Example 61. Eve intercepts the ciphertext $c = (1111\ 1011\ 0000)_2$ from Alice to Bob. She knows that the plaintext begins with $m = (1100\ 0...)_2$. Eve thinks a stream cipher using a LFSR with $x_{n+3} \equiv x_{n+2} + x_n \pmod{2}$ was used. If that's the case, what is the plaintext?

Solution. The initial piece of the keystream is $\mathrm{PRG} = m \oplus c = (1100\ 0...)_2 \oplus (1111\ 1...)_2 = (0011\ 1...)_2$. Each x_n is a single bit, and we have $x_1 \equiv 0$, $x_2 \equiv 0$, $x_3 \equiv 1$. The given LFSR produces $x_4 \equiv x_3 + x_1 \equiv 1$, $x_5 \equiv x_4 + x_2 \equiv 1$, $x_6 \equiv 0$, $x_7 \equiv 1$, and so on. Continuing, we obtain $\mathrm{PRG} = x_1 x_2 ... = (0011\ 1010\ 0111)_2$.
Hence, the plaintext would be $m = c \oplus \mathrm{PRG} = (1111\ 1011\ 0000)_2 \oplus (0011\ 1010\ 0111)_2 = (1100\ 0001\ 0111)_2$.

A PRG is **predictable** if, given the stream it outputs (but not the seed), one can with some precision predict what the next bit will be (i.e. do better than just guessing the next bit).

In other words: the bits generated by the PRG must be indistinguishable from truly random bits, even in the eyes of someone who knows everything about the PRG except the seed.

The PRGs we discussed so far are entirely predictable because the state of the PRGs is part of the random stream they output.

For instance, for a given LFSR, it is enough to know any ℓ consecutive outputs $x_n, x_{n+1}, ..., x_{n+\ell-1}$ in order to predict all future output.

Example 62. (bonus!) The LFSR $x_{n+31} \equiv x_{n+28} + x_n \pmod{2}$ from Example 59, which is used in glibc, is entirely predictable because observing $x_1, x_2, ..., x_{31}$ we know what $x_{32}, x_{33}, ...$ are going to be. Alice tries to reduce this predictability by using only $x_3, x_6, x_9, ...$ as the output of the LFSR. Demonstrate that this PRG is still perfectly predictable by showing the following:

Challenge. Find a simple LFSR which produces $x_3, x_6, x_9, ...$

Send me the LFSR, and an explanation how you found it, by Feb 10 for a bonus point!

Comment. There is nothing special about this LFSR. Moreover, a generalization of this argument shows that only outputting every Nth bit of an LFSR is always going to result in an entirely predictable PRG.

A popular way to reduce predictability is to combine several LFSRs:

Example 63. Let us consider a baby version of CSS (discussed next class). Our PRG uses the LFSR $x_{n+3} \equiv x_{n+1} + x_n \pmod{2}$ as well as the LFSR $x_{n+4} \equiv x_{n+2} + x_n \pmod{2}$. The output of the PRG is the output of these two LFSRs added with carry.

Adding with carry just means that we are adding bits modulo 2 but add an extra 1 to the next bits if the sum exceeded 1. This is the same as interpreting the output of each LFSR as the binary representation of a (huge) number, then adding these two numbers, and outputting the binary representation of the sum.

If we use $(0, 0, 1)$ as the seed for LFSR-1, and $(0, 1, 0, 1)$ for LFSR-2, what are the first 10 bits output by our PRG?

Armin Straub
straub@southalabama.edu

Solution. With seed $0,0,1$ LSFR-1 produces $0,1,1,1,0,0,1,0,1,1,\ldots$

With seed $0,1,0,1$ LSFR-2 produces $0,0,0,1,0,1,0,0,0,1,\ldots$

We now add these two:

	0	1	1	1	0	0	1	0	1	1	$\cdots$
$+$	0	0	0	1	0	1	0	0	0	1	$\cdots$
carry					1						1
	0	1	1	0	1	1	1	0	1	0	$\cdots$

Hence, the output of our PRG is $0,1,1,0,1,1,1,0,1,0,\ldots$.

Important comment. Make sure you realize in which way this CSS PRG is much less predictable than a single LFSR! A single LFSR with ℓ registers is completely predictable since knowing ℓ bits of output (determines the state of the LFSR and) allows us to predict all future output. On the other hand, it is not so simple to deduce the state of the CSS PRG from the output. For instance, the initial $(0,1,\ldots)$ output could have been generated as $(0,0,\ldots)+(0,1,\ldots)$ or $(0,1,\ldots)+(0,0,\ldots)$ or $(1,0,\ldots)+(1,0,\ldots)$ or $(1,1,\ldots)+(1,1,\ldots)$.

[In this case, we actually don't learn anything about the registers of each individual LFSR. However, we do learn how their values have to match up. That's the correlation that is exploited in **correlation attacks**, like the one described next class for the actual CSS scheme.]

Advanced comment. Is the carry important? Yes! Let $a_1, a_2, \ldots$ and $b_1, b_2, \ldots$ be the outputs of LFSR-1 and LFSR-2. Suppose we sum without carry. Then the output is $a_1+b_1, a_2+b_2, \ldots$ (with addition mod 2). If Eve assigns variables $k_1, k_2, \ldots, k_7$ to the $3+4$ seed bits (the key in the stream cipher), then the output of the combined LFSR will be linear in these seven variables (because the a_i and b_i are linear combinations of the k_i). Given just a few more than 7 output bits, a little bit of linear algebra (mod 2) is therefore enough to solve for $k_1, k_2, \ldots, k_7$.

On the other hand, suppose we include the carry. Then the output is $a_1+b_1, a_2+b_2+a_1b_1, \ldots$ (note how a_1b_1 is $1 \pmod 2$ precisely if both a_1 and b_1 are $1 \pmod 2$, which is when we have a carry). This is not linear in the a_i and b_i (and, hence, not linear in the k_i), and we cannot solve for $k_1, k_2, \ldots, k_7$ as before.