ETSI/SAGE Version: 1.7

Specification Date: 30<sup>th</sup> Dec, 2011

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

Document 1: 128-EEA3 and 128-EIA3 Specification

Document History		
V1.0	18 <sup>th</sup> 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
1.5	04-01-2011	A modification of 128-EIA3 and text improved
1.6	01-07-2011	Minor corrections
1.7	30-12-2011	Increase maximum data length for encryption algorithm

**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: Implementor's 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	8
2	INTRODUCTORY INFORMATION	8
	2.1 Introduction	8
	2.2 Notations	
3	CONFIDENTIALITY ALGORITHM 128-EEA3	10
	3.1 Introduction	10
	3.2 Inputs and Outputs	
	3.3 Initialisation	
	3.4 Keystream Generation	11
	3.5 Encryption/Decryption	
4	INTEGRITY ALGORITHM 128-EIA3	12
	4.1 Introduction	12
	4.2 Inputs and Outputs	
	4.3 Initialisation	
	4.4 Generating the keystream	
	4.5 Compute the MAC	
ANI	NEX 1 A C implementation of 128-EEA3	15
ANI	NEX 2 A C implementation of 128-EIA3	16

## **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.

This was to a finish and a succession to	NORMATIV		af aba Cantidana	a litar or a salar
This part of the document Integrity algorithms.	contains the norm	ative specification (	of the Confidenti	ality and

#### 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 sets 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 using a confidentiality key CK. The block of data may be between 1 and 2<sup>32</sup> 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 core algorithms adopted by the MAC are a universal hash and ZUC.

#### 2.2 **Notations**

#### 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:

```
// decimal representation
a = 1234567890
  = 0x499602D2
                        // hexadecimal representation
  = 1001001100101100000001011010010<sub>2</sub> //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. When a variable is broken down into a number of substrings, the leftmost substring is numbered by 0, the next most significant substring is numbered by 1 and so on throughout to the least significant substring.

Let  $a=100100110010110000001011010010_2$ . Then the leftmost bit 1 of integer  $a=10010011001011001001011010010_2$ . represents its most significant bit, and the rightmost bit 0 represents its least significant bit.

Let  $\alpha = 10010010100101100000001011010010_2$ . If  $\alpha$  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 **Operation notations**

In this document, operation notations are defined as follows:

```
a \parallel b
           Concatenation of substrings a and b
| x |
           The smallest integer no less than x
```

**Exclusive-OR** 

a<< t Left shift of integer a by t bits

**Example 4.** For two substrings a= 0x1234 and b= 0x5678, then their concatenation will be c= $a \parallel b$  =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. IK The 128-bit integrity key.

LENGTH The number of bits to be encrypted/decrypted.

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 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 encrypt/decrypt blocks of data under a confidentiality key. The block of data can be between 1 and  $2^{32}$  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
СК	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
С	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 keystream.

Let

```
CK=CK[0] | CK[1] | CK[2] | ... | CK[15]
```

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

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] (  $0 \le i \le 3$ ) are bytes. We set the 128-bit initial vector to ZUC as

```
IV = IV[0] \parallel IV[1] \parallel IV[2] \parallel ... \parallel IV[15], where IV[i] ( 0 \le i \le 15) are bytes. Then IV[0] = COUNT[0], IV[1] = COUNT[1], IV[2] = COUNT[2], IV[3] = COUNT[3], IV[4] = BEARER \parallel DIRECTION \parallel 00_2, IV[5] = IV[6] = IV[7] = 000000000_2, 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 ZUC generate keystream of L words. When each of the word is expanded into a 32-bit string, then we get a binary string z[0], z[1], ...,  $z[32\times L-1]$ , where z[0] is the most significant bit of the first output word of ZUC and z[31] is the least significant bit. To encrypt a message of LENGTH bits, it is required that L=[LENGTH/32].

# 3.5 Encryption/Decryption

Encryption/decryption operations are identical operations and are performed by the exclusive-OR of the input message M with the generated keystream z.

 $M = M[0] \parallel M[1] \parallel M[2] \parallel ... \parallel M[LENGTH-1]$  be the input bit stream of length LENGTH and  $C = C[0] \parallel C[1] \parallel C[2] \parallel ... \parallel C[LENGTH-1]$  be the corresponding output bit stream of length LENGTH, where M[i] and C[i] are bits, i=0,1,2,...,LENGTH-1. Then  $C[i] = M[i] \oplus z[i], i=0,1,2,...,LENGTH-1.$ 

### 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 using an integrity key IK. The message can be between 1 and 65504 bits in length.

## 4.2 Inputs and Outputs

The inputs to the algorithm are given in Table 3, and the output is 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 bits 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 keystream.

```
IK = IK[0] \| IK[1] \| IK[2] \| ... \| IK[15]
```

be the 128-bit integrity key, where IK[i](  $0 \le i \le 15$ ) 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]( $0 \le i \le 15$ ) 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 keystream

Let ZUC generate a keystream of L=LENGTH/32 + 2 words. Denote the generated bit string by  $z[0], z[1], ..., z[32 \times L-1],$  where z[0] is the most significant bit of the first output word of ZUC and z[31]

is the least significant bit.

For each i=0,1,2,...,32×(L-1), let  $z_i = z[i] \| z[i+1] \| ... \| z[i+31]$ . Then each  $z_i$  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 z_i. Set T=T\oplus z_{\mathsf{LENGTH}}. Finally we take T\oplus z_{32\times(L-1)} as the output MAC, i.e. \mathsf{MAC}=T\oplus z_{32\times(L-1)}.
```

INFORMATIVE SECTION  This part of the document is purely informative and does not form part of the normal	itive
specification of the Confidentiality and Integrity algorithms.	
20DD C	

# 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));
    [0]VI
            = (COUNT>>24) & 0xFF;
            = (COUNT>>16) & 0xFF;
= (COUNT>>8) & 0xFF;
= COUNT & 0xFF;
    IV[1]
IV[2]
    IV[3]
    IV[4]
             = ((BEARER << 3) | ((DIRECTION&1) << 2)) & 0xFC;
             = 0;
    IV[5]
IV[6]
             = 0;
    IV[7]
             = 0;
             = IV[0];
    [8]VI
             = IV[1];
= IV[2];
    IV[9]
IV[10]
    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;

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];
             = (COUNT>>24) & 0xFF;
     [0]VI
     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;
             = ((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);
     for (i=0; i<LENGTH; i++) {
          if (GET_BIT(M,i)) {
    T ^= GET_WORD(z,i);
     T ^= GET WORD(z, LENGTH);
     *MAC = T ^ z[L-1];
     free(z);
}
```