Lecture Notes on Error-Correcting Codes and their Applications to Symmetric Cryptography

Anne Canteaut

Inria
Anne.Canteaut@inria.fr
https://www.rocq.inria.fr/secret/Anne.Canteaut/

version: December 4, 2017

Contents

Chapter 10

Reed-Muller Codes and Boolean Functions

Reed-Muller codes are one of the oldest families of error-correcting codes. Even if their minimum distance is relatively small, they have been widely used for error correction because of their very simple decoding algorithm. A notable example is the use of the Reed-Muller code $\mathcal{R}(1,5)$ for transmitting the images sent by the Mariner 9 space probe, in Martian orbit since November 1971. Also, Reed-Muller codes are of major mathematical interest because of their connection with Boolean functions and with finite affine and projective geometries (details on this last issue can be found in [Ass92]).

10.1 Boolean functions and their representations

Definition 10.1 (Boolean function)**.** *A Boolean function of n variables is a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2$. Its* value vector *is the binary vector v_f of length 2^n composed of all $f(x)$ when $x \in \mathbb{F}_2^n$.*

10.1.1 Truth table and Algebraic normal form

A Boolean function is usually defined by its *truth table*, which gives the images of all elements in $\mathbb{F}_2^n$. For instance, Table 10.1 is the truth table of a Boolean function of 3 variables. The value vector of f is the vector of $\mathbb{F}_2^8$ corresponding to the last row in the truth table.

Boolean functions are often identified with their value vectors. In particular, the weight and the support of a Boolean function f refer to the weight and the support of its value vector v_f. Most cryptographic applications use *balanced* Boolean functions, i.e., Boolean

x_1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1
$f(x_1, x_2, x_3)$	0	1	0	0	0	1	1	1

Table 10.1: Truth table of a Boolean function of 3 variables.

functions f whose output is uniformly distributed. This equivalently means that the weight of v_f is half of its length.

Besides the truth table, there are several other representations of Boolean functions which may be more appropriate in some contexts. In coding theory and in cryptography, a very natural representation is the so-called *algebraic normal form* (ANF), which corresponds to the expression of a Boolean function as a multivariate polynomial. Since the n inputs of the function take their values in $\mathbb{F}_2$, they must be considered modulo X^2+X. Therefore, this polynomial has degree at most 1 in each input variable. It follows that any monomial of this polynomial is the product of some input variables. Each monomial can then be characterized by a subset of $I=\{1,\dots,n\}$, i.e., $\prod_{i\in I} x_i$, or equivalently by the n-bit vector u having I as support. This second notation will be extensively used in the context of Boolean functions.

Notation 10.2. For any $u\in\mathbb{F}_2^n$, x^u denotes the monomial in $\mathbb{F}_2[x_1,\dots,x_n]/(x_1^2+x_1,\dots,x_n^2+x_n)$ defined by

$$\prod_{i=1}^{n} x_i^{u_i}\ .$$

The following theorem then shows that any Boolean function can be uniquely represented by a multivariate polynomial, and it also gives a simple formula for computing this polynomial from the value vector of the function.

Theorem 10.3 (Algebraic normal form). *Let f be a Boolean function of n variables. Then, there exists a unique multivariate polynomial in $\mathbb{F}_2[x_1,\dots,x_n]/(x_1^2+x_1,\dots,x_n^2+x_n)$ such that*

$$f(x_1,\dots,x_n)=\sum_{u\in\mathbb{F}_2^n} a_u x^u,\ \textit{with } a_u\in\mathbb{F}_2\ .$$

This multivariate polynomial is called the algebraic normal form (ANF) *of f.*

Moreover, the coefficients of the ANF and the values of f satisfy:

$$a_u=\sum_{x\preceq u} f(x)\ \textit{and}\ f(u)=\sum_{x\preceq u} a_x,$$

where the sums are in $\mathbb{F}_2$ and $x\preceq y$ if and only if $x_i\le y_i$ for all $1\le i\le n$.

Proof. We first show by induction on n that the ANF of an n-variable Boolean function can be uniquely computed from its truth table.

- For $n=1$, it is easy to check that the polynomial a_1x+a_0 with $a_1=f(0)+f(1)$ and $a_0=f(0)$ is the unique polynomial equal to f.
- Induction step. Given an n-variable function f, we consider the two $(n-1)$-variable Boolean functions g and h defined by

  $$g(x_1,\dots,x_{n-1})=f(x_1,\dots,x_{n-1},0)\text{ and }h(x_1,\dots,x_{n-1})=f(x_1,\dots,x_{n-1},1)\ .$$

  Then, we have

  $$f(x_1,\dots,x_n)=(1+x_n)g(x_1,\dots,x_{n-1})+x_nh(x_1,\dots,x_{n-1})$$

or equivalently,

$$f(x_1,\dots,x_n) = g(x_1,\dots,x_{n-1}) + x_n\left(g(x_1,\dots,x_{n-1}) + h(x_1,\dots,x_{n-1})\right) .$$

We apply the induction hypothesis and denote by α_u (resp. β_u) for $u \in \mathbb{F}_2^{n-1}$ the coefficients of the ANF of g (resp. of h). We know that

$$\alpha_u = \sum_{x \preceq u} g(x) = \sum_{x \preceq u} f(x,0) \text{ and } \beta_u = \sum_{x \preceq u} h(x) = \sum_{x \preceq u} f(x,1) .$$

We then deduce that

$$\begin{array}{rcl} f(x_1,\dots,x_n) & = & g(x_1,\dots,x_{n-1}) + x_n\left(g(x_1,\dots,x_{n-1}) + h(x_1,\dots,x_{n-1})\right) \\ & = & \displaystyle\sum_{u \in \mathbb{F}_2^{n-1}} \alpha_u \prod_{i=1}^{n-1} x_i^{u_i} + \sum_{u \in \mathbb{F}_2^{n-1}} (\alpha_u + \beta_u) \left(\prod_{i=1}^{n-1} x_i^{u_i}\right) x_n . \end{array}$$

Therefore, the coefficients a_v, $v = (v_1,\dots,v_n) \in \mathbb{F}_2^n$, of the ANF of f are given by

$$a_v = \begin{cases} \alpha_{v_1,\dots,v_{n-1}} & \text{if } v_n = 0 \\ \alpha_{v_1,\dots,v_{n-1}} + \beta_{v_1,\dots,v_{n-1}} & \text{if } v_n = 1 \end{cases}$$

From the expressions of the coefficients α and β, we deduce that

$$a_v = \begin{cases} \sum_{u \preceq (v_1,\dots,v_{n-1})} f(u,0) & \text{if } v_n = 0 \\ \sum_{u \preceq (v_1,\dots,v_{n-1})} f(u,0) + \sum_{u \preceq (v_1,\dots,v_{n-1})} f(u,1) & \text{if } v_n = 1 \end{cases}$$

implying that

$$a_v = \sum_{u \preceq v} f(u) .$$

Conversely, the values of f are uniquely determined by its ANF since the function over $\mathbb{F}_2^{2^n}$ which maps the value vector of f to the vector of coefficients of its ANF is an involution. Indeed, for any $y \in \mathbb{F}_2^n$, we have

$$\begin{array}{rcl} \displaystyle\sum_{u \preceq y} a_u & = & \displaystyle\sum_{u \preceq y} \sum_{x \preceq u} f(x) \\ & = & \displaystyle\sum_{x \preceq y} f(x) |\{u \in \mathbb{F}_2^n : x \preceq u \preceq y\}| \\ & = & \displaystyle\sum_{x \preceq y} 2^{wt(y)-wt(x)} f(x) . \end{array}$$

All terms in this sum are then zero modulo 2 unless $x = y$. Thus

$$\sum_{u \preceq y} a_u = f(y) ,$$

which means that the transformation we consider is an involution. ◇

Example 10.1. Computing the ANF of the function defined in Table 10.1. Using the previous theorem, we compute the coefficients of the ANF of this Boolean function:

$$\begin{aligned}
a_{000} &= f(000) = 0 \\
a_{100} &= f(100) + f(000) = 1 \\
a_{010} &= f(010) + f(000) = 0 \\
a_{110} &= f(110) + f(010) + f(100) + f(000) = 1 \\
a_{001} &= f(001) + f(000) = 0 \\
a_{101} &= f(101) + f(001) + f(100) + f(000) = 0 \\
a_{011} &= f(011) + f(001) + f(010) + f(000) = 1 \\
a_{111} &= \sum_{x \in \mathbf{F}_2^3} f(x) = wt(f) \bmod 2 = 0 \ .
\end{aligned}$$

Thus, the ANF of f is

$$x_1 + x_1x_2 + x_2x_3 \ .$$

The *degree* of f is then the degree of the largest monomial in the ANF of f, i.e.,

$$\deg f = \max_{u \in \mathbb{F}_2^n : a_u \neq 0} wt(u) \ .$$

For instance the function considered in the previous example has degree 2.

It is worth noticing that there exist several other representations of Boolean functions which may be more convenient than the ANF in some other contexts. For instance, the disjunctive normal form represents the function by some products between variables and negations of variables, which are added by an OR. A disjunctive normal form of the function defined in Table 10.1 is

$$x_1\overline{x_2}\overline{x_3} \,\mathrm{OR}\, x_1\overline{x_2}x_3 \,\mathrm{OR}\, \overline{x_1}x_2x_3 \,\mathrm{OR}\, x_1x_2x_3 \ .$$

Such a representation may be more appropriate than the ANF when we want to determine the smallest circuit which implements the function in a context where only AND, NOT and OR gates are available, see [Weg87] for more details.

10.1.2 Computing the Algebraic Normal Form

The general form of the transformation which associates the coefficients of the ANF to the value vector is

$$\begin{array}{rccl}
\mathcal{M}_n : & \mathbb{F}_2^{2^n} & \to & \mathbb{F}_2^{2^n} \\
 & a = (a_u, u \in \mathbb{F}_2^n) & \mapsto & (b_u, u \in \mathbb{F}_2^n) \quad \text{with } b_u = \sum_{v \preceq u} a_v \ .
\end{array}$$

This function is called *the binary Möbius transform.* Indeed, Möbius inversion is a method for inverting sums over a partially ordered sets. This general inversion formula appears in many contexts in combinatorics. For instance, it leads to the principle of inclusion-exclusion and to the expression of Euler ϕ-function [Rot64, Moe12]. In our context, we have proved in Theorem 10.3 that the transformation $\mathcal{M}_n$ is an involution, so any algorithm for computing the binary Möbius transform can be used both for computing the ANF from the value vector and for computing the value vector from the ANF.

The naive method for computing $\mathcal{M}_n(a)$ consists in evaluating the sum of the coordinates a_v of a over all positions $v \preceq u$ for the 2^n successive elements $u \in \mathbb{F}_2^n$. Since the sum defining b_u has $2^{wt(u)}$ terms, the overall complexity of this algorithm is

$$\sum_{i=0}^{n} \binom{n}{i} 2^i = 3^n .$$

But there exists a faster algorithm for computing the image of an element by $\mathcal{M}_n$ which has complexity $n2^{n-1}$ only. This algorithm exploits the fact that if we decompose any vector $a = (a_u, u \in \mathbb{F}_2^n)$ into two halves, namely $L(a) = (a_{u,0}, u \in \mathbb{F}_2^{n-1})$ and $R(a) = (a_{u,1}, u \in \mathbb{F}_2^{n-1})$, we get the following recursive formula:

$$L(\mathcal{M}_n(a)) = \mathcal{M}_{n-1}(L(a)) \text{ and } R(\mathcal{M}_n(a)) = \mathcal{M}_{n-1}(L(a)) + \mathcal{M}_{n-1}(R(a))$$

where the addition denotes the addition in $\mathbb{F}_2^{2^{n-1}}$. The corresponding algorithm then starts from $(a_u, u \in \mathbb{F}_2^n)$ where the values of u are written in lexicographic order. The k-th step, for $1 \leq k \leq n$, then computes the images by $\mathcal{M}_k$ of the 2^{n-k} vectors of 2^k consecutive bits composing a. The result at Step k is then obtained from the result at Step $(k-1)$ by splitting the vector into blocks of 2^k consecutive bits, and for each block, the first half of the block remains unchanged while the second half is replaced by the sum of both halves. This iterative process is described by Algorithm 1. In this algorithm, the vectors $(a_u, u \in \mathbb{F}_2^n)$ are equivalently represented by 2^n-bit arrays $(a[i], 0 \leq i < 2^n)$, where n-bit integers are identified with n-bit vectors.

Algorithm 1 Evaluating the Möbius transform $\mathcal{M}_n$.

```
**Input:** $(a[i], 0 \leq i < 2^n)$
**Output:** $b = \mathcal{M}_n(a)$
**for** $i$ from 0 to $2^n - 1$ **do**
  $b[i] \leftarrow a[i]$
**end for**
**for** $k$ from 1 to $n$ **do**
  **for** $i$ from 0 to $2^{n-k}$ **do**
    // Compute the image of the $i$-th $2^k$-bit block under $\mathcal{M}_k$
    **for** $j$ from 0 to $2^{k-1} - 1$ **do**
      $b[2^k i + 2^{k-1} + j] \leftarrow b[2^k i + j] + b[2^k i + 2^{k-1} + j] \bmod 2$
    **end for**
  **end for**
**end for**
**return** $b$
```

Example 10.2. Computing the ANF of a 3-variable Boolean function. Let us denote by $f[0], \ldots, f[7]$ the array representing the 8-bit value vector of the 3-variable Boolean function f, namely $f[i] = f(i_0, i_1, i_2)$ where $i = \sum_{j=0}^{2} i_j 2^i$. Then, the operations performed during the three successive steps of Algorithm 1 are described in Table 10.2.

Example 10.3. Computing the ANF of a 5-variable Boolean function in C. If the value vector of the function is stored as 32-bit integer `x`, then the corresponding ANF is computed by the following program.

Table 10.2: Computing the ANF of a 3-variable Boolean function f with Algorithm 1.

i	0	1	2	3	4	5	6	7
$f[i]$	$f[0]$	$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$	$f[6]$	$f[7]$
Step 1	$f[0]$	$\boldsymbol{f[0]+f[1]}$	$f[2]$	$\boldsymbol{f[2]+f[3]}$	$f[4]$	$\boldsymbol{f[4]+f[5]}$	$f[6]$	$\boldsymbol{f[6]+f[7]}$
Step 2	$f[0]$	$f[0]+f[1]$	$\boldsymbol{f[0]+f[2]}$	$\boldsymbol{f[0]+f[1]}$ $\boldsymbol{+f[2]+f[3]}$	$f[4]$	$f[4]+f[5]$	$\boldsymbol{f[4]+f[6]}$	$\boldsymbol{f[4]+f[5]}$ $\boldsymbol{+f[6]+f[7]}$
Step 3	$f[0]$	$f[0]+f[1]$	$f[0]+f[2]$	$f[0]+f[1]$ $+f[2]+f[3]$	$\boldsymbol{f[0]+f[4]}$	$\boldsymbol{f[0]+f[1]}$ $\boldsymbol{f[4]+f[5]}$	$\boldsymbol{f[0]+f[2]}$ $\boldsymbol{+f[4]+f[6]}$	$\boldsymbol{f[0]+f[1]}$ $\boldsymbol{+f[2]+f[3]}$ $\boldsymbol{+f[4]+f[5]}$ $\boldsymbol{+f[6]+f[7]}$

```
  x ^= (x & 0x55555555) << 1;
  x ^= (x & 0x33333333) << 2;
  x ^= (x & 0x0f0f0f0f) << 4;
  x ^= (x & 0x00ff00ff) << 8;
  x ^= x << 16;
```

A more general program for any number of variables is given in [Jou09, Page 287] and can be downloaded from `http://www.joux.biz/algcrypt/PROGRAMS/Walsh_9-2.html`.

10.2 Reed-Muller codes

10.2.1 Definition

Reed-Muller codes are named after Reed [Ree54] and Muller [Mul54]: Muller described the codes while Reed proposed a majority-logic decoding algorithm for them. Reed-Muller codes can be defined over $\mathbb{F}_q$ but we here focus on the binary case. Binary Reed-Muller codes can be defined very easily in terms of Boolean functions.

Definition 10.4 (Reed-Muller codes)**.** *Let m be a positive integer and r an integer such $0 \le r \le m$. The* r-th order binary Reed-Muller code of length 2^m*, denoted by $\mathcal{R}(r,m)$, is the set of the value vectors of all Boolean functions of m variables with degree at most r:*

$$\mathcal{R}(r,m) = \{(f(x), x \in \mathbb{F}_2^m), f : \mathbb{F}_2^m \to \mathbb{F}_2 \text{ with } \deg f \le r\} .$$

In particular $\mathcal{R}(0,m)$ is composed of the all-zero and the all-one 2^m-bit words. It is also known as the *repetition code* of length 2^m. On the other extreme, $\mathcal{R}(m,m)$ contains all 2^m-bit words.

It is worth noticing that, exactly as Reed-Solomon codes, Reed-Muller codes can be seen as *evaluation codes*: they are obtained by evaluating multivariate polynomials with coefficients in $\mathbb{F}_2$ while Reed-Solomon codes in characteristic 2 are obtained by evaluating univariate polynomials with coefficients in $\mathbb{F}_{2^m}$.

Reed-Muller codes satisfy the following simple properties.

Proposition 10.5. *Let m be a positive integer and r an integer such $0 \le r \le m$.*

1. *$\mathcal{R}(r,m)$ is a linear code;*
2. *The value vectors of all monomials of degree at most r form a basis of $\mathcal{R}(r,m)$;*
3. *The dimension of $\mathcal{R}(r,m)$ is*
$$\dim \mathcal{R}(r,m) = \sum_{i=0}^{r} \binom{m}{i} ;$$
4. *$\mathcal{R}(r-1,m) \subset \mathcal{R}(r,m)$.*

Example 10.4. Generator matrix of $\mathcal{R}(1,3)$. $\mathcal{R}(1,3)$ is a linear code of length 8 and dimension $1+3=4$. A generator matrix of $\mathcal{R}(1,3)$ consists of the value vectors of all monomials of degree 0 or 1 in x_1, x_2, x_3:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} .$$

Indeed, the rows of this matrix are equal to the value vectors of the functions $f(x_1, x_2, x_3)$ with respective ANF 1, x_1, x_2 and x_3, where the inputs (x_1, x_2, x_3) of the value vectors are ordered lexicographically.

10.2.2 The $(u|u+v)$ construction

This general construction, described by Plotkin [Plo60], combines two codes of the same length to derive a new code which is twice longer. In particular, it provides a recursive construction of the family of Reed-Muller codes. We first describe the general construction and the properties of the resulting code.

Proposition 10.6 (The $(u|u+v)$ construction)**.** *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two (non necessarily linear) binary codes of the same length n, and of respective sizes M_1, M_2 and minimum distances d_1 and d_2. The $(u|u+v)$ construction forms a new code of length $2n$ as follows:*

$$\mathcal{C} = \{(u|u+v), u \in \mathcal{C}_1, v \in \mathcal{C}_2\} .$$

Then $\mathcal{C}$ is a code of size M_1M_2 and minimum distance $d = \min(2d_1, d_2)$.

Proof. $\mathcal{C}$ has size M_1M_2 since it can easily be checked that two distinct pairs (u_1, v_1) and (u_2, v_2) lead to distinct elements c_1 and c_2 in $\mathcal{C}$. Let us now determine their Hamming distance:

$$d(c_1, c_2) = d(u_1, u_2) + d(u_1 + v_1, u_2 + v_2) .$$

If $v_1 = v_2$, $d(c_1, c_2) = 2d(u_1, u_2) \geq 2d_1$. Otherwise, $v_1 \neq v_2$, and we use that, for any two vectors x and y,

$$\begin{aligned} wt(x+y) &= wt(x) + wt(y) - 2|\operatorname{Supp} x \cap \operatorname{Supp} y| \\ &\geq wt(x) + wt(y) - 2wt(x) \\ &\geq wt(y) - wt(x) . \end{aligned}$$

Then,

$$d(c_1, c_2) \geq wt(u_1 + u_2) + wt(v_1 + v_2) - wt(u_1 + u_2) \geq d_2 .$$

Moreover, the lower bound $d \geq \min(2d_1, d_2)$ is tight, since for any $v \in \mathcal{C}_2$, two codewords $u, u' \in \mathcal{C}_1$ at distance d_1 lead to two codewords $c_1 = (u, u+v)$ and $c_2 = (u', u'+v)$ in $\mathcal{C}$ at distance $2d(u, u') = 2d_1$. Similarly, for any $u \in \mathcal{C}_1$, two codewords $v, v' \in \mathcal{C}_2$ at distance d_2 lead to two codewords $c_1 = (u, v)$ and $c_2 = (u, v')$ in $\mathcal{C}$ at distance $d(v, v') = d_2$. $\diamond$

A generator matrix for the code resulting from the $(u|u+v)$ construction is

$$G = \begin{pmatrix} G_1 & G_1 \\ 0 & G_2 \end{pmatrix} ,$$

where G_1 and G_2 are generator matrices for $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively.

Clearly, any Reed-Muller code of length 2^m can be constructed from Reed-Muller codes of length 2^{m-1} as detailed in the following theorem.

Theorem 10.7. *Let m be a positive integer and r an integer such $1 \leq r < m$. Then,*

$$\mathcal{R}(r,m) = \{(u|u+v), u \in \mathcal{R}(r,m-1), v \in \mathcal{R}(r-1,m-1)\} .$$

Proof. For any Boolean function f of m variables and degree at most r, there exist two functions of $m-1$ variables, namely g and h, with $\deg g \leq r$ and $\deg h \leq r-1$ such that

$$f(x_1,\ldots,x_m) = g(x_1,\ldots,x_{m-1}) + x_m h(x_1,\ldots,x_{m-1}) .$$

The value vector v_f of f can then be decomposed into two halves corresponding to the inputs with $x_m = 0$ (resp. with $x_m = 1$). Then, we get

$$v_f = (v_g|v_g + v_h) ,$$

where v_g and v_h denote the value vectors of g and h and belong to $\mathcal{R}(r,m-1)$ and $R(r-1,m-1)$ respectively. ◇

Note that we can check that the dimensions of $\mathcal{R}(r,m)$ satisfy the relation given in Proposition 10.6:

$$\begin{aligned}
\dim\mathcal{R}(r,m-1) + \dim\mathcal{R}(r-1,m-1) &= \sum_{i=0}^{r}\binom{m-1}{i} + \sum_{i=0}^{r-1}\binom{m-1}{i} \\
&= 1 + \sum_{i=1}^{r}\left(\binom{m-1}{i} + \binom{m-1}{i-1}\right) \\
&= 1 + \sum_{i=1}^{r}\binom{m}{i} = \dim\mathcal{R}(r,m) .
\end{aligned}$$

10.3 Weight distributions of Reed-Muller codes

10.3.1 Minimum distance of $\mathcal{R}(r,m)$

The minimum distance of $\mathcal{R}(r,m)$ can immediately be deduced from the $(u|u+v)$ construction.

Theorem 10.8. *Let m be a positive integer and r an integer such $0 \leq r \leq m$. Then, $\mathcal{R}(r,m)$ has minimum distance 2^{m-r}. Moreover, the value vectors of the monomials of degree r are minimum-weight codewords of $\mathcal{R}(r,m)$.*

Proof. Clearly, the value vector of any monomial of m variables and degree r has weight 2^{m-r}. Then, we only have to prove that $d_{\min}\mathcal{R}(r,m) \geq 2^{m-r}$, by induction on m.

- For $m = 1$, $\mathcal{R}(0,1)$ consists of the value vectors of the constant functions, namely (00) and (11), implying that $d_{\min}\mathcal{R}(0,1) = 2$. Similarly, $\mathcal{R}(1,1)$ is composed of all four elements of $\mathbb{F}_2^2$, implying that $d_{\min}\mathcal{R}(1,1) = 1$.

- Induction step. By combining Theorem 10.7 with Proposition 10.6, we obtain that

$$\begin{aligned}
d_{\min}\mathcal{R}(r,m) &= \min\left(2d_{\min}\mathcal{R}(r,m-1), d_{\min}\mathcal{R}(r-1,m-1)\right) \\
&= \min\left(2^{m-r}, 2^{m-r}\right) = 2^{m-r} .
\end{aligned}$$

◇

Example 10.5. Minimum distance of $\mathcal{R}(1,5)$. The Reed-Muller code $\mathcal{R}(1,5)$ has been used for correcting errors during the transmission of the pictures taken by the Mariner 9 space probe from Martian orbit. An appropriate trade-off between the error-correction capability and the information rate was needed for addressing the bad quality of the communications and the power constraints on board. Each pixel in these images was represented by one out of 64 grayscale values. This 6-bit pixel was then encoded into a 32-bit value by $\mathcal{R}(1,5)$, which is a code of length 32 and dimension 6. The minimum distance of this code is $2^{5-1} = 16$, implying that up to 7 transmission errors can be corrected. More than 7000 pictures have been transmitted by Mariner 9 with this procedure. Details on the decoding algorithms used for space communications can be found in [McE04].

The complete weight distribution of $\mathcal{R}(r,m)$ is known for a few values of r only.

10.3.2 Weight distribution of $\mathcal{R}(1,m)$

Proposition 10.9. *The first-order Reed-Muller code $\mathcal{R}(1,m)$ is composed of the all-zero word, of the all-one word, and of $(2^{m+1}-2)$ words of weight 2^{m-1}.*

Proof. By induction on m.

- For $m = 1$, $\mathcal{R}(1,m)$ is composed of four vectors: (00), (11), (01) and (10).
- Induction step. From Theorem 10.7, we know that $\mathcal{R}(1,m)$ can be decomposed into

$$\mathcal{R}(1,m) = \{(u|u), u \in R(1,m-1)\} \cup \{(u|\bar{u}), u \in R(1,m-1)\}$$

  where $\bar{u}$ denotes the bitwise complement of u. Since $R(1,m-1)$ contains 1 word of weight 0, 1 word of weight 2^{m-1} and (2^m-2) words of weight 2^{m-2}, we deduce that the first set in the decomposition contains 1 word of weight 0, 1 word of weight 2^m and (2^m-2) words of weight 2^{m-1}. The second set is composed of 2^m words of weight 2^{m-1}. The result then follows.

◇

Since the non-constant words in $\mathcal{R}(1,m)$ are the value vectors of all affine functions, we deduce that any m-variable Boolean function of degree 1 has weight 2^{m-1}. This can also be observed from the fact that, for $f(x) = a \cdot x + \varepsilon$ with $a \in \mathbb{F}_2^m$ and $\varepsilon \in \mathbb{F}_2$,

$$\{x \in \mathbb{F}_2^m : f(x) = 1\} = \{x \in \mathbb{F}_2^m : a \cdot x = 1 + \varepsilon\} = \begin{cases} \langle a \rangle^\perp & \text{if } \varepsilon = 1 \\ \mathbb{F}_2^m \setminus \langle a \rangle^\perp & \text{if } \varepsilon = 0 \end{cases}$$

The support of an affine function is then a hyperplane or the complement of a hyperplane, and has size 2^{m-1}.

10.3.3 Weight distribution of $\mathcal{R}(m-1,m)$

Proposition 10.10. *The Reed-Muller code $\mathcal{R}(m-1,m)$ is the code with parameters $[2^m, 2^m-1, 2]$ composed of all 2^m-bit words of even weight.*

Proof. We first observe that $\mathcal{R}(m-1,m)$ contains half of the 2^m-bit words since its dimension is $\sum_{i=0}^{m-1}\binom{m}{i} = 2^m-1$. Then, we only need to prove that any codeword in $\mathcal{R}(m-1,m)$ has an even weight. This comes directly from Theorem 10.3 which shows that the coefficient of degree m in the ANF of an m-variable Boolean function f is equal to the parity of the weight of its value vector. $\diamond$

An equivalent formulation of this result, in terms of Boolean functions, is as follows.

Corollary 10.11. *The value vector of a Boolean function f of n variables has an odd weight if and only if f has degree n.*

This implies that Boolean functions with maximal degree cannot be used in most cryptographic applications since their output distribution is biased.

10.3.4 Weight distribution of $\mathcal{R}(2,m)$

Besides $\mathcal{R}(0,m)$, $\mathcal{R}(1,m)$, $\mathcal{R}(m-1,m)$ and $\mathcal{R}(m,m)$ the complete weight distribution of $\mathcal{R}(2,m)$ is also known [SB70]. The proof can be found in [SB70] or in Chapter 15,§ 2 of [MS77].

Proposition 10.12. *The weights of the codewords of the second-order Reed-Muller code of length 2^m are of the form*

$$w = 2^{m-1} \text{ or } w = 2^{m-1} \pm 2^{m-1-h} \text{ with } 0 \le h \le \left\lfloor \frac{m}{2} \right\rfloor .$$

Moreover, the corresponding weight distribution $A_0, \ldots, A_{2^m}$ is given by

$$\begin{aligned} A_0 &= A_{2^m} = 1 \\ A_{2^{m-1}\pm 2^{m-1-h}} &= 2^{h(h+1)} \times \prod_{i=1}^{h} \frac{(2^{m-2i+2}-1)(2^{m-2i+1}-1)}{(2^{2i}-1)} \\ A_{2^{m-1}} &= 2^{1+m+\binom{m}{2}} - 2\sum_{w=0}^{2^{m-1}-1} A_w . \end{aligned}$$

10.3.5 Duality

We have seen that $\mathcal{R}(m-1,m)$ consists of all even-weight words, i.e., $\mathcal{R}(m-1,m) = \langle \underline{1} \rangle^{\perp}$ where $\underline{1}$ denotes the all-one vector. This means that $\mathcal{R}(m-1,m)$ is the dual of $\mathcal{R}(0,m)$. Actually, this relationship is more general as shown by the following theorem.

Theorem 10.13. *Let m be a positive integer and r an integer such $0 \le r < m$. Then,*

$$\mathcal{R}(r,m)^{\perp} = \mathcal{R}(m-r-1,m) .$$

Proof. Let us consider two codewords $x \in \mathcal{R}(r,m)$ and $y \in \mathcal{R}(m-r-1,m)$. Then, their scalar product is

$$x \cdot y = \sum_{i=1}^{2^m} x_i y_i \bmod 2 = wt\left((x_i y_i)_{1 \leq i \leq 2^m}\right) \bmod 2 .$$

This last vector $(x_i y_i)_{1 \leq i \leq 2^m}$ is the value vector of the function $h = fg$ where f and g are the Boolean functions corresponding to x and y respectively. By definition, $\deg f \leq r$ and $\deg g \leq (m-r-1)$. It follows that h has degree at most $(m-1)$. From Proposition 10.10, we deduce that its value vector has an even weight. Therefore, $x \cdot y \equiv 0$, i.e., $\mathcal{R}(m-r-1,m) \subseteq \mathcal{R}(r,m)^\perp$. Equality between both codes is then deduced from their dimensions. $\diamond$

In particular, for odd m, the Reed-Muller code $\mathcal{R}(\frac{m-1}{2}, m)$ is a self-dual code with parameters $[2^m, 2^{m-1}, 2^{\frac{m+1}{2}}]$.

The weight distribution of a code can be computed from the weight distribution of its dual by MacWilliams identity. It follows that the weight distributions of $\mathcal{R}(r,m)$ for $r \leq 2$ and $r \geq m-3$ are known.

Example 10.6. Weight distributions of all Reed-Muller codes of length 32.

- $\mathcal{R}(0,5)$ is the repetition code of length 32 and dimension 1. Its weight distribution is

$$A_0 = 1 \text{ and } A_{32} = 1 .$$

- From Proposition 10.9, the weight distribution of $\mathcal{R}(1,5)$ is

$$A_0 = A_{32} = 1 \text{ and } A_{16} = 62.$$

- From Proposition 10.12, the weight distribution of $\mathcal{R}(2,5)$ is

$$A_0 = A_{32} = 1,\ A_8 = A_{24} = 620,\ A_{12} = A_{20} = 13888 \text{ and } A_{16} = 36518 .$$

  We can check that this code is a self-dual code by applying the MacWilliams transformation to its weight enumerator

$$W_{\mathcal{C}}(X,Y) = X^{32} + 620X^{24}Y^8 + 13888X^{20}Y^{12} + 36518X^{16}Y^{16} + 13888X^{12}Y^{20} + 620X^8Y^{24} + Y^{32} .$$

  We get

$$2^{-16} W_{\mathcal{C}}(X+Y, X-Y) = W_{\mathcal{C}}(X,Y) .$$

- The weight distribution of $\mathcal{R}(3,5)$ can be deduced from the weight distribution of its dual, $\mathcal{C}^\perp = R(1,5)$. We have

$$W_{\mathcal{C}^\perp} = X^{32} + 62X^{16}Y^{16} + Y^{32} .$$

  Then,

$$\begin{aligned} W_{\mathcal{C}}(X,Y) &= 2^{-6} W_{\mathcal{C}^\perp}(X+Y, X-Y) \\ &= X^{32} + 1240X^{28}Y^4 + 27776X^{26}Y^6 + 330460X^{24}Y^8 + 2011776X^{22}Y^{10} \\ &\quad +7063784X^{20}Y^{12} + 14721280X^{18}Y^{14} + 18796230X^{16}X^{16} + 14721280X^{14}Y^{18} \\ &\quad +7063784X^{12}Y^{20} + 2011776X^{10}Y^{22} + 330460X^8Y^{24} + 27776X^6Y^{26} \\ &\quad +1240X^4Y^{28} + Y^{32} . \end{aligned}$$

- From Proposition 10.10, $\mathcal{R}(4,5)$ is the set of all 32-bit words of even weight.

- $\mathcal{R}(5,5)$ is equal to $\mathbb{F}_2^{32}$.

10.3.6 Other properties of the weights of $\mathcal{R}(r, m)$

Determining the weight distribution of $\mathcal{R}(r, m)$ when $3 \leq r \leq m - 4$ is an open problem. Some partial information is known, including the minimum distance (Theorem 10.8) and the number of low-weight codewords, i.e., the number of codewords of weight w for $d_{\min} \leq w \leq 2.5 d_{\min}$ [KT70, KTA76].

Another information is that the weight of any codeword in $\mathcal{R}(r, m)$ is divisible by some power of 2 whose exponent depends on r and m.

Proposition 10.14. *Let m be a positive integer and r an integer such $0 < r \leq m$. Then, the weights of all codewords in $\mathcal{R}(r, m)$ are divisible by*

$$2^{\lceil \frac{m}{r} \rceil - 1} .$$

This theorem was originally proved by Solomon and McEliece [SM66], but it is usually presented as a consequence of a more general theorem due to McEliece [McE72] on the divisibility of the weight of cyclic codes. A simpler proof can be derived from a classical formula for computing the weight of a Boolean function from its ANF (see e.g. [MM94] or [CHLL97, Page 240]).

This chapter does not present any decoding algorithms for Reed-Muller codes. However, some of them will be described later in the chapters devoted to cryptographic applications. Indeed, decoding a given vector of length 2^m with respect to $\mathcal{R}(r, m)$ is equivalent to finding the best approximation of a given Boolean function by a function of degree r. This problem, for small r, appears in several cryptanalytic techniques including linear cryptanalysis.

Bibliography

[Ass92] Edward F. Assmus Jr. On the Reed-Muller codes. *Discrete Mathematics*, 106-107:25–33, 1992.

[CHLL97] Gérard D. Cohen, Iiro S. Honkala, Simon Litsyn, and Antoine Lobstein. *Covering codes*. North-Holland, 1997.

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, 2009.

[KT70] Tadao Kasami and Nobuki Tokura. On the weight structure of Reed-Muller codes. *IEEE Transactions on Information Theory*, 16(6):752–759, 1970.

[KTA76] Tadao Kasami, Nobuki Tokura, and Saburo Azumi. On the Weight Enumeration of Weights Less than $2.5d$ of Reed-Muller Codes. *Information and Control*, 30(4):380–395, 1976.

[McE72] Robert J. McEliece. Weight congruence for p-ary cyclic codes. *Discrete Mathematics*, 3:177–192, 1972.

[McE04] Robert J. McEliece. The 2004 Shannon lecture. `http://www.systems.caltech.edu/EE/Faculty/rjm/papers/ShannonLecture.pdf`, 2004.

[MM94] Oscar Moreno and Carlos J. Moreno. The MacWilliams-Sloane conjecture on the tightness of the Carlitz-Uchiyama bound and the weights of duals of BCH codes. *IEEE Transactions on Information Theory*, 40(6):1894–1907, 1994.

[Moe12] Möbius inversion. Encyclopedia of Mathematics, 2012. `http://www.encyclopediaofmath.org/index.php?title=M%C3%B6bius_inversion&oldid=23416`.

[MS77] F. Jessie MacWilliams and Neil J.A. Sloane. *The theory of error-correcting codes.* North-Holland, 1977.

[Mul54] David E. Muller. Application of Boolean algebra to switching circuit design and to error detection. *IEEE Transactions on Computers*, 3:6–12, 1954.

[Plo60] Morris Plotkin. Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, 6(4):445–450, 1960.

[Ree54] Irving S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *IEEE Transactions on Information Theory*, 4:38–49, 1954.

[Rot64] Gian-Carlo Rota. On the foundations of combinatorial theory I. Theory of Möbius Functions. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 2(4):340–368, 1964.

[SB70] Neil J. A. Sloane and Elwyn R. Berlekamp. Weight enumerator for second-order Reed-Muller codes. *IEEE Transactions on Information Theory*, 16(6):745–751, 1970.

[SM66] Gustave Solomon and Robert J. McEliece. Weights of cyclic codes. *Journal of Combinatorial Theory*, 1(4), 1966.

[Weg87] Ingo Wegener. *The complexity of Boolean functions.* Wiley-Teubner, 1987.

Chapter 11

Stream Cipher Basics

Symmetric cryptography is widely used since it offers much better performance than public-key encryption. This comparison holds when considering both software-oriented and hardware-oriented platforms. For software implementations on a usual processor, classical asymmetric encryption requires several dozens (or even several hundreds) of kilocycles of the processor for encrypting/decrypting one byte. The symmetric encryption standard, the AES, takes between 20 and 30 cycles for the same operation. More precise figures on several platforms are available on the website of eBACS, a benchmarking suite of cryptographic systems [eba]. All of them show that symmetric encryption is several orders of magnitude faster than asymmetric encryption. Among all possible symmetric ciphers, many dedicated stream ciphers are even faster than common block ciphers. For instance, the throughputs of stream ciphers like Snow 2.0 or Salsa20 correspond to around 5 cycles for encrypting one byte.

The same situation holds when considering the hardware performance. There are many ways to quantify hardware performance depending on the targeted platform: minimal number of gates in a circuit implementing the cipher, power consumption, latency... But for any of these quantities, symmetric encryption outperforms asymmetric encryption. And some stream ciphers have been designed for achieving extremely good performance in constrained environments like on embedded systems.

11.1 Basic principle

11.1.1 Synchronous additive stream ciphers

Stream ciphers are encryption schemes: they include an encryption function and a decryption function which can handle messages of an arbitrary length. In this sense, they cannot be compared with block ciphers, which handle fixed-length inputs only. While formally defining block ciphers is very easy, defining stream ciphers in general is not. For instance, block ciphers with some particular modes of operation (like the CTR mode) are stream ciphers (see Section 11.4.3 for details).

Here, we focus on *synchronous additive* stream ciphers. The encryption function in a synchronous additive stream cipher consists in adding bitwise to the plaintext a binary sequence of the same length, named the *keystream* (or the *running-key*). The keystream, denoted by $\boldsymbol{s} = (s_t)_{t \geq 0}$ is generated independently from the plaintext and from the ciphertext. It should be noted that there exists another family of stream ciphers, called *self-synchronizing stream ciphers*, which is not captured by this definition. The (almost single) practical example is the

CFB mode of operation applied to any block cipher. Self-synchronizing stream ciphers are mainly dedicated to contexts where latency is of primary importance. Therefore, the ciphertext is sent in a very long stream, implying that decryption must often be resynchronized. However, it appears that these ciphers are no longer used today in applications. Instead, synchronous stream ciphers are used: the message is sent as a succession of packets, and any lost packet is resent. For instance in the GSM standard, messages are split in 114-bit frames, corresponding to 4.6 milliseconds. The keystream must then depend on the frame number, which corresponds to an additional parameter named the initial value.

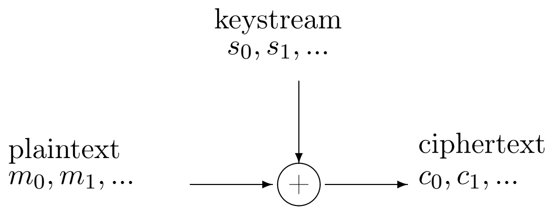


Figure 11.1: Additive synchronous stream cipher.

The oldest and most prominent synchronous stream cipher is obviously the *Vernam cipher* (aka. one-time-pad) [Ver26]. In the Vernam cipher, each keystream bit s_t is chosen at random, independently from the other ones. The Vernam cipher is then a *perfect cipher*, i.e., an unconditionally secure cipher as defined by Shannon [Sha48]. This means that the knowledge of the ciphertext provides absolutely no information about the plaintext. Indeed, if M denotes the n-bit plaintext, C the ciphertext and S the keystream, we have

$$\Pr[C = c | M = m] = \Pr[S = m + c] = 2^{-n}$$

and since this holds for any value m, we deduce that

$$\Pr[C = c] = \sum_m \Pr[C = c|M = m]\Pr[M = m] = 2^{-n} \sum_m \Pr[M = m] = 2^{-n} \ .$$

This implies that the Vernam cipher is perfect. Indeed,

$$\begin{aligned} \Pr[M = m|C = c] &= \frac{\Pr[M = m \text{ and } C = c]}{\Pr[C = c]} \\ &= \frac{\Pr[C = c|M = m]\Pr[M = m]}{\Pr[C = c]} \\ &= \frac{2^{-n}\Pr[M = m]}{2^{-n}} = \Pr[M = m] \ . \end{aligned}$$

In other words, the Vernam cipher is unbreakable. The Vernam cipher is then optimal in terms both of security and of performance. But a major issue is that it requires the use of a secret quantity (the keystream) which has the same length as the plaintext to be encrypted. This is actually a necessary condition for any perfect cipher [Sha49]. Exchanging such a secret in a secure way is of course unpractical in the vast majority of applications, and perfect ciphers have only been used in some high-level communications like diplomatic cables or the most sensitive governmental communications (e.g., in the Moscow-Washington hotline [Kah67, Pages 715-716] or for Soviet diplomatic communications). Obviously, the security of the cipher completely collapses if the keystream is not random. For instance, the fact that some keystream portions have been used twice has been exploited by the United States Army Signal Intelligence Service (a forerunner of the NSA) to decrypt some Soviet communications during the so-called Venona project [Cro95].

11.1.2 Pseudo-random generators

Synchronous stream ciphers can be seen as practical variants of the Vernam cipher. The key idea is that the keystream $\boldsymbol{s}$ is not a random sequence anymore, but a *pseudo-random sequence* derived from a shorter secret quantity which can be exchanged much more easily. The process which maps a short quantity to an arbitrary-length sequence $\boldsymbol{s}$ is called a *cryptographic pseudo-random generator*. Informally, it is a finite-state automaton which produces in a deterministic way a long sequence $\boldsymbol{s}$ from a (short) seed such that, for an adversary who knows everything except the seed, it is impossible to distinguish $\boldsymbol{s}$ from a truly random sequence with a significantly lower complexity than an exhaustive search for the seed.

This notion must be distinguished from the widely-used notion of *random generator*, which also corresponds to a process which generates a random-looking sequence. Such random generators are used for generating cryptographic keys for instance. But, they differ from the previous notion in the sense that they are not deterministic. In other words, the generated sequence is not reproducible. Random generators include some techniques based on physical methods exploiting various entropy sources like radioactive decay or thermal noise. Software-oriented random generators may also use as an entropy source some physical events available to the operating system, mainly the precise timing of interrupts (time between user keystrokes, task-scheduling, network hits, disk-head seek times...)[a].

Deterministic pseudo-random generators include the `random()` or `rand()` function which can be found in any high-level programming language: in a program, n consecutive calls to `random()` produce n random numbers, but all executions of the program generate the same sequence unless the seed of the generator has been modified by some initialization function (usually called `srandom()` or `srand()`). Such pseudo-random generators aim at producing sequences with good statistical properties which can be used in simulations (e.g. for the Monte-Carlo method). However, the output sequence can often be distinguished from a truly random sequence by anyone who knows the specifications of the generator. This is clearly the case for most implementations of `random()`.

A pseudo-random generator then produces at each time instant t an m-bit digit[b] s_t which is determined by the value of its internal state x_t. It can then be defined by the following three building-blocks (see Figure 11.2).

- *an initialization function*, which determines the initial state of the generator, x_0, from the secret key and a public initial value, IV (IV usually corresponds to a frame number). This initialization may be split into two different steps: the first one, named *key-loading*, computes some intermediate quantity which depends on the key only (and not on the IV); the second one, named *IV injection* or *resynchronization*, computes the initial state x_0 from the previously computed intermediate value and from the IV. Decomposing the

[a] `/dev/random` is a special file under Unix-like environments which serves as pseudo-random generators. A more sophisticated pseudo-random generator is Havege `http://www.irisa.fr/caps/projects/hipsor/`[SS03].

[b] Many stream ciphers use binary pseudo-random generators (i.e., $m = 1$), but some generators rather produce longer digits, like bytes or words, depending on the implementation target.

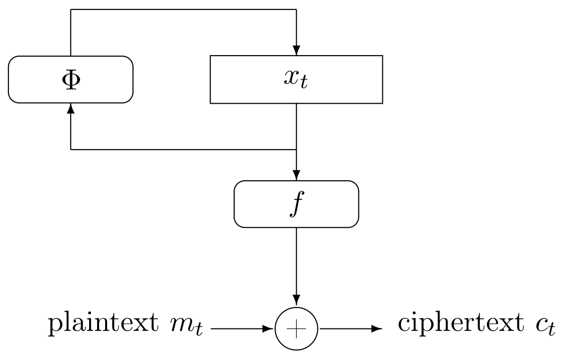


Figure 11.2: Pseudo-random generator for an additive stream cipher.

initialization like this allows a less expensive procedure for changing the IV without changing the key. Indeed, in most applications, the IV is changed much more often than the key, especially in communication protocols using small packets. For instance, in the GSM standard, the IV is modified every 228 bits while the key remains the same during the whole conversation.

- *a transition (or next-state) function*, denoted by Φ, which modifies the internal state between time t and time $(t+1)$. This function may depend on the key, on the IV, and may vary with time. But it is fixed in many hardware-oriented generators, for some obvious implementation reasons.

- *a filtering function*, denoted by f, which at each time instant, extracts the m-bit output digit s_t from the internal state x_t. Exactly as the transition function, the filtering function may depend on the key, on the IV and vary with time, but it is usually fixed for hardware-oriented ciphers.

11.1.3 General functionalities of stream ciphers and usage

Synchronous stream ciphers present some interesting functionalities.

- Encryption or decryption can start as soon as a single bit of the message has been received. This is not the case of a block cipher used with CBC mode for instance, where a whole plaintext block must be received before encryption starts. For this reason, stream ciphers have a low-latency, and they do not require any message buffering.

- Synchronous stream ciphers do not require any padding, i.e., the addition of some bits in order to get a message whose length is a multiple of the block size. This is suitable for instance in low-bandwidth communications, or in protocols which require the transmission of many small packets (in this case, padding may represent a non-negligible part of the transmitted data).

- Decryption does not propagate transmission errors. Indeed, if one bit of the ciphertext has been corrupted during the transmission, it will affect a single bit of the resulting plaintext. This situation is very different from other encryption schemes, like a block cipher in CBC mode, where a single corrupted bit affects the decryption of a whole plaintext block.

For all these reasons, stream ciphers are well-adapted to noisy or low-bandwidth communications. Block-cipher-based stream ciphers, such as the AES with CTR mode, can be used as a first choice in many applications. However, in some particular applications, dedicated stream ciphers must be privileged, for instance if a high encryption/decryption throughput or a very low latency is needed, or in resource-constrained environments like embedded systems (where the size of the circuit, the power consumption... must be minimized).

11.2 Models of attacks

The security of additive stream ciphers is often evaluated by its resistance to *known-plaintext attacks*. Indeed, the knowledge of N bits of plaintext and of the corresponding ciphertext bits is equivalent to the knowledge of N keystream bits. In this context, it is clear that stronger attack models, like chosen-plaintext attacks do not provide any additional information to the attacker. Ciphertext-only attacks can also be considered in a very few cases where some statistical properties of the plaintext source are exploited. But, in the following, we always assume that N keystream bits are known by the attacker. From this knowledge, she might have different goals, corresponding to the following four different types of attacks.

- *key-recovery attacks* aim at recovering the secret key from the knowledge of the keystream bits;

- *initial-state-recovery attacks* aim at recovering the whole initial state of the generator (recovering any complete internal state of the generator is often enough for determining the initial state). Obviously, the knowledge of the secret key is sufficient for recovering the initial state, but the converse does not hold in general. If the transition and filtering functions are public, these attacks enable the adversary to compute as many keystream bits as she wants from this initial state, but they do not enable her to generate any keystream sequence produced from the same key and a different IV. Therefore, they usually have a much more limited impact than key-recovery attacks, especially in protocols handling short packets since the packets are encrypted with different IVs within a session.

- *next-bit-prediction attacks* consist in predicting the value of the next bit from the knowledge of N consecutive keystream bits.

- *distinguishing attacks* aim at determining whether a given N-bit sequence has been produced by the pseudo-random generator from some secret key, or whether it is a truly random sequence.

Distinguishing attacks are obviously much less powerful than all the other mentioned classes. They can nevertheless provide the adversary with some information on the plaintext. For instance, they can be exploited for checking whether an eavesdropped ciphertext corresponds to a given plaintext or not. Such attacks may then be a threat in a context when only a few plaintexts are possible. However, it can be shown that the existence of distinguishing attacks with polynomial complexity is equivalent to the existence of next-bit-prediction attacks [Yao82, BM84]. For this reason, a pseudo-random generator is usually considered secure if it resists all distinguishing attacks having complexity less than the complexity of an exhaustive search for the secret key.

All previously mentioned attacks exploit the knowledge of N keystream bits generated from the same key. However, these keystream bits may be produced from the same IV, or from different IVs. In this second scenario, the IV may be controlled by the attacker (chosen-IV attack) or not. The chosen-IV model is quite powerful but provides a realistic model for the contexts where the IV is randomly chosen (see e.g. [BG07]).

11.3 Generic attacks on stream ciphers

Every building-block in a pseudo-random generator must be chosen with care. In particular, some major criteria can be deduced from the existence of *generic attacks*, i.e., attacks which apply to any generator with similar parameters. As we will see, these attacks impose some general conditions on the building-blocks of the generator.

11.3.1 Period of the sequence of internal states

Since the number of possible internal states of the generator is finite, the sequence of the consecutive internal states $(x_t)_{t\geq 0}$ produced from any given initial state x_0 is an ultimately periodic sequence, i.e., there exists some $t_0 \geq 0$ such that $(x_t)_{t\geq t_0}$ is periodic. Obviously, the corresponding period is upper-bounded by the number of possible internal states. An important requirement is then that the period of the sequence of internal states $(x_t)_{t\geq 0}$ must be large, for any possible initialization. Otherwise, the corresponding keystream sequence would have a small period, implying that the knowledge of a few keystream bits would be sufficient for predicting the whole sequence. In other words, for any initial state x_0, the sequence of all consecutive internal states $(\Phi^t(x_0))_{t\geq 0}$ must have a high period. In our context, high period means that the period must always be higher than the maximal length allowed for a keystream produced from a single key/IV pair.

The smallest period of $(\Phi^t(x_0))_{t\geq 0}$ can be derived from some parameters of the so-called functional graph of the transition function Φ.

Definition 11.1 (Functional graph of a function)**.** *Let F be a function from a finite set $\mathcal{E}$ to itself. The* functional graph *of F is the graph whose vertices are the elements of $\mathcal{E}$ and whose directed edges are the pairs $(x, F(x))$ for all $x \in \mathcal{E}$.*

An example of a functional graph is depicted on Figure 11.3.

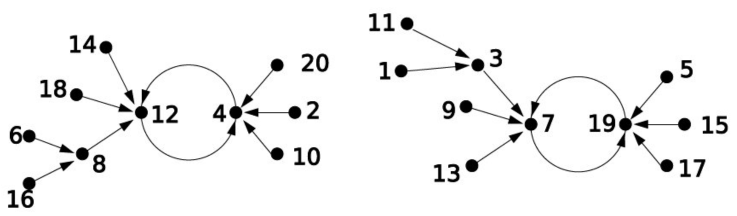


Figure 11.3: Functional graph of the function F defined from $\{1,\dots,20\}$ to itself by $F(x) = \big((x-1)^2 + 2 \bmod 20\big) + 1$ (from [Röc09, Page 72]).

Non-bijective transition functions.

The main parameters of the functional graph of a random function have been investigated in [FO90].

Proposition 11.2. *[FO90][FS09, Pages 462-463] The main parameters of the functional graph of a random function from a set of size N to itself have the following asymptotic forms as N tends to infinity:*

- *expected number of connected components in the graph $\sim \frac{1}{2}\ln N$*
- *expected size of the largest component $\sim dN$ with $d \simeq 0.75782$*
- *expected number of points on a cycle $\sim \sqrt{\frac{\pi N}{2}}$*
- *expected length of the longest cycle $\sim \kappa\sqrt{N}$ with $\kappa = 0.78248$.*

In particular, the functional graph of a random function has a small number of connected components. Around 76 % of all points in the graph are grouped together in the same component, called the *giant component*. This giant component has a large cycle, of size close to $\sqrt{N}$. As an illustration, Figure 11.4 extracted from [QD88] depicts a part of the functional graph of a function from $\mathbb{F}_2^{56}$ to itself derived from the DES block cipher: $K \mapsto \mathsf{DES}_K(0)$ where the 64-bit output of DES is truncated to get a 56-bit value. This functional graph has a giant cycle. The size of this giant cycle, estimated from the scale of the picture, is approximately $5 \times 10^8 \simeq 2^{29}$, which is close to the square root of 2^{56}.

It is worth noticing that these statistics are extensively used for estimating the complexity of several algorithms searching for collisions in the functional graph of a function, including Floyd's cycle-finding algorithm, Pollard's rho-method... (see Chapter 7 in [Jou09] for details on these algorithms).

In the context of a pseudo-random generator with transition function Φ, we are interested in the distribution of the period of the sequence $(\Phi^t(x_0))_{t\geq 0}$ when x_0 varies, which corresponds to the sizes of the cycles in the functional graph of Φ. For a random function Φ, the average values of the pre-period and period of the sequence $(\Phi^t(x_0))_{t\geq 0}$ are as follows.

Proposition 11.3. *Let Φ be a random function from a set of size N to itself. Then, the expectations of the period and of the pre-period of the sequence $(\Phi^t(x_0))_{t\geq 0}$ are both asymptotic to $\sqrt{\frac{\pi N}{8}}$ as N tends to infinity.*

For any pseudo-random generator with an n-bit internal state and a random transition function Φ, this means that, for most initial states x_0, the sequence of internal states produced after roughly $2^{\frac{n}{2}}$ clocks has period close to $2^{\frac{n}{2}}$. Then, after a relatively small number of clocks, all internal states belong to a set of size roughly $2^{\frac{n}{2}}$. In other words, there is a severe entropy loss since the cost of an exhaustive search for the internal state decreases from 2^n to $2^{\frac{n}{2}}$. This weakness can also be exploited in more sophisticated attacks like time-memory-data trade-offs attacks (see Section 11.3.2). This entropy loss comes from the non-bijectivity of the function. Choosing a non-bijective function as a transition function then requires a careful analysis. In particular, it implies that the size of the internal state should be large enough for avoiding these attacks, and this may affect the performance of the cipher. Several stream cipher proposals use a non-bijective transition function, including MICKEY [BD08, BD06],

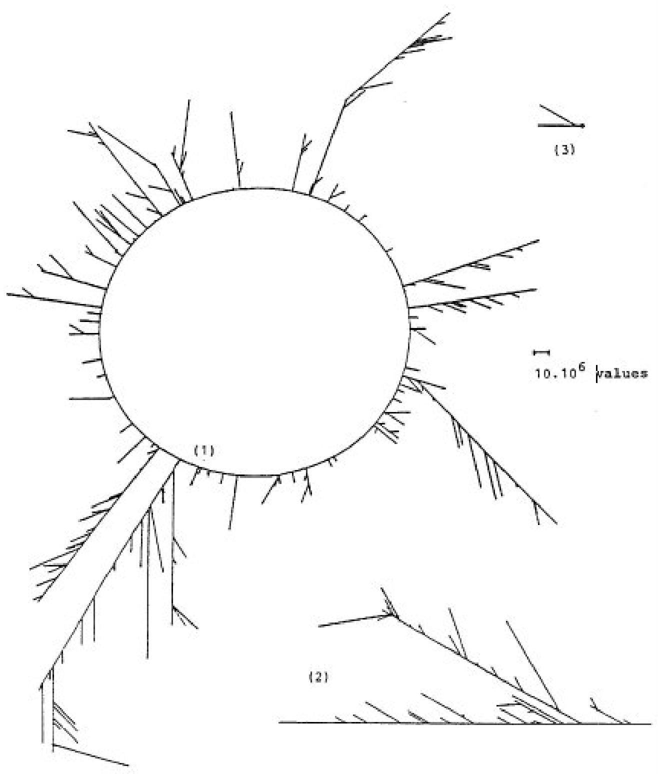


Figure 11.4: Partial functional graph of a function from $\mathbb{F}_2^{56}$ to itself derived from the DES block cipher [QD88].

recommended in the eSTREAM portfolio [ECR05], or the F-FCSR cipher [ABL08]. A precise evaluation of the entropy of the internal state after r iterations of these generators can be found in [HK05, Röc08b, Röc08a].

Bijective transition functions.

In order to avoid any state entropy loss, a permutation can serve as a transition function. In this case, all connected components in the functional graph of the permutation are cycles since a given vertex has exactly one incoming edge. As an illustration, Figure 11.5 extracted from [FS09] depicts the functional graphs of six random permutations of a set of size 500.

The expectations of the main parameters of the functional graph of a random permutation are as follows.

Proposition 11.4. *[FS09, Page 175] For any $r \leq N$, the expectation of the number of cycles of length r in the functional graph of a random permutation of a set of size N is $\frac{1}{r}$.*

The total number of cycles is the graph is H_N where $H_N = \sum_{i=1}^{N} \frac{1}{i}$ is the harmonic number which can be approximated by $H_N \sim \ln N + \gamma$ where $\gamma \simeq 0.57721$ is the so-called Euler's constant.

This implies that the functional graph of a randomly chosen permutation contains a few

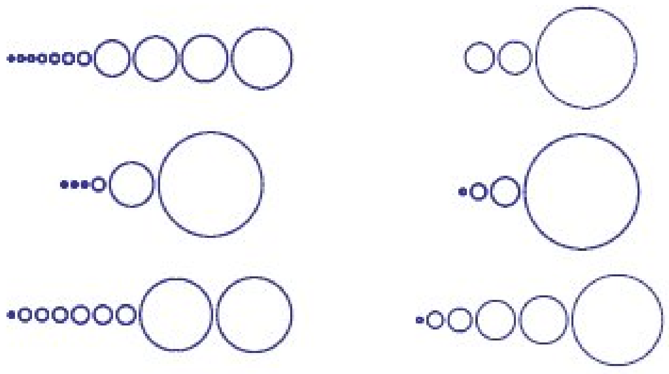


Figure 11.5: Functional graphs of six randomly chosen permutations of a set of size 500, extracted from [FS09, Page 176].

cycles only: some short cycles which include a very small proportion of points, and one or a few long cycles. It can be proved for instance that the probability that a random permutation has a cycle of length greater than or equal to $N/2$ is close to $\ln 2 \simeq 0.69$ [FS09, Page 176]. These results imply that the use of a random transition *permutation* avoids an important entropy loss. However, the existence of short cycles should be taken into account since they correspond to initial states producing a keystream with a short period. Either these unsuitable initial states can be explicitly determined and removed from the set of possible initial states (this corresponds to the situation of LFSR-based transition functions, as we will see in Chapter 12), or the internal state is sufficiently large so that the proportion of unsuitable initial states can be considered as negligible.

It appears from this analysis that the choice of the transition function in a pseudo-random generator can follow one of the two alternative design strategies:

- Choose for Φ a permutation with some known mathematical properties operating on a relatively small internal state: the minimal period of $(\Phi^t(x_0))_{t\geq 0}$ can then be proved to be close to 2^n where n is the size of the internal state. Short cycles are avoided either because the proportion of unsuitable initial states is negligible, or because these initial states cannot be obtained by the initialization process. LFSR-based generators follow this first approach.

- Choose a random-looking function or permutation operating on a relatively large internal state: the period of $(\Phi^t(x_0))_{t\geq 0}$ is then expected to be close to $2^{\frac{n}{2}}$ for most initial states x_0. Short cycles may exist but are unlikely to occur. Table-driven generators like RC4 follow this second approach (see Section 11.4.4).

11.3.2 Time-Memory-Data Trade-off attacks

Principle. The basic principle of time-memory trade-off (TMTO) attacks has been introduced by Hellman in 1980 [Hel80] for analyzing block ciphers, and then improved in the context of some stream ciphers by Babbage [Bab95] and Golic [Gol97]. These techniques are generic methods for inverting a one-way function, i.e., a function F from $\mathcal{E}$ into $\mathcal{F}$ which is such that computing $F(x)$ from x is easy, but finding a preimage x of an element $y \in \mathcal{F}$ is difficult. For finding some $x \in \mathcal{E}$ such that $F(x) = y$, one of the following two trivial algorithms can be used.

- *Exhaustive search*: it consists in computing the images under F of all elements $x \in \mathcal{E}$ until one such that $F(x) = y$ has been found. This algorithm is very time-consuming and must be repeated for each new challenge y.

- *Codebook attack*: it consists in precomputing the images of all $x \in \mathcal{E}$ under F and in storing in memory all pairs $(x, F(x))$ indexed by the values of $F(x)$. For each new challenge y, a simple search in this look-up table is enough for finding an x with $F(x) = y$. This algorithm also requires the evaluation of F on all elements in $\mathcal{E}$, but this computation is independent from the challenge. This is then a precomputation step and it has to be done once for all. The on-line step of the attack, which depends on the challenge, is extremely fast: its complexity is logarithmic in the size of the precomputed table (by dichotomic search), or even constant if an appropriate data structure is used. In this sense, this algorithm compares very favorably with the exhaustive search. However, its main drawback is the memory requirement, since all pairs $(x, F(x))$ need to be stored.

Time-memory trade-off attacks then offer better trade-offs between the previous two methods. They consist of two steps: a precomputation step building a table, and an on-line step which uses the table for finding a preimage of the challenge faster than the exhaustive search. The key issue in this type of algorithms is to find a suitable trade-off between the size of the table (memory complexity) and the time complexity of the on-line phase.

Algorithm 2 provides a method for finding a preimage of a given challenge y for some function F from $\mathcal{E}$ to itself.

Algorithm 2 Basic time-memory trade-off algorithm for inverting $F : \mathcal{E} \to \mathcal{E}$; k is a parameter of the algorithm.

```
/* Precomputation phase */
Randomly choose $M$ elements $x_1, \dots, x_M$ in $\mathcal{E}$
for $i$ from 1 to $M$ do
  $z_i \leftarrow F^k(x_i)$.
  Store $(x_i, z_i)$ in a table $T$, sorted by the $z_i$
end for

/* On-line phase */
Input: $y \in \mathcal{E}$.
for $j$ from 0 to $(k-1)$ do
  $y_j \leftarrow F^j(y)$
  Search in the table whether there exists some $i$ such that $z_i = y_j$
  if $(x_i, y_j)$ is in the table then
    if $F^{k-j}(x_i) = y$ then
      return $F^{k-j-1}(x_i)$.
    end if
  end if
end for
```

It can be checked that the value $x = F^{k-j-1}(x_i)$ returned by Algorithm 2 is a preimage of y. Indeed, by construction, $y_j = F^k(x_i) = F^j(y)$, implying that $F^j(y) = F^j\left[F^{k-j}(x_i)\right]$. Then, if this collision on two images under F^j comes from a collision on the corresponding preimages, i.e. $F^{k-j}(x_i) = y$, we get that $F(x) = y$. However, when F is not a permutation, a collision on the images of F^j does not necessarily imply a collision on the corresponding inputs. Such situations correspond to false alarms, which are discarded by testing whether $F^{k-j}(x_i) = y$.

A preimage can be found by this technique if y belongs to the set $\bigcup_{i=1}^{M}\{F^j(x_i), 0 \le j < k\}$. We need then to be sure that this union of sets covers a large portion of $\mathcal{E}$. However, the number of distinct elements covered by these sets is not equal to Mk in general. Indeed, if F behaves like a random function, then the sequence $(F^j(x_i))_{j\ge1}$ is expected to cycle after $j \simeq \sqrt{|\mathcal{E}|}$. This implies that k should not be too large if we want all $F^j(x_i)$ to be distinct when j varies.

A second issue is that two sequences starting from different x_i can merge at some point. In particular, if the first ν sets $\mathcal{X}(x_i) = \{F^j(x_i), 0 \le j < k\}$ are all composed of distinct points, then any new set $\mathcal{X}(x)$ will collide with the previous ones as soon as $\nu k^2 \ge N$. Then, there is no advantage of increasing parameters M and k beyond $Mk^2 = N$. The problem is that, with this condition, only a small proportion of elements of $\mathcal{E}$ is covered by the precomputed table, as shown by the following proposition. In other words, the success probability of the algorithm remains rather small.

Proposition 11.5. *[Hel80] Let F be modeled as a random function from a set of size N to itself. If $Mk^2 = N$ with both M and k large, then the probability of success of Algorithm 2 is close to* $0.8Mk/N$.

Proof. We first compute a lower bound on the size of the union of all sets

$$X_i = \{F^j(x_i), 0 \le j < k\}, \text{ for } 1 \le i \le M \,.$$

For a given pair (i_0, j_0), the probability that $F^{j_0}(x_{i_0})$ is new is lower-bounded by the probability that all $F^j(x_{i_0})$ for $0 \le j \le j_0$ are new, i.e., by

$$\frac{N-A}{N} \times \frac{N-A-1}{N} \times \ldots \times \frac{N-A-j_0}{N}$$

where A denotes the size of $\cup_{i=1}^{i_0-1} X_i$. Obviously, $A \le (i_0 - 1)k$ which implies that

$$\Pr[F^{j_0}(x_{i_0}) \text{ is new}] \ge \left(\frac{N - i_0 k}{N}\right)^{j_0+1} \,.$$

The success probability p_S of the algorithm is equal to the ratio between the expected size of $\cup_{i=1}^{M} X_i$ and N. Then, we get that

$$p_S \ge \frac{1}{N}\sum_{i=1}^{M}\sum_{j=0}^{k-1}\left(1 - \frac{ik}{N}\right)^{j+1} \,.$$

We now use the identity $(1-x)\sum_{j=0}^{k-1} x^j = (1-x^k)$, applied to $x = 1 - ik/N$. We obtain

$$\begin{aligned} p_S &\geq \frac{1}{k}\sum_{i=1}^{M}\frac{1}{i}\left(1-\frac{ik}{N}\right)\left[1-\left(1-\frac{ik}{N}\right)^k\right] \\ &\approx \frac{1}{k}\sum_{i=1}^{M}\frac{1}{i}\left[1-\left(1-\frac{ik}{N}\right)^k\right] \\ &\approx \frac{1}{k}\sum_{i=1}^{M}\frac{1}{i}\left(1-\exp\left(-\frac{ik^2}{N}\right)\right) \end{aligned}$$

where the last approximation comes from

$$k\ln\left(1-\frac{ik}{N}\right) \approx -\frac{ik^2}{N}$$

when $ik/N \ll 1$. By approximating this sum by an integral, we eventually get

$$\begin{aligned} \frac{1}{k}\sum_{i=1}^{M}\frac{1}{i}\left(1-\exp\left(-\frac{ik^2}{N}\right)\right) &= \frac{1}{k}\sum_{i=1}^{M}\frac{k^2}{N}\times\frac{\left(1-\exp\left(-\frac{ik^2}{N}\right)\right)}{\frac{ik^2}{N}} \\ &\simeq \frac{1}{k}\int_0^{\frac{Mk^2}{N}}\frac{1-e^{-x}}{x}dx \\ &= \frac{Mk}{N}H\left(\frac{Mk^2}{N}\right), \end{aligned}$$

where H is the function $H(u) = \frac{1}{u}\int_0^u \frac{1-e^{-x}}{x}dx$. When $Mk^2/N = 1$, then $H(1) \simeq 0.8$. $\diamond$

For obtaining a better coverage of $\mathcal{E}$ by the set X, an attempt could be to increase M or k. However, it appears that when $u = Mk^2/N$ is greater than 1, the value of $H(u)$ decreases very fast. In other words, increasing either M or k beyond $Mk^2 = N$ would lead to points which are already covered by the table.

The solution proposed by Hellman [Hel80] then consists in constructing several independent tables. Each of these tables is obtained by running Algorithm 2 on a slightly modified version of F, namely $h_\ell \circ F$, where h_ℓ is a random permutation of $\mathcal{E}$ whose role is to "diversify" the computations. This leads to Algorithm 3. Now, for each ℓ, we search for some pair (i, j) such that

$$(h_\ell \circ F)^j(h_\ell(y)) = (h_\ell \circ F)^k(x_i)$$

and we hope that this corresponds to a collision

$$h_\ell(y) = (h_\ell \circ F)^{k-j}(x_i)\ ,$$

or equivalently since h_ℓ is a permutation

$$y = F\left[(h_\ell \circ F)^{k-j-1}(x_i)\right]\ .$$

When considering L independent tables corresponding to L independent permutations h_ℓ, $1 \leq \ell \leq L$, we get that, since $Mk/N \ll 1$, the probability that y belongs to the sets X

Algorithm 3 Hellman's time-memory trade-off algorithm for inverting $F : \mathcal{E} \rightarrow \mathcal{E}$; k is a parameter of the algorithm.

```
/* Precomputation phase */
for $\ell$ from 1 to $L$ do
  Randomly choose $M$ elements $x_1, \dots, x_M$ in $\mathcal{E}$
  for $i$ from 1 to $M$ do
    $z_i \leftarrow (h_\ell \circ F)^k(x_i)$.
    Store $(x_i, z_i)$ in a table $T_\ell$, sorted by the $z_i$
  end for
end for

/* On-line phase */
Input: $y \in \mathcal{E}$.
for $\ell$ from 1 to $L$ do
  for $j$ from 0 to $(k-1)$ do
    $y_j \leftarrow (h_\ell \circ F)^j(h_\ell(y))$
    Search in Table $T_\ell$ whether there exists some $i$ such that $z_i = y_j$
    if $(x_i, y_j)$ is in the table then
      if $(h_\ell \circ F)^{k-j}(x_i) = h_\ell(y)$ then
        return $(h_\ell \circ F)^{k-j-1}(x_i)$.
      end if
    end if
  end for
end for
```

generated from each single h_ℓ is small and close to $0.8Mk/N$. Then, for L such independent tables, the overall success probability is

$$1 - \left(1 - 0.8\frac{Mk}{N}\right)^L \approx 1 - \exp\left(\frac{0.8MkL}{N}\right) .$$

Now, choosing $MkL \simeq N$ leads to a success probability of $1 - e^{-0.8} = 0.55$.

The time complexity of the on-line phase and the total memory cost of Hellman's algorithm are then given by

$$\mathsf{Memory} = LM \text{ and } \mathsf{Time} = Lk \text{ where } MkL \simeq N \text{ and } Mk^2 = N .$$

The trade-off between the time and memory complexities is then described by the curve

$$\mathsf{Time} \times \mathsf{Memory}^2 = N^2 .$$

Indeed, we have

$$\mathsf{Time} \times \mathsf{Memory}^2 = L^3M^2k = \frac{N^3M^2k}{M^3k^3} = N^2 \times \frac{N}{Mk^2} = N^2 .$$

An interesting choice for parameters M, k and L satisfying this condition is

$$M = k = L = N^{1/3} ,$$

which leads to the following overall complexity

$$\mathsf{Memory} = LM = N^{2/3} \text{ and } \mathsf{Time} = Lk = N^{2/3} .$$

The precomputation time equals $LMk = N$, which corresponds to the time complexity of the exhaustive search.

When the function that we need to invert is a function from $\mathcal{E}$ to $\mathcal{F}$, then a similar algorithm applies, but F needs to be adapted so that it can be iterated several times. For instance, if $|\mathcal{E}| < |\mathcal{F}|$, then F is composed with a simple "reduction" function whose role is to truncate $F(x)$. Similarly, when $|\mathcal{E}| > |\mathcal{F}|$, several copies of $F(x)$ can be concatenated to reach the required size. Several improvements of this algorithm have been proposed, based on the notion of *distinguished points* or of *rainbow tables*.

TMTO algorithms can be used for instance for recovering the secret key K of a block cipher from the knowledge of the ciphertext corresponding to a fixed known plaintext, typically $c = E_K(0)$. This kind of method was used for storing the users' passwords on Unix-like systems in a secure way: the password serves as a secret key, and the value $E_K(0)$ is stored on the server. When the user wants to log in to the system, $E_K(0)$ is computed from K and compared to the stored value. Obviously, the security of this technique relies on the fact that recovering K from $E_K(0)$ is difficult when the block cipher resists known-plaintext attacks. However, when the underlying block cipher is the former standard DES, the key-size is 56 bits only. Then, an attack using the previous TMTO algorithm has memory and time complexities equal to 2^{37} which is feasible. It requires a massive precomputation, but this may be accomplished by some collaborative effort. For this reason, storing passwords with this method would not be secure. Another drawback is that using twice the same password (typically on two different servers) would be trivially detected with this method. Instead, some public random data named *salt* is introduced in the computation in order to avoid this type of attacks [MT79]. See e.g. Section 10.2 in [MvOV97] for more details on UNIX passwords.

Case of stream ciphers. The previously discussed algorithm may be adapted to the "one-out-of-many" context, i.e., several challenges y are given to the attacker, but she only needs to find a preimage for a single value of y. The number of simultaneous challenges is then another parameter which usually allows a better trade-off. This situation occurs in state-recovery attacks against some stream ciphers, as pointed out by Babbage and Golic [Bab95, Gol97]. This attack applies when both the transition function and the filtering function of the generator are publicly known, i.e., when they do not dependent on the secret key. The objective is then to recover the initial state of the generator. This can be done by inverting the one-way function F which associates to the n-bit initial state the first n bits of the keystream. But if more than n consecutive keystream bits are known, the attacker can consider several frames of n consecutive bits together, corresponding to several challenges. More precisely, D consecutive keystream bits $s_0 s_1 \dots s_{D-1}$ provide $(D - n + 1)$ challenges $y_t = s_t \dots s_{t+n-1}$. A preimage of any y_t then corresponds to the internal state of the generator at time t. Since the first D bits of the keystream are known, recovering a whole internal state of the generator then enables the attacker to compute all bits generated from the corresponding internal state. Moreover, in most generators, the public transition function can be inverted for computing the initial state. The corresponding algorithm is then named *Time/Memory/Data trade-off*. The simplest version is described by Algorithm 4.

Algorithm 4 Basic time/memory/data trade-off algorithm against stream ciphers [Bab95, Gol97].

```
Input: $s_0, \dots, s_{D+L-1}$, $(D + L)$ consecutive keystream bits
/* Precomputation phase */
Randomly choose $M$ elements $x_1, \dots, x_M$ in $\mathbb{F}_2^n$.
for $i$ from 1 to $M$ do
  $z_i \leftarrow F(x_i)$.
  Store $(x_i, z_i)$ in a table $T$, sorted by the $z_i$
end for

/* On-line phase */
for $t$ from 1 to $D$ do
  $y_t \leftarrow s_t \dots s_{t+n-1}$
  Search in the table whether there exists some $i$ such that $z_i = y_t$
  if $(x_i, y_t)$ is in the table then
    $x_i$ is the internal state of the generator at time $t$
  end if
end for
```

From the birthday paradox, we know that if the size of the precomputed table and the number of challenges satisfy

$$MD = N$$

where $N = 2^n$ is the number of possible internal states, then a collision between the two sets $\{z_i, 1 \le i \le M\}$ and $\{y_t, 1 \le t \le D\}$ is likely to occur. For instance, for $M = D = 2^{n/2}$, the probability of a collision is $1 - e^{-1} \simeq 0.63$ [NS90, Page 18]. This coverage condition then enables us to express the memory and time complexities of the two phases as a function of D

and n:

$$\mathsf{Memory} = M, \quad \mathsf{Time} = D \text{ and } \mathsf{Precomputation} = M = \frac{2^n}{D}.$$

This leads to the following relation between memory and time complexities:

$$\mathsf{Memory} \times \mathsf{Time} = N \ .$$

A particularly interesting point on this curve is given by $M = D = 2^{n/2}$. It corresponds to an algorithm with complexity

$$\mathsf{Memory} = \mathsf{Time} = \mathsf{Data} = \mathsf{Precomputation} = 2^{\frac{n}{2}}.$$

A major consequence of this trade-off is that there exists a generic attack against any pseudo-random generator whose filtering and transition functions do not dependent on the key, which recovers the n-bit internal state of the generator with overall complexity $2^{n/2}$. Since such a generator should resist any attack having complexity less than an exhaustive search for the secret key, we deduce that *the internal state of any such generator must be at least twice longer than the secret key.*

Biryukov-Shamir's improvement. As pointed out by Biryukov and Shamir [BS00], the previous attack can even be further improved by adapting the original Hellman's attack to the case where the attacker has to solve one out of many challenges. This leads to Algorithm 5.

The complexity analysis of Hellman's attack similarly applies to the case with D challenges. As shown before, we need to choose

$$Mk^2 = N$$

and each table then covers roughly Mk states. The number of tables L must then be chosen so that the union of all tables contains at least one of the D challenges, i.e,

$$LMk = \frac{N}{D} \ .$$

The complexities of Algorithm 5 are now

$$\mathsf{Memory} = LM, \quad \mathsf{Time} = LkD \text{ and } \mathsf{Precomputation} = LkM = \frac{N}{D}$$

leading to the following trade-off

$$\mathsf{Time} \times \mathsf{Memory}^2 \times \mathsf{Data}^2 = N^2 \ .$$

Indeed,

$$\mathsf{Time} \times \mathsf{Memory}^2 \times \mathsf{Data}^2 = \frac{N^3 D^2}{Mk^2 D^2} = N^2 \ .$$

This offers more trade-offs than the previous attacks, for instance we can take into account the fact that practical attacks usually need to have data complexity smaller than the time complexity. This constraint can now be achieved by choosing for instance $\mathsf{Time} = \mathsf{Memory} = 2^{n/2}$, leading to $\mathsf{Data} = 2^{n/4}$ and $\mathsf{Precomputation} = 2^{3n/4}$. This trade-off could not be achieved with Algorithm 4 since $\mathsf{Time} = \mathsf{Memory} = 2^{n/2}$ can only be obtained when $\mathsf{Data} = 2^{n/2}$.

Algorithm 5 Improved Time/Memory/Data trade-off algorithm against stream ciphers [BS00].

```
**Input:** $s_0, \dots, s_{D+L-1}$, $(D+L)$ consecutive keystream bits
/* **Precomputation phase** */
**for** $\ell$ from 1 to $L$ **do**
  Randomly choose $M$ elements $x_1, \dots, x_M$ in $\mathcal{E}$
  **for** $i$ from 1 to $M$ **do**
    $z_i \leftarrow (h_\ell \circ F)^k(x_i)$.
    Store $(x_i, z_i)$ in a table $T_\ell$, sorted by the $z_i$
  **end for**
**end for**

/* **On-line phase** */
**for** $t$ from 1 to $D$ **do**
  $y \leftarrow s_t \dots s_{t+n-1}$
  **for** $\ell$ from 1 to $L$ **do**
    **for** $j$ from 0 to $(k-1)$ **do**
      $y_j \leftarrow (h_\ell \circ F)^j(h_\ell(y))$
      Search in Table $T_\ell$ whether there exists some $i$ such that $z_i = y_j$
      **if** $(x_i, y_j)$ is in the table **then**
        **if** $(h_\ell \circ F)^{k-j}(x_i) = h_\ell(y)$ **then**
          $(h_\ell \circ F)^{k-j-1}(x_i)$ is the internal state of the generator at time $t$
        **end if**
      **end if**
    **end for**
  **end for**
**end for**
```

11.3.3 Statistical tests

Since a stream cipher shall resist distinguishing attacks, the underlying pseudo-random generator shall have good statistical properties. In particular, several classical statistical tests have been defined which need to be satisfied by any pseudo-random generator. Passing all these statistical tests is then a necessary security condition, but it is obviously not sufficient.

The first statistical properties required for a pseudo-random generator have been defined by Knuth [Knu69] and Golomb [Gol82]. Since these works, enhanced families of tests have been proposed. Usually, they include some *normality tests*, which determine the probability that some property of the considered sequence is satisfied by a random sequence, and *compression tests* which evaluate whether the considered sequence can be significantly compressed.

Normality tests. These tests are statistical hypothesis tests. The aim is to tell apart two hypotheses: the null hypothesis $\mathcal{H}_0$, which corresponds to the truly random case, and the alternative hypothesis $\mathcal{H}_1$:

- $\mathcal{H}_0$: $\mathbf{s}$ is a random sequence;
- $\mathcal{H}_1$: $\mathbf{s}$ has been produced by the target pseudo-random generator.

Some errors may occur when deciding between these two hypotheses. We distinguish the following two types of errors, usually referred to as *false alarms* and *non-detection*.

true situation	conclusion	
	accept $\mathcal{H}_0$	accept $\mathcal{H}_1$
random	no error	type-I error (false alarm)
produced by the generator	type-II error (non-detection)	no error

In normality tests, we usually consider some property satisfied by the known sequence, and we evaluate the probability that the same property is satisfied by a random sequence. We then decide between the two hypotheses by choosing a relevant value for the false alarm probability, typically between 10^{-2} and 10^{-3}.

The simplest normality test is the *frequency test.* For a binary sequence $(s_t)_{0\leq t<N}$ of length n, with n_0 zeroes and n_1 ones, we consider the following quantity

$$x = \frac{(n_0 - n_1)}{\sqrt{N}} .$$

This quantity corresponds, up to a factor $\sqrt{N}$, to the correlation

$$\sum_{t=0}^{N-1} (-1)^{s_t} .$$

When $\mathbf{s}$ is a random sequence, each $(-1)^{s_t}$ is a random variable uniformly distributed in $\{-1, 1\}$. Then, from the central limit theorem, we deduce that, as N gets larger, the distribution of x approximates the normal distribution with mean 0 and variance 1, i.e.,

$$\Pr[a \leq x \leq b] \simeq \frac{1}{\sqrt{2\pi}} \int_a^b e^{-\frac{u^2}{2}} du .$$

We then choose some decision threshold T and select Hypothesis $\mathcal{H}_1$ if and only if $|x| > T$. The corresponding false alarm probability is then given by

$$\alpha = 2\mathrm{Pr}[x > T] = 2 \times \frac{1}{\sqrt{2\pi}} \int_T^{\infty} e^{-\frac{u^2}{2}} du = \mathsf{erfc}\left(\frac{T}{\sqrt{2}}\right)$$

where erfc is the complementary error function:

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-u^2} du \ .$$

For instance, a binary sequence of length $N = 100$ with 58 ones will be accepted as random since a random sequence satisfies $|x| \geq 1.6$ with probability $\mathrm{erfc}(1.6/\sqrt{2}) = 0.109$.

Similarly, we can test some other properties, including

- the frequency of the values within m-bit blocs;
- the number of runs;
- the longest run of ones within m-bit blocks...

Compression tests. These statistical tests determine whether a given sequence can be significantly compressed, since a significant compression is not feasible for a truly random sequence. The most prominent compression tests include Maurer universal statistical test, Lempel-Ziv test, the linear complexity test...

Batteries of statistical tests. The most prominent statistical test suite is the NIST test suite, detailed in the NIST special publication [Nat10]. A very complete description of all involved tests and an implementation is available on `http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html`.

Some other batteries of tests can be found, for instance the DIEHARD test suite designed by George Marsaglia: `http://stat.fsu.edu/pub/diehard/`.

11.4 The main families of stream ciphers

Pseudo-random generators can be split into several families, depending on their application targets, especially on the resources required for their implementation.

11.4.1 Information-theoretically generators

The aim of these generators is to achieve the same security as the one-time-pad under the hypothesis that the adversary with unlimited computational power has some additional constraints, for instance her storage capacity is limited, or she has a limited network access [Mau92, AR99, Rab05]. All known proposals in this category require a very heavy infrastructure and can only be deployed on a large scale. For instance, they may need a shared source of randomness broadcasted by some satellites, or through the Internet on a huge number of web pages. For this reason, their practical use can be envisaged in the very long term only. But these are the only family of generators which achieve probable security.

11.4.2 Generators based on a difficult mathematical problem

The security of these generators relies on the hardness of some well-known mathematical problems. In other words, it can be proved that distinguishing the generated sequence for a random sequence with a significant advantage implies the existence of an algorithm for solving some difficult problem. But, exactly as most public-key cryptosystems, these generators are mainly based on some problems coming from number theory. An example is the RSA generator, whose transition function is the RSA encryption function:

$$x_{t+1} = x_t^e \bmod pq \; .$$

The output of the generator at time t is then the lowest-significant bit of x_t [ACGS88]. Another example is the *Blum-Blum-Shub generator* [BBS86] whose transition function corresponds to squaring modulo pq. The security of this generator then relies of the hardness of the quadratic residuosity problem. Because they all involve some computations with large numbers, all these generators are extremely slow. They have a limited throughput and their implementation cost is too high for being used in practical stream ciphers.

11.4.3 Generators based on block ciphers

These generators produce a pseudo-random sequence with a block cipher (e.g., the AES) used with some particular mode of operation. The most prominent mode of operation which builds a pseudo-random generator from a block cipher is the *counter mode (CTR)*.

The CTR mode, standardized by the NIST in 2001 [Nat01], is depicted on Figure 11.6. The internal state of the generator corresponds to an n-bit counter where n is the block size of the underlying block cipher. The filtering function depends on the secret key and corresponds to the encryption function of the block cipher under key K. The transition function Φ is an incrementing function, which does not depend on the key, and which guarantees that, for any initial state x_0, all successive internal states are distinct. This transition function is not explicitly defined in the NIST recommendation. The user is free to choose it. The only condition is that all elements in the sequences of counters must be distinct. The most widely used transition function is the incrementing function over integers modulo 2^n. An alternative, which usually has a lower hardware implementation cost (since it does not require any carry), consists in choosing for Φ the transition function of an LFSR with primitive feedback polynomial. In this second case, the sequence of all internal states has period $(2^n - 1)$ provided that the initial state differs from the all-zero word. For any of these transition functions, when the underlying block cipher is ideal, the produced sequence is indistinguishable from a random sequence if and only if fewer than $2^{n/2}$ keystream blocks are known [BDJR97] (see also [Rog11, Chapter 5]). Indeed, a keystream sequence of more than $2^{n/2}$ blocks can be easily distinguished from a random sequence since all constituent blocks are distinct: they correspond to the images of distinct internal states by a permutation. A truly random sequence composed of more than $2^{n/2}$ n-bit blocks is expected to contain a collision.

The initial states shall be chosen so that all values of the internal state for a given key are distinct. In particular, if the adversary can control the initial state by choosing an appropriate IV, the CTR mode is vulnerable to IV-chosen attacks. This is the case if the initial state is derived from the IV by an initialization function which is independent from the secret key. Indeed, the adversary can choose two IVs leading to two initial states x_0 and x'_0 with $x'_0 = x_0 + 1$ for instance.

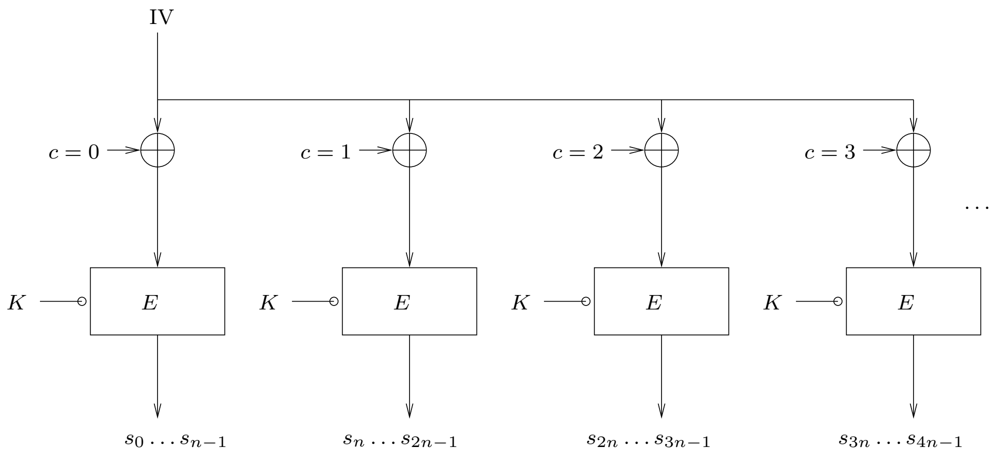


Figure 11.6: Counter mode of operation (CTR).

In such a context, where the adversary can control the IV, the initialization function must be designed in a way such that it is not possible to find a collision between keystream blocks unless around $2^{\frac{n}{2}}$ keystream blocks are considered. This situation occurs for the modified CTR mode proposed by the 3GPP project (3rd Generation PartnerShip Project) in the specifications of the Milenage algorithm, which ensures authentication and key distribution in UMTS [3rd01, Gil03]. This mode is depicted on Figure 11.7, and a more general version is described and studied in [Gil03].

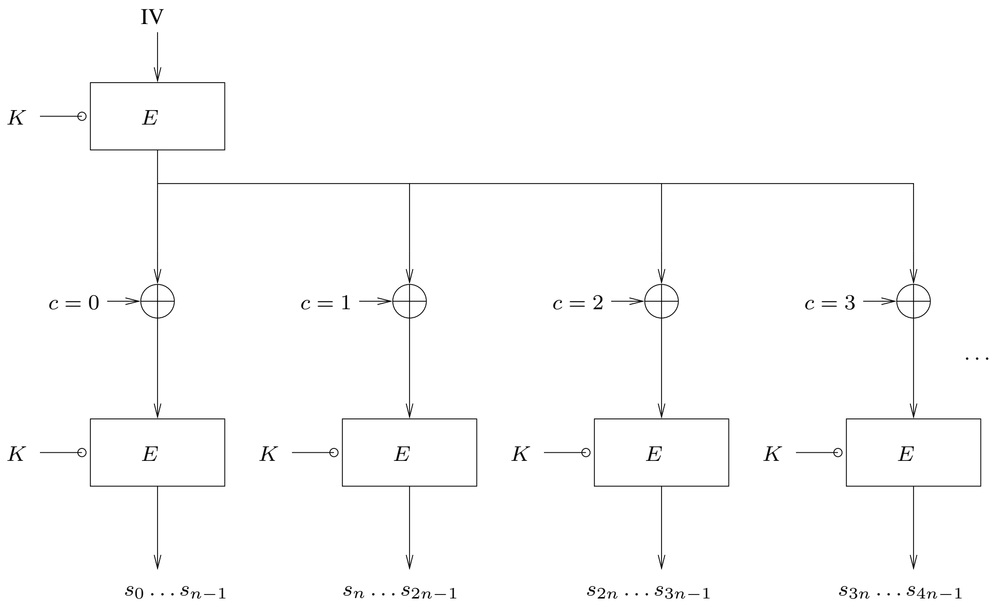


Figure 11.7: Modified Counter mode of operation.

11.4.4 Dedicated generators

These generators are ad-hoc designs and do not rely on any other cryptographic primitive. This family includes all generators with better performance than classical block ciphers. The speed in a software environment, like on a usual PC, of a block cipher such as the AES is around 20 cycles for encrypting one byte, while some dedicated stream ciphers are able to achieve between 3 and 5 cycles per encrypted byte (see [eba]). Similarly, only dedicated generators can have a low-cost hardware implementation with a very limited power consumption. This type of generators is then preferably used in some applications like embedded systems. However, since several devastating attacks have been published on stream ciphers much after the main results on block ciphers, the standardization process for stream ciphers is more recent. MUGI, SNOW 2.0, Rabbit, decim^{v2} and KCipher-2 have been proposed in the last version of the ISO/IEC 18033-4 standard (December 2011). Also, the eSTREAM project [ECR05], initiated by the European Network of Excellence ECRYPT, was an international competition on stream ciphers. It has received more than 30 new stream ciphers in 2005. Among them, 7 ciphers have been included in a portfolio of recommended stream ciphers: HC-128, Rabbit, Salsa20/12 and Sosemanuk for software profile, and Grain v1, MICKEY 2.0 and Trivium for hardware profile[c].

Table-driven generators and RC4. As previously mentioned, dedicated generators mainly rely on two design strategies. The next chapter will focus on the generators with a small internal size, which are appropriate for hardware implementations in constrained environments. An alternative strategy consists in choosing a relatively large internal state, and a random-looking transition function. This strategy has been mainly implemented by table-driven generators. The first and most famous one is RC4 [Riv92]. It has been design in 1987 for RSA Data Security. RC4 is not public and its specifications of RC4 are a trade secret. However, there exists a public and free stream cipher named *Alleged RC4* (aka ARC4). RC4 and ARC4 actually correspond to the same algorithm but the name ARC4 is sometimes preferred to avoid trademark problems.

Even if it is now known to be weak, RC4 has been widely deployed because of its simplicity and its high throughput. For instance, it is one of the possible ciphers in the SSL/TLS protocol (to guarantee confidentiality of web transactions), and also in WEP (former security protocol for IEEE 802.11 wireless networks, superseded by WPA and WPA2). The keysize in RC4 is variable, between 40 and 1024 bits. The cipher does not accommodate any IV. The internal state is an array of 2^n words of n bits. In most cases, $n = 8$ is chosen, i.e., the internal state is an array of 256 bytes. At each clock, this array defines a permutation of n-bit words.

Description. RC4 is composed of an initialization function, which computes the initial state of the array from the secret key. Then, at each clock, the array is modified by permuting two elements, and an n-bit keystream word is produced, which is defined by some element in the array.

Several statistical biases have been detected in the sequence produced by RC4. These biases mainly originate from weaknesses in the initialization phase (see e.g. [MS01, PM09]). A systematic analysis of statistical biases in RC4 has been conducted in [ABP$^+$13] and leads to an attack against the TLS/SSL protocol with RC4.

[c]An 8th cipher, named F-FCSR, was included in the original eSTREAM portfolio but has been removed from the list after some cryptanalytic work.

Algorithm 6 RC4 (All additions in this algorithm are modulo 2^n).

```
Input. $K$, composed of $k$ $n$-bit words, $K[0], \dots, K[k-1]$.
/* Initialization */
for $i$ from 0 to $2^n - 1$ do
  $S[i] \leftarrow i$
end for
$j \leftarrow 0$
for $i$ from 0 to $2^n - 1$ do
  $j \leftarrow j + S[j] + K[i \bmod k]$
  swap values $S[i]$ and $S[j]$
end for

/* Keystream generation */
$i \leftarrow 0, j \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $j \leftarrow j + S[i]$
  swap values $S[i]$ and $S[j]$.
  return $S[S[i] + S[j]]$
until enough keystream words have been generated
```

It is worth noticing that the attack against WEP IEEE 802.11 by Fluhrer, Mantin and Shamir [FMS01] comes from the weak resynchronization mechanism used in the protocol due to the absence of IV.

Bibliography

[3rd01] 3rd Generation PartnerShip Project. 3GPP TS 35.206 - Specification of the MILENAGE algorithm set: An example algorithm Set for the 3GPP Authentication and Key Generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification, 2001. `http://www.3gpp.org/ftp/Specs/html-info/35206.htm`.

[ABL08] François Arnault, Thierry P. Berger, and Cédric Lauradoux. F-FCSR stream ciphers. In *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 170–178. Springer, 2008.

[ABP+13] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the security of RC4 in TLS. In *Proceedings of the 22th USENIX Security Symposium*, pages 305–320. USENIX Association, 2013.

[ACGS88] Werner Alexi, Benny Chor, Oded Goldreich, and Claus-Peter Schnorr. RSA and Rabin Functions: certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, 1988.

[AR99] Yonatan Aumann and Michael O. Rabin. Information theoretically secure communication in the limited storage space model. In *Advances in Cryptology - CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 65–79. Springer-Verlag, 1999.

[Bab95] Steve Babbage. A space/time trade-off in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection*, number 408. IEEE Conference Publication, 1995.

[BBS86] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredicable pseudo-random number generator. *SIAM Journal on Computing*, 15, 1986.

[BD06] Steve Babbage and Matthew Dodd. Submission to the eSTREAM project [ECR05], 2006. `http://www.ecrypt.eu.org/stream/e2-mickey.html`.

[BD08] Steve Babbage and Matthew Dodd. The MICKEY stream ciphers. In *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2008. see also `http://www.ecrypt.eu.org/stream/e2-mickey.html`.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Foundations of Computer Science - FOCS '97*, pages 394–403. IEEE Computer Society, 1997.

[BG07] Côme Berbain and Henri Gilbert. On the security of IV dependent stream ciphers. In *Fast Software Encryption - FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2007.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13, 1984.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[Cro95] William P. Crowell. Remembrances of Venona, 1995. `https://www.nsa.gov/public_info/declass/venona/remembrances.shtml`.

[eba] eBACS: ECRYPT Benchmarking of Cryptographic Systems. `http://bench.cr.yp.to`.

[ECR05] ECRYPT - European Network of Excellence in Cryptology. The eSTREAM Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`, 2005.

[FMS01] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In *Selected Areas in Cryptography - SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2001.

[FO90] Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In *Advances in Cryptology - EUROCRYPT'89,*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, 1990.

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[Gil03] Henri Gilbert. The security of "One-Block-to-Many" Modes of Operation. In *Fast Software Encryption - FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 376–395. Springer-Verlag, 2003.

[Gol82] Solomon W. Golomb. *Shift register sequences.* Aegean Park Press, 1982.

[Gol97] J. Golic. Cryptanalysis of alleged A5 stream cipher. In *Advances in Cryptology - EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1997.

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[HK05] Jin Hong and Woo-Hwan Kim. TMD-Tradeoff and state entropy loss considerations of streamcipher MICKEY. In *Progress in Cryptology - INDOCRYPT 2005*, volume 3797 of *Lecture Notes in Computer Science*, pages 169–182. Springer, 2005.

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis.* Chapman & Hall/CRC, 2009.

[Kah67] David Kahn. *The Codebreakers.* Scribner, 1967.

[Knu69] Donald E. Knuth. *The Art of Computer Programming*, volume 2 - Seminumerical Algorithms. Addison Wesley, 1969.

[Mau92] Ueli Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, 1992.

[MS01] Itsik Mantin and Adi Shamir. A practical attack on broadcast RC4. In *Fast Software Encryption - FSE 2001*, volume 2335 of *Lecture Notes in Computer Science*, pages 152–164. Springer-Verlag, 2001.

[MT79] Robert Morris and Ken Thompson. Password security - A case history. *Commun. ACM*, 22(11):594–597, 1979.

[MvOV97] Affred J. Menezes, Paul C. van Oorshot, and Scott A. Vanstone. *Handbook of applied cryptography.* CRC Press, 1997. `http://www.cacr.math.uwaterloo.ca/hac/`.

[Nat01] National Institute of Standards and Technology. NIST Special Publication 800-38A — Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A, 2001.

[Nat10] National Institute of Standards and Technology. NIST Special Publication 800-22 Revision 1a — A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications. NIST, 2010.

[NS90] Kazuo Nishimura and Masaaki Sibuya. Probability to meet in the middle. *J. Cryptology*, 2(1):13–22, 1990.

[PM09] Goutam Paul and Subhamoy Maitra. On biases of permutation and keystream bytes of RC4 towards the secret key. *Cryptography and Communications*, 1(2):225–268, 2009.

[QD88] Jean-Jacques Quisquater and Jean-Paul Delescaille. Other cycling tests for DES (abstract). In *Advances in Cryptology - CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 255–256. Springer, 1988.

[Rab05] Michael O. Rabin. Provably unbreakable hyper-encryption in the limited access model. In *Proceedings of the 2005 IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security*, pages 34–37. IEEE, 2005.

[Riv92] Ron Rivest. The RC4 encryption algorithm. RSA Data Security, 1992.

[Röc08a] Andrea Röck. Entropy of the internal state of an FCSR in Galois representation. In *Fast Software Encryption - FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 343–362. Springer, 2008.

[Röc08b] Andrea Röck. Stream ciphers using a random update function: Study of the entropy of the inner state. In *Progress in Cryptology - AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2008.

[Röc09] Andrea Röck. *Quantifying Studies of (Pseudo) Random Number Generation for Cryptography.* PhD thesis, Ecole Polytechnique, Palaiseau, France, 2009.

[Rog11] Phillip Rogaway. Evaluation of some blockcipher modes of operation. CRYPTREC report, 2011.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Techical Journal*, 27, 1948.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.

[SS03] André Seznec and Nicolas Sendrier. HAVEGE: a user-level software heuristic for generating empirically strong random numbers. *ACM Trans. Model. Comput. Simul.*, 13(4):334–346, 2003.

[Ver26] Gilbert S. Vernam. Cipher printing telegraph systems for secret wire and radio telegraph communications. *Journal of the American Institute for Electrical Engineers*, (55), 1926.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the IEEE 23rd Annual Symposium on Foundations of Computer Science*, pages 80–91. IEEE, 1982.

Chapter 12

LFSR-based Stream Ciphers

In order to minimize the size of the internal state, stream ciphers dedicated to low-cost hardware implementations may use a linear transition function. Among all such possibilities, linear feedback shift registers (LFSRs) offer several advantages including their performance, their implementation cost and many theoretical results on the statistical properties of the produced sequences. LFSR-based generators are then probably the most commonly studied class of keystream generators. This class includes both hardware-oriented stream ciphers and software-oriented ciphers, but this second type of applications usually relies on non-binary LFSRs, operating on a larger alphabet (e.g. on 32-bit words). The most widely used LFSR-based stream ciphers include E0 (used in the Bluetooth standard), A5/1 used for encrypting the over-the-air communications in the GSM cellular telephone standard, SNOW 2.0 (ISO/IEC 18033-4 standard) and its variant SNOW 3G used in UMTS 3G networks.

In most practical LFSR-based generators used nowadays, the internal state is divided into two parts: one is updated linearly by an LFSR, and the other one is updated with a nonlinear function in order to prevent the main attacks exploiting the linearity of the transition function. This nonlinear part may be small (and seen as a nonlinear memory) as in E0 or SNOW 2.0, or both parts of the internal state may be of equal size, like in MUGI or Grain.

12.1 Main properties of LFSRs

12.1.1 Definitions

An LFSR of length L over $\mathbb{F}_q$ is a finite state automaton which produces a semi-infinite sequence of elements of $\mathbb{F}_q$, $\mathbf{s} = (s_t)_{t\geq 0}$, satisfying a linear recurrence relation of degree L over $\mathbb{F}_q$

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

The L coefficients $c_1, \ldots, c_L$ are elements of $\mathbb{F}_q$. They are called the *feedback coefficients* of the LFSR.

The *Fibonacci representation* of an LFSR of length L over $\mathbb{F}_q$ has the form depicted on Figure 12.1. The register consists of L delay cells, called *stages*, each containing an element of $\mathbb{F}_q$. The contents of the L stages, $s_t, \ldots, s_{t+L-1}$, form the *state* of the LFSR. The L stages are initially loaded with L elements, $s_0, \ldots, s_{L-1}$, which can be arbitrary chosen in $\mathbb{F}_q$; they form the initial state of the register.

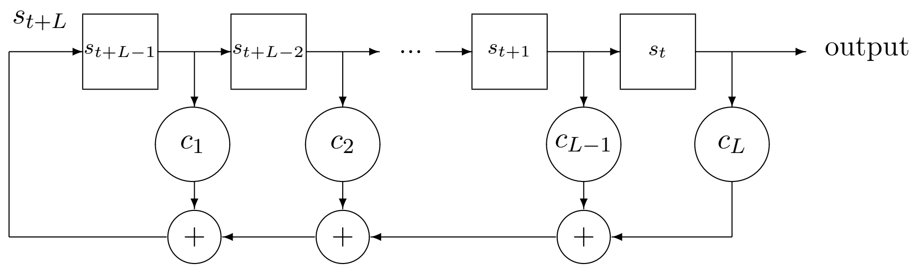


Figure 12.1: Fibonacci representation of an LFSR of length L.

The shift register is controlled by an external clock. At each time unit, each digit is shifted one stage to the right. The content of the rightmost stage s_t is output. The new content of the leftmost stage is the *feedback bit*, s_{t+L}. It is obtained by a linear combination of the contents of the register stages, where the coefficients of the linear combination are given by the feedback coefficients of the LFSR:

$$s_{t+L} = \sum_{i=1}^{L} c_i s_{t+L-i} \ .$$

Therefore, the LFSR implements the linear recurrence relation of degree L:

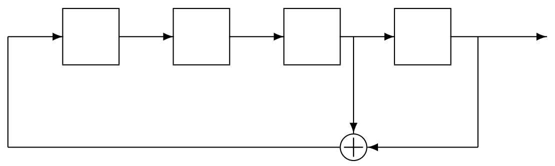
$$s_{t+L} = \sum_{i=1}^{L} c_i s_{t+L-i}, \ \ \forall t \geq 0 \ .$$

Example 12.1. A binary LFSR of length 4. Table 12.1 gives the successive states of the binary LFSR of length 4 with feedback coefficients $c_1 = c_2 = 0$, $c_3 = c_4 = 1$ and with initial state $(s_0, s_1, s_2, s_3) = (1, 0, 1, 1)$. This LFSR is depicted in Figure 12.2. It corresponds to the linear recurrence relation

$$s_{t+4} = s_{t+1} + s_t \bmod 2 \ .$$

The output sequence $s_0 s_1 \ldots$ generated by this LFSR is $1011100\ldots$.

Figure 12.2: Binary LFSR with feedback coefficients $(c_1, c_2, c_3, c_4) = (0, 0, 1, 1)$



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 $c_1, \ldots, c_L$ of an LFSR of length L are usually represented by the LFSR *feedback polynomial* (or *connection polynomial*) defined by

$$P(X) = 1 - \sum_{i=1}^{L} c_i X^i \ .$$

Table 12.1: Successive states of the LFSR with feedback coefficients $(c_1, c_2, c_3, c_4) = (0, 0, 1, 1)$ and with initial state $(s_0, s_1, s_2, s_3) = (1, 0, 1, 1)$

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_t	1	0	1	1	1	1	0	0	0	1	0	0	1	1	0	1
s_{t+1}	0	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0
s_{t+2}	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1
s_{t+3}	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1

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

$$P^\star(X) = X^L P(1/X) = X^L - \sum_{i=1}^{L} c_i X^{L-i} \ .$$

For instance, the feedback polynomial of the binary LFSR shown in Figure 12.2 is $P(X) = 1 + X^3 + X^4$ and its characteristic polynomial is $P^\star(X) = 1 + X + X^4$.

Non-singular LFSRs. An LFSR is said to be *non-singular* if the degree of its feedback polynomial is equal to the LFSR length (*i.e.*, if the feedback coefficient c_L differs from 0). In this case, the transition function of the LFSR is bijective. Any sequence generated by a non-singular LFSR of length L is periodic, and its period does not exceed $q^L - 1$. Indeed, the LFSR has at most q^L different states and the all-zero state is always followed by the all-zero state. Moreover, if the LFSR is singular, all generated sequences are *ultimately periodic*, *i.e.*, the sequences obtained by ignoring a certain number of elements at the beginning are periodic.

12.1.2 Characterization of LFSR output sequences

A given LFSR of length L over $\mathbb{F}_q$ can generate q^L different sequences corresponding to the q^L different initial states and these sequences form a vector space over $\mathbb{F}_q$. The set of all sequences generated by an LFSR with feedback polynomial P is characterized by the following property [Zie59].

Theorem 12.1. *A sequence $(s_t)_{t\geq 0}$ is generated by an LFSR of length L over $\mathbb{F}_q$ with feedback polynomial P if and only if there exists a polynomial $Q \in \mathbb{F}_q[X]$ with $\deg(Q) < L$ such that the generating function of $(s_t)_{t\geq 0}$ satisfies*

$$\sum_{t\geq 0} s_t X^t = \frac{Q(X)}{P(X)} \ .$$

Moreover, the polynomial Q is completely determined by the coefficients of P and by the initial state of the LFSR:

$$Q(X) = -\sum_{j=0}^{L-1} X^j \left(\sum_{k=0}^{k} s_k c_{j-k} \right) \ ,$$

where $P(X) = -\sum_{i=0}^{L} c_i X^i$.

Proof.

$$\begin{aligned}\left(-\sum_{i=0}^{L} c_i X^i\right)\left(\sum_{t=0}^{\infty} s_t X^t\right) &= \sum_{j=0}^{\infty} X^j \left(-\sum_{k=\max(0,j-L)}^{j} s_k c_{j-k}\right)\\ &= -\sum_{j=0}^{L-1} X^j \left(\sum_{k=0}^{j} s_k c_{j-k}\right) + \sum_{j=L}^{\infty} X^j \left(\sum_{k=j-L}^{j} s_k c_{j-k}\right).\end{aligned}$$

Therefore, we deduce that

$$Q(X) = -\left(\sum_{i=0}^{L} c_i X^i\right)\left(\sum_{t=0}^{\infty} s_t X^t\right)$$

is a polynomial of degree strictly less than L if and only if the second right-hand term vanishes, i.e.,

$$\sum_{k=j-L}^{j} s_k c_{j-k} = 0$$

for all $j \geq L$. ◇

This result, which is called the fundamental identity of formal power series of linear recurring sequences, means that there is a one-to-one correspondence between the sequences generated by an LFSR of length L with feedback polynomial P and the fractions $Q(X)/P(X)$ with $\deg(Q) < L$. It has two major consequences. On the first hand, any sequence generated by an LFSR with feedback polynomial P is also generated by any LFSR whose feedback polynomial is a multiple of P. This property is widely used for attacking LFSR-based generators (e.g., in distinguishing attacks, and in fast correlation attacks). It may also be helpful since some multiple of the feedback polynomial may provide more appropriate representations in some contexts.

Example 12.2. Let $\boldsymbol{s}$ be a binary sequence satisfying

$$s_{t+6} = s_{t+4} + s_{t+3} + s_{t+1} + s_t,\ \forall t \geq 6\ .$$

The corresponding feedback polynomial is then $P(X) = 1 + X^2 + X^3 + X^5 + X^6$. It then follows that $\boldsymbol{s}$ also satisfies

$$s_{t+8} = s_{t+7} + s_t$$

since $1 + X + X^8 = (1 + X^2 + X^3 + X^5 + X^6)(1 + X + X^2)$. This alternative recursion may then have a lower implementation cost because of its sparsity.

On the other hand, Theorem 12.1 implies that a sequence generated by an LFSR with feedback polynomial P is also generated by a shorter LFSR with feedback polynomial P' if the corresponding fraction $Q(X)/P(X)$ is such that $\gcd(P,Q) \neq 1$. Thus, amongst all sequences generated by the LFSR with feedback polynomial P, there is one which can be generated by a shorter LFSR if and only if P is not irreducible over $\mathbb{F}_q$. This leads to the following natural notion of *minimal polynomial*.

Definition 12.2. *For any linear recurring sequence $(s_t)_{t\geq 0}$, there exists a unique polynomial P_0 with constant term equal to* 1*, such that the generating function of $(s_t)_{t\geq 0}$ is given by*

$$\sum_{t\geq 0} s_t X^t = Q_0(X)/P_0(X)$$

where P_0 and Q_0 are relatively prime.

Then, the shortest LFSR which generates $(s_t)_{t\geq 0}$ has length $L = \max(\deg(P_0), \deg(Q_0) + 1)$, and its feedback polynomial is equal to P_0. The reciprocal polynomial of P_0, $X^L P_0(1/X)$, is the characteristic polynomial of the shortest LFSR which generates $(s_t)_{t\geq 0}$; it is called the minimal polynomial *of the sequence.*

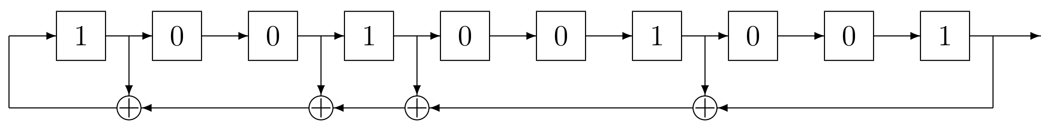
The minimal polynomial of a linear recurring sequence then determines the linear recurrence relation of least degree satisfied by the sequence.

Example 12.3. The binary LFSR of length 10 depicted in Figure 12.3 has feedback polynomial

$$P(X) = 1 + X + X^3 + X^4 + X^7 + X^{10} ,$$

and its initial state $s_0 \dots s_9$ is 1001001001.

Figure 12.3: Example of a LFSR of length 10.



The generating function of the sequence produced by this LFSR is given by

$$\sum_{t\geq 0} s_t X^t = \frac{Q(X)}{P(X)}$$

where Q is deduced from the coefficients of P and from the initial state:

$$Q(X) = 1 + X + X^7 .$$

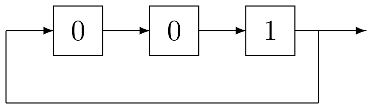
Therefore, we have

$$\sum_{t\geq 0} s_t X^t = \frac{1 + X + X^7}{1 + X + X^3 + X^4 + X^7 + X^{10}} = \frac{1}{1 + X^3} ,$$

since $1 + X + X^3 + X^4 + X^7 + X^{10} = (1 + X + X^7)(1 + X^3)$ in $\mathbb{F}_2[X]$. This implies that $(s_t)_{t\geq 0}$ is also generated by the LFSR with feedback polynomial $P_0(X) = 1 + X^3$ depicted in Figure 12.4. The minimal polynomial of the sequence is then $1 + X^3$.

Obviously, in all cryptographic applications, the feedback polynomials of LFSRs are always chosen irreducible.

Figure 12.4: LFSR of length 3 which generates the same sequence as the LFSR in Figure 12.3



12.1.3 Period of a linear recurring sequence

Another important role played by the minimal polynomial is that it determines the period of a linear recurring sequence.

Proposition 12.3. *The least period of a linear recurring sequence is equal to the order of its minimal polynomial P_0, i.e., to the least positive integer e for which $P_0(X)$ divides $X^e + 1$.*

For instance, the sequence studied in Example 12.3 has minimal polynomial X^3+1. Then, it has period 3. On the other hand, any non-zero sequence generated by the LFSR of length 4 depicted in Figure 12.2 has period $2^4 - 1 = 15$. Indeed, the minimal polynomial of any such sequence corresponds to its characteristic polynomial $P_0(X) = 1 + X + X^4$, because P_0 is irreducible.

We directly deduce from Proposition 12.3 that a sequence has maximal period $2^{\deg P_0} - 1$ if and only if P_0 is a *primitive polynomial.* The sequences produced by an LFSR with primitive feedback polynomial are called *maximal-length sequences* (m-sequences).

12.1.4 Statistical properties of m-sequences

Maximum-length sequences, i.e., the linear recurring sequences produced by an LFSR with primitive feedback polynomial, possess several good statistical properties which make them appropriate building-blocks in keystream generators.

For instance, any binary sequence produced by an LFSR of length L with primitive feedback polynomial satisfy the following properties. The first three properties are called Golomb's randomness postulates [Gol82].

- *Balance property:* The difference between the number of ones and the number of zeroes in any window of $2^L - 1$ consecutive bits is equal to 1:

$$\#\{i,\ s_{t_0+i} = 1,\ 0 \le i < 2^L - 1\} - \#\{i,\ s_{t_0+i} = 0,\ 0 \le i < 2^L - 1\} = 1 \ .$$

- *Runs:* a run is a set of consecutive zeroes flanked by ones, or of consecutive ones flanked by zeroes. For instance, the sequence 0100011 has a run of zeroes of length 3. The proportion of runs of length i within any frame of $(2^L - 1)$ consecutive bits of an m-sequence is 2^{-i}, $0 \le i < L$. Moreover, among all runs of length i, $i \le L - 2$, the number of runs of zeroes and the number of runs of ones are equal. There is exactly one run of length $(L - 1)$ and one run of length L (see Example 12.4).

- *Two-level auto-correlation:* The autocorrelation of a binary sequence of period N is defined by

$$C(\tau) = \sum_{t=0}^{2^L-2} (-1)^{s_t + s_{t+\tau \bmod (2^L-1)}} \ .$$

$C(\tau)$ then measures the distance between the sequence $s_0s_1\ldots s_{2^L-2}$ and the sequence obtained by shifting it by τ positions, i.e., $s_\tau s_{\tau+1}\ldots s_{\tau-1}$. Indeed, we have

$$C(\tau) = 2^L - 1 - 2\#\{t,\ s_t \neq s_{t+\tau \bmod (2^L-1)}, 0 \leq t < 2^L - 1\} .$$

In particular, $C(\tau) = 2^L - 1$ if and only if τ is a multiple of the period of the sequence since the two sequences are identical. For any m-sequence generated by an LFSR with primitive feedback polynomial of degree L, we have that, if τ is not a multiple of (2^L-1), then $C(\tau) = -1$. This means that the sequence is as far as possible from its shifted versions. This property is widely used in telecommunications for synchronization.

- *Multigram property:* the L-tuple $s_ts_{t+1}\ldots s_{t+L-1}$ takes all the $2^L - 1$ nonzero values when t varies between 0 and $(2^L - 2)$.

Some additional properties of m-sequences are detailed in [Hel11].

Example 12.4. Let us consider the first 31 bits of the binary sequence produced by the LFSR of length 5 with primitive feedback polynomial $X^5 + X^3 + 1$ from initial state 10000:

1000010101110110001111100110100

It can be checked that this sequence consists of 16 ones and 15 zeroes. It then satisfies the balance property.

The sequence has 16 runs: 8 runs of length 1 (4 runs of zeroes and 4 runs of ones), 4 runs of length 2 (2 runs of zeroes and 2 runs of ones), 2 runs of length 3 (1 run of zeroes and 1 run of ones), one run of zeroes of length 4 and one run of ones of length 5.

12.1.5 LFSR and multiplication in a finite field

The operation performed by a q-ary LFSR of length L with irreducible feedback polynomial corresponds to a multiplication in the finite field $\mathbb{F}_{q^L}$.

Proposition 12.4. *Let $P^\star$ be an irreducible polynomial in $\mathbb{F}_q[X]$ with degree L. Let $\alpha \in \mathbb{F}_{q^L}$ be a root of $P^\star$ and $\{\beta_0, \ldots, \beta_{L-1}\}$ denote the dual basis of $\{1, \alpha, \ldots, \alpha^{L-1}\}$, i.e.,*

$$\mathsf{Tr}(\alpha^i\beta_j) = \begin{cases} 0 & \text{if } i \neq i \\ 1 & \text{if } i = j \end{cases} ,$$

where Tr denotes the trace function from $\mathbb{F}_{q^L}$ into $\mathbb{F}_q$.

Then, the content of the LFSR with characteristic polynomial $P^\star$ at time $(t+1)$ is equal to its content at time t multiplied by α, where these vectors are identified with elements in the field $\mathbb{F}_{q^L}$ decomposed on the basis $\{\beta_0, \ldots, \beta_{L-1}\}$.

Proof. For any t, we identify an L-tuple $(x_0, \ldots, x_{L-1})$ with an element in the finite field $\mathbb{F}_{q^L}$ by

$$x = x_{L-1}\beta_{n-1} + \ldots + x_0\beta_0 .$$

Then, by definition of the dual basis, we have that, for any $0 \leq i < L$,

$$\begin{aligned} \mathsf{Tr}(\alpha^i x) &= \sum_{j=0}^{L-1} \mathsf{Tr}(x_j\alpha^i\beta_j) \\ &= \sum_{j=0}^{L-1} x_j \mathsf{Tr}(\alpha^i\beta_j) = x_i . \end{aligned}$$

This means that the i-th coordinate of x in the basis $\{\beta_0, \ldots, \beta_{L-1}\}$ is equal to $\mathsf{Tr}(\alpha^i x)$. Let us now compute the coordinates of $y = \alpha x$ in this basis. The i-th coordinate of y is given by

$$\mathsf{Tr}(\alpha^i y) = \mathsf{Tr}(\alpha^{i+1} x) \ .$$

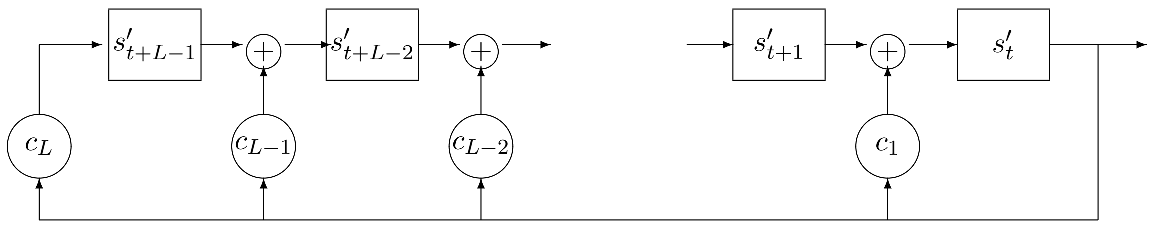
It follows that the i-th coordinate of y is equal to the $(i+1)$-th coordinate of x if $i < L-1$. For $i = L-1$, the last coordinate of y is given by

$$\begin{aligned} \mathsf{Tr}(\alpha^{L-1} y) &= \mathsf{Tr}(\alpha^L x) \\ &= \mathsf{Tr}\left(\sum_{i=1}^{L} c_i \alpha^{L-i} x\right) \\ &= \sum_{i=1}^{L} c_i \mathsf{Tr}(\alpha^{L-i} x) = \sum_{i=1}^{L} c_i x_{L-i} \ , \end{aligned}$$

where the characteristic polynomial is given by $P^\star(X) = X^L - \sum_{i=1}^{L} c_i X^{L-i}$, implying that $\alpha^L = \sum_{i=1}^{L} c_i \alpha^{L-i}$. It follows that the coordinates of $y = \alpha x$ correspond to the content after one clock of the LFSR initialized by x. $\diamond$

Galois representation. Since an LFSR with irreducible feedback polynomial is a device which implements the multiplication by an element α in a finite field, some alternate automorphism between the $\mathbb{F}_{q^L}$ and $\mathbb{F}_q^L$ may be used without modifying the transition function over $\mathbb{F}_{q^L}$. Another natural representation is obtained when the basis $\{1, \alpha, \alpha^2, \ldots, \alpha^{L-1}\}$ is used for representing the elements in $\mathbb{F}_{q^L}$ instead of the dual basis $\{\beta_0, \ldots, \beta_{L-1}\}$. This representation is called the *Galois representation*, and corresponds to the "natural" multiplication circuit, i.e., the multiplication in the so-called polynomial basis. The Galois representation corresponding to the Fibonacci LFSR depicted on Figure 12.1 is given in Figure 12.5.

Figure 12.5: Galois representation of the LFSR depicted on Figure 12.1.



It is worth noticing that Fibonacci and Galois representations have different features. For instance, the Galois representation is obviously more efficient in software than the Fibonacci representation. Also, the diffusion within the register in the Galois representation is faster.

12.2 Linear complexity and LFSR synthesis

A fundamental quantity for a sequence is its linear complexity since it determines the smallest linear recursion satisfied by the sequence, or equivalently the length of the smallest LFSR generating the sequence.

Definition 12.5. *The* linear complexity *of a semi-infinite sequence* $\mathbf{s} = (s_t)_{t \geq 0}$ *of elements of* $\mathbb{F}_q$, $\Lambda(\mathbf{s})$, *is the smallest integer* Λ *such that* $\mathbf{s}$ *can be generated by an LFSR of length* Λ *over* $\mathbb{F}_q$, *and is* ∞ *if no such LFSR exists. By way of convention, the linear complexity of the all-zero sequence is equal to* 0. *The linear complexity of a linear recurring sequence corresponds to the degree of its minimal polynomial.*

The linear complexity $\Lambda(\mathbf{s}^n)$ *of a finite sequence* $\mathbf{s}^n = s_0 s_1 \ldots s_{n-1}$ *of* n *elements of* $\mathbf{F}_q$ *is the length of the shortest LFSR which produces* $\mathbf{s}^n$ *as its first* n *output terms for some initial state.*

The linear complexity of an infinite linear recurring sequence $\mathbf{s}$ and the linear complexity of the finite sequence $\mathbf{s}^n$ composed of the first n digits of $\mathbf{s}$ are related by the following property: if $\mathbf{s}$ is an infinite linear recurring sequence with linear complexity Λ, then the finite sequence $\mathbf{s}^n$ has linear complexity Λ for any $n \geq 2\Lambda$ [Mas69]. Moreover, the unique LFSR of length Λ that generates $\mathbf{s}$ is the unique LFSR of length Λ that generates $\mathbf{s}^n$ for every $n \geq 2\Lambda$.

12.2.1 Linear complexity as a statistical test

For a sequence $\mathbf{s} = s_0 s_1 \ldots$, the sequence of the linear complexities $(\Lambda(\mathbf{s^n}))_{n \geq 1}$ of all subsequences $\mathbf{s}^n = s_0 \ldots s_{n-1}$ composed of the first n terms of $\mathbf{s}$ is called the *linear complexity profile* of $\mathbf{s}$.

Proposition 12.6. *[Rue86, Page 40] The expected linear complexity of a binary sequence* $\mathbf{s}^n = s_0 \ldots s_{n-1}$ *of* n *independent and uniformly distributed binary random variables is*

$$E[\Lambda(\mathbf{s}^n)] = \frac{n}{2} + \frac{4 + \varepsilon(n)}{18} + 2^{-n}\left(\frac{n}{3} + \frac{2}{9}\right) ,$$

where $\varepsilon(n) = n \bmod 2$.

Therefore, it may be possible to distinguish a sequence from a truly random sequence by computing its linear complexity profile and comparing the result to what is expected from Proposition 12.6. Further results on the linear complexity and on the linear complexity profile of random sequences can be found in [Rue86].

12.2.2 Berlekamp-Massey algorithm

The *Berlekamp-Massey algorithm* is an algorithm for determining the linear complexity of a finite sequence and the feedback polynomial of an LFSR of minimal length which generates this sequence. This algorithm is due to Massey [Mas69], who showed that the iterative algorithm proposed in 1967 by Berlekamp [Ber68] for decoding BCH codes can be used for finding the shortest LFSR that generates a given sequence. Given sequence $\mathbf{s}^n$ of length n, the Berlekamp-Massey performs n iterations. The t-th iteration determines an LFSR of minimal length which generates the first t digits of $\mathbf{s}^n$. The algorithm is described in Algorithm 7. In the particular case of a binary sequence, the quantity d' does not need to be stored since it is always equal to 1. Moreover, the feedback polynomial is simply updated by

$$P(X) \leftarrow P(X) + P'(X)X^{t-m} .$$

The number of operations performed for computing the linear complexity of a sequence of length n is $\mathcal{O}(n^2)$. It can be proved that the Berlekamp-Massey algorithm and the Euclidean algorithm are essentially the same [Dor87].

Algorithm 7 The Berlekamp-Massey algorithm.

```
Input: $\mathbf{s}^n = s_0 s_1 \dots s_{n-1}$, a sequence of $n$ elements of $\mathbb{F}_q$.
Output: $\Lambda$, the linear complexity of $\mathbf{s^n}$ and $P$, the feedback polynomial of an LFSR of
length $\Lambda$ which generates $\mathbf{s}^n$.
/* Initialization */
$P(X) \leftarrow 1$, $P'(X) \leftarrow 1$, $\Lambda \leftarrow 0$, $m \leftarrow -1$, $d' \leftarrow 1$.
/* Algorithm */
for $t$ from 0 to $n-1$ do
  $d \leftarrow s_t + \sum_{i=1}^{\Lambda} p_i s_{t-i}$.
  if $d \neq 0$ then
    $T(X) \leftarrow P(X)$.
    $P(X) \leftarrow P(X) - d(d')^{-1} P'(X) X^{t-m}$.
    if $2\Lambda \leq t$ then
      $\Lambda \leftarrow t + 1 - \Lambda$.
      $m \leftarrow t$.
      $P'(X) \leftarrow T(X)$.
      $d' \leftarrow d$.
    end if
  end if
end for
return $\Lambda$ and $P$
```

The LFSR of minimal length that generates a sequence $\mathbf{s}^n$ of length n is unique if and only if $n \geq 2\Lambda(\mathbf{s}^n)$, where $\Lambda(\mathbf{s}^n)$ is the linear complexity of $\mathbf{s}^n$.

Obviously, the linear complexity $\Lambda(\mathbf{s})$ of a semi-infinite linear recurring sequence $\mathbf{s} = (s_t)_{t \geq 0}$ is equal to the linear complexity of the finite sequence composed of the first n terms of $\mathbf{s}$ for any $n \geq \Lambda(\mathbf{s})$. Thus, the Berlekamp-Massey algorithm determines the shortest LFSR that generates an infinite linear recurring sequence $\mathbf{s}$ from the knowledge of any $2\Lambda(\mathbf{s})$ consecutive digits of $\mathbf{s}$.

Example 12.5. Table 12.2 describes the successive steps of the Berlekamp-Massey algorithm applied to the binary sequence of length 7, $s_0 \dots s_6 = 0111100$. The values of Λ and P

t	s_t	d	Λ	$P(X)$	m	$P'(X)$
			0	1	-1	1
0	0	0	0	1	-1	1
1	1	1	2	$1+X^2$	1	1
2	1	1	2	$1+X+X^2$	1	1
3	1	1	2	$1+X$	1	1
4	1	0	2	$1+X$	1	1
5	0	1	4	$1+X+X^4$	5	$1+X$
6	0	0	4	$1+X+X^4$	5	$1+X$

Table 12.2: Successive steps of the Berlekamp-Massey algorithm applied to the binary sequence of length 7, $s_0 \dots s_6 = 0111100$.

obtained at the end of step t correspond to the linear complexity of the sequence $s_0 \dots s_t$ and to the feedback polynomial of an LFSR of minimal length that generates it.

12.3 Classical constructions of LFSR-based generators

It is clear that an LFSR should never be used by itself as a keystream generator. If the feedback coefficients are publicly known (which is usually the case when the LFSR is implemented in hardware), the entire keystream can obviously be recovered from the knowledge of any Λ consecutive bits of the keystream, where Λ is the linear complexity of the running-key (which does not exceed the LFSR length). If the feedback coefficients are kept secret, the entire keystream can be recovered from any 2Λ consecutive bits of the keystream by the Berlekamp-Massey algorithm.

However, LFSRs are extremely fast and low-cost devices and they generate sequences with good statistical properties, in particular with a high period. Therefore, they are often used as building-blocks in dedicated keystream generators, but within a more complex system. In particular, three classical constructions based on LFSR aim at increasing the linear complexity of the generated sequence at a low implementation cost. These three methods have received a lot of attention and have been widely used within stream ciphers. They include the combination generator, the filter generator and the generators based on LFSR with irregular clocking.

12.3.1 Combination generators

A *combination generator* is a keystream generator composed of several LFSRs whose outputs are combined by a Boolean function to produce the keystream. Then, the output sequence $(s_t)_{t\geq 0}$ of a combination generator composed of n LFSRs is given by

$$s_t = f(u_t^1, u_t^2, \dots, u_t^n), \ \ \forall t \geq 0 \ ,$$

where $(u_t^i)_{t\geq 0}$ denotes the sequence generated by the i-th constituent LFSR and f is a function of n variables. In the case of a combination generator composed of n LFSR over $\mathbb{F}_q$, the combining function is a function from $\mathbf{F}_q^n$ into $\mathbb{F}_q$.

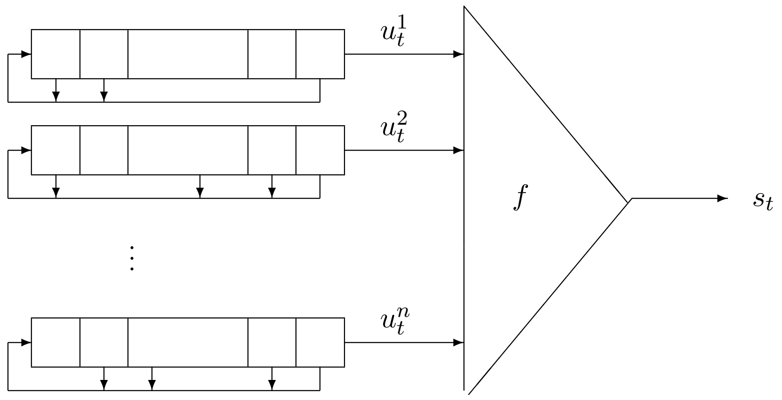


Figure 12.6: Combination generator.

The combining function f should obviously be *balanced*, i.e., its output should be uniformly distributed. The constituent LFSRs should be chosen to have primitive feedback polynomials for ensuring good statistical properties of their output sequences. The characteristics of the constituent LFSRs and the combining function are usually publicly known. The secret parameters are the initial states of the LFSRs, which are derived from the secret key of the cipher by a key-loading algorithm. When the feedback polynomials of the LFSR and the combining function are not known, the reconstruction attack presented in [CF01] enables to recover the complete description of the generator from the knowledge of a large segment of the ciphertext sequence.

Example 12.6. Geffe generator [Gef73] The generator proposed by Geffe [Gef73] is composed of three LFSRs of distinct lengths combined by the function

$$f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3 \ .$$

It is worth noticing that this function corresponds to an IF: if $x_2 = 0$, then $f(x_1, x_2, x_3) = x_3$ and if $x_2 = 1$, $f(x_1, x_2, x_3) = x_1$.

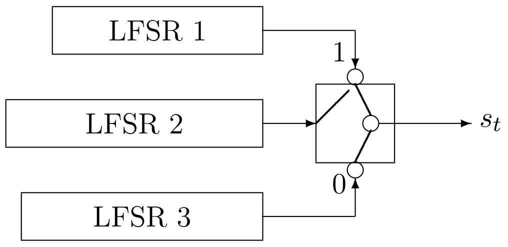


Figure 12.7: Geffe generator.

Linear complexity of the output sequence. The sequence produced by a combination generator is a linear recurring sequence. Its period and its linear complexity can be derived from those of the sequences generated by the constituent LFSRs and from the ANF of the combining function. Indeed, if we consider two linear recurring sequences $\mathbf{u}$ and $\mathbf{v}$ over $\mathbb{F}_q$ with linear complexities $\Lambda(\mathbf{u})$ and $\Lambda(\mathbf{v})$, we have the following properties:

- the linear complexity of the sequence $\mathbf{u} + \mathbf{v} = (u_t + v_t)_{t \geq 0}$ satisfies

  $$\Lambda(\mathbf{u} + \mathbf{v}) \leq \Lambda(\mathbf{u}) + \Lambda(\mathbf{v}) \ ,$$

  with equality if and only if the minimal polynomials of $\mathbf{u}$ and $\mathbf{v}$ are relatively prime. Moreover, in case of equality, the period of $\mathbf{u} + \mathbf{v}$ is the least common multiple of the periods of $\mathbf{u}$ and $\mathbf{v}$.

- the linear complexity of the sequence $\mathbf{uv} = (u_tv_t)_{t \geq 0}$ satisfies

  $$\Lambda(\mathbf{uv}) \leq \Lambda(\mathbf{u})\Lambda(\mathbf{v}) \ ,$$

  where equality holds if the minimal polynomials of $\mathbf{u}$ and $\mathbf{v}$ are primitive and if $\Lambda(\mathbf{u})$ and $\Lambda(\mathbf{v})$ are distinct and greater than 2. Other general sufficient conditions for $\Lambda(\mathbf{uv}) = \Lambda(\mathbf{u})\Lambda(\mathbf{v})$ can be found in [Her86, RS87, GN95].

These results lead to the following general proposition.

Proposition 12.7. *[RS87] Let us consider the combination generator composed of n binary LFSRs with primitive feedback polynomials which are combined by a Boolean function f. If all LFSR lengths $L_1, \dots, L_n$ are distinct and greater than 2 (and if the LFSR initializations differ from the all-zero state), the linear complexity of the output sequence* $\mathbf{s}$ *is equal to*

$$f(L_1, L_2, \dots, L_n)$$

where the algebraic normal form of f is evaluated over integers.

For instance, if four LFSRs of lengths $L_1, \dots, L_4$ satisfying the previous conditions are combined by the Boolean function $x_1x_2 + x_2x_3 + x_4$, the linear complexity of the resulting sequence is $L_1L_2 + L_2L_3 + L_4$. Similar results concerning the combination of LFSRs over $\mathbb{F}_q$ can be found in [Bry86, GN95]. For instance, the linear complexity of the sequence produced by the Geffe generator is $L_1L_2 + L_2L_3 + L_3$ where L_1, L_2 and L_3 denote the lengths of the constituent LFSRs.

A high linear complexity is obviously desirable for a keystream sequence since it ensures that Berlekamp-Massey algorithm becomes computationally infeasible. Thus, the combining function f should have a high *algebraic degree.*

A detailed analysis of the security of the combination generator, especially its resistance to correlation attacks, is provided in Section 12.5.

12.3.2 Filter generator

A *filter generator* is a keystream generator composed of a single LFSR whose content is filtered by a nonlinear function. More precisely, the output sequence of a filter generator corresponds to the output of a nonlinear function whose inputs are taken from some stages of the LFSR. If $(u_t)_{t\geq 0}$ denotes the sequence generated by the LFSR, the output sequence $(s_t)_{t\geq 0}$ of the filter generator is given by

$$s_t = f(u_{t+\gamma_1}, u_{t+\gamma_2}, \dots, u_{t+\gamma_n}), \ \ \forall t \geq 0$$

where f is a function of n variables, n is less than or equal to the LFSR length and $(\gamma_i)_{1\leq i\leq n}$ is a decreasing sequence of non-negative integers called the *tapping sequence.*

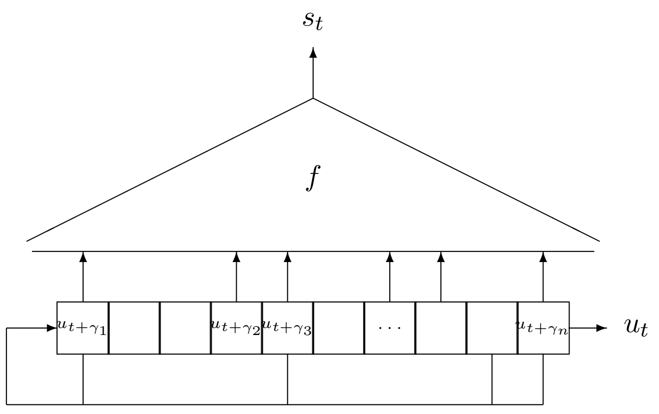


Figure 12.8: Filter generator.

In order to obtain a keystream sequence having good statistical properties, the filtering function f should be *balanced*, and the feedback polynomial of the LFSR should be chosen to be a primitive polynomial.

In a filter generator, the LFSR feedback polynomial, the filtering function and the tapping sequence are usually publicly known. The secret parameter is the initial state of the LFSR which is derived from the secret key of the cipher. The attack presented in [Sie85] enables to construct an equivalent keystream generator from the knowledge of a large segment of the ciphertext sequence when the LFSR feedback polynomial is the only known parameter (i.e., when the filtering function, the tapping sequence and the initial state are kept secret).

Any filter generator is equivalent to a particular combination generator, in the sense that both generators produce the same output sequence: an equivalent combination generator consists of n copies of the LFSR used in the filter generator with shifted initial states; the combining function corresponds to the filtering function.

Linear complexity of the output sequence. The output sequence $\mathbf{s}$ of a filter generator is a linear recurring sequence. Its linear complexity, $\Lambda(\mathbf{s})$, is related to the LFSR length and to the algebraic degree of the filtering function f. For a binary LFSR with a primitive feedback polynomial, we have

$$\Lambda(\mathbf{s}) \leq \sum_{i=0}^{d} \binom{L}{i}$$

where L denotes the LFSR length and d denotes the algebraic degree of f [Key76, Mas01]. The period of $\mathbf{s}$ divides $2^L - 1$. Moreover, if L is a large prime, $\Lambda(\mathbf{s})$ is at least $\binom{L}{d}$ for most filtering functions with algebraic degree d [Rue86]. To achieve a high linear complexity, the LFSR length L and the algebraic degree of the filtering function should be large enough. More precisely, the keystream length available to an attacker should always be much smaller than $\binom{L}{\deg(f)}$.

It is worth noticing that the lower bound on the linear complexity does not hold for all functions. For instance, some examples of filter generators with an LFSR of size L and a filtering function of high degree, but with linear complexity L only can be exhibited [RC10].

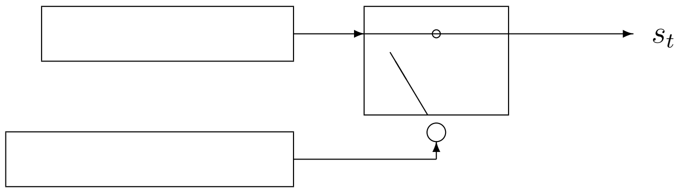
12.3.3 LFSRs with irregular clocking

Another technique for increasing the linear complexity of the produced keystream consists in considering one or several LFSRs, but some LFSR bits decide which LFSR to clock and how often. The most prominent example is the shrinking generator proposed in 1993 by Coppersmith, Krawczyk and Mansour [CKM94]. It is composed of two LFSRs, and the output of the second LFSR controls the clock of the first one. More precisely, if the second LFSR outputs 0, the output bit of the first one is discarded. This generator is depicted on Figure 12.9. Then, it can be proved that the linear complexity of the produced sequence is at least

$$L_A 2^{L_B - 2}$$

where L_A and L_B denote the linear complexities of the two constituent registers. The self-shrinking generator [MS95] and the alternating-step generator [Gün88] are two other examples of clock-controlled generators.

Figure 12.9: The shrinking generator.



12.4 Some widely-used LFSR-based generators

12.4.1 A5/1

A5/1 is the stream cipher used for encrypting over-the-air transmissions in the GSM standard. A5/1 is used in most European countries, whereas a weaker cipher, called A5/2, is used in other countries (a description of A5/2 and an attack can be found in [PFS00]). The description of A5/1 was first kept secret but its design has been reverse-engineered in 1999 by Briceno, Golberg and Wagner [BGW99].

A5/1 has a 64-bit secret key. A GSM conversation is transmitted as a sequence of 228-bit frames (114 bits in each direction) every 4.6 millisecond. Each frame is xored with a 228-bit sequence produced by the A5/1 running-key generator. The initial state of this generator depends on the 64-bit secret key, K, which is fixed during the conversation, and on a 22-bit public *frame number*, F.

Description of the running-key generator. The A5/1 running-key generator is composed of 3 LFSRs of lengths 19, 22 and 23. Their characteristic polynomials are $X^{19} + X^5 + X^2 + X + 1$, $X^{22} + X + 1$ and $X^{23} + X^{15} + X^2 + X + 1$. The internal state of the generator then consists of 64 bits only, which makes it vulnerable to Time-Memory-Data-Trade-off attacks.

For each frame transmission, the 3 LFSRs are first initialized to zero. Then, at time $t = 1, \ldots, 64$, the LFSRs are clocked, and the key bit K_t is xored to the feedback bit of each LFSR. For $t = 65, \ldots, 86$, the LFSRs are clocked in the same fashion, but the $(t - 64)$-th bit of the frame number is now xored to the feedback bits. This initialization phase is depicted on Figure 12.10.

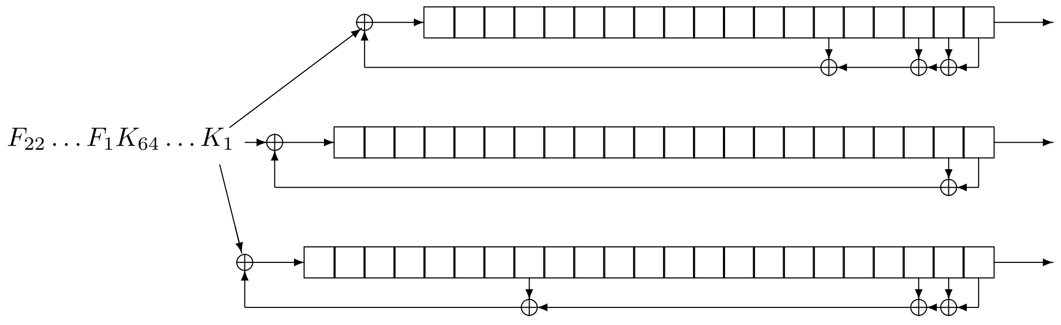


Figure 12.10: Initialization of the A5/1 running-key generator.

After these 86 cycles, the generator runs as depicted on Figure 12.11. Each LFSR has a clocking tap: tap 8 for the first LFSR, tap 10 for the second and the third ones (where the feedback tap corresponds to tap 0). At each unit of time, the majority value b of the 3 clocking bits is computed. A LFSR is clocked if and only if its clocking bit is equal to b. For instance, if the 3 clocking bits are equal to $(1, 0, 0)$, the majority value is 0. The second and third LFSRs are clocked, but not the first one. The output of the generator is then given by the xor of the outputs of the 3 LFSRs. After the 86 initialization cycles, 328 bits are generated with the previously described irregular clocking. The first 100 ones are discarded and the following 228 bits form the running-key.

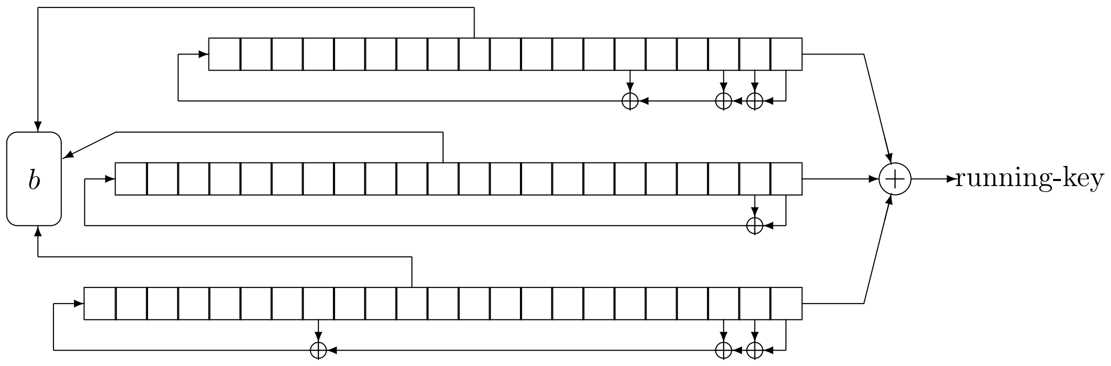


Figure 12.11: A5/1 running-key generator.

Attacks on A5/1. Several time-memory trade-off attacks have been proposed on A5/1 exploiting the small size of the secret key or of the internal state [BD00, BSW00]. They require the knowledge of a few seconds of conversation plaintext and run very fast. Even if they need a huge precomputation time and memory, an optimized version has been implemented in 2008: the group *The Hacker's Choice* has precomputed the huge look-up tables involved in the time-memory-trade-off attack. These tables have also been computed and then released in December 2009 by the *A5/1 cracking project* [NP09], and an improved implementation has been described in [KPPM12].

Another attack due to Ekdahl and Johansson [EJ03] exploits some weaknesses of the key initialization procedure. It has been later improved by Maximov, Johansson and Babbage [MJB04] and then by Barkan and Biham [BB06]. It requires a few minutes using 5-10 seconds of conversation plaintext without any notable precomputation and storage capacity. Most of these attacks can also be turned into ciphertext-only attacks in the context of GSM communications by exploiting the fact that error-correction is performed before encryption in the GSM transmissions [BBK08].

12.4.2 E0

E0 is the stream cipher used for ensuring the confidentiality of communications in the Bluetooth protocol for wireless short-range connectivity [Blu].

The keysize in E0 is 128 bits. More precisely, the number of key bytes, between 1 and 16 is negotiated between the two modules in the protocol, but the key is always extended to a 128-bit word by adding some redundancy when the effective number of key bits is less than

128. The IV has 64 bits which correspond to the 48-bit Bluetooth address, and a 26-bit master counter.

In the Bluetooth protocol, data are transmitted as frames of at most 2745 bits. Each frame is then encrypted by xoring the first output bits of the keystream generator. The generator is initialized with a secret key, which remains the same during the whole session, and an IV which is modified for each new frame.

Description of the running-key generator. E0 is a combination generator composed of four LFSRs, combined by a Boolean function with a four-bit internal memory. This generator can be seen as a variant of the summation generator [Rue86].

This generator is used at two different levels: it is first applied during the initialization phase for generating a 128-bit initial state from the secret key and the IV. Then, the same mechanism is used to produce the keystream from the initial state.

The four LFSRs are binary LFSRs of respective lengths $L_1 = 25$, $L_2 = 31$, $L_3 = 33$, $L_4 = 39$ (i.e., a total of 128 bits) with feedback polynomials

$$\begin{aligned} P_1(x) &= x^{25} + x^{20} + x^{12} + x^8 + 1 \\ P_2(x) &= x^{31} + x^{24} + x^{16} + x^{12} + 1 \\ P_3(x) &= x^{33} + x^{28} + x^{24} + x^4 + 1 \\ P_4(x) &= x^{39} + x^{36} + x^{28} + x^4 + 1 \ . \end{aligned}$$

Let x_t^i denote the output at time t of the i-th LFSR. Then, the 3-bit integer (between 0 and 4) corresponding to the sum of the outputs of the four LFSRs is computed:

$$y_t = x_t^1 + x_t^2 + x_t^3 + x_t^4 = 4y_t^2 + 2y_t^1 + y_t^0$$

where y_t^2, y_t^1, y_t^0 are binary values. The generator also includes some internal memory composed of two 2-bit words, denoted by c_t and c_{t-1} at time t. If $2c_t^1 + c_t^0 = c_t$ denotes the binary decomposition of c_t, then this 2-bit word is updated by

$$\begin{aligned} c_{t+1}^1 &= z_{t+1}^1 + c_t^1 + c_{t-1}^0 \bmod 2 \\ c_{t+1}^0 &= z_{t+1}^0 + c_t^0 + c_{t-1}^1 + c_{t-1}^0 \bmod 2 \end{aligned}$$

where z_t^1, z_t^0 is the binary decomposition of the integer

$$z_t = \left\lfloor \frac{y_{t-1} + c_{t-1}}{2} \right\rfloor \ .$$

This combination generator with memory is directly used for producing the keystream: the keystream at time t is equal to

$$s_t = y_t^0 + c_t^0 \bmod 2 \ .$$

The generator is initialized by the means of an additional level of the previously described system. The four LFSRs are first initialized with the 128-bit value

$$G_1(K_c) \text{ XOR } G_2(IV)$$

where G_1 and G_2 are two affine transformations with a 128-bit output. The two memory words are set to zero. Then, the generator is clocked 200 times, and another affine transformation G_3 is applied to the last 128 bits produced by the generator. The result of this operation then is then used as an initial state for the second-level generator, i.e., for the generator which outputs the keystream. The internal memory of this second-level generator is given by the memory of the first-level generator after the first 200 clocks.

Figure 12.12: E0 keystream generator.

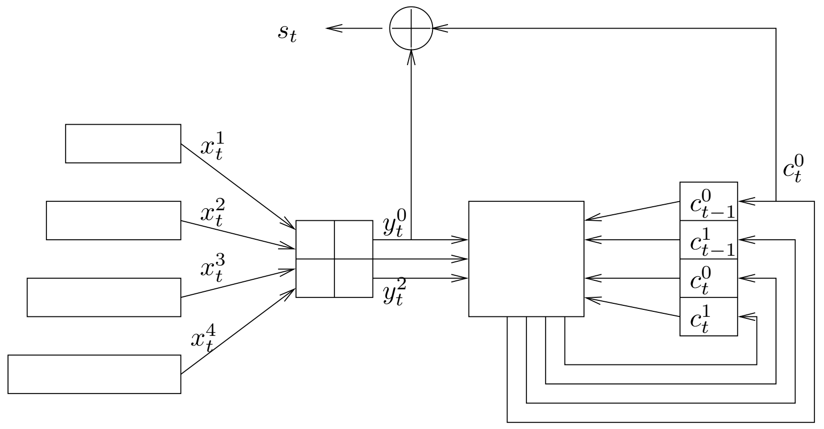
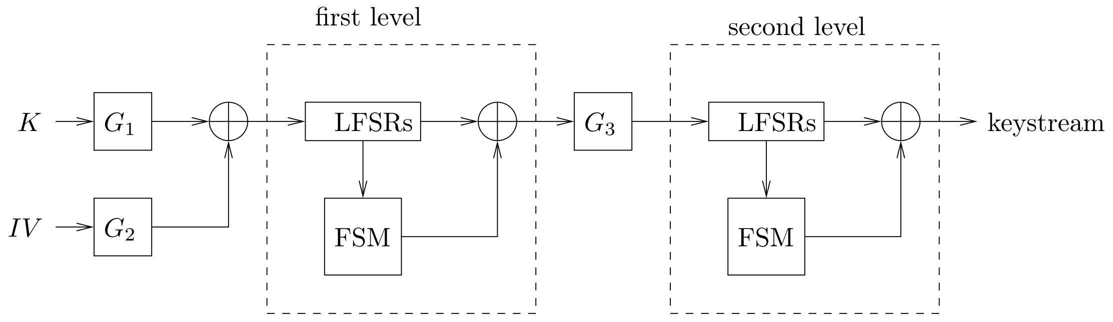


Figure 12.13: Initialization of E0 keystream generator.



Attacks on E0. The combination used in E0 is vulnerable to several attacks, including a linear attack [GBM02], some algebraic and fast algebraic attacks [Arm02, Cou03, HR04], and some fast correlation attacks [LV04b]. But all these attacks require the knowledge of a huge number of consecutive keystream bits generated from the same internal state, which is not the case in the Bluetooth protocol since the generator is resynchronized after 2745 bits.

Nevertheless, some sophisticated correlation attacks due to Yi Lu, Willi Meier and Serge Vaudenay [LV04a, LMV05] take the resynchronization process into account. The most efficient attack recovers the secret key from the knowledge of the first 24 bits of $2^{23.8}$ keystream frames. Its time complexity corresponds to 2^{38} operations. A better trade-off between the on-line computation and the precomputation has been obtained in [ZXF13]. All these results imply that the security level of the E0 stream cipher is very limited.

12.5 Correlation attacks on LFSR-based generators

The *correlation attack* was originally proposed by Siegenthaler in 1985 [Sie85] against the combination generator composed of n LFSRs of lengths $L_1, \dots, L_n$. The correlation attack is a divide-and-conquer technique: it aims at recovering the initial state of each constituent

LFSRs separately from the knowledge of some keystream bits (in a known plaintext attack). This attack requires $\sum_{i=1}^{n}\left(2^{L_i}-1\right)$ trials only, instead of the $\prod_{i=1}^{n}\left(2^{L_i}-1\right)$ tests required by an exhaustive search. A similar ciphertext only attack can also be mounted when there exists redundancy in the plaintext, as mentioned in [Sie85].

12.5.1 General principle

More generally, the correlation attack applies to any keystream generator as soon as the keystream is correlated to the output sequence $\boldsymbol{\sigma}$ of a "reduced generator" whose initial state depends on some key bits only. These key bits can be determined by recovering the initialization of $\boldsymbol{\sigma}$ as follows: an exhaustive search for the initialization of $\boldsymbol{\sigma}$ is performed, and the correct one is detected by computing the correlation between the corresponding sequence $\boldsymbol{\sigma}$ and the keystream. The following description concentrates on binary sequences.

More precisely, we assume as on Figure 12.14 that the internal state of the generator at time t can be divided into two parts, x_t and y_t of respective sizes ℓ and $(n-\ell)$, which are updated independently by two functions Φ_0 and Φ_1. Suppose wlog. that the attacker aims at recovering the first part of the initial state, x_0. The input vector of the filtering function can be decomposed into two parts, x and y according to the previous decomposition. Then, the attack can be mounted if there exists a function g depending on ℓ variables (i.e., depending on x only) whose output coincides with the output of f for more than half of the inputs. In other words, if there exists an ℓ-variable function g such that

$$p_g = \mathrm{Pr}_{X,Y}[f(X,Y)=g(X)] > \frac{1}{2} \ .$$

The existence of such a function g and its optimal choice is discussed in Section 12.5.5.

If $p_g > 1/2$, then the sequence $\boldsymbol{\sigma}(x_0)$ produced by the reduced generator with initial state x_0 and filtering function g is correlated to the keystream $\mathbf{s}$. Indeed, for all $t \geq 0$,

$$\Pr[s_t = \sigma_t] = p_g > \frac{1}{2} \ .$$

12.5.2 Correlation attacks as a decoding problem

It has been observed by Meier and Staffelbach [MS89] that the previously described situation corresponds to a classical problem in the context of error-correction. Indeed, if there exists a correlation between the keystream $\mathbf{s}$ and the output $\boldsymbol{\sigma}$ of the reduced generator, then the keystream subsequence $(s_t)_{t<N}$ can be seen as the result of the transmission of $(\sigma_t)_{t<N}$ through the binary symmetric channel with error probability $p = \Pr[s_t \neq \sigma_t] = 1 - p_g < 1/2$ (see Fig. 12.15). Moreover, if the transition function Φ_0 of the reduced generator is linear, then all bits of $\boldsymbol{\sigma}$ depend linearly on x_0. Therefore, $(\sigma_t)_{t<N}$ is a codeword of a linear code $\mathcal{C}$ of length N and dimension ℓ defined by Φ_0. Thus, recovering the initial state x_0 consists in decoding the running-key subsequence relatively to this linear code.

From Shannon's theorem, we know that $(\sigma_t)_{t<N}$ can only be decoded without errors if the transmission rate of the code is lower than the capacity of the channel. The involved channel is the binary symmetric channel with cross-over probability $p = (1-p_g)$. The capacity of the channel is defined by

$$C(p) = 1 + p\log_2 p + (1-p)\log_2(1-p) \ .$$

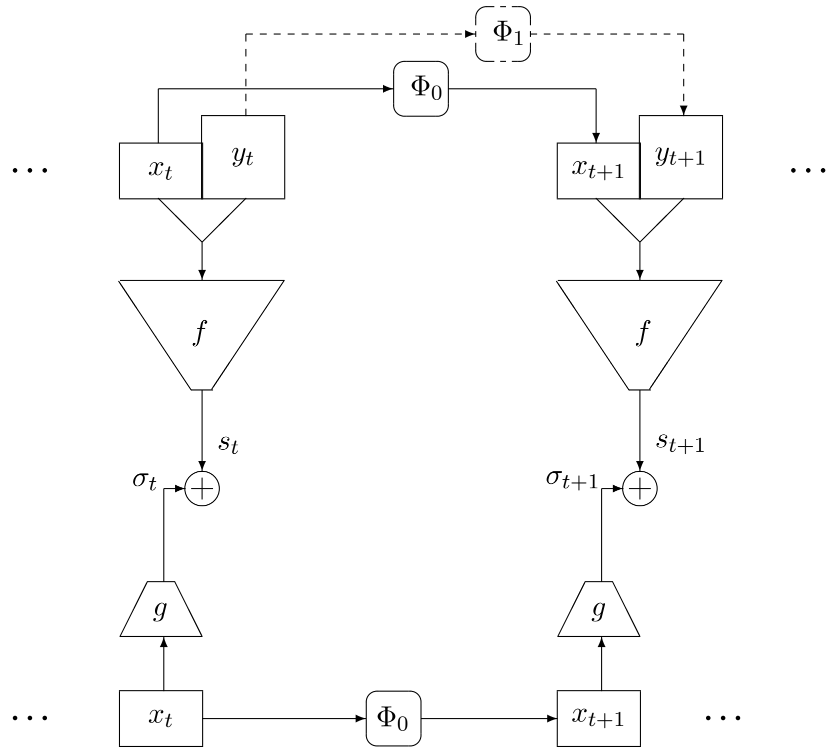


Figure 12.14: Model for the correlation attack.

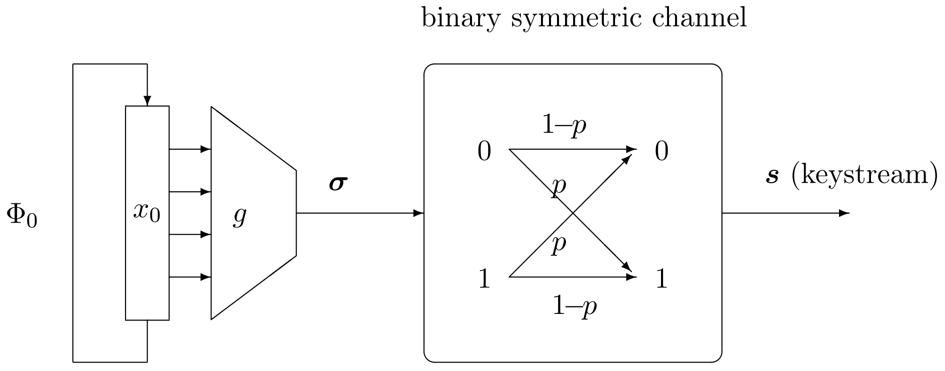


Figure 12.15: Correlation attacks on LFSR-based stream ciphers seen as a decoding problem.

In most situations, p_g is close to 1/2, i.e., $p_g = 1/2(1+\varepsilon)$ with $\varepsilon \ll 1$. Then, we get the following approximation

$$\begin{aligned} C\left(\frac{1}{2}(1-\varepsilon)\right) &= 1 - \frac{1}{2\ln 2}\left[(1-\varepsilon)\ln\left(\frac{1-\varepsilon}{2}\right) + (1+\varepsilon)\ln\left(\frac{1+\varepsilon}{2}\right)\right] \\ &\simeq 1 - \frac{1}{2\ln 2}\left[(1-\varepsilon)(-\varepsilon) + (1+\varepsilon)\varepsilon - 2\ln(2)\right] \\ &= 1 + \frac{\varepsilon^2}{2\ln 2} - 1 = \frac{\varepsilon^2}{\ln 2}\,. \end{aligned}$$

The transmission rate of the code is equal to ℓ/N where ℓ is the size of the targeted part of the initial state x_0 and N is the number of known keystream bits. Then, we deduce from Shannon's theorem that the attack requires the knowledge of

$$N \geq \frac{\ell \ln 2}{\varepsilon^2} \text{ keystream bits.} \tag{12.1}$$

Generator matrix for an LFSR-code. In the particular case where the internal state x_t of the reduced generator is updated by an LFSR of length ℓ, the generator matrix of the underlying code $\mathcal{C}$ can be easily computed from the characteristic polynomial of the LFSR.

Proposition 12.8. *Let $(\sigma_t)_{t\geq 0}$ be a sequence produced by an LFSR of length ℓ with characteristic polynomial $P^\star$. Then, for any $N \geq \ell$, the $\ell \times N$ matrix G such that*

$$(\sigma_0, \ldots, \sigma_{N-1}) = (\sigma_0, \ldots, \sigma_{\ell-1})G$$

is the matrix whose t-th column, $0 \leq t < N$ corresponds to the coefficients of the polynomial $X^t \bmod P^\star(X)$.

Proof. Let x_0 denote the initial state of the reduced generator producing $(\sigma_t)_{t\geq 0}$. From Proposition 12.4, we know that, for any $t \geq 0$,

$$\sigma_t = \mathsf{Tr}(\alpha^t x_0) = \mathsf{Tr}\left(\alpha^t \left(\sum_{i=0}^{\ell-1} \sigma_i \beta_i\right)\right) .$$

where α is a root of $P^\star$ and $\{\beta_0, \ldots, \beta_{\ell-1}\}$ is the dual basis of $\{1, \alpha, \ldots, \alpha^{\ell-1}\}$. Let $X^t \bmod P^\star(X) = \sum_{j=0}^{\ell-1} d_j X^j$. Then, by definition of α, we have that $\alpha^t = \sum_{j=0}^{\ell-1} d_j \alpha^j$. It follows that

$$\begin{aligned}\sigma_t &= \mathsf{Tr}\left(\left(\sum_{j=0}^{\ell-1} d_j \alpha^j\right)\left(\sum_{i=0}^{\ell-1} \sigma_i \beta_i\right)\right) \\ &= \sum_{j=0}^{\ell-1}\sum_{i=0}^{\ell-1} d_j \sigma_i \mathsf{Tr}(\alpha^j \beta_i) \\ &= \sum_{i=0}^{\ell-1} d_i \sigma_i\end{aligned}$$

by definition of the dual basis. This equivalently means that the t-th column of G corresponds to the vector $(d_0, \ldots, d_{\ell-1})$. ◇

12.5.3 Maximum-likelihood decoding for correlation attacks

The original correlation attack presented by Siegenthaler [Sie85] recovers the initial state x_0 by performing a statistical test on the correlation between the observed keystream $\mathbf{s}$ and the output of the reduced generator $\boldsymbol{\sigma}(x_0)$ for all possible values of x_0. This exactly corresponds to a maximum-likelihood decoding procedure. Here, we describe the attack in terms of statistical test as in the original paper.

Proposition 12.9. *Let $(s_t)_{t\geq 0}$ and $(\sigma)_{t\geq 0}$ be two sequences such that $\Pr[s_t \neq \sigma_t] = p$ for all $t \geq 0$. Then, the correlation between these two sequences computed over N bits,*

$$C = \frac{1}{N}\sum_{t=0}^{N-1}(-1)^{s_t+\sigma_t}$$

is a random variable which follows a Gaussian distribution with mean $(1-2p)$ and variance $4p(1-p)/N$.

Proof. We write

$$C = \frac{1}{N}\sum_{t=0}^{N-1}(1-2(s_t \oplus \sigma_t)) = 1 - \frac{2}{N}\sum_{t=0}^{N-1}(s_t \oplus \sigma_t)\ .$$

The N variables $(s_t \oplus \sigma_t)_{0\leq t<N}$ are independent and follow a Bernoulli distribution with mean p and variance $p(1-p)$. Then, from the central limit theorem, we get that, for large N, the distribution of

$$\frac{1}{N}\sum_{t=0}^{N-1}(s_t \oplus \sigma_t)$$

is close to the normal distribution with mean p and variance $p(1-p)/N$. Thus, C follows the normal distribution with mean $(1-2p)$ and variance $4p(1-p)/N$. ◇

In particular, the previous proposition implies that, if the two sequences are uncorrelated, i.e., if $p = 1/2$, then C follows a normal distribution with mean 0 and variance $1/N$.

If the two sequences $(\sigma_t)_{t\geq 0}$ and $(s_t)_{t\geq 0}$ are correlated, then all possible values for the initial state x_0 of $\boldsymbol{\sigma}$ are examined. The correct value for x_0 can be detected by a classical hypothesis testing (see Section 11.3.3 in Chapter 11). The corresponding algorithm is described in Algorithm 8. The value of the correlation is compared to some threshold T, whose value is

Algorithm 8 Original correlation attack.

```
Input. $s_0 s_1 \dots s_{N-1}$, $N$ keystream bits and $p = \Pr[s_t \neq \sigma_t] < 1/2$.
Output. $\sigma_0 \dots \sigma_{\ell-1}$, the initial state of $\boldsymbol{\sigma}$.
Compute the threshold $T$ with (12.2)
for all $\sigma_0, \dots, \sigma_{\ell-1}$ do
  Generate the first $N$ bits of the sequence $\boldsymbol{\sigma}$.
  Compute the correlation between $s_0 s_1 \dots s_{N-1}$ and $\sigma_0\sigma_1 \dots \sigma_{N-1}$:
      $C \leftarrow \frac{1}{N}\sum_{t=0}^{N-1}(-1)^{s_t+u_t \bmod 2}$
  if $C > T$ then
    return $\sigma_0 \dots \sigma_{\ell-1}$
  end if
end for
```

chosen as follows in order to minimize the error probability. If the initial state of $\boldsymbol{\sigma}$ is correct

(Hypothesis $\mathcal{H}_1$), then C follows a normal distribution with mean $(1-2p)$ and variance $4p(1-p)/N$. This means that

$$\Pr[C = x] = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-(1-2p))^2}{2\sigma^2}\right)$$

with $\sigma^2 = 4p(1-p)/N$. Otherwise (Hypothesis $\mathcal{H}_0$), we have

$$\Pr[C = x] = \sqrt{\frac{N}{2\pi}} \exp\left(-\frac{Nx^2}{2}\right) .$$

For instance, these two distributions for $N = 50$ and $p = 1/4$ are plot on Figure 12.16.

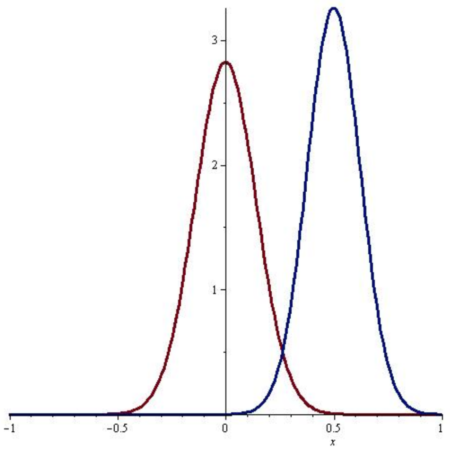


Figure 12.16: Distributions of the correlation for $N = 50$ and $p = 1/4$. The red curve corresponds to Hypothesis $\mathcal{H}_0$ (no correlation), while the blue curve corresponds to $\mathcal{H}_1$ (correct initial state).

Then, the false-alarm probability is

$$\begin{aligned} P_f &= \sqrt{\frac{N}{2\pi}} \int_T^{+\infty} \exp\left(-\frac{Nx^2}{2}\right) dx = \frac{1}{\sqrt{2\pi}} \int_{T\sqrt{N}}^{+\infty} \exp\left(-\frac{y^2}{2}\right) dy \\ &= 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{T\sqrt{N}} \exp\left(-\frac{y^2}{2}\right) dy = 1 - \Phi(T\sqrt{N}) \end{aligned}$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution, i.e.,

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{y^2}{2}\right) dy .$$

It is worth noticing that Φ can also be expressed by the means of the Gauss error function

$$\Phi(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right] .$$

Similarly, the non-detection probability is given by

$$\begin{aligned} P_n &= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{T} \exp\left(-\frac{(x-(1-2p))^2}{2\sigma^2}\right) dx \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{T-(1-2p)}{\sigma}} \exp\left(-\frac{y^2}{2}\right) dy = \Phi\left(\frac{T-(1-2p)}{\sigma}\right) . \end{aligned}$$

We deduce that, for obtaining a given value of P_n, we need to choose the threshold such that

$$\frac{T-(1-2p)}{\sigma} = \Phi^{-1}(P_n)$$

or equivalently

$$T = (1-2p) + 2\Phi^{-1}(P_n)\sqrt{\frac{p(1-p)}{N}} . \tag{12.2}$$

Moreover, we have

$$T\sqrt{N} = \Phi^{-1}(1-P_f)$$

implying that

$$\sqrt{N} = \frac{\Phi^{-1}(1-P_f) - 2\Phi^{-1}(P_n)\sqrt{p(1-p)}}{1-2p} .$$

Typical suitable values for P_f and P_n are $P_n = 1.3 \times 10^{-3}$, i.e., $\Phi^{-1}(P_n) = -3$ and $P_f = 2^{-\ell}$, i.e., $\Phi^{-1}(1-P_f) \simeq \sqrt{\ell}$. With these values, we get that

$$N = \mathcal{O}\left(\frac{\ell}{\varepsilon^2}\right) \text{ where } p = \frac{1}{2}(1-\varepsilon) .$$

It is worth noticing that we recover the data complexity deduced from Shannon's theorem and the value of the capacity of the binary symmetric channel. The time complexity of Algorithm 8 is then

$$\mathsf{Time} = 2^\ell N = \mathcal{O}\left(\frac{\ell 2^\ell}{\varepsilon^2}\right)$$

Maximum-Likelihood decoding with an FFT. The time complexity of the previous algorithm is prohibitive in most situations, except for very small values of ℓ. However, it can be significantly reduced when the transition function Φ_0 is linear (i.e., if the underlying code $\mathcal{C}$ is a linear code). Indeed, maximum-likelihood decoding consists in computing the distance between the N-bit received word (i.e., the keystream in our case) and the 2^ℓ codewords in $\mathcal{C}$. When $\mathcal{C}$ is a linear code, this computation boils down to evaluating the Fourier transform of the ternary function F from $\mathbb{F}_2^\ell$ into $\{-1, 0, 1\}$ defined by

$$\begin{cases} F(g_t) &= (-1)^{s_t} \quad \text{for all } 0 \le t < N \\ F(x) &= \quad 0 \quad\ \ \text{for all } x \notin \{g_t,\ 0 \le t < N\} \end{cases}$$

where g_t is the t-th column of the generator matrix of $\mathcal{C}$. Indeed, the correlation between the keystream and the sequence $\boldsymbol{\sigma}$ produced from a given initial state x_0 corresponds to the

Fourier transform of F at point x_0:

$$\begin{aligned} C(\mathbf{s}, \boldsymbol{\sigma}(x_0)) &= \frac{1}{N}\sum_{t=0}^{N-1}(-1)^{s_t+\sigma_t} \\ &= \frac{1}{N}\sum_{t=0}^{N-1}(-1)^{s_t+x_0\cdot g_t} \\ &= \sum_{x\in\mathbb{F}_2^\ell} F(x)(-1)^{x_0\cdot x} = \widehat{F}(x_0)\ . \end{aligned}$$

Therefore, when the code length N (i.e., the data complexity of the attack) is large, the time complexity can be reduced to $2^\ell\ell$ with a Fast Fourier Transform algorithm [CJM02, Lu06]. However, ML-decoding remains usually infeasible when the size of targeted part of the initial state, x_0, is not very small, typically when $\ell \geq 80$. In this situation, the attack can only be mounted by using some decoding algorithms faster than ML-decoding. The counterpart is of course that such algorithms require a longer keystream segment than the optimal value corresponding to Shannon's theorem (see Equation (12.1)). These variants of the original attack are usually named *fast correlation attacks*. They apply when the transition function of the reduced generator is linear.

12.5.4 Iterative decoding with based on low-density parity-checks

The idea of using an iterative decoding procedure based on low-density parity check (LDPC) equations comes back to Meier and Staffelbach [MS89], even if the algorithm they originally proposed was less efficient than the classical decoding algorithm for LDPC introduced by Gallager [Gal62]. The general principle of this decoding procedure consists in searching for a large number of parity-check relations for $\mathcal{C}$ having a low weight w. By the means of these relations, $\mathcal{C}$ can be seen as an LDPC code, for which there exist efficient decoding algorithms.

Low-weight parity-check equations Low-weight parity-check equations, i.e., with a small number of terms, for the involved code $\mathcal{C}$ can be exploited in fast correlation, but also in many other cryptanalytic techniques as they usually provide an efficient distinguishing procedure. A parity-check equation for the code $\mathcal{C}$ corresponds to a set of columns in the generator matrix which sum to zero. Equivalently, it can be seen as a codeword of the dual code $\mathcal{C}^\perp$.

If the transition function of the reduced generator corresponds to an LFSR, we deduce from the structure of the LFSR code, that its parity-check equations have the following form

$$\sigma_t + \sigma_{t+\tau_1} + \ldots + \sigma_{t+\tau_{w-1}} = 0, \text{ for all } t \geq 0$$

and are satisfied for any sequence $\boldsymbol{\sigma}$ generated by the LFSR. We know from Proposition 12.8 that the t-th column of the generator matrix of the LFSR code corresponds to the coefficients of $X^t \bmod P^\star(X)$, where $P^\star$ is the characteristic polynomial of the LFSR. Therefore, there is a bijection between such a parity-check equations and the polynomials

$$1 + X^{\tau_1} + \ldots + X^{\tau_{w-1}}$$

which are multiples of the LFSR characteristic polynomial $P^\star$. The weight of such an equation is the number of its terms, or equivalently the Hamming weight of the corresponding word in the dual code.

Degree of parity-check equations. The value τ_{w-1} (i.e., the degree of the corresponding polynomial) is also called the degree of the parity-check equation. For estimating the complexity of the attack, we need to determine the number of parity-check equations with a given weight w and with a degree less than some value d, when d varies. Usually, we assume that, for a given w, when d is large enough, the values taken by the polynomials of weight w of the form $(1 + X^{\tau_1} + X^{\tau_2} + \ldots + X^{\tau_{w-1}}) \bmod P^\star(X)$ for all $0 < \tau_1 < \tau_2 < \ldots < \tau_{w-1} < d$ are uniformly distributed in the set of all polynomials modulo $P^\star$ Under this hypothesis, the number of parity-check equations of weight w and of degree at most d is roughly

$$m_w(d) = \frac{\binom{d}{w-1}}{2^{\deg P^\star}} \simeq \frac{d^{w-1}}{(w-1)!\, 2^{\deg P^\star}} \ . \tag{12.3}$$

However, the previous hypothesis does not hold in all situations. It would imply for instance that, for any $P^\star$, the number of codewords of weight exactly w in the dual of the LFSR code $\mathcal{C}^\perp$ of length $N = 2^{\deg P^\star} - 1$ is always close to

$$\frac{N^{w-1}}{w!} \ . \tag{12.4}$$

But clearly, this number highly depends on the algebraic structure of the characteristic polynomial. When $P^\star$ is a primitive polynomial, the corresponding LFSR code of length $N = 2^{\deg P^\star} - 1$ is equivalent to the *simplex code* (i.e., to the shortened first-order Reed-Muller code) [MS77, Page 30]. Its dual is then equivalent to the Hamming code of length N. In this case it can be checked that Formula (12.4) provides a good approximation of the weight distribution [MS77, Page 129]. More generally, when $P^\star$ is a randomly chosen primitive polynomial, simulation results show that (12.3) is a good estimation of the number of parity-check relations of weight w and degree at most d, when d is not too small. It is worth noticing that this holds even if $P^\star$ itself has weight w. Similarly, the minimal degree for a polynomial of weight w multiple of $P^\star$ is close to

$$(w-1)!^{\frac{1}{w-1}} 2^{\frac{\deg P^\star}{w-1}} \ .$$

The situation is much more complicated when $P^\star$ is the product of two primitive polynomials. Then, the previous estimation remains reasonable when the degrees of the polynomials involved in the product are coprime, but does not hold in general. For instance, it is possible to construct some examples where $P^\star$ is the product of two primitive polynomials having the same degree and such that $P^\star$ has no multiple of degree 3 (see e.g. [CTZ01]).

Computing low-degree parity-check equations. There are several algorithms of computing parity-check equations of weight w associated to a polynomial P. The simplest one is described in Algorithm 9. More sophisticated ones are mentioned in [Jou09, Pages 384-386] The corresponding time and memory complexity are

$$\mathsf{Time} = \mathcal{O}(d^{w-v-1}) \text{ and } \mathsf{Memory} = \mathcal{O}(d^v) \ .$$

An interesting trade-off is then obtained for $v = \lfloor \frac{w-1}{2} \rfloor$, leading to

$$\mathsf{Time} = \mathcal{O}(d^{\lceil \frac{w-1}{2} \rceil}) \text{ and } \mathsf{Memory} = \mathcal{O}(d^{\lfloor \frac{w-1}{2} \rfloor}) \ .$$

When $w \geq 5$, an improved variant of this algorithm due to Chose, Joux and Mitton [CJM02] allows to decrease the memory requirement to $\mathsf{Memory} = \mathcal{O}(d^{\lfloor \frac{w}{4} \rfloor})$ with the same time complexity.

Algorithm 9 Algorithm for computing multiples of P of weight w and degree at most d.

```
for each set $(i_1, \ldots, i_v)$ of $v$ elements of $\{1, \cdots, d\}$ do
  $q(X) \leftarrow X^{i_1} + \ldots + X^{i_v} \bmod P(X)$
  store $q$ in a table $T$ such that $T[a] = \{(i_1, \ldots, i_v) : q(X) = a\}$.
end for
for each set $(j_1, \ldots, j_{w-v-1})$ of $w - v - 1$ elements of $\{1, \cdots, d\}$ do
  $A \leftarrow 1 + X^{j_1} + \ldots + X^{w-v-1} \bmod P(X)$
  for all $(i_1, \ldots, i_v) \in T[A]$ do
    return $1 + X^{i_1} + \ldots + X^{i_v} + X^{j_1} + \ldots + X^{w-v-1}$
  end for
end for
```

Iterative decoding of LDPC. A detailed comparison between several iterative decoding algorithms can be found in [Lev04]. For instance, the slightly impaired but simple version of Gallager's algorithm proposed in [CT00] consists in iteratively updating the log-likelihood ratio at each bit position, i.e., the quantity

$$\log\left(\frac{\Pr[\sigma_t = 0]}{\Pr[\sigma_t = 1]}\right) .$$

It is described by Algorithm 10.

Algorithm 10 Iterative decoding algorithm for fast correlation attacks.

```
Input: $s_0 s_1 \ldots s_{N-1}$, $N$ keystream bits and $p = \Pr[s_t \neq \sigma_t] < 1/2$.
/* Initialization */
for all $t$ from 0 to $N - 1$ do
  $L[t] \leftarrow \log(\frac{1-p}{p})$
end for
repeat
  for all $t$ from 0 to $N - 1$ do
    $L'[t] \leftarrow (-1)^{s_t} L[t]$
    for all parity-check equations involving Position $t$, $\sigma_t = \sum_{j \in J} \sigma_j$ do
        $$L'[t] \leftarrow L'[t] + (-1)^{\sum_{j \in J} s_j} \min_{j \in J}(L[j])$$
    end for
  end for
until convergence
for all $t$ from 0 to $N - 1$ do
  if $L'[t] < 0$ then
    $\sigma_t \leftarrow 0$
  else
    $\sigma_t \leftarrow 1$
  end if
end for
return $(\sigma)_{t \geq 0}$
```

The complexity of this algorithm highly depends on the number m_w of parity-check equations of weight w required for convergence. The simulation results presented in [CT00] show that the minimal number of equations can be estimated by

$$m_w = \frac{2\ln 2}{\varepsilon^{2w-4}} ,$$

where $\varepsilon = 1 - 2p$ and p is the error-probability. By combining this result with the expected number of parity-check equations of weight w and degree at most N given by (12.3), we deduce that the required data complexity is

$$N = \left(\frac{1}{\varepsilon}\right)^{\frac{2(w-2)}{w-1}} 2^{\frac{\ell}{w-1}}$$

and the corresponding time complexity

$$\mathsf{Time} = \left(\frac{1}{\varepsilon}\right)^{\frac{2w(w-2)}{w-1}} 2^{\frac{\ell}{w-1}} .$$

12.5.5 Existence of an approximation with fewer variables

As explained in Section 12.5.1, a (fast) correlation attack can be mounted only when the Boolean function f of n variables involved in the keystream generator can be approximated by a function g depending on fewer variables. Typically, in a combination generator composed of n LFSRs, if the combining function f can be approximated by a function g of $\ell < n$ variables, then the attack involves the initial states of only ℓ out of the n constituent LFSRs.

Correlation-immunity order. A natural counter-measure to avoid correlation attacks consists then in using as building-blocks a *correlation-immune function* in the sense of the following definition.

Definition 12.10 (Correlation-immunity [Sie84]). *A Boolean function f is* t-th order correlation-immune *if the probability distribution of its output is unaltered when any t input variables are fixed. Balanced t-th order correlation-immune functions are called* t-resilient.

Note that a t-th order correlation-immune function is k-th order correlation-immune for any $k \leq t$. The correlation-immunity order of a function f then refers to the highest integer t such that f is t-th order correlation-immune.

In a combination generator, the correlation-immunity order t of the combining function determines the minimal number of LFSRs which must be considered together in a correlation attack. Indeed, the keystream produced by the combination generator is then independent of any set of t constituent LFSRs. The smallest number of LFSRs involved in a correlation attack is therefore $t + 1$. But the correlation-immunity order of a Boolean function cannot be chosen as high as we wish: it is limited by the algebraic degree of the function as shown in the next proposition.

Proposition 12.11. *[Sie84] Let f be a Boolean function of n variables. Then its correlation-immunity order t satisfies*

$$t + \deg f \leq n .$$

Moreover, if f is balanced and $t < n - 1$, then

$$t + \deg f \leq n - 1 .$$

Proof. Let $u \in \mathbb{F}_2^n$ such that $wt(u) \geq n-t$. We compute the coefficients of the ANF of f with Theorem 10.3. For $L = \{1, \ldots, n\} \setminus \mathsf{Supp}(u)$, we have

$$a_u = \sum_{x_i=0, i \in L} f(x_1, \ldots, x_n) \bmod 2 = 2^{-(n-wt(u))} wt(f) \bmod 2$$

where the last equality comes from the fact that the probability distribution of the output of f is unchanger when the $(n - wt(u)) \leq t$ variables defined by L are set to 0. If f is balanced, i.e., $wt(f) = 2^{n-1}$, we get that, for all u with $wt(u) \geq n-t$,

$$a_u = 2^{wt(u)-1} \bmod 2 = 0$$

since $wt(u) - 1 \geq n-t-1 > 0$. In other words, all coefficients of degree greater than or equal to $(n-t)$ in the ANF of f are equal to zero, which means than $\deg f < n-t$.

If f is not balanced, we select some word $u^\star$ of weight $(n-t)$. Then,

$$a_{u^\star} = 2^{-t} wt(f) \bmod 2$$

implying that $wt(f) = 2^t a_{u^\star} + \Lambda 2^{t+1}$ for some integer Λ. Now, for any u of weight $(n-t+w)$ with $w \geq 1$, we get

$$a_u = 2^{-t+w} wt(f) = 2^w a_{u^\star} + \Lambda 2^{w+1} = 0 \bmod 2\ .$$

This means that all coefficients in the ANF of degree greater than or equal to $(n-t+1)$ vanish, i.e. $\deg f \leq n-t$. ◇

Approximation of a function by a function of $(t+1)$ variables. Since the combining function in a combination generator is balanced and nonlinear, its correlation-immunity order is at most $(n-3)$. It follows that we can always apply a correlation attack by considering at most $\ell = (n-2)$ LFSRs together. We now show how to determine the best approximation of f by a Boolean function g depending on a fixed subset of ℓ variables.

Proposition 12.12. *[Can02, Zha00] Let f be a function of n variables and let L be a subset of $\{1, \ldots, n\}$ of cardinality ℓ, $L = \{i_1, \ldots, i_\ell\}$. The highest possible value over all ℓ-variable functions g of*

$$p_g = \Pr_X[f(X_1, \ldots, X_n) = g(X_{i_1}, \ldots, X_{i_\ell})]$$

is achieved if and only if

$$\begin{cases} g(x) = 1 & \textit{if } \Pr_Y[f(X,Y) = 1 | X = x] > \frac{1}{2} \\ g(x) = 0 & \textit{if } \Pr_Y[f(X,Y) = 1 | X = x] < \frac{1}{2} \end{cases}$$

It follows that the maximum of p_g is

$$\max_g p_g = \frac{1}{2} + \frac{1}{2^\ell} \sum_{x \in \mathbf{F}_2^\ell} \left| \frac{1}{2} - \Pr_Y[f(X,Y) = 1 | X = x] \right|\ .$$

Proof. Let us decompose the n input variables of f as (X, Y) where X corresponds to the ℓ variables of index $i_1, \ldots, i_\ell$, and Y to the $(n-\ell)$ remaining variables. Let $p_L(x)$, $x \in \mathbb{F}_2^\ell$, denote the probability $p_L(x) = \Pr_Y[f(X,Y) = 1 | X = x]$. In other words, $p_L(x)$ is the probability that f outputs 1 when the ℓ inputs in positions L are fixed and equal to x. For any ℓ-variable g we have

$$\begin{aligned}
p_g &= \mathrm{Pr}_{X,Y}[f(X,Y)=g(X)] \\
&= 2^{-\ell}\left(\sum_{x\in g^{-1}(0)} \mathrm{Pr}[f(X,Y)=0|X=x] + \sum_{x\in g^{-1}(1)} \mathrm{Pr}[f(X,Y)=1|X=x]\right) \\
&= 2^{-\ell}\sum_{x\in g^{-1}(0)}(1-p_L(x)) + 2^{-\ell}\sum_{x\in g^{-1}(1)} p_L(x) \\
&= 2^{-\ell}|g^{-1}(0)| - 2^{-\ell}\sum_{x\in\mathbb{F}_2^\ell}(-1)^{g(x)}p_L(x) \\
&= \frac{1}{2} + 2^{-\ell-1}\sum_{x\in\mathbb{F}_2^\ell}(-1)^{g(x)} - 2^{-\ell}\sum_{x\in\mathbb{F}_2^\ell}(-1)^{g(x)}p_L(x) \\
&= \frac{1}{2} + 2^{-\ell}\sum_{x\in\mathbb{F}_2^\ell}(-1)^{g(x)}\left(\frac{1}{2}-p_L(x)\right) .
\end{aligned}$$

It follows that p_g is maximal if and only if all terms in the sum are greater than or equal to zero, or equivalently

$$g(x) = \begin{cases} 0 & \text{if } p_L(x) < 1/2, \\ 1 & \text{if } p_L(x) > 1/2 \ . \end{cases} \tag{12.5}$$

Note that the value of $g(x)$ can be arbitrarily chosen when $p_L(x) = \frac{1}{2}$. The maximal value of p_g directly follows. ◇

The previous proposition obviously implies that the maximal value of p_g is $1/2$ if ℓ is less than or equal to the correlation-immunity order of f since all $\mathrm{Pr}_Y[f(X,Y)=1|X=x]=1/2$ in this case. Another consequence is that, for $\ell = t+1$ where t is the correlation-immunity order of f, the maximal value of p_g is achieved by an affine function.

Theorem 12.13. *[CT00] Let f be a t-resilient function of n variables and let L be a subset of $\{1,\dots,n\}$ of cardinality $(t+1)$, $L=\{i_1,\dots,i_{t+1}\}$. The highest possible value over all $(t+1)$-variable functions g of*

$$p_g = \mathrm{Pr}_X[f(X_1,\dots,X_n) = g(X_{i_1},\dots,X_{i_{t+1}})]$$

is achieved by the affine function

$$g(x_{i_1},\dots,x_{i_{t+1}}) = \sum_{i\in L} x_i + \varepsilon$$

with $\varepsilon\in\mathbb{F}_2$.

Proof. We here use the same notation as in the proof of the previous proposition. For any $j\in L$, e_j denotes the $(t+1)$-bit vector whose all coordinates are zero except the j-th one. We first prove that, for any $x\in\mathbb{F}_2^{t+1}$ and any $j\in L$, $p_L(x)+p_L(x+e_j)=1$. Indeed, we have

$$\begin{aligned}
p_L(x)+p_L(x+e_j) &= \mathrm{Pr}[f(X,Y)=1|X=x] + \mathrm{Pr}[f(X,Y)=1|X=x+e_j] \\
&= 2\,(\mathrm{Pr}[f(X,Y)=1|\forall i\in L, X_i=x_i]\mathrm{Pr}[X_j=x_j]+ \\
&\quad \mathrm{Pr}[f(X,Y)=1|\forall i\in L\setminus\{j\}, X_i=x_i, X_j\neq x_j]\mathrm{Pr}[X_j\neq x_j]) \\
&= 2\mathrm{Pr}[f(X,Y)=1|\forall i\in L\setminus\{j\}, X_i=x_i] = 1
\end{aligned}$$

where the last equality comes from the fact that f is t-resilient and that the set $L \setminus \{j\}$ has cardinality t. Let g be a $(t+1)$-variable function such that p_g is minimal. Since $p_L(x) + p_L(x+e_j) = 1$ for any $x \in \mathbb{F}_2^{t+1}$ and for any $j \in L$, Condition (12.5) implies that

$$g(x) + g(x+e_j) = 1$$

when $p_L(x) \neq \frac{1}{2}$. Moreover, we can assume that this relation is satisfied for any $x \in \mathbb{F}_2^{t+1}$, because the value of $g(x)$ can be arbitrarily chosen when $p_L(x) = \frac{1}{2}$. It follows that, for any $x \in \mathbb{F}_2^{t+1}$,

$$g(x) = g(0) + \sum_{i \in L} x_i \ .$$

This probability is then maximized when $(-1)^{g(0)}$ and $\left(\Pr_X[f(X) = \sum_{i\in L} X_i] - \frac{1}{2}\right)$ have the same sign. $\diamond$

This result is of great interest because the degree of the approximation g affects the linear complexity of the reduced generator. Indeed, $\boldsymbol{\sigma}$ is produced by a smaller combining generator composed of $\ell = t+1$ (or more) LFSRs combined by g. Then, we know from Section 12.3.1 that the linear complexity of $\boldsymbol{\sigma}$ depends on the degree of g. The previous result then shows that, in a fast correlation attack involving the smallest number of LFSRs, i.e. $(t+1)$, the same combining function g minimizes both the error probability $(1-p_g)$ and the linear complexity of $\boldsymbol{\sigma}$. In this context, since g has degree 1, the minimal polynomial of σ is the least common multiple of the minimal polynomials $P_i^\star$ of the considered LFSRs [Zie59]. In most practical situations, we have $P^\star = \prod_{j=1}^{t+1} P_{i_j}^\star$ since all the involved feedback polynomials are primitive. The N-bit keystream $(s_t)_{t<N}$ can then be seen as the result of the transmission through a noisy channel of a codeword of a linear code of dimension $\sum_{j=1}^{t+1} L_{i_j}$. The previously described decoding algorithm then applies and the analysis of its complexity tends to show that designing a combination generator which provides a reasonnable security is a very difficult (or even impossible) task.

Correlation-immunity order and dual distance of a code. The correlation-immunity order of a Boolean function f can be characterized by a combinatorial property of the code formed by the support of f, i.e., by $f^{-1}(1)$. This combinatorial property corresponds to the notion of *orthogonal array* [Rao47].

Proposition 12.14. *[CCCS92] Let f be a Boolean function of n variables and let $\mathcal{C}_f$ be the code corresponding to its support, i.e., $\mathcal{C}_f = \{x \in \mathbb{F}_2^n : f(x) = 1\}$. Then, f is t-th order correlation immune if and only if any set of t columns in $\mathcal{C}_f$ contains the same number of occurences of every t-tuple.*

Moreover, if $\mathcal{C}_f$ is a linear code, then f is t-th order correlation immune if and only the minimum distance of $\mathcal{C}_f^\perp$ is strictly greater than t.

Proof. By definition $\mathcal{C}_f$ has cardinality $wt(f)$. Let us consider any subset $T \subset \{1,\ldots,n\}$ of size t. Then, f is t-order correlation-immune if and only if, for any fixed value a of the input variables at the positions defined by T, the set $\{f(x), x \in \mathbb{F}_2^n \text{ and } x_i = a_i \forall i \in T\}$ contains exactly $2^{-t}wt(f)$ ones. This equivalently means hat $\mathcal{C}_f$ contains exactly $2^{-t}wt(f)$ words which are equal to a at the positions defined by T.

If $\mathcal{C}_f$ is a linear code and G denotes a $k \times n$ generator matrix of $\mathcal{C}_f$, then the previous condition means that, for any $k \times t$ submatrix G_T of G, the linear system $mG_T = a$ has $2^{k-\ell}$

solutions for any a. This is equivalent to the fact that any $k\times t$ submatrix G_T of G has rank t, i.e., any set of t columns of G are linearly independent. Now, we show that this condition is equivalent to the fact that the minimum distance of $\mathcal{C}_f^\perp$ is greater than t. Indeed $\mathcal{C}_f^\perp$ contains a codeword x if and only if $Gx = 0$, i.e., the $wt(x)$ columns of G corresponding to the support of x sum to zero. In other words, the minimum distance of $\mathcal{C}_f^\perp$ is greater than or equal to $d^\perp$ if and only if any set of $w < d^\perp$ columns of G are linearly independent. ◇

The previous proposition shows that correlation-immunity order of a linear function is determined by the minimum distance of the dual of the code corresponding to its support. A very nice property is that this result holds even if the underlying code is not linear. In this situation, the *dual distance* can be defined as follows.

Definition 12.15 (Dual distance of a code [Del73]). *Let $\mathcal{C}$ be a binary code of length n and size M, and $(A_0,\dots,A_n)$ be its distance distribution, i.e.,*

$$A_i = \frac{1}{M}|\{(x,y)\in\mathcal{C}\times\mathcal{C}, d(x,y)=i\}| \; .$$

Let $(A'_0,\dots,A'_n)$ be the vector obtained by applying the MacWilliams transform:

$$\sum_{i=0}^{n} A'_i X^{n-i}Y^i = \frac{1}{M}\sum_{i=0}^{n} A_i (X+Y)^{n-i}(X-Y)^i \; .$$

The dual distance *of $\mathcal{C}$ is the smallest non-zero integer i such that $A'_i \neq 0$.*

Obviously, when $\mathcal{C}$ is linear, its dual distance corresponds to the minimum distance of the dual code. Now, we will show that, even in the nonlinear case, the dual distance of $\mathcal{C}_f$ is related to the correlation-immunity of f. We will need the following lemma.

Lemma 12.16. *Let $\mathcal{C}$ be a binary code of length n. Then any set of t columns in $\mathcal{C}$ contains the same number of occurences of every t-tuple if and only if for any $y\in\mathbb{F}_2^n$ with $1\le wt(y)\le t$,*

$$\sum_{x\in\mathcal{C}}(-1)^{x\cdot y} = 0 \; .$$

Proof. Let us first consider some $T\subset\{1,\dots,n\}$ of size t and some nonzero $y\in\mathbb{F}_2^n$ such that $\mathsf{supp}(y)\subset T$. For any $a\in\mathbb{F}_2^t$, we denote by $N_T(a)$ the number of codewords in $\mathcal{C}$ which are equal to a at the positions defined by T. Then, we have

$$\sum_{x\in\mathcal{C}}(-1)^{x\cdot y} = \sum_{a\in\mathbb{F}_2^\ell}(-1)^{a\cdot y_T}N_T(a) \; . \tag{12.6}$$

where y_T denotes the restriction of y to the positions in T. Now, we suppose that, for any T of size t, we have $N_T(a) = M2^{-t}$ for all a. We consider any nonzero y of weight at most t and we choose T of size t such that $\mathsf{supp}(y)\subset T$. Then,

$$\sum_{x\in\mathcal{C}}(-1)^{x\cdot y} = \frac{M}{2^t}\left(\sum_{a\in\mathbb{F}_2^\ell}(-1)^{a\cdot y_T}\right) = 0 \; ,$$

since $y_T \neq 0$. Conversely, we consider any subset $T \subset \{1, \ldots, n\}$ of size t. We deduce from (12.6) that, for any $\beta \in \mathbb{F}_2^t$

$$\begin{aligned}\sum_{y_T \in \mathbb{F}_2^t} (-1)^{\beta \cdot y_T} \left(\sum_{x \in \mathcal{C}} (-1)^{x \cdot (y_T, 0)} \right) &= \sum_{y_T \in \mathbb{F}_2^t} (-1)^{\beta \cdot y_T} \sum_{a \in \mathbb{F}_2^t} (-1)^{a \cdot y_T} N_T(a) \\ &= \sum_{a \in \mathbb{F}_2^t} N_T(a) \left(\sum_{y_T \in \mathbb{F}_2^t} (-1)^{(a+\beta) \cdot y_T} \right) = 2^t N_T(\beta) \ ,\end{aligned}$$

by using that $\sum_{y_T \in \mathbb{F}_2^t} (-1)^{(\alpha+\beta)\cdot y_T} = 0$ except for $\alpha + \beta = 0$. Then, if all $\left(\sum_{x \in \mathcal{C}} (-1)^{x \cdot y}\right)$ vanish when $wt(y) \leq t$ except for $y = 0$, we get that, for all β.

$$2^t N_T(\beta) = \sum_{x \in \mathcal{C}} (-1)^{x \cdot 0} = M \ .$$

Both properties are then equivalent. ◇

A direct corollary of the previous lemma is the following characterization of correlation-immune functions in terms of Walsh transform [XM88].

Corollary 12.17. *Let f be a Boolean function of n variables. Then, f is t-th order correlation-immune if and only if for any y such that $1 \leq wt(y) \leq t$,*

$$\sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) + x \cdot y} = 0 \ .$$

Proof. This is derived from the previous lemma by using that for any $y \neq 0$

$$\sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) + x \cdot y} = \sum_{x \in f^{-1}(0)} (-1)^{x \cdot y} - \sum_{x \in f^{-1}(1)} (-1)^{x \cdot y} = -2 \sum_{x \in f^{-1}(1)} (-1)^{x \cdot y} \ .$$

◇

Now, we can prove that the correlation-immunity order of a Boolean function is related to the dual distance of $\mathcal{C}_f = f^{-1}(1)$.

Theorem 12.18 (Correlation-immunity order and dual distance [BGS94, SM95])**.** *Let f be a Boolean function of n variables and let $\mathcal{C}_f$ be the code corresponding to its support, i.e., $\mathcal{C}_f = \{x \in \mathbb{F}_2^n : f(x) = 1\}$. Then, f is t-th order correlation immune if and only if the dual distance of $\mathcal{C}_f$ is strictly greater than t.*

Proof. By combining Proposition 12.14 and Lemma 12.16, we see that we need to show that $\mathcal{C}_f$ has dual distance $d^\perp$ if and only if any nonzero y of weight at most $(d^\perp - 1)$ satisfies

$$\sum_{x \in \mathcal{C}_f} (-1)^{x \cdot y} = 0 \ .$$

The MacWilliams identity can be written by means of Krawtchouk polynomials

$$A'_i = 2^{-k} \sum_{j=0}^{n} A_j P_i(j), \ \forall 0 \leq i \leq n \ ,$$

with

$$P_k(i) = \sum_{j=0}^{k} (-1)^j \binom{i}{j} \binom{n-i}{k-j} .$$

We first prove a well-known property of Krawtchouk polynomials: for any $x \in \mathbb{F}_2^n$ with $wt(x) = i$,

$$\sum_{y \in \mathbb{F}_2^n, wt(y)=j} (-1)^{x \cdot y} = P_j(i) .$$

Let I denote the support of x. Since $x \cdot y$ corresponds to the size of the intersection between the supports of x and y, we get

$$\sum_{y \in \mathbb{F}_2^n, wt(y)=j} (-1)^{x \cdot y} = \sum_{J \subset \{1,\dots,n\}, |J|=j} (-1)^{|I \cap J|} = \sum_{\ell=0}^{j} (-1)^{\ell} N_I(\ell)$$

where

$$N_I(\ell) = |\{J \subset \{1,\dots,n\} \text{ avec } |J| = j : |I \cap J| = \ell\}| = \binom{i}{\ell}\binom{n-i}{j-\ell} .$$

We deduce that

$$\sum_{y \in \mathbb{F}_2^n, wt(y)=j} (-1)^{x \cdot y} = \sum_{\ell=0}^{j} (-1)^{\ell} \binom{i}{\ell}\binom{n-i}{j-\ell} = P_j(i) .$$

Let us now consider some integer w, $1 \le w \le n$. Let us define the matrix M_w of size $M \times \binom{n}{w}$ whose rows are indexed by the words of $\mathcal{C}_f$ and whose columns are indexed by the n-bit words of weight w by

$$(M_w)_{x,y} = (-1)^{x \cdot y} .$$

Multiplying M_w by its transpose, we get that the coefficient of index (x, x') of $M_w(M_w)^T$ is

$$\left(M_w(M_w)^T\right)_{x,x'} = \sum_{y \in \mathbb{F}_2^n, wt(y)=w} (-1)^{(x+x') \cdot y} = P_w(d(x,x')) .$$

Then, we deduce

$$\mathbf{1}\left(M_w M_w^T\right)\mathbf{1}^T = (\mathbf{1}M_w)(\mathbf{1}M_w)^T = \sum_{y \in \mathbb{F}_2^n, wt(y)=w} \left(\sum_{x \in \mathcal{C}_f} (-1)^{x \cdot y}\right)^2 .$$

Therefore,

$$\begin{aligned}\sum_{y \in \mathbb{F}_2^n, wt(y)=w} \left(\sum_{x \in \mathcal{C}_f} (-1)^{x \cdot y}\right)^2 &= \mathbf{1}\left(M_w M_w^T\right)\mathbf{1}^T \\ &= \sum_{x,x' \in \mathcal{C}_f} P_w(d(x,x')) = M \sum_{i=0}^{n} A_i P_w(i) = M^2 A'_w .\end{aligned}$$

It follows that $A'_w = 0$ if and only if $\sum_{x \in \mathcal{C}_f} (-1)^{x \cdot y} = 0$ for all $y \in \mathbb{F}_2^n$ with $wt(y) = w$. $\diamond$

Bibliography

[Arm02] Frederik Armknecht. A linearization attack on the Bluetooth keystream generator. `http://th.informatik.uni-mannheim.de/people/armknecht/E0.ps`, 2002.

[BB06] Elad Barkan and Eli Biham. Conditional estimators: An effective attack on A5/1. In *Selected Areas in Cryptography - SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2006.

[BBK08] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. *Journal of Cryptology*, 21(3):392–429, 2008.

[BD00] Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In *Progress in Cryptology - Indocrypt 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer-Verlag, 2000.

[Ber68] Elwyn R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, 1968.

[BGS94] Jürgen Bierbrauer, K. Gopalakrishnan, and Douglas R. Stinson. Bounds for resilient functions and orthogonal arrays. In *Advances in Cryptology - CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 247–256, 1994.

[BGW99] Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of A5/1. `http://jya.com/a51-pi.htm`, 1999.

[Blu] Bluetooth specifications – version 1.0b. `http://www.bluetooth.com`.

[Bry86] Lennart Brynielsson. On the linear complexity of combined shift register sequences. In *Advances in Cryptology - EUROCRYPT'85*, volume 219 of *Lecture Notes in Computer Science*, pages 156–160. Springer, 1986.

[BSW00] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2000.

[Can02] Anne Canteaut. On the correlations between a combining function and functions of fewer variables. In *IEEE Information Theory Workshop - ITW 2002*, pages 78–81, Bangalore, Inde, October 2002. IEEE Press.

[CCCS92] Paul Camion, Claude Carlet, Pascale Charpin, and Nicolas Sendrier. On correlation-immune functions. In *Advances in Cryptology - CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 86–100. Springer-Verlag, 1992.

[CF01] Anne Canteaut and Eric Filiol. Ciphertext only reconstruction of stream ciphers based on combination generators. In *Fast Software Encryption - FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 165–180. Springer-Verlag, 2001.

[CJM02] Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: an algorithmic point of view. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer-Verlag, 2002.

[CKM94] Don Coppersmith, Hugo Krawczyk, and Yishay Mansour. The shrinking generator. In *Advances in Cryptology - CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[Cou03] Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer-Verlag, 2003.

[CT00] Anne Canteaut and Mickaël Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology - EUROCRYPT'2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000.

[CTZ01] Pascale Charpin, Aimo Tietäväinen, and Victor Zinoviev. On binary cyclic codes with codewords of weight 3 and binary sequences with the trinomial property. *IEEE Transactions on Information Theory*, 47(1):421–425, 2001.

[Del73] P. Delsarte. Four fundamental parameters of a code and their combinatorial signifiance. *Information and Control*, 23(5):407–438, décembre 1973.

[Dor87] Jean-Louis Dornstetter. On the equivalence between berlekamp's and euclid's algorithms. *IEEE Transactions on Information Theory*, 33(3):428–431, 1987.

[EJ03] Patrick Ekdahl and Thomas Johansson. Another attack on A5/1. *IEEE Transactions on Information Theory*, 49(1):284–289, 2003.

[Gal62] Robert G. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, IT-8:21–28, 1962.

[GBM02] Jovan Dj. Golic, Vittorio Bagini, and Guglielmo Morgari. Linear cryptanalysis of Bluetooth stream cipher. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 238–255. Springer-Verlag, 2002.

[Gef73] Philip R. Geffe. How to protect data with ciphers that are really hard to break. *Electronics*, pages 99–101, 1973.

[GN95] Rainer Göttfert and Harald Niederreiter. On the Minimal Polynomial of the Product of Linear Recurring Sequences. *Finite Fields and Applications*, 1(2):204–218, 1995.

[Gol82] Solomon W. Golomb. *Shift register sequences.* Aegean Park Press, 1982.

[Gün88] Christoph G. Günther. Alternating Step Generators Controlled by De Bruijn Sequences. In *Advances in Cryptology - EUROCRYPT'87*, volume 304 of *Lecture Notes in Computer Science*, pages 5–14. Springer, 1988.

[Hel11] Tor Helleseth. Maximal-length sequences. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 763–766. Springer, 2011.

[Her86] Tore Herlestam. On functions of linear shift register sequences. In *Advances in Cryptology - EUROCRYPT '85*, volume 219 of *Lecture Notes in Computer Science*, pages 119–129. Springer-Verlag, 1986.

[HR04] Philip Hawkes and Gregory G. Rose. Rewriting variables: the complexity of fast algebraic attacks on stream ciphers. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 390–406. Springer-Verlag, 2004.

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis.* Chapman & Hall/CRC, 2009.

[Key76] Edwin L. Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Transactions on Information Theory*, 22:732–736, 1976.

[KPPM12] Maria Kalendri, Dionisios Pnevmatikatos, Ioannis Papaefstathiou, and Charalampos Manifavas. Breaking the GSM A5/1 Cryptography Algorithm with Rainbow Tables and High-End FPGAs. In *Field Programmable Logic and Applications - FPL 2012*, pages 747–753. IEEE, Aug 2012.

[Lev04] Sabine Leveiller. *Quelques algorithmes de cryptanalyse du registre filtré.* PhD thesis, Ecole Nationale Supérieure des Télécommunications, Paris, 2004.

[LMV05] Yi Lu, Willi Meier, and Serge Vaudenay. The conditional correlation attack: A practical attack on Bluetooth Encryption. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 97–117. Springer-Verlag, 2005.

[Lu06] Yi Lu. *Applied stream ciphers in mobile communications.* PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2006.

[LV04a] Yi Lu and Serge Vaudenay. Cryptanalysis of Bluetooth keystream generator two-level E0. In *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 483–499. Springer-Verlag, 2004.

[LV04b] Yi Lu and Serge Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer-Verlag, 2004.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, janvier 1969.

[Mas01] James L. Massey. The Ubiquity of Reed-Muller Codes. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes - AAECC-14*, volume 2227 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2001.

[MJB04] Alexander Maximov, Thomas Johansson, and Steve Babbage. An improved correlation attack on A5/1. In *Selected Areas in Cryptography - SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2004.

[MS77] F. Jessie MacWilliams and Neil J.A. Sloane. *The theory of error-correcting codes.* North-Holland, 1977.

[MS89] W. Meier and O. Staffelbach. Fast correlation attack on certain stream ciphers. *Journal of Cryptology*, pages 159–176, 1989.

[MS95] Willi Meier and Othmar Staffelbach. The self-shrinking generator. In *Advances in Cryptology - EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 205–214. Springer-Verlag, 1995.

[NP09] Karsten Nohl and Chris Paget. GSM: SRSLY? In *26th Chaos Communication Congress - 26C3*, 2009.

[PFS00] Slobodan Petrović and Amparo Fúster-Sabater. Cryptanalysis of the A5/2 algorithm. Technical Report 2000/052, Cryptology ePrint Archive, 2000. `http://eprint.iacr.org/`.

[Rao47] Calyampudi Radhakrishna Rao. Factorial experiments derivable from combinatorial arrangements of arrays. *J. Roy. Statist.*, 9:128–139, 1947.

[RC10] Sondre Rønjom and Carlos Cid. Nonlinear equivalence of stream ciphers. In *Fast Software Encryption – FSE 2010*, volume 6147 of *Lecture Notes in Computer Science*, pages 40–54. Springer, 2010.

[RS87] Rainer A. Rueppel and Othmar Staffelbach. Products of linear recurring sequences with maximum complexity. *IEEE Transactions on Information Theory*, 33(1):124–131, 1987.

[Rue86] Rainer A. Rueppel. *Analysis and Design of stream ciphers.* Springer-Verlag, 1986.

[Sie84] Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, IT-30(5):776–780, 1984.

[Sie85] Thomas Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Information Theory*, C-34(1):81–84, 1985.

[SM95] Douglas R. Stinson and James L. Massey. An infinite class of counterexamples to a conjecture concerning nonlinear resilient functions. *Journal of Cryptology*, 8(3):167–173, 1995.

[XM88] Guozheng Xiao and James L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Transactions on Information Theory*, IT-34(3):569–571, 1988.

[Zha00] Muxiang Zhang. Maximum correlation analysis of nonlinear combining functions in stream ciphers. *Journal of Cryptology*, 13(3):301–313, 2000.

[Zie59] Neal Zierler. Linear recurring sequences. *J. Soc. Indus. Appl. Math.*, 7:31–48, 1959.

[ZXF13] Bin Zhang, Chao Xu, and Dengguo Feng. Real Time Cryptanalysis of Bluetooth Encryption with Condition Masking. In *Advances in Cryptology - CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2013.

Chapter 13

Attacks on block ciphers

Block ciphers are the most widely used primitives for ensuring data confidentiality. The existence of several modes of operation leads to encryption schemes with different features and with a reasonable throughput (typically around 20 cycles for encrypting one byte). Therefore, block-cipher-based encryption schemes are appropriate in the vast majority of applications. Another advantage is that these ciphers received a lot of attention since the publication of the DES in the 70s. The most prominent attacks against block ciphers, i.e., differential and linear cryptanalysis, have been developed more than 20 years ago. They are now well-understood and modern block ciphers are designed based on a methodology which guarantees that they are resistant against all these classical attacks.

13.1 Block Cipher Basics

Designing an encryption scheme is a difficult task in particular because it needs to handle messages of an arbitrary length. Therefore, many encryption schemes result from the combination of a cipher operating on fixed-length inputs, named *a block cipher*, and of a mode of operation which describes how this cipher is iteratively used for encrypting data of an arbitrary length.

Definition 13.1. *Let n and κ be two positive integers. A* block cipher *with block-size n and key-size κ is a family of 2^κ permutations E_k of $\mathbb{F}_2^n$, indexed by a key $k \in \mathbb{F}_2^\kappa$.*

In the following, this family is denoted by $(E_k)_{k\in\mathbb{F}_2^\kappa}$ or by $(E_k)_k$ when the key-size is not specified. In practical applications, the key is the single secret parameter within the encryption scheme. Then, the classical security requirement for a block cipher $(E_k)_k$ is that there is no algorithm for recovering the secret key k from the knowledge of some pairs of inputs and outputs of E_k which is significantly more efficient than an exhaustive search for the key. However, the security of modes of operation is usually proved by modeling the block cipher by a pseudo-random permutation. Therefore, a stronger requirement is that a random instance of the block cipher, *i.e.*, E_k for a random secret key k, should be computationally indistinguishable from a random permutation (see *e.g.* [BR05] for formal definitions of the notions of pseudo-random permutation and of strong pseudo-random permutation). Defining a block cipher then boils down to defining 2^κ permutations, as if they had been randomly chosen among all permutations of $\mathbb{F}_2^n$.

13.1.1 Iterated ciphers

For implementation reasons, all classical block ciphers are *iterated ciphers*: they are composed of several round-permutations F_i of $\mathbb{F}_2^n$, where each F_i, $1 \leq i \leq r$, is parametrized by a secret quantity k_i named the round-key, which is derived from the master key k by a key-scheduling algorithm (see Figure 13.1). Parameter r is the number of rounds in the cipher.

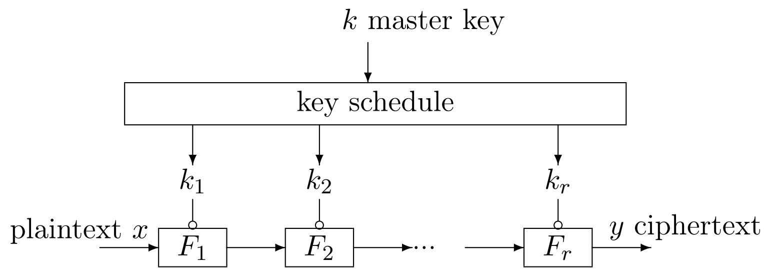


Figure 13.1: Iterated block cipher.

In such an iterated cipher, the r round-permutations F_i are usually chosen to be very similar for two reasons. First, the implementation cost of the iterated cipher in hardware, in terms of number of gates or circuit area, is then roughly reduced to the implementation cost of a single round. Moreover, this type of design provides some simple security arguments. Indeed, it enables the designer to directly derive some property on r rounds of the cipher from a similar property on fewer rounds. However, the rounds should be slightly different in order to resist some structural attacks such as slide attacks [BW99]. This difference may be introduced by the key schedule, *i.e.*, all round permutations can be identical but with different round-keys, or the round permutations may be slightly different, for instance a round-constant can be added to the output. In this second case, all round-keys may be identical. For instance, the block cipher standard AES [FIP01] follows the first construction, while the block cipher LED [GPPR11] follows the second one.

13.1.2 The main types of round functions

There are basically two main constructions for the round permutation: the Feistel construction (and its variants), and the SPN construction.

Feistel ciphers.

The first one is the Feistel construction proposed by Horst Feistel in Lucifer and then used in the former standard DES [FIP99]. It relies on an inner function f_K operating on inputs of size half of the block-size, as depicted on Figure 13.2. This smaller function itself is often composed of several building-blocks. The Feistel construction presents several advantages: first its implementation cost for a given block-size n corresponds to the cost of an $n/2$-bit function. A second advantage is that the round function is an involution (up to the swap of the two inputs), implying that the decryption function is the same as the encryption function, except that the round-keys have to be considered in reverse ordering. This implies that the overhead of decryption on top of encryption is negligible in a Feistel cipher.

The Feistel construction has been widely studied and many results are known which analyze the security of a Feistel cipher by the means of the cryptographic properties of the inner

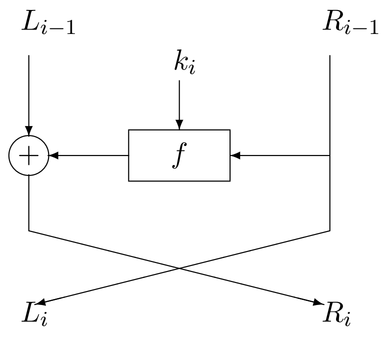


Figure 13.2: Round function in a Feistel cipher.

function. This includes several indistinguishability results [LR88, Pat04] or some results on the resistance against classical statistical attacks [NK95, Nyb95].

Also, there exist some variants of the Feistel construction which allow a higher implementation flexibility, or achieve better performance, including unbalanced Feistel ciphers [SK96], generalized Feistel ciphers [Nyb96] and the MISTY construction [Mat97].

Substitution-Permutation Networks.

Substitution-permutation networks (SPN) are a particular case of the so-called *key-alternating* construction [DR01], which is sometimes referred to as the iterated Even-Mansour construction [EM93]. It consists of an alternation of key-independent round permutations and of round-key additions, where the addition is defined over the vector space $\mathbb{F}_2^n$. Obviously, any operation which can be computed from the plaintext (or from the ciphertext) without involving the secret key does not contribute to the security of the block cipher. Therefore, a key-alternating cipher should both start and end by a round-key addition. Such a cipher is depicted on Figure 13.3.

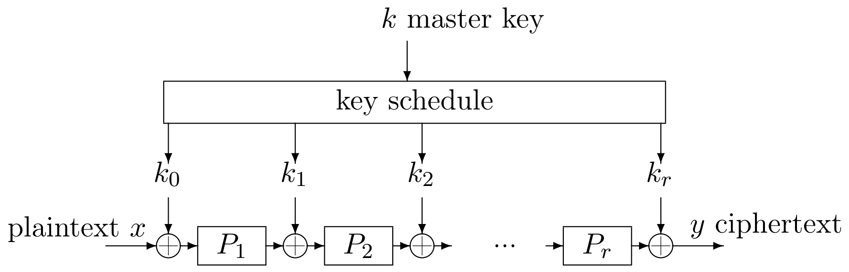


Figure 13.3: Key-alternating cipher.

Moreover, the design strategy for the round permutation in an SPN follows the principles introduced by Shannon [Sha49]:

- *confusion* means making "the relation between the simple statistics of the ciphertext and the simple description of the key a very complex and involved one." This implies for instance that any algebraic relation between these quantities must have a high degree and a large number of terms.

- *diffusion* means "dissipating the statistical structure of the plaintext into long range statistics." This implies that all plaintext bits and key bits must influence all ciphertext bits.

Then, the key idea behind the SPN construction is to decompose the round function into two distinct steps: a nonlinear substitution function Sub for providing confusion and a linear permutation for providing diffusion. Indeed, there is no need for a nonlinear function for providing diffusion since the nonlinearity is already guaranteed by the substitution part. The bottleneck when implementing such a round permutation is the implementation cost of the nonlinear substitution function. Therefore, the classical solution consists in choosing for Sub a permutation corresponding to the concatenation of several copies of a permutation S which operates on a smaller alphabet. This smaller substitution function, which is the only nonlinear part in the cipher, is called the S(ubstitution)-box, by analogy with the terminology used in the former standard DES. In the following, an *Sbox* refers to a vectorial Boolean function, i.e., a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$. In particular, a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$ is called an n-bit Sbox.

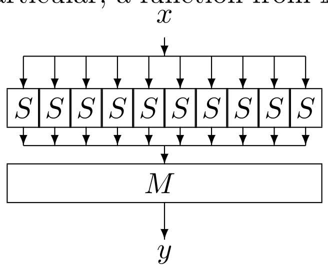


Figure 13.4: Round-permutation of a substitution-permutation network.

The security of this construction with respect to classical statistical attacks will then be discussed in Sections 13.5 and 13.6.

13.2 The AES

When the key-size of the former standard DES has become too small, the NIST has launched a public competition in 1997 to design a successor to the DES. All designers have been invited to submit a block cipher operating on 128-bit blocks, and able to accommodate three different key-sizes, namely 128, 192 and 256 bits. Fifteen proposals have been submitted in 1998, out of which the new standard has been selected in 2000 after a public evaluation process. The selected block cipher, formerly named Rijndael but now known as the AES (Advanced Encryption Standard), has been designed by Joan Daemen and Vincent Rijmen. It follows the SPN construction. The three variants only differ from the number of rounds (which is $r = 10$ for 128-bit keys, 12 for 192-bit keys and 14 for 256-bit keys) and from the key-schedule [FIP01].

The 128-bit word corresponding to the plaintext, and to the input of all successive rounds, is usually represented as a 4×4 matrix of bytes. The round function consists of the successive applications of the following four permutations:

- `SubBytes` is the substitution function. It consists of the parallel application of the same 8-bit permutation S to each input byte;
- `ShiftRows` rotates the rows of the matrix of bytes. More precisely, Row i, $0 \leq i \leq 3$, is rotated to the left by i positions;

- `MixColumns` is a linear transformation, applied in parallel to each column of the matrix. It is defined by a 4×4 matrix with coefficients in $\mathbb{F}_{2^8}$;
- `AddRoundKey` is the round-key insertion which corresponds to a bitwise xor.

All iterations of the round function are identical, except the last one which does not include the `MixColumns` transformation (see Fig 13.5).

13.2.1 The AES Sbox

The AES Sbox, which operates in parallel on each byte of the input of the round function is equivalently defined as a permutation of the finite field with 2^8 elements. The isomorphism between the field $\mathbb{F}_{2^8}$ and the vector space $\mathbb{F}_2^8$ is given by

$$(x_0, x_1, \dots, x_7) \in \mathbf{F}_2^8 \mapsto \sum_{i=0}^{7} x_i X^i$$

where all operations are modulo the irreducible polynomial

$$X^8 + X^4 + X^3 + X + 1.$$

With this identification, the AES Sbox corresponds to the inversion in $\mathbb{F}_{2^8}$ (where the inverse of 0 is 0), i.e.,

$$x \in \mathbf{F}_{2^8} \mapsto x^{254},$$

followed by an affine function over $\mathbb{F}_2^8$:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

13.2.2 `MixColumns`

This 32-bit permutation applies to each column of the internal state. Here, the vector space $\mathbb{F}_2^{32}$ is identified with $(\mathbb{F}_{2^8})^4$ by the previously described isomorphism. Then, `MixColumns` is an $\mathbb{F}_{2^8}$-linear function defined as follows

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \alpha & \alpha+1 & 1 & 1 \\ 1 & \alpha & \alpha+1 & 1 \\ 1 & 1 & \alpha & \alpha+1 \\ \alpha+1 & 1 & 1 & \alpha \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

where α is a root of $X^8 + X^4 + X^3 + X + 1$.

13.2.3 The AES key-schedule

The $(N_r + 1)$ subkeys, where N_r denotes the number of rounds, are derived from the master key by Algorithm 11. These subkeys are represented in an array of 32-bit words, whose 128-bit blocks correspond to the subkeys. The first words in this array are initialized by the master key, implying that the first round-key corresponds to the first 128 key-bits.

Algorithm 11 AES key-schedule for a $32N_k$-bit key, $N_k \in \{4, 6, 8\}$.

```
/* Initialization of the number of rounds */
if $N_k = 4$ then
  $N_r \leftarrow 10$
else if $N_k = 6$ then
  $N_r \leftarrow 12$
else if $N_k = 8$ then
  $N_r \leftarrow 14$
end if
/* Construction of the subkey array $w_0, \dots, w_{4N_r+3}$ */
for $i$ from 1 to $N_k - 1$ do
  $w_i \leftarrow$ Word $i$ of $K$.
end for
for $i$ from $N_k$ to $4N_r + 3$ do
  $t \leftarrow w_{i-1}$
  if $i \equiv 0 \bmod N_k$ then
    Rotate to the left the bytes of $t$
    Apply the Sbox to the four bytes of $t$
    Add to the first byte of $t$ the round constant $\alpha^{i/N_k - 1}$ defined in $\mathbf{F}_{2^8}$.
  else if $i \equiv 4 \bmod N_k$ and $N_k = 8$ then
    Apply the Sbox to the four bytes of $t$
  end if
  $w_i \leftarrow w_{i-N_k} + t$
end for
```

The full AES-128 is depicted on Figure 13.5. Unlike Feistel ciphers, the decryption process for the AES is not directly derived from the encryption function and requires the inverses of the previously described transformations. More details can be found in the NIST publication FIPS 197 [FIP01].

13.3 Algebraic attacks on block ciphers

13.3.1 Basic algebraic attack

The basic principle of algebraic attacks goes back to Shannon's work [Sha49, Page 711]: these techniques consist in expressing the whole cipher as a large system of multivariate algebraic equations. In a known-plaintext attack, the plaintext and ciphertext bits are then replaced by their values and the unknowns correspond to the key bits. The secret key can then be recovered by solving this algebraic system. A major parameter which influences the complexity of such an attack is then the degree of the underlying system.

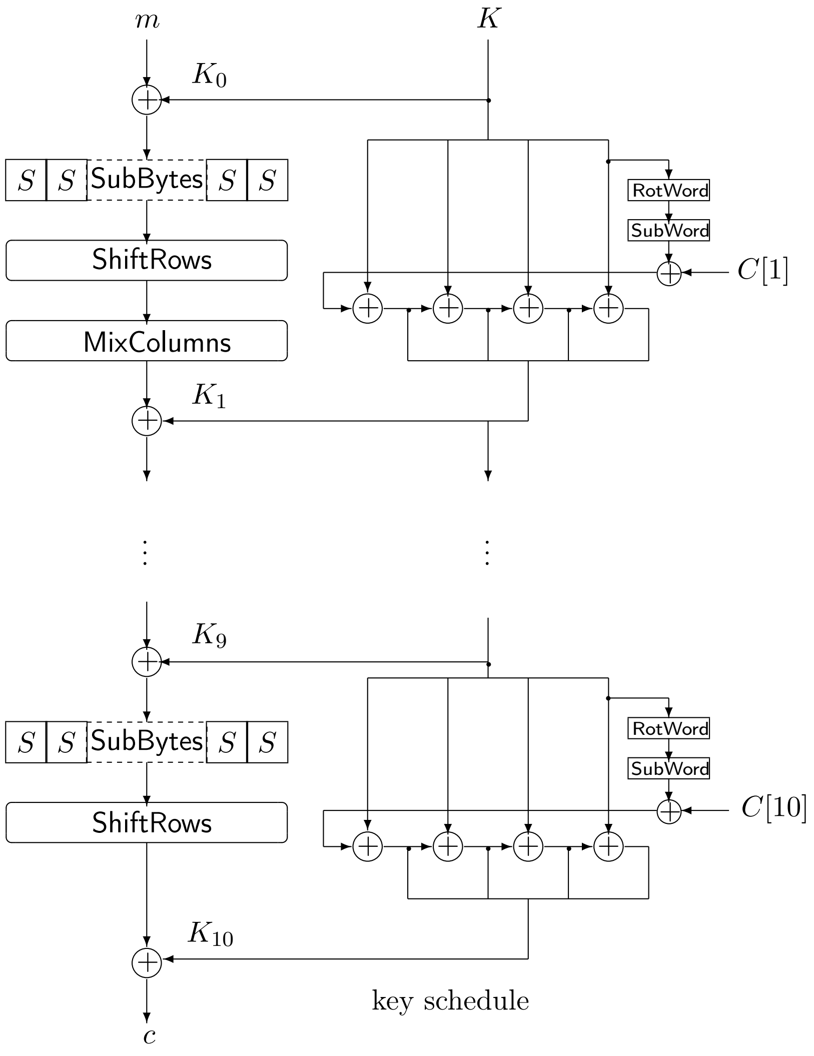


Figure 13.5: AES-128.

A naive method for solving such a system of degree d is the *linearization*. It consists in identifying the system with a linear system of $\sum_{i=1}^{d}\binom{n}{i}$ variables, where each product of i initial variables $(1 \leq i \leq d)$ is seen as a new variable. The solution is then found by a Gaussian reduction (or by more sophisticated techniques) whose time complexity is roughly

$$\left(\sum_{i=1}^{d}\binom{n}{i}\right)^{\omega} \simeq n^{\omega d} ,$$

where ω is the exponent of the matrix inversion algorithm, i.e., $\omega \simeq 2.37$ [CW90]. Some much more sophisticated techniques exist for solving polynomial systems over $\mathbb{F}_2$. Actually, this problem has been extensively studied in computer algebra and it is well-known that some methods based on Gröbner basis algorithms efficiently apply, see e.g. [Fau99, Ste04, Fau02]. Some ad-hoc techniques including XL [CKPS00] or XSL [CP02] have also been proposed. See e.g. [Cid04] and [CAA$^+$08] for a discussion on these algorithms.

Example 13.1. (inspired from [KR11]). Let us consider a toy-cipher composed of a single-round key-alternating cipher which operates on 4-bit inputs under an 8-bit key (K_0, K_1).

$$\begin{tikzcd}
& K_0 \arrow[d] & & & & K_1 \arrow[d] & \\
m \arrow[r] & \oplus \arrow[r] & u \arrow[r] & \boxed{S} \arrow[r] & v \arrow[r] & \oplus \arrow[r] & c
\end{tikzcd}$$

The inner permutation of $\mathbb{F}_2^4$ is defined (with hexadecimal notation) by

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	f	e	b	c	6	d	7	8	0	3	9	a	4	2	1	5
$S_1(x)$	1	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1
$S_2(x)$	1	1	1	0	1	0	1	0	0	1	0	1	0	1	0	0
$S_3(x)$	1	1	0	1	1	1	1	0	0	0	0	0	1	0	0	1
$S_4(x)$	1	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0

The Möbius transform enables us to compute the ANF of the four coordinates of this Sbox:

$$\begin{aligned}
S_1 &= 1 + x_1 + x_3 + x_2x_3 + x_4 + x_2x_4 + x_3x_4 + x_1x_3x_4 + x_2x_3x_4 \\
S_2 &= 1 + x_1x_2 + x_1x_3 + x_1x_2x_3 + x_4 + x_1x_4 + x_1x_2x_4 + x_1x_3x_4 \\
S_3 &= 1 + x_2 + x_1x_2 + x_2x_3 + x_4 + x_2x_4 + x_1x_2x_4 + x_3x_4 + x_1x_3x_4 \\
S_4 &= 1 + x_3 + x_1x_3 + x_4 + x_2x_4 + x_3x_4 + x_1x_3x_4 + x_2x_3x_4
\end{aligned}$$

Using these algebraic expressions, we can now express each ciphertext bit c_i, $1 \leq i \leq 4$, as a multivariate polynomial in the plaintext bits $m_1, \dots, m_4$ and in the key bits $k_1, \dots, k_8$:

$$\begin{aligned}
c_1 + k_5 &= 1 + (m_1 + k_1) + (m_3 + k_3) + (m_2 + k_2)(m_3 + k_3) + (m_4 + k_4) + (m_2 + k_2)(m_4 + k_4) \\
&\quad + (m_3 + k_3)(m_4 + k_4) + (m_1 + k_1)(m_3 + k_3)(m_4 + k_4) + (m_2 + k_2)(m_3 + k_3)(m_4 + k_4) \\
c_2 + k_6 &= 1 + (m_1 + k_1)(m_2 + k_2) + (m_1 + k_1)(m_3 + k_3) + (m_1 + k_1)(m_2 + k_2)(m_3 + k_3) + (m_4 + k_4) \\
&\quad + (m_1 + k_1)(m_4 + k_4) + (m_1 + k_1)(m_2 + k_2)(m_4 + k_4) + (m_1 + k_1)(m_3 + k_3)(m_4 + k_4) \\
c_3 + k_7 &= 1 + (m_2 + k_2) + (m_1 + k_1)(m_2 + k_2) + (m_2 + k_2)(m_3 + k_3) + (m_4 + k_4) + (m_2 + k_2)(m_4 + k_4) \\
&\quad + (m_1 + k_1)(m_2 + k_2)(m_4 + k_4) + (m_3 + k_3)(m_4 + k_4) + (m_1 + k_1)(m_3 + k_3)(m_4 + k_4) \\
c_4 + k_8 &= 1 + (m_3 + k_3) + (m_1 + k_1)(m_3 + k_3) + (m_4 + k_4) + (m_2 + k_2)(m_4 + k_4) + (m_3 + k_3)(m_4 + k_4) \\
&\quad + (m_1 + k_1)(m_3 + k_3)(m_4 + k_4) + (m_2 + k_2)(m_3 + k_3)(m_4 + k_4)
\end{aligned}$$

Every plaintext-ciphertext pair then provides four equations in the eight key-bits:

$$\begin{aligned}
c_1 + k_5 &= S_1(m) + (1+m_3m_4)k_1 + (m_3+m_4+m_3m_4)k_2 + (1+m_2+m_4+m_1m_4+m_2m_4)k_3 \\
&\quad +(1+m_2+m_3+m_1m_3+m_2m_3)k_4 + m_4k_1k_3 + m_3k_1k_4 + (1+m_4)k_2k_3 + (1+m_3)k_2k_4 \\
&\quad +(1+m_1+m_2)k_3k_4 + k_1k_3k_4 + k_2k_3k_4 \\
c_2 + k_6 &= S_2(m) + (m_2+m_3+m_2m_3+m_4+m_2m_4+m_3m_4)k_1 + (m_1+m_1m_3+m_1m_4)k_2 \\
&\quad +(m_1+m_1m_2+m_1m_4)k_3 + (1+m_1+m_1m_2+m_1m_3)k_4 + (1+m_3+m_4)k_1k_2 \\
&\quad +(1+m_2+m_4)k_1k_3 + (1+m_2+m_3)k_1k_4 + m_1k_2k_3 + m_1k_2k_4 + m_1k_3k_4 + k_1k_2k_3 \\
&\quad +k_1k_2k_4 + k_1k_3k_4 \\
c_3 + k_7 &= S_3(m) + (m_2+m_2m_4+m_3m_4)k_1 + (1+m_1+m_3+m_4+m_1m_4)k_2 \\
&\quad +(m_2+m_4+m_1m_4)k_3 + (1+m_2+m_3+m_1m_2+m_1m_3)k_4 + (1+m_4)k_1k_2 \\
&\quad +m_4k_1k_3 + (m_2+m_3)k_1k_4 + k_2k_3 + m_1k_3k_4 + (1+m_1)k_2k_4 + k_3k_4 + k_1k_2k_4 + k_1k_3k_4 \\
c_4 + k_8 &= S_4(m) + (m_3+m_3m_4)k_1 + (m_4+m_3m_4)k_2 + (1+m_1+m_4+m_1m_4+m_2m_4)k_3 \\
&\quad +(1+m_2+m_3+m_1m_3+m_2m_3)k_4 + (1+m_4)k_1k_3 + (m_3)k_1k_4 + m_4k_2k_3 \\
&\quad +(1+m_3)k_2k_4 + (1+m_1+m_2)k_3k_4 + k_1k_3k_4 + k_2k_3k_4
\end{aligned}$$

For instance, the knowledge of $E(0) = \texttt{0x4}$ leads to

$$\begin{aligned}
c_1 + k_5 &= 1 + k_1 + k_3 + k_4 + k_2k_3 + k_2k_4 + k_3k_4 + k_1k_3k_4 + k_2k_3k_4 \\
c_2 + k_6 &= 1 + k_4 + k_1k_2 + k_1k_3 + k_1k_4 + k_1k_2k_3 + k_1k_2k_4 + k_1k_3k_4 \\
c_3 + k_7 &= 1 + k_2 + k_4 + k_1k_2 + k_2k_3 + k_2k_4 + k_3k_4 + k_1k_2k_4 + k_1k_3k_4 \\
c_4 + k_8 &= 1 + k_3 + k_4 + k_1k_3 + k_2k_4 + k_3k_4 + k_1k_3k_4 + k_2k_3k_4
\end{aligned}$$

By collecting such equations, we get a polynomial system of degree 3 with 8 unknowns. Then, we can linearize the system by identifying each monomial in the key bits of degree 2 or 3 with a new unknown:

$$k_9 = k_1k_2,\quad k_{10} = k_1k_3, \ldots\ k_{14} = k_3k_4,\quad k_{15} = k_1k_2k_3, \ldots, k_{18} = k_2k_3k_4$$

transforming the previous four equations into

$$\begin{aligned}
c_1 + k_5 &= 1 + k_1 + k_3 + k_4 + k_{12} + k_{13} + k_{14} + k_{16} + k_{18} \\
c_2 + k_6 &= 1 + k_4 + k_9 + k_{10} + k_{11} + k_{15} + k_{17} + k_{16} \\
c_3 + k_7 &= 1 + k_2 + k_4 + k_9 + k_{12} + k_{13} + k_{14} + k_{17} + k_{16} \\
c_4 + k_8 &= 1 + k_3 + k_4 + k_{10} + k_{13} + k_{14} + k_{16} + k_{18}
\end{aligned}$$

This way, we get a linear system with $8 + \binom{4}{2} + \binom{4}{3} = 18$ unknowns, which can be solved as far as it has enough equations, i.e., as far as 5 plaintext-ciphertext pairs are known, leading to 20 equations.

Practical block ciphers have obviously more than a single round. In this case, the degree of the polynomial system is expected to grow as $(\deg S)^r$ when the number of rounds r increases. Solving such a system then becomes infeasible even for a few rounds only and with sophisticated techniques. An alternative solution consists in using some intermediate variables in order to handle equations of a reasonable degree. For instance, let us consider the following two-round key-alternating cipher

$$\begin{tikzcd}
& K_0 \arrow[d] & & & & K_1 \arrow[d] & & & & K_2 \arrow[d] & \\
m \arrow[r] & \oplus \arrow[r] & u \arrow[r] & \boxed{S} \arrow[r] & v \arrow[r] & \oplus \arrow[r] & w \arrow[r] & \boxed{S} \arrow[r] & x \arrow[r] & \oplus \arrow[r] & c\,.
\end{tikzcd}$$

Then, we can consider the 4 bits of the intermediate value v as four additional unknowns. With this method, a plaintext-ciphertext pair provides 8 equations of degree 3 involving 16 unknowns (the twelve key-bits and the four bits of v). Any additional plaintext-ciphertext pair provides four new equations, but introduces four more unknowns. Therefore, from N plaintext-ciphertext pairs, we obtain a system with $8N$ equations and $(12 + 4N)$ unknowns.

13.3.2 Enhanced algebraic attack

Courtois and Pieprzyk [CP02] have pointed out that it might be possible to lower the degree of the polynomial system that we need to solve even if the round function has a high degree. Indeed, the attacker may equivalently exploit any Boolean relation between the plaintext bits, the ciphertext bits and the key-bits.

Example 13.2. If we come back to our previous toy-cipher, we can check that, even if the inner permutation S has degree 3, there exist some Boolean relations of degree 2 only between its inputs and outputs, for instance it can be checked that

$$x_2x_4 + x_2S_1(x_1, \dots, x_4) + x_2S_2(x_1, \dots, x_4) = 0$$

for all inputs. Any plaintext-ciphertext pair for our single-round toy-cipher then leads to the following quadratic equation

$$(m_4 + c_1 + c_2)k_2 + m_2k_4 + m_2k_5 + m_2k_6 + k_2k_4 + k_2k_5 + k_2k_6 = m_2m_4 + m_2c_1 + m_2c_2\ .$$

A total of 21 linearly independent relations of degree of 2 between the input and output bits of S can be exhibited, implying that the derived polynomial system is easier to solve than the original cipher equations.

An important quantity which affects the complexity of this algebraic attack is obviously the lowest degree we can reach for a Boolean relation between the inputs and outputs of the round permutation.

Proposition 13.2. *Let S be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$. The Boolean relations of degree at most d between the inputs and outputs of S are the elements of the kernel of the binary matrix with $\sum_{i=0}^{d}\binom{2n}{i}$ rows and 2^n columns whose rows correspond to the value vectors of the n-variable Boolean function*

$$x^uS(x)^v, u, v \in \mathbb{F}_2^n \text{ such that } wt(u) + wt(v) \leq d\ .$$

Most notably, the number of linearly independent relations of degree at most d is at least

$$\sum_{i=0}^{d}\binom{2n}{i} - 2^n\ .$$

Proof. This result comes from the fact that the relations between x and $S(x)$ correspond to the functions

$$\sum_{u,v\in\mathbb{F}_2^n} c_{u,v} x^u \left[S(x)\right]^v$$

which vanish for all possible inputs x. Moreover, the degree of such a relation corresponds to the maximal value of $wt(u) + wt(v)$ such that $c_{u,v} = 1$. Any such relation is then defined by a linear combination of the monomials $x^u S(x)^v$ which vanish at all points, i.e., an element in the kernel of the matrix defined by the value vectors of all these monomials. The involved matrix has 2^n columns, and $\sum_{i=0}^{d} \binom{2n}{i}$ rows (corresponding to all possible pairs (u, v)). Then, the dimension of its kernel exceeds the difference between the number of rows and the number of columns. $\diamond$

We deduce for instance that any function from $\mathbb{F}_2^4$ into $\mathbb{F}_2^4$ has at least

$$\sum_{i=0}^{2} \binom{8}{i} - 2^4 = 37 - 16 = 21$$

quadratic relations between its inputs and outputs, as we observed in the toy-example.

Case of the AES. The AES Sbox can be seen as the composition of the inversion over $\mathbb{F}_{2^8}$ with an affine function. More precisely,

$$S : A \circ \varphi^{-1} \circ (\varphi(x))^{254} \ ,$$

where φ is the isomorphism from $\mathbb{F}_2^8$ into $\mathbb{F}_{2^8}$ defined by the basis $\{1, \alpha, \ldots, \alpha^7\}$ and α is a root of $X^8 + X^4 + X^3 + X + 1$. By definition, the inversion Inv over $\mathbb{F}_{2^8}$ satisfies

$$x^2 \,\mathsf{Inv}(x) = x^2 x^{254} = x \ .$$

Since $x \mapsto x^2$ is an $\mathbb{F}_2$-linear mapping over $\mathbb{F}_2^8$, we deduce that this relation corresponds to eight Boolean relations over $\mathbb{F}_2$ between the inputs and outputs of S. Actually, there exists 39 such quadratic relations for the AES Sbox. This is much higher than expected for a randomly chosen mapping over $\mathbb{F}_2^8$, since Proposition 13.2 shows that there is no relation of degree 3 for a random 8-bit Sbox.

Using these relations of degree 2, a quadratic system can be formed by introducing intermediate variables corresponding to the outputs of the successive rounds. However, solving this system is infeasible due to its size: 8000 quadratic equations of 1600 variables.

13.4 Statistical attacks

While algebraic attacks exploit the existence of relations within the cipher which always hold, statistical attacks exploit relations which hold with some probability only. More precisely, these attacks rely on the existence of a *distinguisher*. A distinguisher $\mathcal{D}$ for a family of functions (resp. of permutations) $(F_k)_k$ over $\mathbb{F}_2^n$ is an algorithm which takes as input some pairs (x_i, y_i), $1 \leq i \leq N$, and outputs 0 or 1. The aim of $\mathcal{D}$ is to decide whether these (x_i, y_i) are input-output pairs of a randomly chosen element in family $(F_k)_k$ ($\mathcal{D}$ then returns 1) or of a randomly chosen n-bit function (resp. permutation) ($\mathcal{D}$ returns 0) . The advantage of the distinguisher is then defined by

$$\mathsf{Adv}(\mathcal{D}) = \left| \Pr[\mathcal{D}^\pi = 1 | \pi \in_R \{F_k, k \in \mathbb{F}_2^\kappa\}] - \Pr[\mathcal{D}^\pi = 1 | \pi \in_R \mathsf{Func}_n] \right|$$

where Func_n is the set of n-bit functions. Instead of N plaintext-ciphertext pairs, a d-th order distinguisher can take as inputs N d-tuples of plaintext-ciphertext [Vau99].

The existence of a distinguisher with non-negligible advantage is an unsuitable property for a block-cipher, but it may be only a marginal threat in practice since it may not help to recover the key. It is worth noticing that, like for stream ciphers, the existence of a distinguisher is a real threat if the set of possible plaintexts is small (and can be exhaustively enumerated).

In the case of an iterated cipher

$$E_k = F_{k_r} \circ \ldots \circ F_{k_1}$$

much more serious attacks can be derived from a distinguisher for the so-called *reduced cipher*, i.e., the family $\mathcal{G}$ of all permutations obtained by removing the final round of the original cipher:

$$G_k = F_{k_{r-1}} \circ \ldots \circ F_{k_1} \ .$$

If a distinguisher $\mathcal{D}$ for the reduced cipher can be found, then the attacker can mount a *last-round attack*, which aims at recovering the last-round key k_r from the knowledge of some plaintext-ciphertext pairs[a]. Indeed, the attacker can apply the distinguisher for the reduced cipher to pairs of inputs/outputs of the function

$$H_{\widehat{k}} = F_{\widehat{k}}^{-1} \circ E_k = F_{\widehat{k}}^{-1} \circ F_{k_r} \circ F_{k_{r-1}} \circ \ldots \circ F_{k_1}$$

where $\widehat{k}$ takes all possible values for the last-round key. The input-output pairs $(x, H_{\widehat{k}}(x))$ are derived from plaintext/ciphertext pairs by applying to the ciphertext the inverse of the last-round function $F_{\widehat{k}}^{-1}$. Clearly, if $\widehat{k}$ is a correct guess for the last-round key, i.e. $\widehat{k} = k_r$, then $H_{\widehat{k}}$ belongs to the family of reduced-ciphers. Otherwise, if $\widehat{k}$ is a wrong guess, $H_{\widehat{k}}$ is assumed to have the same behavior as a randomly chosen permutation (this assumption is known as *the wrong-key randomization hypothesis*) [Har96, Kuk99]. Another implicit assumption is that the reduced-cipher G_k has roughly the same behavior for all values of the key. This hypothesis is known as the *fixed-key equivalence hypothesis* in the context of linear cryptanalysis, and the *stochastic equivalence hypothesis* for differential cryptanalysis [LMM91]. It is important to

[a]Once k_r has been recovered, the remaining key-bits can be found either by repeating the same attack but on the ciphers obtained by successively removing the last round, or by an exhaustive search (or even by a combination between both methods).

note that it is not always satisfied especially when the cipher consists of a small number of rounds only (see *e.g.* [DR02, Section 8.7.2], [DR07], [DR09] and [BBL13]).

The algorithm for a last-round attack exploiting a d-th order distinguisher $\mathcal{D}$ is described in Algorithm 12. This description assumes that an exhaustive search for the last-round key is performed, which is often infeasible (and not relevant when the round-keys have the same size as the master key like in AES-128). In most situations, it is then important that the distinguisher does not involve all round-key bits. In this case, the distinguisher enables the attacker to partition the set of all last-round keys into equivalence classes.

Algorithm 12 Last-round attack exploiting a distinguisher $\mathcal{D}$ for the reduced cipher.

```
Inputs. $N$ $d$-tuples of plaintexts $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ and the $N$ corresponding tuples of ciphertexts $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_N$.
Output. A set of candidates for the last-round key.
for all possible values $\widehat{k}$ for the last round-key $k_r$ do
  counter $\leftarrow 0$
  for $i$ from 1 to $N$ do
    $\boldsymbol{y}_i \leftarrow (F^{-1}_{\widehat{K}}(c_{i,1}) \ldots, F^{-1}_{\widehat{K}}(c_{i,d}))$ where $\boldsymbol{c}_i = (c_{i,1}, \ldots, c_{i,d})$
    counter $\leftarrow$ counter $+ \mathcal{D}(\boldsymbol{x}_i, \boldsymbol{y}_i)$
  end for
  if counter $\geq$ threshold then
    return $\widehat{k}$
  end if
end for
```

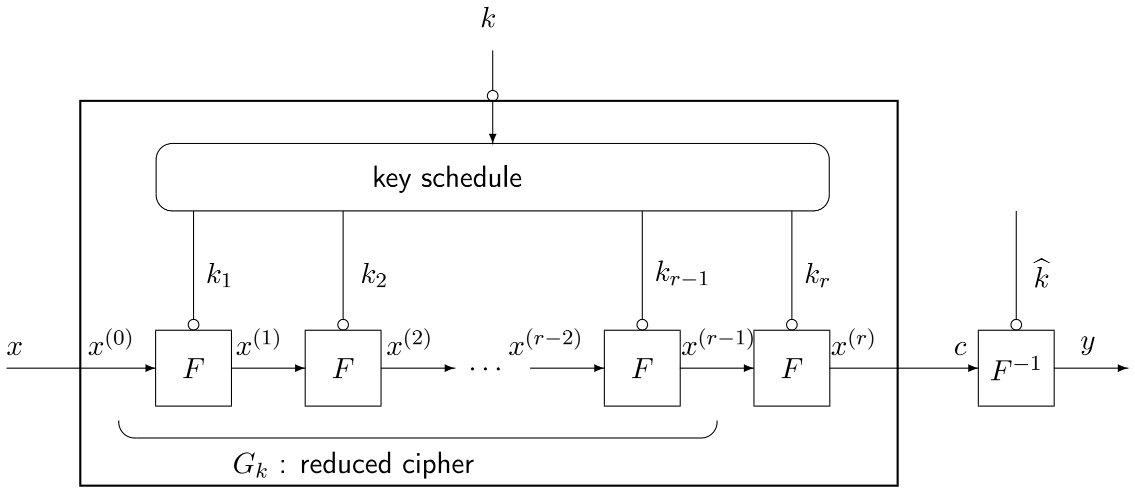


Figure 13.6: Principle of last-round attacks.

13.5 Linear cryptanalysis

13.5.1 Principle

The principle of linear cryptanalysis has been originally presented by Gilbert, Chassé and Tardy-Corfdir [GC91, TCG91] on the block cipher FEAL, and then applied to DES by Matsui [Mat94, Mat95]. It exploits, as a (first-order) distinguisher a linear Boolean relation between the input bits, output bits and the key-bits of the reduced cipher, which is biased, i.e., which holds with a probability different from 1/2. In other words, it uses a triple of *masks* $(\alpha, \beta, \gamma) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \times \mathbb{F}_2^\kappa$ such that

$$\mathrm{Pr}_x[\alpha \cdot x + \beta \cdot G_k(x) + \gamma \cdot k = 0] = \frac{1}{2}(1+\varepsilon) \text{ with } \varepsilon \neq 0 \ .$$

The number N of input-output pairs required for distinguishing the reduced cipher from a random permutation with such a biased relation is at least $\varepsilon^{-2} \times \ln 2$. Indeed, each value $\alpha \cdot x + \beta \cdot G_k(x)$ can be seen as the result of the transmission of $\gamma \cdot k$ through a binary symmetric channel with error-probability $(1-\varepsilon)/2$. The capacity of this channel can then be approximated by $\varepsilon^2/\ln 2$. Since the transmitted word belongs to a code of length N and dimension 1, it can be decoded if its rate, $1/N$, is smaller than the capacity, i.e. if

$$N \geq \frac{\ln 2}{\varepsilon^2} \ .$$

A linear distinguisher for the whole cipher then enables the attacker to recover one information bit of the key, namely $\gamma \cdot k$, from the knowledge of $\ln 2/\varepsilon^2$ plaintext-ciphertext pairs.

Example 13.3. We consider the same cipher with a 4-bit block and an 8-bit key as in Example 13.1. Recall that its inner permutation is defined by

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	f	e	b	c	6	d	7	8	0	3	9	a	4	2	1	5
$S_1(x)$	1	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1
$S_2(x)$	1	1	1	0	1	0	1	0	0	1	0	1	0	1	0	0
$S_3(x)$	1	1	0	1	1	1	1	0	0	0	0	0	1	0	0	1
$S_4(x)$	1	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0

The it can be checked that the linear relation defined by the input mask $\alpha = (1,0,0,1) =$ `0x9` and $\beta = (0,1,0,0) =$ `0x2` has the following value vector (with hexadecimal notation):

$$x_1 + x_4 + S_2(x) = \texttt{0x7ffd} \ .$$

In particular, its Hamming weight is 14 implying that this relation holds with probability $\frac{2}{16} = \frac{1}{8}$. From this biased relation, we can derive a distinguisher for the whole cipher described in Example 13.1:

$$(\alpha \cdot m) \oplus (\beta \cdot c) = (\alpha \cdot K_0) \oplus (\beta \cdot K_1) + 1$$

with probability 7/8. It recovers the key information-bit $(k_1+k_4+k_5+k_8)$ by a simple majority vote between the binary values taken by $m_1 + m_4 + c_2 + 1$ for a few plaintext-ciphertext pairs.

Using the previously described framework, a last-round can be mounted also from a distinguisher on the reduced cipher. Then, the data complexity of the attack increases to

$$2^{\kappa_r}\varepsilon^{-2}$$

where κ_r is the number of information bits of the key for which an exhaustive search is performed in the last-round attack.

13.5.2 Finding good linear approximations

Usually for finding a good linear approximation over some rounds of the cipher, we search for good linear trails (aka linear paths) by chaining some linear approximations over the successive rounds. We search for a linear approximation with input mask α_i and output mask β_i for the i-th round, $1 \leq i \leq r$, such that $\alpha_{i+1} = \beta_i$. The overall probability for the linear approximation over r rounds can then be estimated by the following lemma, known as *piling-up lemma*, which can be easily proved by induction.

Lemma 13.3 (Piling-up lemma [Mat94])**.** *Let $X_1, \ldots X_m$ be m independent random variables which take the value 0 with probability $\frac{1}{2}(1+\varepsilon_i)$ and the value 1 with probability $\frac{1}{2}(1-\varepsilon_i)$. Then*

$$\Pr[X_1 + \ldots + X_m = 0] = \frac{1}{2}\left(1 + \prod_{i=1}^{m} \varepsilon_i\right) .$$

The independence between the successive rounds comes from the round-key insertion, but the validity of this hypothesis can be discussed depending on the key schedule algorithm. However, for mounting a linear attack in practice, we often start by searching for linear approximations with masks (α_i, β_i) with $\alpha_{i+1} = \beta_i$ such that the product $\prod_{i=1}^{r} |\varepsilon_i|$ is maximal. Since all rounds are usually similar, we need to estimate precisely the biases of all linear approximations over a single round.

13.5.3 Biases of linear approximations and Walsh transform

The problem we need to solve is the following. Let F be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$ (typically F is the round-function of an iterated cipher). We want to compute the biases of all Boolean functions of n variables

$$x \mapsto b \cdot F(x) + a \cdot x$$

for all a and all nonzero b in $\mathbb{F}_2^n$, in the sense of the following definition.

Definition 13.4. *The* bias *(aka,* correlation*, or* imbalance*) of an n-variable Boolean function f is*

$$\mathcal{E}(f) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} = 2^n - 2wt(f) .$$

In other words,

$$\Pr_X[f(X) = 1] = \frac{wt(f)}{2^n} = \frac{1}{2}\left(1 - \frac{\mathcal{E}(f)}{2^n}\right) .$$

Most notably, a Boolean function f is balanced if and only if $\mathcal{E}(f) = 0$.

In the following, we denote by φ_a the linear Boolean function $x \mapsto a \cdot x$.

The bias of a linear function is determined as follows.

Lemma 13.5. *Let $a \in \mathbb{F}_2^n$. Then*

$$\mathcal{E}(\varphi_a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x} = \begin{cases} 0 & \text{if } a \neq 0 \\ 2^n & \text{if } a = 0 \end{cases}$$

Proof. Assume that $a \neq 0$ (the result for $a = 0$ is trivial). Then, the set

$$H_a = \{x \in \mathbb{F}_2^n : a \cdot x = 0\}$$

is a hyperplane of $\mathbb{F}_2^n$, i.e., a subspace of dimension $(n-1)$. Then,

$$\sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x} = \#H_a - (2^n - \#H_a) = 0 \; .$$

◇

In linear cryptanalysis, we are interested in the biases $\mathcal{E}(f + \varphi_a)$ when a varies and when f corresponds to all linear combinations of the coordinates of F. A useful tool for studying all $\mathcal{E}(f + \varphi_a)$ is the Walsh transform of f.

Definition 13.6 (Walsh transform)**.** *Let f be a Boolean function of n variables. The Walsh transform of f is the function*

$$\begin{array}{lll} \mathbb{F}_2^n & \to & \mathbb{Z} \\ a & \mapsto & \mathcal{E}(f + \varphi_a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) + a \cdot x} \; . \end{array}$$

Since it corresponds to a discrete Fourier transform, The Walsh transform enjoys all mathematical properties of a Fourier transform. For instance, the Walsh transform is (up to a constant factor) an involution.

Proposition 13.7. *Let f be a Boolean function of n variables. For all $b \in \mathbb{F}_2^n$, we have*

$$\sum_{a \in \mathbb{F}_2^n} (-1)^{a \cdot b} \mathcal{E}(f + \varphi_a) = 2^n (-1)^{f(b)} \; .$$

Proof.

$$\begin{array}{rcl} \sum_{a \in \mathbb{F}_2^n} (-1)^{a \cdot b} \mathcal{E}(f + \varphi_a) & = & \sum_{a \in \mathbb{F}_2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot b} (-1)^{f(x) + a \cdot x} \\ & = & \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} \sum_{a \in \mathbb{F}_2^n} (-1)^{a \cdot (x+b)} \\ & = & \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} \mathcal{E}(\varphi_{x+b}) = 2^n (-1)^{f(b)} \end{array}$$

where the last equality is derived from Lemma 13.5 which states that $\mathcal{E}(\varphi_{x+b}) = 0$ unless $x = b$. ◇

Computing the Walsh transform.

Another useful property of the Fourier transform is that it can be computed by a fast algorithm. Similarly, while the naive method for computing all values of the Walsh transform of an n-variable Boolean function has complexity 2^{2n}, the complexity of the fast algorithm described in Algorithm 13 is proportional to $n 2^n$.

Algorithm 13 Evaluating the Walsh transform of an n-variable Boolean function f.

```
**Input:** $(f[a], 0 \leq a < 2^n)$
**Output:** $(e[a] = \mathcal{E}(f + \varphi_a), 0 \leq a < 2^n)$
for $i$ from 0 to $2^n - 1$ do
  $e[i] \leftarrow (-1)^{f[i]}$
end for
for $k$ from 1 to $n$ do
  for $i$ from 0 to $2^{n-k}$ do
    // Compute the image of the $i$-th $2^k$-bit block
    for $j$ from 0 to $2^{k-1} - 1$ do
      $e'[2^k i + j] \leftarrow e[2^k i + j] + e[2^k i + 2^{k-1} + j] \bmod 2$
      $e'[2^k i + 2^{k-1} + j] \leftarrow e[2^k i + j] - e[2^k i + 2^{k-1} + j] \bmod 2$
    end for
  end for
  $e \leftarrow e'$
end for
return $e$
```

Example 13.4. Let us compute the Walsh transform of the 3-variable function

$$f(x_1, x_2, x_3) = x_1 + x_1x_2 + x_2x_{;}.$$

$f(a)$	0	1	0	0	0	1	1	1
$(-1)^{f(a)}$	1	-1	1	1	1	-1	-1	-1
step 1	0	2	2	0	0	2	-2	0
step 2	2	2	-2	2	-2	2	2	2
$\mathcal{E}(f + \varphi_a)$	0	4	0	4	4	0	-4	0

We deduce that the highest bias (in magnitude) of a linear approximation of this function is 4 and is obtained for instance for $\mathcal{E}(f + x_1)$. This equivalently means

$$\Pr[f(x_1, x_2, x_3) + x_2 = 0] = \frac{1}{2}\left(1 + \frac{4}{2^3}\right) = \frac{3}{4}\ .$$

Linearity and bent functions.

Since linear cryptanalysis exploits a highly biased linear approximation, a natural question for a designer who needs to choose a good nonlinear building-block is to know the smallest possible value we can achieve for the bias of the best linear approximation. This value is determined by the so-called *linearity* of the linear combinations of the coordinates of the function, in the sense of the following definition.

Definition 13.8 (Linearity of a Boolean function)**.** *Let f be a Boolean function of n variables. The* linearity *of f is the highest magnitude of its Walsh coefficients, i.e.,*

$$\mathcal{L}(f) = \max_{a \in \mathbb{F}_2^n} |\mathcal{E}(f + \varphi_a)|\ .$$

There is a general lower bound of the linearity of a Boolean function of n variables, since the Walsh transform, as any Fourier transform, satisfies the Parseval relation.

Proposition 13.9 (Parseval relation)**.** *Let f be a Boolean function of n variables. Then,*

$$\sum_{a\in\mathbb{F}_2^n}[\mathcal{E}(f+\varphi_a)]^2 = 2^{2n} .$$

Proof.

$$\begin{aligned}\sum_{a\in\mathbb{F}_2^n}\mathcal{E}^2(f+\varphi_a) &= \sum_{a\in\mathbb{F}_2^n}\left(\sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)+a\cdot x}\right)\left(\sum_{y\in\mathbb{F}_2^n}(-1)^{f(y)+a\cdot y}\right)\\ &= \sum_{x\in\mathbb{F}_2^n}\sum_{y\in\mathbb{F}_2^n}(-1)^{f(x)+f(y)}\left(\sum_{a\in\mathbb{F}_2^n}(-1)^{a\cdot(x+y)}\right)\\ &= 2^n\sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)+f(x)} = 2^{2n}\end{aligned}$$

where the last-but-one equality is derived from Lemma 13.5. ◇

We directly derive the following lower bound on the linearity of a Boolean function.

Proposition 13.10. *Let f be a Boolean function of n variables. Then*

$$\mathcal{L}(f) \geq 2^{\frac{n}{2}} .$$

The functions for which equality holds are called bent functions. *They exist for even n only, and are not balanced.*

Proof. We first observe that a function f satisfying $\mathcal{L}(f) < 2^{\frac{n}{2}}$ cannot exist. Otherwise, from Parseval relation, we would have

$$2^{2n} = \sum_{a\in\mathbb{F}_2^n}\mathcal{E}^2(f+\varphi_a) < 2^n \times 2^n ,$$

a contradiction.

Obviously, this bound on the linearity is tight when n is even only. Let now f be a bent function. Parseval relation implies that, for all $a \in \mathbb{F}_2^n$,

$$\mathcal{E}^2(f+\varphi_a) = 2^n .$$

In other words, the Wash transform of a bent function has constant magnitude. In particular, for $a = 0$, we get that $\mathcal{E}(f) = \pm 2^{\frac{n}{2}}$ implying that f is not balanced. ◇

Since their output is biased, bent functions are of little use in cryptography. Most notably, any linear combination of the coordinates of a permutation is balanced, and then satisfies $\mathcal{L}(f) > 2^{\frac{n}{2}}$.

When n is odd, bent functions do not exist, and the lowest possible value for $\mathcal{L}(f)$ is unknown for $n \geq 9$. The quadratic function of $(2t+1)$ variables

$$f(x_1,\ldots,x_{2t+1}) = x_1x_2 + x_3x_4 + \ldots + x_{2t-1}x_{2t} + x_{2t+1}$$

satisfies

$$\mathcal{L}(F) = 2^{\frac{n+1}{2}} .$$

We then deduce the following proposition.

Proposition 13.11. *The lowest linearity for a Boolean function of n variables, n odd, satisfies*

$$2^{\frac{n}{2}} < \min_{f \in \mathsf{Bool}_n} \mathcal{L}(f) \leq 2^{\frac{n+1}{2}}$$

where the upper bound is tight for $n \leq 7$ and is not tight for $n \geq 9$.

The fact that the previous upper bound is not tight for $n \geq 9$ was a long-standing open problem solved in 2006 by [KMY07]. More precisely, Table 13.1 gives the lowest possible linearities for an n-variable Boolean function for n odd, $5 \leq n \leq 15$. Note that, by definition, the values of the Walsh transform, including the linearity, are always even.

n	5	7	9	11	13	15
$\min_{f \in \mathsf{Bool}_n} \mathcal{L}(f)$	8	16	$24-30$	$46-60$	$92-120$	$182-216$

Table 13.1: Smallest possible linearity for an n-variable Boolean function, where $a-b$ means that the lowest linearity can be any even integer in this range.

As previously mentioned, most cryptographic applications require the use of balanced functions. The lowest linearity for a balanced function of n variables is also unknown for n even. Only an upper bound on its value is derived from a recursive construction due to Dobbertin [Dob94]. Table 13.2 gives the lowest possible linearities for an n-variable *balanced* Boolean function for $4 \leq n \leq 10$. It is worth noticing that, since the weight of a balanced function is even, its degree is at most $(n-1)$. Then, all $(f + \varphi_a)$ have degree at most $(n-1)$ and an even degree, implying that their biases are divisible by 4.

n	4	5	6	7	8	9	10
$\min_{f \in \mathsf{Bool}_n} \mathcal{L}(f)$	8	8	12	16	$\{20, 24\}$	$\{24, 28, 32\}$	$\{36, 40\}$

Table 13.2: Smallest possible linearities for an n-variable balanced function.

13.5.4 Link with Reed-Muller codes.

The previously mentioned problems on the lowest possible linearity for a Boolean function have been extensively investigated in coding theory since they are related to the determination of the covering radius of the first-order Reed-Muller code. Indeed, the Walsh coefficients of a given Boolean function are determined by the weights of the coset of $R(1,n)$ defined by the value vector of f, as detailed in the following proposition.

Proposition 13.12. *Let f be a Boolean function of n variables. Then, the Hamming weights of the coset $f + R(1,n)$ defined by the value vector of f are*

$$\{2^{n-1} - \frac{1}{2}\mathcal{E}(f + \varphi_a); 2^{n-1} + \frac{1}{2}\mathcal{E}(f + \varphi_a),\ a \in \mathbb{F}_2^n\}\ .$$

In particular, the Hamming distance of f to $R(1,n)$, called the nonlinearity of f, *is defined by*

$$d(f, R(1,n)) = 2^{n-1} - \frac{1}{2}\mathcal{L}(f) .$$

Proof. By definition of $R(1,n)$, the weights of $f + R(1,n)$ correspond to the weights of all words $c = f + \varphi_a + \varepsilon$, $a \in \mathbb{F}_2^n, \varepsilon \in \mathbb{F}_2$. If $\varepsilon = 0$, we get that

$$wt(f + \varphi_a) = 2^{n-1} - \frac{1}{2}\mathcal{E}(f + \varphi_a) .$$

If $\varepsilon = 1$, we have

$$wt(f + \varphi_a + 1) = 2^n - wt(f + \varphi_a) = 2^{n-1} + \frac{1}{2}\mathcal{E}(f + \varphi_a) .$$

The expression of $d(f, R(1,n))$ directly follows. ◇

Finding a function with minimal linearity then boils down to finding a value vector which lies as far as possible from the code $R(1,n)$. This corresponds to the well-known notion of *covering radius.*

Definition 13.13 (Covering radius)**.** *Let $\mathcal{C}$ be a code of length n. The covering radius of $\mathcal{C}$ is the highest Hamming distance between $\mathcal{C}$ and a word in $\mathbb{F}_2^n$:*

$$\rho(\mathcal{C}) = \max_{c \in \mathbb{F}_2^n} d(c, \mathcal{C}) .$$

Therefore, most of the previously mentioned results on the best linearity for a Boolean functions have been first proved in terms of covering radius of the first-order Reed-Muller codes, e.g. [Myk80, PW83, Hou93, Hou96b, Hou96a].

Linearity of Sboxes and almost bent functions.

We have studied the linearity of a Boolean function, but in order to thwart linear attacks, we need to choose a vectorial function (aka an Sbox) such that all linear combinations of its coordinates have a low linearity. Therefore, we define the linearity of an Sbox as follows.

Definition 13.14. *Let S be function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$. For any $b \in \mathbb{F}_2^m$, we denote by S_b the Boolean function*[b] *$x \mapsto b \cdot S(x)$. Then, the linearity of S is the highest linearity of any of its nonzero components, i.e.*

$$\mathcal{L}(S) = \max_{b \in (\mathbb{F}_2^m)^*} \mathcal{L}(S_b) = \max_{a, b \in (\mathbb{F}_2^m)^*} |\mathcal{E}(S_b + \varphi_a)| .$$

The linearity of S is now related to the weights of the following code.

Proposition 13.15 ([CCZ98])**.** *Let S be an Sbox from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$. Let us consider the code Γ_S having for generator matrix the following $(2n+1) \times 2^n$-matrix*

$$\begin{pmatrix} S(0) & S(1) & S(2) & \dots & S(2^n-1) \\ 0 & 1 & 2 & \dots & 2^n-1 \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix}$$

[b]Such a Boolean function is usually called a *component* of S.

where the integers in the first two rows of the matrix correspond to n-bit vectors, and where each element in these two rows is an n-bit column vector. The Hamming weights of this code are

$$\{2^{n-1} - \frac{1}{2}\mathcal{E}(S_b + \varphi_a);\;\; 2^{n-1} + \frac{1}{2}\mathcal{E}(S_b + \varphi_a), a, b \in \mathbb{F}_2^n\}$$

The proof is similar to the proof of Proposition 13.12. Note that the case a and b equal to zero should be included in order to take into account the words of $R(1, n)$. Based on some relationships satisfied by the weight distribution of a code with such parameters, we can deduce a lower bound on the linearity of an Sbox.

Proposition 13.16. *[CV95, CCZ98] Let S be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$. Then*

$$\mathcal{L}(S) \geq 2^{\frac{n+1}{2}} .$$

The Sboxes for which equality holds are called almost bent (AB) functions*. They exist for n odd only.*

When n is even, almost bent functions do not exist, and the lowest possible linearity for an n-bit Sbox is not known. But, the best known value is $\mathcal{L}(S) = 2^{\frac{n}{2}+1}$, and this value is tight for a very few families of Sboxes, including the inversion over $\mathbb{F}_{2^n}$ which is used in the AES.

13.5.5 Bias of a two-round linear trail for the AES.

We now estimate the probability of a linear trail over two rounds of an SPN, like the AES. More precisely, we consider a block cipher with block size $n = tm$ whose nonlinear layer consists of t copies of the same Sbox S over $\mathbb{F}_2^m$. The linear layer then corresponds to the multiplication by an $n \times n$ binary matrix M. Since it does not influence the probability of linear approximations, the last linear layer is removed from the cipher we consider here. The AES fits this general model. More precisely, the linear layer in the AES corresponds to the composition of two functions, `ShiftRows` and `MixColumns`. But `ShiftRows` commutes with the nonlinear layer, implying that two rounds of the AES without the last linear layer, can be seen (up to an initial `ShiftRows`, as the parallel application of four independent functions called the AES superbox (see Figure 13.7). Most properties of the AES regarding statistical attacks only depend on the properties of this superbox.

In the following, we denote by x the input of the SPN, by y the output after the nonlinear layer Sub, by z the output of the linear layer, and finally by u the output the second nonlinear layer (see Figure 13.8). If we search for two-round linear trails, we need to find two linear approximations of the nonlinear layers:

$$\alpha \cdot x + \beta \cdot \mathsf{Sub}(x) = 0 \text{ and } \delta \cdot z + \gamma \cdot \mathsf{Sub}(z) .$$

Moreover, we need to chain these approximations, i.e., the output mask β of the first approximation must be compatible with the input mask δ of the second approximation:

$$\beta \cdot y = \delta \cdot (My) .$$

This last condition can be expressed on the linear masks as follows.

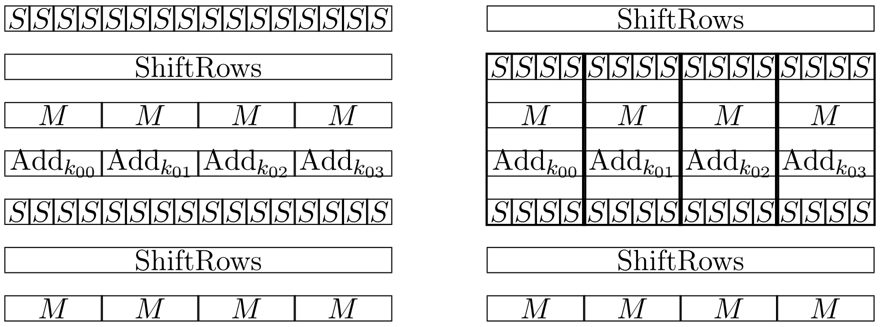


Figure 13.7: Two equivalent representations of two rounds of the AES (without the second key addition): the usual representation on the left, and the representation with superboxes on the right.

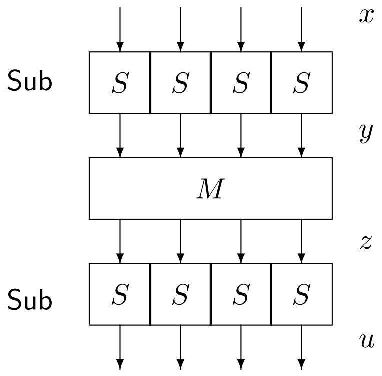


Figure 13.8: Notation for all intermediate variables involved in the AES superbox.

Lemma 13.17. *Two n-bit masks a and b satisfy*

$$a \cdot (Mx) = b \cdot x$$

for all $x \in \mathbb{F}_2^n$ if and only if

$$b = M^T a$$

where M^T is the transpose of M.

Proof. Let M_{ij} denote the coefficients of matrix M. Then, we have

$$\begin{aligned} a \cdot (Mx) &= \sum_{i=1}^{n} a_i (Mx)_i \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} a_i M_{ij} x_j \\ &= \sum_{j=1}^{n} x_j \left(\sum_{i=1}^{n} a_i M_{ij} \right) = (M^T a) \cdot x \ . \end{aligned}$$

◇

Now, if the two rounds are independent, we get by the piling-up lemma that the overall bias of the two-round linear trail is

$$\begin{aligned}\varepsilon &= 2^{-n}\mathcal{E}(\mathsf{Sub}_\beta + \varphi_\alpha) \times 2^{-n}\mathcal{E}(\mathsf{Sub}_\gamma + \varphi_\delta)\\ &= 2^{-n}\prod_{i=1}^{t}\mathcal{E}(S_{\beta_i} + \varphi_{\alpha_i}) \times 2^{-n}\prod_{i=1}^{t}\mathcal{E}(S_{\gamma_i} + \varphi_{\delta_i})\ ,\end{aligned}$$

where all linear masks are decomposed into t m-bit words. Now, for any permutation S, if only one of the masks a and b is zero, we obviously get that $\mathcal{E}(S_b + \varphi_a) = 0$. Therefore, the bias of the two-round linear trail is zero unless α and β have the same support, and δ and γ have the same support. If the two masks are zero, $\mathcal{E}(S_b + \varphi_a) = 2^m$. Then, only the Sboxes of index i with $i \in \mathrm{Supp}(\beta)$ (resp. in $\mathrm{Supp}(\delta)$) are involved in the computation of ε. These Sboxes are called *active Sboxes*. Indeed, we have

$$\begin{aligned}\varepsilon &= \prod_{i\in\mathrm{Supp}(\beta)}(2^{-m}\mathcal{E}(S_{\beta_i} + \varphi_{\alpha_i})) \times \prod_{i\in\mathrm{Supp}(\delta)}(2^{-m}\mathcal{E}(S_{\gamma_i} + \varphi_{\delta_i}))\\ &\le \left(\frac{\mathcal{L}(S)}{2^m}\right)^{wt(\beta)} \times \left(\frac{\mathcal{L}(S)}{2^m}\right)^{wt(\delta)}\\ &\le \left(\frac{\mathcal{L}(S)}{2^m}\right)^{wt(M^T\delta)+wt(\delta)}\ .\end{aligned}$$

It follows that this upper bound on the bias of any two-round linear trail for an SPN is minimized if:

- the Sbox has a low linearity, i.e., $\mathcal{L}(S)$ is as small as possible;
- for any nonzero mask δ, $wt(M^T\delta) + wt(\delta)$ is as high as possible.

This second condition is captured by the notion of *linear branch number* of the linear layer introduced by Daemen [Dae95].

Definition 13.18 (Linear branch number)**.** *The* linear branch number *with respect to* $\mathbb{F}_2^m$ *of the linear function* $x \mapsto Mx$ *over* $\mathbb{F}_2^{mt}$ *is*

$$\min_{x\neq 0}(wt(M^T x) + wt(x))\ .$$

Let $\mathcal{C}_M$ *denote the* $[2t, t]$*-linear code over* $\mathbb{F}_2^m$ *defined by all vectors* $(M^T x, x)$*. Then, the linear branch number of* M *is the minimum distance of* $\mathcal{C}_M$*.*

Using the coding viewpoint, we directly derive an upper bound of the linear branch number from Singleton's bound.

Proposition 13.19. *The* linear branch number *of the linear function* $x \mapsto Mx$ *over* $\mathbb{F}_2^{mt}$ *with respect to* $\mathbb{F}_2^m$ *is at most* $(t+1)$*. The functions* M *with maximal linear branch number are called MDS since the corresponding codes* $\mathcal{C}_M$ *are MDS.*

Most notably, the `MixColumns` transformation used in the AES is a permutation over $(\mathbb{F}_2^8)^4$. Its linear branch number with respect to $\mathbb{F}_2^8$ (the Sbox alphabet) is maximal and equal to 5. Therefore, the building-blocks of the AES have been chosen such that the upper bound on two-round linear trails is minimized:

$$\varepsilon \le \left(\frac{2^5}{2^8}\right)^5 = 2^{-15}\ .$$

13.6 Differential cryptanalysis

13.6.1 Principle

Differential cryptanalysis has been introduced by Biham and Shamir [BS91]. It exploits as a distinguishing property the existence of a *differential*, i.e., of a pair of differences (a, b) in $\mathbb{F}_2^n$ such that

$$p = \mathrm{Pr}_X[E_k(X + a) + E_k(X) = b]$$

is high. For a randomly chosen permutation, the derivative $X \mapsto E_k(X + a) + E_k(X)$ is expected to take every possible nonzero value with uniform probability. It follows that a differential can be used as a distinguisher if the corresponding probability p is significantly different from 2^{-n}. More precisely, the number of input-output pairs required for distinguishing a cipher from a random permutation based on a differential with probability p is (see e.g [BGT11])

$$\mathcal{O}\left(\frac{1}{p}\right) .$$

As for linear cryptanalysis, this distinguisher may be exploited either for the whole encryption function, or for the reduced cipher.

13.6.2 Finding good differential characteristics

A *differential characteristic* (aka *differential path*) for r rounds is a series of $(r + 1)$ differences $\Omega = (a_0, a_1, \ldots, a_r)$ where a_i corresponds to the difference obtained after the i-th round when encrypting two inputs which differ from a_0. The probability of the r-round differential characteristic Ω is then defined as

$$p(\Omega) = \mathrm{Pr}_{X_0}[X_1 + X_1' = a_1; \ldots ; X_r + X_r' = a_r \,|\, X_0 + X_0' = a_0] ,$$

where X_i (resp. X_i') denotes the image of X_0 (resp. of X_0') after the i-th round of E_k. A cipher is said to be a *Markov cipher* [LMM91] when the difference between the outputs of the i-th round depends on the difference between the outputs of the $(i-1)$-th round only. In this case, we get

$$\begin{aligned} p(\Omega) &= \mathrm{Pr}_{X_0}(X_1 + X_1' = a_1; \ldots ; X_r + X_r' = a_r \,|\, X_0 + X_0' = a_0) \\ &= \prod_{i=1}^{r} \mathrm{Pr}_{X_i}[F_{k_i}(X_i + a_{i-1}) + F_{k_i}(X_i) = a_i] , \end{aligned}$$

where F denotes the round permutation of the cipher. The Markovian hypothesis holds for instance when the round keys are independent and uniformly distributed. But this condition on the round keys is usually not satisfied since the round keys are related to each other by the key scheduling algorithm. However, the product of the expected probabilities of the successive one-round characteristics usually provides a good estimate of the expected probability of an r-round differential characteristic.

13.6.3 Probability of a two-round differential characteristic for the AES

We now want to upper-bound the probability of a differential characteristic for two rounds of an SPN cipher like the AES. We use the same notation as in Section 13.5.5 (see Figure 13.8) and focus on the AES superbox. We denote by α and β the input and output differences of the first nonlinear layer, and by δ and γ the input and output differences of the second nonlinear layer. Obviously, the linear layer $x \mapsto Mx$ propagates the differences with probability one in the sense that

$$\Pr_Y[M(Y+\beta)+MY=\delta] = \begin{cases} 1 & \text{if } \delta = M\beta \\ 0 & \text{otherwise.} \end{cases}$$

Using that the superbox is a Markov cipher (for a random round-key), the probability of the differential trail $\Omega = (\alpha, M\beta, \gamma)$ over the superbox is then defined by

$$\begin{aligned} p(\Omega) &= \Pr[\mathsf{Sub}(x+\alpha)+\mathsf{Sub}(x)=\beta] \times \Pr[\mathsf{Sub}(z+M\beta)+\mathsf{Sub}(z)=\gamma] \\ &= \left(\prod_{i=1}^{t} \Pr[S(x_i+\alpha_i)+S(x_i)=\beta_i]\right) \times \left(\prod_{i=1}^{t} \Pr[S(z_i+(M\beta)_i)+S(z_i)=\gamma_i]\right) . \end{aligned}$$

Obviously, since S is permutation, if only one value among the input and output differences, a and b, is zero, we get that $\Pr[S(x+a)+S(x)=b]=0$, while if both values are equal to zero, this probability is equal to one. This implies that $p(\Omega)=0$ unless α and β (resp. $M\beta$ and γ) have the same support. If this condition holds, we get

$$p(\Omega) = \left(\prod_{i\in\mathrm{Supp}(\beta)} \Pr[S(x_i+\alpha_i)+S(x_i)=\beta_i]\right) \times \left(\prod_{i\in\mathrm{Supp}(M\beta)} \Pr[S(z_i+(M\beta)_i)+S(z_i)=\gamma_i]\right)$$

A relevant parameter is then the maximal probability that a given nonzero input difference leads to a given output difference for the Sbox. This quantity is determined by the *differential uniformity* of S.

Definition 13.20 (Differential uniformity [Nyb93]). *Let S be a function from $\mathbb{F}_2^m$ into $\mathbb{F}_2^m$. For any a and b in $\mathbb{F}_2^m$, we define*

$$\delta(a,b) = |\{x \in \mathbb{F}_2^m, S(x+a)+S(x)=b\}| .$$

Then

$$\delta_S = \max_{a\neq 0,b} \delta(a,b)$$

is the differential uniformity *of S.*

The values $(\delta(a,b))_{a,b\in\mathbb{F}_2^m}$ are usually represented as a two-dimensional array called the difference table of S. It follows that, any differential characteristic Ω over the AES superbox satisfies

$$p(\Omega) \leq \left(\frac{\delta(S)}{2^m}\right)^{wt(\beta)+wt(M\beta)} .$$

This upper-bound is then minimized if:

- the Sbox has a low differential uniformity, i.e., $\delta(S)$ is as small as possible;
- for any nonzero mask β, $wt(\beta)+wt(M\beta)$ is as high as possible.

This second condition is captured by the notion of *differential branch number* of the linear layer [Dae95].

Definition 13.21 (Differential branch number)**.** *The* differential branch number *with respect to $\mathbb{F}_2^m$ of the linear function $x \mapsto Mx$ over $\mathbb{F}_2^{mt}$ is*

$$\min_{x \neq 0}(wt(x) + wt(Mx)) \, .$$

Let $\mathcal{C}_M^\perp$ denote the $[2t, t]$-linear code over $\mathbb{F}_2^m$ defined by all vectors (x, Mx). Then, the differential branch number of M is the minimum distance of $\mathcal{C}_M^\perp$.

Proposition 13.22. *The codes $\mathcal{C}_M$ and $\mathcal{C}_M^\perp$ defining the linear and differential branch numbers are dual to each other. In particular, the differential branch number of the linear function $x \mapsto Mx$ over $\mathbb{F}_2^{mt}$ with respect to $\mathbb{F}_2^m$ is maximal (and equal to $(t+1)$) if and only if the linear branch number of M is maximal.*

Proof. Any codeword in $\mathcal{C}_M^\perp$ is of the form (x, Mx) for some $x \in \mathbb{F}_2^{mt}$ while any codeword in $\mathcal{C}_M$ is of the form $(M^T y, y)$ for some $y \in \mathbb{F}_2^{mt}$. The scalar product between two such words always vanish. Indeed,

$$\begin{aligned} (x, Mx) \cdot (M^T y, y) &= \sum_{i=1}^t x_i (Mx)_i + \sum_{i=1}^t (M^T y)_i y_j \\ &= \sum_{i=1}^t x_i \left(\sum_{j=1}^t M_{ji} y_j \right) + \sum_{i=1}^t y_i \left(\sum_{j=1}^t M_{ij} x_j \right) = 0 \, . \end{aligned}$$

This implies that $\mathcal{C}_M^\perp$ is the dual of $\mathcal{C}_M^\perp$. The result then follows from the fact that the dual of an MDS code is MDS (see e.g. [MS77, Page 318]). ◇

In other words, we have proved that the linear layers which guarantee the best resistance to linear cryptanalysis also guarantee the best resistance to differential cryptanalysis. It is worth noticing that this correspondence only holds when the branch numbers are maximal: for instance, a differential branch number equal to t does not imply that the linear branch number equals t.

13.6.4 Differential uniformity of Sboxes

Proposition 13.23 ([NK93])**.** *Let S be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$. Then, its differential uniformity satisfies*

$$\delta(S) \geq 2 \, .$$

The Sboxes for which equality holds are called almost perfect nonlinear (APN) *functions.*

Proof. The proof comes from the fact that

$$\delta(a, b) = |\{x \in \mathbb{F}_2^m, S(x + a) + S(x) = b\}|$$

is always even since, if x is a solution of this equation, $(x + a)$ is a solution too. ◇

The *APN* terminology comes from the fact we call *perfect nonlinear* the functions from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$ such that $\delta(S) = 2^{n-m}$. It can be proved that the perfect nonlinear functions correspond to the functions with linearity $\mathcal{L}(S) = 2^{\frac{n}{2}}$, i.e., such that all their components are bent. Such functions exist only when $n \geq 2m$ [Nyb91]. Therefore, when $m = n$, the optimal differential uniformity is obtained for the almost perfect nonlinear functions.

Link with Reed-Muller codes. The APN property is related to the minimum distance of the dual of the code Γ_S defined in Proposition 13.15.

Proposition 13.24 ([CCZ98])**.** *Let S be an Sbox from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$. Let us consider the code Γ_S having for generator matrix the following $(2n+1) \times 2^n$-matrix*

$$G = \begin{pmatrix} S(0) & S(1) & S(2) & \dots & S(2^n-1) \\ 0 & 1 & 2 & \dots & 2^n-1 \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix}$$

as in Proposition 13.15. Then, the minimum distance of $\Gamma_S^\perp$ equal to 6 if S is APN and equal to 4 otherwise.

Proof. It can be easily checked that the minimum distance of $\Gamma_S^\perp$ is even (since Γ_S contains the all-one codeword), and at least 4. Moreover, it cannot be greater than 6 [DZ84, BT93]. Now, a codeword of weight 4 of $\Gamma_S^\perp$ corresponds to four columns of the generator matrix G which sum to zero (since the words of $\Gamma_S^\perp$ are the column vectors c such that $Gc = 0$). Therefore, $\Gamma_S^\perp$ has minimum distance 4 if and only if there exist four distinct elements x_1, x_2, x_3, x_4 such that

$$x_1 + x_2 + x_3 + x_4 = 0 \text{ and } S(x_1) + S(x_2) + S(x_3) + S(x_4) = 0 \ .$$

Replacing x_2 by $x_1 + a$, we get that it is equivalent to the existence of x_1, x_3 and a such that

$$S(x_1) + S(x_1 + a) = S(x_3) + S(x_3 + a)$$

since $x_4 = x_3 + (x_1 + x_2) = x_3 + a$. All four x_i are distinct if and only if they correspond to four distinct solutions of the equation

$$S(x + a) + S(x) = b$$

with $b = S(x_1) + S(x_1 + a)$. This equivalently means that S is not APN. $\diamond$

The APN and AB properties then correspond to a particular value of the minimum distance of the code $\Gamma_S^\perp$ and Γ_S. Using the relations between the weight distributions of the two codes, it can be deduced that these two properties are related as follows.

Proposition 13.25 ([CV95, CCZ98])**.** *Let S be an Sbox from $\mathbb{F}_2^n$ into $\mathbb{F}_2^n$ and Γ_S be the $[2^n, 2n+1]$-linear code defined in Proposition 13.15. Then, if the minimum distance of Γ_S is maximal (i.e., equal to $2^{n-1} - 2^{\frac{n-1}{2}}$), then the minimum distance of $\Gamma_S^\perp$ is maximal (i.e., equal to 6).*

An equivalent formulation of the previous proposition is that any almost bent Sbox is also APN. But the converse does not hold in general, except for Sboxes of degree 2 [CCZ98].

However, the previous result is relevant when n is odd only, otherwise almost bent functions do not exist. Also, it has been conjectured for a long time that APN permutations of an even number of variables did not exist. This has been disproved by Dillon in 2009 who has exhibited an APN permutation of 6 variables [BDMW10]. But this is (up to composition by affine transformations) the only known example of APN permutation of an even number of variables. In particular, no APN permutation of eight variables is known so far. It follows that, for a permutation depending on an even number of variables n, $n \geq 8$, the best known linearity is $\mathcal{L}(S) = 2^{\frac{n}{2}+1}$ and the best differential uniformity is $\delta(S) = 4$. These two values are reached by the inversion over $\mathbb{F}_{2^n}$ which is used in the AES. It is worth noticing that, when n is divisible by 8, the inversion is the only known permutation of n variables (up to composition by affine transformations) which reaches these two values.

Link with cyclic codes. Among all Sboxes, an interesting case is the family of power permutations (i.e., monomial functions), $S(x) = x^s$ over $\mathbb{F}_{2^n}$. Then, by removing the all-one codeword from the generator matrix of Γ_S and puncturing the resulting code by removing the first column, we get a $[2^n - 1, 2n]$-code Γ_S with generator matrix

$$\begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{2^n-2} \\ 1 & \alpha^s & \alpha^{2s} & \dots & \alpha^{(2^n-2)s} \end{pmatrix},$$

This is the parity-check matrix of a cyclic code of length $(2^n - 1)$ with defining set $\{1, s\}$. Therefore, the search for almost bent power functions boils down to the search for cyclic codes with two zeroes with the highest possible minimum distance and dual distance. For these reasons, the main families of almost bent power functions have been exhibited in some work on cyclic codes. The list of all known almost bent power permutations of n variables, n odd, is given in Table 13.3.

	exponent s	
Quadratic exponents	$2^i + 1$ with $\gcd(i, n) = 1$, $1 \le i \le t$	[Gol68, Nyb93]
Kasami exponents	$2^{2i} - 2^i + 1$ with $\gcd(i, n) = 1$ $2 \le i \le t$	[Kas71]
Welch exponent	$2^t + 3$	[Dob99b, CCD00]
Niho exponent	$2^t + 2^{\frac{t}{2}} - 1$ is t is even $2^t + 2^{\frac{3t+1}{2}} - 1$ if t is odd	[Dob99a, HX01]

Table 13.3: Known almost bent power permutations $S : x \mapsto x^s$ over $\mathbb{F}_{2^n}$ with $n = 2t + 1$

Bibliography

[BBL13] Céline Blondeau, Andrey Bogdanov, and Gregor Leander. Bounds in Shallows and in Miseries. In *Advances in Cryptology - CRYPTO 2013 (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 204–221. Springer, 2013.

[BDMW10] K.A. Browning, J.F. Dillon, M.T. McQuistan, and A.J. Wolfe. An APN permutation in dimension six. In *Finite Fields: Theory and Applications*, volume 518 of *Contemporary Mathematics*, pages 33–42. AMS, 2010.

[BGT11] Céline Blondeau, Benoît Gérard, and Jean-Pierre Tillich. Accurate estimates of the data complexity and success probability for various cryptanalyses. *Designs, Codes and Cryptography*, 59(1-3):3–34, 2011.

[BR05] Mihir Bellare and Phillip Rogaway. Modern Cryptography - Chapter 3: Pseudorandom functions. `http://cseweb.ucsd.edu/~mihir/cse207/`, 2005.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[BT93] A.E. Brouwer and L.M.G.M. Tolhuizen. A sharpening of the Johnsson bound for binary linear codes and the nonexistence of linear codes with Preparata parameters. *Designs, Codes and Cryptography*, 3(2):95–98, 1993.

[BW99] Alex Biryukov and David Wagner. Slide Attacks. In *Fast Software Encryption - FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[CAA$^+$08] Carlos Cid, Martin Albrecht, Daniel Augot, Anne Canteaut, and Ralf-Philipp Weinmann. D.STVL.7 - algebraic cryptanalysis of symmetric primitives. Report of the ECRYPT European Network of Excellence, July 2008. `https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/dstvl7.pdf`.

[CCD00] Anne Canteaut, Pascale Charpin, and Hans Dobbertin. Binary m-sequences with three-valued crosscorrelation: A proof of welch's conjecture. *IEEE Transactions on Information Theory*, 46(1):4–9, 2000.

[CCZ98] Claude Carlet, Pascale Charpin, and Victor Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998.

[Cid04] Carlos Cid. Some Algebraic Aspects of the Advanced Encryption Standard. In *Advanced Encryption Standard - AES 2004*, volume 3373 of *Lecture Notes in Computer Science*, pages 58–66. Springer, 2004.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer-Verlag, 2000.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology - ASIACRYPT'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.

[CV95] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. In *Advances in Cryptology - EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 356–365. Springer-Verlag, 1995.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic programming. *Journal of Symbolic Computation*, (9):251–280, 1990.

[Dae95] Joan Daemen. *Cipher and Hash Function Design. Strategies based on linear and differential cryptanalysis.* PhD thesis, Katholieke Universiteit Leuven, 1995.

[Dob94] Hans Dobbertin. Construction of bent functions and balanced Boolean functions with high nonlinearity. In *Fast Software Encryption - FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 61–74. Springer-Verlag, 1994.

[Dob99a] Hans Dobbertin. Almost perfect nonlinear power functions on $GF(2^n)$: the Niho case. *Information and Computation*, 151(1-2):57–72, 1999.

[Dob99b] Hans Dobbertin. Almost perfect nonlinear power functions on $GF(2^n)$: the Welch case. *IEEE Transactions on Information Theory*, 45(4):1271–1275, 1999.

[DR01] Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. In *IMA International Conference - Coding and Cryptography 2001*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *Journal of Mathematical Cryptology*, 1(3):221–242, 2007.

[DR09] Joan Daemen and Vincent Rijmen. New criteria for linear maps in AES-like ciphers. *Cryptography and Communications*, 1(1):47–69, 2009.

[DZ84] Stefan M. Dodunekov and Victor Zinoviev. A note on Preparata codes. In *Proceedings of the 6th Intern. Symp. on Information Theory, Moscow-Tashkent Part 2*, pages 78–80, 1984.

[EM93] Shimon Even and Yishay Mansour. A Construction of a Cipher From a Single Pseudorandom Permutation. In *Advances in Cryptology - ASIACRYPT'91*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 1993.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.

[Fau02] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*. ACM, 2002.

[FIP99] FIPS 46-3. Data Encryption Standard. Federal Information Processing Standards Publication 46-3, 1999. U.S. Department of Commerce/National Bureau of Standards.

[FIP01] FIPS 197. Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. U.S. Department of Commerce/N.I.S.T.

[GC91] Henri Gilbert and Guy Chassé. A Statistical Attack of the FEAL-8 Cryptosystem. In *Advances in Cryptology - CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 22–33. Springer-Verlag, 1991.

[Gol68] Robert Gold. Maximal recursive sequences with 3-valued recursive crosscorrelation functions. *IEEE Transactions on Information Theory*, 14:154–156, 1968.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew Robshaw. The LED Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[Har96] Carlo Harpes. *Cryptanalysis of iterated block ciphers*, volume 7 of *ETH Series in Information Processing.* Hartung-Gorre Verlag, Konstanz, 1996.

[Hou93] Xiang-Dong Hou. Further results on the covering radii of the Reed-Muller codes. *Designs, Codes and Cryptography*, 3:167–177, 1993.

[Hou96a] Xiang-Dong Hou. Covering radius of the Reed-Muller code $R(1,7)$ - a simpler proof. *Journal of Combinatorial Theory, Series A*, (74):337–341, 1996.

[Hou96b] Xiang-Dong Hou. On the covering radius of $R(1,m)$ in $R(3,m)$. *IEEE Transactions on Information Theory*, 42(3):1035–1037, 1996.

[HX01] H.D.L. Hollmann and Q. Xiang. A proof of the Welch and Niho conjectures on crosscorrelations of binary m-sequences. *Finite Fields and their Applications*, 7(2):253–286, 2001.

[Kas71] Tadao Kasami. The weight enumerators for several classes of subcodes of the second order binary Reed-Muller codes. *Information and Control*, 18:369–394, 1971.

[KMY07] Selçuk Kavut, Subhamoy Maitra, and Melek D. Yücel. Search for Boolean Functions With Excellent Profiles in the Rotation Symmetric Class. *IEEE Transactions on Information Theory*, 53(5):1743–1751, 2007.

[KR11] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion.* Information Security and Cryptography. Springer, 2011.

[Kuk99] Zsolt Kukorelly. *On the validity of certain hypotheses used in linear cryptanalysis*, volume 13 of *ETH Series in Information Processing.* Hartung-Gorre Verlag, Konstanz, 1999.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *Advances in Cryptology - EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer-Verlag, 1991.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.

[Mat95] Mitsuru Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In *Advances in Cryptology - CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science.* Springer-Verlag, 1995.

[Mat97] Mitsuru Matsui. New Block Encryption Algorithm MISTY. In *Fast Software Encryption - FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 1997.

[MS77] F. Jessie MacWilliams and Neil J.A. Sloane. *The theory of error-correcting codes.* North-Holland, 1977.

[Myk80] Johannes Mykkeltveit. The covering radius of the (128,8) Reed-Muller code is 56. *IEEE Transactions on Information Theory*, IT-26(3):359–362, 1980.

[NK93] Kaisa Nyberg and Lars R. Knudsen. Provable security against differential cryptanalysis. In *Advances in Cryptology - CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 566–574. Springer-Verlag, 1993.

[NK95] Kaisa Nyberg and Lars R. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1):27–37, 1995.

[Nyb91] Kaisa Nyberg. Perfect nonlinear S-boxes. In *Advances in Cryptology - EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 378–385. Springer-Verlag, 1991.

[Nyb93] Kaisa Nyberg. Differentially uniform mappings for cryptography. In *Advances in Cryptology - EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer-Verlag, 1993.

[Nyb95] Kaisa Nyberg. Linear Approximation of Block Ciphers. In *Advances in Cryptology - EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Nyb96] Kaisa Nyberg. Generalized Feistel Networks. In *Advances in Cryptology - ASIACRYPT'96*, volume 1163 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1996.

[Pat04] Jacques Patarin. Security of Random Feistel Schemes with 5 or More Rounds. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2004.

[PW83] Nick J. Patterson and Douglas H. Wiedemann. The covering radius of the $[2^{15}, 16]$ Reed-Muller code is at least 16276. *IEEE Transactions on Information Theory*, IT-36(2):443, 1983.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.

[SK96] Bruce Schneier and John Kelsey. Unbalanced Feistel networks and block cipher design. In *Fast Software Encryption - FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer-Verlag, 1996.

[Ste04] Allan Steel. Allan Steel's Gröbner basis timings page, 2004. `http://magma.maths.usyd.edu.au/users/allan/gb/`.

[TCG91] Anne Tardy-Corfdir and Henri Gilbert. A known plaintext attack of FEAL-4 and FEAL-6. In *Advances in Cryptology - CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 172–182. Springer-Verlag, 1991.

[Vau99] S. Vaudenay. Vers une théorie du chiffrement symétrique. Thèse d'Habilitation, Université Paris 7, 1999.