**Linear Feedback Shift Register (LFSR)**

An LFSR is a sequential digital circuit that:

* Consists of n flip-flops (or registers) arranged in a shift register
* Uses a linear function (XOR) of selected bits — called taps — to compute the feedback
* Shifts its contents to the right or left every clock cycle
* Inserts the feedback bit into the first flip-flop4

Applications of LFSRs

|  |  |
| --- | --- |
| Field | Usage |
| Cryptography | Stream ciphers (e.g., A5/1 in GSM) |
| Communications | PN sequence generators, spread spectrum |
| Error detection | CRC (Cyclic Redundancy Check) |
| Testing | Built-in Self-Test (BIST), PRBS generators |
| Digital Logic | Pseudo-random pattern generation |

|  |  |  |  |
| --- | --- | --- | --- |
| Shift Direction | Tap Index Formula | Feedback Inserted At | Bit flow |
| Left Shift | *n - exponent - 1* | *state [0] (LSB)* | *LSB ← MSB* |
| Right Shift | *Just use state[exponent]* | *state[n-1] (MSB)* | *MSB ← LSB* |

**Example 1: x⁴ + x + 1**

**Step 1: Polynomial**

P(x) = x⁴ + x¹ + 1

**Step 2: LFSR Size**

* Highest degree: 4 → needs **4 flip-flops / bits**

So:

state = [s0, s1, s2, s3] # s0 = MSB

**Step 3: Tap Exponents (Ignore x⁴)**

* Tap terms: x¹ and x⁰
* Tap indices:
  + x¹: 4 - 1 - 1 = 2
  + x⁰: 4 - 0 - 1 = 3

Taps: state [2], state [3]

**Step 4: Feedback Equation**

feedback = state [2] ^ state [3] ...for left shift  
feedback =state [1] ^ state [0[ ...for right shift

**Example 2: x⁵ + x³ + 1**

**Step 1: Polynomial**

P(x) = x⁵ + x³ + 1

**Step 2: LFSR Size**

* Degree 5 → needs **5 flip-flops**

state = [s0, s1, s2, s3, s4]

**Step 3: Tap Exponents**

* Ignore x⁵ (defines size)
* Tap terms: x³ and x⁰

Tap indices:

* x³: 5 - 3 - 1 = 1
* x⁰: 5 - 0 - 1 = 4

Taps: state [1] and state [4]

**Step 4: Feedback Equation**

feedback = state [1] ^ state [4]

**Example 3:** x⁵ + x⁴ + x² + x + 1

**Step 1: Polynomial**

P(x) = x⁵ + x⁴ + x² + x + 1

This has **5 terms**:

* Highest degree: **x⁵** → LFSR needs **5 bits**
* Tap terms: x⁴, x², x¹, and x⁰

We will **ignore x⁵** when computing feedback (it defines the register size).

**Step 2: Setup LFSR Register**

state = [s0, s1, s2, s3, s4] # 5-bit register

↑ ↑ ↑ ↑

x⁵ x⁴ x³ x² x¹

index: 0 1 2 3 4

Bit index formula:

index = n - exponent - 1

So, for:

* x⁴ → 5 - 4 - 1 = **0**
* x² → 5 - 2 - 1 = **2**
* x¹ → 5 - 1 - 1 = **3**
* x⁰ → 5 - 0 - 1 = **4**

Tap indices: **0, 2, 3, 4**

**Step 3: Feedback Equation**

feedback = state [0] ^ state [2] ^ state [3] ^ state [4]

You then insert this feedback at the **left (MSB)** and shift the others to the right.

**LFSR Working (Each Cycle)**

We maintain a **5-bit register**:

state = [s0, s1, s2, s3, s4]

At each clock cycle:

1. **Compute feedback:**

feedback = s0 ^ s2 ^ s3 ^ s4

1. **Shift right**:
   * All bits move one position to the right
   * s4 (LSB) is dropped
   * feedback is inserted into s0 (MSB)
2. **New state:**

state = [feedback, s0, s1, s2, s3]

**Initial state:** state = [1, 0, 0, 0, 0]

**Cycle 1:**

* **feedback = 1 ^ 0 ^ 0 ^ 0 = 1**
* **new state = [1, 1, 0, 0, 0]**

**Cycle 2:**

* **feedback = 1 ^ 0 ^ 0 ^ 0 = 1**
* **new state = [1, 1, 1, 0, 0]**

**Cycle 3:**

* **feedback = 1 ^ 1 ^ 0 ^ 0 = 0**
* **new state = [0, 1, 1, 1, 0]**

**Cycle 4:**

* **feedback = 0 ^ 1 ^ 1 ^ 0 = 0**
* **new state = [0, 0, 1, 1, 1]**

**Cycle 5:**

* **feedback = 0 ^ 1 ^ 1 ^ 1 = 1**
* **new state = [1, 0, 0, 1, 1]**

This continues up to 2⁵ - 1 = 31 states (if using a primitive polynomial)

This kind of LFSR is used in:

* Pseudo-random number generation (PRNG)
* Scramblers (e.g., in communication systems)
* Error detection (CRC)
* Stream ciphers

**The initial state must not be all zeros.**

In an LFSR, if you ever reach the state [0, 0, 0, ..., 0], the feedback will always remain 0 (because XOR of all 0s = 0), and the LFSR will stay stuck in that state forever.

This is called the "zero lock-up" condition.

can pick any non-zero-bit pattern as your initial state.

**Primitive polynomial**

A primitive polynomial is a special kind of polynomial used in LFSRs and finite field arithmetic — one that guarantees maximum-length cycles and good randomness properties.

**Why It's Vulnerable**

* LFSRs are **linear** and **fully deterministic**
* If attacker knows:
  + the **polynomial**
  + the **initial seed** (or even a few bits of the stream)
* They can regenerate the entire keystream
* Then decrypt all future (and even past) ciphertexts

This is why **a bare LFSR is not secure** for real-world crypto.

**How Real Ciphers Avoid This**

Real-world stream ciphers (like A5/1, RC4, Salsa20):

* Use **keyed initialization** (secret key + IV)
* Use **nonlinear functions** (multiple LFSRs, S-boxes, etc.)
* Ensure **keystream is unpredictable** even if some output bits are known

**1. Keyed Initialization (Secret Key + IV)**

**Secret Key**

* A long, secret value (e.g., 128 bits)
* Known **only to sender and receiver**
* Used to initialize internal state

**IV (Initialization Vector)**

* A **non-secret random number** (e.g., packet ID, nonce)
* Ensures different output each time **even if same key is reused**
* Prevents replay attacks

**Together:**

* Key + IV are mixed to generate **initial internal state**
* Even if attacker sees output, they can’t work backwards to find key

**2. Nonlinear Functions**

LFSRs are **linear**: output is XOR of past bits → very predictable.

Real ciphers add **nonlinearity** to make analysis hard.

**Techniques:**

|  |  |  |
| --- | --- | --- |
| **Method** | **Used In** | **Description** |
| **Multiple LFSRs** | A5/1 | Several LFSRs run in parallel; combined nonlinearly |
| **S-boxes** | Salsa20, AES | Tables that map inputs to unpredictable outputs |
| **Rotations, additions** | Salsa20 | Mix bits using rotation + XOR + addition |
| **Key scheduling** | RC4 | Uses a permutation of 256 bytes to build state |

**3. Keystream Must Be Unpredictable**

Even if the attacker sees part of the output:

* They should **not** be able to guess future or past bits
* They should **not** be able to deduce the key
* They should **not** be able to clone the generator

This is what makes a stream cipher **cryptographically secure**.

**Real Cipher Examples**

**🔸 A5/1 (used in GSM phones):**

* 3 LFSRs of different lengths: 19, 22, and 23 bits
* Clocked irregularly (based on majority rule)
* Combined in a nonlinear way
* Uses **secret key (64 bits)** to initialize

**🔸 RC4:**

* A 256-byte internal state array
* Uses **key scheduling** to shuffle it
* Generates keystream byte-by-byte via permutations
* Known to have weaknesses now (not recommended for new systems)

**🔸 Salsa20 / ChaCha20:**

* Uses 256-bit key + 64-bit nonce (IV)
* Based on **add-rotate-XOR (ARX)** functions (nonlinear)
* Generates output in 64-byte blocks
* Very fast, widely used (e.g., TLS, VPNs)