Example 61. (bonus!) The LFSR $x_{n+31} \equiv x_{n+28} + x_n \pmod 2$ from Example 58, which is used in glibc, is entirely predictable because observing $x_1, x_2, \ldots, x_{31}$ we know what $x_{32}, x_{33}, \ldots$ are going to be. Alice tries to reduce this predictability by using only $x_3, x_6, x_9, \ldots$ 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 $x_3, x_6, x_9, \ldots$

Send me the LFSR, and an explanation how you found it, by 2/16 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 (in a nonlinear fashion):

Example 62. 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?

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	⋯
+	0	0	0	1	0	1	0	0	0	1	⋯
carry					1						1
	0	1	1	0	1	1	1	0	1	0	⋯

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.

Example 63. In each case, determine if the stream could have been produced by the LFSR $x_{n+5} \equiv x_{n+2} + x_n \pmod 2$. If yes, predict the next three terms.

(STREAM-1) $\ldots, 1, 0, 0, 1, 1, 1, 1, 0, 1, \ldots$ (STREAM-2) $\ldots, 1, 1, 0, 0, 0, 1, 1, 0, 1, \ldots$

Solution. Using the LFSR, the values $1, 0, 0, 1, 1$ are followed by $1, 1, 1, 0, \ldots$ Hence, STREAM-1 was not produced by this LFSR.

On the other hand, using the LFSR, the values $1, 1, 0, 0, 0$ are followed by $1, 1, 0, 1, 1, 1, 0, \ldots$ Hence, it is possible that STREAM-2 was produced by the LFSR (for a random stream, the chance is only $1/2^4 = 6.25\%$ that 4 bits matched up). We predict that the next values are $1, 1, 0, \ldots$

Comment. This observation is crucial for the attack on CSS described in Example 64.

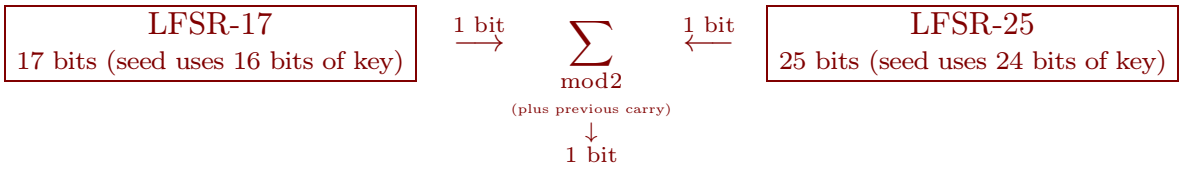
Example 64. (CSS) The CSS (content scramble system) is based on 2 LFSRs and used for the encryption of DVDs. Let us indicate (in a slightly oversimplified way) how to break it.

CSS was introduced in 1996 and first compromised in 1999. One big issue is that its key size is 40 bits. Since $2^{40} \approx 1.1 \cdot 10^{12}$ is small by modern standards, even a direct brute-force attack in time 2^{40} is possible.

However, we will see below that poor design makes it possible to attack it in time 2^{16}.

Historic comment. 40 bits was the maximum allowed by US export limitations at the time.

https://en.wikipedia.org/wiki/Export_of_cryptography_from_the_United_States



CSS PRG combines one 17-bit LFSR and one 25-bit LFSR. The bits output by the CSS PRG are the sum of the bits output by the two LFSRs (this is the usual sum, including carries).

The 40 bit key is used to seed the LFSRs (the 4th bit of each seed is "1", so we need $16 + 24 = 40$ other bits).

Here's how we break CSS in time 2^{16}:

- If a movie is encrypted using MPEG then we know the first few, say x (6-20), bytes of the plaintext.
- As in Example 59, this allows us to compute the first x bytes of the CSS keystream.
- We now go through all 2^{16} possibilities for the seed of LFSR-17. For each seed:
  - We generate x bytes using LFSR-17 and subtract these from the known CSS keystream.
  - This would be the output of LFSR-25. As in Example 63, we can actually easily tell if such an output could have been produced by LFSR-25. If yes, then we found (most likely) the correct seed of LFSR-17 and now also have the correct state of LFSR-25.

This kind of attack is known as a correlation attack.

https://en.wikipedia.org/wiki/Correlation_attack

Comment. Similar combinations of LFSRs are used in GSM encryption (A5/1,2, 3 LFSRs); Bluetooth (E0, 4 LFSRs). All of these are broken; so, of course, they shouldn't be used. However, it is difficult to update things implemented in hardware...