Chapter 3

Design Criteria for Stream Ciphers

3.1 Stream Cipher Building Blocks

The choice of different building blocks and their proper usage is crucial in the design of stream ciphers. A weak selection and a combination of these functions may lead to a design which can then be susceptible to some attacks. The following sections elaborate different building blocks commonly used for building stream ciphers.

3.1.1 Boolean Functions

Boolean functions are very common in stream ciphers. They play a central role in the security of stream ciphers and block ciphers. A brief introduction to Boolean functions, their different forms of representation and few important criteria for evaluating the cryptographic complexity of Boolean functions are given in chapter 2.

3.1.2 Substitution Boxes

An substitution box (S-box) is a basic component of symmetric key algorithms which performs substitution. Substitutions are also known as vectorial Boolean functions. In block ciphers, they are typically used to obscure the relationship between the key and the ciphertext. An $n \times m$ S-box is a
function, \( S: F_2^n \rightarrow F_2^m \) which maps an \( n \)-bit input string \( x \) to an \( m \)-bit output string \( y \) where \( y = S(x), \ x = (x_1, x_2, ..., x_n) \) and \( y = (y_1, y_2, ..., y_m) \) [4]. An \( n \times m \) S-box is equivalent to \( m \) Boolean functions say \( S_i(x) \), known as the component functions. These functions give rise to the explicit equations \( S_i(x) = y_i \). In the case of \( m = 1 \), the S-box reduces to a single Boolean function. An \( n \times m \) S-box can be implemented as a lookup table with \( 2^n \) words of \( m \) bits each. Many block ciphers are designed based on the Shannon's idea of the sequential application of confusion and diffusion. Typically, confusion is provided by some form of substitution boxes and there were a lot of research done for designing techniques to construct cryptographically secure S-boxes. Design techniques for good S-boxes are somewhat sparse in the open literature. Some of the techniques for the construction of S-boxes are pseudorandom generation, finite field inversion, power mappings and heuristic techniques. Among these techniques the finite field inversion in the construction of an S-box provides good cryptographic properties.

Substitution Permutation Networks (SPNs) are a class of secret key ciphers having substitution boxes as critical components. An SPN consists of rounds of substitutions followed by bit permutations, and these two stages convert plaintexts into ciphertexts. To have a secure SPN, S-boxes should satisfy the properties such as avalanche, strict avalanche, differential uniformity, balancedness, completeness, high nonlinearity, algebraic immunity, robustness, correlation immunity and bit independence.

**Avalanche Criterion**

Feistel et al. defined a property of S-boxes known as avalanche criterion [42]. The avalanche criterion is an important cryptographic property of block ciphers which defines that a small number of bit differences in the input leads to an “avalanche” of changes, that is, it results in a large number of output bit differences. More formally, a function \( S: \{0, 1\}^n \rightarrow \{0, 1\}^m \) satisfies the avalanche criterion if whenever one input bit is changed, on the average, half of the output bits change. Formulating this, an \( n \times m \) S-box say \( S: F_2^n \rightarrow F_2^m \) exhibits the avalanche effect if and only if

\[
\sum_{x \in F_2^n} w_H(S(x) \oplus S(x \oplus e_i)) = m \cdot 2^{n-1},
\]
for each $i$ ($1 \leq i \leq n$) (‘$w_H$’ represents the Hamming weight), and $e_i = (0, 0, \ldots, 0, 1, 0, \ldots, 0)$ is the 1 bit error vector of $n$ bit length with 1 occupying the $i$-th bit position.

This means that whenever a single input bit is complemented then on an average one half of the output bits changes (irrespective of the bit positions).

**Strict Avalanche Criterion (SAC)**

Functions which satisfy SAC have important cryptographic applications. SAC was introduced by Webster and Tavares [4] in connection with the study of the design of S-boxes. A Boolean function in $n$ variables is said to satisfy SAC, if complementing any one of the $n$ input bits results in complementing each of the output bit with probability exactly one half. Forre [28] and Bart Preneel [27] extended this concept by defining the higher order SAC. The security of many symmetric ciphers depends on the cryptographic properties of the S-boxes. Although many design methods of S-boxes with desirable properties have been studied, there are a few constructive methods for S-boxes satisfying the SAC. PC and SAC are two important properties of an S-box to resist differential cryptanalysis [43] and are useful in designing cryptographic functions.

**Differential Uniformity**

Functions with a low differential uniformity are interesting from the point of view of cryptography and can be used as S-boxes in symmetric ciphers as they provide good resistance to differential attacks [44]. The lowest possible differential uniformity is 2 and functions with this property are called APN (Almost Perfect Nonlinear). The AES (Advanced Encryption Standard) uses a differentially 4 uniform function based on the inverse function $S(x) = x^{-1}, x \in \mathbb{F}_2^n, S(0) = 0$, which was chosen due to Nybergs suggestion [44]. To avoid the interpolation attack [45], the AES S-box is in general designed by adding a bitwise affine transformation after the inversion mapping. The different cases for deriving S-boxes with help of affine transformations are given as follows:

Let $A, B$ be any invertible matrices of order $n \times n$ with entries from $\mathbb{F}_2^n$ and
let $b \in \mathbb{F}_2^n$ and $d$ be an integer taken modulo $2^n - 1$ then

**Case 1:** $Ax^d \oplus b$

**Case 2:** $(Ax)^d \oplus b$ or $(Ax \oplus b)^d$

**Case 3:** $B(Ax^d \oplus b)$

AES comes under case 1 with $d = -1 \equiv 254 \pmod{2^8 - 1}$ and the hexadecimal number 0x63, that is the binary bit string 01100011 represents $b$.

A function $S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ is said to be differentially uniform $\nabla_S$ if for any $a \neq 0 \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m$ we have

$$|\{x \in \mathbb{F}_2^n : S(x \oplus a) \oplus S(x) = b\}| \leq \nabla_S$$

The value of $\nabla_S$ is obtained with help of the Differential Distribution Table known in short as DDT table (or XOR table). The lower the value of $\nabla_S$, the more resistant the function to differential cryptanalysis.

**Definition 3.1.1 (Difference Distribution Table).** Let $S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ be an S-box. For every $a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m$, define an entry of the difference distribution table as

$$\text{DDT}(a, b) = |\{x \in \mathbb{F}_2^n : S(x) \oplus S(x \oplus a) = b\}|$$

where $a, b$ are called the input difference and output difference respectively.

The differential uniformity of an S-box is $\nabla_S = \max \{\text{DDT}(a, b) : a \neq 0\}$. The S-box is said to be nonlinear if $\nabla_S \leq 2^n$. Also the larger value of differential uniformity for an S-box indicates the high insecurity of the block cipher using this S-box.

If $\nabla_S = 2$ then the function $S$ is called almost perfect nonlinear (APN) function [46]. The inversion mapping $S(x) = x^{2^n - 2}$ with $x \in \mathbb{F}_2^n$, $n$ even is differentially 4 uniform [44]. The substitution boxes $S(x) = x^d$ with $x \in \mathbb{F}_2^n$ for $n$ even, where $d = 2^n - 2^i - 1$ for $i = 1, 2, ..., n - 1$ is differentially 4 uniform.

**Algebraic Immunity**

The algebraic immunity of an S-box is a measure for the complexity of a very general type of algebraic attacks, considering implicit or conditional equations [47]. Algebraic attacks can be efficient against stream ciphers.
based on LFSRs. In stream ciphers, the algebraic immunity of the filter function is a measure for the complexity of algebraic attacks. However, it turned out in some cases that algebraic immunity does not work well for fast algebraic attacks. It remains an open problem if the immunity against fast algebraic attacks is a sufficient criterion for any kind of algebraic attacks on stream ciphers. There is still no efficient method for computation of algebraic immunity of S-boxes for larger variables. The idea of algebraic attack is to set up an algebraic system of equations in key bits and to try to solve it. For such an attack to work, it is crucial that the combining functions have low degree multiples or low degree annihilators. This raises the fundamental issue of determining whether or not a given function has non-trivial low degree multiples or annihilators [35] [48] [49] [50]. The smallest degree for which this happens is called the algebraic immunity of the function.

**Correlation Immunity**

Consider a stream cipher where the state bits of one or more linear feedback shift registers are filtered by an S-box \( S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m \) to form keystream bits. The keystream bits will be XORed with the plaintext to form the ciphertext. Conventionally, an adversary who wants to perform correlation attack on this stream cipher tries to find an approximation of a linear combination of output bits by a linear combination of input bits given as \( u \cdot S(x) \approx w \cdot x \). For correlation attack to be successful, we require that the bias defined by 

\[
\text{Bias} = |\Pr(u \cdot S(x) = w \cdot x)| - \frac{1}{2}, \ u \in \mathbb{F}_2^n, w \in \mathbb{F}_2^n
\]

is large. Conversely, if all linear approximations of \( u \cdot S(x) \) have small bias, then it is secure against correlation attack.

At Crypto 2000, Zhang and Chan [51] observed that instead of taking linear combination of the output bit functions \( u \cdot S(x) \), one can write \( S(x) \) with any Boolean function \( g : \mathbb{F}_2^n \rightarrow \mathbb{F}_2 \) and consider the probability \( \Pr(g(z) = w \cdot x) \), where \( z = S(x) \) corresponds to the output keystream. Hence \( g(z) \approx w \cdot x \) is a linear approximation which can be used in correlation attacks. Now in this case the choice of linear approximations is larger which enables to find linear approximations with larger bias \( |\Pr(g(z) = w \cdot x)| - \frac{1}{2} \). To measure the effectiveness of Zhang and Chan correlation attack, the
notion of unrestricted nonlinearity [52] was introduced by Carlet, Prouff, where they deduced that a high unrestricted nonlinearity is required for protection against correlation attack. Later in the year 2007, Carlet et al. [53] introduced a linear approximation for performing correlation attack, which is more effective than the attack described by Zhang-Chan attack [51]. This attack is known as generalized correlation attack which leads to the definition of generalized correlation immunity. Both usual Correlation Immunity and Generalized Correlation Immunity are defined as follows:

**Definition 3.1.2** (Correlation Immunity of an S-box). The vector function $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ is correlation immune of order $t$ denoted as CI$(t)$ if $u \cdot S(x) \oplus w \cdot x$ is balanced for $1 \leq w_H(w) \leq t$. Moreover if $S(x)$ is balanced, then $S(x)$ is $t$-resilient. Resiliency is essential for protection against divide-and-conquer correlation attack on combinatorial generator described by Siegenthaler. By describing a generalized divide-and-conquer attack (generalized correlation attack) against stream ciphers, Carlet in [53] defines the generalized correlation immunity of S-boxes.

**Definition 3.1.3** (Generalized Correlation Immunity of S-boxes). Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ and let $g, h_i : \mathbb{F}_2^m \to \mathbb{F}_2$, $i = 1, 2, ..., n$ be $m$ variable Boolean functions. Then $S(x)$ is generalized correlation immune of order $t$ if $g(S(x)) \oplus h_1(S(x))x_1 \oplus \ldots \oplus h_n(S(x))x_n$ is balanced, whenever $1 \leq w_H(h_1(z), h_2(z), ..., h_n(z)) \leq t, \forall z \in \mathbb{F}_2^m$.

Moreover if $S(x)$ is balanced, then we say that $S(x)$ is generalized $t$-resilient. It is to be noted that the usual correlation immunity given in definition 3.1.2 is equivalent to the generalized correlation immunity given in definition 3.1.3. Which implies that an $S(x)$ is correlation immunity of order $t$ iff it is generalized correlation immune of order $t$.

**Nonlinearity**

Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an S-box having an $n$-bit input and an $n$-bit output. For any given $a, b \in \mathbb{F}_2^n$, the linear approximation table (LAT) can be
constructed using
\[ \text{LAT}(a, b) = |x \in \mathbb{F}_2^n : a \cdot x = b \cdot S(x)| - 2^{n-1} \] (3.2)

where \( a \cdot b \) denotes the parity (0 or 1) of bitwise product of \( a \) and \( b \). LAT is an important tool to measure the security of the S-boxes against linear cryptanalysis. Large elements of LAT are not desired since they indicate high probability of linear relations between the input and output. Here \( a \) and \( b \) are the input and output masks. The nonlinearity measure of an \( n \times n \) order S-box is related with the maximum entry of the LAT table and is defined as
\[ \text{NL} = 2^{n-1} - \max |\text{LAT}(a, b)| \] (3.3)

The AES S-box is based on inversion mapping over \( \mathbb{F}_2^8 \), which is differentially 4 uniform has nonlinearity 112.

**Robustness**

Let \( L \) be the largest value in the difference distribution table (DDT) of the S-box, and \( R \) be the number of nonzero entries in the first column of the table. In either case the value \( 2^n \) in the first row first entry is not counted. Then we say that the S-box is \( \epsilon \) - robust [54] against differential cryptanalysis, where \( \epsilon \) is defined by \( \epsilon = (1 - \frac{R}{2^n})(1 - \frac{L}{2^n}) \). The Robustness of an S-box is small if \( R \) and \( L \) is large. Such an S-box is extremely prone to differential cryptanalysis.

The construction of S-boxes based on power mappings over finite fields provides security against differential and linear attacks. To give an insight into a list of such S-boxes with their cryptographic properties are shown in table 3.1. These S-boxes in the table are generated using the finite field \( \mathbb{F}_2^8 \), the technique which is used to generate the AES S-box. By this method an S-box is defined as \( S(x) = Ax^d \oplus b \), here \( A \) is an invertible \( 8 \times 8 \) matrix defined over \( \mathbb{F}_2 \), \( d \in \{1, 2, \ldots, 256\} \), \( b \in \mathbb{F}_2^8 \). Affine transformations do not modify cryptographic properties of an S-box but they may improve their algebraic expressions. The S-boxes given in table 3.1 are classified with respect to the power \( d \), the matrix \( A \) and the vector \( b \) are chosen to be the same which are used for generating the AES S-box.
Table 3.1: Classification of power mappings

<table>
<thead>
<tr>
<th>$d$</th>
<th>$\nabla$</th>
<th>Balance</th>
<th>Robust</th>
<th>NL</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2</td>
<td>256</td>
<td>balanced</td>
<td>0.000000</td>
<td>0</td>
<td>No, only 5 component functions of $S$ are complete</td>
</tr>
<tr>
<td>3, 6, 9, 12, 18, 24, 33, 36, 39, 48, 57, 66, 72, 78, 96, 114, 129, 132, 144, 147, 156, 192, 201, 228</td>
<td>2</td>
<td>unbalanced</td>
<td>0.003876</td>
<td>112</td>
<td>yes</td>
</tr>
<tr>
<td>4, 8</td>
<td>256</td>
<td>balanced</td>
<td>0.000000</td>
<td>0</td>
<td>No, only 4 component functions of $S$ are complete</td>
</tr>
<tr>
<td>5, 10, 20, 40, 65, 80, 130, 160</td>
<td>4</td>
<td>unbalanced</td>
<td>0.003845</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>7, 14, 28, 37, 41, 56, 73, 74, 82, 112, 131, 146, 148, 164, 193, 224</td>
<td>6</td>
<td>balanced</td>
<td>0.976562</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>11, 22, 29, 44, 58, 71, 88, 97, 116, 133, 142, 163, 176, 194, 209, 232</td>
<td>10</td>
<td>balanced</td>
<td>0.960938</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>13, 26, 52, 59, 67, 103, 104, 118, 134, 157, 161, 179, 206, 208, 217, 236</td>
<td>12</td>
<td>balanced</td>
<td>0.953125</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>Set,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240</td>
<td>14</td>
<td>unbalanced</td>
<td>0.003693</td>
<td>116</td>
<td>yes</td>
</tr>
<tr>
<td>16, 32</td>
<td>256</td>
<td>balanced</td>
<td>0.000000</td>
<td>0</td>
<td>No, only 3 component functions of S are complete</td>
</tr>
<tr>
<td>17, 34, 68, 136</td>
<td>16</td>
<td>unbalanced</td>
<td>0.003662</td>
<td>120</td>
<td>yes</td>
</tr>
<tr>
<td>19, 38, 47, 49, 76, 94, 98, 121, 137, 151, 152, 188, 196, 203, 229, 242</td>
<td>16</td>
<td>balanced</td>
<td>0.937500</td>
<td>104</td>
<td>yes</td>
</tr>
<tr>
<td>21, 42, 69, 81, 84, 111, 123, 138, 162, 168, 183, 189, 219, 222, 237, 246</td>
<td>4</td>
<td>unbalanced</td>
<td>0.003845</td>
<td>112</td>
<td>yes</td>
</tr>
<tr>
<td>23, 46, 53, 61, 77, 79, 83, 92, 106, 113, 122, 139, 154, 158, 166, 167, 169, 184, 197, 211, 212, 226, 233, 244</td>
<td>16</td>
<td>balanced</td>
<td>0.937500</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>25, 35, 50, 70, 100, 140, 145, 200</td>
<td>6</td>
<td>unbalanced</td>
<td>0.003815</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>27, 54, 99, 108, 141, 177, 198, 216</td>
<td>26</td>
<td>unbalanced</td>
<td>0.003510</td>
<td>80</td>
<td>yes</td>
</tr>
<tr>
<td>31, 62, 91, 107, 109, 124, 143, 173, 181, 182, 199, 214, 218, 227, 241, 248</td>
<td>16</td>
<td>balanced</td>
<td>0.937500</td>
<td>112</td>
<td>yes</td>
</tr>
</tbody>
</table>
### 3.1.3 MDS Matrices

One of the fundamental properties of ciphers is their degree of diffusion, which is often achieved by error-correcting codes, popularly known as Maximum Distance Separable (MDS) codes. The use of MDS matrix in cryptography to provide perfect diffusion was proposed by Vaudenay [55]. The linear code consisting of words formed by the concatenation of inputs and

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Balanced/Unbalanced</th>
<th>Probability</th>
<th>Value</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>43, 86, 89, 101, 149, 172, 178, 202</td>
<td>balanced</td>
<td>0.882812</td>
<td>80</td>
<td>yes</td>
</tr>
<tr>
<td>51, 102, 153, 204</td>
<td>unbalanced</td>
<td>0.003143</td>
<td>116</td>
<td>yes</td>
</tr>
<tr>
<td>55, 110, 115, 155, 185, 205, 220, 230</td>
<td>unbalanced</td>
<td>0.003723</td>
<td>96</td>
<td>yes</td>
</tr>
<tr>
<td>63, 126, 159, 207, 231, 243, 249, 252</td>
<td>unbalanced</td>
<td>0.003815</td>
<td>104</td>
<td>yes</td>
</tr>
<tr>
<td>64, 128</td>
<td>balanced</td>
<td>0.000000</td>
<td>0</td>
<td>No, only 2 component functions of $S$ are complete</td>
</tr>
<tr>
<td>85, 170</td>
<td>unbalanced</td>
<td>0.002625</td>
<td>118</td>
<td>No, none of the component functions of $S$ are complete</td>
</tr>
<tr>
<td>87, 93, 117, 171, 174, 186, 213, 234</td>
<td>unbalanced</td>
<td>0.003448</td>
<td>80</td>
<td>yes</td>
</tr>
<tr>
<td>119, 187, 221, 238</td>
<td>unbalanced</td>
<td>0.003571</td>
<td>112</td>
<td>yes</td>
</tr>
<tr>
<td>127, 191, 223, 239, 247, 251, 253, 254</td>
<td>balanced</td>
<td>0.984375</td>
<td>112</td>
<td>yes</td>
</tr>
<tr>
<td>255, 256</td>
<td>balanced</td>
<td>0.000031</td>
<td>127</td>
<td>No, none of the component functions of $S$ are complete</td>
</tr>
<tr>
<td>95, 125, 175, 190, 215, 235, 245, 250</td>
<td>unbalanced</td>
<td>0.003845</td>
<td>112</td>
<td>yes</td>
</tr>
</tbody>
</table>
outputs of a linear diffusion layer should have the best possible minimum distance to ensure optimal diffusion. Hence, MDS codes having the largest possible minimum distance are a good choice from a security point of view. A matrix is an MDS matrix if and only if every sub-matrix is non-singular. Reed-Solomon codes have the MDS property and are frequently used to obtain the MDS matrices used in cryptographic algorithms. In the ciphers, MDS matrices are multiplied in the field $\mathbb{F}_{2^n}$ to provide diffusion or mix the input bits. There are a few block and stream ciphers which uses MDS matrices to provide diffusion.

Though the use of MDS matrices for diffusion provides good security, it is costly for hardware implementation. Even the most compact implementation of AES which is $\mathbb{F}_{2^8}$ multiplication of an MDS, consumes large percentage of the total Gate Equivalents (GE) for diffusion layer.

**Definition 3.1.4 (MDS Matrix).** An $m \times n$ matrix over a finite field $K$ is an MDS matrix if it is the transformation matrix of an MDS code. In other words, an $(n, k, d)$ code with generator matrix $G = [I|A]$, where $A$ is a $k \times (n-k)$ matrix and $I$ is an identity matrix, is MDS iff every square sub-matrix of $A$ is non-singular.

An MDS code is a linear code $(n, k, d)$ over any field, with $d = n - k + 1$. MDS matrices having entries with maximum number of 1’s and minimum number of distinct elements with low Hamming weights are considered to be efficient. In the AES Mix Column operation, the MDS matrix is a circulant matrix having elements of low Hamming weights, but the number of 1’s in this matrix is eight.

One approach for checking if a $d \times d$ matrix $M$ is MDS is to use $[I|M]$ as a generator matrix and check if the code produces MDS code. In this case if the underlying field is $\mathbb{F}_{2^n}$, then the number of codewords will be $2^{dn}$ and finding the minimum distance of this code is NP-complete. The number of computations in this case will be

$$n^2 \sum_{i=1}^{d} \binom{d}{i} i^3.$$  

Junod and Vaudenay proposed efficient MDS matrices [56] by maximizing the number of 1’s and minimizing other constants in the matrix, which
means MDS matrices needs to be bi-regular arrays. A $2 \times 2$ array with entries in the field is bi-regular if at least one row and one column of the matrix have two different entries. Example of a bi-regular $4 \times 4$ matrix is given below:

$$
\begin{pmatrix}
a & 1 & 1 & 1 \\
1 & 1 & b & a \\
1 & a & 1 & b \\
1 & b & a & 1
\end{pmatrix}
$$

The optimal matrix consists of many 1’s and a few distinct constants $a, b$. It is obvious that multiplying with this method will yield efficient results because of many transparent multiplications with 1’s.

Few examples of MDS matrices used in block ciphers are given below in table 3.2 (entries of the matrix are written in hexadecimal format).

Table 3.2: MDS matrices used in various ciphers

<table>
<thead>
<tr>
<th>MDS matrix</th>
<th>Extended field</th>
<th>Field polynomial $(p(x))$</th>
</tr>
</thead>
</table>
| \[
\begin{bmatrix}
01 & ef & 5b & 5b \\
5b & ef & ef & 01 \\
ef & 5b & 01 & ef \\
ef & 01 & ef & 5b \\
\end{bmatrix}
\] | $\mathbb{F}_{2^8}[x]/\langle p(x) \rangle$ | $x^8 + x^6 + x^5 + x^3 + 1$ used in: Twofish cipher [57] |
| \[
\begin{bmatrix}
02 & 03 & 01 & 01 \\
01 & 02 & 03 & 01 \\
01 & 01 & 02 & 03 \\
03 & 01 & 01 & 02 \\
\end{bmatrix}
\] | $\mathbb{F}_{2^8}[x]/\langle p(x) \rangle$ | $x^8 + x^4 + x^3 + x + 1$ used in: AES [7] |
| $\begin{bmatrix} 05 & 03 & 1e & 10 \\ 04 & 01 & 02 & 0e \\ 10 & 1e & 03 & 05 \\ 0e & 02 & 01 & 04 \end{bmatrix}$ | $F_{2^8}[x]/\langle p(x) \rangle$ | $x^8 + x^4 + x^3 + x + 1$ |
| $\begin{bmatrix} 0a & 1e & 10 & 0e \\ 1e & 0a & 0c & 0d \\ 02 & 0b & 0d & 0c \\ 0e & 0d & 0b & 0a \end{bmatrix}$ | $F_{2^8}[x]/\langle p(x) \rangle$ | $x^8 + x^4 + x^3 + x + 1$ |
| $\begin{bmatrix} 01 & 03 & 04 & 05 \\ 03 & 01 & 05 & 04 \\ 04 & 05 & 01 & 03 \\ 05 & 04 & 03 & 01 \end{bmatrix}$ | $F_{2^8}[x]/\langle p(x) \rangle$ | $x^8 + x^4 + x^3 + x + 1$ |
| $\begin{bmatrix} 01 & 03 & 04 & 05 & 06 & 08 & 0b & 07 \\ 03 & 01 & 05 & 04 & 08 & 06 & 07 & 0b \\ 04 & 05 & 01 & 03 & 0b & 07 & 06 & 08 \\ 05 & 04 & 03 & 01 & 07 & 0b & 08 & 06 \\ 06 & 08 & 0b & 07 & 01 & 03 & 04 & 05 \\ 08 & 06 & 07 & 0b & 03 & 01 & 05 & 04 \\ 0b & 07 & 06 & 08 & 04 & 05 & 01 & 03 \\ 07 & 0b & 08 & 06 & 05 & 04 & 03 & 01 \end{bmatrix}$ | $F_{2^8}[x]/\langle p(x) \rangle$ | $x^8 + x^4 + x^3 + x^2 + 1$ |

used in: Khazad block cipher [58]
3.1.4 T-Functions

T-functions are found to be useful for the design of fast cryptographic primitives and ciphers based on usage of both arithmetic and logical operations. A T-function is a mapping from \( k \)-bit words into \( k \)-bit words such that each \( i \)-th bit of the output depends only on the low-order bits \( 0, ..., i \) of the pre-image \([59] [60] [61] [62] [63] [64]\). All the logical operations, such as XOR, AND, OR, NOT and most of the arithmetic operations modulo \( 2^k \), such as addition, multiplication, subtraction, negation, left shift and their compositions are T-functions. Various methods are known to construct bijective T-functions as well as transitive T-functions. Transitive T-functions are the ones that produce sequences of the longest possible period \( 2^k \) which have been considered as a candidate to replace LFSRs in keystream generators of stream ciphers \([60] [64] [65] [66] [67] [68] [69]\). Sequences produced by T-function based keystream generators are proved to have a number of good cryptographic properties and bijective T-functions can be used to design filter functions of stream ciphers. Klimov and Shamir proposed in their paper \([61]\) a T-function defined as \( f(x) = x + (x^2 \lor C) \mod 2^n \), where \( C \) is a constant integer satisfying \( C \equiv 5 \) or \( 7 \) \( (\mod 2^3) \) used as a pseudorandom number generator. In the construction of synchronous stream ciphers, the T-function is expected to have the single cycle property, which means that the T-function’s repeated application to any initial state goes through all the possible states. To characterize the single cycle property and other cryptographic properties of T-functions, a general theory of T-function over \( p \)-adic integer rings were developed \([70]\) and used \( p \)-adic analysis to construct wide classes of T-functions with provable cryptographic properties.

One drawback with T-functions is that they are efficient only when the word size is the word size of the underlying processor as these functions can be evaluated using simple machine instructions. One possible solution is to use T-functions only as part of a stream cipher construction and not as the keystream generator as a whole. Consider the binary field \( \mathbb{F}_2 \) and \( n \) be an arbitrary non-negative integer. Represent \( x \in \mathbb{F}_2^n \) as a vector \( x = ([x]_0, [x]_1, ..., [x]_{n-1}) \) of length \( n \), where \( [x]_i \) denotes its \((i+1)\)-th bit and \([x]_0\) is the least significant bit of the word \( x \).

**Definition 3.1.5 (T-Function).** A function \( f \) from \( \mathbb{F}_2^n \) to \( \mathbb{F}_2^n \) is called a
T-function if the $i$-th bit of the output $[f(x)]_{i-1}$ depends only on the first $i$ bits of the input $[x]_0, [x]_1, \ldots, [x]_{i-1}$

$$
\begin{bmatrix}
[x]_0 \\
[x]_1 \\
\vdots \\
[x]_{n-1}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
f_0([x]_0) \\
f_1([x]_0, [x]_1) \\
\vdots \\
f_{n-1}([x]_0, [x]_1, \ldots, [x]_{n-1})
\end{bmatrix}
$$

**Definition 3.1.6** (Single Cycle T-Function). Let $f(x) = ([f(x)]_0, [f(x)]_1, \ldots, [f(x)]_{n-1})$ be an invertible T-function over $\mathbb{F}_2^n$. Then $f$ is a single cycle T-function if and only if its Algebraic Normal Form (ANF) has the following form.

$$
[f(x)]_0 = [x]_0 \oplus 1
$$

$$
[f(x)]_i = [x]_i \oplus [x]_0[x]_1\ldots[x]_{i-1} \oplus g_i([x]_0, [x]_1, \ldots, [x]_{i-1})
$$

Where $g_i$ is an $i$ variable Boolean function and degree of $g_i$, $\text{deg}(g_i) < i$, $i \geq 1$.

**Definition 3.1.7** (Multi-Word T-function). A multi-word T-function is a map $T : (\{0, 1\}^n)^m \rightarrow (\{0, 1\}^n)^m$, $x \mapsto T(x) = (T_k(x))_{k=0}^{m-1}$ sending an $m$-tuple of $n$-bit words to another $m$-tuple of $n$-bit words, where each resulting $n$-bit word is denoted as $T_k(x)$, such that for each $0 \leq i < n$, the $i$-th bits of the resulting words $[T_k(x)]_i$ are functions of just the lower input bits $[x]_0, [x]_1, \ldots, [x]_i$, where $[x]_i$ is the $i$-th bits of the $m$-tuple of words $x$.

The set of words on which a T-function acts is sometimes referred to as memory or register and the bit values it contains are said to form a state of the memory. Given a T-function $T$, one may fix an initial state $X^0$ for the memory and iteratively act $T$ on it to obtain a sequence defined by $X^{t+1} = T(X^t)$. Such a sequence will always be eventually periodic and if its periodic part passes through all of the $2^{nm}$ possible states the memory, then the T-function is said to form a single cycle. A single cycle T-function may serve as a good building block for a stream cipher.

**Definition 3.1.8** (Multi-Word Parameter). A multi-word parameter is a map $\alpha : (\{0, 1\}^n)^m \rightarrow \{0, 1\}^n$, $x \mapsto \alpha(x)$ sending an $m$-tuple of $n$-bit words to a single $n$-bit word such that for each $0 \leq i < n$ the $i$-th bit of the
resulting word \( [\alpha(x)]_i \) is a function of just the strictly lower input bits \([x]_0, [x]_1, \ldots, [x]_{i-1}\).

A parameter is a sort of multi-word to single-word T-function for which the \(i\)-th output bit does not depend on the \(i\)-th input bits \([x]_i\).

**Definition 3.1.9** (Multi-Word Single Cycle T-function). The T-function defined by setting \( x \mapsto T(x) = (T(x))_{k=0}^{m-1} \) with \( T_k(x) = x_k \oplus (\alpha_k(x) \land x_0 \land x_1 \land \ldots \land x_{k-1}) \) for \( k = 0, 1, \ldots, m - 1 \), exhibits the single cycle property, when each \( \alpha_k \) is an odd parameter.

Compared to the T-function in definition 3.1.9, Jin Hong et al. ([59], def. 3.7) defined a T-function, which certainly gives a better column mixing.

**Definition 3.1.10.** Let \( S \) be a single cycle S-box and let \( \alpha \) be an odd parameter. If \( S^0 \) is an odd power of \( S \) and \( S^e \) is an even power of \( S \), the mapping \( T(x) = (\alpha(x).U^0(x)) \oplus (\overline{\alpha(x)}.U^e(x)) \) defines a single cycle T-function. Where \( U \) is a mapping defined as \( U : (\{0,1\}^n)^m \rightarrow (\{0,1\}^n)^m \), \( x \rightarrow U(x) \) by setting \( [U(x)]_i = S([x]_i) \).

We say that an S-box has the single cycle property if its cycle decomposition gives a single cycle. That is, starting from any point, if we iteratively act \( S \), we end up going through all possible elements of \( \{0,1\}^m \). \( U \) will not define a single cycle T-function, even when the S-box \( S \) is of a single cycle. Define the logical operations on multi-words as follows:
\( x \mapsto (x_k)_{k=0}^{m-1} \) and \( y \mapsto (y_k)_{k=0}^{m-1} \) be two multi-words then \( x \oplus y = (x_k \oplus y_k)_{k=0}^{m-1} \), and for a single word \( \alpha, \alpha \cdot x = (\alpha \land x_k)_{k=0}^{m-1} \). \( \overline{\alpha(x)} \) denotes the bitwise complement of \( \alpha \).

### 3.2 Design Criteria for Stream Ciphers

Stream ciphers use a keystream generator to generate keystream, which is combined with a message to encrypt or decrypt the message. The most common encryption and decryption function is the binary addition modulo 2, known as the XOR operation. It is used as it is fast and easy to implement in both hardware and software. The keystream and message are then
combined using the XOR operation to produce the ciphertext. Traditional stream ciphers take only single input known as the key which is kept secret. Later to solve the problem of key management due to frequent re-keying, modern stream ciphers are fed with two inputs, one being public known as the initial vector (IV) and the second input kept secret with the end users known as the key. To decrypt the message, the receiver must use the same key and IV to initialize the keystream generator there by producing the same keystream. Keysteam generators for stream ciphers operate by maintaining an internal state and applying update and output function to the state. The operation of a keystream generator has two phases:

a) Initialization Phase

b) Keystream Generation Phase

The initialization phase is further divided into two phases:

i) Key and IV loading phase

ii) State updation phase

**Initialization Phase**

Before encryption of any given message, the keystream generator is needed to go through this phase which is used to form the internal state of the keystream generator. The goal of this phase is to diffuse the key and IV pair well across the entire internal state, so that no mathematical or statistical relationships exits between the key, IV and the corresponding keystream.

**Key-IV Loading Phase**

Here the key and IV are transferred into the internal state of the cipher by using some pre-defined fixed rule. Mostly the secret key and IV are loaded into the state of the generator. At the end of this phase we get a state which we shall call as the loaded state to remove any ambiguity.
**State Updation Phase**

This phase consists of fixed number of iterations (or rounds) say $\delta$, which is used for updating the loaded state with the help of the state update function $U$. The goal of this phase is to provide proper diffusion to the internal state of the generator. To achieve this, a very careful choice of $\delta$ is required. Small value of $\delta$ is ideal for quick performance of the generator which is required for real time applications where frequent rekeying is done. On the contrary, selecting smaller values for $\delta$ does not provide sufficient diffusion leaving the cipher vulnerable to attacks. At the end of this phase we get a state which is known to be the initial state. This phase is followed by the keystream generation phase.

**Keystream Generation Phase**

During this phase the internal state of the generator is updated using a state update function $U$. A single keystream symbol (atleast one bit) is generated for every time we use the state update function to update the internal state. In some cases, the update function used in the initialization phase may not be the same in this phase, and this depends upon the designer. By applying an output function to the internal state, sufficient keystream is generated to encrypt the message.

![Block diagram for different phases of stream cipher](image)

Figure 3.1: Block diagram for different phases of stream cipher
3.2.1 Stream Ciphers

In general stream ciphers are categorized into two types. The first type are those which are derived from block ciphers, and the second are those designed with the help of some dedicated building blocks like Boolean functions, LFSRs etc. Some examples of such stream ciphers are Trivium, Grain, MICKEY, SNOW, Dragon, Salsa, RC4, WG Family, HC-128 etc. Stream ciphers require pseudorandom generators which generates large amounts of keystream. One way of achieving this is to make the keystream periodic. The long periodic keystream can be reproduced by using a short generating key. Since random long bit strings are hard to obtain, in most cryptographic applications we must be satisfied with pseudorandom strings. A pseudorandom bit generator (PRBG) is an efficient algorithm that takes as input a short truly random bitstring known as seed and produces a long bitstring as output. Each statistical test passed by a PRBG shows that the generator does not have a certain statistical weakness, but no matter how many tests we try we cannot be certain that there is no new statistical weakness that has been overlooked. We say that a PRBG passes all polynomial-time statistical tests if no polynomial-time algorithm can distinguish the output of the generator from a truly random bitstring of the same length with a probability significantly greater than $\frac{1}{2}$.

Synchronous Stream Ciphers

Stream ciphers are divided into synchronous and asynchronous (self synchronous) modes. Synchronous stream ciphers are an important class of symmetric encryption algorithms. In synchronous stream ciphers, the keystream is independent from plaintext and ciphertext, and therefore can be pre-computed and will be unaffected by transmission errors. Bit flips in ciphertext affect only a single bit in the corresponding plaintext, which can be useful if the transmission error rate is high. A stream cipher encrypts one digit of plaintext at a time with a time-varying transformation. Let $S$ be the internal state of the cipher. Given a key $K$, if the initial state is given as $S_0$, then for each cipher clock $i = 0, 1, 2, ...$ the encryption process is governed by the following three equations as given in 1.3 which generates keystream $z_t$, ciphertext $c_t$ and the next state $S_t$ at time $t$: 
Keystream Function: \( H(S_t, K) = z_t \) where \( H \) is the output function

Encryption Function: \( E(z_t, m_t) = c_t \)

Next State Function: \( S_{t+1} = U(S_t) \) here \( U \) is the update function

An additive binary stream cipher is defined to be a synchronous stream cipher in which \( E \) is equal to addition modulo 2 operation.

![Block representation of synchronous stream cipher](image)

**Properties**

- Sender and Receiver must be synchronized using the same key and operating at the same state within that key.
- Insertion/Deletion/Replay cause loss of synchronization.
- Re-synchronization may need re-initialization and/or special marks in the stream at regular intervals.
- Modified digit does not affect decryption of other digits.
- Due to lack of error propagation, the adversary can determine ciphertext and plaintext pairs.
Asynchronous or Self Synchronous Stream Ciphers

A self-synchronous or asynchronous stream cipher is one in which the keystream is generated as a function of the key and a fixed number of previous ciphertext digits. Self-synchronous or asynchronous stream ciphers use a finite number of previous \( n \) ciphertext digits to compute the keystream. They are also known as ciphertext autokey (CTAK). This approach has the advantage that the receiver will automatically synchronize with the keystream generator after receiving a fixed finite number of ciphertext digits, making it easier to recover the message stream if digits are dropped or added to the ciphertext stream.

\[ S_i = (c_{i-t}, c_{i-t+1}, ..., c_{i-1}), \quad S_0 = (c_{-t}, c_{-t+1}, ..., c_{-1}) \]

is the initial state

Keystream: \( z_i = H(S_i, K) \)

Ciphertext: \( c_i = E(z_i, m_i) \)

General Block diagram of self-synchronous stream cipher is given below.

![Figure 3.3: Encryption in self-synchronous stream cipher](image)

![Figure 3.4: Decryption in self-synchronous stream cipher](image)

Properties

- An insertion, deletion, or change in ciphertext characters results in loss of only a fixed number of deciphered plaintext characters, after
which the deciphering self-synchronizes.

- A ciphertext error in transmission affects at most $t$ characters of the deciphered plaintext.
- Better diffusion of plaintext statistics.

Self-synchronizing stream ciphers are much more closely related to block ciphers than synchronous stream ciphers. The most widely adopted approach to self-synchronizing stream encryption is the use of a block cipher in CFB mode. Single-bit self-synchronizing stream encryption has a specific advantage over all other types of encryption. In providing an existing communication system with encryption, single-bit self-synchronizing stream encryption can be applied without the need for additional synchronization or segmentation. For channels that suffer from error bursts, self-synchronizing stream encryption is the likely solution. Examples of self-synchronous stream ciphers are Moustique [71], Mosquito [72] and cipher Hiji-bij-bij [73].

**One Time Pad**

Stream ciphers are derived from the one-time pad (OTP) that uses a key of the same length as the plaintext and has been proven to be theoretically secure. Stream ciphers deduce a keystream (endless and never repeating in the ideal case, but at least with a period that is much longer than the number of encrypted bits) from a shorter key to overcome the disadvantage of requiring such a long key. The output of the keystream generator at a specific time is determined by an internal state. Both state and key must be impossible to recover by looking at the keystream, moreover it should be indistinguishable from random noise. From a theoretical point of view, analyzing the security of a stream cipher can be narrowed down completely to the analysis of the pseudorandomness of the keystream.

**Linear Feedback Shift Register (LFSR)**

An LFSR is essentially an elementary algorithm for generating a keystream, which has the following desirable properties:
• Easy to implement in hardware.
• Produce sequences of long period.
• Produce sequences with good statistical properties.
• Can be readily analyzed using algebraic techniques.

**Definition 3.2.1 (Galois Field).** The Galois field $\mathbb{F}_{p^n}$ is a quotient ring $\mathbb{F}_p[x]/(g(x))$ where $g(x)$ is any monic irreducible polynomial of degree $n$ with coefficients in $\mathbb{F}_p$, and $(g(x))$ is the maximal principle ideal generated by $g(x)$.

**Definition 3.2.2 (Primitive Polynomial).** A polynomial $f(x)$ is a primitive polynomial if and only if the field element $x$ generates the cyclic group of non-zero field elements of the finite field $\mathbb{F}_{p^n}$ where $p$ is prime and $n \geq 2$.

As a generator of the non-zero field elements, $f(x)$ satisfies the equations $x^{p^n-1} \neq 1 (mod f(x))$ and $x^m \neq 1 (mod f(x))$ for $1 \leq m \leq p^n - 2$

Number of distinct primitive polynomials of degree $n$ is given by $\frac{\phi(p^n-1)}{n}$, where $\phi$ is the Euler totient function.

An LFSR is defined by $n$ stages, labeled $r_0, r_1, ..., r_{n-1}$, each storing $l$ bits ($l > 0$), and having $l$ input and output bits, and a timer which mark clock cycles $i = 0, 1, 2, ...$. When $l = 1$, the LFSR is known as bit oriented otherwise word oriented.

The $i$-th clock cycle is defined as:

- The contents of stage 0 is taken as output.
- The contents of $r_i$ is shifted to $r_{i-1}$, $1 \leq i \leq n - 1$
- Stage $r_{n-1}$ is the sum of a pre-arranged subset of stages $0, 1, ..., n - 1$.

Denote the contents of stage $r_j$ at time $i$ by $s_{i+j}$, and the algorithm for updating the contents of stage $r_{n-1}$ is given below as a recurrence relation

$$s_{n+i} = \sum_{j=0}^{n-1} c_{n-j}s_{i+j} \quad (3.4)$$

46
where \( c_i, s_i \in \mathbb{F}_2 \), \( i = 0, 1, \ldots, n \) and \( c_0 = 1 \). We can express the relation in \( 3.4 \) as

\[
\sum_{j=0}^{n} c_{n-j}s_{i+j} = 0
\]

We can identify the constants \( c_k \) with the coefficients of a polynomial

\[
g(x) = \sum_{k=0}^{n} c_k x^k
\]

which in general called as the connection polynomial of the LFSR. Moreover, if we consider the output bits of the LFSR \( s_j \) as the coefficients of a power series

\[
s(x) = \sum_{j=0}^{\infty} s_j x^j
\]

Then \( s(x)g(x) \) gives a polynomial \( f(x) \) of degree less than \( n \) or in other words the power series in relation \( 3.5 \) takes the form \( s(x) = f(x)/g(x) \).

An LFSR is said to be nonsingular if \( c_n \neq 0 \). There are \( 2^n \) possible states leading to a maximal period \( 2^n - 1 \) for any LFSR of length \( n \). The connection polynomial of an LFSR with maximal period is called primitive polynomial. A polynomial \( g(x) \) of degree \( n \) is said to be primitive if the order of \( x \) modulo \( g(x) \) equals \( 2^n - 1 \). The output sequence of an LFSR has period \( N \) if and only if \( (x^N + 1)s(x) \) is polynomial of degree at most \( N - 1 \).

An irreducible connection polynomial of a LFSR must divide \( (x^N + 1) \) where \( N \) is the period of any nonzero output sequence. Hence the period of a sequence generated by an LFSR is independent of the nonzero initial state if the connection polynomial is irreducible and the period achieves maximum \( 2^n - 1 \) only when the connection polynomial is primitive. These maximum length sequences generated by LFSR are known as \( m \)-sequences (maximal length linear sequences) which are said to have good cryptographic properties.

For cryptographic purposes, it is desirable to have sequences with very long period and the period grows exponentially with the length of the LFSR. For a bit based LFSR, we need to use primitive polynomials over \( \mathbb{F}_2 \), and primitive polynomial over the extension fields of \( \mathbb{F}_2 \) for word based LFSR. Keeping in mind the cryptographic significance of LFSRs of lengths 80 and 128, we list a few primitive polynomials of degree 80 and
128 over the field $\mathbb{F}_2$ in table 3.3. If the primitive polynomials have a small number of terms, then the output of the LFSR is easily computed. But from the point of view of cryptology applications it is often necessary to have the primitive polynomials with large number of terms for resisting correlation attacks [13]. Examples of stream ciphers whose design is based on bit based LFSR are Grain v1, A5/1. Some of the stream cipher designs based on word based LFSR are Turing, SOBER, WG-29, WG-8, WG-7, Snow.

Table 3.3: Primitive Polynomials over the field $\mathbb{F}_2$

<table>
<thead>
<tr>
<th>Degree</th>
<th>Primitive polynomials (with power of each term listed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>80, 70, 51, 50, 37, 10, 0</td>
</tr>
<tr>
<td>80</td>
<td>80, 75, 27, 17, 0</td>
</tr>
<tr>
<td>80</td>
<td>80, 67, 57, 42, 29, 18, 0</td>
</tr>
<tr>
<td>128</td>
<td>128, 121, 90, 58, 47, 32, 0</td>
</tr>
<tr>
<td>128</td>
<td>128, 77, 35, 11, 0</td>
</tr>
<tr>
<td>128</td>
<td>128, 95, 57, 45, 38, 36, 0</td>
</tr>
<tr>
<td>128</td>
<td>128, 113, 96, 80, 63, 48, 32, 16, 0</td>
</tr>
<tr>
<td>128</td>
<td>128, 105, 83, 62, 42, 21, 0</td>
</tr>
<tr>
<td>128</td>
<td>128, 126, 125, 124, 121, 119, 117, 116, 114, 113, 110, 109, 108, 105, 104</td>
</tr>
<tr>
<td></td>
<td>102, 101, 97, 96, 92, 91, 88, 87, 86, 83, 82, 80, 79, 78, 75, 73, 70, 69, 66, 65</td>
</tr>
<tr>
<td></td>
<td>64, 63, 61, 60, 59, 57, 56, 49, 48, 47, 42, 41, 39, 38, 37, 36, 35, 33, 32, 24, 23</td>
</tr>
<tr>
<td></td>
<td>22, 21, 20, 19, 17, 16, 14, 13, 11, 10, 9, 8, 7, 2, 0</td>
</tr>
<tr>
<td>128</td>
<td>128, 127, 126, 124, 123, 122, 120, 118, 116, 115, 114, 113, 110, 109, 102</td>
</tr>
<tr>
<td></td>
<td>101, 100, 99, 97, 96, 93, 92, 91, 90, 89, 88, 87, 85, 82, 78, 77, 75, 74, 72, 70</td>
</tr>
<tr>
<td></td>
<td>69, 67, 66, 63, 62, 60, 59, 57, 56, 55, 52, 51, 48, 46, 43, 38, 32, 27, 25, 20, 19</td>
</tr>
<tr>
<td></td>
<td>18, 16, 14, 13, 11, 10, 9, 7, 6, 5, 4, 3, 0</td>
</tr>
<tr>
<td></td>
<td>100, 99, 98, 97, 96, 94, 91, 88, 86, 85, 80, 79, 78, 77, 74, 73, 71, 70, 68, 66, 63</td>
</tr>
<tr>
<td></td>
<td>60, 58, 57, 56, 54, 53, 50, 48, 46, 44, 40, 38, 37, 36, 35, 33, 31, 30, 26, 25, 22</td>
</tr>
<tr>
<td></td>
<td>20, 17, 11, 10, 9, 6, 5, 3, 2, 0</td>
</tr>
</tbody>
</table>
Testing Primitive Polynomials

Let \( F_q \) be the finite field with \( q \) elements and \( F_q[x] \) be the ring of polynomials in one variable \( X \) with coefficients in \( F_q \). If \( f(X) \in F_q[x] \) is of degree \( n \) and \( f(0) \neq 0 \), then \( f(X) \) divides \( X^e - 1 \) for some positive integer \( e \leq q^n - 1 \). The least \( e \) is called the order of \( f(X) \) and is denoted by \( \text{ord}_f(X) \). We say that a monic polynomial \( f(X) \in F_q[x] \) of degree \( n \) is primitive if \( f(0) \neq 0 \) and \( \text{ord}_f(X) = q^n - 1 \). A basic reference for the study of primitive polynomials is ( [74], Chap.3). In particular it is sufficient to test the primitiveness of \( f(X) \) only for the values \( i = (p^m - 1)/p' \), where \( p' \) is a prime factor of \( p^m - 1 \) in \( x^i \equiv 1 \pmod{f(X)} \).

Word Oriented Linear Feedback Shift Register

Zeng, Han and He [75] introduced word oriented linear feedback shift register, called \( \sigma-LFSR \) which is said to meet the demands of efficiency and good cryptographic properties. It turned out that the notion of \( \sigma-LFSR \) is equivalent to Niederreiter’s multiple recursive matrix method [76]. The author of [76] also conjectured that the number of primitive \( \sigma-LFSR \) of order \( n \) in the binary case which was later extended over the finite field \( F_{q^m} \). Another generalization to LFSRs was considered by Tsaban and Vishne [77] in the form of transformation shift registers (TSRs). A TSR
is a particular case of a $\sigma-LFSR$, and they are efficient when it comes to software implementation. Explicit formula for the number of primitive TSRs is not known. Let $m$ be a positive integer, $\mathbb{F}_{q^m}$ be the finite field with $q^m$ elements. Let $M_m(\mathbb{F}_2)$ denote the $m \times m$ matrix ring over $\mathbb{F}_2$ and $GL_m(\mathbb{F}_2) \subseteq M_m(\mathbb{F}_2)$ be the general linear group. Let $n$ be a positive integer, $C_0, C_1, \ldots, C_{n-1} \in M_m(\mathbb{F}_2)$. If the sequence $s_{\infty} = s_0, s_1, \ldots$ over $\mathbb{F}_{2^m}$ satisfies

$$s_{i+n} = -(C_0 s_i + C_1 s_{i+1} + \ldots + C_{n+1}s_{i+n-1}) \text{ for } i = 0, 1, \ldots \quad (3.6)$$

Then this recurrence is called the $\sigma-LFSR$ of order $n$, $s_{\infty}$ is the sequence generated by the $\sigma-LFSR$, and the matrix polynomial $f(x)$ given by

$$f(x) = x^n + C_{n-1}x^{n-1} + \ldots + C_1x + C_0 \in M_m(\mathbb{F}_2)[x]$$

is the $\sigma$-polynomial of the $\sigma-LFSR$ in relation 3.6. It is to be noted that $s_{\infty}$ is periodic if and only if $C_0$ is invertible, that is $C_0 \in GL_m(\mathbb{F}_2)$. If the period of $s_{\infty}$ is $2^{mn} - 1$, then the $\sigma-LFSR$ is known as the primitive $\sigma-LFSR$. Necessary and sufficient condition for primitive $\sigma-LFSR$ to exit is given as follows:

**Definition 3.2.3** (Primitive $\sigma-LFSR$). Let the sequence $s_{\infty} = s_0, s_1, \ldots$ be generated by $\sigma-LFSR$ in equation 3.6 with $\sigma$-polynomial $f(x) = x^n + C_{n-1}x^{n-1} + \ldots + C_1x + C_0 \in M_m(\mathbb{F}_2)[x]$ where $C_0 \in GL_m(\mathbb{F}_2)$ and $C_l = (c_{ij}^{(l)})_{m \times m}$ for $l = 0, 1, \ldots, n - 1$,

$$F(x) = (f^{ij}(x))_{m \times m} \in M_m(\mathbb{F}_2[x])$$

be the corresponding polynomial matrix of $f(x)$ where

$$f^{ij}(x) = (\delta^{ij}x^n) + \sum_{l=0}^{n-1} c_{ij}^{(l)} x^l \in \mathbb{F}_2[x], \delta^{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Then $\sigma-LFSR$ is a primitive if and only if the determinant $|F(x)|$ is a primitive polynomial of degree $mn$ over $\mathbb{F}_2$.

The $m$-sequences produced by maximum period length LFSRs certainly have good statistical properties and are therefore frequently used in the de-
sign of PRBGs for stream ciphers. However, an LFSR by itself is completely unsuitable for cryptographic purposes, because the Berlekamp-Massey algorithm gives an efficient means for determining the feedback connections for any LFSR of length $L$ if any subsequence of the output of length $2L$ is known. This algorithm shows that any PRBG must have large linear complexity as there is always the possibility that an attacker could calculate the shortest LFSR which duplicates the output of a PRBG. The simplest way to get rid of the defects in using an LFSR as a PRBG is to use a nonlinear Boolean function to generate the bitstring. Few methods to achieve this are by using nonlinear combinations of LFSRs, nonlinear feedback shift registers, clock controlled generators, feedback with carry shift registers are discussed in the following sections of this chapter.

**Nonlinear Combination Generators**

An LFSR should never be used by itself as a keystream generator. If the feedback coefficients of an LFSR are public, then the entire keystream can obviously be recovered from the knowledge of any $L$ consecutive bits of the keystream, where $L$ is the linear complexity of the running-key (which does not exceed the LFSR length). Hence using an LFSR as a PRBG is cryptographically very weak because of the Berlekamp-Massey algorithm. If we use the outputs from $n$ LFSRs as the input $n$ bit vector to a nonlinear Boolean function $f(x_1, x_2, \ldots, x_n)$, then the output bit string made up of the values of $f$ might be a good pseudorandom bit generator. This is called a nonlinear combination generator. But a nonlinear combination generator is subject to correlation attacks unless the function $f$ is correlation immune of sufficiently high order. Under some conditions, it is possible to easily compute the linear complexity of the output function $f$ for a nonlinear combination generator, if the linear complexities of the input LFSRs are known. Hence the combining function must have a sufficiently high linear complexity which is obtained by choosing a combining function of high degree. But unfortunately, there is a tradeoff between high degree and high order correlation immunity for $f$.

Rueppel [78] pointed out that introducing even one bit of memory into the nonlinear combination generator allows one to avoid the tradeoff be-
between degree and correlation immunity. The combiner with memory is explained in the following section.

**Combiner with Memory**

A \((k, m)\) combiner with \(k\) inputs and \(m\) memory bits is a finite state machine which is defined by an output function

\[
f : \{0, 1\}^m \times \{0, 1\}^k \rightarrow \{0, 1\}
\]

and a memory function given by

\[
h : \{0, 1\}^m \times \{0, 1\}^k \rightarrow \{0, 1\}^m
\]

For a stream \((X_1, X_2, \ldots)\) of inputs, \(X_t \in \{0, 1\}^k\) and an initial assignment \(M_1 \in \{0, 1\}^m\) of the memory, an output bitstream \((z_1, z_2, \ldots)\) is defined according to \(z_t = f(M_t, X_t)\) and \(M_{t+1} = h(M_t, X_t)\) for all \(t > 0\). Here \(f\) and \(h\) are Boolean functions. For keystream generation, the stream of inputs \((X_1, X_2, \ldots)\) are updated by some devices like LFSRs. The initial states are determined by the secret key and an initialization vector. The stream cipher \(E_0\) used in Bluetooth is an example of combiner with memory, \(k = 4\) inputs and \(m = 4\) bit memory. The stream of inputs is here produced by the outputs of 4 LFSRs of length 128 bits (25, 31, 33, and 39) in total. Few stream ciphers with memory are SNOW, Sosemanuk [1], ZUC etc. A concept related to combiners with memory are feedback with carry shift registers (FCSRs) [79].

This results in the following general framework of synchronous stream ci-
Figure 3.6: Schematic representation of combiner with memory

\[
\begin{align*}
  z_t &= f(X_t, M_t) \\
  c_t &= p_t \oplus z_t \\
  X_{t+1} &= L \cdot X_t \\
  M_{t+1} &= h(X_t, M_t)
\end{align*}
\]

Where \( p_t, z_t, c_t \), are respectively the plaintext, the keystream and ciphertext at time \( t = 0, 1, 2, \ldots \). The initial state \((X_0, M_0)\) is determined by the initialization mechanism, which combines a secret key \( K \) and a known initialization vector \( IV \) with an initialization function say \( f_{\text{init}} \) such that \((X_0, M_0) = f_{\text{init}}(K, IV)\).

**Irregularly Clocked LFSRs in Generators**

In the case of irregularly clocked LFSRs in generators, the bits of the output sequence are not produced each time that the underlying shift registers are clocked, but rather at irregular clock intervals. This provides a new way to introduce nonlinearity in the operation of the generators. The output sequence say \( s(t) \) of LFSR-1 serves to control the generator output sequence say \( z(t) \), of the output sequence of LFSR-2. Basically, LFSR-1 is regularly clocked and its output \( s(t) \) determines the clocking of LFSR-2. A simple example of this is the step-1/step-2 generator of Gollman and Chambers [80], in which LFSR 2 is stepped once at time \( t \) if \( s(t) = 0 \), and is stepped twice if \( s(t) = 1 \).
Nonlinear Filter Generators

Different method of using an LFSR in a PRBG without immediately allowing an attack via the Berlekamp-Massey algorithm involves applying a nonlinear Boolean function $f$ to the outputs at a fixed number of stages in a single LFSR. Since the LFSR outputs are filtered through the function $f$, this method is known as a nonlinear filter generator.

Figure 3.7: Non-linear filter generator

Linear Complexity of the Nonlinear Filter Generator: Suppose a function $f(x_1, x_2, ..., x_L)$ in the nonlinear filter generator has degree $k$, then Edwin Key in 1976 showed that the linear complexity of the output sequence $z(t)$ is at most $L_k = \sum_{j=1}^{k} \binom{L}{j}$. Massey and Serconek in 1994, based on discrete Fourier transform showed that in the special case where $L$ is a prime and $f$ is a product of two distinct variables (say $x_i x_j$, $i < j$), there is always equality in the bound which is defined as $L + \binom{L}{2}$. Also note that the set of all output sequences produced when $f$ varies over all polynomials of degree $k$, the fraction of sequences with linear complexity $L_k$ is at least $e^{-1/L}$.

The basic idea behind correlation attacks and its variants on nonlinear filter generators is to attack the generator by finding good correlations between the generator output and some linear combinations of the inputs to the filter function $f$. That is the goal is to determine how much information a given filter function $f(g(x))$ leaks about its inputs, over all linear functions $g(x)$. These types of attacks are known as affine approximation attacks or conditional correlation attacks. Golic J. D, introduces the inversion attack and the linear model approach which exploits the fact that the generator output, with probability different from one half, satisfies the same linear
recurrence as the LFSR sequence. There is also a possibility of the so-called fast correlation attacks on nonlinear filter generators [81].

**Non-linear Feedback Shift Register (NFSR)**

NFSRs are known to be more resistant to cryptanalytic attacks than Linear Feedback Shift Registers. Non-Linear Feedback Shift Registers are simply the generalization of LFSRs by replacing the linear update function by any non-linear Boolean function. Unlike LFSR, the construction of large NFSRs with guaranteed long periods remains an open problem, other than the generation of a De Bruijn sequence, done by extending a maximal-length LFSR with $n$ stages. Few stream cipher algorithms whose design depends upon the component NFSR are the Grain, Trivium, Achterbahn, Keeloq etc.

**Feedback with Carry Shift Register (FCSR)**

A Feedback with Carry Shift Register (FCSR) can be seen as an alternative to an LFSR. The idea of using FCSRs to generate sequences for cryptographic applications was initially proposed by Klapper and Goresky [79]. FCSR shares several cryptographically important properties with LFSRs and is also easily implementable. F-FCSR-H and F-FCSR [82] are some of the FCSR based stream ciphers developed as part of the eSTREAM project. A Feedback with Carry Shift Register (FCSR) is an automaton which computes the binary expansion of a 2-adic number, where $p$ and $q$ are some integers, with $q$ is odd, $q < 0 < p < |q|$. The size $n$ of the FCSR is such that $n+1$ is the bit length of $|q|$. In the stream cipher construction, $p$ depends on the secret key and the IV, and $q$ is a public parameter. An FCSR can be realized using either a Galois or Fibonacci architecture [83].

**Fibonacci Architecture**

In the FCSR architecture the basic shift register is provided with a small amount of auxiliary memory $m$ which is a non-negative integer. The register is initially loaded with $a_0, a_1, ..., a_{r-1}$. The contents of the tapped cells are added as integers to the current contents of the memory to form an integer sum. The parity bit is fed back into the first cell of the shift register.
while the higher order bits are retained for the new value of the memory.

Figure 3.8: Fibonacci architecture of FCSR

The state consists of two parts, one main register \( y = (a_0, a_1, \ldots, a_{r-2}, a_{r-1}) \) and one memory \( m \). The feedback is given by \((q_1, q_2, \ldots, q_r)\). The output sequence is given by the linear recurrence with carry:

\[
2m_t + a_t = m_{t-1} + \sum_{i=1}^{r} q_ia_{t-i}
\]

which can be solved for \( m_t \) and \( a_t \) since \( a_t \in \{0, 1\} \).

**Galois Architecture**

Here the bits \( q_1, q_2, \ldots, q_r \) \((q_r \neq 0)\) are multipliers. The cells denoted by \( c_1, c_2, \ldots, c_{r-1} \) are carry bits. The \( \sum \) symbol represents the full adder. At the \( j \)-th adder, the following input bits are received.

i) \( a_j \) from the preceding cell

ii) \( a_0q_0 \) from the feedback line and

iii) \( c_j \) from the memory cell

These are added to form a sum. At the next clock cycle, this sum modulo 2 is passed on to the next cell in the register, and the higher order bit is used to replace the memory.

An IV dependent FCSR based stream cipher, which is a modification to BEAN stream cipher is proposed in chapter 4.
Figure 3.9: Galois architecture of FCSR

NIST and Structural Tests

A variety of different statistical tests can be applied to a keystream to evaluate that it is generated by a truly random source. There are various test suites available in the literature. Donald Ervin Knuth [84] presented several empirical tests. The DIEHARD tests [85] are a battery of statistical tests for measuring the quality of a random number generator. They were developed by George Marsaglia and published in the year 1995, which consists of 15 independent statistical randomness tests which includes Birthday Spacings Test, Overlapping 5-Permutation Test, Binary Rank Test, Bitstream Test OPSO, OQSO (Overlapping-Quadruples-Sparse-Occupancy), DNA Test, Count the 1s Test Parking Lot Test, Minimum Distance Test, 3D Spheres Test, Squeeze Test, Overlapping Sums Test, Runs Test, Craps Test.

Crypt-X [86] suite developed in the Information Security Research Centre at Queensland University of Technology includes frequency, binary derivative, change point, runs, sequence complexity and linear complexity tests. NIST [87] Statistical Test Suite consists of 16 tests which includes frequency, block frequency, runs, longest run, matrix rank, spectral, non-overlapping template matchings, overlapping template matchings, universal test, Lempel-Ziv complexity, linear complexity, serial cumulative sums, runs, approximate entropy, random excursions and variants. These statistical tests are designed to evaluate the randomness properties of a finite sequence. For the evaluation of block ciphers (presented for AES) Soto [88] proposed nine different ways to generate large number of data streams from a block cipher and tested these streams using the statistical tests available in NIST test suite. To test the randomness properties of stream ciphers
using these tools, one has to generate a large amount of keystream (> $10^6$ bits) and apply these statistical tests. Failing from these tests do not usually lead to key or internal state recovery, but can be used for distinguishing the keystream from a truly random one. This kind of testing can be considered as a black box approach, since the internal structure of the cipher is not taken into account.

Later in the year 2006, Meltem Sonmez Turan, proposed a statistical analysis method [89] which considers the relationship between key, initial vector, internal state and the keystream. The objective is not to leak any information about the internal state or secret key from the availability of keystream and initial vector. The author proposed the following six statistical tests for analyzing synchronous stream ciphers. Out of which the first four tests considers only the key and IV as inputs and last two test considers the internal structure of the ciphers after the initialization process of the cipher.

**Key/Keystream Correlation Test.** The purpose of this test is to evaluate the correlation between the key and the corresponding bits of keystream using a fixed IV.

**IV/Keystream Correlation test.** Tests the correlation between IV and the corresponding keystream using a fixed key.

**Frame Correlation Test.** Considers the correlation between keystreams using different IV values.

**Diffusion Test.** This test examines the diffusion property of each bit of key and IV on the keystream. To satisfy diffusion, each bit of IV and key should affect the keystream. Minor changes in the IV or key should result in random looking changes in the keystream.

**Internal State Correlation Test.** The purpose of this test is to analyze the effect of similar IVs on the internal state of the cipher.

**Internal State/Keystream Correlation Test.** The main idea of
this test is that at any time, if the internal state has a distinguishing property such as low/high weight, the following keystream part should behave randomly in terms of its weight. These above six tests may be considered as one of the testing criteria while designing secure stream ciphers. For details of these tests for implementation one can refer the paper [89].

3.2.2 Stream Ciphers Derived from Block Ciphers

Given a plaintext of arbitrary length, block ciphers break it down to blocks of the desired length and use padding for the final block. Each block is encrypted separately with the same key, which results in identical ciphertext blocks for identical plaintext blocks. This mode of encryption is known as Electronic Code Book (ECB) mode of operation, and is not recommended in many situations as it does not hide data patterns well. Furthermore, ciphertext blocks are independent from each other, allowing an attacker to substitute, delete or replay blocks unnoticed. To overcome the disadvantages of ECB, NIST specified three other modes of operation known as CBC (cipher block chaining), CFB (cipher feedback mode) and OFB (output feedback mode) modes that use chaining or feedback. The feedback modes in fact turn the block cipher into a stream cipher by using the algorithm as a keystream generator. In CFB, a block cipher is used to build a self-synchronizing stream cipher that provides confidentiality. OFB uses the previous output of the block cipher instead of the previous ciphertext. That makes the ciphertext independent of plaintext and ciphertext, and therefore identifies OFB as a synchronous stream cipher. It is a confidentiality mode that is build very similar to CFB, but differs in properties like error propagation. Examples of stream ciphers derived from block ciphers are LEX [90] and AEGIS [91] which depends on AES.