3.4 The BM Algorithm as a Cryptanalytic Tool

We revisit the cryptanalysis of an XOR ciphertext in Section 2.3 and explore how well the BM algorithm performs in this example following the cycle "construct – predict – adjust" as in Section 2.10. Remember the ciphertext:

```
10011100 10100100 01010110 10100110 01011101 10101110
01100101 10000000 00111011 10000010 11011001 11010111
00110010 11111110 01010011 10000010 10101100 00010010
11000110 01010101 00001011 11010011 01111011 10110000
10011111 00100100 00001111 01010011 11111101
```

For use with SageMath we provisionally fix its first 48 bits:

```
ciphtext = [1,0,0,1,1,1,0,0,1,0,1,0,0,1,0,0,0,1,0,1,0,
   1,1,0,1,0,1,0,0,1,1,0,0,1,0,1,1,1,0,1,1,0,1,0,1,1,1,0]
```

As in Section 2.3 we suspect that the cipher is XOR with a key stream from an LFSR, but now *of unknown length*. As before we guess that the text is in German and might begin with the word "Treffpunkt". To solve the cryptogram we need some bits of plaintext, say the first t letters (assumed in the 8-bit ISO 8859-1 character set), making up $8t$ bits of the key stream.

Let us tentatively start with two letters of plaintext: `Tr`, and the corresponding 16 keystream bits

```
          10011100 10100100  (ciphertext)
   Tr =   01010100 01110010  (assumed plaintext)
          -------- --------
          11001000 11010110  (keystream)
```

After attaching the Sage modules `Bitblock.sage`, `FSR.sage`, and `bmAlg.sage` from Appendix C (or Part II, Appendix E.1) we use the interactive commands

```
sage: kbits = [1,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0]
sage: res = bmAlg(kbits)
sage: fbpol = res[1]; fbpol
T^8 + T^7 + T^5 + T^4 + T^3 + T^2 + T + 1
```

This result tells us that the shortest LFSR that generates our 16 keystream bits has length 8 and the taps 1, 2, 3, 4, 5, 7, 8 set. Next we initialize this LFSR in SageMath (note the reverse order of the bits in the initial state):

```
sage: coeff = [1,1,1,1,1,0,1,1]
sage: reg = LFSR(coeff)
sage: start = [0,0,0,1,0,0,1,1]
sage: reg.setState(start)
```

Using this LFSR we predict 32 more, hence altogether 48 tentative keystream bits:

```
sage: testkey = reg.nextBits(48); testkey
[1,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0,1,0,0,1,1,1,0,0,
 0,0,0,0,0,1,1,0,0,0,0,1,0,1,1,1,0,1,1,1,1,0,0,0]
```

These tentative key bits yield 48 bits of experimental plaintext, represented by 6 bytes in decimal notation:

```
sage: testplain = xor(ciphtext,testkey)
sage: testtext = []
sage: for i in range(6):
    block = testplain[8*i:8*i+8]
    nr = bbl2int(block)
    testtext.append(nr)
sage: testtext
[84, 114, 202, 160, 74, 214]
```

or, written as ISO 8859-1 characters, “TrÊ⊔JÖ” (where ⊔ represents the non-breaking space)—a definitive failure.

So let us guess one more letter of plaintext: `Tre`, and use the corresponding 24 keystream bits

```
           10011100 10100100 01010110   (ciphertext)
    Tre =  01010100 01110010 01100101   (assumed plaintext)
           -------- -------- --------
           11001000 11010110 00110011   (keystream)
```

As above we apply the BM algorithm interactively and get an LFSR of length 12 with feedback polynomial $T^{12}+T^{10}+T^9+T^8+T^6+T^5+T^3+T+1$, hence taps $1, 3, 5, 6, 8, 9, 10, 12$. Setting up the LFSR and predicting 48 keystream bits:

```
sage: coeff = [1,0,1,0,1,1,0,1,1,1,0,1]
sage: reg = LFSR(coeff)
sage: start = [1,0,1,1,0,0,0,1,0,0,1,1]
sage: reg.setState(start)
sage: testkey = reg.nextBits(48); testkey
[1,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0,0,0,1,1,0,0,1,1,
 0,1,0,1,0,0,0,0,1,0,0,0,1,0,1,1,0,1,0,1,0,0,0,1]
```

we again get 48 bits of experimental plaintext, as bytes in decimal notation: `[84, 114, 101, 246, 214, 255]`. The translation to ISO 8859-1 yields the next flop: “TreöÖÿ”.

As next step we use four letters of known plaintext `Tref` (as in Section 2.3) and derive 32 tentative keystream bits:

```
         10011100 10100100 01010110 10100110  (ciphertext)
Tref =   01010100 01110010 01100101 01100110  (assumed plaintext)
         -------- -------- -------- --------
         11001000 11010110 00110011 11000000  (keystream)
```

The BM algorithm yields an LFSR of length 16 with feedback polynomial $T^{16} + T^5 + T^3 + T^2 + 1$, hence taps $2, 3, 5, 16$. It predicts 48 keystream bits:

```
sage: coeff = [0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,1]
sage: reg = LFSR(coeff)
sage: start = [0,1,1,0,1,0,1,1,0,0,0,1,0,0,1,1]
sage: reg.setState(start)
sage: testkey = reg.nextBits(48); testkey
[1,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0,0,0,1,1,0,0,1,1,
 1,1,0,0,0,0,0,0,0,0,1,1,1,0,1,1,1,0,0,0,1,1,1,0]
```

and the experimental plaintext `[84, 114, 101, 102, 102, 32]` that looks promising: "Treff⊔" (where ⊔ here represents simple space character).

Sure of victory we decipher the complete text:

```
sage: cstream = "10011100101...1111111101"
sage: fullcipher = str2bbl(cstream)
sage: start = [0,1,1,0,1,0,1,1,0,0,0,1,0,0,1,1]
sage: reg.setState(start)
sage: keystream = reg.nextBits(232)
sage: fullplain = xor(fullcipher,keystream)
sage: fulltext = []
sage: for i in range(232/8):
    block = fullplain[8*i:8*i+8]
    nr = bbl2int(block)
    fulltext.append(nr)
sage: fulltext
[84,114,101,102,102,32,109,111,114,103,101,110,32,56,32,85,104,114,
 32,66,97,104,110,104,111,102,32,77,90]
 T  r   e   f   f   _  m   o   r   g   e   n   _  8  _  U  h   r
 _  B  a  h   n   h   o   f   _  M  Z
```

"Meeting tomorrow at 8 p. m. train station Mainz".

Remark

The success of this cryptanalytic approach crucially depends on the LFSR scenario, or in other words on a linearity profile like that in Figure 3.3 for the keystream. If the keystream comes from another kind of source we expect a linearity profile as in Figure 3.4 and shall not be able to make a stable prediction before the plaintext is exhausted.

We could also try nonlinear FSRs in an analoguous way as in Appendix B. Unfortunately most trials—even if the recursive profile stabilizes—will find a trivial FSR that allows no prediction beyond the end of the already known partial key sequence, see Appendix B. Then the approach "construct – predict – adjust" cannot work better than by guessing more keystream bits in a purely random way.