§6.5 : Baby CSS.

It turns out that the main vulnerability of the LFSR and LFSR sum methods just introduced (at least, the main vulnerability when it comes to known plaintext attacks) is linearity of the system. An attacker has powerful algebraic techniques (such as row reduction / linear algebra) at their disposal to reverse engineer everything.

Our fix: Introduce some "nonlinearity" that preserves the simplicity of encryption but makes the linear algebra methods used in the known plaintext attack fail.

The trick for Baby CSS is essentially:

Replace XOR in LFSR sum with binary addition.

I.e.

Instead of this:

$$\begin{array}{r} 101001 \\ \oplus\, 011001 \\ \hline 110000 \end{array}$$ } XOR

we have to carry some 1's:

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ 1} \\ 101001 \\ +\, 011001 \\ \hline 100010 \end{array}$$

§6.5 : Baby CSS.

It turns out that the main vulnerability of the LFSR and LFSR sum methods just introduced (at least, the main vulnerability when it comes to known plaintext attacks) is <u>linearity</u> of the system. An attacker has powerful algebraic techniques (such as row reduction / linear algebra) at their disposal to reverse engineer everything.

Our fix: Introduce some "nonlinearity" that preserves the simplicity of encryption but makes the linear algebra methods used in the known plaintext attack fail.

The trick for Baby CSS is essentially:

Replace XOR in LFSR sum with binary addition.

I.e.

Instead of this:

$$\begin{array}{r} 101001 \\ \oplus\, 011001 \\ \hline 110000 \end{array} \quad \Big\} \text{ XOR}$$

we have to carry some 1's:

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ 1} \\ 101001 \\ +\, 011001 \\ \hline 100010 \end{array}$$

However we'll also chunk the keystreams into blocks of length 5 when summing in this way (this fixes the issue of having to add right-to-left when carrying, while wanting to generate a keystream left-to-right).

Baby CSS Algorithm:

- Choose a 3-bit LFSR and a 5-bit LFSR. As before, assume the rightmost bit in the seed of each is 1, to avoid the possibility of accidentally seeding with the zero.
- Generate the LFSR-3 and LFSR-5 keystreams. Chunk them into blocks of 5, and add (not XOR). Carry leftover ones (at the end of the block) to the next block. This generates the Baby CSS keystream.
- XOR the keystream with the plaintext to get the ciphertext.
- To decrypt, generate the keystream again and XOR it with the ciphertext.

Example:

Suppose LFSR-3 is $b_3 = b_2' + b_1'$ and

LFSR-5 is $c_5 = c_4' + c_2' + c_1'$.

Seed LFSR-3 with 0 1 1 ← we insist on this 1

Seed LFSR-5 with 1 0 1 0 1 ←