Recent Results on Stream Ciphers

Subhamoy Maitra

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

26 August, 2020

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 should have a unique one-to-one correspondence.
- Key 128 bits, key-stream 2048 bits, not all the key-stream patterns can be generated.

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.
- For stream cipher the attacker will have access to certain amount of key-stream

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

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]

Hardware Stream Ciphers

- LFSR, NFSR, Boolean Functions
- Keys involved only during KSA
- PRGA does not involve keys
- The state size must be twice the key size to protect against Generic TMDTO attack.
- Total space required:
  - The space to store the key (might be non-volatile)
  - The space for LFSR, NFSR, Counter (volatile)

Lightweight Stream Ciphers: New Direction

- Secret key fixed with the device (Where is the key stored?)
- Use secret key during PRGA
- Cost of non-volatile memory less
- Reduce size of volatile memory

Comparisons

Cipher	Key size	IV size	State size	Initialization rounds
Lizard	120(80)	64	121 (90 NFSR + 31 NFSR)	256
Plantlet	80	90	101 (61 LFSR + 40 NFSR)	320
Sprout	80	70	80 (40 LFSR + 40 NFSR)	320
Grain v1	80	64	160 (80 LFSR + 80 NFSR)	160

Table: Comparison of Plantlet with its predecessors in terms of LFSR and NFSR sizes.

Present Situation

- Grain v1 had some cryptanalysis very recently
- Sprout (FSE 2015) immediately attacked
- Plantlet (based on Sprout) and Lizard were presented in FSE 2017
- We have checked Plantlet is weaker than Sprout in terms of Fault Attack (IEEE TC 2017)
- We have mounted a TMDTO attack on Lizard (IEEE TC 2018)

LFSR Based Stream Ciphers

Bit-oriented LFSR

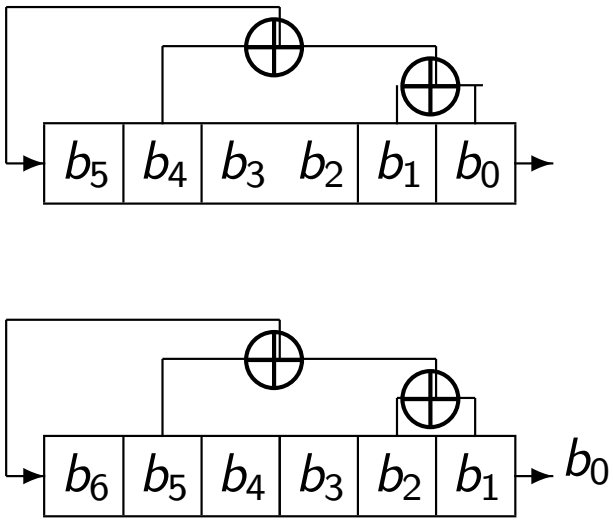


Figure: LFSR: One step evolution

- 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.)

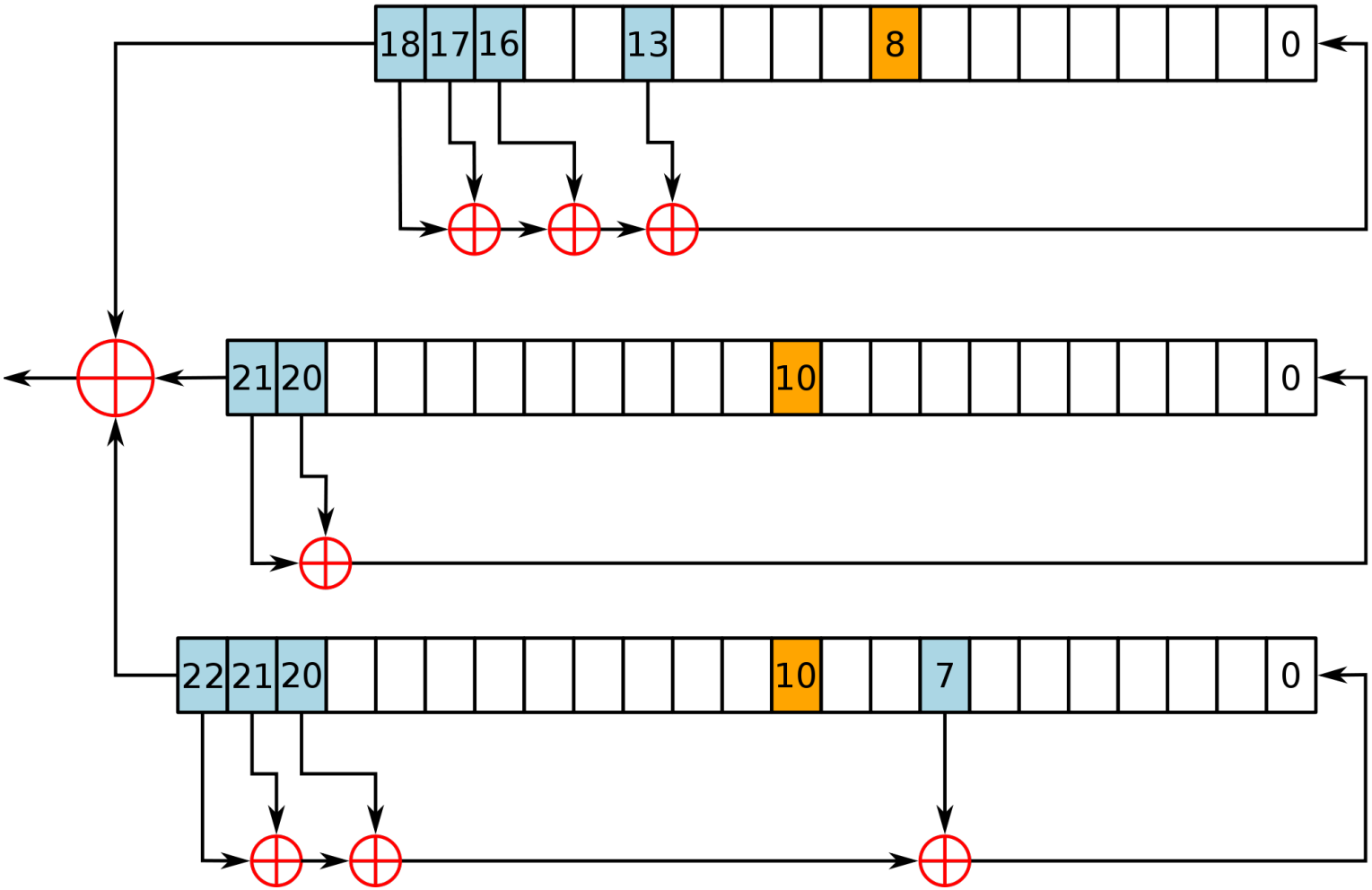
- Primitive polynomial provides maximum length cycle, $2^d - 1$ for degree d. Well known as m-sequence.
- By itself, not cryptographically secure, but useful building block for pseudo randomness.
- In the domain of communications, known as p-n sequence.
- Easy and efficient implementation in hardware,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

An Example: A5/1 (1987)

- Used in GSM Mobile in Europe and USA.
- 64-bit key and 22-bit frame number.
- Three irregularly clocked LFSR's.

LFSR Length	Connection polynomial	Clocking bit
19	$x^{19} + x^5 + x^2 + x + 1$	8
22	$x^{22} + x + 1$	10
23	$x^{23} + x^{15} + x^2 + x + 1$	10

A5/1 (1987)



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.

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.
- Here the Hardware is not constrained.

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 SAGEs 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".

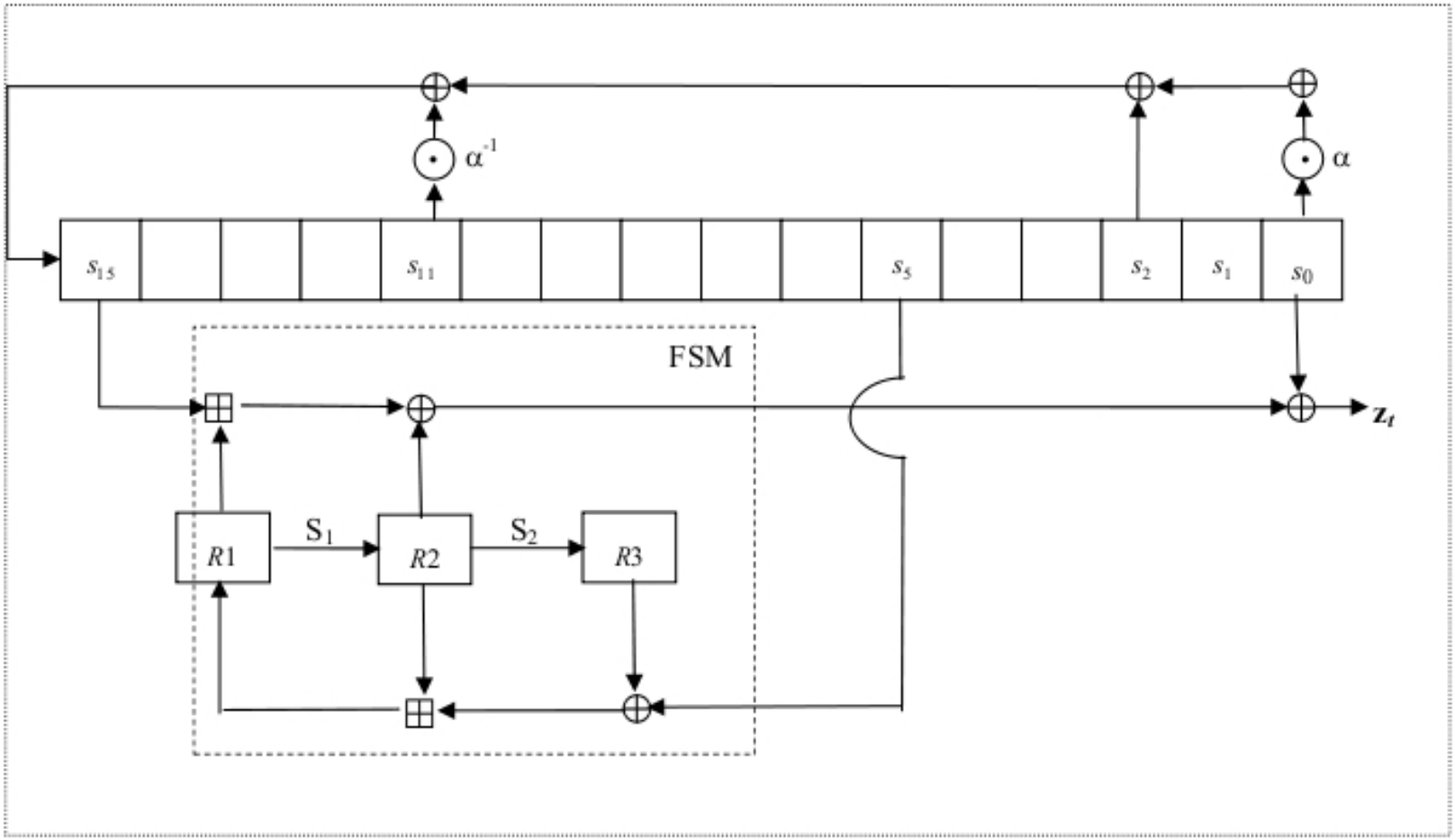
SNOW 3G

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

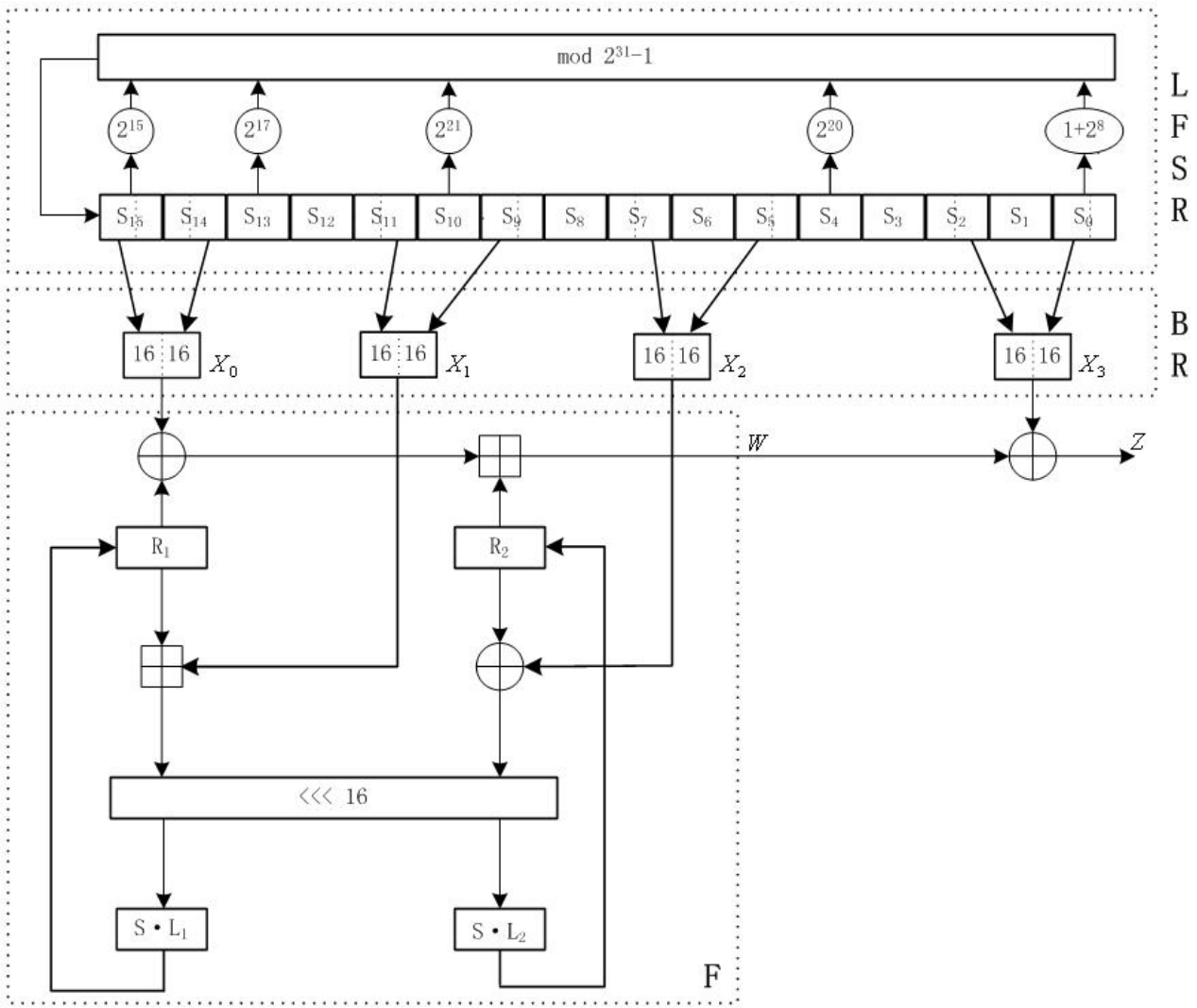
- B. Debraize, I. M. Corbella: Fault Analysis of the Stream Cipher SNOW 3G. FDTC 2009.
- The attack claims to recover the secret key with only 22 fault injections.
- No other attack is known against SNOW 3G today.

ZUC

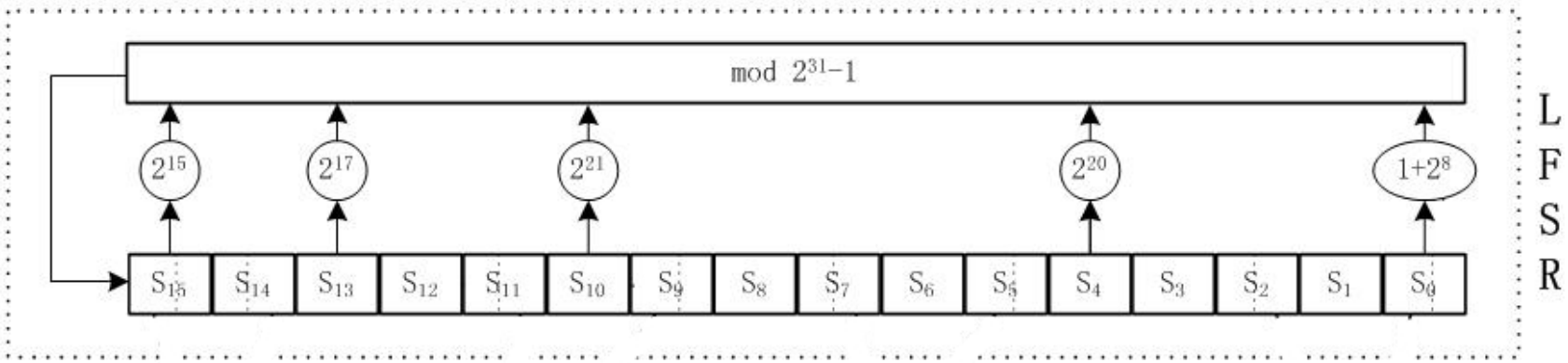
ZUC Algorithm

- 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



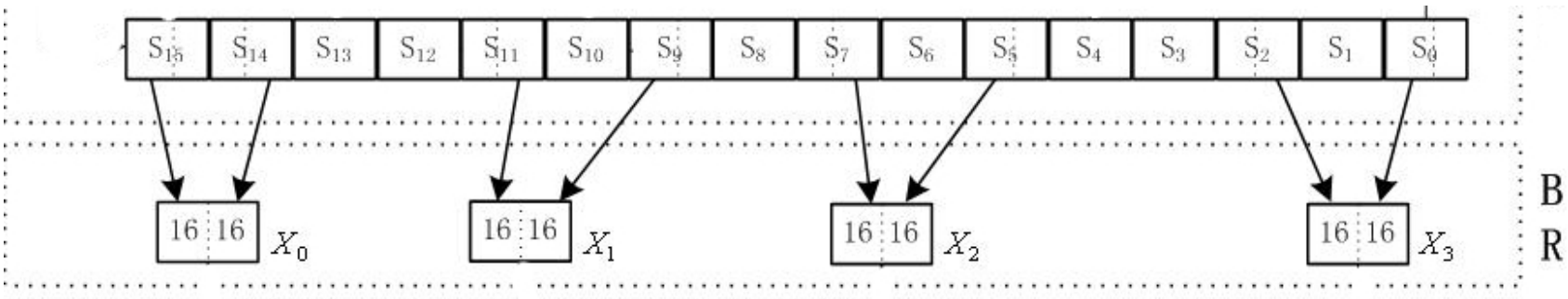
Mentioned in Design and Evaluation Report (v1.1, pp. 17/40)

- 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 $p = 2^{31} - 1$.

LFSR loading

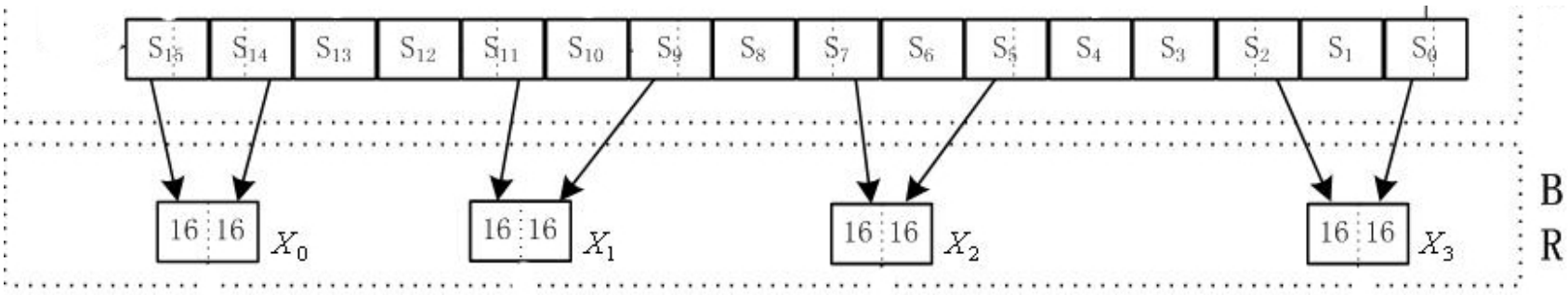
- $k = k_0 \| k_1 \| k_2 \| \ldots \| k_{15}$
- $k = iv_0 \| iv_1 \| iv_2 \| \ldots \| iv_{15}$
- $s_i = k_i \| d_i \| iv_i$
- $31 = 8 + 15 + 8$
- What if d_i's are created by some mixing of k_j and iv_l? As example: $d_i = d_i' \| (k_j \boxplus iv_l)$, d_i' is 7 bits.
- This may produce certain kinds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

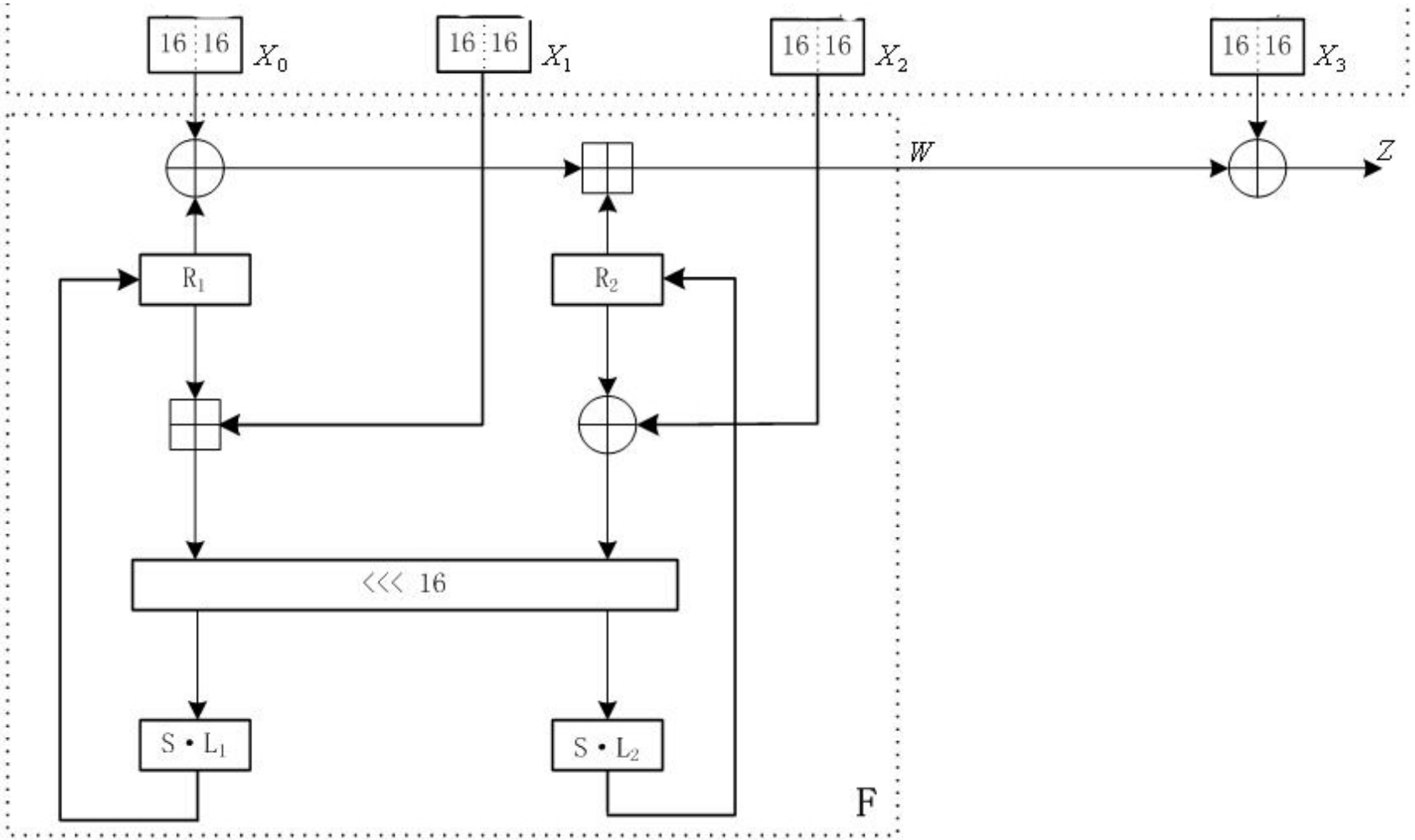


$$S_{15,t}^{(15)} = X_{0,t}^{(16)} = X_{0,t+1}^{(15)} = X_{1,t+4}^{(31)} = X_{1,t+6}^{(0)}$$
$$= X_{2,t+8}^{(31)} = X_{2,t+10}^{(0)} = X_{3,t+13}^{(31)} = X_{3,t+15}^{(0)}$$

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 $w_0 = k_0, \ldots, k_{b-1}$, $w_1 = k_1, \ldots, k_b$, ..., $w_{t-b} = k_{t-b}, \ldots, k_{t-1}$. 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}.

Software Stream Ciphers

- No space constraint
- ARX strategy: Add, Rotate, XOR
- Add is the only nonlinear operation
- Very fast in any standard processor
- Speed of KSA vs Speed of PRGA

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 40 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.
- Recent attacks on WPA and TLS; our work in JoC.

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.
- Needs to be replaced in several applications.
- Possible replacement by ChaCha

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 yet other than 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

Salsa20

Introduction

- Salsa20 was designed by Bernstein in 2005 as a candidate for eStream
- `http://cr.yp.to/snuffle.html`
- Salsa20/12 has been accepted in the eStream software portfolio
- Attacks till 8 rounds are known
- Revised to design ChaCha, which is being used in many standards now

Data Structure

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}.$$

Each word is of 32 bits. Total 16 words.

256-bit key

- $c_0 = 0x61707865, c_1 = 0x3320646e,$
  $c_2 = 0x79622d32, c_3 = 0x6b206574,$
- 256-bit key $k_0, \ldots, k_7$
- 64-bit nonce v_0, v_1
- 64-bit counter t_0, t_1
- Since this is with 256-bit keys, we can refer it as 256-bit Salsa20.

128-bit key

- One can use the same cipher with 128-bit key, where $k_i = k_{i+4}$, for $0 \leq i \leq 3$
- $c_0 = 0x61707865, c_1 = 0x3120646e$, $c_2 = 0x79622d36, c_3 = 0x6b206574$.
- One may note the little differences in c_1, c_2 for the 128-bit and 256-bit version.

Quarter Round

- The basic nonlinear operation of Salsa20 is the quarterround function.
- Each quarterround(a, b, c, d) consists of four ARX rounds, each of which comprises of one addition (A), one cyclic left rotation (R) and one XOR (X) operation as given below.

$$\left.\begin{aligned} b &= b \oplus ((a + d) \lll 7), \\ c &= c \oplus ((b + a) \lll 9), \\ d &= d \oplus ((c + b) \lll 13), \\ a &= a \oplus ((d + c) \lll 18). \end{aligned}\right\} \quad (1)$$

Row & Column Round

- Each columnround works as four quarterrounds on each of the four columns of the state matrix
- Each rowround works as four quarterrounds on each of the four rows of the state matrix
- columnround, rowroundworks one after another
- Salsa20/20: 10 columnround, 10 rowround interleaved

Column Round & Transpose

- In each round, first apply quarterround on all the four columns in the following order:

$$\text{quarterround}(x_0, x_4, x_8, x_{12}),\ \text{quarterround}(x_5, x_9, x_{13}, x_1),$$
$$\text{quarterround}(x_{10}, x_{14}, x_2, x_6),\ \text{quarterround}(x_{15}, x_3, x_7, x_{11}),$$

- And then a transpose(X) as follows:

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \rightarrow X^T = \begin{pmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{pmatrix}.$$

Number of Rounds

- By $X^{(r)}$, we mean that r such rounds have been applied on the initial state X.
- $X^{(0)}$ is the same as the initial state X.
- Finally, after R rounds we have $X^{(R)}$.
- Then a keystream block of 16 words or 512 bits is obtained as

$$Z = X + X^{(R)}.$$

- For Salsa20, $R = 20$.
- The one accepted in eStream software portfolio is Salsa20/12, where $R = 12$.
- Naturally, more rounds will provide better security and less rounds will provide higher speed.
- No concept of different KSA/PRGA

Reversible State Transition

- Each Salsa20 round is reversible as the state-transition operations are reversible.
- If $X^{(r+1)} = \text{round}(X^{(r)})$, then $X^{(r)} = \text{reverseround}(X^{(r+1)})$, where reverseround is the inverse of round and consists of first transposing the state and then applying the inverse of quarterround for each column as follows.

$$\left.\begin{aligned} a &= a \oplus ((d + c) \lll 18), \\ d &= d \oplus ((c + b) \lll 13), \\ c &= c \oplus ((b + a) \lll 9), \\ b &= b \oplus ((a + d) \lll 7). \end{aligned}\right\} \quad (2)$$

Broad view

- Hardware vs Software
- Hardware vs Hardware (towards less gates in implementation)
- Software vs Software (How many bytes/clock?)
- Frequency of modification in Key/IV: relates to cost of Key Scheduling Algorithm, encrypting long/short key stream
- Reversibility (if not collision, if yes TMDTO)
- Proof of security vs statistical analysis of algorithms
- Issues of National/International Standardization
- Authenticated Encryption with Associated Data (AEAD)

Thank You