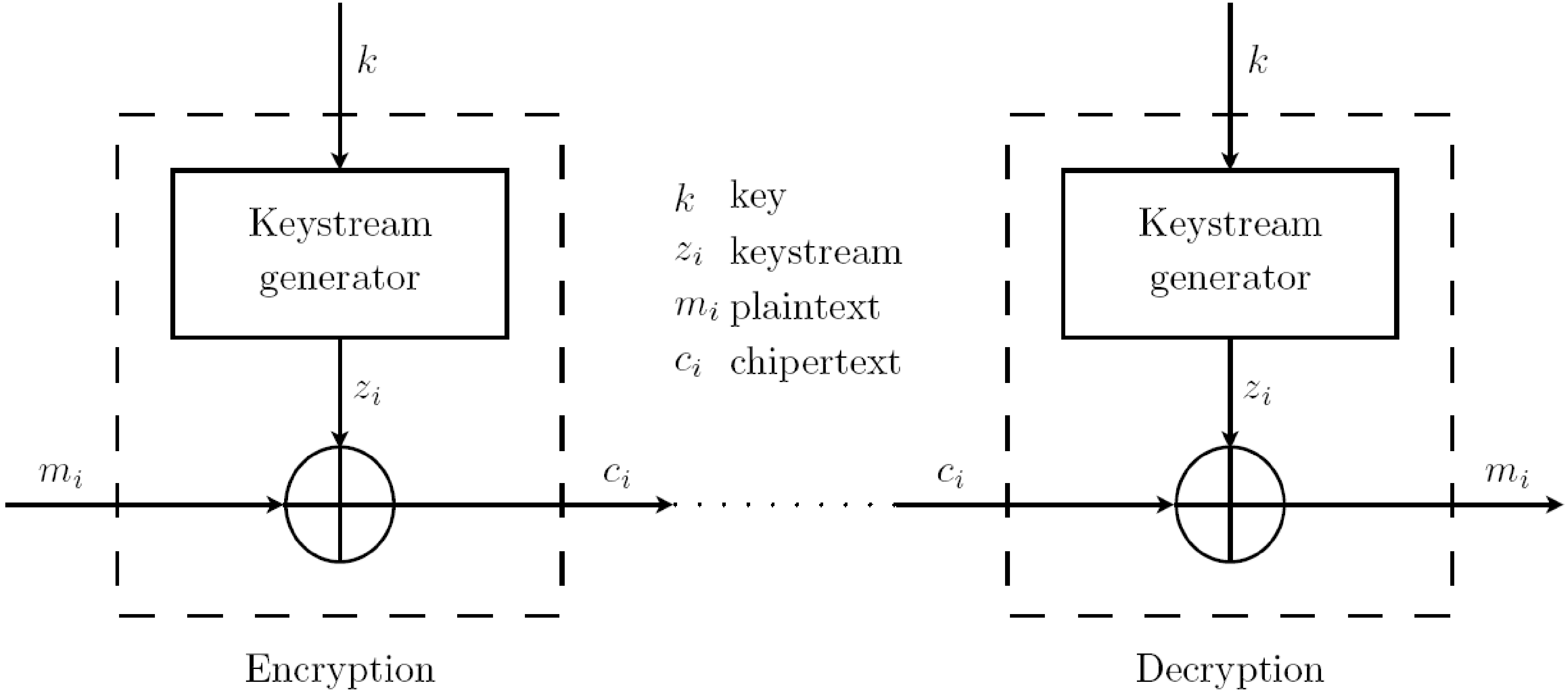
Stream Cipher Design

An evaluation of the eSTREAM candidate Polar Bear

Additive Stream Cipher

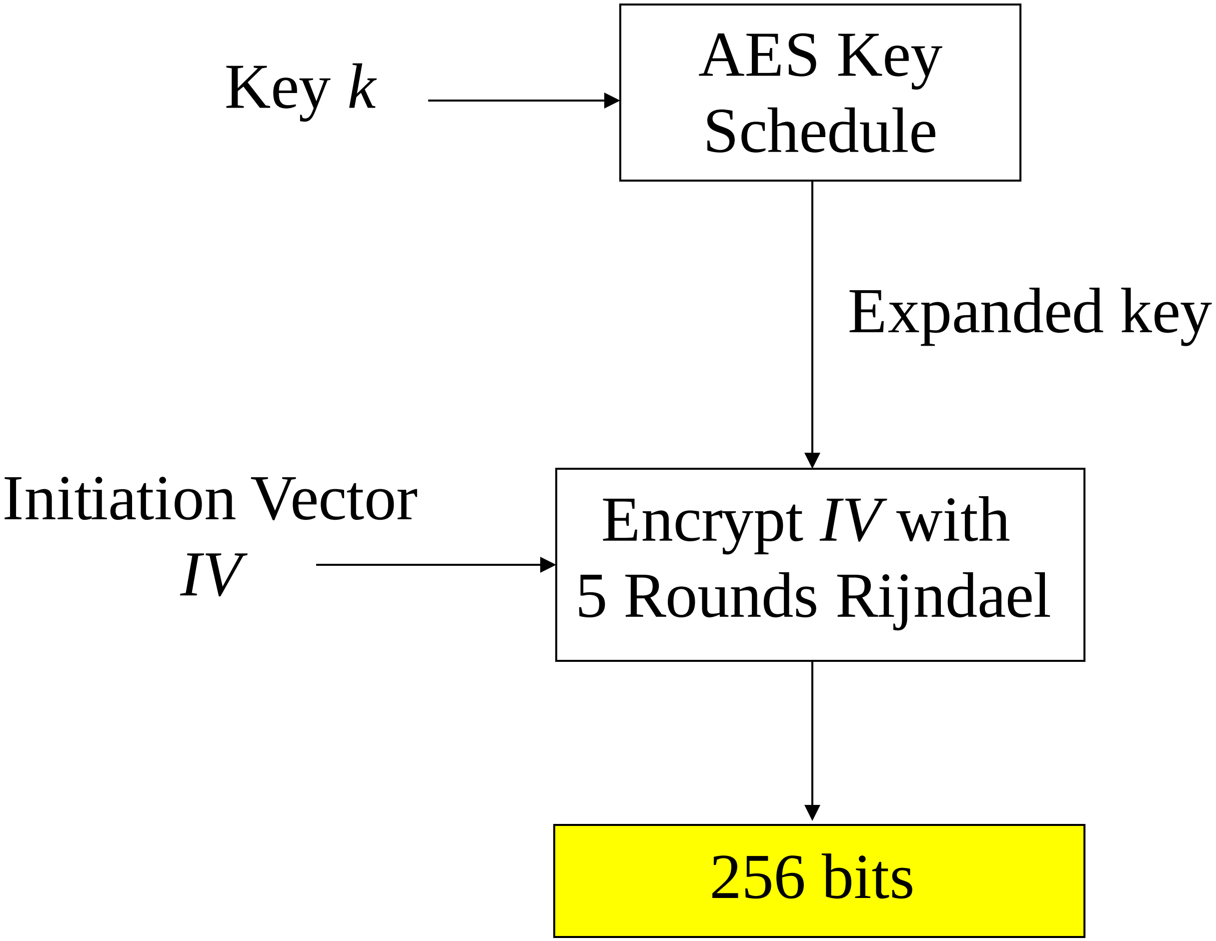


k	key
z_i	keystream
m_i	plaintext
c_i	chipertext

Properties

- Given a truly random secret-key k, output a sting of length l >> k that appears to be random.
- No attack should be faster than exhaustive search $O(2^{128})$

Polar Bear Key and IV Schedule



Initial Internal State

Dynamic permutation *D8* = *T8* =

99	17	253	…

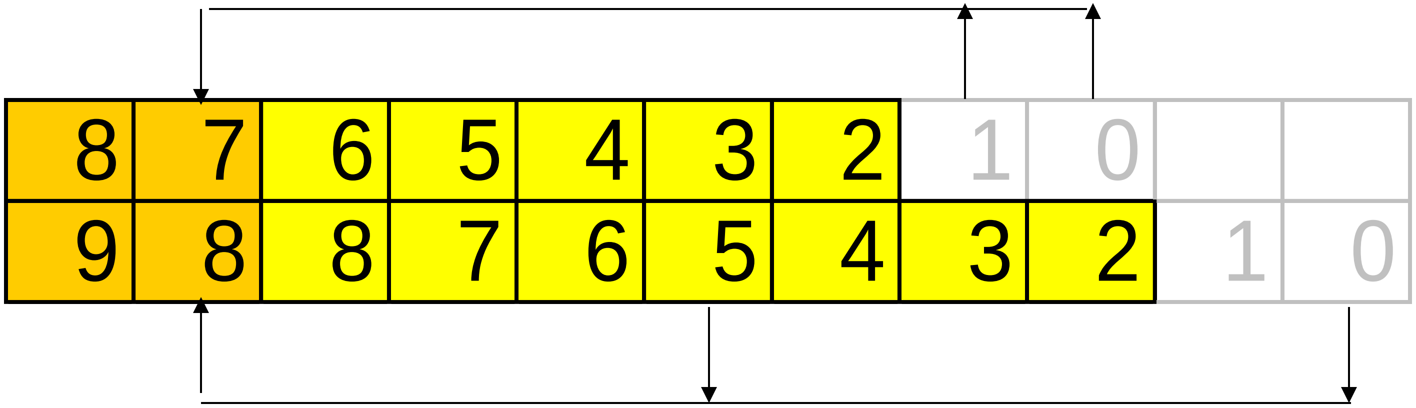
16 bit quantity *S* =

0

LFSR *R0*	6	5	4	3	2	1	0		
LFSR *R1*	8	7	6	5	4	3	2	1	0

Next State Function

- Step *R0* and *R1* two or three steps according to 14th and 15th bits in S
- Feedback Polynomials Ax^7+Bx^6+1 and Cx^9+Dx^4+1 over $F_{2^{16}}$



Next State Function

- For *i = 0,1* do

□ α^0_0	α^0_1	□α^0_2	□α^0_3
□ α^1_0	α^1_1	□α^1_2	□α^1_3

1. Let $\beta^i_j = D_8(\alpha^i_j)$, $j = 0, 1, 2, 3$

2. Swap elements in D_8 by

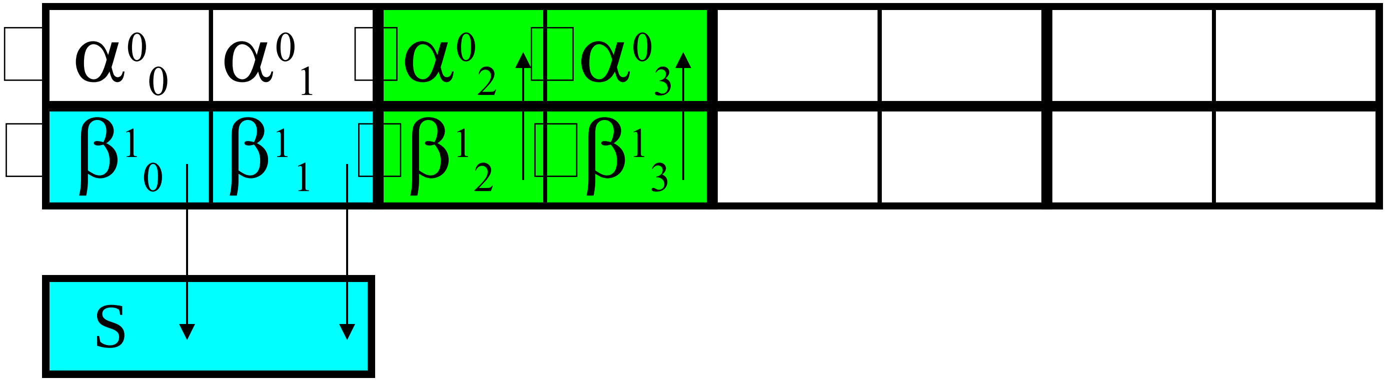
$$D_8(\alpha^i_0) \leftarrow \beta^i_2, \qquad D_8(\alpha^i_1) \leftarrow \beta^i_0$$
$$D_8(\alpha^i_2) \leftarrow \beta^i_3, \qquad D_8(\alpha^i_3) \leftarrow \beta^i_1$$

Next State Function

- Update *S* and *R0[5]* according to

$$S = S +_{16} \beta_0^1 || \beta_1^1$$

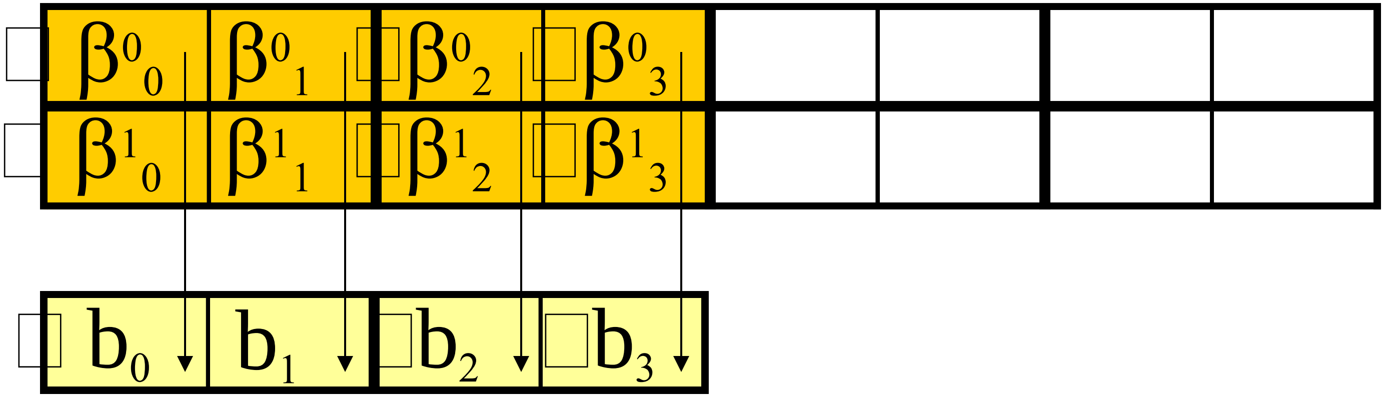
$$R0[5] = R0[5] +_{16} \beta_2^1 || \beta_3^1$$



Output Generation

- Output $b_0||b_1||b_2||b_3$

  where $\mathrm{b}_j = \beta_j^0 \oplus \beta_j^1 \qquad j = 0, 1, 2, 3$



Weakness 1 - Permutation

1. Let $\beta_j^i = D_8(\alpha_j^i)$, $j = 0, 1, 2, 3$

2. Swap elements in D_8 by

   $D_8(\alpha_0^i) \leftarrow \beta_2^i$,

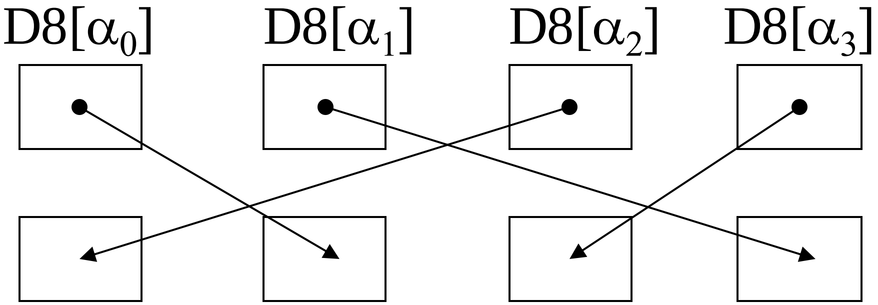
   $D_8(\alpha_1^i) \leftarrow \beta_0^i$,

   $D_8(\alpha_2^i) \leftarrow \beta_3^i$

   $D_8(\alpha_3^i) \leftarrow \beta_1^i$

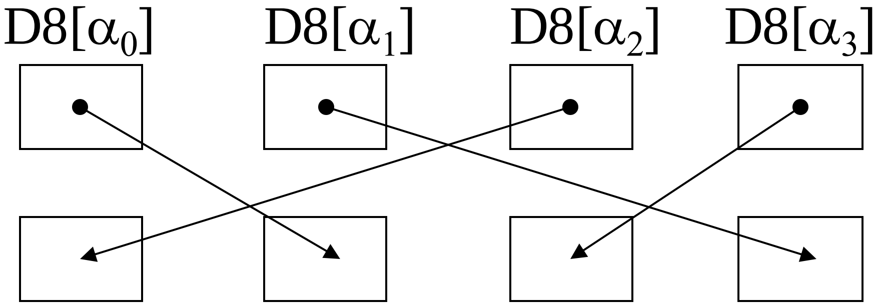
Weakness 1 - Permutation

- All α different

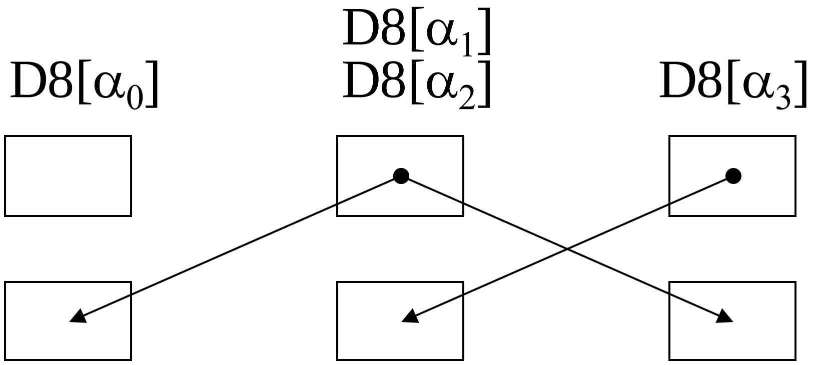


Weakness 1 - Permutation

- All α different



- But if for example $\alpha_1 = \alpha_2$



Weakness 2 - State recovery

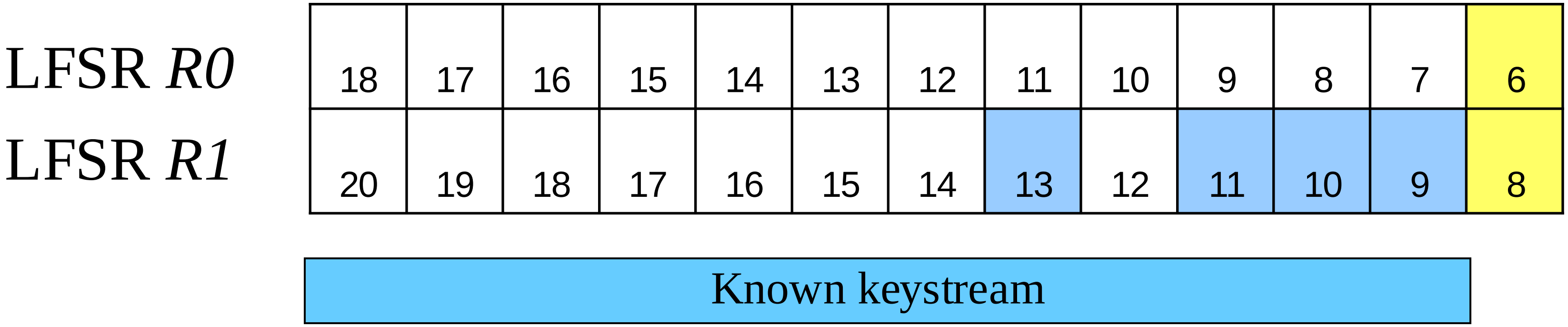
- Let the first six stepping of *R0* and *R1* be 2-steppings. Probability 2^{-10}
- Let no pair of the 48 first α be equal. The probability for this is > 2^{-7}

$$\frac{256!}{(256-48)! \cdot 256^{48}} > 2^{-7}$$

- *D8* can now be treated as constant.
- Let the first 24 bytes of plaintext be known

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

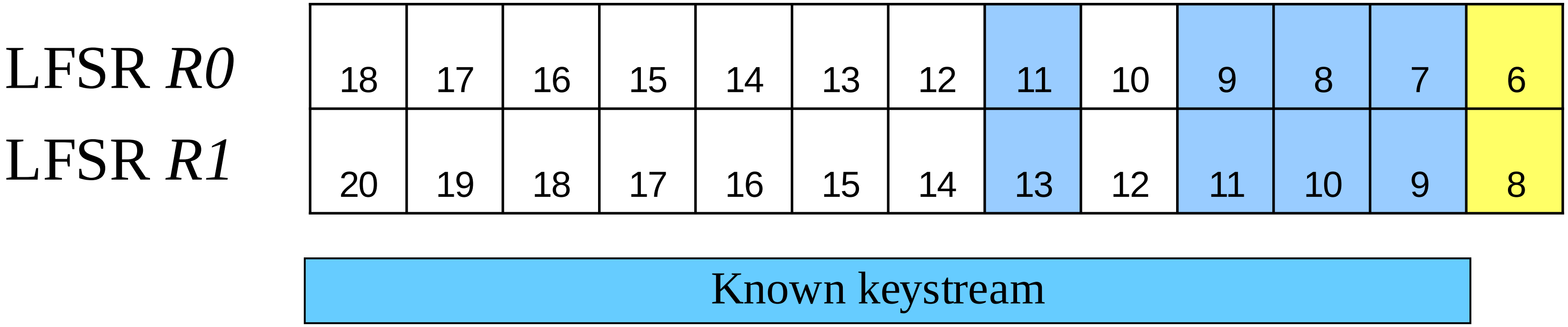


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

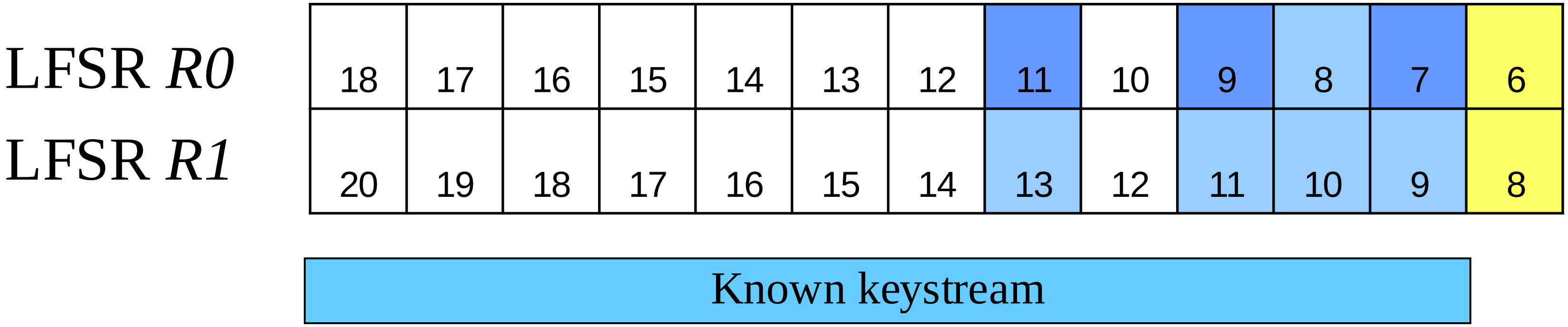


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

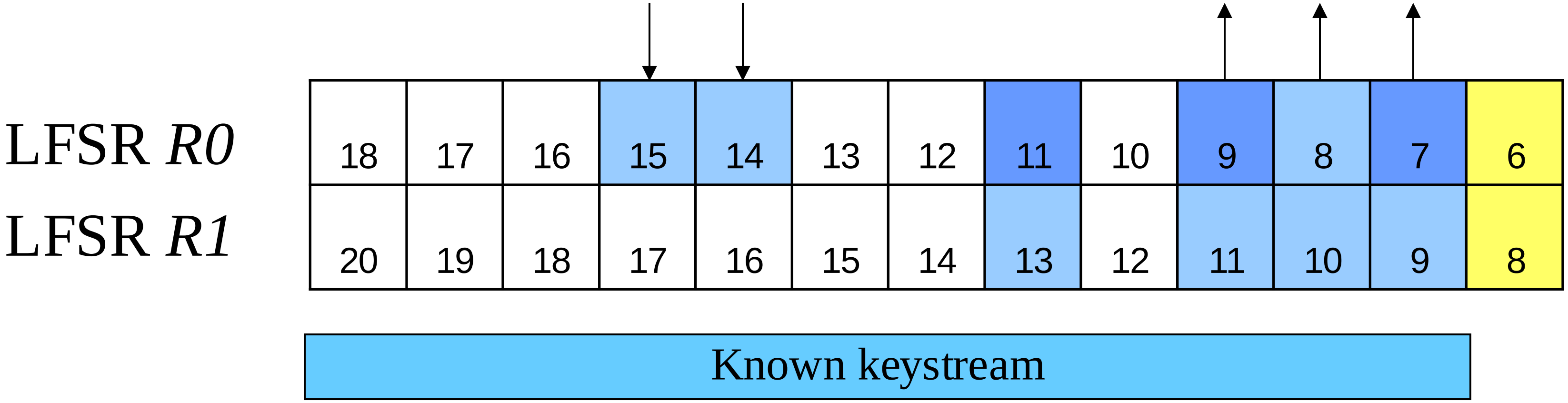


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

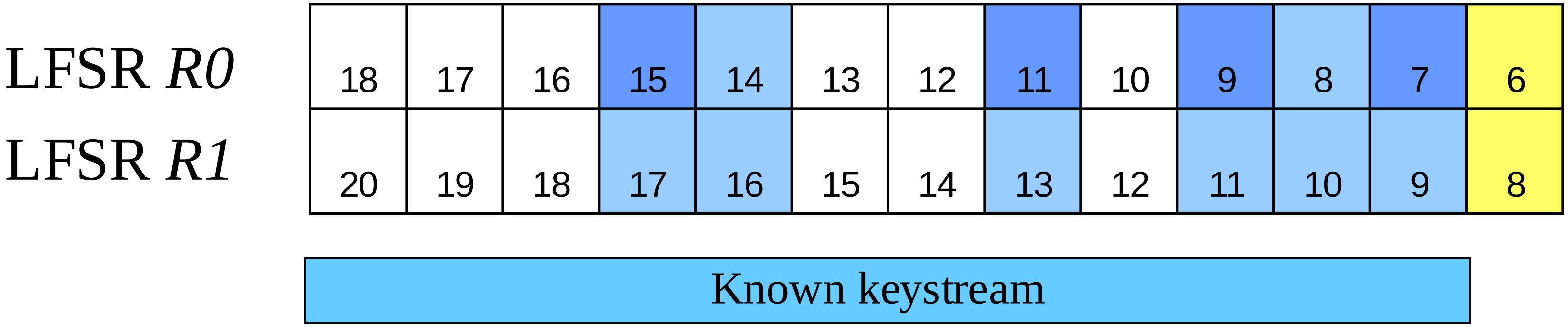


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

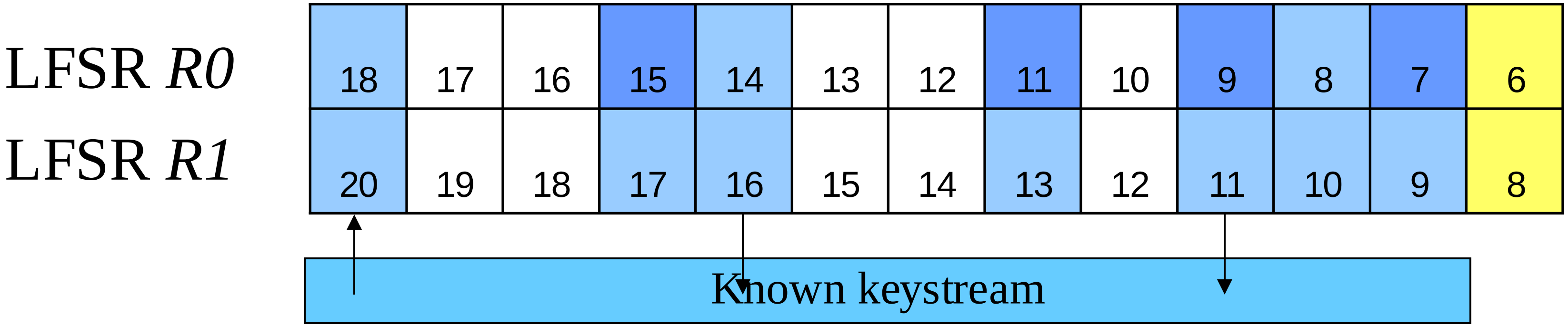


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

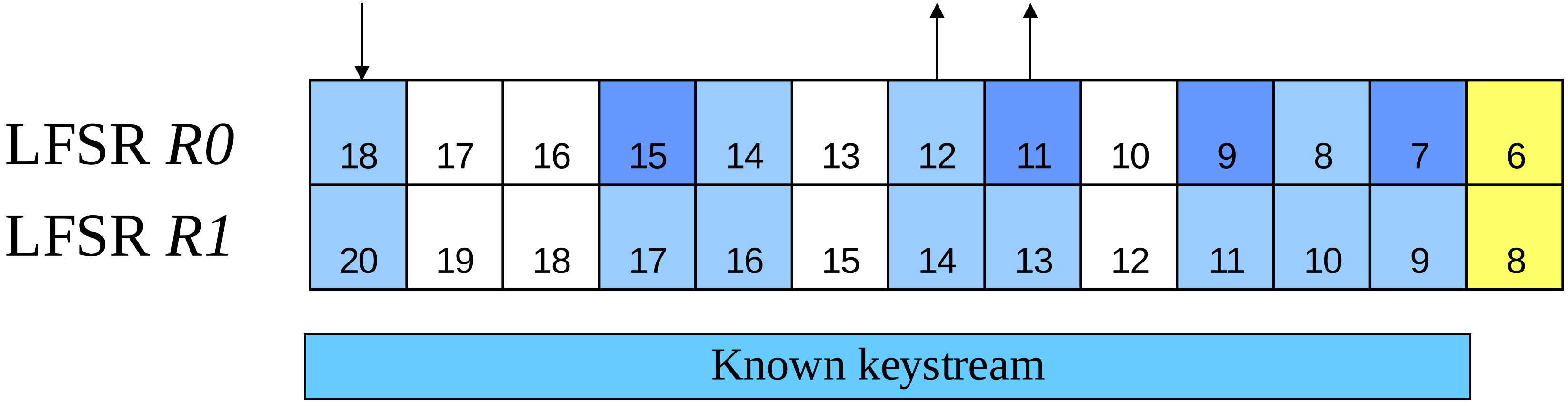


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

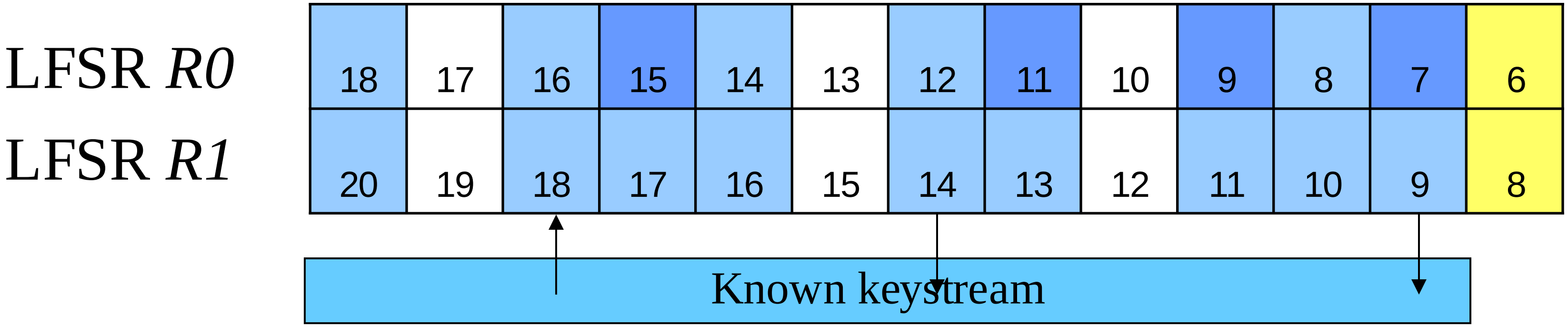


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

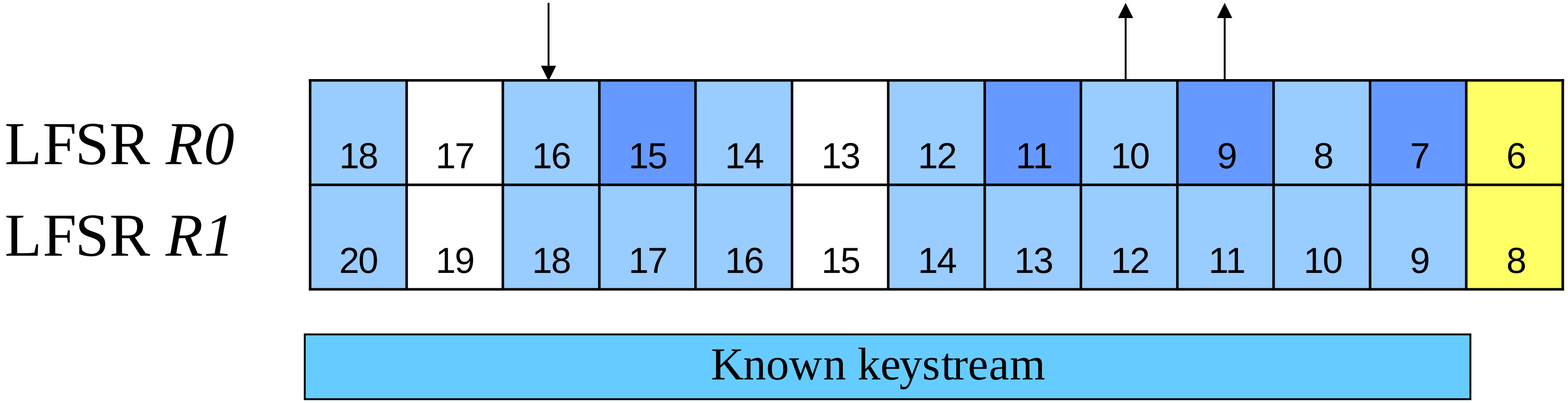


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

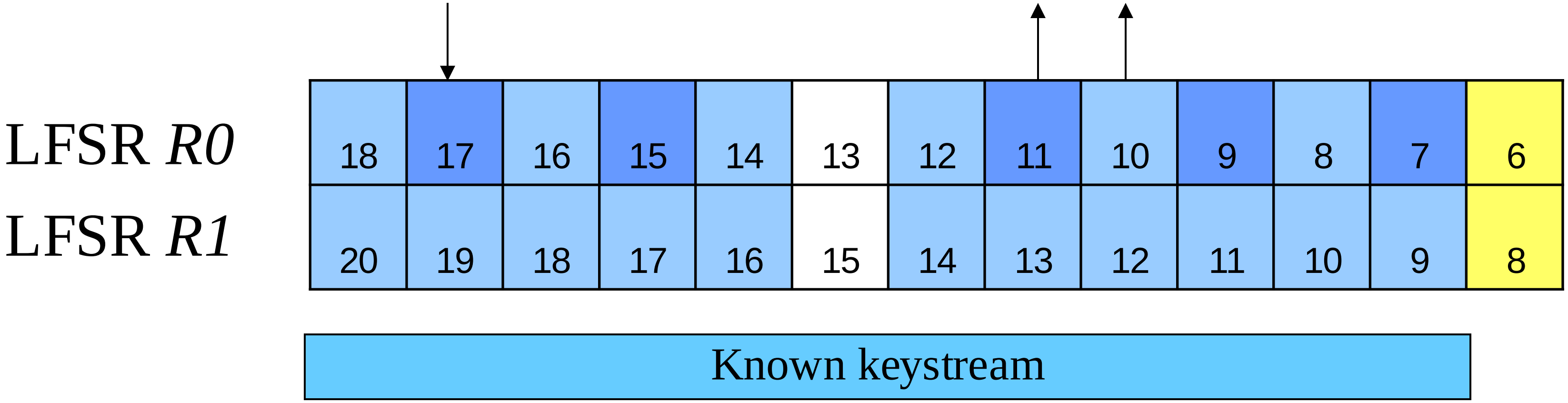


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

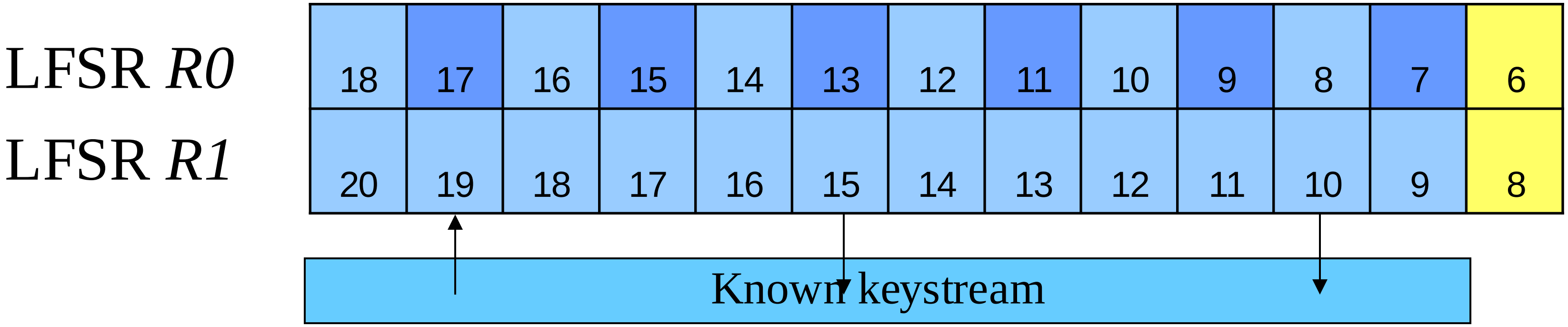


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*

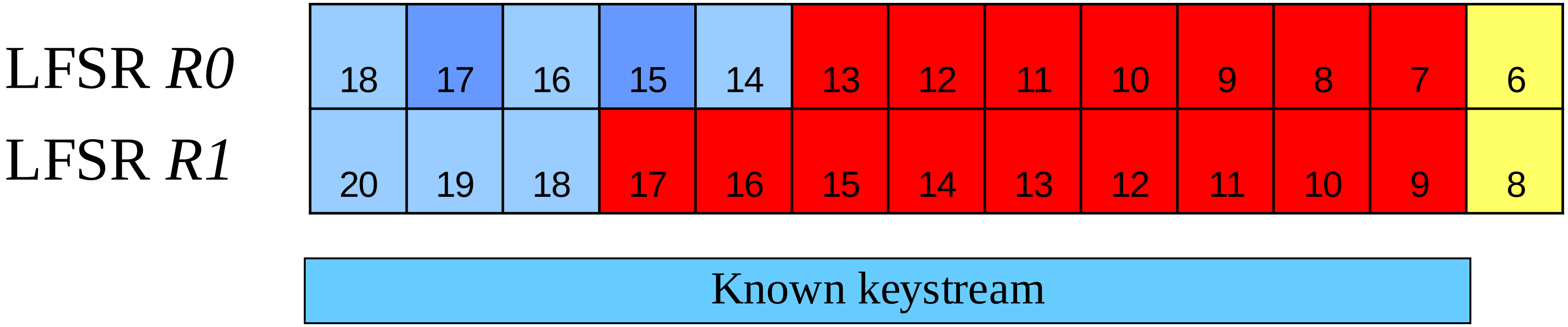


LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

Weakness 2 - State recovery

- Then it suffices to guess the 64 bits *R1[9]-R1[11]* and *R1[13]*



LFSR *R0*	18	17	16	15	14	13	12	11	10	9	8	7	6
LFSR *R1*	20	19	18	17	16	15	14	13	12	11	10	9	8

Known keystream

- Time complexity $O(2^{81}) < O(2^{128})$

Why was this attack possible

- Known array *D8*.
- Short LFSRs. It is even possible to guess whole the small LFSR and still be under O(2^{128}).
- Trinom as feedback polynomials.
- The LFSR stages are too related by stepping, output, and update.

Polar Bear 2.0

- Permute *T8* as a function of the expanded key *k'*. Initiate *D8* to this permuted *T8**.

```
For $i = 0 \ldots 511/767$
    SWAP( $T8[\ i\ ]$, $T8[\ k'_i\ ]$ )
```

Polar Bear 2.0

- Permute *T8* as a function of the expanded key *k'*. Initiate *D8* to this permuted *T8**.

```
For $i = 0 \ldots 511/767$
    SWAP( $T8[\ i\ ]$, $T8[\ k'_i\ ]$ )
```

- Use another `KeyExpansion` than AES, as AES `KeyExpansion` is weak and have bad statistical properties.

Polar Bear 2.0

- Different very small changes that would have made the above attack much harder. Mainly changes of index.

New security issues

- Can the permutated array give information about the key?
- Many different IV under one key. Can an attacker use this to determine the permutated array?
- Related keys.