Cryptanalysis of the Loiss Stream Cipher

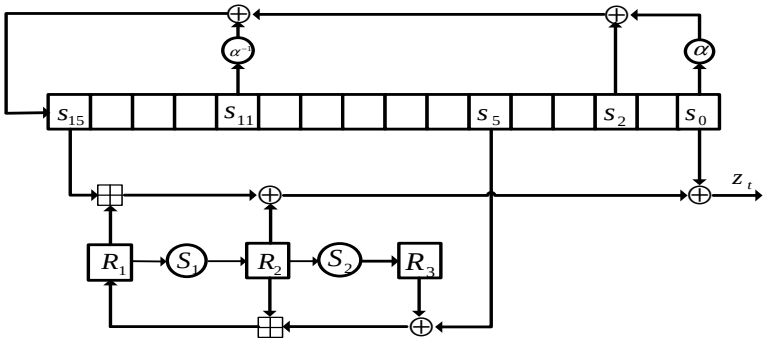
Alex Biryukov[1], Aleksandar Kircanski[2] and Amr M. Youssef[2]

[1]: University of Luxembourg, LACS, Luxembourg
[2]: Concordia University, CIISE, Canada

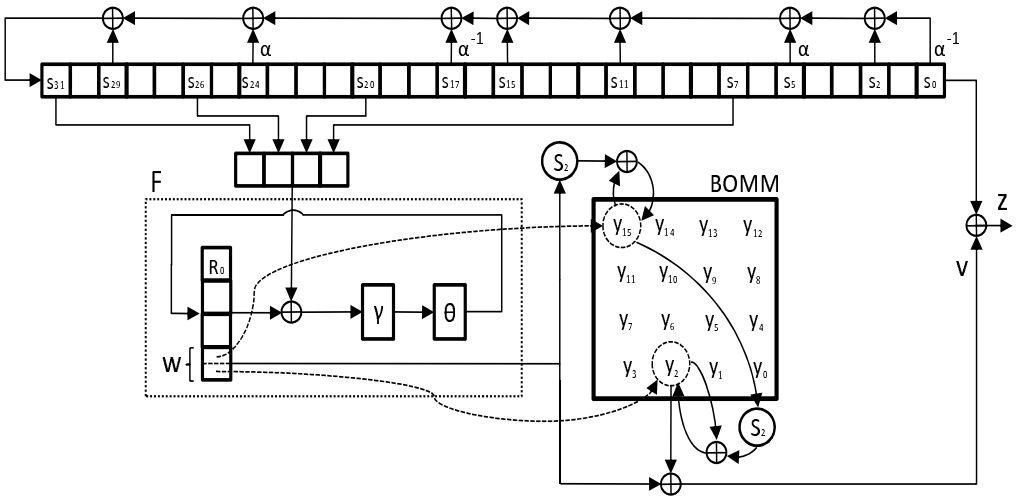
Selected Areas in Cryptography (SAC) 2012
Windsor, Canada

Motivation

Snow 3G:



Loiss:



- A new RC4-like component added
- more security or less security?

Overview

- Introduction (LFSR-based word-oriented stream ciphers)
- Specification of Loiss
- Attacking Loiss
- Complexity of the attack
- Conclusion

LFSR-based word-oriented ciphers

Two main components

- LFSR: (usually) over GF(2^{32})
- FSM: introduces non-linearity to the process (S-box, modular addition, modular multiplication,...)

Popular design strategy, fast encryption in software

- SNOW 2.0 (ISO standard)
- SNOW 3G (3GPP)
- SOSEMANUK (eStream)
- ZUC (proposed for 4G mobile standard)

Loiss stream cipher

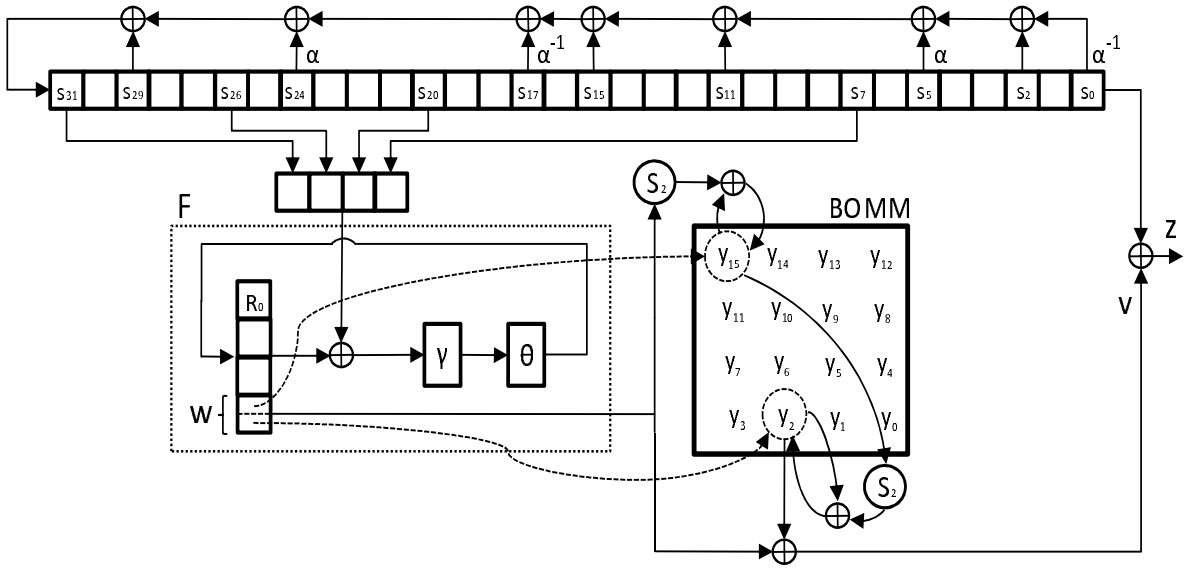
Loiss: an LFSR-based byte-oriented cipher

Targets byte-oriented platforms

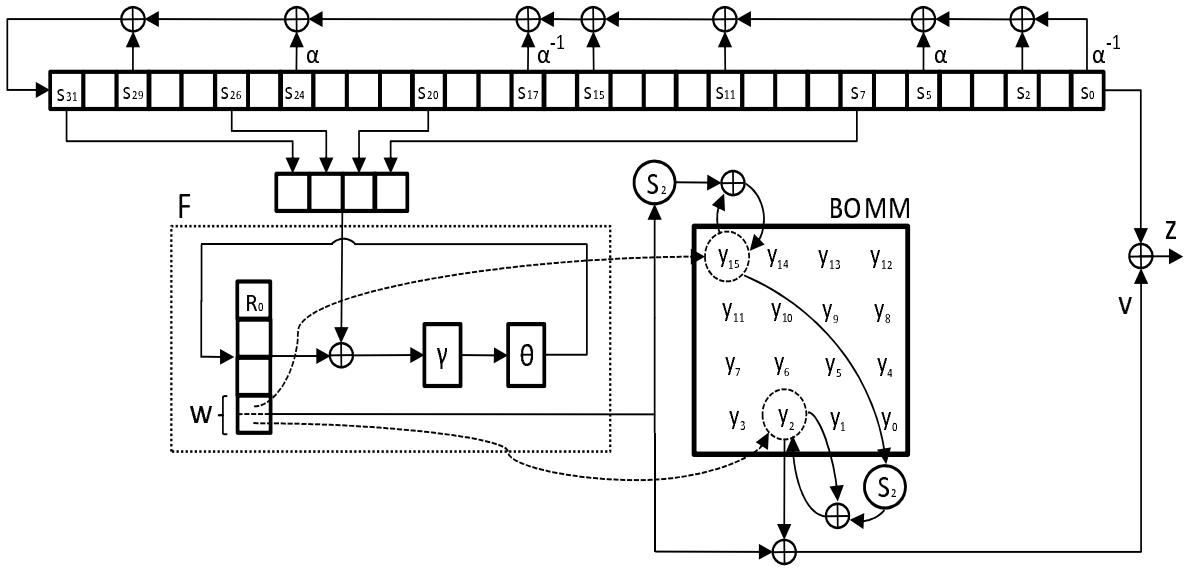
128-bit key, 128-bit IV

Designed by Dengguo Feng *et al.* from State Key Laboratory of Information Security in China

**Major novelty: the FSM contains a new component: BOMM (Byte Oriented Mixer with Memory)

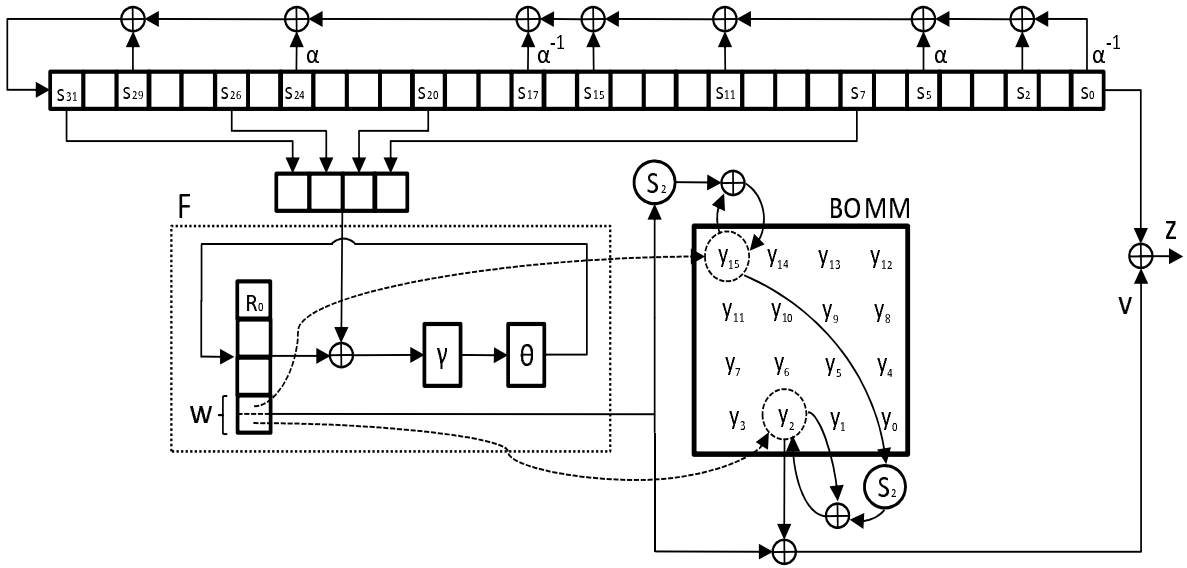


- Inner state: 416 bits (32$\times$ 8 + 4$\times$8 + 16$\times$8)
- A 32-byte LFSR
- FSM: *R* (4 bytes) and BOMM (16 bytes)



LFSR: let α be the root of $\pi(x) = x^8 + x^7 + x^5 + x^3 + 1$ in $F_2[x]/(\pi(x))$. Characteristic polynomial of the LFSR:

$$f(x) = x^{32} + x^{29} + \alpha x^{24} + \alpha^{-1} x^{17} + x^{15} + x^{11} + \alpha x^5 + x^2 + \alpha^{-1}.$$



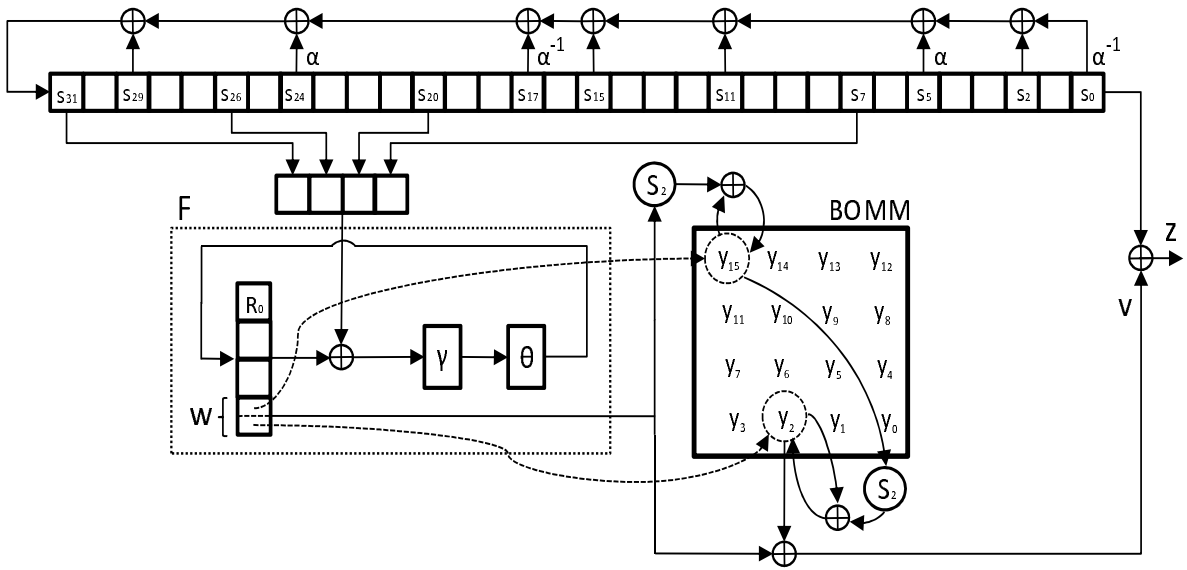
$F : F_2^{32} \times F_2^{32} \rightarrow F_2^8$

Input: 4 LFSR bytes, Memory: 4-byte register R.

Update:

- $\gamma(x_1|x_2|x_3|x_4) = S_1(x_1)|S_1(x_2)|S_1(x_3)|S_1(x_4)$
- $\theta(x) = x \oplus (x \lll 2) \oplus (x \lll 10) \oplus (x \lll 18) \oplus (x \lll 24)$

Output: Left-most byte of R



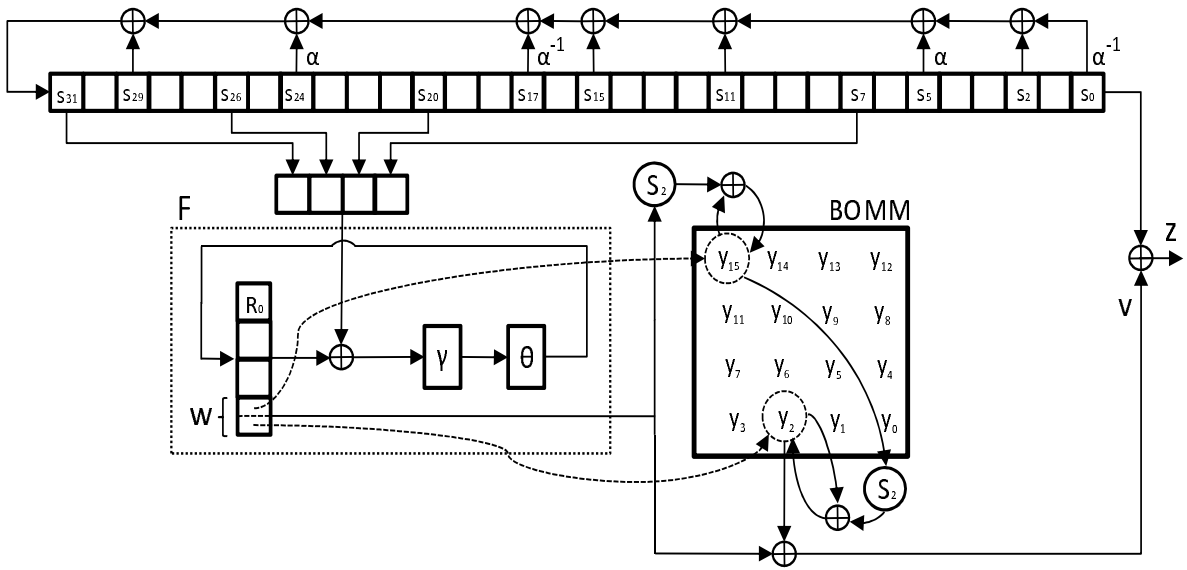
BOMM: $F_2^8 \times F_2^{128} \longrightarrow F_2^8$

Input: w, i.e., the left-most byte of R, Memory: register R

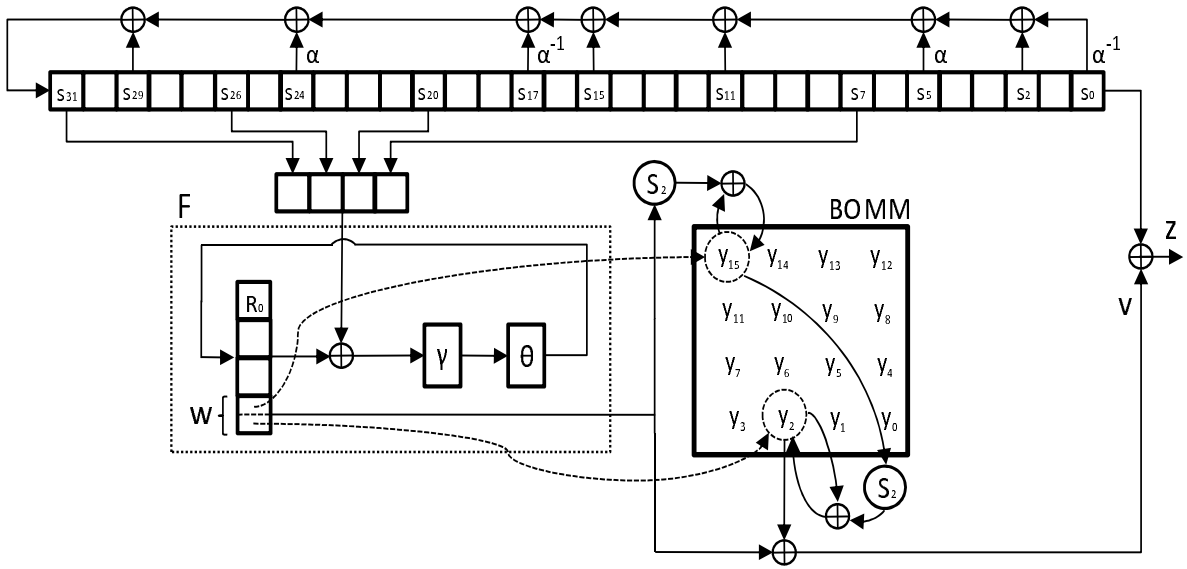
Update:

- Choose two pseudo-random bytes
- Update them non-linearly

Output: pseudo-random byte xor-ed to w



- Key loading, LFSR:
  - $s_{15}|s_{14}|\ldots s_0 = K_{15}|K_{14}|\cdots|K_0$
  - $s_{31}|s_{30}|\ldots s_{16} = IV_{15} \oplus K_{15}|IV_{14} \oplus K_{14}|\cdots|IV_0 \oplus K_0$
- BOMM: $y_{15}|y_{14}|\cdots y_0 = IV_{15}|IV_{14}|\cdots|IV_0$
- R: set to zero



Initialization procedure: 64 Loiss steps without output

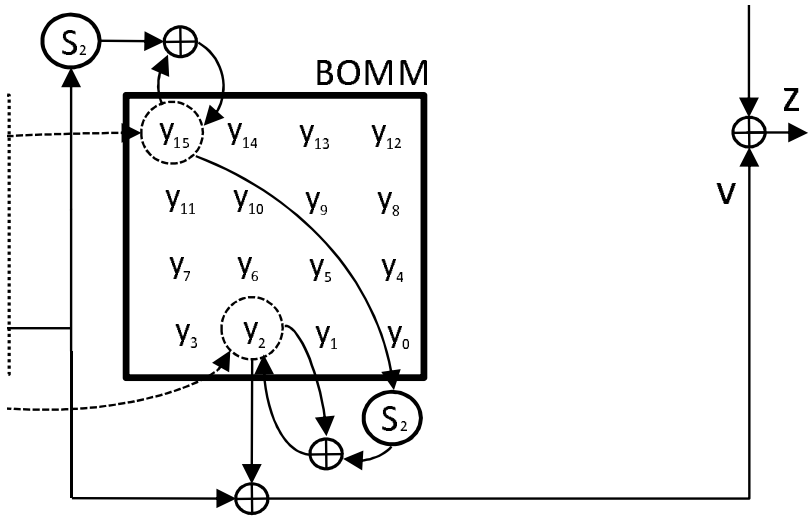
Connect the FSM output to the LFSR update function

- LFSR bits depend non-linearly on the (K, IV)

A common strategy to increase the mixing in the initialization procedure (SNOW 2.0, SNOW 3G)

Attacking the Loiss initialization procedure

- Attack against the initialization procedure
- Differential-style attack
- Related-key resynchronization attack
- Detect a particular event that occured early in the procedure
- Get the equations in key bits for the events to hold
- Solve equations



Observation

The BOMM does not do a good job in diffusing differences.

- Assume a one-byte difference in BOMM
- One Loiss step: the diff. will not be diffused with $p \approx (\frac{15}{16})^2$
- Reason: only two pseudo-random elements updated (RC4-like)

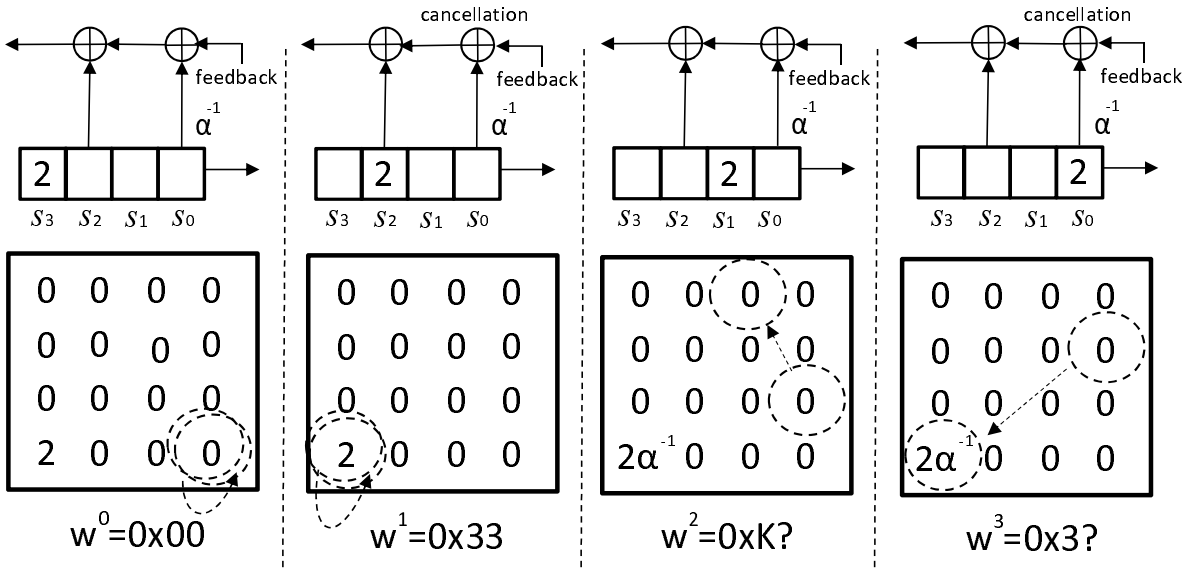
Natural attempt:

- Choose $(\delta K, \delta IV)$ so that only one BOMM byte is active
- Disallowed by key-IV loading procedure

Any change in the (K, IV) will introduce a change in the LFSR.

Attack strategy:

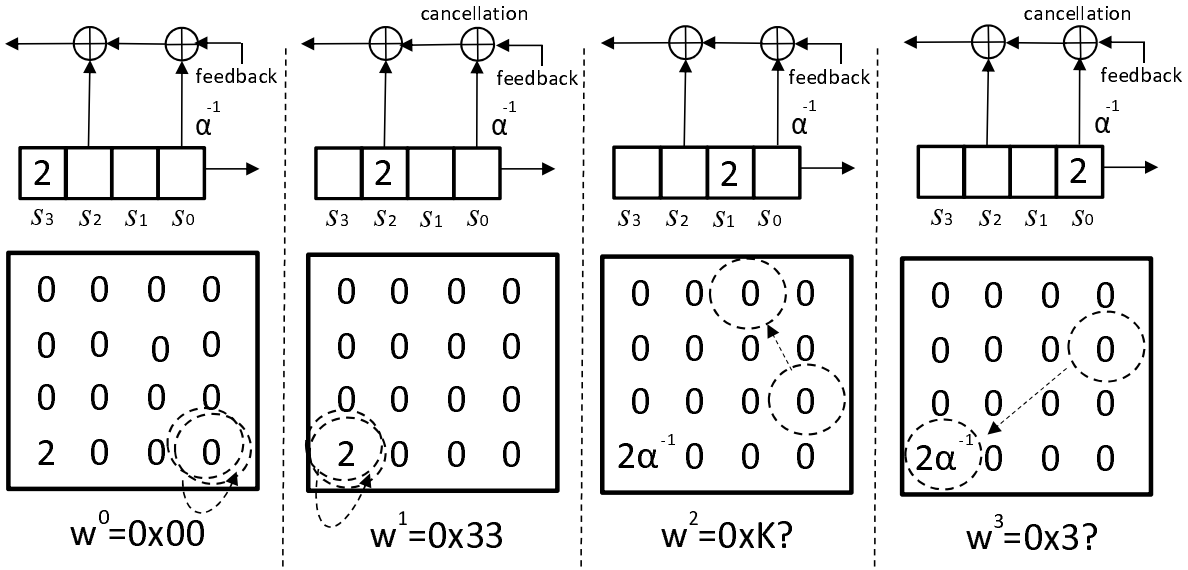
- Start with a low-weight difference in BOMM and LFSR (and a zero-difference in R register)
- Have the LFSR difference cancelled out in the early steps
- Thanks to bad BOMM diffusion, pass through all of the 64 steps of initialization



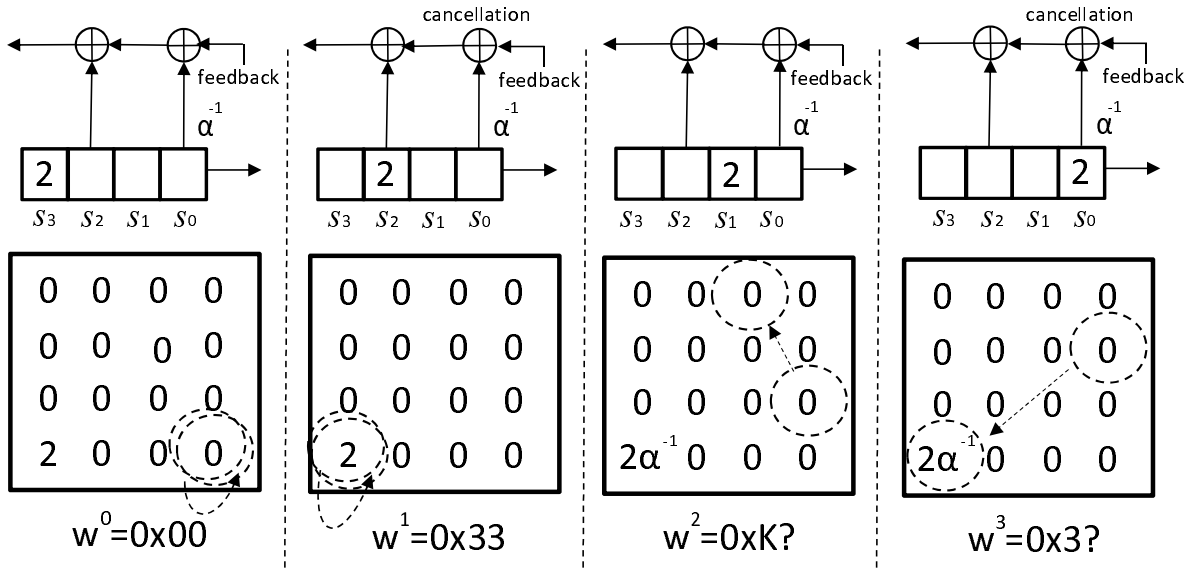
- Analysis of how can the starting cancellation happen
- Starting difference:

$$K_3 \oplus K_3' = IV_3 \oplus IV_3' = 0x2$$

- This starting difference now in s_3 (LFSR), y_3 (BOMM)

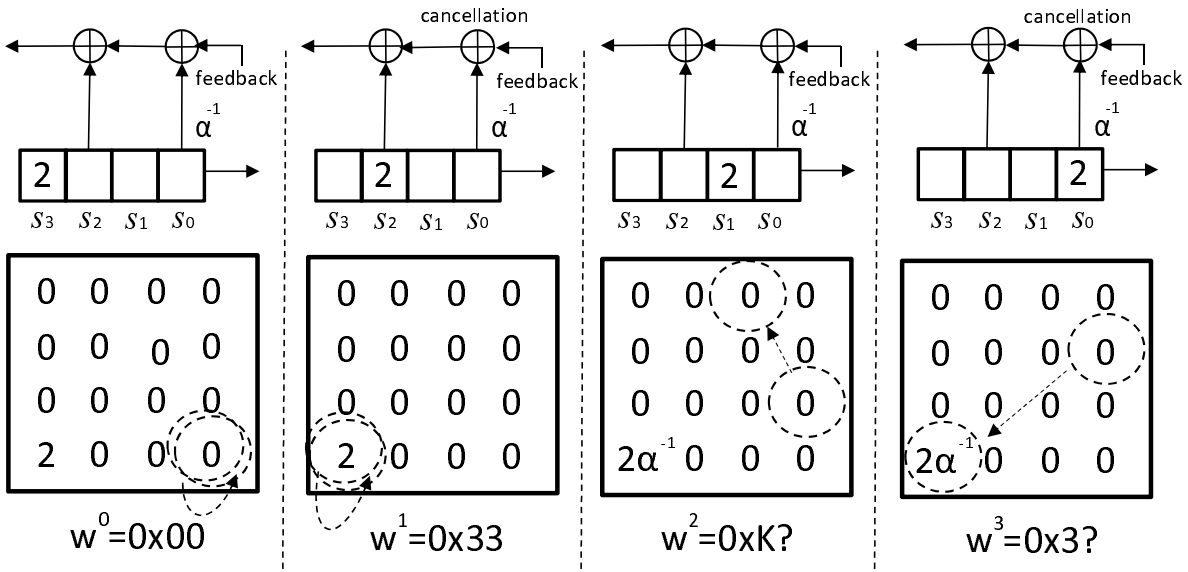


- The LFSR difference shifts to the right
  - On its way, it passes through two taps
- BOMM is connected to the LFSR through feed-forward
- Allows stopping the LFSR difference diffusion



Analysis of the full LFSR difference cancellation:

1. step: y_0 updated twice, zero-difference remains
2. step: y_3 updated twice, LFSR tap difference cancelled
   - the difference changes: $0x2 \rightarrow 0x2\alpha^{-1}$
3. step: y_2 not consulted
4. step: y_3 tap difference cancelled



Lemma

This starting difference fully cancels out in the LFSR after 4 steps iff

$$(w^0, w^1, w^2, w^3) = (0x00, 0x33, 0xK?, 0x3?)$$

where w is the leftmost value of the R register.

Passing through the whole initialization

Summary of the differential distinguisher for the initialization:

- Fix δK and δIV.
- Resynchronize the cipher for different IVs.
- Early-step cancellation, i.e.

$$(w^0, w^1, w^2, w^3) = (0x00, 0x33, 0xK?, 0x3?)$$

  happens with $p_w \approx 2^{-12.1}$.
- After 4 steps, left with 1-byte difference in BOMM.
- No further diffusion in 64 steps with $p_s \approx 2^{-11.3}$.

Pass through the initialization finishing in only 1-byte difference (in BOMM): $p = p_s \times p_w = 2^{-12.1} \times 2^{-11.3} = 2^{-23.4}$

Distinguishing the cancellation event

Goal is to distinguish between

- Loiss keystream output difference, where only 1 BOMM byte active
- Uniform random byte-sequence

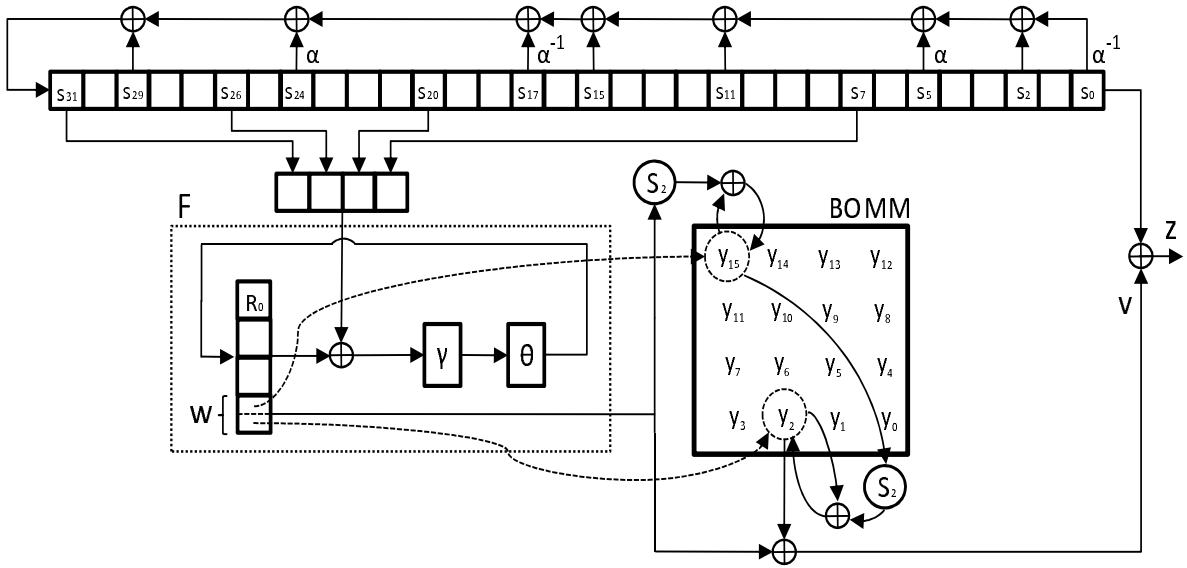
Basis for distinguisher: many zero-differences in the output
Distinguisher:

- Count the number of zero-differences in first n bytes
- If this count is $\geq m$ return *Loiss keystreams*, otherwise return *Random*.

In fact a class of distinguishers depending on (n, m).

How do we measure the quality of a distinguisher?
False negative and false positive probabilities:

(n, m)	P[false positive] $\approx$	P[false negative] $\approx$
$(16, 6)$	$2^{-35.1}$	$2^{-22.41}$
$(16, 8)$	$2^{-50.4}$	$2^{-16.00}$
$(24, 8)$	$2^{-44.6}$	$2^{-24.01}$
$(24, 10)$	$2^{-59.2}$	$2^{-19.91}$
$(32, 10)$	$2^{-54.2}$	$2^{-27.6}$
$(32, 12)$	$2^{-68.3}$	$2^{-20.68}$



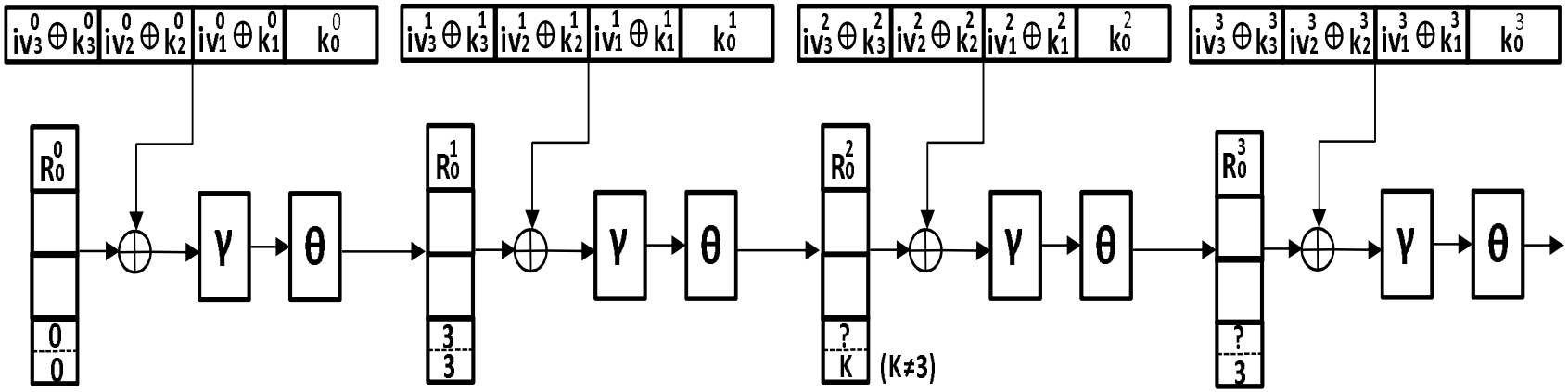
So far, we detected some of the *IV* pairs for which

$$(w^0, w^1, w^2, w^3) = (0x00, 0x33, 0xK?, 0x3?) \qquad (1)$$

where w is the MSB of the R. What is this good for?

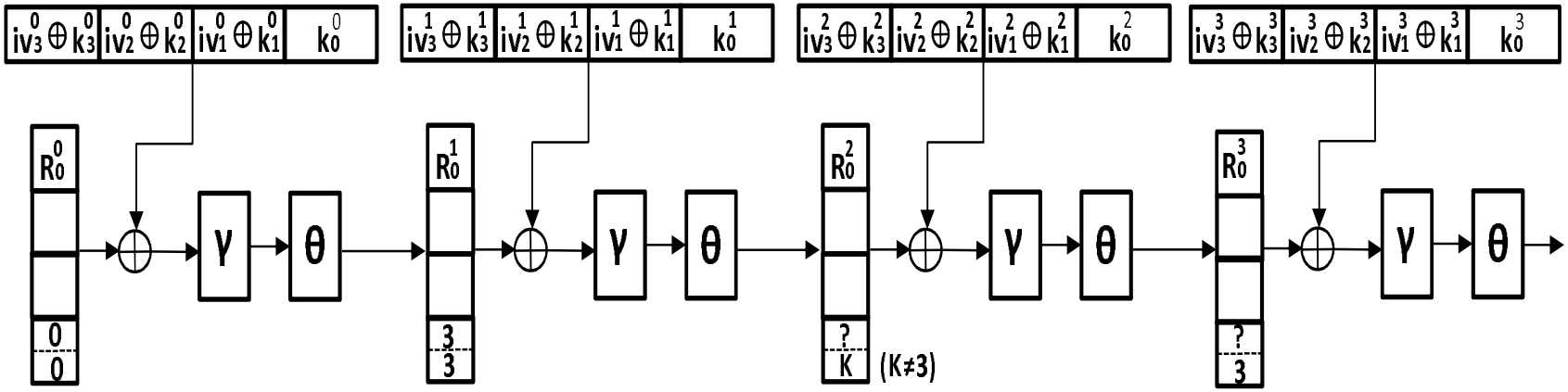
Definition

Let the *correct IVs* be the IVs for which (1) holds.



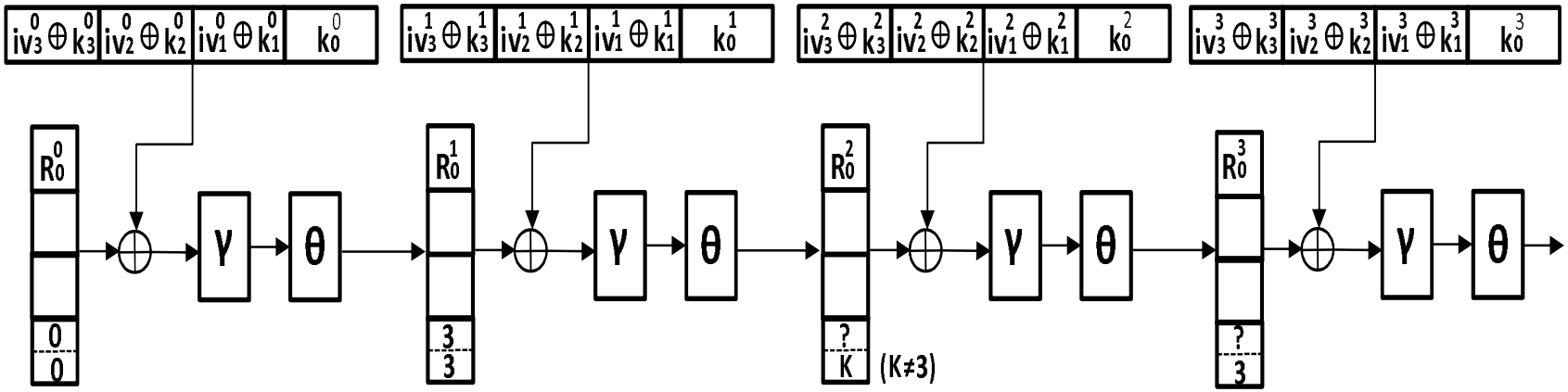
Aim: use correct IVs to recover the key.

- First 4 init steps can be seen as 4 rounds of a block cipher
- Relation $(w^0, w^1, w^2, w^3) = (0x00, 0x33, 0xK?, 0x3?)$ that holds for *correct IVs*: a block cipher inner state "leak"
- The "block cipher key" are the LFSR bytes
- Goal: recover the "block cipher key" (LFSR bytes)



To recover the "block cipher key":

- Take one *correct* IV
- Guess the 32-bit unknown "round" key
- Calculate the first-round output
- If $w^1 \neq 0x33$, discard the key guess.



- First round: the round-key elimination criterion $R^1 \gg 24 = 0x33$: $p = 2^{-8}$
- Second round: the round-key elimination criterion $R^2 \gg 28 \neq 3$: $\frac{15}{16}$
- Third round: $R^3 \gg 28 = 3$: $p = 2^{-4}$.

This specifies the number of needed *correct* IVs

Attack complexity

The attack is a related-key attack requiring

- Computational work: $\approx 2^{26}$ Loiss initializations
- Resynchronization with $\approx 2^{25.8}$ chosen-IVs
- Offline precomputation $\approx 2^{26}$ Loiss initializations
- Storage space of 2^{32} words

Conclusion

Loiss stream cipher is a byte-oriented SNOW-like cipher.

A new, efficient, component (BOMM), reminiscent of the RC4 S-box, was added.

The BOMM has bad diffusion properties.

Attack idea: cancel out the difference everywhere but in the BOMM.

Independently of our result, a similar attack idea was used by Lin Ding and Jie Guan (Computer Journal, 2012).

The new component reduced the security of the cipher.

Thank you