

ISO/IEC JTC 1/WG 7
Working Group on Sensor Networks

Document Number:	N059
Date:	2010-07-06
Replace:	
Document Type:	Liaison Organization Contribution
Document Title:	Liaison Statement from JTC 1/SC 27/WG 2 to JTC 1/WG 7 on the ISO/IEC 2 nd WD 29192-3
Document Source:	JTC 1/SC 27/WG 2
Document Status:	For consideration at the 2 nd WG 7 meeting in US.
Action ID:	FYI
Due Date:	
No. of Pages:	36

ISO/IEC JTC 1/WG 7 Convenor:

Dr. Yongjin Kim, Modacom Co., Ltd (Email: cap@modacom.co.kr)

ISO/IEC JTC 1/WG 7 Secretariat:

Ms. Jooran Lee, Korean Standards Association (Email: jooran@kisi.or.kr)



REPLACES:

<p style="text-align: center;">ISO/IEC JTC 1/SC 27 Information technology - Security techniques Secretariat: DIN, Germany</p>
--

DOC TYPE: text for working draft

TITLE: Text for ISO/IEC 2nd WD 29192-3 — Information technology — Security techniques — Lightweight cryptography — Part 3: Stream ciphers

SOURCE: Project Editor (Hirotaka Yoshida)

DATE : 2010-07-01

PROJECT: 29192-3 (1.27.82.03)

STATUS: In accordance with resolutions 1 and 6 (contained in SC 27 N8789) of the 40th SC 27/WG 2 meeting held in Melaka (Malaysia) 19th – 23rd April 2010 , this document is being circulated to National Bodies and liaison organizations for STUDY AND COMMENT.

National Bodies and liaison organizations of SC 27 are requested to send their comments / contributions on the hereby attached document directly to the SC 27/WG 2 Secretariat sc27wg2-secretary@ipa.go.jp as soon as possible but no later than **2010-09-05**.

PLEASE NOTE: For comments please use THE SC 27 TEMPLATE separately attached to this document.

ACTION: COM

DUE DATE: 2010-09-05

DISTRIBUTION: P-, O- and L-Members
W. Fumy, SC 27 Chairman
M. De Soete, SC 27 Vice Chair
E. J. Humphreys, K. Naemura, M. Bañón, M.-C. Kang, K. Rannenber, WG-Conveners

MEDIUM: <http://isotc.iso.org/livelink/livelink/open/jtc1sc27>

NO. OF PAGES: 1 + 34

ISO/IEC JTC 1/SC 27 N **8757**

Date: 2010-07-01

ISO/IEC WD 29192-3.2

ISO/IEC JTC 1/SC 27/WG 2

Secretariat: DIN

Information technology — Security techniques — Lightweight cryptography — Part 3: Stream ciphers

Technologies de l'information — Techniques de sécurité — Cryptographie pour environnements contraints — Partie 3: Chiffrements à flot

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International Standard
Document subtype:
Document stage: (20) Preparatory
Document language: E

Copyright notice

This ISO document is a working draft or committee draft and is copyright-protected by ISO. While the reproduction of working drafts or committee drafts in any form for use by participants in the ISO standards development process is permitted without prior permission from ISO, neither this document nor any extract from it may be reproduced, stored or transmitted in any form for any other purpose without prior written permission from ISO.

Requests for permission to reproduce this document for the purpose of selling it should be addressed as shown below or to ISO's member body in the country of the requester:

Secretariat of ISO/IEC JTC 1/SC 27
DIN German Institute for Standardization
DE-10772 Berlin

Tel. + 49 30 2601 2652

Fax + 49 30 2601 4 2652

E-mail krystyna.passia@din.de

Web <http://www.jtc1sc27.din.de/en> (public web site)

<http://isotc.iso.org/livelink/livelink/open/jtc1sc27> (SC 27 documents)

Reproduction for sales purposes may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

Contents

Page

Foreword	v
Introduction.....	vi
1 Scope	1
2 Normative reference	1
3 Terms and definitions	1
4 Symbols and abbreviated terms	3
4.1 Symbols.....	3
4.2 Left-truncation of bits	4
4.3 Shift operation	4
4.4 Variable $l(k)$	4
5 General models for stream ciphers	4
5.1 Synchronous Keystream generators	4
5.2 Output functions.....	5
5.2.1 General model of output function.....	5
5.2.2 Binary-additive output function	5
6 Dedicated keystream generators	6
6.1 <i>Enocoro-128v2</i> keystream generator	6
6.1.1 Introduction to <i>Enocoro-128v2</i>	6
6.1.2 Initialization function <i>Init</i>	7
6.1.3 Next-state function <i>Next</i>	8
6.1.4 Keystream function <i>Strm</i>	8
6.1.5 Function ρ	9
6.1.6 Function λ	9
6.1.7 Function L_{8432}	9
6.1.8 Function S_8	10
6.2 <i>Enocoro-80</i> keystream generator	11
6.2.1 Introduction to <i>Enocoro-80</i>	11
6.2.2 Initialization function <i>Init</i>	11
6.2.3 Next-state function <i>Next</i>	12
6.2.4 Keystream function <i>Strm</i>	12
6.2.5 Function ρ	13
6.2.6 Function λ	13
6.2.7 Function L_{8431}	13
6.3 Trivium keystream generator	14
6.3.1 Overview.....	14
6.3.2 Internal State.....	14
6.3.3 Initialization function <i>Init</i>	15
6.3.4 Next-state function <i>Next</i>	16
6.3.5 Keystream function <i>Strm</i>	16
Annex A (informative) Examples	17
A.1 Example for <i>Enocoro-128v2</i>	17
A.1.1 Key, initialization vector, and keystream triplets	17
A.1.2 Sample internal states	19
A.2 Example for <i>Enocoro-80</i>	20
A.2.1 Key, initialization vector, and keystream triplets	20
A.2.2 Sample internal states	20
A.3 Example for Trivium	21
A.3.1 Key, initialization vector, and keystream triplets	21

A.3.2	Internal Sequence Bits	22
A.3.3	Internal State	24
A.3.4	Parallelism	25
Annex B	(informative) Usage Notes	26
B.1	Usage Note for Trivium	26
B.1.1	Parallelism	26
B.1.2	Recommended Use of Initialization Values	26
Bibliography	28

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29192-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *Security techniques*.

ISO/IEC 29192 consists of the following parts, under the general title *Information technology — Security techniques — Lightweight cryptography*:

- *Part 1: General*
- *Part 2: Block ciphers*
- *Part 3: Stream ciphers*
- *Part 4: Mechanisms using asymmetric techniques*

Introduction

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this Part of ISO/IEC 29192 may involve the use of patents.

The ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of these patent rights has assured the ISO and IEC that they are to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with the ISO and IEC. Information may be obtained from:

ISO/IEC JTC 1 “Patent Database”

Database is publicly available at:

<http://isotc.iso.org/>

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 29192 may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — Security techniques — Lightweight cryptography — Part 3: Stream ciphers

1 Scope

This part of ISO/IEC 29192 specifies stream cipher algorithms. A stream cipher is an encryption mechanism that uses a keystream to encrypt a plaintext in bitwise or block-wise manner. There are two types of stream cipher: a synchronous stream cipher, in which the keystream is only generated from the secret key (and an initialization vector) and a self-synchronizing stream cipher, in which the keystream is generated from the secret key and some past ciphertexts (and an initialization vector). Typically the encryption operation is the additive bitwise XOR operation between a keystream and the message. This standard describes pseudorandom number generators for producing keystream for stream ciphers.

2 Normative reference

The following referenced document is indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- ISO/IEC 18033-4:2005, *Information technology — Security techniques — Encryption algorithms — Part 4: Stream ciphers*.

3 Terms and definitions

For the purposes of this part of ISO/IEC 29192, the terms and definitions given in ISO/IEC 29192-1 and the following apply.

3.1

big-endian

method of storage of multi-byte numbers with the most significant bytes at the lowest memory addresses

[ISO/IEC 10118-1: 2000]

3.2

ciphertext

data which has been transformed to hide its information content

[ISO/IEC 10116: 1997]

3.3

decryption

reversal of a corresponding encipherment

[ISO/IEC 11770-1: 1996]

3.4

encryption

(reversible) transformation of data by a cryptographic algorithm to produce ciphertext, i.e., to hide the information content of the data

[ISO/IEC 9797-1: 1996]

3.5

initialization value

value used in defining the starting point of an encipherment process

[ISO 8372: 1987]

3.6

key

sequence of symbols that controls the operation of a cryptographic transformation (e.g. encipherment, decipherment)

[ISO/IEC 11770-1: 1996]

3.7

keystream function

function that takes as input the current state of the keystream generator and (optionally) part of the previously output ciphertext, and gives as output the next part of the keystream

3.8

keystream generator

state-based process (i.e. a finite state machine) that takes as inputs a key, an initialization vector, and if necessary the ciphertext, and that outputs a keystream, i.e. a sequence of bits or blocks of bits, of arbitrary length

3.9

next-state function

function that takes as input the current state of the keystream generator and (optionally) part of the previously output ciphertext, and gives as output a new state for the keystream generator

3.10

plaintext

unenciphered information

[ISO/IEC 9797-1: 1999]

3.11

padding

appending extra bits to a data string

[ISO/IEC 10118-1: 2000]

3.12

secret key

key used with symmetric cryptographic techniques by a specified set of entities

[ISO/IEC 11770-3: 1999]

3.13

state

current internal state of a keystream generator

4 Symbols and abbreviated terms

4.1 Symbols

$0x$	Prefix for hexadecimal values.
$0^{(n)}$	n -bit variable where 0 is assigned to every bit.
AND	Bitwise logical AND operation.
a_i	Variables in an internal state of a keystream generator.
b_i	Variables in an internal state of a keystream generator.
C_i	Ciphertext block.
$F[x]$	The polynomial ring over the finite field F .
$GF(2^n)$	Finite field of exactly 2^n elements.
<i>Init</i>	Function which generates the initial internal state of a keystream generator.
<i>IV</i>	Initialization vector.
K	Key.
<i>Next</i>	Next-state function of a keystream generator.
n	Block length.
OR	Bitwise logical OR operation.
<i>Out</i>	Output function combining keystream and plaintext in order to generate ciphertext.
P	Plaintext.
P_i	Plaintext block.
R	Additional input to <i>Out</i> .
<i>Strm</i>	Keystream function of a keystream generator.
S_i	Internal state of a keystream generator.
Z	Keystream.
Z_i	Keystream block.
$\lceil x \rceil$	The smallest integer greater than or equal to the real number x .
$\neg x$	Bitwise complement operation.
\bullet	Polynomial multiplication.
\parallel	Bit concatenation.
$+_m$	Integer addition modulo 2^m .

\oplus Bitwise XOR (eXclusive OR) operation.

\otimes Operation of multiplication of elements in the finite field $GF(2^n)$.

EXAMPLE E.g. $C = A \otimes B$: In this operation, the finite field is represented as a selected irreducible polynomial $F(x)$ of degree n with binary coefficients, the n -bit blocks $A = \{a_{n-1}, a_{n-2}, \dots, a_0\}$ and $B = \{b_{n-1}, b_{n-2}, \dots, b_0\}$ (where the a_i and b_i are bits) are represented as the polynomials, $A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_0$ and $B(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \dots + b_0$ respectively, then let $C(x) = A(x) \bullet B(x) \bmod F(x)$, i.e. $C(x)$ is the polynomial of degree at most $n-1$ obtained by multiplying $A(x)$ and $B(x)$, dividing the result by $F(x)$, and then taking the remainder. If $C(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_0$ (where the c_i are bits) then let C be the n -bit block $\{c_{n-1}, c_{n-2}, \dots, c_0\}$.

$<<_n t$ t -bit left shift in an n -bit register.

$>>_n t$ t -bit right shift in an n -bit register.

$<<<_n t$ t -bit left circular rotation in an n -bit register.

$>>>_n t$ t -bit right circular rotation in an n -bit register.

4.2 Left-truncation of bits

The operation of selecting the j leftmost bits of an array $A=(a_0, a_1, \dots, a_{m-1})$ to generate a j -bit array is written

$$(j \sim A) = (a_0, a_1, \dots, a_{j-1})$$

This operation is defined only when $1 \leq j \leq m$ [ISO/IEC 10116].

4.3 Shift operation

A "shift operation" *Shift* is defined as follows: Given an n -bit variable X and a k -bit variable F where $1 \leq k \leq n$, the effect of the shift function *Shift* is to produce the n -bit variable

$$\text{Shift}_k(X \mid F) = (x_k, x_{k+1}, \dots, x_{n-1}, f_0, f_1, \dots, f_{k-1}) \quad (k < n)$$

$$\text{Shift}_k(X \mid F) = (f_0, f_1, \dots, f_{k-1}) \quad (k = n)$$

The effect is to shift the bits of array X left by k places, discarding x_0, x_1, \dots, x_{k-1} and to place the array F in the rightmost k places of X . When $k = n$ the effect is to totally replace X by F [ISO/IEC 10116].

4.4 Variable $I(k)$

The variable $I(k)$ is a k -bit variable where 1 is assigned to every bit [ISO/IEC 10116].

5 General models for stream ciphers

5.1 Synchronous Keystream generators

A synchronous keystream generator is a finite-state machine. It is defined by:

1. An initialization function, *Init*, which takes as input a key K and an initialization vector IV , and outputs an initial state S_0 for the keystream generator. The initialization vector should be chosen so that no two messages are ever encrypted using the same key and the same IV .
2. A next-state function, *Next*, which takes as input the current state of the keystream generator S_i , and outputs the next state of the keystream generator S_{i+1} .

3. A keystream function, $Strm$, which takes as input a state of the keystream generator S_i , and outputs a keystream block Z_i .

When the synchronous keystream generator is first initialized, it will enter an initial state S_0 defined by

$$S_0 = Init(IV, K).$$

On demand the synchronous keystream generator will for $i=0,1,\dots$:

1. Output a keystream block $Z_i = Strm(S_i, K)$.
2. Update the state of the machine $S_{i+1} = Next(S_i, K)$.

Therefore to define a synchronous keystream generator it is only necessary to specify the functions $Init$, $Next$ and $Strm$, including the lengths and alphabets of the key, the initialization vector, the state, and the output block.

5.2 Output functions

5.2.1 General model of output function

This subclause specifies a stream cipher output function, i.e. technique to be used in a stream cipher to combine a keystream with plaintext to derive ciphertext.

An output function for a synchronous or a self-synchronizing stream cipher is an invertible function Out that combines a plaintext block P_i , a keystream block Z_i , and some other input R if necessary to give a ciphertext block C_i ($i \geq 0$). A general model for stream cipher output function is now defined.

Encryption of a plaintext block P_i by a keystream block Z_i is given by:

$$C_i = Out(P_i, Z_i, R),$$

and decryption of a ciphertext block C_i by a keystream block Z_i is given by:

$$P_i = Out^1(C_i, Z_i, R).$$

The output function shall be such that, for any keystream block Z_i , plaintext block P_i , and other input R , we have that

$$P_i = Out^1(Out(P_i, Z_i, R), Z_i, R).$$

5.2.2 Binary-additive output function

A binary-additive stream cipher is a stream cipher in which the keystream, plaintext, and ciphertext blocks are binary digits, and the operation to combine plaintext with keystream is bitwise XOR. Let n to be the bit length of P_i . This function is specified by

$$Out(P_i, Z_i, R) = P_i \oplus Z_i.$$

The operation Out^1 is specified by

$$Out^1(C_i, Z_i, R) = C_i \oplus Z_i.$$

6 Dedicated keystream generators

6.1 *Enocoro-128v2* keystream generator

6.1.1 Introduction to *Enocoro-128v2*

Enocoro-128v2 is a keystream generator which uses a 128-bit secret key K , a 64-bit initialization vector IV , and a state variable S_i ($i \geq 0$) consisting of 34 bytes, and outputs a keystream block Z_i at every iteration of the function *Strm*.

NOTE This keystream generator is originally proposed in [2].

The state variable S_i is sub-divided into a combination of a 2-byte variable:

$$a^{(i)} = (a_0^{(i)}, a_1^{(i)}),$$

where $a_j^{(i)}$ is a byte (for $j = 0, 1$), and a 32-byte variable:

$$b^{(i)} = (b_0^{(i)}, b_1^{(i)}, \dots, b_{31}^{(i)}),$$

where $b_j^{(i)}$ is a byte (for $j = 0, 1, \dots, 31$).

The *Init* function, defined in detail in 6.1.2, takes as input the 128-bit key K and the 64-bit initializing vector IV , and produces the initial value of the state variable $S_0 = (a^{(0)}, b^{(0)})$.

The *Next* function, defined in detail in 6.1.3, takes as input the 34-byte state variable $S_i = (a^{(i)}, b^{(i)})$ and produces as output the next value of the state variable $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.

The *Strm* function, defined in detail in 6.1.4, takes as input the 34-byte state variable $S_i = (a^{(i)}, b^{(i)})$ and produces as output the keystream block Z_i .

Let the inputs from the buffer to the ρ function be $b_{k1}, b_{k2}, b_{k3}, b_{k4}$. The parameters which define the λ function are denoted by $q_1, p_1, q_2, p_2, q_3, p_3$. *Enocoro-128v2* uses the following values for these parameters:

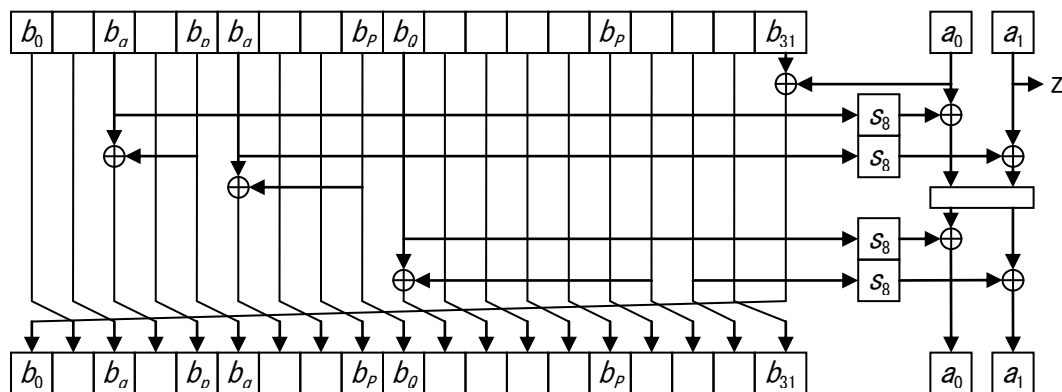
$$k_1 = 2, k_2 = 7, k_3 = 16, k_4 = 29,$$

$$p_1 = 6, p_2 = 15, p_3 = 28,$$

$$q_1 = 2, q_2 = 7, q_3 = 16.$$

Enocoro-128v2 uses operations over the finite field $GF(2^8)$. In the polynomial representation, $GF(2^8)$ is realized as $GF(2)[x] / \phi_{8432}(x)$, where $\phi_{8432}(x)$ is an irreducible polynomial of degree 8 defined over $GF(2)$. The *Enocoro-128v2* keystream generator uses the following irreducible polynomial:

$$\phi_{8432}(x) = x^8 + x^4 + x^3 + x^2 + 1.$$

Figure 1 — Schematic view of *Enocoro-128v2*

6.1.2 Initialization function *Init*

The initialization of *Enocoro-128v2* is divided into six steps. The initialization function *Init* is as follows:

Input: 128-bit key K , 64-bit initialization vector IV .

Output: Initial value of the state variable $S_0 = (a^{(0)}, b^{(0)})$.

a) Set the key K into the part of the state variable $b_j^{(-96)}$ as follows:

- Set $(K_0 || K_1 || \dots || K_{15}) = K$, where K_j is 8 bits for $j=0,1,2,\dots,15$.
- For $j=0,1,2,\dots,15$, set $b_j^{(-96)} = K_j$.

b) Set the initialization vector IV into the part of the state variable $b_j^{(-96)}$ as follows:

- Set $(I_0 || I_1 || \dots || I_7) = IV$, where I_j is 8 bits for $j=0,1,2,\dots,7$.
- For $j=0,1,2,\dots,7$, set $b_{j+16}^{(-96)} = I_j$.

c) Set the constants into the part of the state variable $a_j^{(-96)}$ and $b_j^{(-96)}$ as follows:

- Set $b_{24}^{(-96)} = C_0 = 0x66$,
- Set $b_{25}^{(-96)} = C_1 = 0xe9$,
- Set $b_{26}^{(-96)} = C_2 = 0x4b$,
- Set $b_{27}^{(-96)} = C_3 = 0xd4$,
- Set $b_{28}^{(-96)} = C_4 = 0xef$,
- Set $b_{29}^{(-96)} = C_5 = 0x8a$,

- Set $b_{30}^{(-96)} = C_6 = 0x2c$,
- Set $b_{31}^{(-96)} = C_7 = 0x3b$,
- Set $a_0^{(-96)} = C_8 = 0x88$,
- Set $a_1^{(-96)} = C_9 = 0x4c$.

d) Set a 8-bit counter $ctr = 1$

e) For $i = -96, -95, \dots, -1$, iterate the following steps 96 times:

- $b_{31}^{(i+1)} = b_{31}^{(i)} \oplus ctr$,
- $ctr = 2 \otimes ctr$,
- Set $S_{i+1} = \text{Next}(a, b)$.

f) Output S_0 .

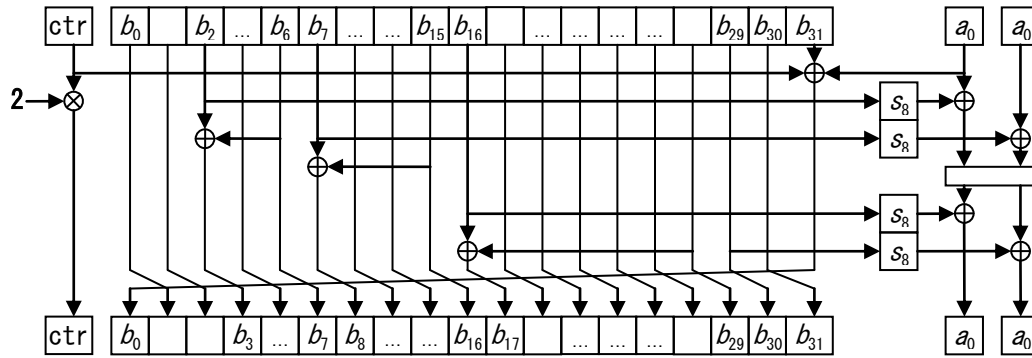


Figure 2 — State update during the initialization of *Enocoro-128v2*

6.1.3 Next-state function *Next*

The next-state function of *Enocoro-128v2* is described as a combination of ρ and λ . The next-state function *Next* of *Enocoro-128v2* is as follows:

Input: State variable $S_i = (a^{(i)}, b^{(i)})$.

Output: Next value of the state variable $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.

- Set $a^{(i+1)} = \rho(a^{(i)}, b^{(i)})$. The detailed description of the function ρ is given in 6.1.5
- Set $b^{(i+1)} = \lambda(b^{(i)}, a_0^{(i)})$. The detailed description of the function λ is given in 6.1.6
- Set $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.
- Output S_{i+1} .

6.1.4 Keystream function *Strm*

The keystream function *Strm* is as follows:

Input: State variable S_i .

Output: Keystream block Z_i .

- Set $Z_i = a_1^{(i)}$.
- Output Z_i .

6.1.5 Function ρ

The function ρ is the composition of XORs, a non-linear transformation using the function S_8 , a linear transformation using the matrix L_{8432} . The function ρ is as follows:

Input: State variable $a^{(i)}$, four 8-bit parameters $b_{k1}^{(i)}$, $b_{k2}^{(i)}$, $b_{k3}^{(i)}$, $b_{k4}^{(i)}$.

Output: The next value of the state variable $a^{(i+1)}$.

- Set $u_0 = a_0^{(i)} \oplus s_8[b_{k1}^{(i)}]$.
- Set $u_1 = a_1^{(i)} \oplus s_8[b_{k2}^{(i)}]$.
- Set $(v_0, v_1) = L_{8432}(u_0, u_1)$,
- Set $a_0^{(i+1)} = v_0 \oplus s_8[b_{k3}^{(i)}]$,
- Set $a_1^{(i+1)} = v_1 \oplus s_8[b_{k4}^{(i)}]$.
- Output $a^{(i+1)}$.

6.1.6 Function λ

The function λ is as follows:

Input: State variable $b^{(i)}$, 8-bit parameter $a_0^{(i)}$.

Output: The next value of the state variable $b^{(i+1)}$.

- Set $b_j^{(i+1)} = b_{j-1}^{(i)}$, for $j \neq 0, q_1 + 1, q_2 + 1, q_3 + 1$,
- Set $b_0^{(i+1)} = b_{31}^{(i)} \oplus a_0^{(i)}$,
- Set $b_{qj+1}^{(i+1)} = b_{qj}^{(i)} \oplus b_{pj}^{(i)}$, for $j = 1, 2, 3$,
- Output $b^{(i+1)}$.

6.1.7 Function L_{8432}

The function L_{8432} is the internal function of the ρ function. Let us denote the input and the output to the L_{8432} function as U and V respectively. The function L_{8432} is as follows:

Input: 16-bit string U .

Output: 16-bit string V .

- Set $(u_0, u_1) = U$, where u_i is an 8-bit string and an element of $GF(2^8)$.
- Set

$$\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = L_{8432}(u_0, u_1) = \begin{pmatrix} 1 & 1 \\ 1 & 0x2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix}.$$

where 0x02 is the hexadecimal expressions of the elements of $GF(2^8)$ which is realized as $GF(2)[x] / \phi_{8432}(x)$

- Set $V = v_0 \parallel v_1$.
- Output V .

6.1.8 Function S_8

Function S_8 uses operations over the finite field $GF(2^4)$. In the polynomial representation, $GF(2^4)$ is realized as $GF(2)[x] / \phi_{41}(x)$, where $\phi_{41}(x)$ is an irreducible polynomial of degree 4 defined over $GF(2)$. The *Enocoro-128v2* keystream generator uses the following irreducible polynomial:

$$\phi_{41}(x) = x^4 + x + 1,$$

The Sbox (substitution box) S_8 defines a permutation which maps 8-bit inputs to 8-bit outputs. It has also SPS structure and it consists of 4 small Sboxes s_4 which map 4-bit inputs to 4-bit outputs and a linear transformation l defined by a 2-by-2 matrix over $GF(2^4)$. The linear transformation l is defined as

$$l(x, y) = \begin{pmatrix} 1 & 0x4 \\ 0x4 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \quad x, y \in GF(2^4)$$

Let us denote the input and the output to the S_8 function as X and Z respectively. The function S_8 is as follows:

Input: 8-bit string X .

Output: 8-bit string Z .

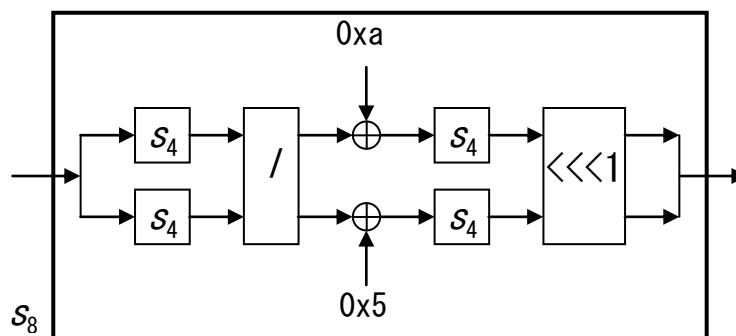
- Set $(x_0, x_1) = X$, where x_i is an 4-bit string and an element of $GF(2^4)$.
- Set

$$\begin{aligned} y_0 &= s_4[s_4[x_0] \oplus 0x4 \bullet s_4[x_1] \oplus 0xa], \\ y_1 &= s_4[0x4 \bullet s_4[x_0] \oplus s_4[x_1] \oplus 0x5], \end{aligned}$$

where 0x4, 0x5, 0xa are the hexadecimal expressions of the elements of $GF(2^4)$ and the Sbox s_4 is defined as

$$s_4[16] = \{1, 3, 9, 10, 5, 14, 7, 2, 13, 0, 12, 15, 4, 8, 6, 11\}.$$

- Set $Z = (y_0 \parallel y_1) \lll_8 1$.
- Output Z .

Figure 3 — Sbox S_8

6.2 Enocoro-80 keystream generator

6.2.1 Introduction to *Enocoro-80*

Enocoro-80 is a keystream generator which uses a 80-bit secret key K , a 64-bit initialization vector IV , and a state variable S_i ($i \geq 0$) consisting of 22 bytes, and outputs a keystream block Z_i at every iteration of the function *Strm*.

NOTE This keystream generator is originally proposed in [3].

The state variable S_i is sub-divided into a combination of a 2-byte variable:

$$a^{(i)} = (a_0^{(i)}, a_1^{(i)}),$$

where $a_j^{(i)}$ is a byte (for $j = 0, 1$), and a 20-byte variable:

$$b^{(i)} = (b_0^{(i)}, b_1^{(i)}, \dots, b_{19}^{(i)}),$$

where $b_j^{(i)}$ is a byte (for $j = 0, 1, \dots, 19$).

The *Init* function, defined in detail in 6.2.2, takes as input the 80-bit key K and the 64-bit initializing vector IV , and produces the initial value of the state variable $S_0 = (a^{(0)}, b^{(0)})$.

The *Next* function, defined in detail in 6.2.3, takes as input the 22-byte state variable $S_i = (a^{(i)}, b^{(i)})$ and produces as output the next value of the state variable $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.

The *Strm* function, defined in detail in 6.3.5, takes as input the 22-byte state variable $S_i = (a^{(i)}, b^{(i)})$ and produces as output the keystream block Z_i .

Let the inputs from the buffer to the ρ function be $b_{k1}, b_{k2}, b_{k3}, b_{k4}$. The parameters which define the λ function are denoted by $q_1, p_1, q_2, p_2, q_3, p_3$. *Enocoro-80* uses the following values for these parameters:

$$k_1 = 1, k_2 = 4, k_3 = 6, k_4 = 16,$$

$$p_1 = 3, p_2 = 5, p_3 = 15,$$

$$q_1 = 1, q_2 = 4, q_3 = 6.$$

6.2.2 Initialization function *Init*

The initialization of *Enocoro-80* is divided into five steps. The initialization function *Init* is as follows:

Input: 80-bit key K , 64-bit initialization vector IV .

Output: Initial value of the state variable $S_0 = (a^{(0)}, b^{(0)})$.

a) Set the key K into the part of the state variable $b_j^{(-40)}$ as follows:

- Set $(K_0 || K_1 || \dots || K_9) = K$, where K_j is 8 bits for $j=0,1,2,\dots,9$.
- For $j=0,1,2,\dots,9$, set $b_j^{(-40)} = K_j$.

b) Set the initialization vector IV into the part of the state variable $b_j^{(-40)}$ as follows:

- Set $(I_0 || I_1 || \dots || I_7) = IV$, where I_j is 8 bits for $j=0,1,2,\dots,7$.
- For $j=0,1,2,\dots,7$, set $b_{j+10}^{(-96)} = I_j$.

c) Set the constants into the part of the state variable $a_j^{(-40)}$ and $b_j^{(-40)}$ as follows:

- Set $b_{18}^{(-40)} = C_0 = 0x66$,
- Set $b_{19}^{(-40)} = C_1 = 0xe9$,
- Set $a_0^{(-40)} = C_2 = 0x4b$,
- Set $a_1^{(-40)} = C_3 = 0xd4$.

d) For $i=-40,-39,\dots,-1$, iterate the following steps 40 times:

- Set $S_{i+1} = \text{Next}(a, b)$.

e) Output S_0

6.2.3 Next-state function *Next*

The next-state function of *Enocoro-80* is described as a combination of ρ and λ . The next-state function *Next* of *Enocoro-80* is as follows:

Input: State variable $S_i = (a^{(i)}, b^{(i)})$.

Output: Next value of the state variable $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.

- Set $a^{(i+1)} = \rho(a^{(i)}, b^{(i)})$. The detailed description of the function ρ is given in 6.2.5
- Set $b^{(i+1)} = \lambda(b^{(i)}, a_0^{(i)})$. The detailed description of the function λ is given in 6.2.6
- Set $S_{i+1} = (a^{(i+1)}, b^{(i+1)})$.
- Output S_{i+1} .

6.2.4 Keystream function *Strm*

The keystream function *Strm* is as follows:

Input: State variable S_i .

Output: Keystream block Z_i .

- Set $Z_i = a_1^{(i)}$.

Output Z_i .

6.2.5 Function ρ

The function ρ is the composition of XORs, a non-linear transformation using the function S_8 , a linear transformation using the matrix L_{8431} . The function S_8 is described in 6.1.8. The function ρ is as follows:

Input: State variable $a^{(i)}$, four 8-bit parameters $b_{k1}^{(i)}$, $b_{k2}^{(i)}$, $b_{k3}^{(i)}$, $b_{k4}^{(i)}$.

Output: The next value of the state variable $a^{(i+1)}$.

- Set $u_0 = a_0^{(i)} \oplus s_8[b_{k1}^{(i)}]$.
- Set $u_1 = a_1^{(i)} \oplus s_8[b_{k2}^{(i)}]$.
- Set $(v_0, v_1) = L_{8431}(u_0, u_1)$,
- Set $a_0^{(i+1)} = v_0 \oplus s_8[b_{k3}^{(i)}]$,
- Set $a_1^{(i+1)} = v_1 \oplus s_8[b_{k4}^{(i)}]$.
- Output $a^{(i+1)}$.

6.2.6 Function λ

The function λ is as follows:

Input: State variable $b^{(i)}$, 8-bit parameter $a_0^{(i)}$.

Output: The next value of the state variable $b^{(i+1)}$.

- Set $b_j^{(i+1)} = b_{j-1}^{(i)}$, for $j \neq 0, q_1 + 1, q_2 + 1, q_3 + 1$,
- Set $b_0^{(i+1)} = b_{19}^{(i)} \oplus a_0^{(i)}$,
- Set $b_{qj+1}^{(i+1)} = b_{qj}^{(i)} \oplus b_{pj}^{(i)}$, for $j = 1, 2, 3$,
- Output $b^{(i+1)}$.

6.2.7 Function L_{8431}

Function L_{8431} uses operations over the finite field $GF(2^8)$. In the polynomial representation, $GF(2^8)$ is realized as $GF(2)[x] / \phi_{8431}(x)$, where $\phi_{8431}(x)$ is an irreducible polynomial of degree 8 defined over $GF(2)$. The *Enocoro-128v2* keystream generator uses the following irreducible polynomial:

$$\phi_{8431}(x) = x^8 + x^4 + x^3 + x + 1.$$

The function L_{8431} is the internal function of the ρ function. Let us denote the input and the output to the L_{8431} function as U and V respectively. The function L_{8431} is as follows:

Input: 16-bit string U .

Output: 16-bit string V .

- Set $(u_0, u_1) = U$, where u_i is an 8-bit string and an element of $GF(2^8)$.
- Set

$$\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = L_{8431}(u_0, u_1) = \begin{pmatrix} 1 & 1 \\ 1 & 0x2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix}.$$

where 0x02 is the hexadecimal expressions of the elements of $GF(2^8)$ which is realized as $GF(2)[x]/\phi_{8431}(x)$.

- Set $V = v_0 \parallel v_1$.
- Output V .

6.3 Trivium keystream generator

6.3.1 Overview

TRIVIUM is a keystream generator which takes as input an 80-bit secret key $K = (K_0, \dots, K_{79})$, an 80-bit initialization value $IV = (IV_0, \dots, IV_{79})$, and generates up to 2^{64} bits of keystream z_0, z_1, \dots, z_{N-1} .

The keystream bits z_i are computed by combining the elements of three internal bit sequences $\{a_i\}$, $\{b_i\}$, and $\{c_i\}$, which themselves are generated by iterating three interconnected nonlinear recurrence relations. The exact relations are specified in 6.3.4.

The first 288 sequence bits involved in the recursion are initialized using the secret key, the initialization value, and some constant bits. The next 1152 triplets (a_i, b_i, c_i) , starting from index $i = -1152$, are computed recursively, but without producing any output. These 1152 initial iterations are referred to as blank rounds.

Each subsequent iteration, starting from $i = 0$, outputs one keystream bit z_i , which is computed by XORing together a subset of six sequence bits. This is repeated until all requested keystream bits have been generated.

In the following sections, the complete keystream generation algorithm is described more formally using the framework introduced in 5. The internal state S_i is defined in 6.3.2, and the functions *Init*, *Next*, and *Strm* are specified in 6.3.3, 6.3.4, and 6.3.5.

6.3.2 Internal State

Since each new triplet (a_i, b_i, c_i) only depends on a limited number of earlier sequence bits, there is no need to keep the entire sequences in memory. At any point in time i , it suffices for the algorithm to maintain an internal state S_i consisting of the following 288 sequence bits:

$$S_i = (a_{i-1}, \dots, a_{i-93}, b_{i-1}, \dots, b_{i-84}, c_{i-1}, \dots, c_{i-111}).$$

NOTE In a straightforward hardware implementation of TRIVIUM, this internal state would be stored in shift registers, as sketched in Figure 4. The bits in the registers (represented by boxes in the figure) are shifted in clockwise direction after each iteration.

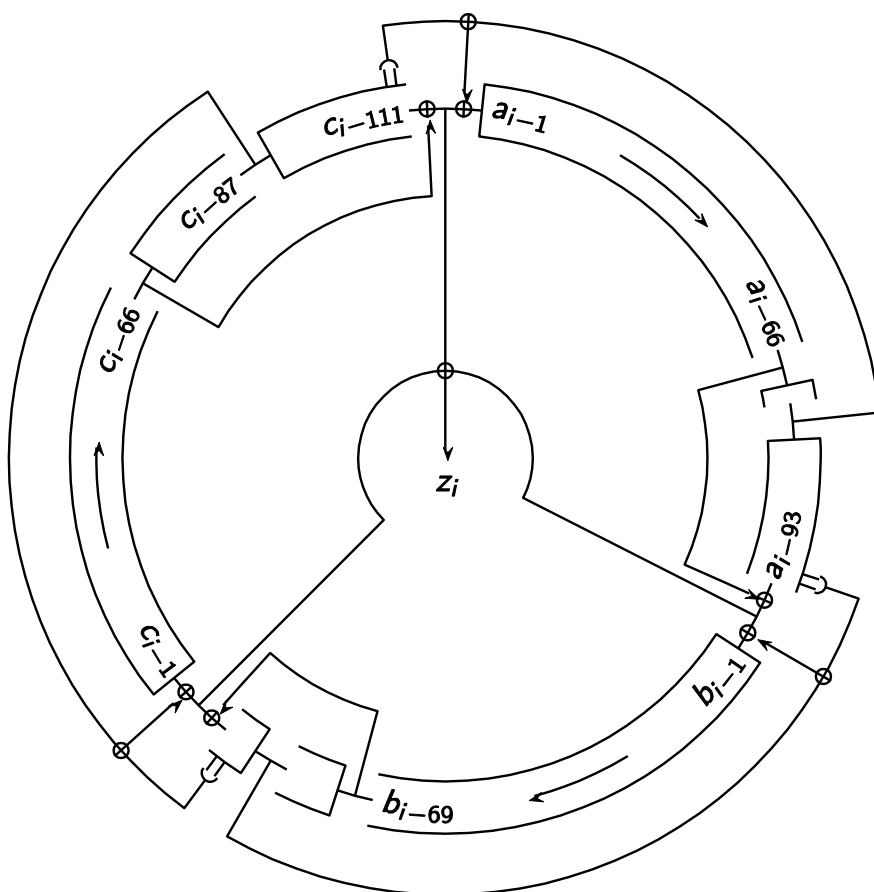


Figure 4 — An implementation of TRIVIUM using shift registers

6.3.3 Initialization function *Init*

The internal state of TRIVIUM is initialized using the following *Init* function.

Input: 80-bit key K , 80-bit initialization value IV .

Output: Initial value of the internal state $S_0 = (a_{-1}, \dots, a_{-93}, b_{-1}, \dots, b_{-84}, c_{-1}, \dots, c_{-111})$.

a) Set $i = -1152$, and initialize the 288 bits of S_i as follows:

- Set $(a_{i-93}, \dots, a_{i-1}) = (0, \dots, 0, K_0, \dots, K_{79})$.
- Set $(b_{i-84}, \dots, b_{i-1}) = (0, \dots, 0, IV_0, \dots, IV_{79})$.
- Set $(c_{i-111}, \dots, c_{i-1}) = (1, 1, 1, 0, \dots, 0)$.

b) For $i = -1151, -1150, \dots, -1, 0$:

- Set $S_i = \text{Next}(S_{i-1})$.

c) Output S_0 .

6.3.4 Next-state function *Next*

The next-state function *Next* is defined below.

Input: Internal state $S_i = (a_i - 1, \dots, a_i - 93, b_i - 1, \dots, b_i - 84, c_i - 1, \dots, c_i - 111)$.

Output: Next value of the internal state $S_{i+1} = (a_i, \dots, a_i - 92, b_i, \dots, b_i - 83, c_i, \dots, c_i - 110)$.

a) Compute the bits a_i , b_i , and c_i :

$$\text{— Set } a_i = c_i - 66 \oplus c_i - 111 \oplus c_i - 110 \cdot c_i - 109 \oplus a_i - 69.$$

$$\text{— Set } b_i = a_i - 66 \oplus a_i - 93 \oplus a_i - 92 \cdot a_i - 91 \oplus b_i - 78.$$

$$\text{— Set } c_i = b_i - 69 \oplus b_i - 84 \oplus b_i - 83 \cdot b_i - 82 \oplus c_i - 87.$$

b) Output $S_{i+1} = (a_i, \dots, a_i - 92, b_i, \dots, b_i - 83, c_i, \dots, c_i - 110)$.

6.3.5 Keystream function *Strm*

The output function *Strm* is defined as follows.

Input: Internal state $S_i = (a_i - 1, \dots, a_i - 93, b_i - 1, \dots, b_i - 84, c_i - 1, \dots, c_i - 111)$.

Output: Keystream bit z_i .

$$\text{Output } z_i = c_i - 66 \oplus c_i - 111 \oplus a_i - 66 \oplus a_i - 93 \oplus b_i - 69 \oplus b_i - 84.$$

Annex A (informative) Examples

A.1 Example for *Enocoro-128v2*

A.1.1 Key, initialization vector, and keystream triplets

IV = 00 00 00 00 00 00 00 00

key = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Key stream =

```

63 d7 da 6b 55 73 7f cf 57 34 b6 77 3a e7 72 e8 e6 5c b3 bd a0 75 e6 b6 94 1c e3 e5 ca 28 2a 1e
54 97 d7 af 12 a2 f0 4e b3 19 d1 fe ce 75 58 0a df d2 f8 f3 bc ee 9e c5 9d c4 1e c3 f6 0e cf 0b
1e cb 1b 64 75 2d 2a c6 20 86 e3 26 c4 5e a5 90 a8 a3 53 2c e3 5d 61 18 45 e7 f4 70 fc 5c 28 19
f4 ff a3 53 c4 b9 8e 1c a4 eb f5 e9 9d 36 f9 83 0a 99 39 8d 04 7f f4 72 aa 07 01 db 50 4e ba 19
4a 60 6a 70 7f 78 eb f7 47 b1 4b 1f 96 d9 f4 3c c4 61 d4 51 fc e1 2b 3d a9 24 ab df f4 52 56 1c
d0 55 c8 42 08 3c 47 38 e2 c6 66 85 b4 07 0c c5 62 c1 4b 4b 48 b3 91 79 f5 41 43 cc ab 28 76 67
58 ad 0b 77 ec 6d 80 6a 8e 1f 8d be ca 64 7e f1 90 c5 06 2d 82 c7 22 59 fa f0 76 5f 7a be 88 43
14 df 9a 2c 03 5a 7f 04 3a d8 55 20 96 e7 2b 3a b0 ad f8 b3 a0 d6 91 67 77 45 5b 85 82 f5 66 11
dc 3d 1b bf 21 59 41 47 48 96 84 b7 2c c0 c3 f1 26 4d f4 b2 ff b6 30 92 aa d4 1c 0b 74 b2 70 e1
44 ac f1 51 51 52 8f 5d 71 8b 49 2a 75 d7 68 16 3d 93 29 50 01 9a 83 44 a2 6f 09 74 ed fd 9a 64
a2 b6 8d 54 3d 08 07 d0 ab 12 78 e3 6e e4 00 3b 6c 1e b4 b6 8b ae 3e 3e 2f e9 94 95 d3 af a6 96
16 18 d6 5a a9 e5 75 a8 77 fd 53 2b f8 85 d6 25 37 47 66 bf 00 46 94 06 d3 7e cf 57 cb 9c eb a6
f9 3a 31 42 a9 7f 1f 49 40 c8 1f be 64 8f 54 1a fd a3 fb c1 4c a6 58 1f ae d2 33 df e9 9d 0a 83
21 0d 67 17 0e 24 bb e5 f6 83 a0 16 94 dc 89 34 8d ea f5 52 be d0 40 49 74 7b d4 a4 db 80 42 80
28 e4 35 53 f1 78 0f 2b aa 9b 6b 22 65 4c 35 01 bb 07 7b 74 90 19 88 28 6b bf 92 46 11 9d e1 e4
c1 16 36 c6 48 10 d7 63 d4 ab eb 3b 65 d4 96 be 1d 13 b9 04 7d d4 45 60 c3 86 d9 4a 29 4c 14 75
bd e6 3a cd 6d 5f 70 f4 c7 28 6a 9e 9e 47 e8 54 36 b5 8d 5b 36 a0 a8 bb c9 b3 b6 c1 8e b6 34 ac
4c d3 6c 82 36 4f 13 eb 61 cb b4 18 8d b6 cc 8c 35 cc be a0 be 81 ae c5 c4 a1 ef 5a e7 a3 c1 99
5d a1 16 e2 24 df 1f fa 84 54 60 36 8f b8 70 96 84 1d 9e af 81 02 8f ae 32 5e a5 6c 9c 92 11 ef
f5 7d f8 51 6f af 0c 3a 27 78 b9 3e 24 3f cc a5 ff d2 7b a8 43 90 5f 3d 2c d0 81 ba f4 6b d9 8b
73 b3 9b 16 59 65 eb 66 f6 e1 8f 73 55 c4 af f5 f9 4d 47 75 a3 63 de 54 ee bf 24 b4 0a 34 b4 d0
64 72 3c 24 09 dd cf 7c 02 ac 99 89 7f 1d 3c 09 61 83 22 c3 28 70 76 97 3b a2 44 37 09 41 8c da
2d ad d0 f8 f9 96 eb 47 63 13 2b c9 7b 94 25 a6 e0 97 55 72 c5 eb 5d 35 b3 91 5a 86 54 4d dd 7c
9c 62 9d 2e e3 cb 05 3b 29 cb 3f d2 17 49 33 42 ed 9b e8 09 42 cc 9f 1e 90 3f 7e 29 8a e1 50 38
6f 8a af 46 76 62 63 e3 9f 91 ec b8 e1 3e cb 1c 31 de 72 3f da e1 2d a5 53 52 4a 4a 1d 7e 91 d0
c5 e5 9f c3 36 44 e1 9d ff 98 99 56 3b 2f f5 2f cb 01 ee 6b 65 89 b7 0f a8 18 ea a6 ac 55 6f f6
8c 2c 6c 96 bc b0 39 75 15 0e c1 d9 45 01 f4 62 1b 07 37 fc 1a 52 7f 93 33 20 3f db 06 51 d7 4d
29 ff aa ae 38 bd 7a 30 74 d8 59 4c 6e 7c 8f 5a 5b 0a 0f 8d c4 bf 4f 40 9b c4 9b 56 0f 6c e2 bf
69 0b f8 97 58 c1 d0 b5 ff 18 7a b4 51 b0 8c 2f 9f c4 90 8c d8 59 fc a8 a6 2f 18 92 e4 e3 89 3a
5c f3 48 e5 10 a4 59 53 5e 0e af 42 6f 05 6c be 24 b2 4b 42 50 0d 02 1b 08 6b 93 a4 41 19 7a a5
9f be 0d 6e b5 1e 22 ca f9 66 2c c8 72 75 60 c2 32 89 8c 58 de a4 3f 92 39 d7 a8 27 74 87 b6 77
b3 c8 a6 39 0b cd bd 90 ba d6 bc 18 2e a4 77 db 41 69 f0 be 17 e5 a0 c0 77 38 46 87 bb ae 3d b6

```

IV = 00 10 20 30 40 50 60 70

Key = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

Key stream =

```

c8 c8 ee 43 3b 0d c0 40 e5 3b c5 06 ea 21 ad 82 20 05 88 89 b7 c8 45 b8 fb bc fc 26 66 d6 5a ce
f5 37 59 b9 7c fb 57 d6 e3 f9 aa a2 68 fe 25 2e e8 bc 58 c3 b9 71 dd 7c bc 8d d2 76 fd 6c 5c 2f
80 67 ec f2 35 da 09 58 51 c6 51 4d 5d 9e 83 b7 01 35 e7 8c eb aa f1 d8 40 1d e4 7a af 13 87 69
20 d9 e6 f4 fe 99 32 af 7b 0b e0 74 c2 23 7f 19 bd 0c 45 8f 27 5b b1 c3 ca 31 b1 7c c2 43 89 e5
48 b1 40 b6 91 1a 40 95 17 39 37 c9 2a 92 5f 2f 0b 24 68 2c e0 3f 07 d2 1c bc 57 a4 2d 20 65 fc
16 22 02 3d 60 d3 3c a8 e2 d7 9d 36 cd 96 85 c4 72 f6 23 ff d3 65 ab b0 ed 0c b7 6f 5f d9 40 7b
2a bf 67 3f f3 e8 10 f6 ce 2d f2 8a 71 1c f8 88 75 18 df 50 ee 7b 6e b2 61 f4 00 8b 81 57 9e 39
4f e2 82 8a c9 02 06 1e 53 d5 fd 2d e6 11 fd a4 68 82 9d 76 25 f1 ac 47 e3 b3 13 58 cd 5a 19 80

```

```

a3 8c 2d 0d d3 17 b1 61 ae 49 0c c9 14 cf 9a dd 04 39 e6 c5 b0 b1 11 e0 86 37 19 06 3c 69 89 79
7d fa 47 18 f9 b2 b6 75 8e 5b 67 7b d1 cd c5 64 6b a3 b8 a8 7d a6 48 44 63 8b b7 16 ef 90 65 4f
48 9d 13 33 16 67 c0 09 58 37 8c 82 0a 40 75 58 66 e4 f4 ff a9 bc 7d cb 3f 39 52 4d 63 92 b1 e7
u4 60 04 12 b7 fa 56 24 c6 ad a6 41 ea 5b 12 44 c8 91 9e 3d 30 05 48 45 39 42 4f 0c c9 24 c3 1b
bd da 6e 4e 2a ee 2c 1a e1 09 d9 04 be 0d 1e c0 15 0b a5 67 95 9c b3 44 3c e8 43 00 e1 84 c5 db
4c 72 20 7c 20 5e f5 0a 2f 43 40 f7 8d 06 72 2a 7e 93 ac 91 7a 05 da 05 40 ab 0b b3 93 d9 bf c0
31 26 0d 6d 16 37 8b 6d 39 87 94 7b 99 f7 d1 21 81 93 37 c2 70 ce 16 89 11 28 8c d3 f0 bf 9e 1b
c3 f1 de 42 16 f2 d6 1d f5 15 a8 86 82 1a 7b 5b b2 ea 09 97 7a 32 eb 76 bd 5c 16 32 4a 5a 5b c3
5d 7e de 8c 94 1c da bf 8c 25 63 fd 17 90 68 3d e7 c5 c6 eb a6 4b 02 a6 ad c0 65 93 54 76 b0 d9
aa ba 3c 32 3c 3b 35 fb b3 03 af ff 65 a7 2a d9 f1 1b 4b ee 5d 46 6a a0 cf 3a 63 d6 d3 1e 59 d9
b4 bb 14 da 94 f6 15 f8 87 d6 45 ed c9 ca 73 e4 3f 3e 17 a8 82 13 e9 cf 5b 9b 6c f0 11 55 3f 4d
4a 39 f7 cf c1 fd 88 ec f5 eb 2e d7 34 03 16 06 13 14 3c 5b 60 c6 51 80 25 0c fa 20 81 bd 9d 47
75 c4 fb fe a4 76 df 41 db 48 92 85 9d 2c 31 73 28 25 d0 46 60 42 94 72 0a 0f 70 df c9 b6 cd 01
5b 51 3e 37 56 71 a6 72 ea 65 35 1e f4 a5 e3 b4 4e 92 8a d0 1f 2b a9 5c 8e 85 bf 5a 5d a6 7b 68
29 c4 d4 4d 29 ea 2b 88 bc 76 e3 da 78 17 6d 76 3b 67 fb 4a 0b 24 a9 e2 15 df 95 b7 b5 61 45 eb
2e 93 33 67 1e 6b cc 3d 53 bd d7 fa 83 23 a0 67 32 ad 06 24 38 7f 12 1c cb 83 a9 dd 3e 51 e8 8b
05 6f c8 b6 09 0b 18 ab c3 dd 23 8e 37 41 87 43 ce 70 6a 7b 59 73 0a 6e 94 92 04 1c e0 ac d8 68
3c 7f 20 44 a8 c3 02 8a ca 3f 5b 35 9d a8 fd 56 0d 61 84 d9 b5 ff ea b5 08 0e 9a f5 58 63 d3 83
e6 ec 81 19 04 7c 75 ee 34 70 21 01 e0 26 96 a5 1b 13 a8 89 9c bd 0c f8 6b 2d c6 f9 0c 0b 56 a0
be 6c 84 06 06 00 4b a8 3e 39 94 72 72 52 74 56 21 f5 8f d0 41 ba dd 6c 0e b0 8e 60 bf 11 2f 16
85 2d 63 0b c9 1f 8b 9e 1c 62 a8 c7 27 93 28 86 00 5c ce ee 06 e5 a6 00 69 86 26 1b e7 e9 13 f6
01 66 a6 5c 44 50 d3 39 6b c6 52 b1 37 0a 82 7b d5 5d 09 da f2 32 81 96 d3 28 c3 db c9 56 41 d9
f4 f2 e3 ba 2a 6d 89 64 9f 39 01 91 de 43 42 36 9f b4 00 e0 cc 8c 69 d5 cc df 74 3e 33 44 ab 75
00 df 7a b7 12 41 08 34 be a7 fa 65 f9 5b ca 9f 7c f6 5d 23 93 6c 30 e2 ee 64 89 a5 74 02 4f 50

```

IV = 80 90 a0 b0 c0 d0 e0 f0

Key = 0f 0e 0d 0c 0b 0a 09 08 07 06 05 04 03 02 01 00

Key stream =

```

f7 73 f9 b4 3f 1c b2 3c e4 19 8f 11 28 89 64 a3 e1 20 2e 6d ea 7d c8 07 7b 5d b1 5e cb 67 c8 6e
49 19 27 80 10 65 1a aa 1a 67 36 2e a8 da 89 72 de 17 cb 8d 37 14 67 93 c8 8b 2f dc 3d 26 d3 06
59 51 bb eb 64 d2 ef 44 09 98 e4 e9 60 bd ef 0e 7a 41 9e c5 68 4c bc 26 6a c1 64 aa 7e 94 18 b9
01 bf cc 79 9d 64 db 87 dd a2 3f a1 71 6e 67 2f c7 b1 c1 7b 6d 22 15 1c 38 ee dd ca 17 b4 8d 56
bf b1 61 89 fe 7e 5a f4 62 72 98 08 63 49 f8 8f 81 51 13 cd a3 65 e0 18 b2 38 b8 ef 72 cd f7 18
15 26 d1 1a 53 11 d2 9c 1d 5e 11 24 ca 11 23 99 29 10 97 0d 1e e2 01 13 fa b6 be 17 0f 59 ba ce
cf 75 d4 f6 65 b0 db 54 62 ab be 9a 15 3e a5 00 b7 7e b1 fc 19 b1 3f c6 15 16 5c a4 6c 3d c2 e6
79 9e ce 56 d3 77 aa 1a da 05 d5 1b 9d a2 89 79 3a a1 db 92 df 3d a5 ef c5 5f b1 b2 91 0e 5e e1
81 a7 51 37 b0 3a dc 8b a3 20 19 78 54 68 9c 09 99 32 1c 4e fb 85 ce a0 82 a0 d9 f3 ab d1 4e ae
0a da 33 db 5b d5 7b cf b3 3b 21 57 fd e3 5e 1b 6c 40 31 39 54 7d 9a 76 04 32 06 1d 1e b2 6a 41
53 75 ea 5e c0 39 e1 76 da 88 2a 38 98 d9 31 ea 48 cd a0 56 57 0a 6a f5 71 1f 27 07 d3 59 5e 86
af 81 73 22 64 1c 96 db 85 d7 d1 5d cd f6 af f4 b1 5b d6 77 34 04 ee 2c a0 fe f1 ac 85 29 cb 33
aa 00 97 b0 d8 13 5f bf d4 6e 7f ea 23 f3 f9 15 58 f1 bc 84 ce 53 32 48 52 b2 23 3b 41 f5 b4 eb
2f 8f 61 c0 af 33 fe d7 0d 70 23 87 7c 06 30 c8 63 c4 a7 f3 23 c9 7a e8 29 cb 30 67 e0 f8 b5 1d
06 b7 78 87 27 83 02 eb 40 b0 04 48 44 19 86 6c 3f ba 07 e4 ff ae 3e 17 3c 5e e6 6b 18 a5 57 64
84 fb 9d 85 c2 8b 02 07 1b df a7 8c f2 2c 2f d6 a1 03 51 6c fc b2 02 d5 85 2c 3d 73 02 d6 23 86
6c 8b 0d fd e0 84 25 43 62 50 ef e7 29 fc b9 47 69 36 f0 33 00 bb f9 b8 b4 98 c2 e4 ab f6 fe 76
e3 ab 63 2c 1a 2c 3e 48 46 2a 40 41 5c 6e ad 3a 44 f1 8b 9d b8 ff 6f 58 6c d1 a5 2e 5a 6f 64 ab
5a d0 bf f3 9a b5 54 e0 06 2d 63 3d 0f 63 fe a4 7e d8 7d e1 23 cd 68 8c f3 3f 27 3b 34 80 ec 89
9a a2 d5 c4 21 4f 2b 00 e5 66 62 71 cb 5f 89 f4 9d 1c cf 9d 95 23 4c 36 38 42 38 69 ad bc b8 eb
f8 f4 08 6d f2 ff 5e 76 1a 48 b9 93 3a 1d dc 16 23 63 32 67 e3 f9 b4 5d ff b3 bc 0b 5b 28 59 1b
54 85 b1 16 93 36 bf be 76 d1 c9 14 68 3d 04 46 9f ad d5 da 26 84 69 b2 f5 1f 99 af 83 67 01 bf
1f ba 28 e5 2f bc 51 6c 1e 2a a3 02 c5 73 1d 41 a8 48 f8 9d f6 67 63 e6 27 3c a7 d9 a8 b0 9a e1
e5 b2 ff 5e 56 6d cd 8f b5 85 34 c3 59 e1 ed ef 7c e2 2f e6 87 1f e7 c9 2e ad 9c 51 2b 22 b1 b8
d2 02 cf 0e e7 ce 8c 30 03 ca 84 9b 16 dd 4d 65 0a ec 17 fc 8e aa 87 c3 01 a2 eb 1d 69 2e d5 57
d5 6e 33 3b 7a 8b 2e 1f 32 1e 22 e9 d7 69 f2 1d f2 4e d5 74 6d 50 92 d5 1c 8f 9a 86 f6 93 9f 81
f5 16 e5 61 a9 a1 55 ed 22 ab c0 30 95 c1 3a 87 2d d6 e4 5a 88 27 34 b6 ee 83 f4 ef 41 ed 7a 3d
9c 2e 1f 82 3b 28 23 65 e1 1a 48 94 e1 cd de 03 6d db 88 1a b0 04 ee 55 86 77 ca 4f 5e 16 f6 09
b3 bc d1 31 5a 5f 2f a6 d3 9b 19 47 f3 07 a6 f9 b2 e5 e4 7a b1 66 95 0d f2 66 85 2f 22 f5 ce 03
25 69 e0 d7 f4 85 c6 f9 f6 2e b9 0d 2a 64 da fe f6 2e 65 01 56 7b 61 5e ab 93 d0 fe 2e b7 c2 83
64 c4 84 b3 8f 93 38 4c 45 41 a8 b8 be 63 32 6c af 85 45 da f6 1d 92 19 a7 ae 00 d9 e1 aa 2e 83
ff 32 54 01 1a 1c 1a bb b9 b2 f5 2f 10 bf 2e ab 3f cc a9 17 c1 26 5b 23 ad 66 e1 94 49 42 a2 f6

```

A.1.2 Sample internal states

IV = 10 00 10 00 10 00 10 00

Key = 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00

Key stream =

```
6c 1b 26 05 d1 97 f7 9f d4 60 4d 13 13 93 89 2e 29 6d 5d 50 f7 e6 07 10 ac 62 56 01 b3 e6 5e a6
ac db 78 0d a9 fb 39 1a 65 10 52 86 23 28 7c 82 51 ea 54 1f 4f e3 0c 1e 94 46 dd f1 57 c3 aa 0e
a4 0f 82 f4 ef 18 47 c9 a1 0a c6 21 ba 32 7a 03 98 3d 35 74 b9 85 52 45 14 5b d4 99 aa 36 50 56
ee ac 08 d5 22 b0 1c 98 ec 21 c3 af d3 9c ba 62 e1 ea b9 76 d7 e0 02 f0 e3 7a d4 f0 f3 ab 9c d6
d9 0c 67 86 4d 3b 40 00 aa 8a e7 79 bf 2a 46 f3 7d bb 09 3d 06 61 4e 24 70 3c e7 d6 d9 8a f1 16
8d 11 8f 90 84 2a b1 43 3f 10 85 2a 39 14 ea 00 75 ae a7 bc 1a 9f 3b 5b c4 90 bf 2b 1f 34 b9 30
07 50 31 fa 3c 68 6d 8e a9 bd d7 d7 3b 2a 7a e5 26 08 41 68 67 5e ce 14 de 8c 75 cb be 5a 7a 05
67 07 eb 0d a5 f1 c1 85 52 e3 e4 9f 3a a8 b8 cd ef 3c c0 45 6f 43 69 0a a4 3a 5b 29 3f 9a 46 b3
8b fc 00 4d de 45 84 80 b4 7a e5 10 d3 21 20 56 3c f5 6b b0 1c cd d7 17 6e 10 0b 5a 0a 02 b8 e2
fe 7e 13 d2 52 ca ea ea b6 5a 38 6f 70 20 33 17 68 c7 37 79 17 33 23 36 09 01 97 d0 84 ad 30 54
a0 fa b1 7f dd 9c 7e ac fc 88 1e 9e b7 73 ed 93 85 02 f3 c2 45 06 d1 be 2a 2d b1 56 8e 48 cf 1a
52 0f 29 56 7a 5d 59 d2 42 81 e2 c4 e0 45 eb 6c 60 62 07 40 d9 03 55 b9 29 35 7b d5 4b 17 ad 92
fe 6c 9d de 2b b9 d9 1e ba 00 7e 6a e5 c6 46 f0 12 cd ed 28 ee cf b2 3f e8 10 e6 0f 4d 84 60 2c
44 76 a2 f8 57 d1 3d 9d ba c2 90 89 a0 2b 13 f0 87 38 e7 cd 58 80 ff 02 2a ba d2 4f 18 ef 87 e0
da 00 5f 93 d4 68 07 da 3d e9 0c 4b c5 46 9a 8e e4 ac 36 b7 ca b2 79 99 1e 5c 3d ce 9d 0e bb 38
3f e1 01 d5 6c fc ee 99 4d f2 6c 01 73 c5 ba c4 30 8d e3 0f b9 c4 9f 28 44 80 f3 f9 41 47 37 48
3e 95 68 46 69 7f 6e f4 54 8d 4c 7b f6 fc a3 ff 61 c5 3a 26 11 91 ea b5 05 db 44 7b 72 b3 54 80
f0 c8 0f a7 d9 ba d1 74 e8 4a c2 c7 ff b1 f6 a2 56 23 cd 10 d7 04 2d e0 df 32 57 c2 3f 02 37 3f
13 42 71 64 3f d4 34 ab 75 76 ee 94 3b 79 8b c4 07 14 80 2f c2 60 1e 54 8b 22 9a 9d bb e5 2d dd
10 01 cf 40 fd 33 18 58 13 ca ab 3b d0 4a f1 61 5b 23 e9 d6 56 13 a7 a1 ce 2e 81 8b 2e 7e f3 38
03 bb 95 e0 ea d2 8f 25 ae 69 0d d9 6f 27 19 8e 13 8b 8a d9 98 a7 84 5b a7 a5 bb 0c 2b dd f8 8f
7d 68 80 b0 04 16 68 29 73 3e ce 2a 36 c2 90 8a 13 64 f6 2d 84 c8 02 1c 6b f5 65 41 6f 4d 40 f9
b6 78 3d a2 15 8e 3b 1f 8f b7 73 20 01 00 85 a1 3c 82 6e 4c ba 41 aa 48 75 b9 97 62 c8 a3 e2 c3
d7 c1 0b f8 b9 aa a7 41 c3 a4 a1 e5 f8 2a aa 58 46 82 6a cc ef 07 3c ff 6f 8c 2c 0b 0b 87 b1 7d
b1 92 2f cb dc 3b be 87 8e aa cd 63 98 55 d9 b2 f1 c0 62 93 fc e2 ae 36 7a 07 c9 10 f1 b0 77 cb
20 b9 30 66 4b 93 b8 f0 f7 93 f4 86 1c b9 dd 1d 41 7b 72 74 d5 74 85 6a 95 04 b2 48 88 d4 53 58
c3 ae d0 f8 a0 71 47 42 40 3b 26 4d ec b9 ae 7e cb be eb 93 fb f9 c2 dd 7c 00 e3 5a db dd 9c 3b
9c 74 8e 92 1e 6d 24 96 05 91 63 22 e4 77 37 43 de 4b 92 7f 45 6f 31 6e 1c 2c b0 6a 57 88 ec 76
7c f4 02 b4 81 e5 bf 74 84 37 73 48 54 87 65 e7 85 7d 6d e7 73 5c 08 91 ab 92 82 45 31 f2 d2 66
f7 1a b8 0a 60 3f ee ff a5 2d 4f 62 20 ba 24 02 88 69 52 68 c0 39 0c 5a 51 37 c5 76 49 f7 9b d7
0f 88 f7 bb 2d bb 04 68 f8 74 53 2a bd ad d5 b9 5a 9d 55 c1 da 78 b6 22 81 bd 2e b9 01 d5 2b 8f
a4 aa 3b e7 77 eb ed 55 2a a1 c4 45 3c ce ee 10 5a 35 d1 a5 1c f0 0e 5f 8f c8 53 c1 dd 79 22 b2
```

round = -97

state = 88 4c

buffer = 01 00 01 00 01 00 01 00 01 00 01 00 01 00 01 00 10 00 10 00 10 00 10 00 66 e9 4b d4 ef 8a 2c
3b

round = -96

state = ee bf

buffer = 01 00 00 00 01 00 01 00 01 00 01 00 01 00 01 00 ff 00 10 00 10 00 10 00 66 e9 4b d4 ef 8a 2c
b2

round = -95

state = 03 b6

buffer = 01 00 00 00 01 00 00 00 01 00 01 00 01 00 01 d4 ff 00 10 00 10 00 10 00 66 e9 4b d4 ef 8a c0
b2

round = -94

state = d6 29

buffer = 00 00 00 00 01 00 00 00 01 00 01 00 01 00 4a d4 ff 00 10 00 10 00 10 00 66 e9 4b d4 ef 8d c0
b2

round = -93

state = 92 8c

```
buffer = 00 00 00 00 00 00 00 00 01 00 01 00 01 e9 4a d4 ff 00 10 00 10 00 10 00 66 e9 4b d4 31 8d c0
b2
```

◦

•

•

```
round = 1019
state = 01 dd
buffer = f4 31 e6 90 73 74 d8 1b ad b0 36 a6 3f 3e 4c af db 8e c8 18 c4 bb aa 16 cc 92 62 cf a1 80 82
b5
```

```
round = 1020
state = 81 79
buffer = f4 31 e6 91 73 74 ee 1b ad b0 36 be 3f 3e 4c af db 8e c8 18 30 bb aa 16 cc 92 62 cf a1 80 82
b5
```

```
round = 1021
state = 41 22
buffer = f4 31 67 91 73 c4 ee 1b ad b0 fe be 3f 3e 4c af db 8e c8 ad 30 bb aa 16 cc 92 62 cf a1 80 82
b5
```

```
round = 1022
state = 88 b2
buffer = f4 70 67 91 de c4 ee 1b ad 3e fe be 3f 3e 4c af db 8e 4a ad 30 bb aa 16 cc 92 62 cf a1 80 82
b5
```

```
round = 1023
state = 7c ba
buffer = 7c 70 67 8a de c4 ee 1b 76 3e fe be 3f 3e 4c af db 0e 4a ad 30 bb aa 16 cc 92 62 cf a1 80 82
b5
```

A.2 Example for *Enocoro-80*

A.2.1 Key, initialization vector, and keystream triplets

```
Key = 00 00 00 00 00 00 00 00 00 00 00
IV = 00 00 00 00 00 00 00 00
Key Stream = c9 22 79 45 6e be 3b ff d8 d4 73 12 3e ce b9 57
```

```
Key = 00 01 02 03 04 05 06 07 08 09
IV = 00 10 20 30 40 50 60 70
Key Stream = 9b 0a 97 39 4b 58 72 73 3d bf 9e e5 0c 33 73 3e
```

```
Key = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
IV = 00 00 00 00 00 00 00 00
Key Stream : 65 32 87 06 6e ad 27 95 4c aa 4d c0 65 f2 85 ac
```

A.2.2 Sample internal states

```
Key = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
IV = 00 00 00 00 00 00 00 00
Key Stream = 65 32 87 06 6e ad 27 95 4c aa 4d c0 65 f2 85 ac
```

```
round = -64
state = 88 4c
buffer = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 66 e9 4b d4 ef 8a 2c
3b

round = -63
state = a7 8e
buffer = b3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ef 00 00 00 00 00 00 00 66 e9 4b d4 ef 8a
2c

round = -62
state = 4a f9
buffer = 8b b3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 d4 ef 00 00 00 00 00 00 00 66 e9 4b d4 ef
8a

.

.

.

round = 14
state = ce 85
buffer = 89 21 31 7c 4b 46 63 44 f9 4e d9 e6 e1 50 7f 3b 68 6d 23 f0 8f 71 3b 59 03 2e 2f 19 ee 30 34
cb

round = 15
state = f0 ac
buffer = 05 89 21 52 7c 4b 46 63 7f f9 4e d9 e6 e1 50 7f 3b 86 6d 23 f0 8f 71 3b 59 03 2e 2f 19 ee 30
34

round = 16
state = 7e 04
buffer = c4 05 89 67 52 7c 4b 46 1c 7f f9 4e d9 e6 e1 50 7f 22 86 6d 23 f0 8f 71 3b 59 03 2e 2f 19 ee
30
```

A.3 Example for Trivium

A.3.1 Key, initialization vector, and keystream triplets

This section provides a numerical example of an 80-bit key, an 80-bit initialization value, and the first 128 corresponding bits of keystream produced by TRIVIUM.

Note that TRIVIUM is specified on bit level, and is indifferent to the order in which these bits are grouped into bytes. In order to simplify the verification of this example on software platforms with different endianness conventions, each group of eight bits is printed in two different hexadecimal formats. The first format maps the first bit of each byte to the most significant bit, and is better suited for big-endian platforms; the second one uses the reverse ordering, and is more natural on little-endian platforms.

80-bit Key:	[MSB first]	[LSB first]
0...31: 11110000 01000110 10101101 00010000	F0 46 AD 10	0F 62 B5 08
32...63: 11011010 01110101 10000000 00101010	DA 75 80 2A	5B AE 01 54
64...79: 11100101 01011111	E5 5F	A7 FA

80-bit IV:		[MSB first]	[LSB first]
0...31:	00010100 11110001 01101111 10111010	14 F1 6F BA	28 8F F6 5D
32...63:	00100011 11010100 01001001 10011111	23 D4 49 9F	C4 2B 92 F9
64...79:	00000110 11100011	06 E3	60 C7

First 128 bits of keystream:		[MSB first]	[LSB first]
0...31:	00100101 00011100 00110110 10110110	25 1C 36 B6	A4 38 6C 6D
32...63:	01101110 00100100 00011001 11111100	6E 24 19 FC	76 24 98 3F
64...95:	01010111 10110001 01111101 11001110	57 B1 7D CE	EA 8D BE 73
96..127:	00101000 10100111 01111111 11111000	28 A7 7F F8	14 E5 FE 1F

A.3.2 Internal Sequence Bits

The values of the internal sequence bits a_i , b_i , and c_i which were computed in order to generate the previous example, are listed below.

i:	-1280	-1272	-1264	-1256	-1248	-1240	-1232	-1224
a[i]:					00000	00000000	11110000	01000110
b[i]:						0000	00010100	11110001
c[i]:			1110000	00000000	00000000	00000000	00000000	00000000

i:	-1216	-1208	-1200	-1192	-1184	-1176	-1168	-1160
a[i]:	10101101	00010000	11011010	01110101	10000000	00101010	11100101	01011111
b[i]:	01101111	10111010	00100011	11010100	01001001	10011111	00000110	11100011
c[i]:	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

[the 1152 blank rounds start here]

i:	-1152	-1144	-1136	-1128	-1120	-1112	-1104	-1096
a[i]:	01010101	01101000	10000110	11010011	10101100	00000001	01010111	00101010
b[i]:	11111000	10001000	00001010	00000000	00000111	11010100	01001010	10010111
c[i]:	10001010	00101110	11001110	00000011	00000000	00000001	10111100	10001101

i:	-1088	-1080	-1072	-1064	-1056	-1048	-1040	-1032
a[i]:	11011000	00100000	11110111	10110110	01011101	01100001	01110001	11110101
b[i]:	11001111	10001110	11101000	10001110	01101100	01000100	00110100	00110110
c[i]:	00101111	00110010	00000000	11001100	11011101	00000010	10100100	11011001

i:	-1024	-1016	-1008	-1000	-992	-984	-976	-968
a[i]:	10110001	00000011	10000111	10001001	00011100	10010001	11000100	01011101
b[i]:	01111100	01011101	11011010	10001001	11110100	01101000	00100010	10011110
c[i]:	11111001	10101100	11100000	00000011	01100001	11101010	11010111	11011111

i:	-960	-952	-944	-936	-928	-920	-912	-904
a[i]:	11010000	01011000	11111110	00111001	11011011	01101010	00001100	10101111
b[i]:	11010011	10010100	10000001	00101010	10100101	00101111	11101100	11111110
c[i]:	11110001	11101100	10110101	11100001	11111001	11001011	11100100	10011001

i:	-896	-888	-880	-872	-864	-856	-848	-840
a[i]:	11010010	11110110	00101110	01011110	00000001	01101001	11001101	10111010
b[i]:	10011010	11111011	01010010	10101010	11110001	10111000	00100101	01000011
c[i]:	10100001	00011001	01100110	10110011	11000100	01010000	00010110	11000111

i:	-832	-824	-816	-808	-800	-792	-784	-776
a[i]:	11111010	00010101	11011101	01000110	11001000	01101000	00001001	10101111
b[i]:	10111000	00010111	00000101	11111100	01010011	00000000	00000110	10111110
c[i]:	01100100	11001110	10000010	01100110	10010000	01101010	00010101	10011111

i:	-768	-760	-752	-744	-736	-728	-720	-712
a[i]:	11010011	00001010	10000110	11010011	10101101	11001110	00001110	10000111
b[i]:	00100100	11011101	01101001	11010101	01110101	01011100	00100100	00010101
c[i]:	00111111	10111001	01101011	01100111	11001111	11111100	00001000	01100101

i:	-704	-696	-688	-680	-672	-664	-656	-648
a[i]:	10110100	10101011	00101111	00111011	01000001	11010111	11110001	11011110
b[i]:	10101001	01111010	01101111	01000111	11110100	01110010	00101000	01111100
c[i]:	00100101	11100001	10110111	01101111	00000111	11010001	01110101	11101001

i:	-640	-632	-624	-616	-608	-600	-592	-584
a[i]:	10101011	00001100	10010100	10111010	00001010	00110000	10101110	00110011
b[i]:	10001011	11001100	11011010	01101000	11001000	00010011	10110111	01000101
c[i]:	10010111	00011010	01100011	10101000	01010011	11111001	11001100	00000111

i:	-576	-568	-560	-552	-544	-536	-528	-520
a[i]:	01100111	01111000	11101101	11101100	10101000	01111000	11000000	00000101
b[i]:	00011101	11110000	10011101	00101000	10010011	11001011	10101101	00010010
c[i]:	01100100	00111101	01111101	00110000	10101101	00101001	01010001	10011100

i:	-512	-504	-496	-488	-480	-472	-464	-456
a[i]:	10001001	01100100	00110000	00010110	11110110	00011110	01011010	11111100
b[i]:	01010111	01001111	11111111	00000110	00000010	01110111	10001100	11001010
c[i]:	10000100	11000011	11011000	11000010	00110100	10110110	10110101	01111110

i:	-448	-440	-432	-424	-416	-408	-400	-392
a[i]:	00010111	10011010	10101101	11100010	10011101	11000100	01010100	00000001
b[i]:	10010010	11011110	01010001	00000110	00001110	10111100	00011111	10011000
c[i]:	00011011	11001000	01100011	01000100	10110111	10010010	11011000	00100100

i:	-384	-376	-368	-360	-352	-344	-336	-328
a[i]:	10010101	11001010	00100100	01000011	11010010	00000110	11101101	00111000
b[i]:	11000111	11111111	00111000	01100001	00101110	10111100	11011100	11100101
c[i]:	10100000	00100000	10100111	10000001	11000101	10100011	10001000	11111111

i:	-320	-312	-304	-296	-288	-280	-272	-264
a[i]:	11101010	00101111	10001011	11100101	10011111	10110001	10010101	00010111
b[i]:	01101101	00110011	00110110	11110000	10100011	01010100	00100011	10011011
c[i]:	00100001	10010100	00101100	00000011	01111101	10110100	11110110	00000000

```

      i: -256      -248      -240      -232      -224      -216      -208      -200
          |         |         |         |         |         |         |
a[i]:  01011100  00110001  11111000  00010100  11100001  10010110  11110110  00000000
b[i]:  11011001  01100101  10111100  11111110  11101101  10100010  10001011  00101000
c[i]:  10110010  10101110  00111000  00100011  11011010  10110001  10101101  10010010

```

```

      i: -192      -184      -176      -168      -160      -152      -144      -136
          |         |         |         |         |         |         |
a[i]:  11101110  01000101  00011110  00100010  00101101  10100001  00111001  10111000
b[i]:  01010110  11111111  10110011  01001000  10101000  00001110  11001011  10100001
c[i]:  11110101  01111100  11100011  00011100  11010111  11001101  00100100  00010110

```

```

      i: -128      -120      -112      -104      -96      -88      -80      -72
          |         |         |         |         |         |         |
a[i]:  00001010  01100100  11100101  11010001  01011011  10111000  00101011  11010010
b[i]:  00001011  11001001  11101100  01111101  11110100  01100011  11011111  01000100
c[i]:  00001001  00011110  01011101  10000000  10100011  11110010  11000101  01110011

```

```

      i: -64      -56      -48      -40      -32      -24      -16      -8
          |         |         |         |         |         |         |
a[i]:  10001100  01001111  00000001  11110100  11101010  00011101  01100010  01001110
b[i]:  11010001  00010111  11001000  10010011  10110000  00110010  11101101  11110000
c[i]:  11101110  11001100  01110011  10111101  01110100  11010000  11111100  11110011

```

[the keystream generation starts here]

```

      i:  0        8        16        24        32        40        48        56
          |         |         |         |         |         |         |
a[i]:  10110000  11010000  00101100  10000111  01110001  00001100  00111011  10010111
b[i]:  00110000  11000001  11100010  10110110  11111010  01010001  10001001  11110011
c[i]:  10001101  00110110  00000100  10001000  10001001  11101111  10110010  10000001

```

```

z[i]:  00100101  00011100  00110110  10110110  01101110  00100100  00011001  11111100

```

```

      i:  64       72       80       88       96       104       112       120
          |         |         |         |         |         |         |
a[i]:  11111101  01010000  01101011  11100100  00000011  10011000  10110111  10000000
b[i]:  01001010  10011011  11011010  10010001  11011000  00011001  00000011  11111110
c[i]:  00010111  10011100  11101111  00010011  11111101  11110100  01101100  11111001

```

```

z[i]:  01010111  10110001  01111101  11001110  00101000  10100111  01111111  11111000

```

A.3.3 Internal State

As mentioned in 6.3.2, a typical implementation of TRIVIUM will only need to maintain an internal state of 288 bits. The content of the internal state S_0 after the 1152 blank rounds is printed below.

```

      i: -128      -120      -112      -104      -96      -88      -80      -72
          |         |         |         |         |         |         |
a[i]:                                     11011 10111000 00101011 11010010
b[i]:                                     0011 11011111 01000100
c[i]:          1011101 10000000 10100011 11110010 11000101 01110011

```

```

      i: -64      -56      -48      -40      -32      -24      -16      -8
          |         |         |         |         |         |         |
a[i]:  10001100  01001111  00000001  11110100  11101010  00011101  01100010  01001110
b[i]:  11010001  00010111  11001000  10010011  10110000  00110010  11101101  11110000
c[i]:  11101110  11001100  01110011  10111101  01110100  11010000  11111100  11110011

```


A.3.4 Parallelism

The following example illustrates how TRIVIUM's parallelism enables implementers to compute 64 bits of b_i at once, using three 64-bit XOR operations and one 64-bit AND operation.

i:	0	8	16	24	32	40	48	56
(1):	10100011	00010011	11000000	01111101	00111010	10000111	01011000	10010011
(2):	11011101	11000001	01011110	10010100	01100010	01111000	00001111	10100111
(3):	10111011	10000010	10111101	00101000	11000100	11110000	00011111	01001110
(4):	01110111	00000101	01111010	01010001	10001001	11100000	00111110	10011101
(5):	01111101	00010011	01000100	01011111	00100010	01001110	11000000	11001011

(6):	00110000	11000001	11100010	10110110	11111010	01010001	10001001	11110011

(1) = a[i - 66]
 (2) = a[i - 93]
 (3) = a[i - 92]
 (4) = a[i - 91]
 (5) = b[i - 78]

(6) = b[i] = (1) XOR (2) XOR [(3) AND (4)] XOR (5)

Annex B (informative) Usage Notes

B.1 Usage Note for Trivium

B.1.1 Parallelism

A useful feature of the recurrence relations used in TRIVIUM is that the bits computed at a given point in time only affect subsequent computations after a delay of at least 66 iterations. As a consequence, up to 66 consecutive iterations (the most natural choices are 8, 16, 32, or 64) can be computed in parallel without any interference. An illustration of this property is given in B.1.1.

NOTE Note that there are probably not many applications of TRIVIUM for which it would *not* make sense to exploit this parallelism to at least some extent. Parallel hardware implementations can achieve a significantly lower power consumption or higher throughput in exchange for a modest increase in area. In software, TRIVIUM's parallelism makes it possible to take advantage of the largest word size available on a given architecture.

B.1.2 Recommended Use of Initialization Values

This section provides recommendations on how to use initialization values in the most effective way. Given the fact that TRIVIUM uses a relatively short 80-bit secret key, an improper use of initialization values may reduce its security to a dangerously low level. It is important to note that TRIVIUM's initialization value serves two purposes:

- a) It allows data, encrypted with the same secret key, to be split into chunks which can be decrypted in arbitrary order.
- b) It increases the security level against generic attacks.

In order to better reflect these two different purposes, it is useful to split the 80-bit initialization value IV into two components, I and V :

$$(IV_{79}, \dots, IV_0) = (0, \dots, 0, I_{n-1}, \dots, I_0) \oplus (V_{79}, \dots, V_0).$$

The first component I is a simple n -bit counter which uniquely identifies each chunk of data. It is assumed to be publicly known, and if its value cannot be derived in any other way, then it needs to be transmitted for each chunk. Its length n depends on the maximum number of randomly accessible data chunks that the application should be able to encrypt under a single key. Note that there is often no need to make n very large, as illustrated in the following examples.

EXAMPLE 1 A 1x speed DVD drive reads data at 10Mbit/s and has a typical access time of 100ms. In order to access arbitrary parts of an encrypted disc without causing any additional delays, the decryption device will have to generate keystream at a speed of 10Mbit/s and be able to reach any point in the keystream within 100ms. This requirement can easily be met by reinitializing TRIVIUM with a different value of I after each chunk of 1Mbit. In this case, a 16-bit counter I would suffice to encrypt a 4.7GB disc.

EXAMPLE 2 In applications involving real-time communication (e.g., voice conversations), it typically does not make sense to decrypt data in any different order than the one used during encryption. The whole conversation can hence be encrypted as a single stream, eliminating the need for a counter, i.e., $n = 0$. Note however that in order to compensate for small synchronization differences, or for data packets arriving out-of-order or being dropped, the keystream bits will probably need to be generated at a slightly higher speed than the transmission rate and temporarily kept in a buffer.

The second component V can be used to increase TRIVIUM's resistance against generic attacks, and the most effective way to do so, is to treat it as an additional secret key. That is, whenever a new key is needed, both K and V are initialized simultaneously using a larger 160-bit secret key K' :

$$(K_0, \dots, K_{79}, V_0, \dots, V_{79}) = (K'_0, \dots, K'_{79}, K'_{80}, \dots, K'_{159}).$$

It is important to realize that the use of such an extended key K' will not necessarily increase the security of TRIVIUM against dedicated attacks. The guess-and-determine attack proposed by Maximov and Biryukov [1], for instance, requires an effort roughly equivalent to an exhaustive search over a 90-bit key space, and this does not depend on how TRIVIUM is initialized. Generic brute-force attacks, on the other hand, will necessitate considerably larger computational resources, especially when n can be kept relatively small. Considering the fact that the most efficient dedicated attacks require the adversary to intercept very large amounts of data (in the order of hundreds of petabytes in the case of [1]), the use of an extended key will in practice significantly increase the security margin of TRIVIUM.

Should users decide, because of specific constraints of the application, to deviate from the recommended procedure outlined above, then they should at least take measures to enforce the following rules.

- c) Two different streams of data should never be encrypted with the same key K and the same initialization value IV . A violation of this rule would expose the XOR of the two plaintexts to the adversary. If the two data streams contain some redundancy, then this information would often suffice to recover both of them.

The same initialization value should not be reused with a large number of different keys. Suppose for instance that a single publicly known initialization value would be used with 2^{24} (16 million) different 80-bit secret keys. In that case, recovering at least one of those 80-bit keys would not be harder than recovering a single 56-bit key.

Bibliography

- [1] A. Maximov and A. Biryukov. Two Trivial Attacks on TRIVIUM. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Selected Areas in Cryptography, SAC 2007*, volume 4876 of *Lecture Notes in Computer Science*, pages 36–55. Springer-Verlag, 2007.
- [2] Hitachi, Ltd, "Stream Cipher Enocoro Evaluation Report". CRYPTREC submission package. <http://www.sdl.hitachi.co.jp/crypto/enocoro/index-en.html>
- [3] D. Watanabe, K. Ideguchi, J. Kitahara, K. Muto, H. Furuichi, T. Kaneko, "Enocoro-80: A Hardware Oriented Stream Cipher". *ARES 2008*: 1294-1300