ETSI/SAGE **Version: 1.4**
Specification **Date: 30th July 2010**

Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 1: 128-EEA3 and 128-EIA3 Specification

Document History		
V1.0	**18th June 2010**	**Publication**
1.2	**26-07-2010**	**Improvements to C code**
1.3	**27-07-2010**	**Minor corrections to C code**
1.4	**30-07-2010**	**Corrected preface**

Blank Page

PREFACE

This specification has been prepared by the 3GPP Task Force, and gives a detailed specification of the 3GPP confidentiality algorithm ***128-EEA3*** and the 3GPP integrity algorithm ***128-EIA3***. This document is the first of three, which between them form the entire specification of the 3GPP Confidentiality and Integrity Algorithms:

- Specification of the 3GPP Confidentiality and Integrity Algorithms ***128-EEA3*** & ***128-EIA3***.
  Document 1: ***128-EEA3*** and ***128-EIA3*** Specifications.

- Specification of the 3GPP Confidentiality and Integrity Algorithms ***128-EEA3*** & ***128-EIA3***.
  Document 2: **ZUC** Specification.

- Specification of the 3GPP Confidentiality and Integrity Algorithms ***128-EEA3*** & ***128-EIA3***.
  Document 3: Implementors' Test Data.

The normative part of the specification of the ***128-EEA3*** (confidentiality) and ***128-EIA3*** (integrity) algorithms is in the main body of this document. The annexes to this document are purely informative, and contain implementation program listings of the cryptographic algorithms specified in the main body of this document, written in the programming language C.

The normative section of the specification of the stream cipher (**ZUC**) on which ***128-EEA3*** and ***128-EIA3*** are based is in the main body of Document 2.

TABLE OF CONTENTS

1 OUTLINE OF THE NORMATIVE PART 7

2 INTRODUCTORY INFORMATION 7
2.1 Introduction 7
2.2 Notation 7

3 CONFIDENTIALITY ALGORITHM *128-EEA3* 9
3.1 Introduction 9
3.2 Inputs and Outputs 9
3.3 Initialisation 9
3.4 Keystream Generation 10
3.5 Encryption/Decryption 10

4 INTEGRITY ALGORITHM *128-EIA3* 11
4.1 Introduction 11
4.2 Inputs and Outputs 11
4.3 Initialisation 11
4.4 Generating the key stream 11
4.5 Compute the MAC 12

ANNEX 1 A C implementation of *128-EEA3* 14

ANNEX 2 A C implementation of *128-EIA3* 15

REFERENCES

[1] Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 1: f8 and f9 specifications; (3GPP TS35.201 Release 6).

[2] 3GPP System Architecture Evolution (SAE); Security architecture; (3GPP TS33.401 Release 9).

[3] Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification.

NORMATIVE SECTION

This part of the document contains the normative specification of the Confidentiality and Integrity algorithms.

1 OUTLINE OF THE NORMATIVE PART

Section 2 introduces the algorithm and describes the notation used in the subsequent sections.
Section 3 specifies the confidentiality algorithm ***128-EEA3.***
Section 4 specifies the integrity algorithm ***128-EIA3.***

2 INTRODUCTORY INFORMATION

2.1 Introduction

Within the security architecture of the LTE system there are standardized algorithms for confidentiality and integrity. Two set of algorithms 128-EEA1/128-EIA1 and 128-EEA2/128-EIA2 have already been specified [1-2]. In this document the third set of these algorithms (***128-EEA3***/***128-EIA3***) based on ZUC [3] are proposed.
The confidentiality algorithm ***128-EEA3*** is a stream cipher that is used to encrypt/decrypt blocks of data under a confidentiality key **CK**. The block of data may be between 1 and 20000 bits long. The algorithm uses **ZUC** as a keystream generator.
The integrity algorithm ***128-EIA3*** computes a 32-bit MAC (Message Authentication Code) of a given input message using an integrity key **IK**. The approach adopted uses **ZUC.**

2.2 Notation

2.2.1 Radix

In this document, integers are represented as decimal numbers unless specified otherwise. We use the prefix "0x" to indicate hexadecimal numbers and the subscript "2" to indicate a number in binary representation.
Example 1. Integer *a* can be written in different representations:

a = 1234567890 // decimal representation
= 0x499602D2 // hexadecimal representation
= $1001001100101100000001011010010_2$ //binary representation

2.2.2 Bit/Byte ordering

All data variables in this document are presented with the most significant bit/byte on the left and the least significant bit/byte on the right hand side. When a variable is broken down into a number of substrings, the left most substring is numbered 0, the next most significant is numbered 1 and so on through to the least significant.
Example 2. Let a=$1001001100101100000001011010010_2$. Then the leftmost bit 1 of integer a represents its most significant bit, and the rightmost bit 0 represents its least significant bit.
Example 3. Let a =$10010010100101100000001011010010_2$. If a is subdivided into 4 of 8-bit substrings a [0], a [1], a [2] and a [3], then we have

a [0] = 10010010_2, a [1] = 10010110_2,
a [2] = 00000010_2, a [3] = 11010010_2.

2.2.3 Symbols

In this document, notations are defined as follows:

$a \| b$ Concatenation of substrings a and b
$\lceil x \rceil$ The least integer no less than x
$\oplus$ The exclusive OR
$a \ll t$ The t-bit left shift of integer a

Example 4. For any two substrings *a* and *b*, the string *c* made from the concatenation of *a* and *b*, also follows the rules defined in section 2.2, i.e., from the left to the right are the more significant digits to the less significant ones. For instance,

a= 0x1234,

b= 0x5678.

Then

$c=a \| b$ =0x12345678.

2.2.4 List of Variables

COUNT	The 32-bit counter.
BEARER	The 5-bit bearer identity.
DIRECTION	The 1-bit input indicating the direction of transmission.
CK	The 128-bit confidentiality key.
LENGTH	The 32-bit length of the input message.
M	The input message.
C	The output message.
Key	The 128-bit initial key to ZUC.
IV	The 128-bit initial vector to ZUC.
L	The number of key words generated by ZUC.
z[i]	The i-th key words generated by ZUC.
k[i]	The i-th key bit of keystream generated by ZUC.

3 CONFIDENTIALITY ALGORITHM *128-EEA3*

3.1 Introduction

The confidentiality algorithm ***128-EEA3*** is a stream cipher that is used to en/decrypt blocks of data under a confidentiality key. The block of data can be between 1 and 20,000 bits in length.

3.2 Inputs and Outputs

The inputs to the algorithm are given in Table 1, the output in Table 2.

Table 1 The inputs to 128-EEA3

Parameter	Size(bits)	Remark
COUNT	32	The counter
BEARER	5	The bearer identity
DIRECTION	1	The direction of transmission
CK	128	Confidentiality key
LENGTH	32	The length of the input message
M	LENGTH	The input bit stream

Table 2 The output of 128-EEA3

Parameter	Size(bits)	Remark
C	LENGTH	The output bit stream

3.3 Initialisation

In this section we define how ZUC's parameters, the initial key Key and the initial vector IV, are initialized with the confidentiality key CK and initialization variables before the generation of key bit stream.

Let

CK=CK[0] ‖ CK[1] ‖ CK[2] ‖ … ‖ CK[15]

be the 128-bit confidentiality key, where CK[i](0≤i≤15) are bytes. We set the 128-bit initial key Key to ZUC as

Key = Key[0] ‖ Key[1] ‖ Key[2] ‖ … ‖ Key[15],

where Key[i](0≤i≤15) are bytes. Then

Key[i]=CK[i], i=0,1,2,…,15.

Let

COUNT=COUNT[0] ‖ COUNT[1] ‖ COUNT[2] ‖ COUNT[3]

be the 32-bit counter, where COUNT[i](0≤i≤3) are bytes. We set the 128-bit initial vector to ZUC as

IV = IV[0] ‖ IV[1] ‖ IV[2] ‖ … ‖ IV[15],

where IV[i](0≤i≤15) are bytes. Then

IV[0] = COUNT[0], IV[1] = COUNT[1],
IV[2] = COUNT[2], IV[3] = COUNT[3],
IV[4] = BEARER||DIRECTION||00,
IV[5] = IV[6] = IV[7] = 0,
IV[8] = IV[0], IV[9] = IV[1],
IV[10] = IV[2], IV[11] = IV[3],
IV[12] = IV[4], IV[13] = IV[5],
IV[14] = IV[6], IV[15] = IV[7].

3.4 Keystream Generation

Let L = $\lceil LENGTH/32 \rceil$.

ZUC is run to generate key words

z[1], z[2], …, z[L]

under the initial key Key and initial vector IV defined as in section 3.3, where z[1] is the first key word generated by ZUC, z[2] is the next, and so on.

Let

k[0], k[1], …, k[LENGTH-1]

be the key bit stream corresponding to the above key words z[1], z[2], …, z[L]. Then k[0] is the most significant bit and k[31] is the least significant bit of z[1], k[32] is the most significant bit of z[2], and so on.

3.5 Encryption/Decryption

En/decryption operations are identical operations and are performed by the exclusive-OR of the input message M with the generated key stream k.

Let

M = M[0] ‖ M[1] ‖ M[2] ‖ … ‖ M[LENGTH-1]

be the input bit stream of length LENGTH and

C = C[0] ‖ C[1] ‖ C[2] ‖ … ‖ C[LENGTH-1]

be the corresponding output bit stream of length LENGTH, where M[i] and C[i] are bits, i=0,1,2,…,LENGTH-1. Then

C[i] = M[i] $\oplus$ k[i] , i=0,1,2,…,LENGTH-1.

4 INTEGRITY ALGORITHM *128-EIA3*

4.1 Introduction

The integrity algorithm ***128-EIA3*** is a message authentication code (MAC) function that is used to compute the MAC of an input message under an integrity key. The message can be between 1 and 20,000 bits in length.

4.2 Inputs and Outputs

The inputs to the algorithm are given in Table 3, the output in Table 4.

Table 3 The inputs to 128-EIA3

Parameter	Size(bits)	Remark
COUNT	32	The counter
BEARER	5	The bearer identity
DIRECTION	1	The direction of transmission
IK	128	The integrity key
LENGTH	32	The length of the input message
M	LENGTH	The input message

Table 4 The output of 128-EIA3

Parameter	Size(bits)	Remark
MAC	32	The MAC

4.3 Initialisation

In this section we define how ZUC's parameters, the initial key Key and the initial vector IV, are initialized with the integrity key IK and initialization variables before the generation of key stream. Let

IK = IK[0] ‖ IK[1] ‖ IK[2] ‖ … ‖ IK[15]

be the 128-bit integrity key, where IK[i](0≤i≤15) are bytes. We set the 128-bit initial key Key to ZUC as

Key = Key[0] ‖ Key[1] ‖ Key[2] ‖ … ‖ Key[15]

where Key[i](0≤i≤15) are bytes. Then

Key[i] = IK[i], i=0,1,2,…,15.

Let the 32-bit counter COUNT be

COUNT=COUNT[0] ‖ COUNT[1] ‖ COUNT[2] ‖ COUNT[3]

where COUNT[i] are bytes, i=0,1,2,3. We set the 128-bit initial vector IV to ZUC as

IV = IV[0] ‖ IV[1] ‖ IV[2] ‖ … ‖ IV[15],

where IV[i](0≤i≤15) are bytes. Then

IV[0] = COUNT[0], IV[1] = COUNT[1],
IV[2] = COUNT[2], IV[3] = COUNT[3],
IV[4] = BEARER||000_2, IV[5] =00000000_2,
IV[6] = 00000000_2, IV[7] = 00000000_2,
IV[8] = IV[0] $\oplus$ (DIRECTION << 7), IV[9] = IV[1],
IV[10] = IV[2], IV[11] = IV[3],
IV[12] = IV[4], IV[13] = IV[5],
IV[14] = IV[6] $\oplus$ (DIRECTION << 7), IV[15] = IV[7].

4.4 Generating the key stream

Let N=LENGTH+64 and L=$\lceil N/32 \rceil$.

ZUC is run to generate L key words z[1], z[2], …, z[L] with the initial key KEY and the initial vector

IV defined as in section 4.3, where z[1] is the first key word generated by ZUC, z[2] is the next, and so on.

Let

$k[0], k[1], \ldots, k[31], k[32], \ldots, k[N-1]$

be the key bit stream corresponding to the above key words z. Then k[0] is the most significant bit and k[31] is the least significant bit of z[1], k[32] is the most significant bit of z[2], and so on.

For each i=0,1,2,…,N-32, let

$k_i = k[i] \parallel k[i+1] \parallel \ldots \parallel k[i+31]$.

Then each k_i is a 32-bit word.

4.5 Compute the MAC

LET T be a 32-bit word. Set T = 0.

For each i=0,1,2,…,LENGTH-1, if M[i] = 1, then

$T = T \oplus k_i$.

Set

$T = T \oplus k_{LENGTH}$.

Finally we take $T \oplus k_{N-32}$ as the output MAC, i.e.

$MAC = T \oplus k_{N-32}$.

INFORMATIVE SECTION

This part of the document is purely informative and does not form part of the normative specification of the Confidentiality and Integrity algorithms.

ANNEX 1
A C implementation of *128-EEA3*

```
typedef unsigned char u8;
typedef unsigned int  u32;

/* The ZUC algorithm, see ref. [3]*/
void ZUC(u8* k, u8* iv, u32* ks, int len)
{
    /* The initialization of ZUC, see page 17 of ref. [3]*/
    Initialization(k, iv);

    /*  The procedure of generating keystream of ZUC, see page 18 of ref. [3]*/
    GenerateKeystream(ks, len);
}

void EEA3(u8* CK,u32 COUNT,u32 BEARER,u32 DIRECTION,u32 LENGTH,u32* M,u32* C)
{
    u32 *z, L, i;
    u8  IV[16];

    L   = (LENGTH+31)/32;
    z   = (u32 *) malloc(L*sizeof(u32));

    IV[0]    = (COUNT>>24) & 0xFF;
    IV[1]    = (COUNT>>16) & 0xFF;
    IV[2]    = (COUNT>>8)  & 0xFF;
    IV[3]    = COUNT       & 0xFF;

    IV[4]    = ((BEARER << 3) | ((DIRECTION&1)<<2)) & 0xFC;
    IV[5]    = 0;
    IV[6]    = 0;
    IV[7]    = 0;

    IV[8]    = IV[0];
    IV[9]    = IV[1];
    IV[10]   = IV[2];
    IV[11]   = IV[3];

    IV[12]   = IV[4];
    IV[13]   = IV[5];
    IV[14]   = IV[6];
    IV[15]   = IV[7];

    ZUC(CK,IV,z,L);

    for (i=0; i<L; i++)
    {
        C[i] = M[i] ^ z[i];
    }

    free(z);
}
```

ANNEX 2
A C implementation of *128-EIA3*

```
typedef unsigned char  u8;
typedef unsigned int   u32;
void ZUC(u8* k, u8* iv, u32* keystream, int length); /*see Annex 1*/
u32 GET_WORD(u32 * DATA, u32 i)
{
    u32 WORD, ti;

    ti  = i % 32;
    if (ti == 0) {
        WORD = DATA[i/32];
    }
    else {
        WORD = (DATA[i/32]<<ti) | (DATA[i/32+1]>>(32-ti));
    }

    return WORD;
}
u8 GET_BIT(u32 * DATA, u32 i)
{
    return (DATA[i/32] & (1<<(31-(i%32)))) ? 1 : 0;
}
void EIA3(u8* IK,u32 COUNT,u32 DIRECTION,u32 BEARER,u32 LENGTH,u32* M,u32* MAC)
{
    u32 *z, N, L, T, i;
    u8 IV[16];

    IV[0]   = (COUNT>>24) & 0xFF;
    IV[1]   = (COUNT>>16) & 0xFF;
    IV[2]   = (COUNT>>8) & 0xFF;
    IV[3]   = COUNT & 0xFF;

    IV[4]   = (BEARER << 3) & 0xF8;
    IV[5]   = IV[6] = IV[7] = 0;

    IV[8]   = ((COUNT>>24) & 0xFF) ^ ((DIRECTION&1)<<7);
    IV[9]   = (COUNT>>16) & 0xFF;
    IV[10]  = (COUNT>>8) & 0xFF;
    IV[11]  = COUNT & 0xFF;

    IV[12]  = IV[4];
    IV[13]  = IV[5];
    IV[14]  = IV[6] ^ ((DIRECTION&1)<<7);
    IV[15]  = IV[7];

    N   = LENGTH + 64;
    L   = (N + 31) / 32;
    z   = (u32 *) malloc(L*sizeof(u32));
    ZUC(IK, IV, z, L);

    T = 0;
    for (i=0; i<LENGTH; i++) {
        if (GET_BIT(M,i)) {
            T ^= GET_WORD(z,i);
        }
    }
    T ^= GET_WORD(z,LENGTH);

    *MAC = T ^ GET_WORD(z,LENGTH+32);
    free(z);
}
```