Evolution of Stream Ciphers towards ZUC

Subhamoy Maitra

Applied Statistics Unit
Indian Statistical Institute, Kolkata
subho@isical.ac.in

3rd Generation Partnership Project (3GPP)

3rd Generation Partnership Project (3GPP)

- Collaboration between groups of telecom associations
- Goal is to make a globally applicable 3G mobile phone system specification within the scope of the International Mobile Telecommunications-2000 project of the International Telecommunication Union (ITU)
- Release 8 (2008): First LTE release.
- Release 9 (2009): SAES Enhancements, WiMAX and LTE/UMTS Interoperability. Dual-Cell HSDPA with MIMO, Dual-Cell HSUPA.
- Release 10 (in progress): LTE Advanced fulfilling IMT Advanced 4G requirements.

Last Few Nights ...

... and At Last I Thought

Lets stick to Cryptography

Plan of this Talk

There are several discussions related to why ZUC is designed.

We look into this from a completely different viewpoint!

Plan of this Talk

There are several discussions related to why ZUC is designed.

We look into this from a completely different viewpoint!

We study the evolution of stream ciphers ...

... *and try to understand what are the technical reasons that may motivate a cryptographer to design a stream cipher like ZUC.*

We cover a few important stream ciphers in this process.

3GPP Security Algorithms

3GPP Confidentiality and Integrity Algorithms

Security architecture of 3GPP system contains standardized algorithms for confidentiality ($f8$) and integrity ($f9$).

Two sets are already specified:

1. 3GPP Release 8: 128-EEA1, 128-EIA1: Uses SNOW 3G
2. 3GPP Release 9: 128-EEA2, 128-EIA2: Uses AES 128

The third set of algorithms is designed using ZUC algorithm:

- 128-EEA3: 3GPP Confidentiality algorithm
- 128-EIA3: 3GPP Integrity algorithm

AES 128 Block Cipher

Fixed block size of 128 bits and a key size of 128.

Basic operations:

1. KeyExpansion: Rijndael key schedule
2. Initial Round: AddRoundKey
3. SubBytes, ShiftRows, MixColumns, AddRoundKey
4. Final Round: SubBytes, ShiftRows, AddRoundKey

The algorithms:

- 128-EEA2: AES in Counter mode
- 128-EIA2: AES in Cipher-based MAC (CMAC) mode

AES: Recent Cryptanalytic Results

IACR ePrint Archive (eprint.iacr.org)

- 2010/257: Feasible Attack on the 13-round AES-256. Biryukov, Khovratovich
- 2010/248: Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. Biryukov, Nikolić
- 2009/374: Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds. Biryukov, Dunkelman, Keller, Khovratovich, Shamir
- 2009/317: Related-key Cryptanalysis of the Full AES-192 and AES-256. Biryukov, Khovratovich
- 2009/242 Examples of differential multicollisions for 13 and 14 rounds of AES-256. Biryukov, Khovratovich, Nikolić
- 2009/241: Distinguisher and Related-Key Attack on the Full AES-256. Biryukov, Khovratovich, Nikolić

These results do not affect AES-128

Stream Cipher

Basic Idea

Parties: Alice (Sender/Receiver) and Bob (Receiver/Sender)

Procedure

- Alice and Bob share a stream of random data (keystream) K_i, where $i = 0, 1, \ldots$
- The plaintext stream M_i is XOR-ed with K_i to generate the cipher stream C_i. $[C_i = M_i \oplus K_i]$
- The cipher stream C_i is XOR-ed with K_i to generate the plaintext stream M_i. $[M_i = C_i \oplus K_i]$

One Time Pad

- Alice and Bob may sit on a table and toss an unbiased coin enough number of times to generate the keystream bits.
- Once some portion of the keystream is used for encryption, it will never be used again.

Not practical!

Pseudorandom Generator

- Alice and Bob share a small key
  E.g., toss the coin for 128 times to generate the secret key
- Initialize some deterministic algorithm on a classical computer with this secret key.
- After the initialization, the algorithm will keep on generating *random-looking bitstream*, the keystream bits K_i.
- The small key and K_i have a unique one-to-one correspondence.

A practical solution!

Cryptographic Security

- Kerckhoff's Principle: The security of a cipher should rely on the secrecy of the key only!
- Attacker knows every detail of the cryptographic algorithm except the key.
- Keeping the design secret in commercial domain has no scientific justification. It may be leaked easily.
- The design should be such that the designer himself cannot break the system without knowing the key. No trapdoor.
- Design should be known to everybody for evaluation.

Obscurity is the opposite of "transparency" or "transparentness". This never helps to achieve cryptographic security.

Cryptanalytic Models (General)

- **Cipher-text only attack:** Attacker knows ciphertext of several messages encrypted with the same and/or many keys.
- **Known plain-text attack:** Attacker knows "ciphertext, plaintext" pair for several messages, i.e., some amount of keystream in case of stream ciphers.
- **Chosen plain-text attack:** Attacker can choose the plaintext that gets encrypted (thereby potentially getting more information). Same as above for stream ciphers.
- **Chosen cipher-text attack:** Attacker can choose a series of ciphertexts. A decryption oracle is available and the attacker gets the plaintexts corresponding to these ciphertexts. Same as above for stream ciphers.

Cryptanalytic Models (Stream Cipher)

Required amount (say N bits) of keystream available.

- **Key Recovery Attack:** Try to recover the key. Efficiency of the attack depends on how less N is, and what is the time and space complexity of the algorithm.
- **Distinguishing Attack:** Try to distinguish the keystream from ideal random stream. Need to find an event that will distinguish. Efficiency depends on how less N is.
- **Malleability:** Transformations on the ciphertext to produce meaningful changes in the plaintext.
- **Fault Attacks:** The cipher may become weaker if random faults are injected.

Basic Design Ideas

Initial Remarks

- Involvement of *linear* and *nonlinear* elements together.
- Efficiency on Hardware and Software Platforms.
- In Hardware domain mostly LFSRs are used as linear elements and combining functions (may be with some amount of memory) are used as nonlinear elements.
- The designs of SNOW and ZUC are advanced implementation of this strategy. [May also be used efficiently in software]

Before getting into that let us concentrate on some state-of-the-art software stream ciphers to provide basic intuitions.

RC4

RC4

Designed by Ron Rivest for RSA Data Security in 1987? (Alleged RC4)

- S-Box $S = (S[0], \ldots, S[N-1])$ of length N, each location storing $\log_2 N$ bits. (typically, $N = 256$)
- A secret key k of size l bytes (typically, $5 \leq l \leq 16$).
- An array $K = (K[0], \ldots, K[N-1d])$ is used to hold the secret key, where the key is repeated in K at key length boundaries. i.e., $K[y] = k[y \bmod l]$ for $0 \leq y \leq N-1$.
- Repetition of same key makes it hard to find collision (Matsui, FSE 2009).

RC4 KSA

Input: Secret Key Array K.

Output: Random looking S-Box S generated using K.

- for $i = 0, \ldots, N-1$ $S[i] = i$;
- Initialize counter: $j = 0$;
- for $i = 0, \ldots, N-1$
  - $j = j + S[i] + K[i]$;
  - Swap $S[i] \leftrightarrow S[j]$;

Design Strategy:
Randomness is achieved by the secret key and swapping. The secret key is used upto this stage, not after that.

RC4 PRGA

Input: Random looking S-Box S generated using K.

Output: Pseudorandom keystream bytes.

- Initialize the counters: $i = j = 0$;
- While you need keystream bytes
  - $i = i + 1$;
  - $j = j + S[i]$;
  - Swap $S[i] \leftrightarrow S[j]$;
  - Output $Z = S[S[i] + S[j]]$;

Design Strategy:
Swap continues, one deterministic and one pseudorandom index.
Double indexing $Z = S[S[i] + S[j]]$ provides the nonlinearity.

RC4 Cryptanalysis

More than 20 high quality publications in over two decades.

- Most results identify weaknesses in the initial keystream bytes.
- Example $P(z_2 = 0) \approx \frac{2}{N}$, Mantin-Shamir, FSE 2001.
  - Lesson: Run the PRGA for a few initial rounds and do not use those bytes.
  - As if part of KSA. KSA requires more time.
- Mantin's distinguisher ($ABTAB$ pattern, $2^{26.5}$ bytes), Eurocrypt 2005.
- Maximov-Khovratovich state recovery attack, Time complexity 2^{241}, Crypto 2008. Can be used to recover the secret key: Maitra-Paul, SAC 2007.

RC4: Current Status

- The design is nice and simple.
- That invites a lot of cryptanalytic results.
- The cipher is well studied.
- Requires discarding some amount of initial keystream bytes.
- Till date, quite safe as 128-bit stream cipher.
- How can we securely incorporate IV and MAC in RC4?
- Hardware at the speed of one byte per clock available. Sengupta-Sinha-Maitra-Sinha, Indocrypt 2010.

Not-So-Simple Designs

- Consider that we need a word oriented (32-bit) stream cipher.
- More speed and security required.
- Not easy to maintain an array of 2^{32} locations to implement a 32-bit instance of RC4.
- More Security margin obviously requires more time/memory.
- Efficient software implementation may reduce time.

The eSTREAM Project

An effort to get secure stream ciphers satisfying current requirements:

> *ECRYPT Stream Cipher Project*
> *http://www.ecrypt.eu.org/stream/*

This multi-year effort running from 2004 to 2008 has identified a portfolio of promising new stream ciphers.

- It is expected that research on the eSTREAM submissions in general, and the portfolio ciphers in particular, will continue.
- It is also possible that changes to the eSTREAM portfolio might be needed in the future.

The eSTREAM Portfolio

The eSTREAM Portfolio (revision 1) as of September 2008

The eSTREAM portfolio has been revised and the portfolio now contains the following ciphers:

Profile 1 (SW)	Profile 2 (HW)
HC-128	Grain v1
Rabbit	MICKEY v2
Salsa20/12	Trivium
SOSEMANUK	

HC-128

HC-128

Designed by Hongjun Wu

[Scaled down version of HC-256 (FSE 2004)]

- Synchronous stream-cipher with 32-bit word output per step
- A software stream cipher, available at
  http://www.ecrypt.eu.org/stream/hcp3.html
- 128-bit secret key
- The key and IV setup takes about 27,300 clock cycles
- Encryption speed is 3.05 cycles/byte on Pentium M processor
- No cryptanalytic result that can contest the claimed security conjectures by the designer

Notation

$+ : x + y$ means $x + y \bmod 2^{32}$, where $0 \leq x, y < 2^{32}$

$\boxminus : x \boxminus y$ means $x - y \bmod 512$.

$\oplus$: bit-wise exclusive OR.

$\|$: concatenation.

$x \gg n$: right shift operator, x being right shifted n bits.

$x \ll n$: left shift operator, x being left shifted n bits.

$x \ggg n$: right rotation operator. $x \ggg n$ means $((x \gg n) \oplus (x \ll (32 - n))$, where $0 \leq n < 32$, $0 \leq x < 2^{32}$.

$\lll$: left rotation operator. $x \lll n$ means $((x \ll n) \oplus (x \gg (32 - n))$.

Data Structures

- Two tables P and Q, each with 512 many 32-bit elements are used as internal states of HC-128.
- A 128-bit key array $K[0, \ldots, 3]$ and a 128-bit initialization vector $IV[0, \ldots, 3]$ are used, where each entry of the array is a 32-bit element.
- s_t denotes the keystream word generated at the t-th step, $t = 0, 1, 2, \ldots$.

Functions

$$f_1(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3),$$
$$f_2(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10),$$

$$g_1(x, y, z) = ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8),$$
$$g_2(x, y, z) = ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8),$$

$$h_1(x) = Q[x^{(0)}] + Q[256 + x^{(2)}],$$
$$h_2(x) = P[x^{(0)}] + P[256 + x^{(2)}],$$

where $x = x^{(3)} \| x^{(2)} \| x^{(1)} \| x^{(0)}$ is a 32-bit word with four bytes: $x^{(0)}$ (least significant) , $x^{(1)}, x^{(2)}$ and $x^{(3)}$ (most significant)

Key and IV setup

Secret key: $K[0, \ldots, 3]$
Initialization vector: $IV[0, \ldots, 3]$
$K[i+4] = K[i]$ and $IV[i+4] = IV[i]$ for $0 \leq i \leq 3$.

Repetition of same key & IV.
While coming back in KSA, one gets stuck here.

The key and IV are expanded into an array $W[0, \ldots, 1279]$ as:

$$W[i] = \begin{cases} K[i] & 0 \leq i \leq 7; \\ IV[i-8] & 8 \leq i \leq 15; \\ f_2(W[i-2]) + W[i-7] + & \\ f_1(W[i-15]) + W[i-16] + i & 16 \leq i \leq 1279 \end{cases}$$

Key and IV setup (cont'd.)

Update the tables P and Q with the array W as follows.

$$P[i] = W[i + 256], \text{ for } 0 \leq i \leq 511$$
$$Q[i] = W[i + 768], \text{ for } 0 \leq i \leq 511$$

Run 1024 steps and use the outputs to replace the table elements:

$$P[i] = (P[i] + g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511])) \oplus h_1(P[i \boxminus 12])$$
for $i = 0$ to 511

$$Q[i] = (Q[i] + g_2(Q[i \boxminus 3], Q[i \boxminus 10], Q[i \boxminus 511])) \oplus h_2(Q[i \boxminus 12])$$
for $i = 0$ to 511

The Keystream Generation Algorithm

$i = 0$;
repeat until enough keystream bits are generated {
    $j = i \bmod 512$;
    if $(i \bmod 1024) < 512$ {
        $P[j] = P[j] + g_1(P[j \boxminus 3], P[j \boxminus 10], P[j \boxminus 511])$;
        $s_i = h_1(P[j \boxminus 12]) \oplus P[j]$;
    }
    else {
        $Q[j] = Q[j] + g_2(Q[j \boxminus 3], Q[j \boxminus 10], Q[j \boxminus 511])$;
        $s_i = h_2(Q[j \boxminus 12]) \oplus Q[j]$;
    }
    end-if
    $i = i + 1$;
}
end-repeat

Cryptanalytic Results on HC-128

- Wu, the designer of HC-128, presented a distinguisher that requires 2^{156} keystream words. That is based on the 0-th bit.
- Extended to all other bits of the words by Maitra - Paul - Raizada - Sen - Sengupta (WCC 2009, accepted in DCC).
- Observation by Dunkelman in the eStream discussion forum:
  A small observation on HC-128.
  http://www.ecrypt.eu.org/stream/phorum/read.php?1,1143
  (Date: November 14, 2007)
  Shows that the keystream words of HC-128 leak information regarding secret states.
- Also been sharpened by Maitra-Paul-Raizada-Sen-Sengupta

XOR and Modulo Addition

XOR-Approximation of Binary Addition

- Let $X^{(i)}$ denote the i-th bit of an integer X, $i \geq 0$ ($i = 0$ stands for the LSB).
- Let $X_1, X_2, \ldots, X_n$ be n independent and uniformly distributed integers.
- Define $S_n = \sum_{k=1}^{n} X_k$ and $L_n = \bigoplus_{k=1}^{n} X_k$.
- Denote $p_n^i = \Pr\left(S_n^{(i)} = L_n^{(i)}\right)$.
- Denote $p_n = \lim_{i \to \infty} p_n^i$

Values of p_n^i for some values of n and i

$n \setminus i$	1	2	3	4
2	0.75	0.625	0.5625	0.53125
3	0.5	0.375	0.34375	0.335938
4	0.375	0.390625	0.439453	0.468994

Theoretical Results

- $p_n^0 = 1$, the LSB is same for both modulo sum and XOR
- $p_2^i = \frac{1}{2} + \frac{1}{2^{i+1}}$, $p_2 = \frac{1}{2}$
- $p_3^i = \frac{1}{3}(1 + \frac{1}{2^{2i-1}})$, i.e., $p_3 = \frac{1}{3}$.
  S. Maitra, G. Paul, S. Raizada, S. Sen, R. Sengupta. Some Observations on HC-128. To be published in Designs, Codes and Cryptography.
- Modulo additions provide nonlinearity.

Keeping one addition (SNOW 3G, ZUC) better than two (HC-128).

Theoretical Results (cont'd.)

- For even n, $p_n = \frac{1}{2}$.
- For odd n, $p_n \rightarrow \frac{1}{2}$ as $n \rightarrow \infty$.
- For odd n, n small, p_n may not be close to $\frac{1}{2}$.
- Detailed general results are available at eprint archive 2009/047: P. Sarkar. On Approximating Addition by Exclusive OR.

LFSR Based Stream Ciphers

Bit-oriented LFSR

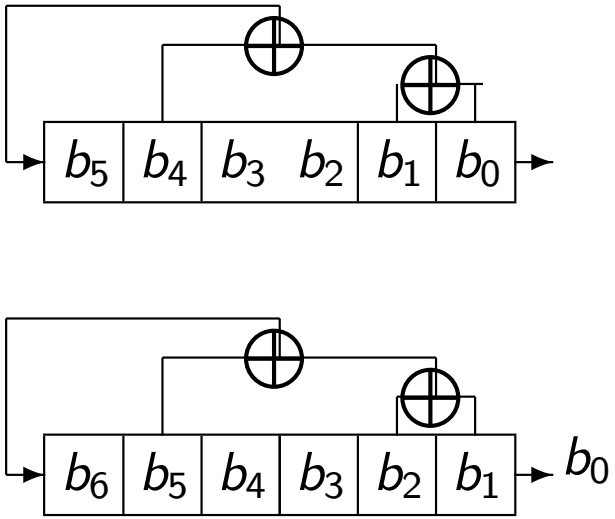


Figure: LFSR: One step evolution

- Recurrence Relation: $s_{t+6} = s_{t+4} \oplus s_{t+1} \oplus s_t$
- Polynomial over $GF(2)$: $x^6 + x^4 + x^1 + 1$

Bit-oriented LFSR (cont'd.)

- Primitive polynomial provides maximum length cycle, $2^d - 1$ for degree d. Well known as m-sequence.
- By itself, not cryptographically secure, but useful building block for pseudorandomness.
- In the domain of communications, known as p-n sequence.
- Easy and efficient implementation in hardware, using registers (Flip Flops) and simple logic gates.
- Deep mathematical development for a long time.
- Elegant results in the area of Linear Complexity.

Nonlinear Combiner Model

- Take n LFSRs of different length (may be pairwise prime).
- Initialize them with seeds.
- In each clock, take the n-many outputs from the LFSRs, which are fed as n-inputs to an n-variable Boolean function.
- May be some memory element is added.

Nonlinear Filter-Generator Model

- Take one LFSR.
- Initialize that with a seed.
- In each clock, take the n-many outputs from the LFSR from different locations, which are fed as n-inputs to an n-variable Boolean function.
- May be considered with additional memory element.
- The Boolean function and memory together form a Finite State Machine.

Boolean Functions

Basics

Formally: $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$, where $\mathbb{F}_2 = GF(2)$ is a finite field.
It can be viewed as $f : \{0,1\}^n \rightarrow \{0,1\}$
Interpreted as the output column of its *truth table*

X_3	X_2	X_1	$f(X_3, X_2, X_1)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

$$f = [f(0,0,0), f(1,0,0), \ldots, f(1,1,1)] = [1,1,0,1,1,0,0,0]$$

Representation

Truth Table Representation

A binary string of length 2^n (evaluation of f at all inputs)

$$f(X_1, X_2, \ldots, X_n) = [f(0, 0, \cdots, 0), f(1, 0, \cdots, 0), f(0, 1, \cdots, 0), \ldots, f(1, 1, \cdots, 1)]$$

ANF Representation

Algebraic Normal Form (n-variable polynomial over $\mathbb{F}_2$)

$$f = a_0 \oplus \sum_{i=1}^{n} a_i X_i \oplus \sum_{1 \le i < j \le n} a_{ij} X_i X_j \oplus \cdots \oplus a_{1\ldots n} X_1 \ldots X_n$$

$a_0, a_{ij}, \ldots, a_{1\ldots n} \in \mathbb{F}_2 = \{0, 1\}$ and $\oplus$ in $\mathbb{F}_2$ means XOR

Example: $f = 1 \oplus X_2 \oplus X_1 X_3 \oplus X_2 X_3$

Affine Functions

Let $f = a_0 \oplus \sum_{i=1}^{n} a_i X_i \oplus \sum_{1 \le i < j \le n} a_{ij} X_i X_j \oplus \cdots \oplus a_{1\ldots n} X_1 \ldots X_n \in \Omega_n$

and define $\deg(f)$ = the degree of this ANF polynomial

Affine Functions: $A_n = \{f \in \Omega_n : \deg(f) \le 1\}$

$$f = a_0 \oplus \sum_{i=1}^{n} a_i X_i = a_0 \oplus a_1 X_1 \oplus \cdots \oplus a_n X_n$$

Linear Functions: $L_n = \{f \in A_n : a_0 = 0\}$

$$f = \sum_{i=1}^{n} a_i X_i = a_1 X_1 \oplus \cdots \oplus a_n X_n$$

$|A_n| = 2^{n+1}$, $|L_n| = 2^n$ and $A_n = L_n \cup \overline{L_n}$

Nonlinearity

DEFINITION: Measure of Nonlinearity = Distance of f from A_n.

NOTION OF DISTANCE: Natural Hamming distance

$$dist(f, g) = |\{x \in \{0,1\}^n : f(x) \neq g(x)\}| = wt(f \oplus g)$$

NONLINEARITY: Minimum distance from A_n,

$$nl(f) = \min_{l \in A_n} dist(f, l)$$

Walsh Transform

Suppose that $X, \omega \in \{0,1\}^n$ are two vectors in $\mathbb{F}_2^n$
Represent $X = (X_1, X_2, \ldots, X_n)$ and $\omega = (\omega_1, \omega_2, \ldots, \omega_n)$

Inner Product: $X \cdot \omega = X_1\omega_1 \oplus X_2\omega_2 \oplus \cdots \oplus X_n\omega_n$

Walsh Transform of $f \in \Omega_n$ is defined as

$$W_f(\omega) = \sum_{X \in \{0,1\}^n} (-1)^{f(X) \oplus X \cdot \omega}$$

Nonlinearity can be related to Walsh Transform as

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{\omega \in \{0,1\}^n} |W_f(\omega)|$$

Important Cryptographic Properties

- Balancedness: Necessary to achieve Pseudo-Random sequence
- Algebraic Degree: To achieve high Linear Complexity
- Nonlinearity: For higher Confusion and resistance against: Best Affine Approximation (BAA) Attack and Linear Cryptanalysis.
- Autocorrelation: To achieve higher Diffusion, and to resist Differential Cryptanalysis. PC, SAC, differential uniformity.
  $x \to F(x)$ is a permutation of $\mathbb{F}_{2^n}$. Differential uniformity requires $\#\{x|F(x)+F(x+\alpha)=\beta\}$ must be small for any β (for any fixed nonzero α).
- Correlation Immunity: To resist Correlation Attack
- Algebraic Immunity: To resist Algebraic Attack

Nonlinear Filter Generator Model With Memory

Current Trend: State-of-the-art View

- Concept: More than one bit processed together (32-bit words)
- Use LFSRs over larger fields: need the LFSR evolution operations to be efficient.
- $GF(2^{32})$ or $GF(2^{31}-1)$ to relate with 32-bit words of modern processors. Are we moving towards 64-bit words?
- FSM contains S-boxes and Registers.
- Registers are memory words.
- S-boxes are multiple output Boolean functions.

SNOW and ZUC: SAGE's view

SAGE: Security Algorithms Group of Experts

- One stated objective for the design was that the new algorithms be substantially different from the first and second LTE algorithm sets, in such a way that an attack on any one algorithm set would be unlikely to lead to an attack on either of the others.
- In SAGE's view this objective is not fully met - there are some architectural similarities between ZUC and SNOW 3G, and it is possible that a major advance in cryptanalysis might affect them both.
- However, there are important differences too, so ZUC and SNOW 3G by no means "stand or fall together".

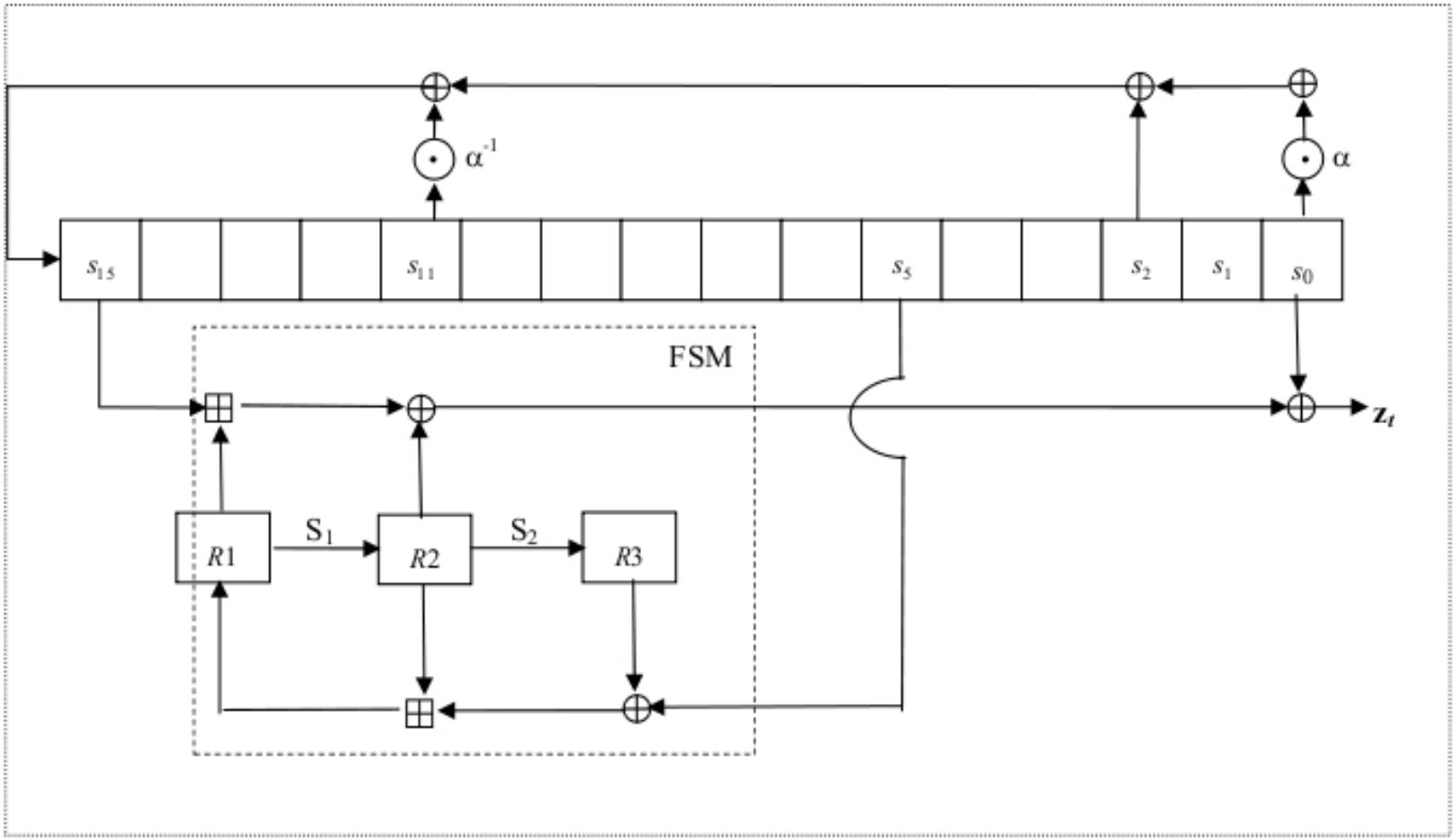
SNOW 3G

SNOW 3G Stream Cipher

LFSR based stream cipher: 32-bit words with 128-bit key.

- An LFSR of 32-bit words, length 16
- A Finite State Machine (FSM) as a non-linear model
- Based on the earlier versions SNOW 1.0 and SNOW 2.0
- Derived from the stream cipher SNOW 2.0, with improvements against algebraic cryptanalysis and distinguishing attacks.
- SNOW 1.0, SNOW 2.0, and SNOW 3G are developed by Thomas Johansson and Patrik Ekdahl.

SNOW 3G Structure



SNOW 3G: Simple Analysis

- $Z_t = (s_{15,t} \boxplus R1_t) \oplus R2_t \oplus s_{0,t}$
- Approximation: $Z_t \approx (s_{15,t} \oplus R1_t) \oplus R2_t \oplus s_{0,t}$
- If $R1_t = R2_t$ (happens with probability $\frac{1}{2^{32}}$), then $Z_t \approx s_{15,t} \oplus s_{0,t}$.

Better understanding of $R1, R2$ may provide nontrivial results relating the keystream words and LFSR words.

SNOW 3G: Simple Analysis (cont'd.)

- $Z_t = (s_{15,t} \boxplus R1_t) \oplus R2_t \oplus s_{0,t}$
- Two values directly from the LFSR
- Two values from the registers
- Let us have the term "directly use" for the LFSR words that are XOR-ed/Added to generate the keystream words. Here such terms are $s_{15,t}$, and $s_{0,t}$.
- A word of the LFSR is "directly used" twice to generate two different keywords which are 15 clocks apart.
- Let us have the term "indirectly use" for the words that are flowed to the FSM. Here such term is $s_{5,t}$.

SNOW 3G: Simple Analysis (cont'd.)

- $Z_t = (s_{15,t} \boxplus R1_t) \oplus R2_t \oplus s_{0,t} \approx (s_{15,t} \oplus R1_t) \oplus R2_t \oplus s_{0,t}$
- $Z_{t+15} \approx (s_{15,t+15} \oplus R1_{t+15}) \oplus R2_{t+15} \oplus s_{0,t+15} = (s_{15,t+15} \oplus R1_{t+15}) \oplus R2_{t+15} \oplus s_{15,t}$
- $Z_t \oplus Z_{t+15} \approx (s_{0,t} \oplus s_{15,t+15}) \oplus (R1_t \oplus R2_t \oplus R1_{t+15} \oplus R2_{t+15})$.
- If $(R1_t \oplus R2_t \oplus R1_{t+15} \oplus R2_{t+15}) = 0$ (happens with probability $\frac{1}{2^{32}}$), then $Z_t \oplus Z_{t+15} \approx (s_{0,t} \oplus s_{15,t+15})$

Better understanding of $R1, R2$ may provide nontrivial results relating the keystream words and LFSR words.

SNOW 3G: Fault Analysis

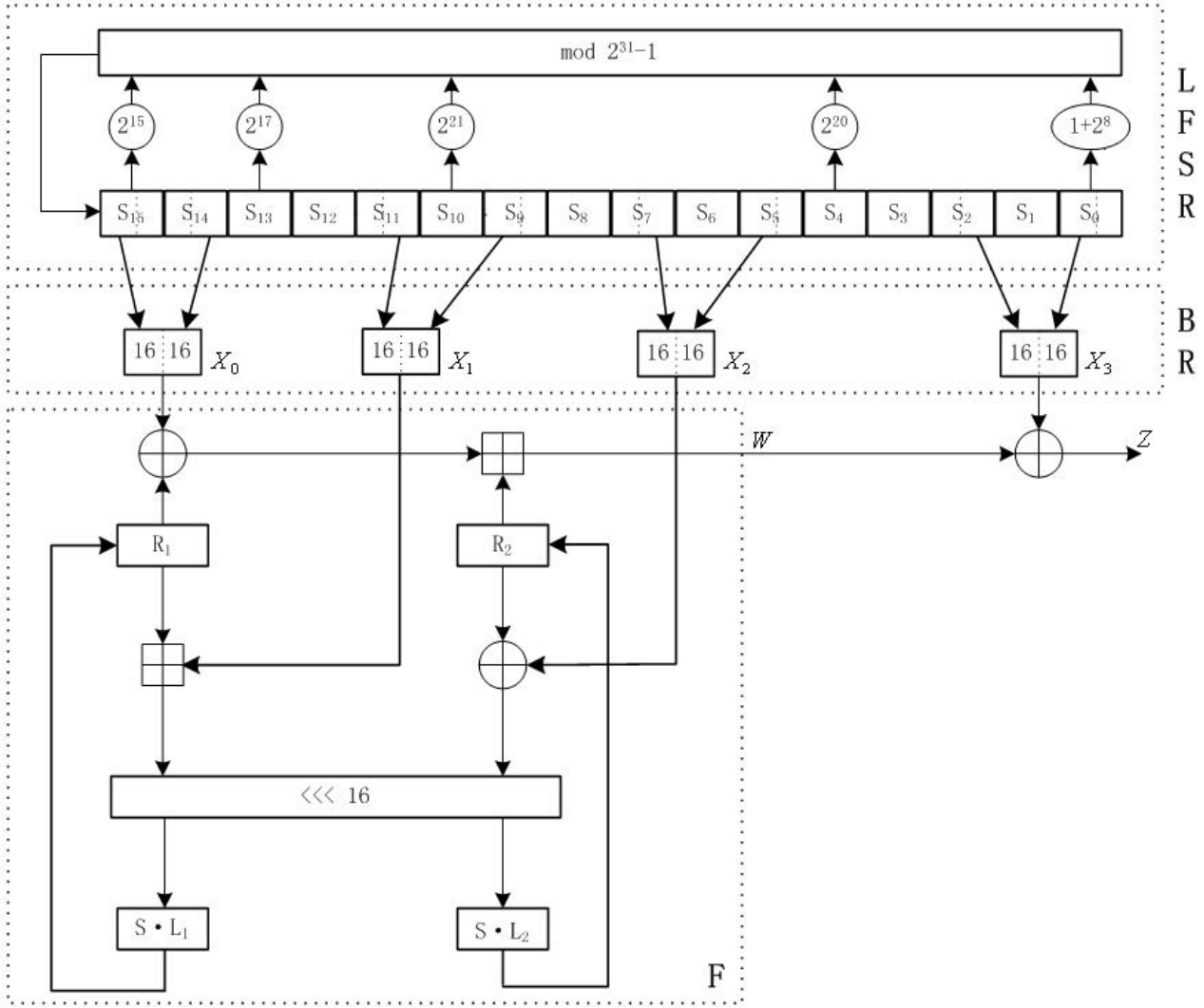
- B. Debraize, I. M. Corbella: Fault Analysis of the Stream Cipher SNOW 3G. FDTC 2009.
- The attack claims to recover the secret key with only 22 fault injections.
- No other attack is known against SNOW 3G today.

ZUC

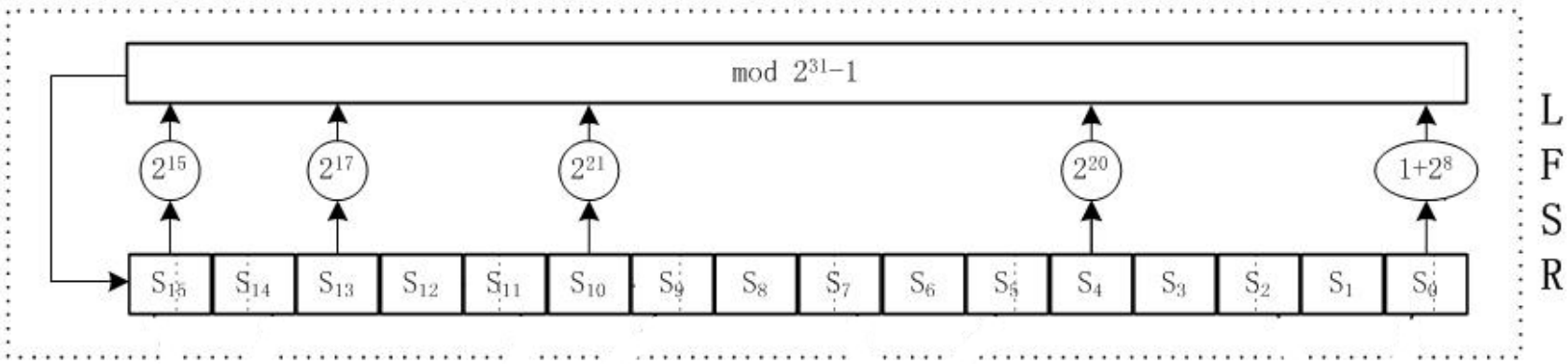
ZUC Algorithm

- LFSR based Stream Cipher
- 31-bit LFSR words
- 32-bit keystream words
- 128-bit key
- A Finite State Machine (FSM) as a non-linear core

ZUC Algorithm



ZUC LFSR



Mentioned in Design and Evaluation Report (v1.1, pp. 17/40)

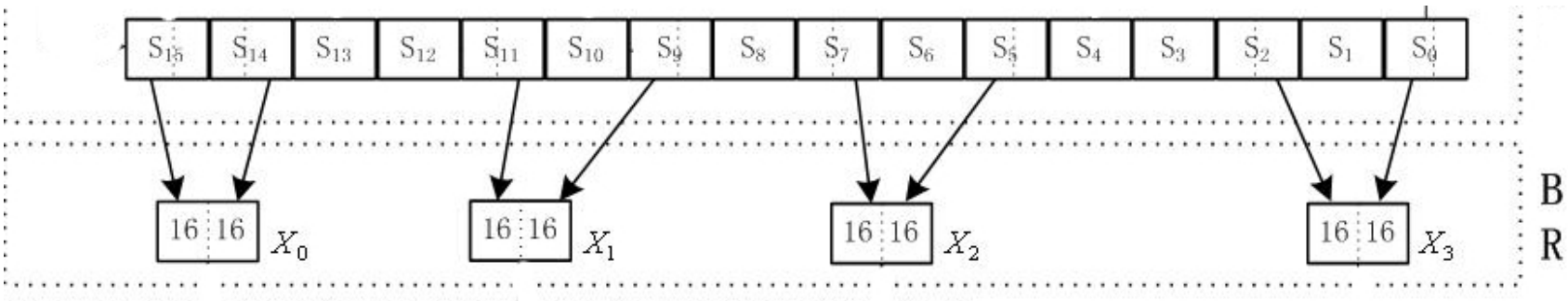
- Period of each coordinate sequence generated by ZUC is around 2^{496}.
- Linear complexity of the coordinate sequences is $\frac{p(p^{16}-1)}{2(p-1)}$, where $p = 2^{31} - 1$.

We require further study in this area.

LFSR loading

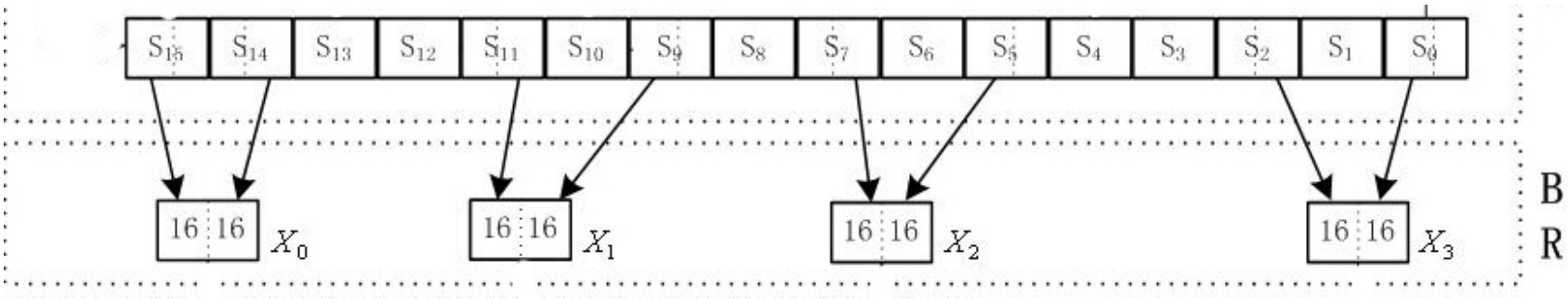
- $k = k_0 \| k_1 \| k_2 \| \ldots \| k_{15}$
- $k = iv_0 \| iv_1 \| iv_2 \| \ldots \| iv_{15}$
- $s_i = k_i \| d_i \| iv_i$
- $31 = 8 + 15 + 8$
- What if d_i's are created by some mixing of k_j and iv_l? As example: $d_i = d_i' \| (k_j \boxplus iv_l)$, d_i' is 7 bits.
- This may produce certain kinds of repetition of the keybits as in software stream ciphers RC4 & HC-128.

ZUC BR (Bit Reorganization)



- $X_0 = S_{15H} \| S_{14L}$;
- $X_1 = S_{11H} \| S_{9H}$;
- $X_2 = S_{7L} \| S_{5H}$;
- $X_3 = S_{2L} \| S_{0H}$;

ZUC Analysis

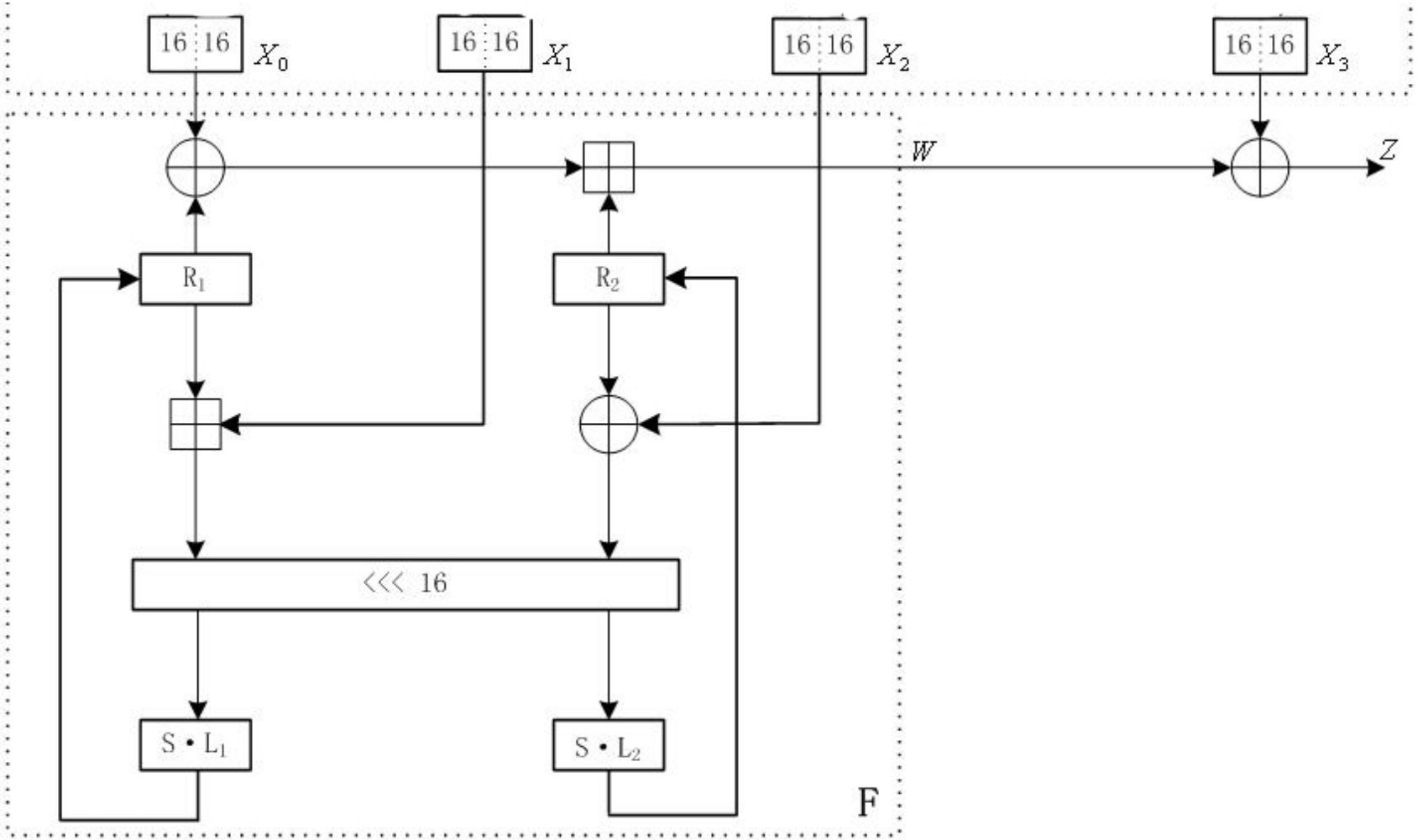


$$S_{15,t}^{(15)} = X_{0,t}^{(16)} = X_{0,t+1}^{(15)} = X_{1,t+4}^{(31)} = X_{1,t+6}^{(0)}$$
$$= X_{2,t+8}^{(31)} = X_{2,t+10}^{(0)} = X_{3,t+13}^{(31)} = X_{3,t+15}^{(0)}$$

Note that $X_{0,t}^{(16)} = X_{0,t+1}^{(15)} = X_{3,t+13}^{(31)} = X_{3,t+15}^{(0)} = S_{15,t}^{(15)}$ are used directly from the LFSR.

Same LFSR bit used 4 times in 4 different keystream words.

ZUC FSM



ZUC FSM

$F(X_0, X_1, X_2)$

- $W = (X_0 \oplus R_1) \boxplus R_2$;
- $W_1 = R_1 \boxplus X_1$;
- $W_2 = R_2 \oplus X_2$;
- $R_1 = S(L_1(W_{1L} \| W_{2H}))$;
- $R_2 = S(L_2(W_{2L} \| W_{1H}))$;

S is a 32×32 S-box, L_1 and L_2 are linear transformations.

S and L_1, L_2

The S-Box:

- S is composed by 4 juxtaposed 8×8 S-boxes, $S = (S_0, S_1, S_0, S_1)$.
- For S_0, its nonlinearity, differential uniformity, algebraic degree and algebraic immunity are 96, 8, 5 and 2 respectively.

  Suboptimal: for easy hardware implementation.
- For S_1, its nonlinearity, differential uniformity, algebraic degree and algebraic immunity are 112, 4, 7 and 2 respectively.

$$L_1(X) = X \oplus (X \lll 2) \oplus (X \lll 10) \oplus (X \lll 18) \oplus (X \oplus 24)$$
$$L_2(X) = X \oplus (X \lll 8) \oplus (X \lll 14) \oplus (X \lll 22) \oplus (X \lll 30)$$

The Integrity Algorithm (Basic Idea)

- Given a key, generate the keystream $k_0, \ldots, k_{t-1}$, say.
- Generate b-bit words $w_0 = k_0, \ldots, k_{b-1}$, $w_1 = k_1, \ldots, k_b$, $\ldots$, $w_{t-b} = k_{t-b}, \ldots, k_{t-1}$. Sliding technique.
- Message $m_0, \ldots, m_{u-1}$, $u < t$.
- tag is b bit word, initialized to zero, say.
- for $i = 0$ to $u - 1$, if $m_i = 1$ the $tag = tag \oplus w_i$.

$H_k(M) \rightarrow tag$: Universal hash function with collision probability $\frac{1}{2^b}$

EIA3: $b = 32$ is fixed. This gives a collision probability of 2^{-32}.

Key-ed Hash

The construction in case of EIA-3 can be seen as a Toeplitz version on top of Gilbert-MacWilliams-Sloane (GMS) or Carter-Wegman hash over $GF(2)$.

Proof of the collision probability:
"A New Multi-Linear Universal Hash Family", P. Sarkar, Aug 2010
A more general treatment is available in the above paper, which generalizes the GMS construction.

Points to note:

- Hardware efficiency may be a trouble as it operates bit-by-bit.
- Speed may be increased by using LFSRs and few logic gates as suggested in the paper by Sarkar, but the design also needs to be changed slightly.

Concluding Remarks

- Needs more time to study ZUC.
- Security of the cipher is a conjecture, as it is true for many other ciphers.
- SNOW had the opportunity to be evolved with several versions. Same thing for HC-128.
- Is there any possibility for minor design modifications of ZUC?
  - Key repetition during LFSR loading
  - More carefully designed BR
  - Redesigning MAC for hardware efficiency

Acknowledgments

I would like to express my gratitude to

- Prof. Dongdai Lin for kindly inviting me.
- Prof. Claude Carlet for being involved in the process.
- Mr. Sourav Sen Gupta (fun images), Mr. Santanu Sarkar, and Dr. Goutam K. Paul for providing inputs on this presentation.

Thank You

and

Best Wishes for ZUC