ETSI/SAGE	**Version: 1.1**
Technical report	**Date: 11th August 2010**

Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 4: Design and Evaluation Report

Document History		
0.1	**20th June 2010**	**First draft of main technical text**
1.0	**11th August 2010**	**First public release**
1.1	**11th August 2010**	**A few typos corrected and text improved**

Reference

Keywords

3GPP, security, SAGE, algorithm

ETSI Secretariat

Postal address

F-06921 Sophia Antipolis Cedex - FRANCE

Office address

650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

X.400

c= fr; a=atlas; p=etsi; s=secretariat

Internet

secretariat@etsi.fr
http://www.etsi.fr

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for ETSI members and non-members, and can be found in SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://www.etsi.org/ipr).

At the time of writing of this report, no information of any Intellectual Property Rights (IPRs) has been drawn to the attention neither to the Task Force nor to ETSI.

No guarantee can be given as to the existence of other IPRs not referenced in SR 000 314 (or the updates on the ETSI Web server (http://www.etsi.org/ipr) which are, or may be, or may become, essential to the present document.

Foreword

This Report has been produced by the ETSI SAGE Task Force on the Design of the third LTE encryption and integrity protection algorithms 128-EEA3 and 128-EIA3 (SAGE TF 3GPP).

The work described in this report was undertaken in response to a request made by 3GPP.

1 Scope

This public report contains a detailed summary of the design and evaluation work performed during the development of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 and 128-EIA3. The report also includes summaries of evaluations that were conducted by independent external evaluators, and reflects modifications that were done to the design based on this feedback.

2 References

For the purposes of this report, the following references apply:

[1] 3G TS 33.401 V 9.3.1 (2010-04) 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3GPP System Architecture Evolution (SAE); Security architecture (Release 9).

[2] ETSI/SAGE Specification. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 1: 128-EEA3 and 128-EIA3 Specification; Version: 1.0; Date: 18th June 2010.

[3] ETSI/SAGE Specification. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification; Version: 1.0; Date: 18th June 2010.

Additional references to external documents are provided in Annex A.

3 Abbreviations

For the purposes of the present report, the following abbreviations apply:

AES	Advanced Encryption Standard
CK	Cipher Key
ETSI	European Telecommunications Standards Institute
GF(*q*)	The Galois field with *q* elements
3GPP	3rd Generation Partnership Project
EPS	Evolved Packet System (comprising LTE and SAE)
EEA	EPS Encryption Algorithm
EIA	EPS Integrity Algorithm
128-EEA3	Proposed third algorithm fulfilling the EEA requirement
128-EIA3	Proposed third algorithm fulfilling the EIA requirement
IBS	Input Bit Stream
IK	Integrity Key
IPR	Intellectual Property Rights
IV	Initialization Vector
LFSR	Linear Feedback Shift Register
LTE	Long Term Evolution (radio network)
MAC	Message Authentication Code
OBS	Output Bit Stream
OTP	One Time Pad
SA3	3GPP Systems and Architecture Group
SAE	System Architecture Evolution (core network)

SAGE	Security Algorithms Group of Experts
SAGE TF 3GPP	SAGE Task Force for the design of the standard 3GPP Confidentiality and Integrity Algorithms
XL	Extended Linearization

4 Structure of this report

The material presented in this report is organised in the subsequent sections, as follows:

- Section 5 provides background information on the third 3GPP Confidentiality and Integrity Algorithms 128-EEA3 and 128-EIA3;
- Section 6 provides a summary of the algorithm parameters;
- Section 7 consists of a brief presentation of the actual designs;
- Section 8 provides information on overall design rationale for ZUC;
- Section 9 provides information on design and evaluation of ZUC components;
- Section 10 provides information on overall rationale on 128-EEA3 and 128-EIA3 constructions;
- Section 11 gives the evaluation of resistance against cryptanalytic attacks;
- Section 12 concludes the evaluation;
- Annex A includes a list of external references that are related to the results in this report.

5 Background to the design and evaluation work

The security architecture for LTE is specified in ref. [1]. This includes various services that need to be provided by standardised cryptographic algorithms. In particular, two standardised algorithms are required for the radio interface, namely:

- EEA – Encryption algorithm
- EIA – Integrity algorithm

Before this work began, two encryption and integrity algorithm sets had already been developed and standardised for LTE. The first set, 128-EEA1 and 128-EIA1, is based on SNOW 3G; the second, 128-EEA2 and 128-EIA2, is based on AES. (The prefix “128-” indicates that the algorithms take a 128-bit secret key.)

3GPP SA3 agreed in May 2009 on a requirement for a third encryption and integrity algorithm set – one designed in China, so that the Chinese authorities would permit its use in that country.

The resulting algorithm set is based on a core stream cipher algorithm named ZUC, after Zu Chongzhi, the famous Chinese scientist from history. The algorithms were designed by experts at the Data Assurance and Communication Security Research Center (DACAS) of the Chinese Academy of Sciences.

Of course, “an algorithm from China” is not enough of a requirement. It was agreed that a robust, three-phase evaluation programme would be followed:

- evaluation by an ETSI SAGE task force;
- evaluation by two funded teams of academic experts, delivering their results ([32],[33]) to the ETSI SAGE task force;

- after that evaluation, if the task force recommended that the algorithm (modified or not) is suitable for acceptance into the standard, then a public evaluation phase would take place before final standardisation.

The SAGE task force set high security goals for the algorithm set. To recommend the algorithms, it was not sufficient merely that no practical attacks were found during the evaluation. Rather, the task force required that the algorithms should be judged to have a high security margin, and also that the design rationale behind all components of the algorithm should be clear and transparent.

6 Algorithm parameters

The text in this section is taken directly from [1].

6.1 EEA – Encryption algorithm

The input parameters to the ciphering algorithm are a 128-bit cipher key named KEY, a 32-bit COUNT, a 5-bit bearer identity BEARER, the 1-bit direction of the transmission i.e. DIRECTION, and the length of the keystream required i.e. LENGTH. The DIRECTION bit shall be 0 for uplink and 1 for downlink.

Figure 6.1 illustrates the use of the ciphering algorithm EEA to encrypt plaintext by applying a keystream using a bit per bit binary addition of the plaintext and the keystream. The plaintext may be recovered by generating the same keystream using the same input parameters and applying a bit per bit binary addition with the ciphertext.

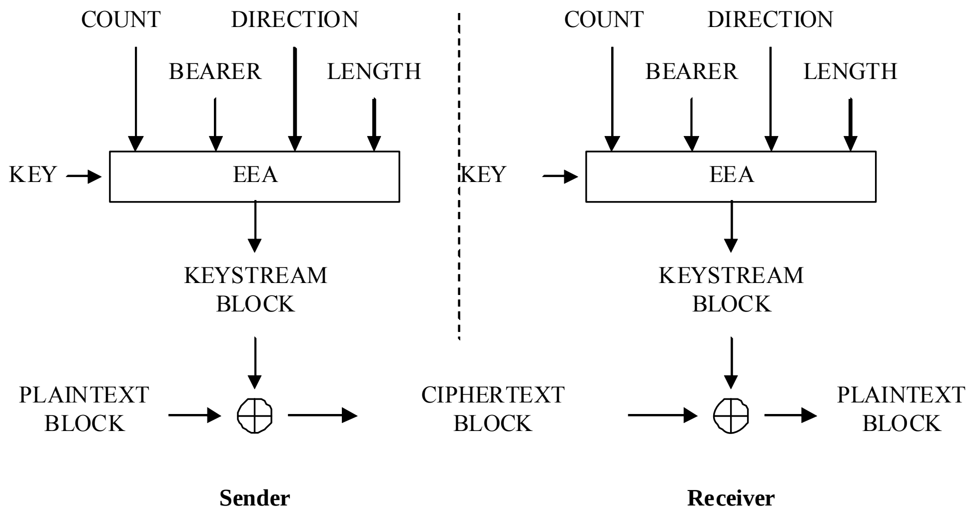

Figure 6.1: Ciphering of data

Based on the input parameters the algorithm generates the output keystream block KEYSTREAM which is used to encrypt the input plaintext block PLAINTEXT to produce the output ciphertext block CIPHERTEXT.

The input parameter LENGTH shall affect only the length of the KEYSTREAM BLOCK, not the actual bits in it.

6.2 EIA – Integrity algorithm

The input parameters to the integrity algorithm are a 128-bit integrity key named KEY, a 32-bit COUNT, a 5-bit bearer identity called BEARER, the 1-bit direction of the transmission i.e. DIRECTION, and the message itself i.e. MESSAGE. The DIRECTION bit shall be 0 for uplink and 1 for downlink. The bit length of the MESSAGE is LENGTH.

Figure 6.2 illustrates the use of the integrity algorithm EIA to authenticate the integrity of messages.

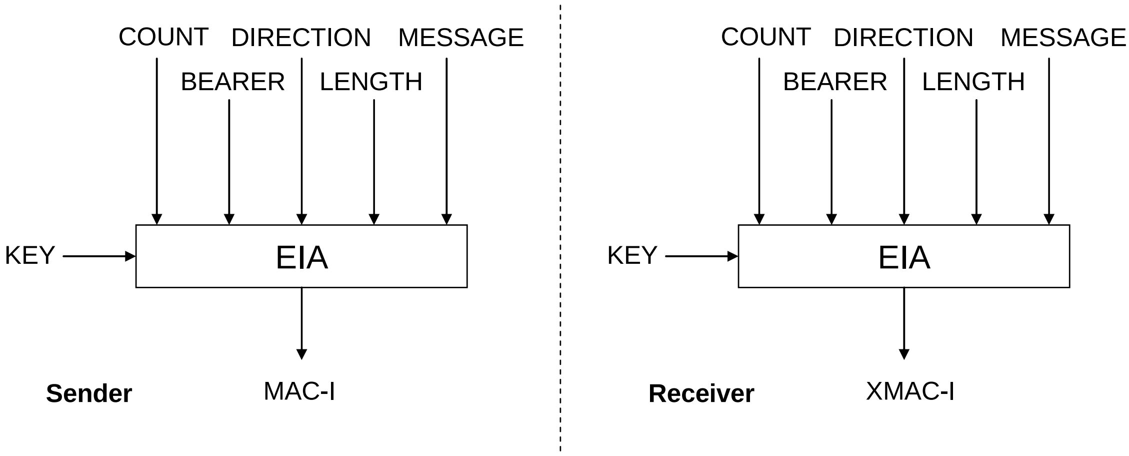

Figure 6.2: Derivation of MAC-I/NAS-MAC (or XMAC-I/XNAS-MAC)

Based on these input parameters the sender computes a 32-bit message authentication code (MAC-I/NAS-MAC) using the integrity algorithm EIA. The message authentication code is then appended to the message when sent. The receiver computes the expected message authentication code (XMAC-I/XNAS-MAC) on the message received in the same way as the sender computed its message authentication code on the message sent and verifies the data integrity of the message by comparing it to the received message authentication code, i.e. MAC-I/NAS-MAC.

7 Summary of the 128-EEA3 and 128-EIA3 algorithms

The detailed specifications of the 128-EEA3 and 128-EIA3 algorithms can be found in ref. [2] and ref.[3]. This section includes a brief overview of the 128-EEA3 and 128-EIA3 designs. The basic building block for both 128-EEA3 and 128-EIA3 is the stream cipher algorithm ZUC, which is composed of three components with an internal state of 560 bits initialized from a 128-bit cipher key K and a 128-bit initialization vector IV.

7.1 The stream cipher ZUC

The structure of ZUC is depicted in the following diagram:

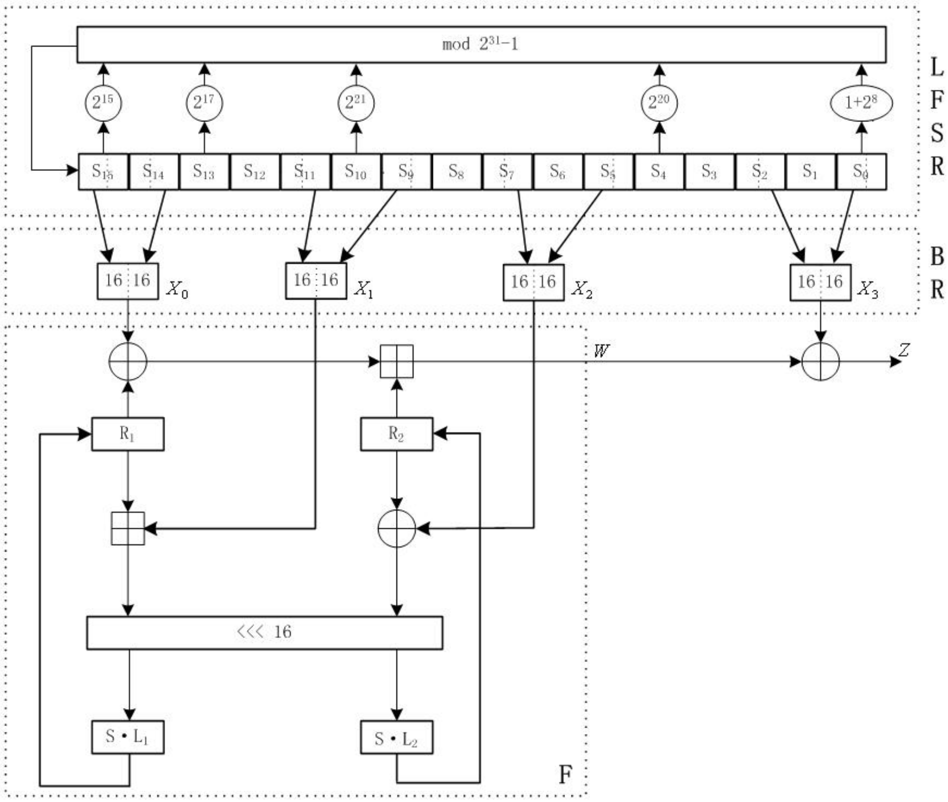

Figure 7.1: The structure of ZUC

ZUC consists of a Linear Feedback Shift Register (LFSR), a bit reorganization and a nonlinear function *F*. The LFSR is constructed from 16 register units, each holding 31 bits, and the feedback is defined by a primitive polynomial over the finite field GF(2^{31}-1) . The bit reorganization extracts 128 bits from the state of the LFSR and forms four 32-bit words which will be used by the nonlinear function *F*. The nonlinear function *F* is based upon two 32-bit registers R_1 and R_2. The operation of the nonlinear function *F* involves input from the bit reorganization and uses two S-boxes S_0 and S_1. The mixing operations are the exclusive OR, the cyclic shift and the addition modulo 2^{32}. See ref. [3] for details on the specification of S-boxes.

7.2 Confidentiality function 128-EEA3

The new confidentiality algorithm (128-EEA3) encrypts and decrypts frames using ZUC as a standard synchronous stream cipher. Ref. [2] defines how the system parameters COUNT, BEARER and DIRECTION are used together with the Confidentiality Key (CK) to initialise the keystream generator.

The output from ZUC consists of 32-bit words that are XORed to the corresponding Input Bit Stream (IBS).

The main stream cipher principles of 128-EEA3 are shown in Figure 7.2. The produced cipher text block is denoted as Output Bit Stream (OBS).

For decryption the same scheme is used to recover the plain text block (IBS) from the received cipher text (OBS). Sender and receiver will synchronize for each frame using the frame counter COUNT-C.

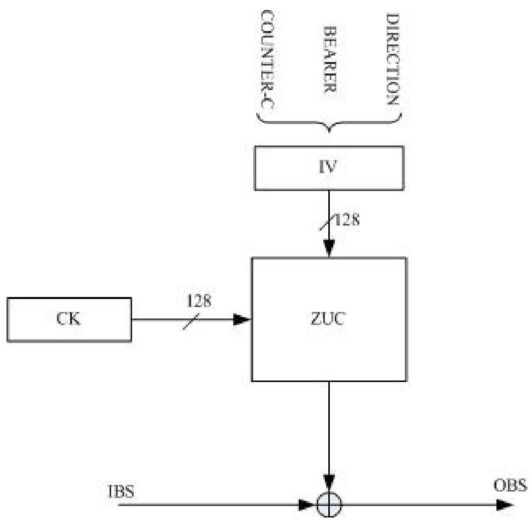

Figure 7.2: Principles of the 128-EEA3 encryption operation

7.3 Integrity function 128-EIA3

The new integrity algorithm (128-EIA3) computes a 32-bit Message Authentication Code (MAC) on an input message under an integrity key IK. We assumed that the message could be between 1 and 20000 bits in length (but the algorithm supports longer messages if needed). The 128-EIA3 algorithm is based on universal hashing.

Prior to processing the message, the ZUC generator is initialized with the integrity key IK and the initialization vector IV, and a keystream with length 64 bits more than that of the message is generated.

The message is padded with a bit 1. Set the initial value of an accumulator variable T to be 0. For each bit of the padded message, if the *i*-th bit of message is 1, then the accumulator accumulates T with the word defined by the successive 32 bits starting from the *i*-th bit of the keystream. Finally T is viewed as the MAC of the message and is outputted.

The structure of 128-EIA3 is depicted in Figure 7.3.

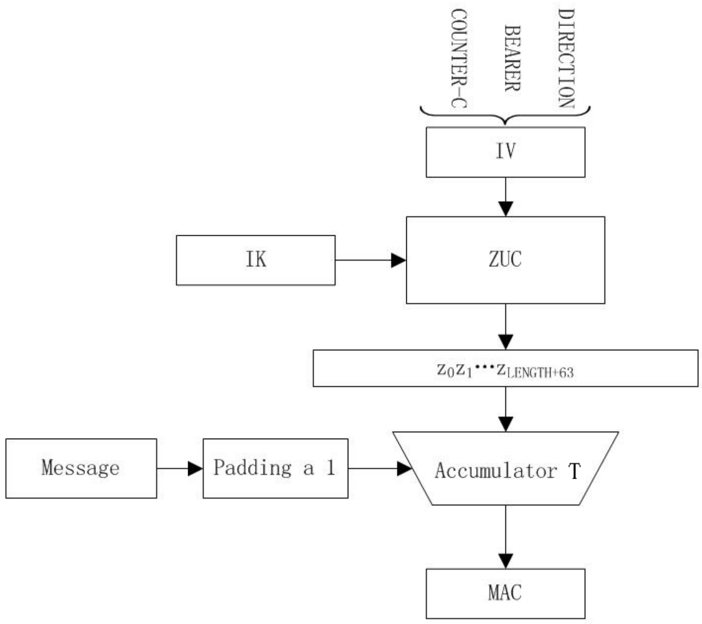

Figure 7.3: Principles of the 128-EIA3 MAC computation

8 Overall design rationale for ZUC

The essential design goals for the third suite of 3GPP confidentiality and integrity algorithms were that the algorithms should:

- provide a high level of security within the 3GPP context;
- meet the implementation requirements deemed by 3GPP — in particular, allow a low power, low gate count implementation in hardware;
- the fundamental cryptographic principles be substantially different from the AES-based and SNOW 3G-based constructions – so that progress in cryptanalysis towards one of the algorithms should not directly lead to attacks against the other.

The ZUC algorithm adopts a linear feedback shift register (LFSR) that generates *m*-sequences over the prime field GF(2^{31}-1) as source sequences of the algorithm, which obviously have substantial differences from traditional stream ciphers that are based on *m*-sequences over the finite field GF(2) or its extension field GF(2^n). The *m*-sequences over the prime field GF(2^{31}-1) are not highly structured over GF(2) – for example, they have high linear complexity over GF(2), low advantage of bit coincidence between mod p and mod 2 sums, and many other good cryptographic properties. This component of *m*-sequence generator has significant contribution to the ZUC algorithm against bit-oriented cryptographic attacks, including fast correlation attacks, linear distinguishing attacks, and algebraic attacks.

The design of the nonlinear function F adopts some structures in block ciphers, namely S-boxes and a mixture of XOR and modular addition. This design gives protection against many well known cryptographic attacks, including guess and determine attacks, fast correlation attacks and linear distinguishing attacks. Between the nonlinear function F and the feedback shift registers, there is a bit-reorganization layer. Since the algebraic structures of the linear feedback shift register, the bit-reorganization layer, and the nonlinear function F, are entirely different, the combination of them is expected to enhance the overall security. The

LFSR on its own can of course be analysed over GF(2^{31}-1), but the additional layers make it hard to pursue that analysis through the cipher as a whole.

The following types of attacks against the stream cipher ZUC have been particularly considered:

- algebraic attacks;
- guess and determine attacks;
- fast correlation attacks;
- distinguishing attacks.

Attacks against the 128-EEA3 and 128-EIA3 constructions are also considered.

9 Design and evaluation of ZUC components

9.1 Design of the LFSR

As explained in section 8, ZUC uses a LFSR operating over GF(2^{31}-1) instead of one operating over GF(2) or GF(2^n). In this section we analyse some properties of LFSRs over GF(2^{31}-1), and explain how the characteristic polynomial of the LFSR was selected.

9.1.1 Subsequences derived from the binary coordinates are equivalent by shift

Let $p=2^{31}-1$ and GF(p) be the finite field with p elements. For any given primitive polynomial $f(x)$ over GF(p), let $\underline{a}$ be an m-sequence over GF(p) generated by $f(x)$. Then it can be written in 2-adic representation, i.e., in terms of its binary coordinate sequences as:

$$\underline{a}=\underline{a}_0+\underline{a}_1 2+\ldots+\underline{a}_{30}2^{30}\ , \tag{1}$$

where $\underline{a}_i$ ($0\le i\le 30$) is a binary sequence composed of 0's and 1's, which is called the i-coordinate sequence of $\underline{a}$.

Denote by $G(f(x),p)$ the set of all sequences generated by $f(x)$ over GF(p). For any $\underline{a}\in G(f(x),p)$, we have

$$2\underline{a}=\underline{a}_{30}+\underline{a}_0 2+\ldots+\underline{a}_{29}2^{30} \bmod p \tag{2}$$

On the other hand, for any $\underline{a}=(a_0,a_1,a_2,\ldots)\in G(f(x),p)$, it can be represented as follows:

$$\underline{a}=(\mathrm{Tr}(\lambda\alpha^0),\mathrm{Tr}(\lambda\alpha^1),\mathrm{Tr}(\lambda\alpha^2),\ \ldots)$$

where α is a root of $f(x)$, $\lambda\in$ GF(p)(α), and Tr() is a trace function from GF(p)(α) to GF(p). Hence we get

$$2\ \underline{a}=(2\mathrm{Tr}(\lambda\alpha^0),2\mathrm{Tr}(\lambda\alpha^1),2\mathrm{Tr}(\lambda\alpha^2),\ \ldots)$$

$$=(\mathrm{Tr}(2\lambda\alpha^0),\mathrm{Tr}(2\lambda\alpha^1),\mathrm{Tr}(2\lambda\alpha^2),\ \ldots)$$

$$=(\mathrm{Tr}(\lambda\alpha^m),\mathrm{Tr}(\lambda\alpha^{m+1}),\mathrm{Tr}(\lambda\alpha^{m+2}),\ \ldots)$$

$$=(a_m,a_{m+1},a_{m+2},\ \ldots) \qquad (3)$$

where m=5/31 ($p^{\deg(f(x))}$-1) such that α^m=2.

This means that the 1-coordinate sequence of $\underline{a}$ is obtained by shifting its 0-coordinate sequence by m bits to the left, and the 2-coordinate sequence of $\underline{a}$ is obtained by shifting its 1-coordinate sequence by m bits to the left, and so on, and the 30-coordinate sequence of $\underline{a}$ is obtained by shifting its 29-coordinate sequence by m bits to the left.

Moreover, since the coordinate sequences are all periodic, they are equivalent by shift.

9.1.2 The LFSR sequences are entropy-lossless under the modulo 2 operation

In this section we will prove that, the set $G(f(x),p)$ of the m-sequences generated by $f(x)$, are entropy-lossless under the modulo 2 operation. More precisely, we have

Theorem 1 *Let $f(x)$ be a primitive polynomial over* GF*(p), and let $G(f(x),p)$ denote the set of sequences over* GF*(p) generated by $f(x)$. Then for any $\underline{a},\underline{b}\in G(f(x),p)$, we have that, $\underline{a}=\underline{b}$ if and only if $\underline{a}$* mod 2=$\underline{b}$ mod 2 *holds.*

Proof: The necessity is obvious. Now we prove the sufficiency. Assume the contrary that there exist $\underline{a}$, $\underline{b}\in G(f(x),p)$ satisfying that $\underline{a}\neq \underline{b}$ and $\underline{a}$ mod 2 = $\underline{b}$ mod 2. We will show that in this case, both $\underline{a}$ and $\underline{b}$ are not the all-zero sequence. Without loss of generality, let $\underline{a}$ = $\underline{0}$, then we have

$$\underline{b}_0 = \underline{b} \bmod 2 = \underline{a} \bmod 2 = \underline{0}.$$

By the property in section 9.1.1 that all the coordinate sequences are equivalent by shift, we must have $\underline{b}$ = $\underline{0}$. Hence $\underline{a}$ = $\underline{b}$, which is a contradiction. So neither $\underline{a}$ nor $\underline{b}$ can be a zero sequence.

Let $\underline{c}$ = $\{c_t\}_{t\geq 0}$ such that $c_t=a_t+b_t$ mod p for all $t\geq 0$, that is, $\underline{c}=\underline{a}+\underline{b}$. Then $\underline{c}\in G(f(x),p)$. Now consider positions t such that c_t=0. If $a_t\neq 0$ for some t, then we must have $a_t+b_t=p$. It follows that a_t mod 2$\neq$ b_t mod 2, that is, $\underline{a}$ mod 2$\neq$ $\underline{b}$ mod 2. A contradiction. Thus for all positions t such that c_t=0, we have $a_t=b_t$=0. Note that $\underline{c}\neq \underline{0}$ and is an m-sequence, we observe a state (c_t,0,…,0) of $\underline{c}$ generated the LFSR defined by $f(x)$, where $c_t\neq 0$. The corresponding states of $\underline{a}$ and $\underline{b}$ are (a_t,0,…,0) and (b_t,0,…,0) respectively. Since both $\underline{a}$ and $\underline{b}$ are non-zero sequences, thus both a_t and b_t are not equal to zero. Let $a_t=\lambda b_t$ mod p, where $0\neq\lambda\in$ GF(p). Then $\underline{a}=\lambda\underline{b}$ mod p holds. Since $\underline{a}\neq \underline{b}$, we must have that λ>1. Note that $\underline{b}$ is an m-sequence over GF(p), there must exists an integer t such that b_t=1, and in which case we have $a_t=\lambda$. By the equality that a_t mod 2=b_t mod 2 we know that, λ must be an odd number. Once again by using the properties of m-sequences, we know that there exists an integer t such that $b_t=[2p/\lambda]$, where $[x]$ is the minimum integer no more than x. Note that $p<\lambda b_t\leq 2p$, therefore we have

$$a_t=\lambda b_t \bmod p=\lambda b_t-p$$

and hence we get

$$a_t \bmod 2 = b_t+1 \bmod 2 \neq b_t \bmod 2.$$

This means that $\underline{a} \bmod 2 \neq \underline{b} \bmod 2$. This is a contradiction against the assumption, and hence the conclusion of the theorem holds. ■

By theorem 1, and taking into account the properties in section 9.1.1 that all the coordinate sequences are equivalent by shift, we know that, the binary sequences resulted from the modulo 2 operation are entropy-lossless.

9.1.3 On the period and linear complexity of the coordinate sequences

From section 9.1.2 we know that, the binary sequences derived from the modulo 2 operation of the sequences over GF(p) generated by a primitive polynomial $f(x)$ are entropy-lossless. This means that every coordinate sequences are also entropy-lossless, which also means that they have the same period with the original sequences, i.e.

$$\mathrm{T}(\underline{a}_i)=\mathrm{T}(\underline{a})=p^{\deg(f(x))}-1,\ i=0,1,\ldots,30, \tag{4}$$

where T() denotes the period of sequences.

Therefore a chosen primitive polynomial $f(x)$ with suitable degree can guarantee that the period of each coordinate sequence is sufficiently large, for example, if $\deg(f(x))=16$, then the period of each coordinate sequence of sequences generated by $f(x)$ is $p^{16}-1\approx 2^{496}$.

From section 9.1.2 we know that the coordinate sequences are equivalent by shift, which means that these sequences have the same linear complexity. It is pointed out in ref [4] that the linear complexity of the coordinate sequences is

$$\mathrm{LC}(\underline{a}_i)=\tau\,(p^{16}-1)/(p-1),\ \ i=0,1,\ldots,30, \tag{5}$$

where τ is the linear complexity over GF(2) of the binary sequence $\{\zeta^t \bmod 2\}_{t\geq 0}$, where ζ is a primitive element of the finite field GF(p). In most cases, τ is about a half of p. Hence the linear complexity of the coordinate sequences LC($\underline{a}_i$) is about a half of their period.

9.1.4 On the advantage of bit coincidence between mod *p* and mod 2 sums

Let $J=\{1,2,\ldots,p\}$, $x_1,x_2,\ldots,x_k$ be k independent probabilistic variables over J complying with uniform probability distribution. Let their addition modulo p be as follows:

$$y= x_1+x_2+\ldots+x_k \bmod p,$$

where $y\in J$. Below we will discuss the bit coincidence between two probabilistic variables.

Let $u= x_1\oplus x_2\oplus \ldots\oplus x_k$. Write both y and u in their 2-adic representation as

$$y=y_0+y_1 2+y_2 2^2+\ldots+y_{30}2^{30},$$

$$u=u_0+u_1 2+u_2 2^2+\ldots+u_{30}2^{30},$$

where y_i and u_i will take value 0 or 1, i=0,1,…,30. Then the advantage of the bit coincidence between y_i and u_i are defined as

$$\Pr(y_i=u_i)-1/2.$$

It is also called the i-th bit coincidence between y and u, because y_i and u_i are the i-th coordinate of y and u respectively. Now we have the following conclusion:

Theorem 2 *For any given integer k, let* ϵ_i *be the i-th bit coincidence between y and u,* i=0,1,…,30*, i.e.,*

$$\epsilon_i=\Pr(y_i=u_i)-1/2.$$

Then we have

$$\epsilon_0=\epsilon_1=\ldots=\epsilon_{30}=\epsilon(k,p), \tag{6}$$

where $\epsilon(k,p)$ *is a constant that is only related to k and p.*

Proof: We know that $2a\equiv a<<<_{31}1 \bmod p$ for arbitrary $a\in J$, where $<<<_{31}$ denotes the 31-bit cyclic shift to the left. So the i^{th} bit of $\sum(x_j)$ is the same as the $(i+1)^{\text{th}}$ bit of $\sum(2x_j)$, and also the i^{th} bit of $\bigoplus(x_j)$ is the same as the $(i+1)^{\text{th}}$ bit of $\bigoplus(2x_j)$. Hence $\epsilon_i=\epsilon_{i+1}$ for all i. ■

Let

$$J_{k,t}=\{(x_1,x_2,\ldots,x_k)|\, x_1+x_2+\ldots+x_k\le t, 1\le x_i\le p, 1\le i\le k\}$$

and denote by $N_{k,t}$ the number of elements in the set $J_{k,t}$. Then we have:

1. When k=1:

$$N_{1,t}=\begin{cases}0 & \text{if } t<1\\ t & \text{if } 1\le t\le p.\\ p & \text{if } t<1\end{cases} \tag{7}$$

2. When k>1:

$$N_{k,t}=\sum_{1\le i\le p} N_{k-1,t-i}. \tag{8}$$

By equations (7) and (8), it is possible to compute the value of any $N_{k,t}$ in theory. Now we give a specific method to compute $\epsilon(k,p)$ using $N_{k,t}$.

Theorem 3 *For any integer k,*$1\le k<p$*, let* $\epsilon(k,p)$ *and* $N_{k,t}$ *be as defined above. Then we have*

$$\epsilon(k,p)=1/p^k\sum_{0\le i\le [k/2]}(N_{k,(2i+1)p-1}-N_{k,2ip-1})-1/2 \tag{9}$$

Proof: For any $x_i\in J$, $1\le i\le k$, write

$$\sum_{1\le i\le k} x_i=X_0+X_1p$$

for integers X_0, X_1, where $0\le X_0< p$. Then $u_0= X_0+X_1 \bmod 2$ and $y_0= X_0 \bmod 2$. So they are equal if and only if $X_1 \bmod 2=0$, that is, if and only if

$$2ip\le \sum_{1\le j\le k} x_j <(2i+1)p$$

for some integer $i\ge 0$, and the number of x_i (i=1,2,…,k) for which this holds is

$$N_{k,(2i+1)p-1}- N_{k,2ip-1}$$

However it is straightforward to show by induction that $N_{k,t} = p^k$ for $t\ge kp$, so we only need to consider $i\ge 0$ for which $2ip -1< kp$, that is, $2ip\le kp$, that is, $i\le [k/2]$. Hence the required result holds for $\epsilon(k,p)$. ■

By using equation (9), we can compute the advantage of bit coincidence for the cases when k=2,3,4,5,6,7 and 8, as shown in Table 9.1 (The values in Table 9.1 have been corrected by the evaluation report [33]).

Table 9.1: The values of $\epsilon(k,p)$, where k=2,3,4,5,6,7,8

k	2	3	4	5	6	7	8
$\epsilon(k,p)$	-2^{-32}	$-2^{-2.585}$	$-2^{-33.685}$	$2^{-3.907}$	$2^{-33.322}$	$-2^{-5.212}$	$-2^{-33.890}$

From Table 9.1 it is seen that, when k is even, then the bit coincidence is small, and when k is odd, then the bit coincidence can be significantly large, which is determined by the value of k, and has little to do with the value of p.

9.1.5 Fast implementations of the addition and multiplication over GF(p)

Since the modulus $p=2^{31}-1$ is a special prime, the addition and multiplication over GF(p) has a fast implementation shown as below.

Let $x, y\in$ GF(p). Write $x+y=z+c2^{31}$ for some integer $c\in \{0,1\}$. Then

$$x+y \bmod p = z+c .$$

This means that the modulo p addition does not have to do the modulo p operation. (We are using the ZUC convention that mod p integers are in the range 1 to p, rather than 0 to p-1.)

For arbitrary integer $0\le k\le 30$, we have

$$2^k x \bmod p=x<<<_{31}k$$

(which means x, treated as a 31-bit value, cyclically shifted towards the msb by k bit positions.) So when the Hamming weight of a multiplier a is small, for example, $a=2^i+2^j+2^k$, the multiplication ax over GF(p) can be calculated according to the following method:

$$ax \equiv (x<<<_{31}i)+(x<<<_{31}j)+ (x<<<_{31}k) \bmod p.$$

This means that the modulo p multiplication can be done by cyclic shifts and modulo p addition, hence the modulo p operation does not need to be actually implemented.

9.1.6 Choice of the primitive polynomial $f(x)$

In section 9.1.1-9.1.5, some properties of m-sequences over GF(p) are revealed, and it shows that we should choose some primitive polynomial $f(x)$ with even number of terms whose nonzero coefficients have as low as possible Hamming weights. More precisely, the choice of the primitive polynomial $f(x)$ should meet the following criteria:

1. The Hamming weight of each nonzero coefficient is as low as possible;
2. The sum of the Hamming weights of nonzero coefficients (ignoring the x^{16} term) is an even number;
3. The coefficient of the term with the second highest degree should be nonzero;
4. The difference of degrees of term with nonzero coefficients (ignoring the x^{16} term) should be pair-wise different;
5. The difference of positions of 1's of nonzero coefficients in their 2-adic representation should be pair-wisely different.

According to the above criteria, the following polynomial is chosen

$$f(x)=x^{16}-(2^{15}x^{15}+2^{17}x^{13}+2^{21}x^{10}+2^{20}x^{4}+(2^{8}+1)). \tag{10}$$

Since $f(x)$ is a primitive polynomial over GF(p), the LFSR will generate an m-sequence with period $p^{16}-1\approx 2^{496}$. By section 9.1.3, the GF(2) linear complexity of the coordinate sequences of those generated by $f(x)$ is about a half of their period, i.e.,2^{495}.

9.2 Design of the bit reorganization

The bit reorganization layer is the connection between the LFSR and the nonlinear function F, and it extracts 128 bits from the state of the LFSR and forms four 32-bit words which will be used by the nonlinear function F. The design of the bit-reorganization mainly refers to the following criteria:

1. Suitable for software implementation;
2. The four 32-bit words from the bit-reorganization have good randomness in the statistical sense;
3. The number of the overlapping bits of four 32-bit words in successive times is small.

Based on the above criteria, the bit-reorganization works as follows:

a) $X_0=s_{15H}\|\ s_{14L}$;

b) $X_1=s_{11L}\|\ s_{9H}$;

c) $X_2=s_{7L}\|\ s_{5H}$;

d) $X_3=s_{2L}\|\ s_{0H}$,

where X_i (i=0,1,2,3) are the output of the bit-reorganization, and the subscripts H and L of some cell s indicate the high 16 bits and low 16 bits of s respectively.

As for the bit reorganization constructed above, within two consecutive times, i.e, time t and time t+1, there is one bit in common among the inputs; in time t and t+2, there are 3 bits in common among the inputs; in time t and t+3, there are 17 bits in common among the inputs; in time t and t+4, there are 33 bits in common among the inputs. Common bits in other time intervals can be calculated easily.

Note: in the original ZUC design, the 128 bits were extracted from different stages of the shift register (i.e. in Figure 7.1, the eight arrows down from the LFSR to the BR block came from different "half words" in the register). It was noted that, at times t and t+1, there were 16 bits in common among the inputs. While no attack was identified to exploit this property, it was felt on reflection to be slightly undesirable; hence the arrow positions have been changed, reducing the overlap at times t and t+1 to a single bit.

In ZUC algorithm, since elements in GF(p) are defined to be within the set {1,2,…,p}, therefore during the feedback process of the 16-order LFSR, the value 0 should be replaced by p. It is noted that the LFSR will generate an m-sequence of period $T=p^{16}-1$. So in one period of such sequence, there will be no such state when all the register values are equal to p. It is easy to check the following conclusion:

Theorem 4 $\Pr(s_i=p)=(p^{15}-1)/(p^{16}-1)$, *and for any* $1\le a\le p-1$ *we have*

$$\Pr(s_i=a)=p^{15}/(p^{16}-1). \tag{11}$$

From Theorem 4 we can get the following conclusion immediately:

Theorem 5

$$\Pr(s_{iH}=0)=\Pr(s_{iL}=0)=(2^{15}-1)p^{15}/(p^{16}-1),$$

$$\Pr(s_{iH}=2^{16}-1)=\Pr(s_{iL}=2^{16}-1)=(2^{15}p^{15}-1)/(p^{16}-1),$$

$$\Pr(s_{iH}=a)=\Pr(s_{iL}=a)=2^{15}p^{15}/(p^{16}-1),$$

where a is a 16-bit binary string which is neither all-zero nor all-one.

Let X_0,X_1,X_2,X_3 be the four 32-bit words obtained by the bit reorganization process.

By theorem 5 and the well-known statistical properties of m-sequences, we have the following conclusions:

Corollary 1 *Let a and b be two arbitrary 16-bit binary strings which are neither all-zero nor all-one. Then*

$$\Pr(X_0=(a,b))= 2^{15}p^{15}/(p^{16}-1)\times 2^{15}p^{15}/(p^{16}-1) \tag{12}$$

Corollary 2 *Let a be an arbitrary 16-bit binary strings which are neither all-zero nor all-one,* **1** *is a 16-bit all-one binary string. Then we have*

$$\Pr(X_0=(a,\mathbf{1}))= \Pr(X_0=(\mathbf{1}, a))= 2^{15}p^{15}/(p^{16}-1)\times(2^{15}p^{15}-1)/(p^{16}-1) \tag{13}$$

Corollary 3 *Let a be an arbitrary 16-bit binary strings which are neither all-zero nor all-one,* **0** *is a 16-bit all-zero binary string. Then we have*

$$\Pr(X_0=(a,\mathbf{0}))= \Pr(X_0=(\mathbf{0}, a))= 2^{15}p^{15}/(p^{16}-1)\times(2^{15}-1)p^{15}/(p^{16}-1) \quad (14)$$

Corollary 4 *Let* **1** *be a 16-bit all-one string,* **0** *be a 16-bit all-zero string, then we have*

$$\Pr(X_0=(\mathbf{0},\mathbf{1}))= \Pr(X_0=(\mathbf{1},\mathbf{0}))= (2^{15}p^{15}-1)/(p^{16}-1)\times(2^{15}-1)p^{15}/(p^{16}-1) \quad (15)$$

Corollary 5 *Let* **0** *be a 32-bit all-zero string, then we have*

$$\Pr(X_0=\mathbf{0})= (2^{15}-1)p^{15}/(p^{16}-1)\times(2^{15}-1)p^{15}/(p^{16}-1) \quad (16)$$

Corollary 6 *Let* **1** *be a 32-bit all-one string, then we have*

$$\Pr(X_0=\mathbf{1})= (2^{15}p^{15}-1)/(p^{16}-1)\times(2^{15}p^{15}-1)/(p^{16}-1) \quad (18)$$

Corollary 7 *The random variables X_0,X_1,X_2,X_3 have the same probability distribution.*

Based on the above corollaries, we can roughly treat X_0,X_1,X_2,X_3 as having uniform probability distribution. With respect to their distinguishability with real uniform probability distribution, it can be reflected from the biasness of their Euclidean squares. Take X_0 as an example, we have

$$\varepsilon(X_0)=\sum_{a=0}^{2^{32}-1}[P(X_0=a)-\frac{1}{2^{32}}]^2 \approx 2^{-77} \quad (19)$$

The above means that it needs at least $N=\epsilon^{-1}\approx 2^{77}$ instances of the variable before it can be distinguished from a real random one.

9.3 Design of the nonlinear function *F*

The nonlinear function *F* is a compression function from 96 bits to 32 bits. Its design adopts some structures from block cipher design. Considering both security and performance requirements, the design of the nonlinear function *F* mainly refers to the following criteria:

1. Take 96 bits as input and output a 32-bit word;
2. The nonlinear function *F* should carry memories;
3. The nonlinear function *F* should use S-boxes in order to improve high nonlinearity and other cryptographic properties;
4. The nonlinear function *F* should use linear transformation with good diffusion;
5. The output sequence of the nonlinear function *F* should be balanced and have large period;
6. The nonlinear function *F* should be suitable for software and hardware implementations;
7. The cost of hardware implementations of the nonlinear function *F* should be low.

9.3.1 Design of the S-boxes S_0 and S_1

Two S-boxes are used in the nonlinear function *F* and named S_0 and S_1 respectively. Since the LFSR is shown to have good behaviour in resisting against algebraic attacks over GF(2), thus the algebraic immunity is not the highest priority for the S-boxes.

9.3.1.1 Design of S-box S_0

The design of the S-box S_0 mainly refers to the following three criteria:

1. The cost of its hardware implementation is low;
2. S_0 should have high nonlinearity;
3. S_0 should have low differential uniformity.

Based on the above consideration, the S-box S_0 is designed using a Feistel structure, see Figure 9.1.

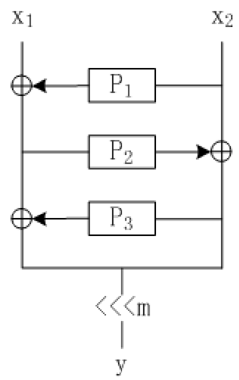

Figure 9.1: The structure of the S-box S_0

In Figure 9.1, both x_1 and x_2 are 4-bit strings, m=5, and P_1,P_2,P_3 are transformations over GF(16), which are defined as in Tables 9.2, 9.3 and 9.4.

Table 9.2: The transformation P_1

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	9	15	0	14	15	15	2	10	0	4	0	12	7	5	3	9

Table 9.3: The transformation P_2

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	8	13	6	5	7	0	12	4	11	1	14	10	15	3	9	2

Table 9.4: The transformation P_3

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	2	6	10	6	0	13	10	15	3	3	13	5	0	9	12	13

As for the S-box S_0, it is easy to check that its nonlinearity, differential uniformity, algebraic degree and algebraic immunity are 96, 8, 5 and 2 respectively. Due to its algebraic structure, no more than 11 linearly independent quadratic equations in the eight input and eight output bits can be established.

9.3.1.2 Design of S-box S_1

The design of the S-box S_1 mainly refers to the following two criteria:

1. The nonlinearity of the S-box S_1 is as high as possible;

2. The differential uniformity of the S-box S_1 is as low as possible.

According to the above criteria, the S-box S_1 is based on the inversion over the finite field GF(256) defined by the binary polynomial $x^8+x^7+x^3+x+1$, and composes one affine function after the inversion. More precisely, the S-box S_1 can be written as follows

$$S_1=\mathrm{M}x^{-1}+\mathrm{B} \tag{20}$$

where B=0x55 and M is a matrix of size 8×8 and defined as below:

$$\begin{pmatrix} 0&1&1&1&1&0&0&1\\ 1&0&1&1&1&1&0&0\\ 1&1&0&1&0&1&1&0\\ 1&1&1&0&0&0&1&1\\ 0&1&1&1&1&1&1&0\\ 1&0&1&1&0&1&1&1\\ 1&1&0&1&1&0&1&1\\ 1&1&1&0&1&1&0&1 \end{pmatrix}.$$

Since the S-box S_1 is affine equivalent to that of the advance encryption standard AES, thus S_1 has the same properties as that of AES, including nonlinearity, differential uniformity, algebraic degree and algebraic immunity. More precisely, its nonlinearity, differential uniformity, algebraic degree and algebraic immunity are 112, 4, 7 and 2 respectively. Due to its algebraic structure, no more than 39 linearly independent quadratic equations in the eight input and eight output bits can be established.

9.3.2 Design of the linear transformations L_1 and L_2

The goal of the linear transformations mainly provide good diffusion. Their designs mainly refer to the following two criteria:

1. L_1 and L_2 must have maximal differential and linear branch number.

2. L_1 and L_2 are suitable for software and hardware fast implementations.

As for the second criterion, one realization is to choose some functions composed of the operations exclusive-or $\oplus$ and the cyclic shift <<<, which are friendly to software and hardware implementations. Let GF(2)[x] be the polynomial ring over the binary field GF(2). Consider the quotient ring GF(2)[x]/(x^{32}+1). A 32-bit string $a=a_{31}a_{30}\ldots a_0$ is identified with an element $a(x)$ according to the bijection φ from GF(2)^{32} to GF(2)[x]/(x^{32}+1):

$$a=a_{31}a_{30}\cdots a_0 \quad\rightarrow\quad a(x)=\sum_{i=0}^{31}a_i x^i.$$

It is easy to check that the following holds:

$$\varphi(a\oplus b)=\varphi(a)+\varphi(b)$$

$$\varphi(a<<<k)=\varphi(a)x^k.$$

So the linear transformations composed of the exclusive-or and the cyclic shift can be viewed as polynomials over GF(2)[x]/(x^{32}+1), for example, the linear transformation L_1 used in ZUC can be written in the polynomial form:

$$L_1(x)=x^{24}+x^{18}+x^{10}+x^2+x.$$

Note that $x^{32}+1=(x+1)^{32}$, hence any polynomial $L(x)$ in GF(2)[x]/($x^{32}+1$) is invertible if and only if $L(1)\neq 0$, and if and only if $w(L)$ is odd, where $w(L)$ is defined as the number of monomials in $L(x)$ and is called the weight of $L(x)$.

For any given polynomial $L(x)$ in GF(2)[x]/($x^{32}+1$) with odd weight, we have $L^{32}(x)= L(x^{32})=L(1)=1$. So

$$L^{-1}(x)=L^{31}(x)=L(x^{16})L(x^8)L(x^4)L(x^2)L(x).$$

As for the first criterion, when a byte is viewed as a basic data unit, it is known that both maximum differential branch number and maximum linear branch number are 5. It is known that the weight of the linear transformation $L(x)$ with maximum differential branch number and maximum linear branch number must be greater than or equal to 4, i.e., $w(L)\geq 4$. Since a linear transformation $L(x)$ with $w(L)$=4 is not a permutation, therefore the weight of the linear transformation $L(x)$ with maximum differential branch number and maximum linear branch number is at least 5.

By testing all the possible polynomials $L(x)$ of weight 5, only two polynomials $L_1(x)$ and $L_2(x)$ are found to meet the above conditions:

$$L_1(x)=x^{24}+x^{18}+x^{10}+x^2+x,$$

$$L_2(x)=x^{30}+x^{22}+x^{14}+x^8+x,$$

which are the ones used in ZUC. The inversions of the above polynomials $L_1(x)$ and $L_2(x)$ are as below:

$$L_1^{-1}(x)=x^{30}+x^{24}+x^{22}+x^{18}+x^{16}+x^{14}+x^{12}+x^8+x^4+x^2+x,$$

$$L_2^{-1}(x)=x^{30}+x^{28}+x^{24}+x^{20}+x^{18}+x^{16}+x^{14}+x^{10}+x^8+x^2+\mathrm{x}.$$

The linear transformations $L_1(x)$ and $L_2(x)$ defined above both have maximum linear branch number and maximum differential branch number, both are equal to 5. It is found that the matrix representation of $L_1(x)$ and that of $L_2(x)$ over GF(2) happen to be transpose matrices of each other.

Note: The polynomial $L_1(x)$ is identical to a linear function in the block cipher SMS4.

10 128-EEA3 and 128-EIA3 constructions

10.1 128-EEA3 construction

The 128-EEA3 construction is identical to that of 128-EEA1 except that the underlying algorithms being different. The 128-EEA1 is based on the SNOW 3G algorithm, while 128-EEA3 is based on the ZUC algorithm. It is believed that the ZUC algorithm has different security resistance against known attacks, and hopefully have different resistance against unknown attacks.

10.2 128-EIA3 construction

Two possible principles for the design of 128-EIA3 are considered. One option was to re-use the SNOW 3G-based scheme for f9 developed for the first suite of 3GPP algorithm. However, the implementation of 128-EIA1 itself would take considerable amount of hardware resources except for SNOW 3G.

The alternative is to design a entirely different integrity algorithm. The 128-EIA3 algorithm just does so, and is designed to have a more efficient implementation. When ZUC is implemented for EEA3, the EIA3 takes very little extra hardware resources for implementation. The following provides its rationale behind 128-EIA3 construction.

There are three basic approaches to construct Message Authentication Codes (MAC): block-cipher-based construction, e.g. OMAC [5], hash-function-based construction e.g. HMAC [6], universal-hash-function-based construction, e.g. GMAC[7]. The construction of 128-EIA3 uses the third approach of MAC construction.

The use of universal hash functions as a tool for building a secure MAC has received much attention within the cryptographic community due to its nice security properties and efficient implementation. Theoretic researches show that [8][9], if the universal hash is an ϵ-AXU, then the probability of the successful attack (i.e. forging a valid MAC of some new message) is no greater than ϵ. Therefore, as long as ϵ is small enough and the one-time pad encryption algorithm is secure, then the universal-hash-function-based MAC is secure.

Definition 1 *Let $H: K\times D\rightarrow R$ be a function, where K,D and R are the key space, the message space and the digest space respectively. Then H is called ϵ-AXU (ϵ-Almost-Xor-Universal) if for any $x, x'\in D$, $x\neq x'$ and $y\in R$, we have* $\Pr(H(k,x)\oplus H(k,x')=y)\leq \epsilon$*, where k is randomly chosen from K.*

This document uses the following universal hash function H: $K\times\{0,1\}^{*}\rightarrow\{0,1\}^{32}$ as the message authentication code function:

$$H(k,x)=\begin{pmatrix} k_1 & k_2 & \cdots & k_{m+1} \\ k_2 & k_3 & \cdots & k_{m+2} \\ \vdots & \vdots & \ddots & \vdots \\ k_{32} & k_{33} & \cdots & k_{m+32} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ 1 \end{pmatrix}, \tag{21}$$

where m is the bit length of the message, and it needs (m+32) ZUC keystream bits in order to handle a m-bit message.

The universal hash function for 128-EIA3 is similar to the one based on Toeplitz matrix discussed in [10] and [11], with the exception that this new function can handle the variable-input-length messages. Then the following conclusion holds.

Theorem 6 *Let $H(k,x)$ be defined as above. Then $H(k,x)$ is 2^{-32}-AXU.*

Proof: Let $x=x_1x_2\ldots x_m$ and the $32\times(m+32)$ matrix

$$A_x = \begin{pmatrix} x_1 & \cdots & \cdots & x_m & 1 & 0 & \cdots & 0 \\ 0 & x_1 & \cdots & \cdots & x_m & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & x_1 & \cdots & \cdots & x_m & 1 \end{pmatrix}.$$

Then

$$H(k,x) = A_x \begin{pmatrix} k_1 \\ k_2 \\ \vdots \\ k_{m+32} \end{pmatrix}.$$

For any x, x', $x \neq x'$ and $y \in \{0,1\}^{32}$ consider the following cases:

1) If $|x|=|x'|$. Let $|x|=m$. Then

$$H(k,x) \oplus H(k,x') = (A_x + A_{x'}) \begin{pmatrix} k_1 \\ k_2 \\ \vdots \\ k_{m+32} \end{pmatrix}.$$

Since $A_x+A_{x'}$ is a full rank matrix, and the dimension of the solution space of is m, therefore

$\Pr(H(k,x) \oplus H(k,x')=y) \leq 2^m/2^{m+32}=2^{-32}$.

2) If $|x| \neq |x'|$. Let $m=|x|>|x'|=m'$, then

$$H(k,x') = (A_{x'}, O) \begin{pmatrix} k_1 \\ k_2 \\ \vdots \\ k_{m+32} \end{pmatrix},$$

where O is an $32\times(m-m')$ all-zero matrix. Similar to the case 1), it is known that $A_x+(A_{x'},O)$ is a full rank matrix, hence we also have

$$\Pr(H(k,x) \oplus H(k,x')=y) \leq 2^m/2^{m+32}=2^{-32}.$$

Combining the above two cases, the conclusion of theorem 6 holds. ■

11 Resistance against cryptanalytic attacks

11.1 Weak key attacks

First of all, it is worth to mention that the ZUC initialization process preserves the entropy of keys, i.e. there is a one-to-one mapping from the initial state of ZUC to the state after its initialization. It is observed by the evaluation report [33] that the map

$(K,IV)\rightarrow$algorithm state immediately after initialization is injective. To see this, it suffices to show that the clocking rule during the initialization stage is bijective. So suppose that the state $(s_0,s_1,\ldots,s_{15},R_1,R_2)$ clocks to $(s'_0,s'_1,\ldots,s'_{15},R'_1,R'_2)$ and that X_0, X_1 and X_2 are the words derived from the former state from during the process. Given $(s'_0,s'_1,\ldots,s'_{15},R'_1,R'_2)$, one can

- compute X_0, X_1 and X_2 from $s'_0,s'_1,\ldots,s'_{15}$;
- compute R_1 and R_2 from R'_1,R'_2, X_1 and X_2;
- compute the output W of the nonlinear function F from X_0, R_1 and R_2;
- compute $s_0,s_1,\ldots,s_{15}$ from $s'_0,s'_1,\ldots,s'_{15}$and W.

This shows that the clocking rule during ZUC initialization is bijective, and the map defined as above is injective. This property helps to see the likeliness of weak key attacks on ZUC.

For a given (K, IV)-pair, it is called a weak (K, IV)-pair if all registers s_i are equal to p after the ZUC initialization with K and IV, and the corresponding key K is called a weak key. As for the ZUC algorithm, it is very unlikely to exist such a weak state resulting from a (K, IV)-pair. This is because, when all register stages s_i are equal to p in an initial working state, we view the 64-bit states of R_1 and R_2 of FSM after ZUC's initialization as 64 binary variables, and track ZUC back to the initial state $(s_0,s_1,\ldots,s_{15},R_{1,0},R_{2,0})$. For any possible (R_1,R_2) pair, since ZUC's initialization preserves key entropy, there must be an unique initial state $(s_0,s_1,\ldots,s_{15},R_{1,0},R_{2,0})$ that results in this (R_1,R_2) pair. Hence in this case each bit of the initial state $(s_0,s_1,\ldots,s_{15},R_{1,0},R_{2,0})$ can be uniquely represented as a function of the 64 binary variables of R_1 and R_2. Note that at the very beginning of the initialization, $R_{1,0}=0$, $R_{2,0}=0$ and $s_i^{[8\text{-}23]}=D_i$ is a constant, where $s_i^{[8\text{-}23]}$ denotes all bits of register s_i from 8 to 23, $i=0,1,\ldots,15$. It follows that we can establish 64+16×15=304 equations about the 64 binary variables of R_1 and R_2. Since the update of ZUC's states is a nonlinear process, it is roughly viewed that those equations are pairwise statistically independent. Since the number of such equations is far more than that of the binary variables, in general such an equation system is very unlikely to have a solution. Though it is beyond personal computational capability to ensure that there does not exist such a weak state (the time complexity to verify this is about $O(2^{64})$), it is believed that it is very unlikely that such a weak key K exists.

11.2 Guess-and-Determine Attacks

The general idea of guess-and-determine attack in ref [12]-[14] is: by guessing part of the internal state of the target algorithm, combining with some known mathematical relations for the algorithm, to deduce other unknown internal states.

ZUC appears to have strong resistance against guess-and-determine attacks. It is seen that the ZUC algorithm has 16×31+2×32=560 bits of internal states. Assume that an attacker observes these internal states at some time interval, and he tries to guess r bits of these states to determine the rest 560-r states. Assume that the 560 bits of internal states have uniform probability distribution. Then the attacker needs at least $[(560+31-r)/32]$ key-words (the 32-bit words from the key stream) to establish algebraic equations, so that to possibly determine all the rest unknown bits. For a successful guess-and-determine attack, it has to satisfy that $r<128$. In this case the attacker needs at least 14 words of secret keys. This means that the algebraic equations established using these 14 words will involve the values of R_1 and R_2 in F

for at least 14 time intervals. Since the mechanism to update R_1 and R_2 is complicated and nonlinear, to deduce the values of the memory cells in the next time interval requires the current values of R_1 , R_2 and the inputs X_1 and X_2 which has to be guessed, and they have been 128 bits all together. If the attacker tries to guess the values of the memory cells R_1 and R_2 in different time intervals, then the number of bits to be guessed is apparently no less than 128. This discussion only considered 2 consecutive time intervals, and in practice, since the real number of key-words needed to perform an attack is far larger than 2, it means that the number of internal bits to be guessed is far larger than 128. In this sense the ZUC algorithm has good resistance against the guess-and-determine attack.

11.3 BDD Attacks (from evaluation report [32])

An attack based on Binary Decision trees has been proposed by Krause [15], and it has a very low data complexity. In view of the large state size (560 bits), a straightforward application of this attack has no chance to succeed. We have not been able to identify a way that would optimize the attack so that it would approach the 2^{128} limit. We confirm that a BDD attack is no better than exhaustive search.

11.4 Inversion Attacks (from evaluation report [32])

The applicability of the inversion [16] and generalized inversion [17] attacks on filter generators are evaluated. As the two outer values s_0 and s_{15} are both used to produce the key stream, these attacks do not apply to ZUC.

11.5 Linear Distinguishing Attacks

The basic idea of linear distinguishing attacks [18]-[20] is first to establish a probabilistic linear relationship between the output key stream and the input sequence from the LFSR. Then, using the fixed linear iteration formula of the LFSR sequence, to obtain a linear iteration formula for the key stream that holds with certain probability, and if the probability is large enough, then the output key stream can be distinguished from random sequences.

As for the ZUC algorithm, we first consider the linear approximation of 2-round F. At two consecutive times, t and t+1, we have

$$(X_{0,t} \oplus R_{1,t}) \boxplus R_{2,t} = W_t \tag{23}$$

$$(X_{0,t+1} \oplus R_{1,t+1}) \boxplus R_{2,t+1} = W_{t+1} \tag{24}$$

$$W_1 = R_{1,t} \boxplus X_{1,t} \tag{25}$$

$$W_2 = R_{2,t} \oplus X_{2,t} \tag{26}$$

$$R_{1,t+1} = S(L_1(W_{1L} \| W_{2H})) \tag{27}$$

$$R_{2,t+1} = S(L_2(W_{2L} \| W_{1H})) \tag{28}$$

In the above, the nonlinear function F includes the S-boxes and modulo 2^{32} addition $\boxplus$. One method to solve a system of nonlinear equations is to introduce new intermediate variables $R_{1,t}$, $R_{2,t}$, $R_{1,t+1}$, $R_{2,t+1}$, W_1 and W_2. Now we estimate the linear approximation of equations (23), (24), (25), (27) and (28), in which there are 3 modulo 2^{32} additions, 2 of S-box operations, see Figure 11.1.

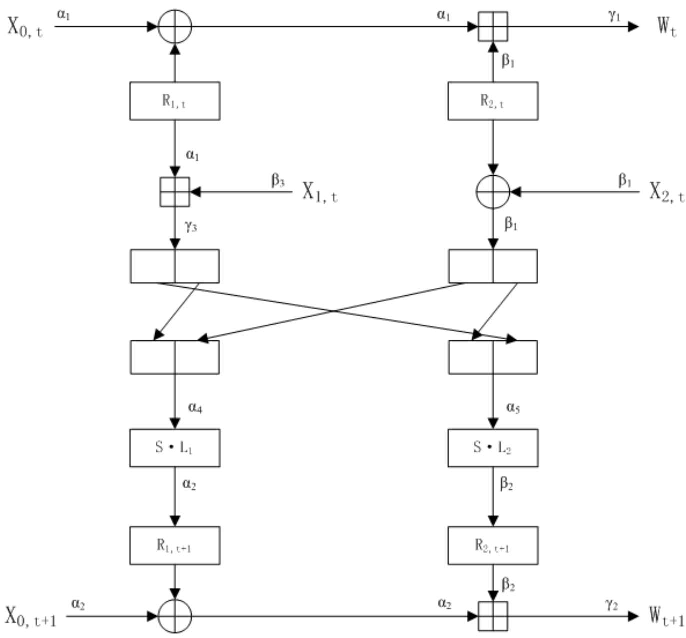

Figure 11.1: The linear approximation of two-round F

From Equation (23) we have:

$$\alpha_1\cdot(X_{0,t}\oplus R_{1,t}) \oplus \beta_1\cdot R_{2,t}=\gamma_1\cdot W_t.$$

From Equation (24) we have:

$$\alpha_2\cdot(X_{0,t+1}\oplus R_{1,t+1}) \oplus \beta_2\cdot R_{2,t+1}=\gamma_2\cdot W_{t+1}.$$

From Equation (25) we have:

$$\gamma_3\cdot W_1=\alpha_1\cdot R_{1,t} \oplus \beta_3\cdot X_{1,t}$$

From Equation (27) we have:

$$\alpha_2\cdot R_{1,t+1}=\alpha_4\cdot(W_{1L}\| W_{2H})$$

From Equation (28) we have:

$$\beta_2\cdot R_{2,t+1}=\alpha_5\cdot W_{2L}\| W_{1H}$$

From the above process of linear approximation, it can be seen that, the intermediate variables $R_{1,t}$, $R_{2,t}$, $R_{1,t+1}$, $R_{2,t+1}$, W_1 and W_2 can be eliminated if and only if $\beta_1=\alpha_{4L}\| \alpha_{5H}$ and $\gamma_3=\alpha_{5L}\| \alpha_{4H}$ hold simultaneously. So we have the relation between the output variables W_t, W_{t+1} and the input variables $X_{0,t}$, $X_{0,t+1}$, $X_{1,t}$ and $X_{2,t}$:

$$\alpha_1\cdot X_{0,t}\oplus \alpha_2\cdot X_{0,t+1}\oplus \beta_3\cdot X_{1,t} \oplus \beta_1\cdot X_{2,t} =\gamma_1\cdot W_t\oplus \gamma_2\cdot W_{t+1} \quad (29)$$

In our experiments, we searched the case when there is only one active S-box, and the best advantage from the linear approximations of Equation (29) that can be found is $2^{-22.6}$, where the values of $\alpha_1,\alpha_2,\beta_1,\beta_3,\gamma_1$ and γ_2 are shown in Table 11.1 (The values in Table 11.1 are taken from the evaluation report [33]).

Table 11.1: The values of $\alpha_1,\alpha_2,\beta_1,\beta_3,\gamma_1$ and γ_2 from the best available linear approximation

Advantage	α_1	α_2	β_1	β_3	γ_1	γ_2
$2^{-22.6}$	01040405	00300000	01010405	01860607	01040607	00200000

In the following we will construct a (probabilistic) linear iteration relationship of key stream $\{Z_t\}_{t\geq 0}$ using the above linear approximation from two aspects:

From sections 9.1.2 and 9.1.3 it can be seen that, the coordinate sequences of an LFSR sequence $\underline{a}$ over the prime field GF($_p$) have large distance m($\approx 2^{493}$) for shift-equivalence and large linear complexity d ($\approx 2^{495}$). These linear relationships can be used to eliminate the LFSR state variables in the linear approximation (29) to get a linear approximation of $\{Z_t\}_{t\geq 0}$, which requires at least 2^{493} bits of key stream, which is practically intractable.

Another approach is to find a multiple of $f(x)$ with low degree such that the number of its nonzero terms is as small as possible and the weight of its all nonzero coefficients is one, and then establish linear approximations of the LFSR by such a multiple, finally combine those linear approximations with those of 2-round F, i.e., equation (29), and establish a distinguisher only depending on the keystream. In the worst case a trinomial multiple with low degree is found (since the degree of any binomial multiple of $f(x)$ must be a multiple of(p^{16}-1)/(p-1), thus its degree is at least $O(2^{465})$ and too large). Under the assumption that the LFSR is linear over GF(2), when the best linear approximation (29) is used to construct a distinguisher, the advantage is about$2^{3-1}\epsilon_F^3\approx 2^{-66.4}$. The number of keystream bits used by the above distinguisher is about $2^{132.8}$. If the effect of linear hulls is considered, the required number of keystream bits may be reduced somewhat, but the complexity of the attack will still be greater than the exhaustive key search. However, it should be pointed out that in the above worst case we ignored the difference between mod p and mod 2, which in fact is quite significant. Hence it seems extremely unlikely that a linear cryptanalysis with practical complexity is feasible.

11.6 Algebraic Attacks

The general idea of algebraic attacks [21]-[24] is, to treat the whole encryption algorithm as an over-defined system of algebraic equations. Then the initial key or all the internal states at certain time interval can be recovered by solving this over-defined system of multivariate algebraic equations using some traditional methods for solving systems of multivariate-equations, say linearization, re-linearization, Gröbner basis, XL method, F4 and F5 methods.

In the ZUC algorithm, since the feedback for the LFSR sequences determined by the characteristic polynomial (10) is nonlinear at bit level, the algebraic equations established using (10) at bit level will have algebraic degree at least 2 over GF(2). In the following, we show how a system of algebraic equations can be established based on the modulo p addition of 2 integers.

Let $x,y,z\in J$, $z=x+y \bmod p$, $x=x_{30}x_{29}\ldots x_1x_0$, $y=y_{30}y_{29}\ldots y_1y_0$, $z=z_{30}z_{29}\ldots z_1z_0$. Let c_{i+1} denote the carry when the i-th bits are summed, and let $c_0=c_{31}$. Then we have

$$z_i=x_i\oplus y_i\oplus c_i$$

$$c_{i+1}= x_iy_i\oplus (x_i\oplus y_i)\, c_i$$

Furthermore we can get

$$x_{i+1}\oplus y_{i+1}\oplus z_{i+1}= x_iy_i\oplus (x_i\oplus y_i)(x_i\oplus y_i\oplus z_i)$$

The above are in fact some algebraic equations of degree 2 for the corresponding bits of x,y,z. Similarly, we can establish algebraic equations for the case of modulo p additions of k variables. In general, when k is small, the algebraic equations for the k inputs $x_1,x_2,\ldots,x_k$ and the output y will have algebraic degree k, here $y=x_1+x_2+\ldots+x_k \bmod p$. For the ZUC algorithm, since the feedback of the LFSR involves the modulo p addition of 6 terms, the algebraic equations for the states $s_{15},s_{13},s_{10},s_4,s_0$ and the feedback output s_{16} have algebraic degree 6.

In section 9.3.1, we discussed some properties of the S-boxes. The 2 S-boxes in the nonlinear function F are algebraic immune of order 2, hence quadratic equations can be established for their inputs and outputs. Moreover, since the modulo 2^{32} addition $\boxplus$ will remain the linear relationship on the least significant bits, the other bits satisfy quadratic equations. If an intermediate variable W_1 is introduced, then the whole nonlinear function F can be represented by a system of algebraic equations of algebraic degree no more than 2 for the input variables X_0,X_1,X_2, the output W, the memory cells R_1,R_2 and the intermediate variable W_1. If no intermediate variables are introduced, then an equivalent system of algebraic equations for the input variables X_0,X_1,X_2, the output variable W and the memory cells R_1,R_2 can be established, but the algebraic equations are of algebraic degree 3.

Since the ZUC algorithm has a total number of 31×16+2×32=560 internal state bits, to an observer, it is reasonable to assume that these bits are in random distribution at a certain time interval. Note that the algorithms outputs a 32-bit word at every time pulse, hence in order to establish a over-defined system of algebraic equations that have a unique solution, it needs at least 18=[(560+31)/32] key-words. In the following we discuss a few ways to establish algebraic equations when 18 key-words are used:

1. To eliminate all the intermediate states introduced during the LFSR feedback

From the above discussion we know that, the LFSR feedback can induce some algebraic equations of degree 6 for s_{16} and $s_{15},s_{13},s_{10},s_4,s_0$. In addition, since X_1 will be affected by s_{16} after 5 time pulses, and the computation of X_1 using F will need 2 nonlinear operations: $\boxplus$ and the S-boxes, which can be established by algebraic equations of degree no more than 2 respectively. The process to eliminate s_{16} from the current system of equations will result in the increase of the algebraic degree of the whole system, more precisely it will become a system of algebraic equations of algebraic degree 8. In this case the total number of variables involved is 1648=(16×31+18×2×32).

2. To keep all the intermediate states introduced in the LFSR feedback as new variables

When all the intermediate states for the LFSR feedback computation are treated as new variables, since the algebraic equations for the LFSR states that can be established are of degree 6, hence the degree of the whole system of the algebraic equations are of degree 6. Under the condition that 18 key-words in total are used, then the total number of variables in such a system of algebraic equations is 2175(=16×31+17×31+18×2×32).

3. To establish quadratic equations using the intermediate variables

We now consider to introduce new variables for the intermediate states to establish a system of algebraic equations of degree 2. For the computation of the LFSR feedback, we consider the following process:

$$y_1=(1+2^8)s_0 \bmod p,$$

$$y_2=2^{20}s_4+y_1 \bmod p,$$

$$y_3=2^{21}s_{10}+y_2 \bmod p,$$

$$y_4=2^{17}s_{13}+y_3 \bmod p,$$

$$s_{16}=2^{15}s_{15}+y_4 \bmod p.$$

Every equation of the above ones can induce 93 linearly independent quadratic equations, hence for the LFSR feedback computation, if we introduce intermediate variables y_1,y_2,y_3,y_4, then we can establish 465 linearly independent quadratic equations for all the states and all the intermediate variables. In addition, if we introduce variable W_1 in the nonlinear function F, then it is also possible to establish a system of quadratic equations for the whole nonlinear function F with respect to the input variables X_0,X_1,X_2, the output variable W, the memory cells R_1,R_2 and the intermediate variable W_1. Where the ⊞ operation can establish 93 linearly independent quadratic equations involving 93 variables, and the S-boxes S_0 and S_1 can establish 11 and 39 linearly independent quadratic equations respectively. Therefore the whole ZUC algorithm can form a system of quadratic equations. When an attacker obtains 18 key-words, the total number of variables in the system is

$$16\times31+2\times32+1+17(5\times31+3\times32-2)=4792$$

and the number of linearly independent quadratic equations is

$$93+17(93\times5+2\times93+39+11)=12010.$$

The precise complexity of solving the above three algebraic systems is unknown, but using Proposition 6 of [25], it follows that an XL-like approach on this system is very unlikely to be faster than an exhaustive search for the key.

Note that the above discussion assumes that 18 key-words are available, this is the minimum number of key-words that can establish an over-defined system of algebraic equations. However in practice, when the linearization method is used to solve an algebraic equation system, the number of independent equations that can be established is likely to be far smaller than that of the variables, and hence the number of key-words needed will have to be larger than 18. This increased number of key-words will inevitably increase the number of intermediate variables, and the practical complexity is far higher than the above theoretical result.

11.7 Chosen IV Attacks

For a good stream cipher, after the initialization, each bit of the IV/Key should contribute to every bit of all the states of the cipher, and any differential of the IV/Key will result in an almost-uniform and unpredictable differential of the internal states. This means that any differential of the IV/Key will result in almost-uniform and indistinguishable differential of the output key streams.

The chosen-IV/Key attack is one of the two main attacks targeting at the initialization state of stream ciphers. Compared with the frequency of changing the Key, the change of IV is more frequent, and the IV is known to the public, so the chosen IV attack is more threatening. On the other hand, in the initialization of ZUC, the importance of the initial key and that of IV seems to be the same. So we will mainly consider the resistance of ZUC algorithm against the chosen-IV attack.

The initialization process of the ZUC algorithm has 32 iterations. We will consider the influence of any changes that the IV has to each of the iterations, i.e., when a specific differential of the IV is chosen, and when the Key is fixed, how the differential varies in each round of iteration.

After some computing test and analysis, we found that when the 9-th byte of ΔIV is chosen to be non-zero (i.e., ΔIV[9]≠ 0), and the remaining 15 bytes of ΔIV are all 0, then the differential propagates the slowest. Hence in the following we give a more detailed analysis on this case.

Let ΔIV[9]=a≠ 0, then the differentials of the 16 registers of the LFSR before the 1 round iteration are:

$$(s_{15},s_{14},s_{13},s_{12},s_{11},s_{10},s_9,s_8,s_7,s_6,s_5,s_4,s_3,s_2,s_1,s_0)=(0,0,0,0,0,0,a,0,0,0,0,0,0,0,0,0).$$

Then 32 rounds of iteration will take place. Table 3.1 gives the differentials of the 16 registers of the LFSR after 1-32 rounds of iteration, where the i-th row in the table is the 16 differentials of the registers after i rounds of iteration(i=1,2,…,32). This is the best differential path that we could find. It is easy to see that, after 18 rounds of iteration, we have s_0=0. This means that the LFSR state is not random even after 18 rounds of iteration. By the cryptographic properties of the S-boxes and the L-transforms in the nonlinear function F, we know that b and c are not very random. However states marked with asterisk (*) in the low left corner of Table 11.2 indeed have good randomness (where the states on the left have better randomness than those on the right). Hence we believe that, after 32 rounds of iteration, the differentials of the 16 registers in the LFSR will be fairly even and unpredictable.

Table 11.2: The differentials of the LFSR in each round of iteration

round	s_{15}	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0
0	0	0	0	0	0	0	a	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	a	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	a	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	a	0	0	0	0	0	0
4	b	0	0	0	0	0	0	0	0	0	a	0	0	0	0	0
5	c	b	0	0	0	0	0	0	0	0	0	a	0	0	0	0
6	*	c	b	0	0	0	0	0	0	0	0	0	a	0	0	0
7	*	*	c	b	0	0	0	0	0	0	0	0	0	a	0	0
8	*	*	*	c	b	0	0	0	0	0	0	0	0	0	a	0
9	*	*	*	*	c	b	0	0	0	0	0	0	0	0	0	a

10	*	*	*	*	*	c	b	0	0	0	0	0	0	0	0	0
11	*	*	*	*	*	*	c	b	0	0	0	0	0	0	0	0
12	*	*	*	*	*	*	*	c	b	0	0	0	0	0	0	0
13	*	*	*	*	*	*	*	*	c	b	0	0	0	0	0	0
14	*	*	*	*	*	*	*	*	*	c	b	0	0	0	0	0
15	*	*	*	*	*	*	*	*	*	*	c	b	0	0	0	0
16	*	*	*	*	*	*	*	*	*	*	*	c	b	0	0	0
17	*	*	*	*	*	*	*	*	*	*	*	*	c	b	0	0
18	*	*	*	*	*	*	*	*	*	*	*	*	*	c	b	0
19	*	*	*	*	*	*	*	*	*	*	*	*	*	*	c	b
20	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	c
21	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

If we look at the differential variation of R_1 and R_2, it is noted that before the first iteration, R_1 and R_2 are both set to be zero. A simple deduction will see that after the 1-2 rounds of iteration, the differentials of R_1 and R_2 are all zero. However after the 3-th round of iteration, although the differential of R_1 is still zero, that of R_2 is nonzero. Since the update of R_1 and R_2 involves S-boxes and the L-transforms, after the 4-th round of iteration, the differentials of R_1 and R_2 have better randomness with the increase of the number of rounds of iteration. From this behaviour it is believed that, after 32 rounds of iteration, the differentials of R_1 and R_2 will be fairly random and unpredictable. The above analysis shows that any differential of the IV will result in very random differential between the output key streams.

As pointed out in [32], the chosen-IV attack can be extended into more than 18 rounds. But it seems that the maximum number of rounds one can cover with for chosen-IV differentials is 20. It can't be ruled out that there exist differentials covering more rounds of the initialization phase with a positive probability (but less than one), but this requires incorporation of differences through the function F over several rounds. We believe that such an approach will not lead to differentials of probability higher than 2^{-128} for the 32 rounds of the initialization procedure.

11.8 Time-Memory-Data Trade-Off Attacks

11.8.1 Attacks against the function mapping internal state to keystream

The stream cipher comfortably resists traditional Time-Memory Tradeoff (TMTO) attacks against the internal state, because the state size is well over 2.5 times the secret key length (see [27]).

11.8.2 Attacks against the function mapping Key/IV pair to keystream

However, both external evaluation teams noted that the algorithm parameters used in LTE lend themselves to the alternative type of TMTO attack proposed by Hong and Sarkar in [29] and [30]. This is not a criticism of the ZUC cipher or the 128-EEA3 design — the comments apply equally to 128-EEA1 and 128-EEA2.

Apart from the 128-bit secret key, the other inputs to the encryption algorithm are a 32-bit COUNT, a 5-bit BEARER and a 1-bit DIRECTION. Collectively these act as the non-secret "Initialisation Vector" (IV) for the cipher. Of these, only the 5-bit BEARER is at all unpredictable by an attacker intercepting encrypted messages – the COUNT is reset to 0 for each new secret key, and DIRECTION is uplink or downlink.

Hong and Sarkar describe generic Time-Memory Tradeoff (TMTO) attacks against stream ciphers, with the following form:

- the attacker precomputes keystreams for many Key/IV pairs;
- the attacker intercepts a large number of keystreams generated using many different secret keys[1];
- the attacker's goal is to recover the secret key used for <u>any one</u> of those keystreams.

We measure the complexity of an attack by the one-off pre-computation time 2^p, the "online" computing time 2^t for the actual attack instance, the computer memory requirement 2^m, and the number 2^d of keystreams that the attacker has intercepted (from different keys – we assume that the first keystream sequence, for COUNT=0, is intercepted). Let the secret key length be $k = 128$, and let the number of unpredictable bits of IV be $v = 5$. Attacks are possible as follows:

- **[Babbage-Golic tradeoff]** By computing and storing a table of 2^m key/IV pairs and the keystreams they generate, the attacker can probably determine the key and used to generate one of the intercepted keystreams, if $m+d \geq k+v$. For instance, a table of 2^{93} (key/IV, keystream) values will be enough if 2^{40} keystreams are intercepted. The table can be computed once, and then used for repeated attack attempts on different sets of intercepted keystreams.
- **[Biryukov-Shamir tradeoff]** By precomputing 2^p key/IV pairs, and storing a table of size 2^m, an attack is possible with online time complexity t as long as $t \geq 2d$, $p+d \geq k+v$, and $m+t \geq k+v$. For example, with 2^{40} keystreams, an attack is possible with one-off precomputation time 2^{93}, and online time complexity 2^{80}, but computer memory requirement only 2^{53}.

Increasing the number of unpredictable IV bits increases the complexities of these attacks.

These observations were fed back to 3GPP SA3. Note that the attack model is somewhat questionable: it assumes that the attacker "wins" if she can recover the key for just one of the many keystreams she has intercepted. The complexity of recovering the key for any *particular* keystream is not improved significantly by these attacks, compared to generic key search techniques.

Nevertheless, there would be some advantages in increasing the number of unpredictable IV bits, if a source of them were readily available. All of the 128-EEA1/2/3 algorithms could easily be adapted to accommodate longer IVs.

12 Conclusion of the evaluation

The evaluation assessed many different classes of attacks and concluded that none were likely to succeed. A few components of the algorithm were identified for which the initially stated design rationale was not completely clear; further discussion with the designers has addressed those points, and the fuller explanations are now included in this design and evaluation report.

One stated objective for the design was that the new algorithms be substantially different from the first and second LTE algorithm sets, in such a way that an attack on any one algorithm set would be unlikely to lead to an attack on either of the others. In SAGE's view this objective

---

[1] It's optimistic to assume that an attacker can intercept keystreams, rather than just encrypted versions of unknown messages; but this is the standard attack assumption when we are trying to design a cipher that will be secure irrespective of what's being encrypted.

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”;

- and in any case the raison d’être of this new algorithm set is very different from that of the first two, so the objective is considerably less important than making the first and second algorithm sets different from each other.

SAGE therefore does not consider this a barrier to acceptance of the new algorithms. Indeed, both of the paid evaluation teams noted that the ZUC design inherits some strong security properties from SNOW 3G, while adding further protection against as yet unknown attacks.

Overall, taking into account all the feedback from the two paid evaluation teams, the SAGE task force concluded that the new algorithms are fit for purpose. The security margin appears to be high, and the design rationale clear. The SAGE task force has no objection to 128-EEA3 and 128-EIA3 being included in the standards.

Annex A - External references

[4] A. H. Chan and R. Games, On the linear span of binary sequences from finite geometries, q odd, in Advances in Cryptology: Proceedings of Crypto 1986, Springer-Verlag, Berlin, 1987, 405-417.

[5] Iwata, T. & Kurosawa, K. OMAC: One-Key CBC MAC. *Fast Software Encryption 2003, Springer-Verlag,* 2003*, LNCS 2887*, 129-153.

[6] Bellare, M. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. *Advances in Cryptology - CRYPTO 2006, Springer-Verlag,* 2006*, LNCS 4117*, 602-619.

[7] NIST Special Publication 800-38D. Recommendation for Block. Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. November, 2007.

[8] Carter, J. L., Wegman, M. N. Universal Classes of Hash Functions. *Journal of Computer and System Sciences,* 1979*, 18*, 143-154.

[9] Shoup, V. On Fast and Provably Secure Message Authentication Based on Universal Hashing *Advances in Cryptology -- CRYPTO 1996, Springer-Verlag,* 1996*, LNCS 1109*, 313-328.

[10] Krawczyk, H. LFSR-based hashing and authentication. Crypto'94. *Springer-Verlag,* 1994*, LNCS 839*, 313-328.

[11] Krawczyk, H. New hash functions for message authentication. Crypto'95. *Springer-Verlag,* 1995*, LNCS*, 129-139.

[12] P. Hawkes and G. Rose, Guess and determine attacks on SNOW, In Selected Area of Cryptography--SAC2002, LNCS 2595, pp.37-46, 2002.

[13] C.D. Canniere, Guess and determine attacks on SNOW, NESSIE Public Document, NES/DOC/KUL/WP5/011/a, 2001.

[14] H. Ahmadi, T. Eghlidos and S. Khazaei, Improved guess and determine Attack on SOSEMANUK, Tehran, Iran, 2006, http://www.ecrypt.eu.org/stream/sosemanukp3.html.

[15] M. Krause. BDD-Based Cryptanalysis of Keystream Generators. In L. Knudsen, editor, Advances in Cryptology - EUROCRYPT 2002, volume 2332 of Lecture Notes in Computer Science, pages 222–237. Springer-Verlag, 2002.

[16] J. Golic, A. Clark and E. Dawson. Inversion Attack and Branching. In J. Pieprzyk et al., editors, ACISP 1999, volume 1587 of Lecture Notes in Computer Science, pages 88–102. Springer-Verlag, 1999.

[17] J. Golic, A. Clark and E. Dawson. Generalized Inversion Attack on Nonlinear Filter Generators. IEEE Transactions on Computers, Vol. 49, No. 10, pages 1100–1109, October 2000.

[18] Dai Watanabe, Alex Biryukov,Christophe De Cannière. A distinguishing attack of SNOW 2.0 with linear masking method. In M.Matsui and R.Zuccherato eds. Selected

Areas in Cryptography 2003. Lecture Notes in Computer Science 3006. pp.222-233,2004. Springer-Verlag Berlin Heidelberg 2004.

[19] Don Coppersmith,Shai Halevi,Charanjit Jutla. Cryptanalysis of stream ciphers with linear masking. In M.Yung ed. CRYPTO 2002, LNCS 2442, pp.515-532,2002. Springer-Verlag Berlin Heidelberg 2002.

[20] Kaisa Nyberg,Johan Wallén. Improved Linear Distinguishers for SNOW 2.0. M.J.B. Robshaw ed. Fast Software Encryption 2006, Lecture Notes in Computer Science 4047, pp.144-162,2006. International association for cryptologic research 2006.

[21] Nicolas T. Courtois and Wili. Meier, Algebraic Attacks on Stream Ciphers with Linear Feedback, In Advances in Cryptology-EUROCRYPT 2003, LNCS 2656, pp. 346-359, Springer-Verlag, 2003.

[22] Sondre Ronjom, Tor Helleseth, Attacking the Filter Generator over GF(2m), in Workshop Record of SASC 2007: The State of the Art of Stream Ciphers,eSTREAM report 2007/011 (2007).

[23] WilliMeier, Enes Pasalic, and Claude Carlet, Algebraic attacks and decomposition of Boolean functions, In Advances in Cryptology-EUROCRYPT, 2004, LNCS 3027, Berlin: Springer-Verlag, 2004, pp. 474 - 491.

[24] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations.In Bart Preneel, editor, Advances in Cryptology - EUROCRYPT 2000, volume 1807of Lecture Notes in Computer Science, pages 392–407. Springer, 2000.

[25] Claus Diem. The XL-Algorithm and a Conjecture from Commutative Algebra. In Pil Joong Lee, editor, Advances in Cryptology - ASIACRYPT 2004, volume 3329 of Lecture Notes in Computer Science, pages 323–337. Springer, 2004.

[26] S. H. Babbage. Improved exhaustive search attacks on stream ciphers. In IEE European Convention on Security and Detection, volume 408, pages 161-165, 1995.

[27] S. H. Babbage and M. W. Dodd. The MICKEY Stream Ciphers. In New Stream Cipher Designs, volume 4986 of Lecture Notes in Computer Science, pages 191-209. Springer, 2008.

[28] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Advances in Cryptology ASIACRYPT 2000, volume 1976 of Lecture Notes in Computer Science, pages 1-13. Springer, 2000.

[29] J. Hong and P. Sarkar. New Applications of Time Memory Data Tradeoffs. In Advances in Cryptology | ASIACRYPT 2005, volume 3788 of Lecture Notes in Computer Science, pages 353-372. Springer, 2005.

[30] J. Hong and P. Sarkar. Rediscovery of Time Memory Tradeoffs. Cryptology ePrint Archive, Report 2005/090, 2005. http://eprint.iacr.org/.

[31] O. Dunkelman and N. Keller. Treatment of the initial value in Time-Memory-Data Tradeoff Attacks on Stream Ciphers. Information Processing Letters, 107(5):133-137, 2008.

[32] L.R. Knudsen, B.Preneel, V.Rijmen, Evaluation of ZUC, ABT Crypto, Version 1.1, 9 May 2010.

[33] C.Cid, S.Murphy, F.Piper, M.Dodd, ZUC Algorithm Evaluation Report, Codes & Ciphers Ltd., 7 May 2010.