3.8 Correlation Attacks—the Achilles Heels of Combiners

Let $f\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ be the combining function of a nonlinear combiner. The number

$$K_f := \#\{x = (x_1, \ldots, x_n) \in \mathbb{F}_2^n \mid f(x) = x_1\}$$

counts the coincidences of the value of the function with its first argument. If it is $> 2^{n-1}$, then the probability of a coincidence,

$$p = \frac{1}{2^n} \cdot K_f > \frac{1}{2},$$

is above average, and the combined output sequence "correlates" with the output of the first LFSR more then expected by random. If $p < \frac{1}{2}$, then the correlation deviates from the expected value in the other direction.

The cryptanalyst can exploit this effect in an attack with known plaintext. We suppose that she knows the "hardware", that is the taps of the registers, and also the combining function f. She seeks the initial states of all the LFSRs. We assume she knows the bits $k_0, \ldots, k_{r-1}$ of the key stream. For each of the 2^{l_1} initial states of the first LFSR she generates the sequence $u_0, \ldots, u_{r-1}$, and counts the coincidences. The expected values are

$$\frac{1}{r} \cdot \#\{i \mid u_i = k_i\} \approx \begin{cases} p & \text{for the correct initial state of LFSR 1,} \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

If r is large enough, she can determine the true initial state of LFSR 1 (with high probability) for a cost of $\sim 2^{l_1}$. She continues with the other registers, and finally identifies the complete key with a cost of $\sim 2^{l_1} + \cdots + 2^{l_n}$. Note that the cost is exponential, but significantly lower than the cost $\sim 2^{l_1} \cdots 2^{l_n}$ of the naive exhaustion of the key space.

In the language of linear cryptanalysis from Part II she made use of the linear relation

$$f(x_1, \ldots, x_n) \overset{p}{\approx} x_1$$

for f. Clearly she could use any linear relation as well to reduce the complexity of key search. (A more in-depth analysis of the situation leads to the notion of correlation immunity that is related with the linear potential.)

Correlations from the Geffe generator

From the truth table 3.2 we get the correlations produced by the Geffe generator. Thus the probabilities of coincidences are

$$p = \begin{cases} \frac{3}{4} & \text{for register 1 } (x), \\ \frac{3}{4} & \text{for register 2 } (y), \\ \frac{1}{2} & \text{for register 3 } (z = \text{control bit}). \end{cases}$$

x	0	0	0	0	1	1	1	1
y	0	1	0	1	0	1	0	1
z	0	0	1	1	0	0	1	1
$f(x,y,z)$	0	0	0	1	1	1	0	1

Table 3.2: Truth table of the GEFFE function

linear form	0	z	y	$y+z$	x	$x+z$	$x+y$	$x+y+z$
representation	000	001	010	011	100	101	110	111
potential λ	0	0	1/4	1/4	1/4	1/4	0	0
probability p	1/2	1/2	3/4	1/4	3/4	3/4	1/2	1/2

Table 3.3: Coincidence probabilities of the GEFFE function

A correlation attack easily detects the initial states of registers 1 and 2—the battery registers—given only a short piece of an output sequence. Afterwards exhaustion finds the initial state of register 3, the control register.

We exploit this weakness of the GEFFE generator for an attack in Sage sample 3.6 that continues Sage sample 3.2. Since we defined the linear profile for objects of the class `BoolMap` only, we first of all have to interpret the function `geff` as a Boolean map, that is a one-element list of Boolean functions. Then the linear profile is represented by a matrix of 2 columns and 8 rows. The first column `[64, 0, 0, 0, 0, 0, 0, 0]` shows the coincidences with the linear form 0 in the range. So it contains no useful information, except the denominator 64 that applies to all entries. The second row `[0, 0, 16, 16, 16, 16, 0, 0]` yields the list of coincidence probabilities p (after dividing it by 64) in Table 3.3, using the formula

$$p = \frac{1}{2} \cdot (\pm\sqrt{\lambda} + 1).$$

If $\lambda = 0$, then $p = 1/2$. If $\lambda = 1/4$, then $p = 1/4$ or $3/4$. For deciding between these two values for p we use Table 3.2.

Sage Example 3.6 Linear profile of the Geffe function

```
sage: g = BoolMap([geff])
sage: linProf = g.linProf(); linProf
[[64,0], [0,0], [0,16], [0,16], [0,16], [0,16], [0,0], [0,0]]
```

In Sage sample 3.7 we apply this finding to the 100 element sequence from Sage sample 3.5. The function `coinc` from the Sage module `Bitblock.sage` in Appendix E.1 of Part II counts the coincidences. For the first register we

find 73 coincidences, for the second one 76, for the third one only 41. This confirms the values 75, 75, 50 predicted by our theory.

Sage Example 3.7 Coincidences for the Geffe generator

```
sage: coinc(outlist15,outlist)
73
sage: coinc(outlist16,outlist)
76
sage: coinc(outlist17,outlist)
41
```

Cryptanalysis of the Geffe Generator

These results promise an effortless analysis of our sample sequence. For an assessment of the success probability we consider a bitblock $b \in \mathbb{F}_2^r$ and first ask how large is the probability that a random bitblock $u \in \mathbb{F}_2^r$ coincides with b at exactly t positions. For an answer we have to look at the symmetric binomial distribution (where $p = \frac{1}{2}$ is the probability of coincidence at a single position): The probability of exactly t coincidences is

$$B_{r,\frac{1}{2}}(t) = \frac{\binom{r}{t}}{2^r}.$$

Hence the cumulated probability of up to T coincidences is

$$\sum_{t=0}^{T} B_{r,\frac{1}{2}}(t) = \frac{1}{2^r} \cdot \sum_{t=0}^{T} \binom{r}{t}.$$

If r is not too large, then we may explicitly calculate this value for a given bound T. If on the other hand r is not too small, then we approximate the value using the normal distribution. The mean value of the number of coincidences is $r/2$, the variance, $r/4$, and the standard deviation, $\sqrt{r}/2$.

In any case for $r = 100$ the probability of finding at most (say) 65 coincidences is 0.999, the probability of surpassing this number is 1 ‰. For the initial state of register 1 we have to try $2^{15} = 32786$ possibilities (generously including the zero state $0 \in \mathbb{F}_2^{15}$ into the count). So we expect about 33 oversteppings with at least 66 coincidences. One of these should occur for the true initial state of register 1 that we expect to produce about 75 coincidences. Maybe it even produces the maximum number of coincidences.

Sage sample 3.8 shows that this really happens. However the maximum number of coincidences, 73, occurs twice in the histogram. The first occurrence happens at index 13705, corresponding to the initial state 011010110001001, the correct solution. The second occurrence, at index

Sage Example 3.8 Analysis of the Geffe generator—register 1

```
sage: clist = []
sage: histogr = [0] * (nofBits + 1)
sage: for i in range(0,2**15):
....:     start = int2bbl(i,15)
....:     reg15.setState(start)
....:     testlist = reg15.nextBits(nofBits)
....:     c = coinc(outlist,testlist)
....:     histogr[c] += 1
....:     clist.append(c)
....:
sage: print(histogr)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 12, 12, 37, 78, 116, 216,
 329, 472, 722, 1003, 1369, 1746, 1976, 2266, 2472, 2531, 2600,
 2483, 2355, 2149, 1836, 1574, 1218, 928, 726, 521, 343, 228, 164,
 102, 60, 47, 36, 13, 8, 7, 4, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
sage: mm = max(clist)
sage: ix = clist.index(mm)
sage: block = int2bbl(ix,15)
sage: print("Maximum =", mm, "at index", ix, ", start value", block)
Maximum = 73 at index 13705 , start value\
 [0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1]
```

Sage Example 3.9 Analysis of the Geffe generator—continued

```
sage: ix = clist.index(mm,13706); ix
31115
sage: print(int2bbl(ix,15))
[1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1]
```

31115, see Sage sample 3.9, yields the false solution 111100110001011 that eventually leads to a contradiction.

Sage sample 3.10 shows the analogous analysis of register 2. Here the maximum of coincidences, 76, is unique, occurs at index 27411 corresponding to the initial state 0110101100010011, and provides the correct solution.

To complete the analysis we must yet determine the initial state of register 3, the control register. The obvious idea is to exhaust the 2^{17} different possibilities. There is a shortcut since we already know 51 of the first 100 bits of the control register: At a position where the values of registers 1 and

Sage Example 3.10 Analysis of the Geffe generator—register 2

```
sage: clist = []
sage: histogr = [0] * (nofBits + 1)
sage: for i in range(0,2**16):
....:     start = int2bbl(i,16)
....:     reg16.setState(start)
....:     testlist = reg16.nextBits(nofBits)
....:     c = coinc(outlist,testlist)
....:     histogr[c] += 1
....:     clist.append(c)
....:
sage: print(histogr)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 3, 4, 8, 17, 25, 51, 92, 171,
 309, 477, 750, 1014, 1423, 1977, 2578, 3174, 3721, 4452, 4821,
 5061, 5215, 5074, 4882, 4344, 3797, 3228, 2602, 1974, 1419,
 1054, 669, 434, 306, 174, 99, 62, 38, 19, 10, 3, 0, 1, 0, 0,
 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0]
sage: mm = max(clist)
sage: ix = clist.index(mm)
sage: block = int2bbl(ix,16)
sage: print("Maximum =", mm, "at index", ix, ", start value", block)
Maximum = 76 at index 27411 , start value\
 [0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1]
```

2 differ, the control bit is necessarily 0 if the final output coincides with register 1, and 1 otherwise. Only at positions where registers 1 and 2 coincide the corresponding bit of register 3 is undetermined.

```
 register 1: 10010001101011011100001001101101000001110110110000
 register 2: 11001000110101100011001111000000001110111000111000
 register 3: -1-00--0-1101-110001---00-1-00-1--1101--110---0---
bitsequence: 11010001110001101101001001001100001100111010110000

        ... 00101101101111111001001001010101110001110011001011
        ... 00100011101111010010011110010110111100101110010001
        ... ----110-------1-1-11-0-100----01--01-1-001-1-00-1-
        ... 00100001101111010010001101010100110100110110001001
```

In particular we already know 11 of the 17 initial bits, and are left with only $2^6 = 64$ possibilities to try.

$u_{17} = u_{14} + u_0$	$0 = 1 + u_0$	$u_0 = 1$	
$u_{19} = u_{16} + u_2$	$1 = 0 + u_2$	$u_2 = 1$	
$u_{20} = u_{17} + u_3$	$u_{20} = 0 + 0$	$u_{20} = 0$	
$u_{22} = u_{19} + u_5$	$u_{22} = u_5 + 1$	$u_5 = u_{22} + 1$	
$u_{23} = u_{20} + u_6$	$0 = u_{20} + u_6$	$u_6 = u_{20}$	$u_6 = 0$
$u_{25} = u_{22} + u_8$	$u_{25} = u_{22} + u_8$	$u_8 = u_{22} + u_{25}$	$u_8 = u_{22}$
$u_{27} = u_{24} + u_{10}$	$u_{27} = 0 + 1$	$u_{27} = 1$	
$u_{28} = u_{25} + u_{11}$	$0 = u_{25} + 0$	$u_{25} = 0$	
$u_{30} = u_{27} + u_{13}$	$u_{30} = u_{27} + u_{13}$	$u_{13} = u_{27} + u_{30}$	$u_{13} = u_{30} + 1$
$u_{33} = u_{30} + u_{16}$	$u_{33} = u_{30} + 0$	$u_{30} = u_{33}$	$u_{30} = 1$
$u_{36} = u_{33} + u_{19}$	$0 = u_{33} + 1$	$u_{33} = 1$	
$u_{39} = u_{36} + u_{22}$	$u_{39} = 0 + u_{22}$	$u_{22} = u_{39}$	
$u_{42} = u_{39} + u_{25}$	$0 = u_{39} + u_{25}$	$u_{39} = u_{25}$	$u_{39} = 0$

Table 3.4: Determination of the control register's initial state

But even this may be further simplified, since the known and the unknown bits obey linear relations of the type $u_n = u_{n-3} + u_{n-17}$. The unknown bits of the initial state are u_0, u_2, u_5, u_6, u_8, u_{13}. The solution follows the columns of Table 3.4, that immediately give

$$u_0 = 1,\ u_2 = 1,\ u_6 = 0.$$

The remaining solutions are

$$u_8 = u_{22} = u_{39} = 0,\ u_5 = u_{22} + 1 = u_8 + 1 = 1,\ u_{13} = u_{30} + 1 = 0.$$

Hence the initial state of the control register is `01101011000100111`, and we know this is the correct solution. We don't need to bother with the second possible solution for register 1 since we already found a constellation that correctly reproduces the sequence.