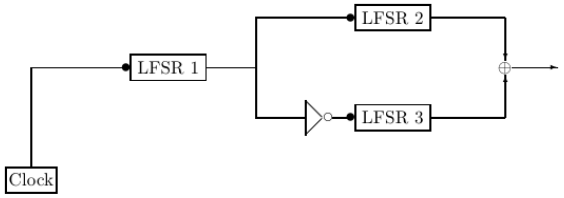
LFSR

Stop-and-Go Generators

To avoid Correlation problem:
one (or more) LFSR is used to clock the others

The *alternating* stop-and-go generator is defined with three LFSRs.
LFSR1 x_1 is used to trigger the clock signal between x_2 and x_3,

- the output x is obtained as $x_2 \oplus x_3$
- If $x_1 = 0$, LFSR2 is clocked; otherwise LFSR3 is clocked

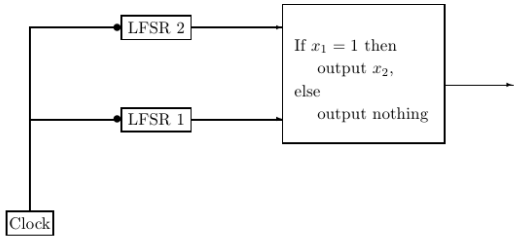


LFSR

The Shrinking Generator

There are two simultaneously clocked LFSRs (LFSR1 x_1, and LFSR2 x_2) with a single output x

- If x_1 is 1, then $x = x_2$
- Else, discard both x_1 and x_2 and forward the LFSRs



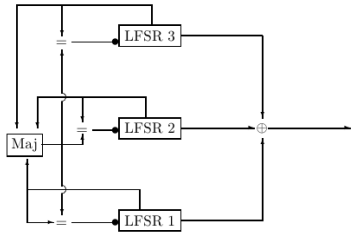
LFSR-based ciphers: A5

Introduced in 1987 to encrypt the on-air traffic of GSM mobile-phone networks

- A5/1 was designed secretly. The design was reverse engineered in 1999
- Three LFSRs:
  - $C_1(X) = x^{19} + x^5 + x^2 + x + 1$
  - $C_2(X) = x^{22} + x + 1$
  - $C_3(X) = x^{23} + x^{15} + x^2 + x + 1$

  with a 64-bit state in total, while the output is simply the xor of the registers

- Bit values in positions 10 (C_1), 11 (C_2), 12 (C_3) are evaluated with a majority arbiter
- At each clock step, LFSR$_i$ is clocked if the majority value is equal to the clocking bit c_i



LFSR-based ciphers: A5

- A5/1 is no longer a secure cipher (partial attacks were published)
- The variant A5/2 is also completely broken. [Barkan et al., 2003]
- The current standard A5/3, for UMTS/3G mobile networks, is a block cipher
- A5/3 is a Feistel based cipher with a 128-bit key and 64-bit block size (highly efficient to implement in hardware)
- Some cryptanalytic results are known on A5/3, but none of these lead to practical attacks in the real world scenario

Software-Oriented Stream Ciphers

Alternatives:

- Block ciphers (... or hash functions) in CFB, OFB, CTR modes
- Stream ciphers designed for software: RC4

Ron's Cipher 4 (RC4)

- Invented by Ron Rivest in 1984
- It a byte-oriented stream cipher: any digit of ptx, ctx or keystream is a byte
- It is very fast and popular in SW implementations
- It was recommended in SSL/TLS protocols for communication between web browsers and servers and both in the WEP (Wireless Equivalent Privacy) protocol and in the Wi-Fi Protected Access (WPA) protocol (a.k.a. TKIP - Temporal Key Integrity Protocol), part of IEEE 802.11 standards for wireless LANs.
  Currently, it should not be used anymore!!!

Ron's Cipher 4 (RC4)

Cipher key

- `keylength`: from 5 bytes (40-bit) up to 256 bytes (2k-bit)

Cipher state

- A permutation of the integers $\{0, 1, \ldots, 255\}$, stored in an array S with 256 bytes
- Two 8-bit index-pointers (denoted i and j)

Key-scheduling algorithm (KSA)

- The permutation array S is initialized with a variable length key k, typically stored in an array with size between 5 and 256 bytes ($k[0], k[1], \ldots$)
  for $i = 0$ **to** 255 **do** $S[i] \leftarrow i$ **endfor**
  $j \leftarrow 0$
  for $i = 0$ **to** 255 **do**
  $j \leftarrow (j + S[i] + k[i \bmod \texttt{keylength}]) \bmod 256$
  `swap values of` $S[i]$ `and` $S[j]$
  endfor

Ron's Cipher 4 (RC4)

Encryption/Decryption Algorithms

- Each byte of the ptx is xor-ed with a byte of the keystream
- The bytes of the keystream are generated using the following pseudo-random generation algorithm (PRNG)

$i \leftarrow j \leftarrow 0$
while `GeneratingOutput` **do**
  $i \leftarrow (i + 1) \bmod 256$ //each byte in S is used once after 256 iterations
  $j \leftarrow (j + S[i]) \bmod 256$ // the output depends non-linearly from S
  `swap values of` $S[i]$ `and` $S[j]$ // S is modified at each iteration
  $k \leftarrow S[(S[i] + S[j]) \bmod 256]$ // The output is hard to link with the state S
  `output` k
endwhile

Replacements for RC4, WEP/WPA

March 2013, RC4 is broken (1)

20th International Workshop on Fast Software Encryption 2013
Full Plaintext Recovery Attack on Broadcast RC4
by Takanori Isobe et al.

- Recover Any byte of the same ptx from ONLY ctxs encrypted by different keys
- Based on strong biases set of the first 257 bytes
  - Given 2^{32} ciphertexts with different keys, any byte of first 257 bytes of the plaintext are recovered with probability of more than 0.5

March 2013, RC4 is broken (2)

D. J. Bernstein in http://cr.yp.to/talks/2013.03.12/slides.pdf
computed all possible biases (given one byte of the ctx, how probable it is for a state byte to have a certain value)

Replacements for WEP/WPA

WPA2

Recommended as replacement in IEEE 802.11i std. and subsequent versions

- encryption: AES-CTR mode
- integrity: the last block of an AES-CBC encryption is used as message authentication code