Technical Specification Group Services and System Aspects
Meeting #18, New Orleans, USA, 9-12 December 2002

TSGS#18(02)0721

Source: **SA WG3**

Title: **4 CRs to 55.216, 55.217, 55.218 and 55.919 (Rel-6): EGPRS algoritm**

Document for: **Approval**

Agenda Item: **7.3.3**

SA doc#	Spec	CR	R	Phase	Subject	Cat	Current Version	WI	SA WG3 doc#
SP-020721	55.216	001		Rel-6	EGPRS algoritm	F	6.0.0	SEC1-CSALGO1	S3-020657
SP-020721	55.217	001		Rel-6	EGPRS algoritm	F	6.0.0	SEC1-CSALGO1	S3-020658
SP-020721	55.218	001		Rel-6	EGPRS algoritm	F	6.0.0	SEC1-CSALGO1	S3-020659
SP-020721	55.919	001		Rel-6	EGPRS algoritm	F	6.0.0	SEC1-CSALGO1	S3-020656

3GPP TSG-SA3 Meeting #26
Oxford, England, 19-22 November

Tdoc ⌘S3-020657

CR-Form-v7

CHANGE REQUEST

⌘ **TS 55.216** **CR** **001** ⌘**rev** **-** ⌘ Current version: **6.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☒ Radio Access Network ☐ Core Network ☒

Title:	⌘	EGPRS algoritm
Source:	⌘	SA WG3
Work item code:	⌘	SEC1-CSALGO1
Date:	⌘	14/11/2002
Category:	⌘	**F**
Release:	⌘	REL-6

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
Rel-4 (Release 4)
Rel-5 (Release 5)
Rel-6 (Release 6)

Reason for change: ⌘ At SA3 #25 Ericsson presented a discussion paper in S3-020545 asking for clarification on the algorithm to be used for EGPRS.

The following extract has been taken from the SA3 #25 meeting report:

"TD S3-020545 A5/3 and GEA3 and their relation with EGPRS. This was introduced by Ericsson and questions the use of A5/3 for EDGE and the data-rate for EGPRS and asks SA WG3 to discuss the issues raised in order to provide any necessary CRs to the next SA WG3 meeting. It was confirmed that A5/3 and GEA3 were suitable for both GSM/GPRS and EDGE variants, the algorithm specifications are unclear on this: ***The modulation scheme used in the PS domain does not affect the GEA3 algorithm mechanism. A5/3 (CS domain) has 2 modes of use, GSM standard mode and GSM EDGE mode****. No CR to TS 43.020 was thought necessary, as implementers need to look at the algorithm specifications where the two modes of operation are clarified. It was agreed, however, to create a CR to the Technical Report TR 55.919 to clarify the use of the term "EDGE" in the specifications and the EGPRS bit-rates.* ***K. Boman agreed to do this for the next SA WG3 meeting****."*

It is proposed to change and clarify the wording in the Technical Specifications as well as TS 55.216.

Summary of change:	⌘	The term "EDGE" has been deleted from TS 55.216 as it very confusing i.e. the definition is unclear in 3GPP whether it applies for enhanced circuit-switched data or enhanced GPRS or both. The term ECSD has been introduced as it is defined in 21.905 Vocabulary for 3GPP Specifications and stands for enhanced circuit-switched data. The term EGPRS has been introduced as it is defined in 21.905 Vocabulary for 3GPP Specifications and stands for enhanced GPRS. It's been clarified that GEA3 shall be used for EGPRS.
Consequences if not approved:	⌘	It's unclear whether: - the term EDGE means enhanced circuit-switched data or enhanced GPRS or both; - what algorithm that shall be used for EGPRS.

Clauses affected:	⌘					
Other specs affected:	⌘	**Y**	**N**		⌘	
			X	Other core specifications		
		X		Test specifications		55.217, 55.218
			X	O&M Specifications		
Other comments:	⌘	SAGE draft 1.0 (technically equivalent to SA#17 approved version 1.0.0) is used for this CR, as electronic versions of 3GPP specification is not available on the 3GPP FTP site at present.				

ETSI/SAGE Specification	Version: 1.0 Date: 30th May 2002

Specification of the A5/3 Encryption Algorithms for GSM and ECSD~~DGE~~, and the GEA3 Encryption Algorithm for GPRS

Document 1: A5/3 and GEA3 Specifications

Document History		
V1.0	**30th May 2002**	**First publication**

PREFACE

This specification has been prepared by the 3GPP Task Force, and gives a detailed specification of the **A5/3** encryption algorithms for GSM and ECSD~~DGE~~, and of the **GEA3** encryption algorithm for GPRS.

This document is the first of three, which between them form the entire specification of the **A5/3** and **GEA3** algorithms:

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 1: **A5/3** and **GEA3** Specifications.

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 2: Implementors' Test Data.

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 3: Design Conformance Test Data.

The normative part of the specification of the **A5/3** and **GEA3** algorithms is in the main body of this document. The annexes to this document are purely informative. Annex A gives a specification of the 3GPP ***f*8** confidentiality algorithm, bringing out the (deliberate) commonality between it and the **A5/3** and **GEA3** algorithms. Annex B contains illustrations of functional elements of the algorithms, while Annex C contains an implementation program listing of the cryptographic algorithms specified in the main body of this document, written in the programming language C.

Documents 2 and 3 above are also purely informative.

The normative part of the specification of the block cipher (**KASUMI)** on which the **A5/3** and **GEA3** algorithms are based can be found in [5].

Blank Page

TABLE OF CONTENTS

REFERENCES

[1] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 1: **A5/3** and **GEA3** Specifications.

[2] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 2: Implementors' Test Data.

[3] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 3: Design Conformance Test Data.

[4] Specification of the 3GPP Confidentiality and Integrity Algorithms;
Document 1: *f8* and *f9* specifications.

[5] Specification of the 3GPP Confidentiality and Integrity Algorithms;
Document 2: **KASUMI** specification.

NORMATIVE SECTION

This part of the document contains the normative specifications of the **A5/3** and **GEA3** encryption algorithms.

1. OUTLINE OF THE NORMATIVE PART

Section 2 introduces the algorithms and describes the notation used in the subsequent sections.

Section 3 specifies a core function **KGCORE**.

Section 4 specifies the encryption algorithm **A5/3** for GSM in terms of the function **KGCORE**.

Section 5 specifies the encryption algorithm **A5/3** for ECSD~~DGE~~ in terms of the function **KGCORE**.

Section 6 specifies the encryption algorithm **GEA3** for GPRS in terms of the function **KGCORE**.

2. INTRODUCTORY INFORMATION

2.1. Introduction

In this document are specified three ciphering algorithms: **A5/3** for GSM, **A5/3** for ECSD~~DGE~~, and **GEA3** for GPRS (including EGPRS). The algorithms are stream ciphers that are used to encrypt/decrypt blocks of data under a confidentiality key $\mathbf{K_C}$. Each of these algorithms is based on the **KASUMI** algorithm that is specified in reference [5]. **KASUMI** is a block cipher that produces a 64-bit output from a 64-bit input under the control of a 128-bit key. The algorithms defined here use **KASUMI** in a form of output-feedback mode as a keystream generator.

The three algorithms are all very similar. We first define a core keystream generator function **KGCORE** (section 3); we then specify each of the three algorithms in turn (sections 4, 5 and 6) in terms of this core function.

2.2. Notation

2.2.1. Radix

We use the prefix **0x** to indicate **hexadecimal** numbers.

2.2.2. Conventions

We use the assignment operator '=', as used in several programming languages. When we write

<*variable*> = <*expression*>

we mean that <*variable*> assumes the value that <*expression*> had before the assignment took place. For instance,

$$x = x + y + 3$$

means

(new value of x) becomes (old value of x) + (old value of y) + 3.

2.2.3. Bit/Byte ordering

All data variables in this specification are presented with the most significant bit (or byte) on the left hand side and the least significant bit (or byte) on the right hand side. Where a variable is broken down into a number of sub-strings, the left most (most significant) sub-string is numbered 0, the next most significant is numbered 1 and so on through to the least significant.

For example an n-bit **STRING** is subdivided into 64-bit substrings $\mathbf{SB_0, SB_1 \ldots SB_i}$ so if we have a string:

0x0123456789ABCDEFFEDCBA98765432l086545381AB594FC28786404C50A37…

we have:

$\mathbf{SB_0}$ = 0x0123456789ABCDEF
$\mathbf{SB_1}$ = 0xFEDCBA9876543210
$\mathbf{SB_2}$ = 0x86545381AB594FC2
$\mathbf{SB_3}$ = 0x8786404C50A37…

In binary this would be:

000000010010001101000101011001111000100110101011110011011110111111111110…

with $\mathbf{SB_0}$ = 00000001001000110100010101100111100010011010101111001101111011111
$\mathbf{SB_1}$ = 11111110110111001011101010011000011101100101010000110010000100000
$\mathbf{SB_2}$ = 10000110010101000101001110000001101010110101100101001111110000010
$\mathbf{SB_3}$ = 10000111100001100100000001001100010100001010001101110…

2.2.4. List of Symbols

=	The assignment operator.
⊕	The bitwise exclusive-OR operation
||	The concatenation of the two operands.
$KASUMI[x]_k$	The output of the **KASUMI** algorithm applied to input value **x** using the key **k**.
X[i]	The i^{th} bit of the variable **X**. (**X = X[0] || X[1] || X[2] || …..**).
Y{i}	The i^{th} octet of the variable **Y**. (**Y = Y{0} || Y{1} || Y{2} || …..**).
Z_i	The i^{th} 64-bit block of the variable **Z**. **(Z = Z_0 || Z_1 || Z_2 || ….)**.

2.3. List of Variables

A	a 64-bit register that is used within the **KGCORE** function to hold an intermediate value.
BLKCNT	a 64-bit counter used in the **KGCORE** function.
BLOCK1	a string of keystream bits output by the **A5/3** algorithm — 114 bits for GSM, 348 bits for ECSD~~DGE~~.

BLOCK2	a string of keystream bits output by the **A5/3** algorithm — 114 bits for GSM, 348 bits for ECSD~~DGE~~.
BLOCKS	an integer variable indicating the number of successive applications of **KASUMI** that need to be performed.
CA	an 8-bit input to the **KGCORE** function.
CB	a 5-bit input to the **KGCORE** function.
CC	a 32-bit input to the **KGCORE** function.
CD	a 1-bit input to the **KGCORE** function.
CE	a 16-bit input to the **KGCORE** function.
CK	a 128-bit input to the **KGCORE** function.
CL	an integer input to the **KGCORE** function, in the range $1\ldots2^{19}$ inclusive, specifying the number of output bits for **KGCORE** to produce.
CO	the output bitstream (**CL** bits) from the **KGCORE** function.
COUNT	a 22-bit frame dependent input to both the GSM and ECSD~~DGE~~ **A5/3** algorithms.
DIRECTION	a 1-bit input to the **GEA3** algorithm, indicating the direction of transmission (uplink or downlink).
INPUT	a 32-bit frame dependent input to the **GEA3** algorithm.
K_C	the cipher key that is an input to each of the three cipher algorithms defined here. Although at the time of writing the standards specify that K_C is 64 bits long, the algorithm specifications here allow it to be of any length between 64 and 128 inclusive, to allow for possible future enhancements to the standards.
KLEN	the length of K_C in bits, between 64 and 128 inclusive (see above).
KM	a 128-bit constant that is used to modify a key. This is used in the **KGCORE** function.
KS[i]	the i^{th} bit of keystream produced by the keystream generator in the **KGCORE** function.
KSB_i	the i^{th} block of keystream produced by the keystream generator in the **KGCORE** function. Each block of keystream comprises 64 bits.
M	an input to the **GEA3** algorithm, specifying the number of octets of output to produce.
OUTPUT	the stream of output octets from the **GEA3** algorithm.

3. CORE FUNCTION KGCORE

3.1. Introduction

In this section we define a general-purpose keystream generation function **KGCORE**. The individual encryption algorithms for GSM, GPRS and ECSD~~DGE~~ will each be defined in subsequent sections by mapping the relevant inputs to the inputs of **KGCORE**, and mapping the output of **KGCORE** to the relevant output.

3.2. Inputs and Outputs

The inputs to **KGCORE** are given in table 1, the output in table 2:

Parameter	Comment
CA	8 bits **CA[0]…CA[7]**
CB	5 bits **CB[0]…CB[4]**
CC	32 bits **CC[0]…CC[31]**
CD	A single bit **CD[0**]
CE	16 bits **CE[0]…CE[15]** (see Note 1 below)
CK	128 bits **CK[0]….CK[127]**
CL	An integer in the range $1…2^{19}$ inclusive, specifying the number of output bits to produce

Table 1. **KGCORE** inputs

Parameter	Comment
CO	**CL** bits **CO[0]…CO[CL-1]**

Table 2. **KGCORE** output

Note 1: All the algorithms specified in this document assign a constant, all-zeroes value to **CE**. More general use of **CE** is, however, available for possible future uses of **KGCORE**.

3.3. Components and Architecture

(See fig 1 Annex B)

The function **KGCORE** is based on the block cipher **KASUMI** that is specified in [2]. **KASUMI** is used in a form of output-feedback mode and generates the output bitstream in multiples of 64 bits.

The feedback data is modified by static data held in a 64-bit register **A**, and an (incrementing) 64-bit counter **BLKCNT**.

3.4. Initialisation

In this section we define how the keystream generator is initialised with the input variables before the generation of keystream bits as output.

We set the 64-bit register **A** to **CC || CB || CD || 0 0 || CA || CE**

i.e. **A = CC[0]…CC[31] CB[0]…CB[4] CD[0] 0 0 CA[0]…CA[7] CE[0]…CE[15]**

We set the key modifier **KM** to 0x55555555555555555555555555555555

We set $\mathbf{KSB_0}$ to zero.

One operation of **KASUMI** is then applied to the register **A**, using a modified version of the confidentiality key.

$$\mathbf{A = KASUMI[\ A\ ]_{CK \oplus KM}}$$

3.5. Keystream Generation

Once the keystream generator has been initialised in the manner defined in section 3.4, it is ready to be used to generate keystream bits. The keystream generator produces bits in blocks of 64 at a time, but the number **CL** of output bits to produce may not be a multiple of 64; between 0 and 63 of the least significant bits are therefore discarded from the last block, depending on the total number of bits specified by **CL**.

So let **BLOCKS** be equal to (**CL**/64) rounded up to the nearest integer. (For instance, if **CL** = 128 then **BLOCKS** = 2; if **CL** = 129 then **BLOCKS** = 3.)

To generate each keystream block (**KSB**) we perform the following operation:

For each integer **n** with $1 \leq \mathbf{n} \leq \mathbf{BLOCKS}$ we define:

$$\mathbf{KSB_n = KASUMI[\ A \oplus BLKCNT \oplus KSB_{n-1}]_{CK}}$$

where **BLKCNT = n-1**

The individual bits of the output are extracted from $\mathbf{KSB_1}$ to $\mathbf{KSB_{BLOCKS}}$ in turn, most significant bit first, by applying the operation:

For **n** = 1 to **BLOCKS**, and for each integer ***i*** with $0 \leq \boldsymbol{i} \leq 63$ we define:

$$\mathbf{CO[((n-1)*64)+i] = KSB_n[i]}$$

4. A5/3 ALGORITHM FOR GSM ENCRYPTION

4.1. Introduction

The GSM **A5/3** algorithm produces two 114-bit keystream strings, one of which is used for uplink encryption/decryption and the other for downlink encryption/decryption.

We define this algorithm in terms of the core function **KGCORE**.

4.2. Inputs and Outputs

The inputs to the algorithm are given in table 3, the output in table 4:

Parameter	Size (bits)	Comment
COUNT	22	Frame dependent input **COUNT[0]…COUNT[21]**
$\mathbf{K_C}$	64–128	Cipher key $\mathbf{K_C[0]}$… $\mathbf{K_C[KLEN-1]}$, where **KLEN** is in the range 64…128 inclusive (see Notes 1 and 2 below)

Table 3. GSM **A5/3** inputs

Parameter	Size (bits)	Comment
BLOCK1	114	Keystream bits **BLOCK1[0]…BLOCK1[113]**
BLOCK2	114	Keystream bits **BLOCK2[0]…BLOCK2[113]**

Table 4. GSM **A5/3** outputs

Note 1: At the time of writing, the standards specify that $\mathbf{K_C}$ is 64 bits long. This specification of the **A5/3** algorithm allows for possible future enhancements to support longer keys.

Note 2: It must be assumed that $\mathbf{K_C}$ is unstructured data — it must not be assumed, for instance, that any bits of $\mathbf{K_C}$ have predetermined values.

4.3. Function Definition

(See fig 2 Annex B)

We define the function by mapping the GSM **A5/3** inputs onto the inputs of the core function **KGCORE**, and mapping the output of **KGCORE** onto the outputs of GSM **A5/3**.

So we define:

CA[0]…CA[7] = **0 0 0 0 1 1 1 1**

CB[0]…CB[4] = **0 0 0 0 0**

CC[0]…CC[9] = **0 0 0 0 0 0 0 0 0 0**

CC[10]…CC[31] = **COUNT[0]…COUNT[21]**

CD[0] = 0

CE[0]…CE[15] = **0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

CK[0]…CK[KLEN-1] = K_C[0]…K_C[KLEN-1]

If **KLEN** < 128 then

CK[KLEN]…CK[127] = K_C[0]…K_C[127 – KLEN]

(So in particular if **KLEN** = 64 then **CK = K_C || K_C**)

CL = 228

Apply **KGCORE** to these inputs to derive the output **CO[0]…CO[227]**.

Then define:

BLOCK1[0]…BLOCK1[113] = **CO[0]…CO[113]**

BLOCK2[0]…BLOCK2[113] = **CO[114]…CO[227]**

5. A5/3 ALGORITHM FOR ECSD~~DGE~~ ENCRYPTION

5.1. Introduction

The ECSD~~DGE~~ **A5/3** algorithm produces two 348-bit keystream strings, one of which is used for uplink encryption/decryption and the other for downlink encryption/decryption.

We define this algorithm in terms of the core function **KGCORE**.

5.2. Inputs and Outputs

The inputs to the algorithm are given in table 5, the output in table 6:

Parameter	Size (bits)	Comment
COUNT	22	Frame dependent input **COUNT[0]…COUNT[21]**
$\mathbf{K_C}$	64–128	Cipher key $\mathbf{K_C[0]}$… $\mathbf{K_C[KLEN-1]}$, where **KLEN** is in the range 64…128 inclusive (see Notes 1 and 2 below)

Table 5. ECSD~~DGE~~ **A5/3** inputs

Parameter	Size (bits)	Comment
BLOCK1	348	Keystream bits **BLOCK1[0]…BLOCK1[347]**
BLOCK2	348	Keystream bits **BLOCK2[0]…BLOCK2[347]**

Table 6. ECSD~~DGE~~ **A5/3** outputs

Note 1: At the time of writing, the standards specify that $\mathbf{K_C}$ is 64 bits long. This specification of the **A5/3** algorithm allows for possible future enhancements to support longer keys.

Note 2: It must be assumed that $\mathbf{K_C}$ is unstructured data — it must not be assumed, for instance, that any bits of $\mathbf{K_C}$ have predetermined values.

5.3. Function Definition

(See fig 3 Annex B)

We define the function by mapping the ECSD~~DGE~~ **A5/3** inputs onto the inputs of the core function **KGCORE**, and mapping the output of **KGCORE** onto the outputs of ECSD~~DGE~~ **A5/3**.

So we define:

CA[0]…CA[7] = **1 1 1 1 0 0 0 0**

CB[0]…CB[4] = **0 0 0 0 0**

CC[0]…CC[9] = **0 0 0 0 0 0 0 0 0 0**

CC[10]…CC[31] = **COUNT[0]…COUNT[21]**

CD[0] = 0

CE[0]…CE[15] = **0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

CK[0]…CK[KLEN-1] = K_C[0]…K_C[KLEN-1]

If **KLEN** < 128 then

CK[KLEN]…CK[127] = K_C[0]…K_C[127 – KLEN]

(So in particular if **KLEN** = 64 then **CK = K_C || K_C**)

CL = 696

Apply **KGCORE** to these inputs to derive the output **CO[0]…CO[695]**.

Then define:

BLOCK1[0]…BLOCK1[347] = **CO[0]…CO[347]**

BLOCK2[0]…BLOCK2[347] = **CO[348]…CO[695]**

6. GEA3 ALGORITHM FOR GPRS ENCRYPTION

6.1. Introduction

The GPRS **GEA3** algorithm produces an M-byte keystream string. M can vary; in this specification we assume that M will never exceed $2^{16} = 65536$.

We define this algorithm in terms of the core function **KGCORE**.

6.2. Inputs and Outputs

The inputs to the algorithm are given in table 7, the output in table 8:

Parameter	Size (bits)	Comment
INPUT	32	Frame dependent input **INPUT[0]…INPUT[31]**
DIRECTION	1	Direction of transmission indicator **DIRECTION[0]**
$\mathbf{K_C}$	64–128	Cipher key $\mathbf{K_C[0]}$… $\mathbf{K_C[KLEN-1]}$, where **KLEN** is in the range 64…128 inclusive (see Notes 1 and 2 below)
M		Number of octets of output required, in the range 1 to 65536 inclusive

Table 7. **GEA3** inputs

Parameter	Size (bits)	Comment
OUTPUT	8**M**	Keystream octets **OUTPUT{0}…OUTPUT{M-1}**

Table 8. **GEA3** outputs

Note 1: At the time of writing, the standards specify that $\mathbf{K_C}$ is 64 bits long. This specification of the **GEA3** algorithm allows for possible future enhancements to support longer keys.

Note 2: It must be assumed that $\mathbf{K_C}$ is unstructured data — it must not be assumed, for instance, that any bits of $\mathbf{K_C}$ have predetermined values.

6.3. Function Definition

(See fig 4 Annex B)

We define the function by mapping the **GEA3** inputs onto the inputs of the core function **KGCORE**, and mapping the output of **KGCORE** onto the outputs of **GEA3**.

So we define:

CA[0]…CA[7] = **1 1 1 1 1 1 1 1**

CB[0]…CB[4] = **0 0 0 0 0**

CC[0]…CC[31] = **INPUT[0]…INPUT[31]**

CD[0] = DIRECTION[0]

CE[0]…CE[15] = **0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

CK[0]…CK[KLEN-1] = K_C[0]…K_C[KLEN-1]

If **KLEN** < 128 then

CK[KLEN]…CK[127] = K_C[0]…K_C[127 – KLEN]

(So in particular if **KLEN** = 64 then **CK = K_C || K_C**)

CL = 8**M**

Apply **KGCORE** to these inputs to derive the output **CO[0]…CO[8M-1]**.

Then for $0 \leq i \leq M-1$ define:

OUTPUT{*i*} = **CO[8*i*]…CO[8*i* + 7]**

where **CO[8*i*]** is the most significant bit of the octet.

INFORMATIVE SECTION

This part of the document is purely informative and does not form part of the normative specification of A5/3 and GEA3.

ANNEX A
Specification of the 3GPP Confidentiality Algorithm *f*8

A.1 Introduction

The algorithms defined in this specification have been designed to have much in common with the 3GPP confidentiality algorithm, to ease simultaneous implementation of multiple algorithms. To clarify this, a specification of ***f*8** is given here in terms of the core function **KGCORE**. For the definitive specification of ***f*8**, the reader is referred to [5].

A.2 Inputs and Outputs

The inputs to the algorithm are given in table A.1, the output in table A.2:

Parameter	Size (bits)	Comment
COUNT	32	Frame dependent input **COUNT[0]…COUNT[31]**
BEARER	5	Bearer identity **BEARER[0]…BEARER[4]**
DIRECTION	1	Direction of transmission **DIRECTION[0]**
CK	128	Confidentiality key **CK[0]…CK[127]**
LENGTH		The number of bits to be encrypted/decrypted (1-20000)

Table A.1. ***f8*** inputs

Parameter	Size (bits)	Comment
KS	1-20000	Keystream bits **KS[0]…KS[LENGTH-1]**

Table A.2. ***f8*** output

Note: The definitive specification of ***f*8** includes a bitstream **IBS** amongst the inputs, and gives the output as a bitstream **OBS**; both of these bitstreams are **LENGTH** bits long. **OBS** is obtained by the bitwise exclusive-or of **IBS** and **KS**. We present just the keystream generator part of ***f*8** here, for closer comparison with **A5/3** and **GEA3**.

A.3 Function Definition

(See fig 5 Annex B)

We define the function by mapping the ***f*8** inputs onto the inputs of the core function **KGCORE**, and mapping the output of **KGCORE** onto the outputs of ***f*8**.

So we define:

CA[0]…CA[7] = **0 0 0 0 0 0 0 0**

CB[0]…CB[4] = **BEARER[0]…BEARER[4]**

CC[0]…CC[31] = **COUNT[0]…COUNT[31]**

CD[0] = DIRECTION[0]

CE[0]…CE[15] = **0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

CK[0]…CK[127] = CK[0]…CK[127]

CL = LENGTH

Apply **KGCORE** to these inputs to derive the output **CO[0]…CO[LENGTH-1]**.

Then define:

KS[0]…KS[LENGTH-1] = **CO[0]…CO[LENGTH-1]**

ANNEX B
Figures of the Algorithms

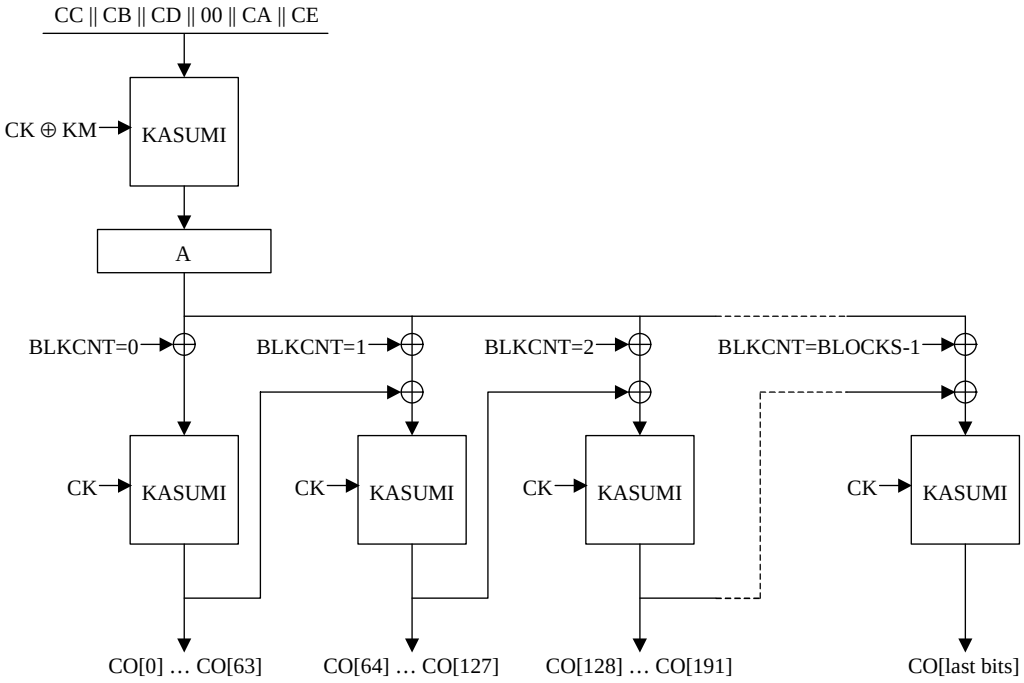

Figure 1: **KGCORE** Core Keystream Generator Function

Note: **BLKCNT** is specified as a 64-bit counter so there is no ambiguity in the expression $\mathbf{A \oplus BLKCNT \oplus KSB_{n-1}}$ where all operands are of the same size. In a practical implementation, where the keystream generator is required to produce no more than a certain number of bits, only the least significant few bits of the counter need to be realised.

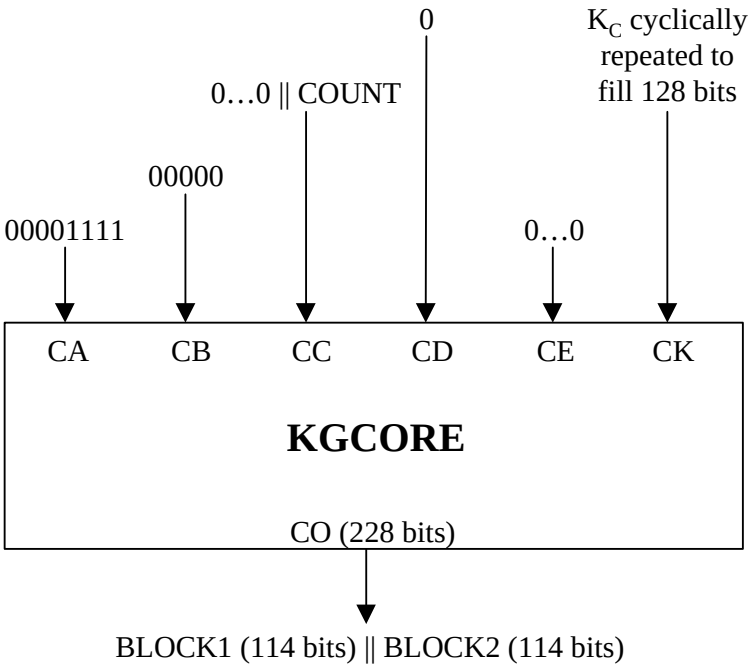

Figure 2: GSM **A5/3** Keystream Generator Function

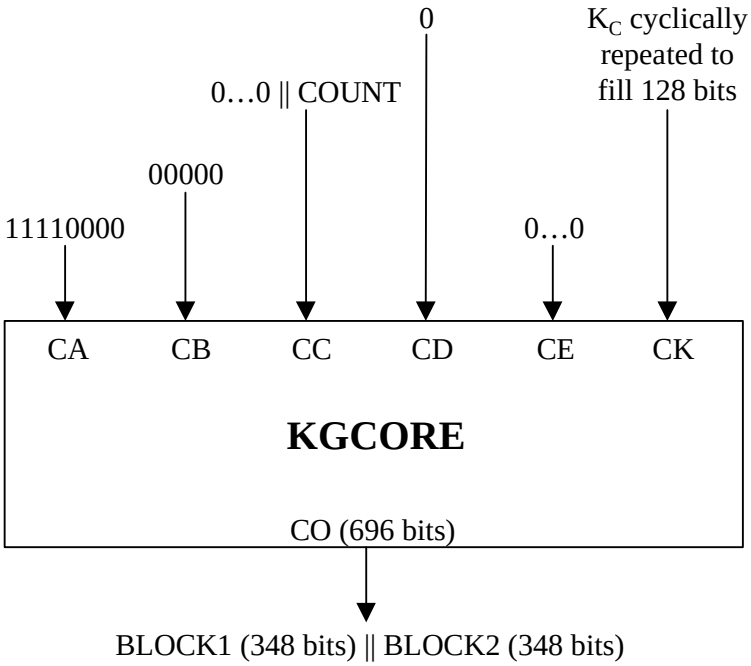

Figure 3: ECSD~~DGE~~ **A5/3** Keystream Generator Function

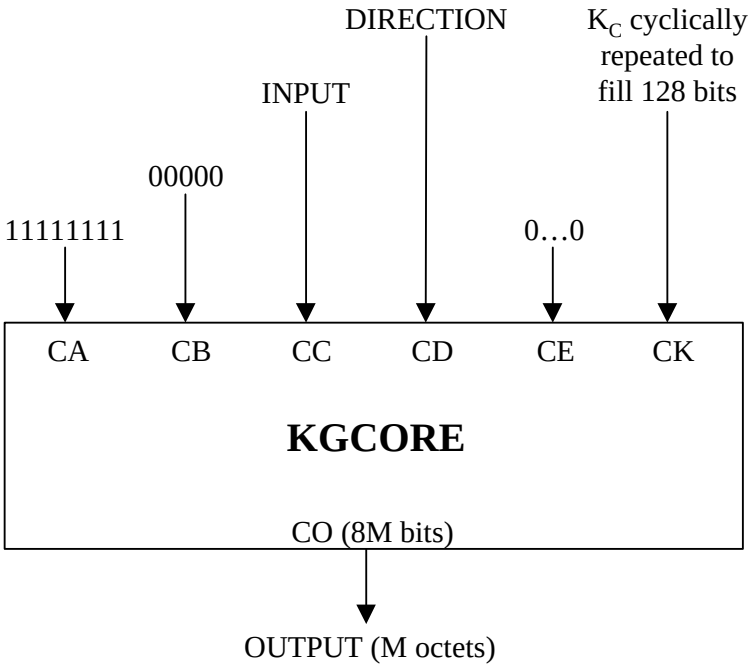

Figure 4: **GEA3** Keystream Generator Function

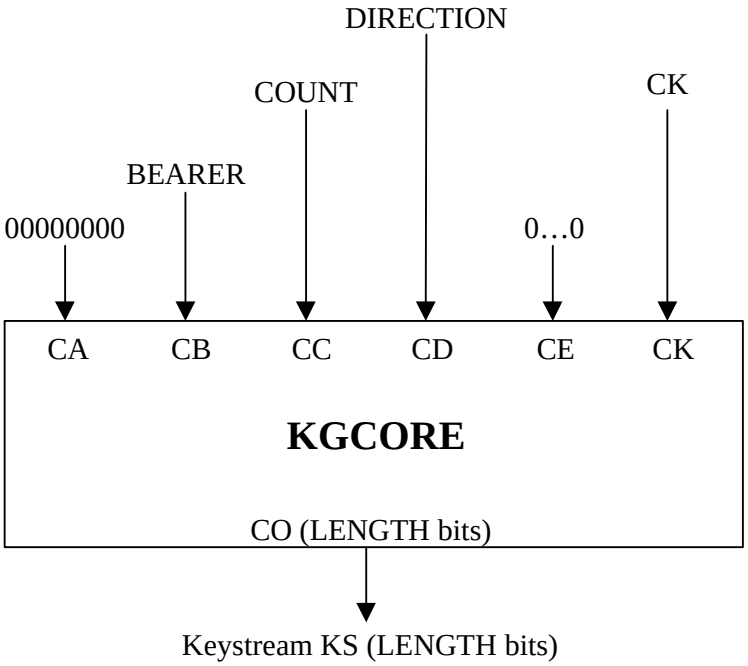

Figure 5: 3GPP *f*8 Keystream Generator Function

	GSM **A5/3**	ECSD~~DGE~~ **A5/3**	**GEA3**	***f*8**
CA	**0 0 0 0 1 1 1 1**	**1 1 1 1 0 0 0 0**	**1 1 1 1 1 1 1 1**	**0 0 0 0 0 0 0 0**
CB	**0 0 0 0 0**	**0 0 0 0 0**	**0 0 0 0 0**	**BEARER**
CC	**0...0||COUNT**	**0...0||COUNT**	**INPUT**	**COUNT**
CD	**0**	**0**	**DIRECTION**	**DIRECTION**
CE	**0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**			
CK	$\mathbf{K_C}$ repeated to fill 128 bits			**CK**
CO	**BLOCK1||BLOCK2**	**BLOCK1||BLOCK2**	**OUTPUT**	**KS**

Table B.1: GSM **A5/3**, ECSD~~DGE~~ **A5/3**, **GEA3** and ***f*8** in terms of **KGCORE**

ANNEX C
Simulation Program Listings

kasumi.h

```
/*---------------------------------------------------------
 *                   Kasumi.h
 *---------------------------------------------------------*/

typedef unsigned  char   u8;
typedef unsigned short  u16;
typedef unsigned   int  u32;

/*----- a 64-bit structure to help with endian issues -----*/

typedef union {
    u32 b32[2];
    u16 b16[4];
    u8  b8[8];
} REGISTER64;

/*------------- prototypes --------------------------------*/

void KeySchedule( u8 *key );
void Kasumi( u8 *data );
```

kasumi.c

```
/*---------------------------------------------------------------------
 *                       Kasumi.c
 *---------------------------------------------------------------------
 *
 *  A sample implementation of KASUMI, the core algorithm for the
 *  3GPP Confidentiality and Integrity algorithms.
 *
 *  This has been coded for clarity, not necessarily for efficiency.
 *
 *  This will compile and run correctly on both Intel (little endian)
 *  and Sparc (big endian) machines. (Compilers used supported 32-bit ints).
 *
 *  Version 1.1     08 May 2000
 *
 *---------------------------------------------------------------------*/

#include "Kasumi.h"

/*--------- 16 bit rotate left ------------------------------------------*/

#define ROL16(a,b) (u16)((a<<b)|(a>>(16-b)))

/*------- unions: used to remove "endian" issues ----------------------*/

typedef union {
    u32 b32;
    u16 b16[2];
    u8  b8[4];
} DWORD;

typedef union {
    u16 b16;
    u8  b8[2];
} WORD;
```

```
/*-------- globals: The subkey arrays ---------------------------------*/

static u16 KLi1[8], KLi2[8];
static u16 KOi1[8], KOi2[8], KOi3[8];
static u16 KIi1[8], KIi2[8], KIi3[8];

 /*---------------------------------------------------------------------
 *  FI()
 *      The FI function (fig 3).  It includes the S7 and S9 tables.
 *      Transforms a 16-bit value.
 *---------------------------------------------------------------------*/
static u16 FI( u16 in, u16 subkey )
{
    u16 nine, seven;
    static u16 S7[] = {
        54, 50, 62, 56, 22, 34, 94, 96, 38, 6, 63, 93, 2, 18,123, 33,
        55,113, 39,114, 21, 67, 65, 12, 47, 73, 46, 27, 25,111,124, 81,
        53, 9,121, 79, 52, 60, 58, 48,101,127, 40,120,104, 70, 71, 43,
        20,122, 72, 61, 23,109, 13,100, 77, 1, 16, 7, 82, 10,105, 98,
        117,116, 76, 11, 89,106, 0,125,118, 99, 86, 69, 30, 57,126, 87,
        112, 51, 17, 5, 95, 14, 90, 84, 91, 8, 35,103, 32, 97, 28, 66,
        102, 31, 26, 45, 75, 4, 85, 92, 37, 74, 80, 49, 68, 29,115, 44,
        64,107,108, 24,110, 83, 36, 78, 42, 19, 15, 41, 88,119, 59, 3};
    static u16 S9[] = {
        167,239,161,379,391,334,  9,338, 38,226, 48,358,452,385, 90,397,
        183,253,147,331,415,340, 51,362,306,500,262, 82,216,159,356,177,
        175,241,489, 37,206, 17,  0,333, 44,254,378, 58,143,220, 81,400,
         95,  3,315,245, 54,235,218,405,472,264,172,494,371,290,399, 76,
        165,197,395,121,257,480,423,212,240, 28,462,176,406,507,288,223,
        501,407,249,265, 89,186,221,428,164, 74,440,196,458,421,350,163,
        232,158,134,354, 13,250,491,142,191, 69,193,425,152,227,366,135,
        344,300,276,242,437,320,113,278, 11,243, 87,317, 36, 93,496, 27,
        487,446,482, 41, 68,156,457,131,326,403,339, 20, 39,115,442,124,
        475,384,508, 53,112,170,479,151,126,169, 73,268,279,321,168,364,
        363,292, 46,499,393,327,324, 24,456,267,157,460,488,426,309,229,
        439,506,208,271,349,401,434,236, 16,209,359, 52, 56,120,199,277,
        465,416,252,287,246,  6, 83,305,420,345,153,502, 65, 61,244,282,
        173,222,418, 67,386,368,261,101,476,291,195,430, 49, 79,166,330,
        280,383,373,128,382,408,155,495,367,388,274,107,459,417, 62,454,
        132,225,203,316,234, 14,301, 91,503,286,424,211,347,307,140,374,
         35,103,125,427, 19,214,453,146,498,314,444,230,256,329,198,285,
         50,116, 78,410, 10,205,510,171,231, 45,139,467, 29, 86,505, 32,
         72, 26,342,150,313,490,431,238,411,325,149,473, 40,119,174,355,
        185,233,389, 71,448,273,372, 55,110,178,322, 12,469,392,369,190,
          1,109,375,137,181, 88, 75,308,260,484, 98,272,370,275,412,111,
        336,318,  4,504,492,259,304, 77,337,435, 21,357,303,332,483, 18,
         47, 85, 25,497,474,289,100,269,296,478,270,106, 31,104,433, 84,
        414,486,394, 96, 99,154,511,148,413,361,409,255,162,215,302,201,
        266,351,343,144,441,365,108,298,251, 34,182,509,138,210,335,133,
        311,352,328,141,396,346,123,319,450,281,429,228,443,481, 92,404,
        485,422,248,297, 23,213,130,466, 22,217,283, 70,294,360,419,127,
        312,377,  7,468,194,  2,117,295,463,258,224,447,247,187, 80,398,
        284,353,105,390,299,471,470,184, 57,200,348, 63,204,188, 33,451,
         97, 30,310,219, 94,160,129,493, 64,179,263,102,189,207,114,402,
        438,477,387,122,192, 42,381,  5,145,118,180,449,293,323,136,380,
         43, 66, 60,455,341,445,202,432, 8,237, 15,376,436,464, 59,461};

    /* The sixteen bit input is split into two unequal halves,  *
     * nine bits and seven bits - as is the subkey             */

    nine  = (u16)(in>>7);
    seven = (u16)(in&0x7F);

    /* Now run the various operations */

    nine  = (u16)(S9[nine]  ^ seven);
```

```
    seven = (u16)(S7[seven] ^ (nine & 0x7F));

    seven ^= (subkey>>9);
    nine  ^= (subkey&0x1FF);

    nine  = (u16)(S9[nine]  ^ seven);
    seven = (u16)(S7[seven] ^ (nine & 0x7F));

    in = (u16)((seven<<9) + nine);

    return( in );
}

 /*---------------------------------------------------------------------
 * FO()
 *      The FO() function.
 *      Transforms a 32-bit value.  Uses <index> to identify the
 *      appropriate subkeys to use.
 *---------------------------------------------------------------------*/
static u32 FO( u32 in, int index )
{
    u16 left, right;

    /* Split the input into two 16-bit words */

    left  = (u16)(in>>16);
    right = (u16) in;

    /* Now apply the same basic transformation three times         */

    left ^= KOi1[index];
    left  = FI( left, KIi1[index] );
    left ^= right;

    right ^= KOi2[index];
    right  = FI( right, KIi2[index] );
    right ^= left;

    left ^= KOi3[index];
    left  = FI( left, KIi3[index] );
    left ^= right;

    in = (((u32)right)<<16)+left;

    return( in );
}

/*---------------------------------------------------------------------
 * FL()
 *      The FL() function.
 *      Transforms a 32-bit value.  Uses <index> to identify the
 *      appropriate subkeys to use.
 *---------------------------------------------------------------------*/
static u32 FL( u32 in, int index )
{
    u16 l, r, a, b;

    /* split out the left and right halves */

    l = (u16)(in>>16);
    r = (u16)(in);

    /* do the FL() operations            */

    a  = (u16) (l & KLi1[index]);
    r ^= ROL16(a,1);
```

```
    b  = (u16)(r | KLi2[index]);
    l ^= ROL16(b,1);

    /* put the two halves back together */

    in = (((u32)l)<<16) + r;

    return( in );
}

 /*---------------------------------------------------------------------
 * Kasumi()
 *      the Main algorithm (fig 1).  Apply the same pair of operations
 *      four times.  Transforms the 64-bit input.
 *---------------------------------------------------------------------*/
void Kasumi( u8 *data )
{
    u32 left, right, temp;
    DWORD *d;
    int n;

    /* Start by getting the data into two 32-bit words (endian corect) */

    d = (DWORD*)data;
    left  = ((((u32)d[0].b8[0])<<24)+(((u32)d[0].b8[1])<<16)
+(d[0].b8[2]<<8)+(d[0].b8[3]);
    right = ((((u32)d[1].b8[0])<<24)+(((u32)d[1].b8[1])<<16)
+(d[1].b8[2]<<8)+(d[1].b8[3]);
    n = 0;
    do{ temp = FL( left, n   );
        temp = FO( temp,  n++ );
        right ^= temp;
        temp = FO( right, n   );
        temp = FL( temp,   n++ );
        left ^= temp;
    }while( n<=7 );

    /* return the correct endian result */
    d[0].b8[0] = (u8)(left>>24);     d[1].b8[0] = (u8)(right>>24);
    d[0].b8[1] = (u8)(left>>16);     d[1].b8[1] = (u8)(right>>16);
    d[0].b8[2] = (u8)(left>>8);      d[1].b8[2] = (u8)(right>>8);
    d[0].b8[3] = (u8)(left);         d[1].b8[3] = (u8)(right);
}

/*---------------------------------------------------------------------
 * KeySchedule()
 *      Build the key schedule.  Most "key" operations use 16-bit
 *      subkeys so we build u16-sized arrays that are "endian" correct.
 *---------------------------------------------------------------------*/
void KeySchedule( u8 *k )
{
    static u16 C[] = {
        0x0123,0x4567,0x89AB,0xCDEF, 0xFEDC,0xBA98,0x7654,0x3210 };
    u16 key[8], Kprime[8];
    WORD *k16;
    int n;

    /* Start by ensuring the subkeys are endian correct on a 16-bit basis */

    k16 = (WORD *)k;
    for( n=0; n<8; ++n )
        key[n] = (u16)((k16[n].b8[0]<<8) + (k16[n].b8[1]));

    /* Now build the K'[] keys */
```

```
    for( n=0; n<8; ++n )
        Kprime[n] = (u16)(key[n] ^ C[n]);

    /* Finally construct the various sub keys */

    for( n=0; n<8; ++n )
    {
        KLi1[n] = ROL16(key[n],1);
        KLi2[n] = Kprime[(n+2)&0x7];
        KOi1[n] = ROL16(key[(n+1)&0x7],5);
        KOi2[n] = ROL16(key[(n+5)&0x7],8);
        KOi3[n] = ROL16(key[(n+6)&0x7],13);
        KIi1[n] = Kprime[(n+4)&0x7];
        KIi2[n] = Kprime[(n+3)&0x7];
        KIi3[n] = Kprime[(n+7)&0x7];
    }
}
/*---------------------------------------------------------------------
 *              e n d    o f    k a s u m i . c
 *--------------------------------------------------------------------*/
```

kgcore.c

```
/*-----------------------------------------------------------
 *              KGCORE
 *-----------------------------------------------------------
 *
 *  A sample implementation of KGCORE, the heart of the
 *  A5/3 algorithm set.
 *
 *  This has been coded for clarity, not necessarily for
 *  efficiency.
 *
 *  This will compile and run correctly on both Intel
 *  (little endian) and Sparc (big endian) machines.
 *
 *  Version 0.1     13 March 2002
 *
 *-----------------------------------------------------------*/

#include "kasumi.h"
#include <stdio.h>


/*-----------------------------------------------------------
 * KGcore()
 *      Given ca, cb, cc, cd, ck, cl generate c0
 *-----------------------------------------------------------*/
void KGcore( u8 ca, u8 cb, u32 cc, u8 cd, u8 *ck, u8 *co, int cl )
{
    REGISTER64 A;        /* the modifier          */
    REGISTER64 temp; /* The working register */
    int i, n;
    u8  key[16],ModKey[16];      /* Modified key       */
    u16 blkcnt;          /* The block counter */

    /* Copy over the key */

    for( i=0; i<16; ++i )
        key[i] = ck[i];

    /* Start by building our global modifier */

    temp.b32[0]  = temp.b32[1]  = 0;
```

```
    A.b32[0]     = A.b32[1]     = 0;

    /* initialise register in an endian correct manner*/

    A.b8[0]  = (u8) (cc>>24);
    A.b8[1]  = (u8) (cc>>16);
    A.b8[2]  = (u8) (cc>>8);
    A.b8[3]  = (u8) (cc);
    A.b8[4]  = (u8) (cb<<3);
    A.b8[4] |= (u8) (cd<<2);
    A.b8[5]  = (u8) ca;

    /* Construct the modified key and then "kasumi" A */

    for( n=0; n<16; ++n )
        ModKey[n] = (u8)(ck[n] ^ 0x55);
    KeySchedule( ModKey );
    Kasumi( A.b8 );  /* First encryption to create modifier */

    /* Final initialisation steps */

    blkcnt = 0;
    KeySchedule( key );

    /* Now run the key stream generator */

    while( cl > 0 )
    {
        /* First we calculate the next 64-bits of keystream */

        /* XOR in A and BLKCNT to last value */

        temp.b32[0] ^= A.b32[0];
        temp.b32[1] ^= A.b32[1];
        temp.b8[7]  ^= blkcnt;

        /* KASUMI it to produce the next block of keystream */

        Kasumi( temp.b8 );

        /* Set <n> to the number of bytes of input data   *
         * we have to modify.  (=8 if length <= 64)      */

        if( cl >= 64 )
            n = 8;
        else
            n = (cl+7)/8;

        /* copy out the keystream */

        for( i=0; i<n; ++i )
            *co++ = temp.b8[i];
        cl -= 64;        /* done another 64 bits */
        ++blkcnt;        /* increment BLKCNT */
    }
}

/*-------------------------------------------------------------
 *          e n d    o f    K G c o r e . c
 *-------------------------------------------------------------*/
```

a53f.c

```
/*-------------------------------------------------------
```

```
 *              A5/3
 *-------------------------------------------------------
 *
 *  A sample implementation of A5/3, the functions of the
 *  A5/3 algorithm set.
 *
 *  This has been coded for clarity, not necessarily for
 *  efficiency.
 *
 *  This will compile and run correctly on both Intel
 *  (little endian) and Sparc (big endian) machines.
 *
 *  Version 0.1     13 March 2002
 *
 *-------------------------------------------------------*/

#include "kasumi.h"
#include <stdlib.h>

void KGcore( u8 ca, u8 cb, u32 cc, u8 cd, u8 *ck, u8 *co, int cl );

/*-------------------------------------------------------
 * BuildKey()
 *  The KGcore() function expects a 128-bit key.  This
 *  function builds that key from shorter length keys.
 *-------------------------------------------------------*/
static u8 *BuildKey( u8 *k, int len )
{
    static u8 ck[16];       /* Where the key is built */
    int i, n, sf;
    u8 mask[]={0x1,0x3,0x7,0xF,0x1F,0x3F,0x7F,0xFF};

    i = (len+7)/8;          /* Round to nearest byte  */
    if (i > 16 )
        i = 16;             /* limit to 128 bits   */
    for( n=0; n<i; ++n ) /* copy over the key      */
        ck[n] = k[n];
    sf = len%8;             /* Any odd key length?    */

    /* If the key is less than 128-bits we need to replicate *
     * it as many times as is necessary to fill the key.   */

    if( len < 128 )
    {
        n = 0;
        if( sf )/* Doesn't align to byte boundaries */
        {
            ck[i-1] &= mask[sf];
            ck[i-1] += ck[0]<<sf;
            while( i<16 )
            {
                ck[i] = (ck[n]>>(8-sf)) + (ck[n+1]<<sf);
                ++n;
                ++i;
            }
        }
        else
            while( i<16 )
                ck[i++] = ck[n++];
    }
    return( ck );
}


/*-------------------------------------------------------
 * The basic A5/3 functions.
```

```
 *  These follow a standard layout:
 *  - From the supplied key build the 128-bit required by
 *    KGcore()
 *  - Call the KGcore() function with the appropriate
 *    parameters
 *  - Take the generated Keystream and repackage it
 *    in the required format.
 */

/*-------------------------------------------------------
 * The standard GSM function
 *-------------------------------------------------------*/
void GSM( u8 *key, int klen, int count, u8 *block1, u8 *block2 )
{
    u8 *ck, data[32];
    int i;

    ck=BuildKey( key, klen );
    KGcore( 0x0F, 0, count, 0, ck, data, 228 );
    for( i=0; i<15; ++i )
    {
        block1[i] = data[i];
        block2[i] = (data[i+14]<<2) + (data[i+15]>>6);
    }
    block1[14] &= 0xC0;
    block2[14] &= 0xC0;
}

/*-------------------------------------------------------
 * The standard GSM ECSDDGE function
 *-------------------------------------------------------*/
void ECSDDGE( u8 *key, int klen, int count, u8 *block1, u8 *block2 )
{
    u8 *ck, data[87];
    int i;

    ck=BuildKey( key, klen );
    KGcore( 0xF0, 0, count, 0, ck, data, 696 );
    for( i=0; i<44; ++i )
    {
        block1[i] = data[i];
        block2[i] = (data[i+43]<<4) + (data[i+44]>>4);
    }
    block1[43] &= 0xF0;
    block2[43] &= 0xF0;
}

/*-------------------------------------------------------
 * The standard GEA3 function
 *-------------------------------------------------------*/
void GEA3( u8 *key, int klen, u32 input, u8 direction, u8 *block, int m )
{
    u8 *ck, *data;
    int i;


    data = malloc( m );
    ck=BuildKey( key, klen );
    KGcore( 0xFF, 0, input, direction, ck, data, m*8 );
    for( i=0; i<m; ++i )
        block[i] = data[i];
    free( data );
}

/*-------------------------------------------------------
 *  E n d  o f  A 5 3 f . c
```

```
 *-------------------------------------------------------*/
```

a53f.h

```
void GSM( u8 *key, int klen, int count, u8 *block1, u8 *block2 );
void ECSDDGE( u8 *key, int klen, int count, u8 *block1, u8 *block2 );
void GEA3( u8 *key, int klen, u32 input, u8 direction, u8 *block, int m );
```

3GPP TSG-SA3 Meeting #26
Oxford, England, 19-22 November

Tdoc ⌘S3-020658

CR-Form-v7

CHANGE REQUEST

⌘ **TS 55.217** **CR** **001** ⌘**rev** **-** ⌘ Current version: **6.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☒ Radio Access Network ☐ Core Network ☒

Title:	⌘	EGPRS algoritm
Source:	⌘	SA WG3
Work item code:	⌘	SEC1-CSALGO1
Date:	⌘	14/11/2002
Category:	⌘	**F**
Release:	⌘	REL-6

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
Rel-4 (Release 4)
Rel-5 (Release 5)
Rel-6 (Release 6)

Reason for change: ⌘ At SA3 #25 Ericsson presented a discussion paper in S3-020545 asking for clarification on the algorithm to be used for EGPRS.

The following extract has been taken from the SA3 #25 meeting report:

"TD S3-020545 A5/3 and GEA3 and their relation with EGPRS. This was introduced by Ericsson and questions the use of A5/3 for EDGE and the data-rate for EGPRS and asks SA WG3 to discuss the issues raised in order to provide any necessary CRs to the next SA WG3 meeting. It was confirmed that A5/3 and GEA3 were suitable for both GSM/GPRS and EDGE variants, the algorithm specifications are unclear on this: ***The modulation scheme used in the PS domain does not affect the GEA3 algorithm mechanism. A5/3 (CS domain) has 2 modes of use, GSM standard mode and GSM EDGE mode****. No CR to TS 43.020 was thought necessary, as implementers need to look at the algorithm specifications where the two modes of operation are clarified. It was agreed, however, to create a CR to the Technical Report TR 55.919 to clarify the use of the term "EDGE" in the specifications and the EGPRS bit-rates.* ***K. Boman agreed to do this for the next SA WG3 meeting****."*

It is proposed to change and clarify the wording in the Technical Specifications as well as TS 55.217.

Summary of change: ⌘ The term "EDGE" has been deleted from TS 55.217 as it very confusing i.e. the definition

	is unclear in 3GPP whether it applies for enhanced circuit-switched data or enhanced GPRS or both. The term ECSD has been introduced as it is defined in 21.905 Vocabulary for 3GPP Specifications and stands for enhanced circuit-switched data. The term EGPRS has been introduced as it is defined in 21.905 Vocabulary for 3GPP Specifications and stands for enhanced GPRS. It's been clarified that GEA3 shall be used for EGPRS.
Consequences if not approved: ⌘	It's unclear whether: - the term EDGE means enhanced circuit-switched data or enhanced GPRS or both; - what algorithm that shall be used for EGPRS.

Clauses affected:	⌘				
		Y	**N**		
Other specs affected:	⌘	**X**		Other core specifications	⌘ 55.216
		X		Test specifications	55.218
			X	O&M Specifications	
Other comments:	⌘	SAGE draft 1.0 (technically equivalent to SA#17 approved version 1.0.0) is used for this CR, as electronic versions of 3GPP specification is not available on the 3GPP FTP site at present.			

ETSI/SAGE Specification	**Version: 1.0** **Date: 27th May 2002**

Specification of the A5/3 Encryption Algorithms for GSM and ECSD~~DGE~~, and the GEA3 Encryption Algorithm for GPRS

Document 2: Implementors' Test Data

Document History		
1.0	**27th May 2002**	**Initial version**

PREFACE

This specification has been prepared by the 3GPP Task Force, and gives a detailed specification of the **A5/3** encryption algorithms for GSM and ECSD~~DGE~~, and of the **GEA3** encryption algorithm for GPRS.

This document is the second of three, which between them form the entire specification of the **A5/3** and **GEA3** algorithms:

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 1: **A5/3** and **GEA3** Specifications.

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 2: Implementors' Test Data.

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 3: Design Conformance Test Data.

The normative part of the specification of the **A5/3** and **GEA3** algorithms is in the main body of Document 1. The annexes to this document are purely informative.

Documents 2 (this document) and 3 are also purely informative.

The normative part of the specification of the block cipher (**KASUMI)** on which the **A5/3** and **GEA3** algorithms are based can be found in [5].

Blank Page

TABLE OF CONTENTS

REFERENCES

[1] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 1: **A5/3** and **GEA3** Specifications.

[2] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 2: Implementors' Test Data.

[3] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 3: Design Conformance Test Data.

[4] 3GPP TS 35.201: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 1: *f8* and *f9* specifications, V4.1.0

[5] 3GPP TS 35.202: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 2: **KASUMI** specification, V4.0.0

[6] 3GPP TS 35.203: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 3: Implementors' Test Data, V4.0.0

1. OUTLINE OF THE IMPLEMENTORS' TEST DATA

Section 2 introduces the algorithm and describes the notation used in the subsequent sections.

Section 3 provides test data for the encryption algorithm A5/3 for GSM.

Section 4 provides test data for the encryption algorithm A5/3 for ECSD~~DGE~~.

Section 5 provides test data for the encryption algorithm GEA3 for GPRS.

2. INTRODUCTORY INFORMATION

2.1. Introduction

In this document the implementors' test data are given for three ciphering algorithms: **A5/3** for GSM, **A5/3** for ECSD~~DGE~~, and **GEA3** for GPRS (including EGPRS). The algorithms are stream ciphers that are used to encrypt/decrypt blocks of data under a confidentiality key $\mathbf{K_C}$. Each of these algorithms is based on the **KASUMI** algorithm that is specified in reference [5]. **KASUMI** is a block cipher that produces a 64-bit output from a 64-bit input under the control of a 128-bit key. The algorithms defined in [1] use **KASUMI** in a form of output-feedback mode as a keystream generator.

The three algorithms are all very similar. In Document 1 they are specified in terms of a core function **KGCORE** (section 3); The implementors' test data will reflect this as it will show the input/output data of **KGCORE** as well as important steps in the calculation inside **KGCORE**.

2.2. Notation

2.2.1. Radix

We use the prefix **0x** to indicate **hexadecimal** numbers.

2.2.2. Conventions

We use the assignment operator '=', as used in several programming languages. When we write

<*variable*> = <*expression*>

we mean that <*variable*> assumes the value that <*expression*> had before the assignment took place. For instance,

$$x = x + y + 3$$

means

(new value of x) becomes (old value of x) + (old value of y) + 3.

2.2.3. Bit/Byte ordering

All data variables in this specification are presented with the most significant bit (or byte) on the left hand side and the least significant bit (or byte) on the right hand side. Where a variable is broken down into a number of sub-strings, the left most (most significant) sub-string is numbered 0, the next most significant is numbered 1 and so on through to the least significant.

For example an n-bit **STRING** is subdivided into 64-bit substrings $\mathbf{SB_0,SB_1…SB_i}$ so if we have a string:

0x0123456789ABCDEFFEDCBA98765432l086545381AB594FC28786404C50A37…

we have:

$\mathbf{SB_0}$ = 0x0123456789ABCDEF
$\mathbf{SB_1}$ = 0xFEDCBA9876543210
$\mathbf{SB_2}$ = 0x86545381AB594FC2
$\mathbf{SB_3}$ = 0x8786404C50A37…

In binary this would be:

00000001001000110100010101100111100010011010101111001101111101111111111110…

with $\mathbf{SB_0}$ = 00000001001000110100010101100111100010011010101111001101111101111
$\mathbf{SB_1}$ = 11111110110111001011101010011000011101100101010000011001000010000
$\mathbf{SB_2}$ = 10000110010101000101001110000001101010110101100101001111111000010
$\mathbf{SB_3}$ = 10000111100001100100000001001100010100001010001101l1…

2.2.4. Presentation of input/output data

The basic data processed by the algorithm A5/3 are blocks of two times 114 bits (GSM) resp. 348 bits (ECSD~~DGE~~). In general in this document the data is presented in hexadecimal format as bytes, thus the last byte shown as part of an input or output data block may include 0 to 6 bits that are ignored once the block size has been reached (the least significant bits of the byte are ignored).

2.3. List of Variables

A	a 64-bit register that is used within the **KGCORE** function to hold an intermediate value.
BLKCNT	a 64-bit counter used in the **KGCORE** function.
BLOCK1	a string of keystream bits output by the **A5/3** algorithm — 114 bits for GSM, 348 bits for ECSD~~DGE~~.
BLOCK2	a string of keystream bits output by the **A5/3** algorithm — 114 bits for GSM, 348 bits for ECSD~~DGE~~.
CA	an 8-bit input to the **KGCORE** function.
CB	a 5-bit input to the **KGCORE** function.
CC	a 32-bit input to the **KGCORE** function.
CD	a 1-bit input to the **KGCORE** function.
CE	a 16-bit input to the **KGCORE** function reserved for future use. This input will not be shown in this document.
CK	a 128-bit input to the **KGCORE** function.
CL	an integer input to the **KGCORE** function, in the range $1…2^{19}$ inclusive, specifying the number of output bits for **KGCORE** to produce.
COUNT	a 22-bit frame dependent input to both the GSM and ECSD~~DGE~~ **A5/3** algorithms.

DIRECTION	a 1-bit input to the **GEA3** algorithm, indicating the direction of transmission (uplink or downlink).
INPUT	a 32-bit frame dependent input to the **GEA3** algorithm.
K_C	the cipher key that is an input to each of the three cipher algorithms defined here. Although at the time of writing the standards specify that K_C is 64 bits long, the algorithm specifications here allow it to be of any length between 64 and 128 inclusive, to allow for possible future enhancements to the standards.
KLEN	the length of K_C in bits, between 64 and 128 inclusive (see above).
M	an input to the **GEA3** algorithm, specifying the number of octets of output to produce.
OUTPUT	the stream of output octets from the **GEA3** algorithm.

~~4.~~3. ALGORITHM A5/3 FOR GSM

~~4.1.~~3.1. Overview

The test data sets presented here are for the algorithm A5/3 for GSM. No detailed data of internal states of Kasumi are presented here as these are covered in section 3 of document [6].

For GSM, the DIRECTION bit is not applicable and is set to zero. The COUNT variable is 22 bits in length.

~~4.2.~~3.2. Format

Each test set starts by showing the various inputs to the function and their mapping to KGCORE inputs. Thereafter the input/output values of the initial KASUMI operation are shown. Finally the feedback and the resulting keystream block is shown in a table.

The first test set will also give the results in binary format to explicitly show the relationship between the hexadecimal and binary format.

~~4.3.~~3.3. Test Set 1

Input A5/3	
KLEN	64 Bits
Kc	0x2BD6459F82C5BC00
COUNT	0x24F20F
Input KGC	
Key	0x2BD6459F82C5BC002BD6459F82C5BC00
CA	0x0F
CB	0
CC	0x0024F20F
CD	0
CL	228

Init		
Initial A	0x0024F20F000F0000	
Key used	0x7E8310CAD790E9557E8310CAD790E955	
Modified A	0x91B5F53F0EFCA154	
Key now	0x2BD6459F82C5BC002BD6459F82C5BC00	
Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x91B5F53F0EFCA154	0x889EEAAF9ED1BA1A
1	0x192B1F90902D1B4F	0xBBD8436232E45728
2	0x2A6DB65D3C18F67E	0xD01AA89133DA73C1
3	0x41AF5DAE3D26D296	0x1EAB68B7D89BC841

BLOCK1: 0x889EEAAF9ED1BA1ABBD8436232E440

100010001001111011101010101011111001111011010001101110100001101010111011100001000011011000100011001011100100011

BLOCK2: 0x5CA3406AA244CF69CF047AADA2DF40

010111001010001101000000011010101010001001000100110011110110100111001111000001000111101010110110100010110111110

4.4.3.4. Test Set 2

Input A5/3		
KLEN	64 Bits	
Kc	0x952C49104881FF48	
COUNT	0x061272	
Input KGC		
Key	0x952C49104881FF48952C49104881FF48	
CA	0x0F	
CB	0	
CC	0x00061272	
CD	0	
CL	228	
Init		
Initial A	0x00061272000F0000	
Key used	0xC0791C451DD4AA1DC0791C451DD4AA1D	
Modified A	0x3E6A82C79F192DC7	
Key now	0x952C49104881FF48952C49104881FF48	
Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x3E6A82C79F192DC7	0xFB4D5FBCEE13A333
1	0xC527DD7B710A8EF5	0x89285686E9A5C942
2	0xB742D44176BCE487	0x40DE38150115F15F
3	0x7EB4BAD29E0CDC9B	0x8D9D98B91A94B296

BLOCK1: 0xFB4D5FBCEE13A33389285686E9A5C0

BLOCK2: 0x25090378E0540457C57E367662E440

~~4.5.~~3.5. Test Set 3

Input A5/3		
KLEN	64 Bits	
Kc	0xEFA8B2229E720C2A	
COUNT	0x33FD3F	
Input KGC		
Key	0xEFA8B2229E720C2AEFA8B2229E720C2A	
CA	0x0F	
CB	0	
CC	0x0033FD3F	
CD	0	
CL	228	
Init		
Initial A	0x0033FD3F000F0000	
Key used	0xBAFDE777CB27597FBAFDE777CB27597F	
Modified A	0x46C50F2C98B65D25	
Key now	0xEFA8B2229E720C2AEFA8B2229E720C2A	
Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x46C50F2C98B65D25	0x0E4015755A336469
1	0x48851A59C285394D	0xC3DD8680E3035BC4
2	0x851889AC7BB506E3	0x19A78AD3862C1090
3	0x5F6285FF1E9A4DB6	0xC68A391FE8A6ADEB

BLOCK1: 0x0E4015755A336469C3DD8680E30340

BLOCK2: 0x6F10669E2B4E18B042431A28E47F80

~~4.6.~~3.6. Test Set 4

Input A5/3	
KLEN	80 Bits
Kc	0x5ACB1D644C0D51204EA5
COUNT	0x156B26
Input KGC	
Key	0x5ACB1D644C0D51204EA55ACB1D644C0D
CA	0x0F
CB	0
CC	0x00156B26
CD	0
CL	228
Init	
Initial A	0x00156B26000F0000
Key used	0x0F9E4831195804751BF00F9E48311958
Modified A	0x3071F1EC67B203EB
Key now	0x5ACB1D644C0D51204EA55ACB1D644C0D

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x3071F1EC67B203EB	0xE095306AD5086E2E
1	0xD0E4C186B2BA6DC4	0xAC7F3107DE4FA22D
2	0x9C0EC0EBB9FDA1C4	0xC1DFC97D5BC5661D
3	0xF1AE38913C7765F5	0xD6096F476AEDC64B

BLOCK1: 0xE095306AD5086E2EAC7F3107DE4F80

BLOCK2: 0x88B7077F25F56F1598775825BD1D80

~~4.7.~~3.7. Test Set 5

Input A5/3		
KLEN	128 Bits	
Kc	0xD3C5D592327FB11C4035C6680AF8C6D1	
COUNT	0x0A59B4	
Input KGC		
Key	0xD3C5D592327FB11C4035C6680AF8C6D1	
CA	0x0F	
CB	0	
CC	0x000A59B4	
CD	0	
CL	228	
Init		
Initial A	0x000A59B4000F0000	
Key used	0x869080C7672AE4491560933D5FAD9384	
Modified A	0x0CAAEC5C175B5A03	
Key now	0xD3C5D592327FB11C4035C6680AF8C6D1	
Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x0CAAEC5C175B5A03	0xDCE64362AB5F89C1
1	0xD04CAF3EBC04D3C3	0x1EF0B305166570F4
2	0x125A5F59013E2AF5	0x889D5511E9E3575D
3	0x8437B94DFEB80D5D	0x062B5CED6039506A

BLOCK1: 0xDCE64362AB5F89C11EF0B305166540

BLOCK2: 0xC3D222755447A78D5D7418AD73B580

~~5.~~4. ALGORITHM A5/3 FOR ECSD~~DGE~~

~~5.1.~~4.1. Overview

The test data sets presented here are for the algorithm A5/3 for ECSD~~DGE~~. No detailed data of the internal states of Kasumi are presented here as these are covered in section 3 of document [6].

For ECSD~~DGE~~, the DIRECTION bit is not applicable and is set to zero. The COUNT variable is 22 bits in length. ECSD~~DGE~~ allows block sizes up to 348 bits for BLOCK1 and BLOCK2. As A5/3 for ECSD~~DGE~~ always produces two times 348 bits, the superfluous bits of each output block have to be discarded.

~~5.2.~~4.2. Format

Each test starts by showing the various inputs to the function and their mapping to KGCORE inputs. Thereafter the input/output values of the initial KASUMI operation are shown. Finally the feedback and the resulting keystream block is shown in a table.

The first test set will also give the results in binary format to explicitly show the relationship between the hexadecimal and binary format.

~~5.3.~~4.3. Test Set 1

Input A5/3	
KLEN	64 Bits
Kc	0x2BD6459F82C5BC00
COUNT	0x24F20F
Input KGC	
Key	0x2BD6459F82C5BC002BD6459F82C5BC00
CA	0xF0
CB	0
CC	0x0024F20F
CD	0
CL	696
Init	
Initial A	0x0024F20F00F00000
Key used	0x7E8310CAD790E9557E8310CAD790E955
Modified A	0xF3B3A9E4CDB3EA39
Key now	0x2BD6459F82C5BC002BD6459F82C5BC00

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0xF3B3A9E4CDB3EA39	0xF75E663ACEA21EC9
1	0x04EDCFDE0311F4F1	0xD0BDE98B6C33B819
2	0x230E406FA1805222	0x299E830A1A2E2F91
3	0xDA2D2AEED79DC5AB	0x4326BEF515089B6D
4	0xB0951711D8BB7150	0xB0F271AFB9609F90
5	0x4341D84B74D375AC	0x5202CDCF51426D17
6	0xA1B1642B9CF18728	0x2DB47BFED3E6D83D
7	0xDE07D21A1E553203	0x14F4876366CCCD5B
8	0xE7472E87AB7F276A	0xFAE85B27C9B49F2F
9	0x095BF2C30407751F	0x7775B0B504905F27
10	0x84C61951C923B514	0xB5AE62B8269EA9BB

BLOCK1:
0xF75E663ACEA21EC9D0BDE98B6C33B819299E830A1A2E2F914326BEF51508
9B6DB0F271AFB9609F905202CDC0

```
11110111010111100110011000111010110011101010001000011110110010
01110100001011110111101001100010110110110000110011101110000001
10010010100110011110100000110000101000011010001011100010111110
01000101000011001001101011111011110101000101010000100010011011
01101101101100001111001001110001101011111011100101100000100111
11100100000101001000000010110011011100
```

BLOCK2:
0xF51426D172DB47BFED3E6D83D14F4876366CCCD5BFAE85B27C9B49F2F777
5B0B504905F27B5AE62B8269EA90

11110101000101000010011011010001011100101101101101000111101111
11111011010011111001101101100000111101000101001111010010000111
01100011011001101100110011001101010111111101011101000010110
11001001111100100110110100100111110010111101110111011101011011
00001011010100000100100100000101111100100111101101011010111001
10001010111000001001101001111010101001

5.4.4.4. Test Set 2

Input A5/3	
KLEN	64 Bits
Kc	0x952C49104881FF48
COUNT	0x061272
Input KGC	
Key	0x952C49104881FF48952C49104881FF48
CA	0xF0
CB	0
CC	0x00061272
CD	0
CL	696
Init	
Initial A	0x0006127200F00000
Key used	0xC0791C451DD4AA1DC0791C451DD4AA1D
Modified A	0x50736152B21A65A1
Key now	0x952C49104881FF48952C49104881FF48

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x50736152B21A65A1	0xE1876AA5B250B2B8
1	0xB1F40BF7004AD718	0xD58ADE52844E84E1
2	0x85F9BF003654E142	0x09A38FF6A87FCC7B
3	0x59D0EEA41A65A9D9	0x72FC8387494086DB
4	0x228FE2D5FB5AE37E	0xA2D2A1EE189DB569
5	0xF2A1C0BCAA87D0CD	0xA9245157CDD323EA
6	0xF95730057FC9464D	0x3518270A162C054E
7	0x656B4658A43660E8	0x120F5C703AE0AB32
8	0x427C3D2288FACE9B	0x4498D40D56268745
9	0x14EBB55FE43CE2ED	0xC41BC58D71DD255C
10	0x9468A4DFC3C740F7	0xCAC6BDA3B244397E

BLOCK1:
0xE1876AA5B250B2B8D58ADE52844E84E109A38FF6A87FCC7B72FC83874940
86DBA2D2A1EE189DB569A9245150

BLOCK2:
0x7CDD323EA3518270A162C054E120F5C703AE0AB324498D40D56268745C41
BC58D71DD255CCAC6BDA3B244390

~~5.5.~~4.5. Test Set 3

Input A5/3		
KLEN	64	
Kc	0xEFA8B2229E720C2A	
COUNT	0x33FD3F	
Input KGC		
Key	0xEFA8B2229E720C2AEFA8B2229E720C2A	
CA	0xF0	
CB	0	
CC	0x0033FD3F	
CD	0	
CL	696	
Init		
Initial A	0x0033FD3F00F00000	
Key used	0xBAFDE777CB27597FBAFDE777CB27597F	
Modified A	0x0950A4B18725EF90	
Key now	0xEFA8B2229E720C2AEFA8B2229E720C2A	
Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x0950A4B18725EF90	0x09B49CE620E4A36B
1	0x00E43857A7C14CFA	0x7956186C8F248B61
2	0x7006BCDD080164F3	0x50DC2362B3F41F6F
3	0x598C87D334D1F0FC	0x28F486D9A80BB879
4	0x21A422682F2E57ED	0xDA4FE349E72EF975
5	0xD31F47F8600B16E0	0x5A5015902B17EE1D
6	0x5300B121AC32018B	0xF32D9302567E470E
7	0xFA7D37B3D15BA899	0xA3A26B0FFCDE60DF
8	0xAAF2CFBE7BFB8F47	0xB8A28C10609AEC74
9	0xB1F228A1E7BF03ED	0xCA1EEDF3BAA3334C
10	0xC34E49423D86DCD6	0x28E7E4DDA38A4AEE

BLOCK1:
0x09B49CE620E4A36B7956186C8F248B6150DC2362B3F41F6F28F486D9A80B
B879DA4FE349E72EF9755A501590

BLOCK2:
0x02B17EE1DF32D9302567E470EA3A26B0FFCDE60DFB8A28C10609AEC74CA1
EEDF3BAA3334C28E7E4DDA38A4A0

~~5.6.~~4.6. Test Set 4

Input A5/3	
KLEN	80
Kc	0x5ACB1D644C0D51204EA5
COUNT	0x156B26

Input KGC	
Key	0x5ACB1D644C0D51204EA55ACB1D644C0D
CA	0xF0
CB	0
CC	0x00156B26
CD	0
CL	696
Init	
Initial A	0x00156B2600F00000
Key used	0x0F9E4831195804751BF00F9E48311958
Modified A	0x4653A8F67463F66B
Key now	0x5ACB1D644C0D51204EA55ACB1D644C0D

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x4653A8F67463F66B	0x0E5874E1CB66E2C0
1	0x480BDC17BF0514AA	0x438D49C5744C966F
2	0x05DEE133002F6006	0x001845E736CF365E
3	0x464BED1142ACC036	0xF842FAFFE4D28733
4	0xBE11520990B1715C	0x7C4A5FE5ACB03A39
5	0x3A19F713D8D3CC57	0x52810E30BB4AEDC7
6	0x14D2A6C6CF291BAA	0xA066C740F8D8A862
7	0xE6356FB68CBB5E0E	0x157E2A18F7E35173
8	0x532D82EE8380A710	0x6DF7CC7728172CA1
9	0x2BA464815C74DAC3	0xFE2736DE2DB08FBE
10	0xB8749E2859D379DF	0x14A0FF88C1AF790A

BLOCK1:
0x0E5874E1CB66E2C0438D49C5744C966F001845E736CF365EF842FAFFE4D2
87337C4A5FE5ACB03A3952810E30

BLOCK2:
0x0BB4AEDC7A066C740F8D8A862157E2A18F7E351736DF7CC7728172CA1FE2
736DE2DB08FBE14A0FF88C1AF790

~~5.7.~~4.7. Test Set 5

Input A5/3	
KLEN	128
Kc	0xD3C5D592327FB11C4035C6680AF8C6D1
COUNT	0x0A59B4
Input KGC	
Key	0xD3C5D592327FB11C4035C6680AF8C6D1
CA	0xF0
CB	0
CC	0x000A59B4
CD	0
CL	696
Init	
Initial A	0x000A59B400F00000
Key used	0x869080C7672AE4491560933D5FAD9384
Modified A	0x4278E040B3401A23
Key now	0xD3C5D592327FB11C4035C6680AF8C6D1

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x4278E040B3401A23	0x9887368E48257E17
1	0xDAFFD6CEFB656435	0x2EFF14BABC114DB5
2	0x6C87F4FA0F515794	0x159C2E3D0521AFCD
3	0x57E4CE7DB661B5ED	0x04487995989C35F8
4	0x463099D52BDC2FDF	0xF26C005D4CBDE2E3
5	0xB014E01DFFFDF8C5	0x9F18572D6DD0D2FA
6	0xDD60B76DDE90C8DF	0x85CF7C5FF04CDEC1
7	0xC7B79C1F430CC4E5	0x2318F01D418F4BBB
8	0x6160105DF2CF5190	0xF5FAFF8DFEA66C92
9	0xB7821FCD4DE676B8	0x47A8F64DEBF71A36
10	0x05D0160D58B7001F	0x4FBB36F39161ECB9

BLOCK1:
0x9887368E48257E172EFF14BABC114DB5159C2E3D0521AFCD04487995989C35F8F26C005D4CBDE2E39F185720

BLOCK2:
0xD6DD0D2FA85CF7C5FF04CDEC12318F01D418F4BBBF5FAFF8DFEA66C9247A8F64DEBF71A364FBB36F39161EC0

~~6.~~5. ALGORITHM GEA3 FOR GPRS

~~6.1.~~5.1. Overview

The test data sets presented here are for the algorithm GEA3 for GPRS. No detailed data of the internal states of Kasumi are presented here as these are covered in section 3 of document [6].

~~6.2.~~5.2. Format

Each test starts by showing the various inputs to the function and their mapping to KGCORE inputs. Thereafter the input/output values of the initial KASUMI operation are shown. Finally the feedback and the resulting keystream block is shown in a table.

The first test set will also give the results in binary format to explicitly show the relationship between the hexadecimal and binary format.

~~6.3.~~5.3. Test Set 1

Input GEA3	
KLEN	64 Bits
Kc	0x2BD6459F82C5BC00
INPUT	0x5124F20F
DIRECTION	1
M	51

Input KGC	
Key	0x2BD6459F82C5BC002BD6459F82C5BC00
CA	0xFF
CB	0
CC	0x5124F20F
CD	1
CL	408
Init	
Initial A	0x5124F20F04FF0000
Key used	0x7E8310CAD790E9557E8310CAD790E955
Modified A	0xDF1F9CA88B8F3248
Key now	0x2BD6459F82C5BC002BD6459F82C5BC00

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0xDF1F9CA88B8F3248	0xF0270AAF26851D2A
1	0x2F389607AD0A2F63	0x4E88CC48CBFC740D
2	0x919750E040734647	0x94ACAB8495D27A7E
3	0x4BB3372C1E5D4835	0x154F5DA9E991EF8A
4	0xCA50C101621EDDC6	0x4198C7369655E5B9
5	0x9E875B9E1DDAD7F4	0x72DA2B05CF4CD394
6	0xADC5B7AD44C3E1DA	0xB132EBAD0F8F793E

OUTPUT:
0xF0270AAF26851D2A4E88CC48CBFC740D94ACAB8495D27A7E154F5DA9E991
EF8A4198C7369655E5B972DA2B05CF4CD394B132EB

```
11110000001001110000101010101111001001101000010100011101001010
10010011101000100011001100010010001100101111111100011101000000
11011001010010101100101010111000010010010101110100100111101001
11111000010101010011110101110110101001111010011001000111101111
10001010010000011001100011000111001101101001011001010101111001
01101110010111001011011010001010110000010111001111010011001101
00111001010010110001001100101110101 1
```

6.4.5.4. Test Set 2

Input GEA3	
KLEN	64 Bits
Kc	0x952C49104881FF48
INPUT	0xD3861272
DIRECTION	0
M	51
Input KGC	
Key	0x952C49104881FF48952C49104881FF48
CA	0xFF
CB	0
CC	0xD3861272
CD	0
CL	408

Init	
Initial A	0xD386127200FF0000
Key used	0xC0791C451DD4AA1DC0791C451DD4AA1D
Modified A	0x619D4068C2D10D43
Key now	0x952C49104881FF48952C49104881FF48

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x619D4068C2D10D43	0x9B7B516B15FB65E2
1	0xFAE61103D72A68A0	0x83B722DBE3A2CFCB
2	0xE22A62B32173C28A	0x0B255CFB38D529B9
3	0x6AB81C93FA0424F9	0x61BC04129D5C6565
4	0x0021447A5F8D6822	0xAA25C31E63D10A04
5	0xCBB88376A1000742	0x8191BC1F17E67ECA
6	0xE00CFC77D537738F	0xAA509A78B1A8ABEF

OUTPUT:
0x9B7B516B15FB65E283B722DBE3A2CFCB0B255CFB38D529B961BC04129D5C
6565AA25C31E63D10A048191BC1F17E67ECAAA509A

~~6.5.~~5.5.Test Set 3

Input GEA3	
KLEN	64 Bits
Kc	0xEFA8B2229E720C2A
INPUT	0x4AB3FD3F
DIRECTION	0
M	51

Input KGC	
Key	0xEFA8B2229E720C2AEFA8B2229E720C2A
CA	0xFF
CB	0
CC	0x4AB3FD3F
CD	0
CL	408

Init	
Initial A	0x4AB3FD3F00FF0000
Key used	0xBAFDE777CB27597FBAFDE777CB27597F
Modified A	0x3A06D4477FA3DF28
Key now	0xEFA8B2229E720C2AEFA8B2229E720C2A

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x3A06D4477FA3DF28	0x0306B1F1E6286F27
1	0x390065B6998BB00E	0x148FF4F081164EA3
2	0x2E8920B7FEB59189	0x05E3296121F56491
3	0x3FE5FD265E56BBBA	0xA3BBEAB48EF824B3
4	0x99BD3EF3F15BFB9F	0x64D304946DCA4677
5	0x5ED5D0D31269995A	0x3F3A548642C68545
6	0x053C80C13D655A6B	0xC0FEE047AA50EBDF

OUTPUT:
0x0306B1F1E6286F27148FF4F081164EA305E3296121F56491A3BBEAB48EF8
24B364D304946DCA46773F3A548642C68545C0FEE0

~~6.6.~~5.6. Test Set 4

Input GEA3		
KLEN	80 Bits	
Kc	0x5ACB1D644C0D51204EA5	
INPUT	0xA1056B26	
DIRECTION	1	
M	51	
Input KGC		
Key	0x5ACB1D644C0D51204EA55ACB1D644C0D	
CA	0xFF	
CB	0	
CC	0xA1056B26	
CD	1	
CL	408	
Init		
Initial A	0xA1056B2604FF0000	
Key used	0x0F9E4831195804751BF00F9E48311958	
Modified A	0xA4B67C463FF7B4D5	
Key now	0x5ACB1D644C0D51204EA55ACB1D644C0D	
Feedback		
BLKCNT	KASUMI Input	Keystream
0	0xA4B67C463FF7B4D5	0xAA7906987B717A55
1	0x0ECF7ADE4486CE81	0xD58DA45465C74030
2	0x713BD8125A30F4E7	0xDA5AB70DAD711ECA
3	0x7EECCB4B9286AA1C	0x119EE76FFAA8D228
4	0xB5289B29C55F66F9	0x7EEE1314CD5E5333
5	0xDA586F52F2A9E7E3	0xCAB78DF6DACE4B86
6	0x6E01F1B0E539FF55	0x2814AD2CB44D715F

OUTPUT:
0xAA7906987B717A55D58DA45465C74030DA5AB70DAD711ECA119EE76FFAA8
D2287EEE1314CD5E5333CAB78DF6DACE4B862814AD

~~6.7.~~5.7. Test Set 5

Input GEA3	
KLEN	128 Bits
Kc	0xD3C5D592327FB11C4035C6680AF8C6D1
INPUT	0x0A3A59B4
DIRECTION	0
M	51
Input KGC	
Key	0xD3C5D592327FB11C4035C6680AF8C6D1
CA	0xFF
CB	0
CC	0x0A3A59B4
CD	0
CL	408

Init	
Initial A	0x0A3A59B400FF0000
Key used	0x869080C7672AE4491560933D5FAD9384
Modified A	0x5AEE7A85947015B1
Key now	0xD3C5D592327FB11C4035C6680AF8C6D1

Feedback		
BLKCNT	KASUMI Input	Keystream
0	0x5AEE7A85947015B1	0x6E217CE41EBEFB5E
1	0x34CF06618ACEEEEE	0xC8094C1597429006
2	0x92E73690033285B5	0x5E42BABC9AE35654
3	0x04ACC0390E9343E6	0xA53085CE68DFA442
4	0xFFDEFF4BFCAFB1F7	0x6A2FF0AD4AF33410
5	0x30C18A28DE8321A4	0x06A3F84B7613ACB4
6	0x5C4D82CEE263B903	0xFBDC342DCFF787DA

OUTPUT:
0x6E217CE41EBEFB5EC8094C15974290065E42BABC9AE35654A53085CE68DF
A4426A2FF0AD4AF3341006A3F84B7613ACB4FBDC34

3GPP TSG-SA3 Meeting #26
Oxford, England, 19-22 November

Tdoc ⌘S3-020659

CR-Form-v7

CHANGE REQUEST

⌘ **TS 55.218** **CR** **001** ⌘**rev** **-** ⌘ Current version: **6.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ [ ] ME [X] Radio Access Network [ ] Core Network [X]

Title:	⌘	EGPRS algoritm
Source:	⌘	SA WG3
Work item code:	⌘	SEC1-CSALGO1
Date:	⌘	14/11/2002
Category:	⌘	**F**
Release:	⌘	REL-6

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
Rel-4 (Release 4)
Rel-5 (Release 5)
Rel-6 (Release 6)

Reason for change: ⌘ At SA3 #25 Ericsson presented a discussion paper in S3-020545 asking for clarification on the algorithm to be used for EGPRS.

The following extract has been taken from the SA3 #25 meeting report:

"TD S3-020545 A5/3 and GEA3 and their relation with EGPRS. This was introduced by Ericsson and questions the use of A5/3 for EDGE and the data-rate for EGPRS and asks SA WG3 to discuss the issues raised in order to provide any necessary CRs to the next SA WG3 meeting. It was confirmed that A5/3 and GEA3 were suitable for both GSM/GPRS and EDGE variants, the algorithm specifications are unclear on this: ***The modulation scheme used in the PS domain does not affect the GEA3 algorithm mechanism. A5/3 (CS domain) has 2 modes of use, GSM standard mode and GSM EDGE mode****. No CR to TS 43.020 was thought necessary, as implementers need to look at the algorithm specifications where the two modes of operation are clarified. It was agreed, however, to create a CR to the Technical Report TR 55.919 to clarify the use of the term "EDGE" in the specifications and the EGPRS bit-rates.* ***K. Boman agreed to do this for the next SA WG3 meeting****."*

It is proposed to change and clarify the wording in the Technical Specifications as well as TS 55.218.

Summary of change: ⌘ The term "EDGE" has been deleted from TS 55.218 as it very confusing i.e. the definition

	is unclear in 3GPP whether it applies for enhanced circuit-switched data or enhanced GPRS or both. The term ECSD has been introduced as it is defined in 21.905 Vocabulary for 3GPP Specifications and stands for enhanced circuit-switched data. The term EGPRS has been introduced as it is defined in 21.905 Vocabulary for 3GPP Specifications and stands for enhanced GPRS. It's been clarified that GEA3 shall be used for EGPRS.
Consequences if not approved: ⌘	It's unclear whether: - the term EDGE means enhanced circuit-switched data or enhanced GPRS or both; - what algorithm that shall be used for EGPRS.

Clauses affected:	⌘				
		Y	**N**		
Other specs affected:	⌘	**X**		Other core specifications	⌘ 55.216
		X		Test specifications	55.217
			X	O&M Specifications	
Other comments:	⌘	SAGE draft 1.0 (technically equivalent to SA#17 approved version 1.0.0) is used for this CR, as electronic versions of 3GPP specification is not available on the 3GPP FTP site at present.			

ETSI/SAGE	**Version: 1.0**
Specification	**Date: 27th May 2002**

Specification of the A5/3 Encryption Algorithms for GSM and ECSD~~DGE~~, and the GEA3 Encryption Algorithm for GPRS

Document 3: Design Conformance Test Data

Document History		
1.0	**27th May 2002**	**Initial Version**

PREFACE

This specification has been prepared by the 3GPP Task Force, and gives a detailed specification of the **A5/3** encryption algorithms for GSM and ECSD~~DGE~~, and of the **GEA3** encryption algorithm for GPRS.

This document is the third of three, which between them form the entire specification of the **A5/3** and **GEA3** algorithms:

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 1: **A5/3** and **GEA3** Specifications.

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 2: Implementors' Test Data.

- Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DGE~~, and the **GEA3** Encryption Algorithm for GPRS.
  Document 3: Design Conformance Test Data.

The normative part of the specification of the **A5/3** and **GEA3** algorithms is in the main body of Document 1. The annexes to this document are purely informative. Documents 2 and 3 (this document) are also purely informative.

The normative part of the specification of the block cipher (**KASUMI)** on which the **A5/3** and **GEA3** algorithms are based can be found in [5].

Blank Page

TABLE OF CONTENTS

REFERENCES

[1] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 1: **A5/3** and **GEA3** Specifications.

[2] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 2: Implementors' Test Data.

[3] Specification of the **A5/3** Encryption Algorithms for GSM and ECSD~~DCE~~, and the **GEA3** Encryption Algorithm for GPRS;
Document 3: Design Conformance Test Data.

[4] 3GPP TS 35.201: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 1: *f8* and *f9* specifications, V4.1.0

[5] 3GPP TS 35.202: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 2: **KASUMI** specification, V4.0.0

[6] 3GPP TS 35.203: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms: Document 3: Implementors' Test Data, V4.0.0

1. OUTLINE OF THE DESIGN CONFORMANCE TEST DATA

Section 2 introduces the algorithms and describes the notation used in the subsequent sections.

Section 3 provides test data for the encryption algorithm A5/3 for GSM.

Section 4 provides test data for the encryption algorithm A5/3 for ECSD~~DGE~~.

Section 5 provides test data for the encryption algorithm GEA3 for GPRS.

2. INTRODUCTORY INFORMATION

2.1. Introduction

In this document black box test data are given for three ciphering algorithms: **A5/3** for GSM, **A5/3** for ECSD~~DGE~~, and **GEA3** for GPRS (including EGPRS). The algorithms are stream ciphers that are used to encrypt/decrypt blocks of data under a confidentiality key $\mathbf{K_C}$. Each of these algorithms is based on the **KASUMI** algorithm that is specified in reference [5]. **KASUMI** is a block cipher that produces a 64-bit output from a 64-bit input under the control of a 128-bit key. The algorithms defined in [1] use **KASUMI** in a form of output-feedback mode as a keystream generator. No test data will be given for **KASUMI**, as these can be found in [6].

2.2. Notation

2.2.1. Radix

We use the prefix **0x** to indicate **hexadecimal** numbers.

2.2.2. Bit/Byte ordering

All data variables in this specification are presented with the most significant bit (or byte) on the left hand side and the least significant bit (or byte) on the right hand side. Where a variable is broken down into a number of sub-strings, the left most (most significant) sub-string is numbered 0, the next most significant is numbered 1 and so on through to the least significant.

For example an n-bit **STRING** is subdivided into 64-bit substrings $\mathbf{SB_0, SB_1 \ldots SB_i}$ so if we have a string:

0x0123456789ABCDEFFEDCBA98765432108654538 1AB594FC28786404C50A37…

we have:

$\mathbf{SB_0}$ = 0x0123456789ABCDEF
$\mathbf{SB_1}$ = 0xFEDCBA9876543210
$\mathbf{SB_2}$ = 0x86545381AB594FC2
$\mathbf{SB_3}$ = 0x8786404C50A37…

In binary this would be:

00000001001000110100010101100111100010011010101111001101111101111111111110…

with **SB_0** = 00000001001000110100010101100111100010011010101111001101111101111
SB_1 = 11111110110111001011101010011000011101100101010000110010000100000
SB_2 = 10000110010101000101001110000001101010110101100101001111110000010
SB_3 = 10000111100001100100000001001100010100001010001101111…

2.2.3. Presentation of input/output data

The basic data processed by the algorithm A5/3 are blocks of two times 114 bits (GSM) resp. 348 bits (ECSD~~DGE~~). In general in this document the data is presented in hexadecimal format as bytes, thus the last byte shown as part of an input or output data block may include 0 to 6 bits that are ignored once the block size has been reached (the least significant bits of the byte are ignored).

2.3. List of Variables

BLOCK1	a string of keystream bits output by the **A5/3** algorithm — 114 bits for GSM, 348 bits for ECSD~~DGE~~.
BLOCK2	a string of keystream bits output by the **A5/3** algorithm — 114 bits for GSM, 348 bits for ECSD~~DGE~~.
COUNT	a 22-bit frame dependent input to both the GSM and ECSD~~DGE~~ **A5/3** algorithms.
DIRECTION	a 1-bit input to the **GEA3** algorithm, indicating the direction of transmission (uplink or downlink).
INPUT	a 32-bit frame dependent input to the **GEA3** algorithm.
K_C	the cipher key that is an input to each of the three cipher algorithms defined here. Although at the time of writing the standards specify that K_C is 64 bits long, the algorithm specifications here allow it to be of any length between 64 and 128 inclusive, to allow for possible future enhancements to the standards.
KLEN	the length of K_C in bits, between 64 and 128 inclusive (see above).
M	an input to the **GEA3** algorithm, specifying the number of octets of output to produce.
OUTPUT	the stream of output octets from the **GEA3** algorithm.

2.4. Coverage

For each of the algorithms the test data have been selected such that, provided the entire set of tests is run:

- Each key bit will have been in both the ‘1’ and the ‘0’ states.
- Each bit of the initialisation fields (COUNT, DIRECTION) will have been in both the ‘1’ and the ‘0’ states.
- Every entry in the internal S-boxes of **KASUMI** will have been used.

The **KASUMI** coverage is already being reached with the 64 bit test sets of each algorithm.

3. ALGORITHM A5/3 FOR GSM

3.1. Overview

The test data sets presented here are for the algorithm A5/3 for GSM. For GSM, the DIRECTION bit is not applicable and is set to zero.

3.2. Format

Each test starts by showing the various inputs (K_C, COUNT) to the function. Thereafter both keystream blocks are shown. The first test set will also list all values in their binary representations.

3.3. Test Set 1

3.3.1. Binary Representation

KLEN	64
Kc	0010101111101011001011001111110000010110001011011110000000000
COUNT	1001001111001000001111

BLOCK1:
```
10001000100111101110101010101111100111101101000110111010000110
10101110111101100001000011011000100011001011100100001
```

BLOCK2:
```
01011100101000110100000001101010101000100100010011001111011010
01110011111000001000111101010101101101000101101111101
```

3.3.2. Hexadecimal Representation

KLEN	64
Kc	0x2BD6459F82C5BC00
COUNT	0x24F20F

BLOCK1: `0x889EEAAF9ED1BA1ABBD8436232E440`

BLOCK2: `0x5CA3406AA244CF69CF047AADA2DF40`

3.4. Test Set 2

KLEN	64
Kc	0x952C49104881FF48
COUNT	0x061527

BLOCK1: 0xAB7DB38A573A325DAA76E4CB800A40

BLOCK2: 0x4C4B594FEA9D00FE8978B7B7BC1080

3.5. Test Set 3

KLEN	64
Kc	0xEFA8B2229E720C2A
COUNT	0x33FD3F

BLOCK1: 0x0E4015755A336469C3DD8680E30340

BLOCK2: 0x6F10669E2B4E18B042431A28E47F80

3.6. Test Set 4

KLEN	64
Kc	0x3451F23A43BD2C87
COUNT	0x0E418C

BLOCK1: 0x75F7C4C51560905DFBA05E46FB54C0

BLOCK2: 0x192C95353CDF979E054186DF15BF00

3.7. Test Set 5

KLEN	64
Kc	0xCAA2639BE82435CF
COUNT	0x2FF229

BLOCK1: 0x301437E4D4D6565D4904C631606EC0

BLOCK2: 0xF0A3B8795E264D3E1A82F684353DC0

3.8. Test Set 6

KLEN	64
Kc	0x7AE67E87400B9FA6
COUNT	0x2F24E5

BLOCK1: 0xF794290FEF643D2EA348A7796A2100

BLOCK2: 0xCB6FA6C6B8A705AF9FEFE975818500

3.9. Test Set 7

KLEN	64
Kc	0x58AF69935540698B
COUNT	0x05446B

BLOCK1: 0x749CA4E6B691E5A598C461D5FE4740

BLOCK2: 0x31C9E444CD04677ADAA8A082ADBC40

3.10. Test Set 8

KLEN	64
Kc	0x017F81E5F236FE62
COUNT	0x156B26

BLOCK1: 0x2A6976761E60CC4E8F9F52160276C0

BLOCK2: 0xA544D8475F2C78C35614128F1179C0

3.11. Test Set 9

KLEN	64
Kc	0x1ACA8B448B767B39
COUNT	0x0BC3B5

BLOCK1: 0xA4F70DC5A2C9707F5FA1C60EB10640

BLOCK2: 0x7780B597B328C1400B5C74823E8500

3.12. Test Set 10

KLEN	80
Kc	0x5ACB1D644C0D512041A5
COUNT	0x1D5157

BLOCK1: 0x8EFAEC49C355CCD995C2BF649FD480

BLOCK2: 0xF3A2910CAEDF587E976171AAF33B80

3.13. Test Set 11

KLEN	80
Kc	0x9315819243A043BEBE6E
COUNT	0x2E196F

BLOCK1: 0xAA08DB46DD3DED78A612085C529D00

BLOCK2: 0x0250463DA0E3886F9BC2E3BB0D73C0

3.14. Test Set 12

KLEN	128
Kc	0x3D43C388C9581E337FF1F97EB5C1F85E
COUNT	0x35D2CF

BLOCK1: 0xA2FE3034B6B22CC4E33C7090BEC340

BLOCK2: 0x170D7497432FF897B91BE8AECBA880

3.15. Test Set 13

KLEN	128
Kc	0xA4496A64DF4F399F3B4506814A3E07A1
COUNT	0x212777

BLOCK1: 0x89CDEE360DF9110281BCF57755A040

BLOCK2: 0x33822C0C779598C9CBFC49183AF7C0

4. ALGORITHM A5/3 FOR ECSD~~DGE~~

4.1. Overview

The test data sets presented here are for the algorithm A5/3 for ECSD~~DGE~~.

For ECSD~~DGE~~, the DIRECTION bit is not applicable and is set to zero. ECSD~~DGE~~ allows block sizes up to 348 bits for BLOCK1 and BLOCK2. As A5/3 for ECSD~~DGE~~ always produces two times 348 bits, the superfluous bits of each output block have to be discarded.

4.2. Format

Each test starts by showing the various inputs (K_C, COUNT) to the function. Thereafter both keystream blocks are shown. The first test set will also list all values in their binary representations.

4.3. Test Set 1

4.3.1. Binary Representation

KLEN	64
Kc	0010101111010110010110011111100000101100010110111100000000000
COUNT	100100111100100000111

BLOCK1:
```
11110111010111100110011000111010110011101010001000011110110010
01110100001011110111101001100010110110110000110011101110000001
10010010100110011111010000011000010100001101000101110001011111 0
01000101000011001001101011111011110101000101010000100010011011
01101101101100001111001001110001101011111011100101100000100111
11100100000101001000000010110011011100
```

BLOCK2:
```
11110101000101000010011011010001011100101101101101000111101111
11111011010011111001101101100000111101000101001111010010000111
01100011011001101100110011001101010110111111101011101000010110
11001001111100100110110100100111111001011110111011101101011011
00001011010100000100100100000101111100100111101101011010111001
10001010111000001001101001111010101001
```

4.3.2. Hexadecimal Representation

KLEN	64
Kc	0x2BD6459F82C5BC00
COUNT	0x24F20F

BLOCK1:
```
0xF75E663ACEA21EC9D0BDE98B6C33B819299E830A1A2E2F914326BEF51508
9B6DB0F271AFB9609F905202CDC0
```

BLOCK2:
```
0xF51426D172DB47BFED3E6D83D14F4876366CCCD5BFAE85B27C9B49F2F777
5B0B504905F27B5AE62B8269EA90
```

4.4. Test Set 2

KLEN	64
Kc	0x952C49104881FF48
COUNT	0x061271

BLOCK1:
0x7A48E94F5949D6145C6A8918C9136ABEF03D44EF8815F01981999A06E1D2
4A324EE2553879B85F88CF8A5A70

BLOCK2:
0x056D9F4C43D82878A6EA70C6007DF5BC27FF134A06889E5164AFCEE6ED99
D2DEF25BC0DDB25B7C77E9210910

4.5. Test Set 3

KLEN	64
Kc	0xEFA8B2229E720C2A
COUNT	0x33FD3F

BLOCK1:
0x09B49CE620E4A36B7956186C8F248B6150DC2362B3F41F6F28F486D9A80B
B879DA4FE349E72EF9755A501590

BLOCK2:
0x02B17EE1DF32D9302567E470EA3A26B0FFCDE60DFB8A28C10609AEC74CA1
EEDF3BAA3334C28E7E4DDA38A4A0

4.6. Test Set 4

KLEN	64
Kc	0x3451F23A43BD2C87
COUNT	0x0041BC

BLOCK1:
0x1257046374CDC415B8B920FBBA0B5AC14165A157704F0C0ADB14F457708B
F71B2B19291C796395AECE0512C0

BLOCK2:
0xFBE2DE7861EBDD918FB450E4AA66C4405B8A90C80A1F94F07316A60EC429
9E1DB5CBEE1A900344914F194EF0

4.7. Test Set 5

KLEN	64
Kc	0xCAA2639BE82435CF
COUNT	0x1FF209

BLOCK1:
0x1640244FFF0A22021A3B8B7604661B518ADEACE830191F024D16E1808168
7799129E37466C67B4805E71D4E0

BLOCK2:
0xE62268E32C9A61FF2386849D6330A09D4A8AB99D9D905D0E4191B8D6DFAD
3E924FBB026B214D5AC5E3D9CCC0

4.8. Test Set 6

KLEN	80
Kc	0x5ACB1D644C0D512041A5
COUNT	0x156B26

BLOCK1:
0xAE630E6400A71DD02B24789C13157DE0B89525B040EF772341E3F5B5E353
3C488998C5904A47C399874CC120

BLOCK2:
0x1995B34B89FB53BF9278FED919EE8CCE20AE54E2EF295D92DD74D871D344
82A40ECE60ECB9ED15CCD9337C90

4.9. Test Set 7

KLEN	80
Kc	0x9315819243A043BEBE6E
COUNT	0x2E196F

BLOCK1:
0xB4AF6C69B33BD7A3921BDE4C7780FADDE7B169D82D63DC969577588C37BA
C61E5C07C10B18F4E466E244AB70

BLOCK2:
0x376F8B04E7F675844CD704F207D5D60ACD2050D4D4A94E37C3E911758735
419894BF2213F910D8F3DCCBE970

4.10. Test Set 8

KLEN	128
Kc	0x3D43C388C9581E337FF1F97EB5C1F85E
COUNT	0x35D2CF

BLOCK1:
0x566A5690468114D018FC796FAA1C58EA96BC49BA3CCC426E19F3E800D508
BBC65608B97CD5F1AA7DCE0510B0

BLOCK2:
0x1418CD8B91E369BD363ECF2C70644AD0819E33DACF33925AAE31A6BDCEA2
6391F918DFDEB60ECDF66AC603D0

4.11. Test Set 9

KLEN	128
Kc	0xA4496A64DF4F399F3B4506814A3E07A1
COUNT	0x212777

BLOCK1:
0x9440D02F6267722222FF55767A15679A446A9F1BB84EE1B25792BC6E2EFC
0A3D7A423C506808021AB401E020

BLOCK2:
0x8266AA6D07CE062AB6DB85F53B9244052093BDAD7A9D06DBEF9C1FB73959
CFC5BFE4F25062429873E7DB5000

5. Algorithm GEA3 for GPRS

5.1. Overview

The test data sets presented here are for the algorithm GEA3 for GPRS.

5.2. Format

Each test starts by showing the various inputs (K_C, COUNT, DIRECTION, M) to the function. Thereafter both keystream blocks are shown. The first test set will also list all values in their binary representations.

5.3. Test Set 1

5.3.1. Binary Representation

KLEN	64
Kc	0010101111010110010110011111100000101100010110111100000000000
INPUT	10001110100101000010000110100011
DIRECTION	0
M	59

OUTPUT:

```
01011111001101011001011100001001110111110100101010000110100000
01000001011011000101111011011011001001000000011001010000101000
00001111100010000000101101001000110111001100110111000010101011
11111011101101010000010101110110111110111101000011010101001110
11101011101100100001110100000111001111001100101110111111101100
10110101110000011010111101011110101111111111010011011100011111
11001001011011100011100101110000110100010100001111011100101100
10011000100100000001010100100000100110
```

5.3.2. Hexadecimal Representation

KLEN	64
Kc	0x2BD6459F82C5BC00
INPUT	0x8E9421A3
DIRECTION	0
M	59

OUTPUT:

```
0x5F359709DE950D0105B17B6C90194280F880B48DCCDC2AFEED415DBEF435
4EEBB21D073CCBBFB2D706BD7AFFD371FC96E3970D143DCB2624054826
```

5.4. Test Set 2

KLEN	64
Kc	0x952C49104881FF48
INPUT	0x5064DB71
DIRECTION	0
M	59

OUTPUT:

```
0xFDC03D738C8E14FF0320E59AAF75760799E9DA78DD8F888471C4AEAAC184
9633A26CD84F459D265B83D7D9B9A0B1E54F4D75E331640DF19E0DB0E0
```

5.5. Test Set 3

KLEN	64
Kc	0xEFA8B2229E720C2A
INPUT	0x4BDBD5E5
DIRECTION	1
M	59

OUTPUT:

```
0x4718A2ADFC90590949DDADAB406EC3B925F1AF1214673909DAAB96BB4C18
B1374BB1E99445A81CC856E47C6E49E9DBB9873D0831B2175CA1E109BA
```

5.6. Test Set 4

KLEN	64
Kc	0x3451F23A43BD2C87
INPUT	0x893FE14F
DIRECTION	0
M	59

OUTPUT:

```
0xB46B1E284E3F8B63B86D9DF0915CFCEDDF2F061895BF9F82BF2593AE4847
E94A4626C393CF8941CE15EA7812690D8415B88C5730FE1F5D410E16A2
```

5.7. Test Set 5

KLEN	64
Kc	0xCAA2639BE82435CF
INPUT	0x8FE17885
DIRECTION	1
M	59

OUTPUT:

```
0x9FEFAF155A26CF35603E727CDAA87BA067FD84FF98A50B7FF0EC8E95A0FB
70E79CB93DEE2B7E9AB59D050E1262401571F349C68229DDF0DECC4E85
```

5.8. Test Set 6

KLEN	64
Kc	0x1ACA8B448B767B39
INPUT	0x4F7BC3B5
DIRECTION	0
M	59

OUTPUT:

```
0x514F6C3A3B5A55CA190092F7BB6E80EF3EDB738FCDCE2FF90BB387DDE75B
BC32A04A67B898A3DFB8198FFFC37D437CF69E7F9C13B51A868720E750
```

5.9. Test Set 7

KLEN	80
Kc	0x5ACB1D644C0D512041A5
INPUT	0xF0A7F9D0
DIRECTION	1
M	59

OUTPUT:

```
0x1CC337BCFA4E339713BD8B4C42C2E7571BE86B6B7C56EDB662199B1705BA
CB692D377DB61812B31B58A923F7F13AEFD21AAFBB28739979124A3EE5
```

5.10. Test Set 8

KLEN	80
Kc	0x9315819243A043BEBE6E
INPUT	0x0B5B6901
DIRECTION	0
M	59

OUTPUT:

0x23D335BE02460D89AB609C32E2DF8CB04F336FB358FB74778AC0331EBE00
FFAE8D218EEE5CD181B3BC1580B6D0D7FD6DAC2DFF34654AD9545EB293

5.11. Test Set 9

KLEN	128
Kc	0x3D43C388C9581E337FF1F97EB5C1F85E
INPUT	0x48571AB9
DIRECTION	0
M	59

OUTPUT:

0xFC7314EF00A63ED0116F236C5D25C54EEC56A5B71F9F18B4D7941F84E422
ACBDE5EEA9A204679002D14F312F3DEE2A1AC917C3FBDC3696143C0F5D

5.12. Test Set 10

KLEN	128
Kc	0xA4496A64DF4F399F3B4506814A3E07A1
INPUT	0xEB04ADE2
DIRECTION	1
M	59

OUTPUT:

0x2AEB5970FB06B718027D048488AAF24FB3B74EA4A6B1242FF85B108FF816
A303C72757D9AAD862B835D1D287DBC141D0A28D79D87BB137CD1198CD

3GPP TSG-SA3 Meeting #26
Oxford, England, 19-22 November

Tdoc ⌘S3-020656

CR-Form-v7

CHANGE REQUEST

⌘ **TR 55.919** **CR** **001** ⌘**rev** **-** ⌘ Current version: **6.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☒ Radio Access Network ☐ Core Network ☒

Title:	⌘	Algoritms for ECSD and EGPRS
Source:	⌘	SA WG3
Work item code:	⌘	SEC1-CSALGO1
Date:	⌘	14/11/2002
Category:	⌘	**F**
Release:	⌘	REL-6

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
Rel-4	*(Release 4)*
Rel-5	*(Release 5)*
Rel-6	*(Release 6)*

Reason for change: ⌘ At SA3 #25 Ericsson presented a discussion paper in S3-020545 asking for clarification on the algorithm to be used for EGPRS. The following extract has been taken from the SA3 #25 meeting report:

"TD S3-020545 A5/3 and GEA3 and their relation with EGPRS. This was introduced by Ericsson and questions the use of A5/3 for EDGE and the data-rate for EGPRS and asks SA WG3 to discuss the issues raised in order to provide any necessary CRs to the next SA WG3 meeting. It was confirmed that A5/3 and GEA3 were suitable for both GSM/GPRS and EDGE variants, the algorithm specifications are unclear on this: ***The modulation scheme used in the PS domain does not affect the GEA3 algorithm mechanism. A5/3 (CS domain) has 2 modes of use, GSM standard mode and GSM EDGE mode****. No CR to TS 43.020 was thought necessary, as implementers need to look at the algorithm specifications where the two modes of operation are clarified. It was agreed, however, to create a CR to the Technical Report TR 55.919 to clarify the use of the term "EDGE" in the specifications and the EGPRS bit-rates.* ***K. Boman agreed to do this for the next SA WG3 meeting****."*

Summary of change: ⌘ The term "EDGE" has been deleted from TR 55.919 as it very confusing i.e. the definition is unclear in 3GPP whether it applies for enhanced circuit-switched data or enhanced GPRS or both.
The term ECSD has been introduced as it is defined in 21.905 Vocabulary for 3GPP Specifications and stands for enhanced circuit-switched data.
The term EGPRS has been introduced as it is defined in 21.905 Vocabulary for 3GPP Specifications and stands for enhanced GPRS.
It's been clarified that GEA3 shall be used for EGPRS.
It has been clarified in chapter 6.4.2 that the technical data as data-rates and initialisations and so on are not applicable for EGPRS.

Consequences if not approved:	⌘	It's unclear whether: - the term EDGE means enhanced circuit-switched data or enhanced GPRS or both; - what algorithm that shall be used for EGPRS.

Clauses affected:	⌘	1, 2, 3, 5, 6, 7, 9, 10, 11			
		Y	**N**		
Other specs affected:	⌘			Other core specifications	⌘
				Test specifications	
				O&M Specifications	
Other comments:	⌘				

3GPP TR 55.919 V6.0.0 (2002-09)

Technical Report

3rd Generation Partnership Project;
Technical Specification Group Services and System Aspects;
3G Security;
Specification of the A5/3 Encryption Algorithms for GSM and ECSD~~DGE~~, and the GEA3 Encryption Algorithm for GPRS ~~and EGPRS~~;
Document 4: Design and Evaluation Report
(Release 6)

Keywords

3GPP, GPRS, security, algorithm

3GPP

Postal address

3GPP support office address

650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet

http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

Introduction

This Report has been produced by ETSI SAGE Task Force for the design of the GSM A5/3 and GEA3 encryption algorithms.

The work described in this report was undertaken in response to a request made by Security Group GSM Association. The work was done under supervision of the ETSI Mobile Competence Centre (MCC) and the GSM Association.

1 Scope

This Technical Report has been prepared by the ETSI SAGE GSM A5/3 Task Force, and gives a detailed report on the design and evaluation of the **A5/3** encryption algorithms for GSM and ECSD~~DGE~~, and of the **GEA3** encryption algorithm for GPRS (~~and~~including EGPRS).

This document is an accompanying report to the specification and test data documents listed below. Together with the 3GPP Kasumi specification, ref [7], these documents form the entire specifications of the **A5/3** and **GEA3** algorithms:

- Specification of the A5/3 Encryption Algorithms for GSM and ~~EDGE~~ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 1: A5/3 and GEA3 Specifications.
- Specification of the A5/3 Encryption Algorithms for GSM and ~~EDGE~~ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 2: Implementors' Test Data.
- Specification of the A5/3 Encryption Algorithms for GSM and ~~EDGE~~ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 3: Design Conformance Test Data.

This public report contains a detailed summary of the work performed during the design and evaluation of the GSM A5/3 algorithm for GSM and ECSD~~DGE~~ and the GEA3 Encryption algorithm for GPRS~~ and EGPRS~~. It contains all results and findings from this work and should be read as a supplement to the formal specification documents, ref. [3] - [5]. Some of the results in this report were initially published in the 3GPP Report on the Evaluation of 3GPP Standard Confidentiality and Integrity Algorithms, ref. [8].

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document.*

[1] GSM 03.20: "Digital cellular telecommunications system (Phase 2+); Security related network functions".

[2] GSM Association Specification for A5/3. "Requirements Specification for the GSM A5/3 Encryption Algorithm (Version 2.0 final)".

[3] TS 55.216: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the A5/3 Encryption Algorithms for GSM and ~~EDGE~~ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 1: A5/3 and GEA3 Specifications".

[4] TS 55.217: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the A5/3 Encryption Algorithms for GSM and ~~EDGE~~ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 2: Implementors' Test Data".

[5] TS 55.218: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the A5/3 Encryption Algorithms for GSM and ~~EDGE~~ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 3: Design Conformance Test Data".

[6] 3GPP TS 35.201 version 4.1.0: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 1: *f8* and *f9* Specification".

[7] 3GPP TS 35.202 version 4.0.0: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification".

[8] "Security Algorithms Group of Experts (SAGE); Report on the Evaluation of 3GPP Standard Confidentiality and Integrity Algorithms". Version 2.0, 2000-10-06.

[9] ISO/IEC 9797-1:1999(E): "Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1".

[10] ISO/IEC 10116:1996: "Information technology – Security techniques – Modes of operation for an *n*-bit block cipher algorithm".

Additional references to external documents are provided in Annex A.

3 Abbreviations

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

APN	Almost perfect non-linear
A5/3	Encryption algorithm for GSM and ECSD~~DGE~~
BLCKCNT	Blockcounter used in A5/3 and GEA3
CA	Input parameter to the KGCORE function – 8 bit

CB	Input parameter to the KGCORE function – 5 bit
CBC	Cipher Block Chaining
CC	Input parameter to the KGCORE function – 32 bit
CD	Input parameter to the KGCORE function – 1 bit
C_i	Round constant used in KASUMI key scheduling
CK	Cipher Key
CO	Output bitstream from the KGCORE function
DSP	Digital Signal Processor
~~EDGE~~	~~Enhanced Data rates for GSM Evolution~~
ETSI	European Telecommunications Standards Institute
FI	Component function of KASUMI
FL	Component function of KASUMI
FO	Component function of KASUMI
f8	UMTS confidentiality (encryption) algorithm
f9	UMTS integrity algorithm
GEA3	Encryption algorithm for GPRS ~~and EGPRS~~
GF(q)	The finite field of q elements
3GPP	3rd Generation Partnership Project
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
GSMA	GSM Association
IV	Initialisation Vector
K_C	GSM Cipher Key
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
KI	Component of round key in KASUMI
KL	Component of round key in KASUMI
KO	Component of round key in KASUMI
MAC	Message Authentication Code
MCC	Mobile Competence Centre
MS	Mobile Station
LP	Linear probability
OFB	Output feedback mode
SAGE	Security Algorithms Group of Experts
SAGE TF 3GPP	SAGE Task Force for the design of the standard 3GPP Confidentiality and Integrity Algorithms
S7	Substitution box used in KASUMI
S9	Substitution box used in KASUMI
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
USIM	User Services Identity Module
XOR	Exclusive Or operation

4 Structure of this report

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

- Clause 5 provides background information on KASUMI and the development of GSM A5/3 and GEA3;
- Clause 6 provides a summary of the algorithm requirements;
- Clause 7 provides a summary of design and evaluation criteria;
- Clause 8 provides a brief description of the KASUMI block cipher and the A5/3 and GEA3 modes of use;
- Clause 9 provides some background information on the chosen design;
- Clause 10 gives an overview of the evaluation work carried out by ETSI SAGE Task Force and other parties and the conclusions of the evaluations;
- Clause 11 lists the specific quality measures taken during the project.

- 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 development of new standardised encryption algorithms for use in GSM (including ECSD~~DGE~~) and GPRS (including EGPRS) systems was conducted by an enlarged ETSI SAGE task force in response to a request from the GSMA security group. The purpose was to develop a modern and strong encryption algorithm for use in these systems based upon previous work done for 3GPP (ref. [8]). It was especially mentioned in the requirements that the algorithm should be based on the 3GPP algorithm KASUMI. It was also clear that available resources would not allow to develop a new algorithm from scratch.

The ETSI SAGE group decided on the following strategies for the work:

- Invite external experts from the 3GPP task force to enlarge the ETSI SAGE group for the project.
- Make re-use of results and analysis from the 3GPP project, but achieve necessary cryptographic separation between the different designs.
- Invite interested manufacturers to comment on the needs for compliance with the 3GPP f8 confidentiality function.
- Try to avoid or minimise the need for additional statistical testing and external evaluation.

6 Summary of algorithm requirements

This section gives a summary of the algorithms requirements described in ref. [2].

6.1 Use of the algorithm

The algorithm shall only be used for GSM, ECSD~~DGE~~, GPRS and EGPRS encryption as described in ref.[1].

More specifically the use of the algorithm is as follows:

- The algorithm is used to encrypt user and signalling data over the air interface;
- The algorithm will be used for GSM;
- The algorithm will be used for ECSD~~DGE~~, enhanced circuit-switched data in GSM;
- The algorithm will be used for GPRS packet radio service in GSM.
- The algorithm will be used for EGPRS, enhanced GPRS in GSM.

6.2 Types of implementation

The normal method for implementing the algorithm is in hardware or DSP.

6.3 Type and parameters of algorithm

The type and parameters of the algorithm are identical to those of the A5 algorithm, which are specified in ref.[1]. Also some additional requirements applied. The requirements are summarised here.

- The algorithm shall be based on KASUMI as defined by 3GPP
- The algorithm is a binary key stream generator.
- The inputs are the key K_c, a time variant parameter COUNT and optionally a direction bit DIRECTION.

- The outputs of the ciphering algorithm are two binary blocks BLOCK1 and BLOCK2.

Relation of the input and output parameters is illustrated in figure 1.

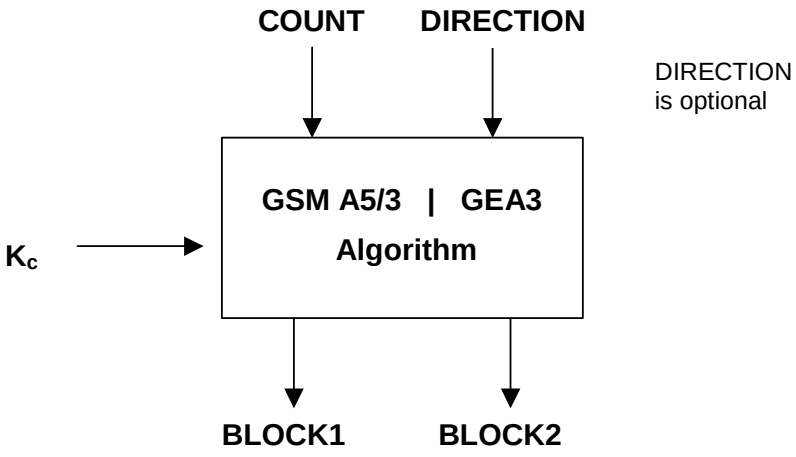

Figure 1 – Algorithm Parameters

The parameters of the algorithms are to be as follows:

Table 1: Algorithm parameters

	GSM (A5/3)	ECSD~~DGE~~ (A5/3)	GPRS ~~and EGPRS~~ (GEA3)
K_c	64 – 128 bits	64 – 128 bits	64 – 128 bits
COUNT	22 bits	22 bits	32 bits
BLOCK1	114 bits	348 bits	M bytes
BLOCK2	114 bits	348 bits	Not Applicable
DIRECTION	Not Applicable	Not Applicable	1 bit

6.4 Implementation and operational considerations

6.4.1 GSM/ECSD~~DGE~~

The performance requirements for the A5 cipher algorithm used for GSM/ECSD~~DGE~~ are given in GSM 03.20. The current version is 8.1.0.

GSM

For ciphering, Algorithm A5 produces, each 4.615 ms, a sequence of 114 encipher/decipher bits (here called BLOCK) which is combined by a bit-wise modulo 2 addition with the 114-bit plain text block.

For each slot, deciphering is performed on the MS side with the first block (BLOCK1) of 114 bits produced by A5, and enciphering is performed with the second block (BLOCK2). As a consequence, on the network side BLOCK1 is used for enciphering and BLOCK2 for deciphering. Therefore Algorithm A5 must produce **two blocks of 114 bits** (i.e. BLOCK1 and BLOCK2) **each 4.615 ms**.

ECSD~~DGE~~

In ECSD~~DGE~~ the block size is greater than 114 bits. With ECSD~~DGE~~ a modification of the usage of the A5 algorithm is employed which produces BLOCK 1 and BLOCK2 which **each contain 348 bits**. The other parameters are not

modified. The modified algorithm produces **both blocks** during a TDMA frame duration, i.e. **4.615 ms**. The blocks are combined by bit-wise modulo 2 addition with the plaintext data.

It is possible in ECSD~~DGE~~ that the plaintext data block for either uplink or downlink is shorter than 348 bits. In this case only the first part of the corresponding output parameter BLOCK is used in the bit-wise addition and the rest of the bits are discarded.

6.4.2 ~~6.4.2~~ GPRS~~/EGPRS~~

Notice that the following GPRS performance requirements for the MS in this chapter are not applicable for EGPRS.

The GPRS performance requirements are specified in GSM 02.60. In **GSM 01.61 version 8.0.0 release 1999**, Section 6.4, the requirements for the GPRS ciphering algorithm are stated as follows:

> Requirements refer to an MS, which admits only 1 timeslot GPRS communication (see note 1), and to an MS, which admits GPRS communication over the maximum number of timeslots (see note 2).
>
> NOTE 1: An MS which admits only one time slot GPRS communication, the maximum capacity in each direction is 21.4 kbit/s (total rate up to 42.8 kbit/s), 12 initialisations per second are assumed (assuming packet length of 500 octets) (scenario 1).
>
> NOTE 2: An MS would have a maximum throughput of all 8 timeslots in both directions each transmitting and receiving at their maximum rate of 21.4 kbit/s (total rate up to 342.4 kbit/s), 100 initialisations per second are assumed (assuming packet length of 500 octets) (scenario 2).

The performance requirements, on the GPRS ciphering algorithm, as used in scenario 1, are expected to be similar to the performance of the existing A5 algorithm.

It is also expected that the performance increases linearly depending on the number of timeslots, the MS is able to use for GPRS.

The clock speed of the mobile may be assumed to be 50MHz.

6.4.3 Implementation complexity

It should be possible to implement the algorithm in hardware using available technology with less than 10000 gates.

6.5 Security of the algorithm

- The algorithm needs to be designed with a view to its continuous use for a period of at least 15 years.
- The security shall be such that there are no known plaintext attacks on the algorithm needing significantly fewer operations than an exhaustive key search.

7 Design and Evaluation Criteria

7.1 Design Criteria:

Based upon the requirements listed in section 6, the task force agreed on the following criteria for the algorithm design:

- The design should be based on the GSM Association Requirements Specification of A5/3, ref. [2];
- There should be one common design that supports the GSM, ECSD~~DGE~~, ~~GPRS~~ and ~~E~~GPRS modes.
- The new design should be as close to the UMTS f8 design as possible.
- Any differences with the UMTS f8 design should in the first place be achieved through the algorithm inputs.
- The modes for GSM, ECSD~~DGE~~, ~~GPRS~~ and ~~E~~GPRS should be cryptographically separated; Preferably these modes should also be cryptographically separated from the UMTS f8.

- The complexity and performance of the algorithm should be comparable to the UMTS f8.

7.2 Evaluation criteria

The agreed criteria for the evaluation work are summarised in the following principles:

- The evaluation of the A5/3 should where possible use the results of the UMTS f8 evaluation; the evaluation should be focused on the aspects where A5/3 and UMTS f8 differ.
- Statistical evaluation is not required, unless there are good arguments for statistical evaluation of specific aspects of the A5/3 design.
- The A5/3 Algorithm needs to be designed with a view to its continuous use for a period of at least 15 years.
- The security shall be such that there are no known keystream attacks on the algorithm predicting a significant amount of additional keystream even with chosen input needing significantly fewer operations than an exhaustive key search (note)

NOTE: An inherent limitation on the GSM security is that the frame counter is limited to 22 bits.

- The algorithm should be no more vulnerable to attacks distinguishing the keystream from a random sequence than other well known block-cipher based constructions, e.g. output feedback mode.

8 GSM A5/3 and GEA3 Encryption Algorithms

The detailed specifications of the A5/3 and GEA3 algorithms are found in ref. [3] and ref. [7]. For this report we include a general overview of the design. The basic building block is the block cipher KASUMI, which is a Feistel block cipher with a block size of 64 bits and a 128-bit cipher key.

8.1 KASUMI

The structure of KASUMI is depicted in the following diagrams:

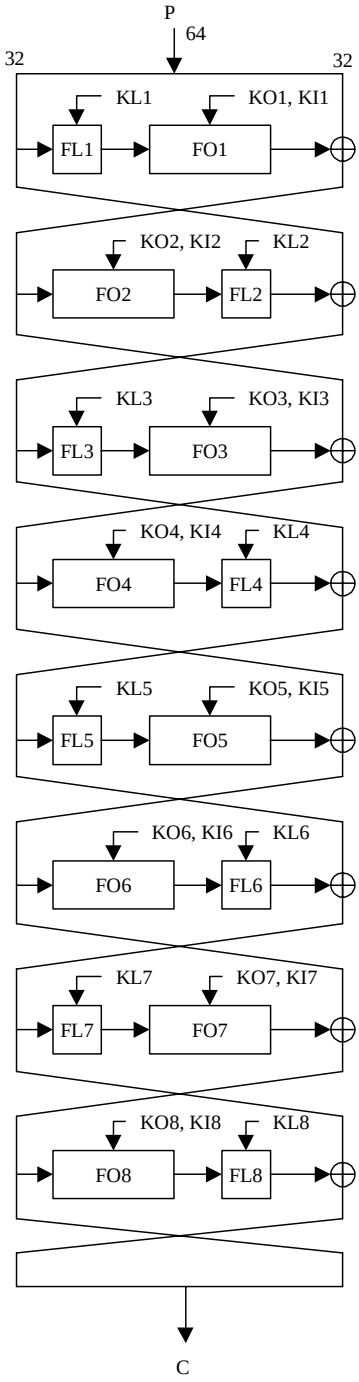

Fig. 2: KASUMI

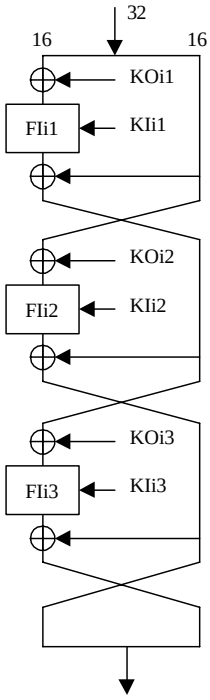

Fig.3: FO Function

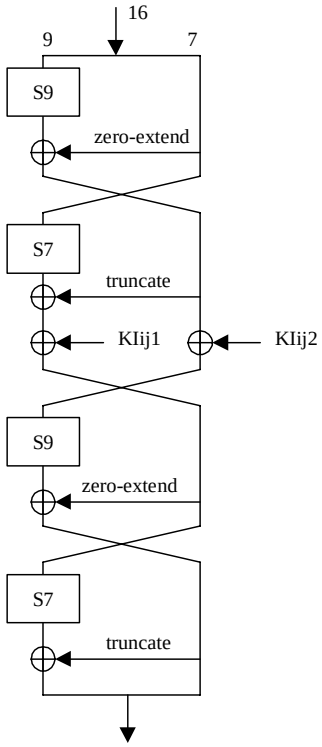

Fig.4: FI Function

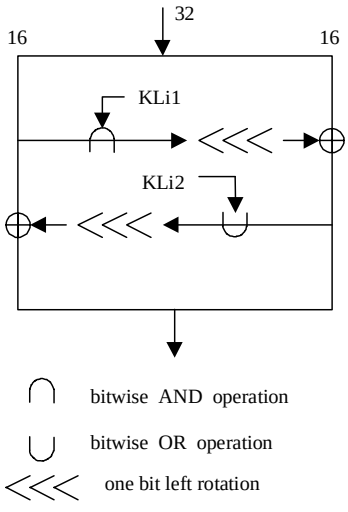

Fig.5: FL Function

KASUMI encrypts a 64-bit input by iterating a round function 8 times. The round function consists of the composition a 32-bit non-linear mixing function (FO) and a 32-bit linear mixing function (FL). The FO-function is again an iterated "ladder-design" consisting of 3 rounds of a 16-bit non-linear mixing function FI. In turn, FI is again defined as a 4-

round structure using non-linear look-up tables S7 and S9. All functions involved will mix the data input with key material. See ref. [7] for details on the specification of S-boxes and generation of round keys.

8.2 Confidentiality function KGCORE

The stream cipher KGCORE used for encryption of data frames in A5/3 and GEA3 is constructed from KASUMI in a variant of the standard Output Feedback Mode (OFB) ref. [10], with 64-bit feedback. The construction is depicted in the following diagram:

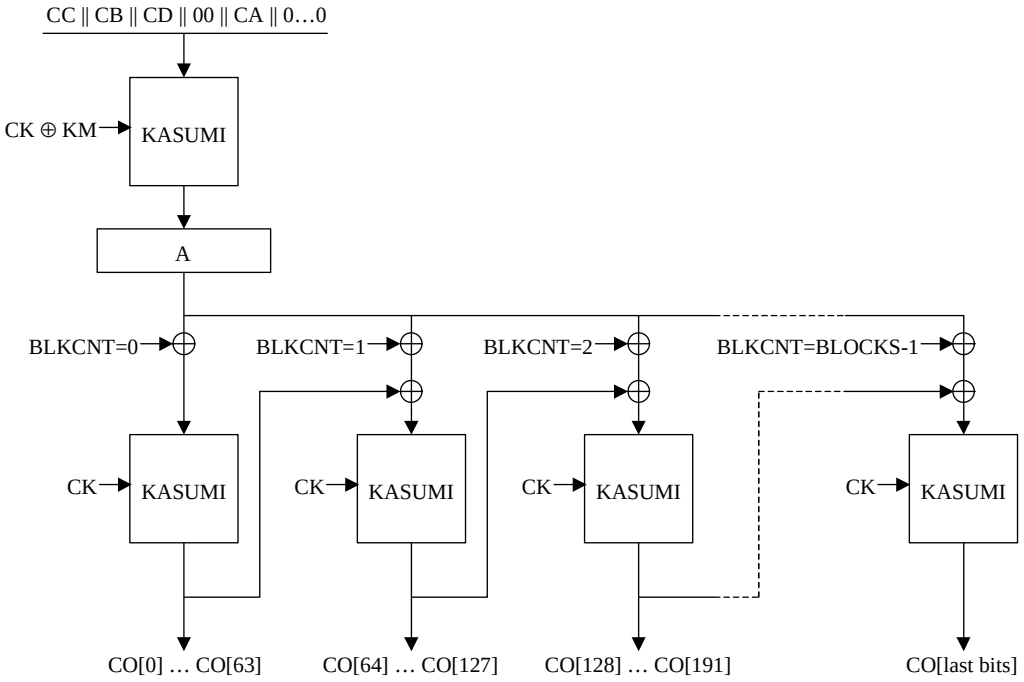

Figure 6: The confidentiality function KGCORE

During a pre-computation phase, the system parameters CC, CB, CD and CA are padded with zeroes to become a full length data block and KASUMI encrypted with a derived key CK⊕ KM. The output of this process is a 64-bit register value A, which is part of the input in each subsequent KASUMI computation. The input parameter CA is used to provide the cryptographic separation between the use of the algorithm within the three systems, and also with the UMTS f8 function.

Subsequent blocks (64 bits) of keystream are then generated by running KASUMI in output feedback mode with additional input of A and the block counter (BLKCNT) to the feedback. The cipher text is then produced as the exclusive or of the keystream bits and the plaintext bits.

9 Rationale for the chosen design

9.1 General comments

The essential design goals for the A5/3 and GEA3 encryption algorithms were that the algorithm should:

- provide a high level of security within the GSM/ECSD~~DGE~~/GPRS~~/EGPRS~~ context;
- meet their implementation requirements - in particular, allow a low power, low gate count implementation in hardware.

The designers have therefore deliberately avoided over-designing the algorithm. They wanted the algorithms to be secure against all practical attacks in the GSM/ECSD~~DGE~~/GPRS~~/EGPRS~~ context, and carefully decided not to over-complicate them just to provide a very high security margin against unrealistic theoretical attacks.

The following types of attacks against the underlying block cipher KASUMI were particularly considered:

- linear cryptanalysis;
- differential cryptanalysis, and variants such as impossible differentials, "miss in the middle", etc;
- higher order differential cryptanalysis and interpolation, including probabilistic higher order analysis;
- identifying any classes of weak keys.

No weak keys were found. There are chosen plaintext and/or related key attacks against KASUMI reduced to 5 rounds, see section 9.2. We believe that with further analysis it might be possible to extend some attacks to 6 rounds, but not to the full 8 round KASUMI. In any case, the more powerful attacks do not translate to practical attacks against the A5/3 and GEA3 algorithms in the operational context.

There are several obvious ways to increase the security margin offered by KASUMI. These include:

- increasing the number of rounds;
- adding a fourth round to the FO function;
- making the key schedule more complicated.

All of these were considered, and rejected as adding complexity for no practical gain.

Attacks against the A5/3 and GEA3 constructions were also considered. The current construction is a good example of the pragmatic approach to the design. Given a very long sequence of keystream (of order 2^{38} bits), it would be possible to identify a small amount of structure in the keystream, which could be classified as an attack (or at least an imperfection); but in the GSM/ECSD~~DGE~~/GPRS~~/EGPRS~~ context, such long frames of keystream will not occur, so the designers saw no need to protect further against this kind of attack.

9.2 Design Policy of MISTY1

The A5/3 and GEA3 crypto engine KASUMI is based on the block cipher MISTY1, ref. [21], which was designed according to the following three principles:

- MISTY should have a numerical basis for its security;
- MISTY should be reasonably fast in software on any processor;
- MISTY should be sufficiently fast in hardware implementation.

The algorithm was designed to be provably secure against differential and linear cryptanalysis. This results from building the algorithm according to provable constructions from smaller components with known resistance against these two types of attacks. The Feistel structure of MISTY1 is recursively repeated in the smaller round function FO and in the kernel FI.

The unequal division of FI is due to the fact that bijective functions of odd size are generally better than those of even size from the viewpoint of provable security against linear and differential cryptanalysis.

In selecting the S-boxes *S*7 and *S*9, the following criteria were adopted:

- Their average differential/linear probability must be minimal;
- Their delay time in hardware is as short as possible;
- Their algebraic degree is high, if possible.

The resulting functions were found by searching for functions of the form $A(x^t)$ over $GF(2^7)$ and $GF(2^9)$, where A is a bijective linear transformation. The non-linear degree of *S*7 is 3 and the non-linear degree of *S*9 is 2.

For the purpose of avoiding possible attacks other than differential and linear cryptanalysis, the design of MISTY1 was supplemented with the simple and fast function FL. This function is linear for a fixed key, but has a variable form depending on the key value.

The key scheduling part of MISTY1 was designed according to the following principles:

- The size of the key is 128 bits;
- The size of the subkey is 256 bits;
- Every round is affected by all key bits;
- As many subkey bits as possible affect every round.

9.3 Changes from MISTY1 to KASUMI

This section summarises the changes that have been done to MISTY1 during the design of KASUMI.

9.3.1 Data Encryption Part

a) *Changing the location of the FL functions.*
This makes hardware simpler; (but a bit slower - this drawback is recovered with other changes. Note that this structure does not block parallel computation of two FI functions).

b) *Removing the subkey KO_{i4} in the FO function.*
This makes hardware simpler and faster; as the FO function now has a simple repetitive structure.

c) *Adding rotate shift functions in the FL function.*
It is assumed that this makes cryptanalysis harder and has no negative impact on hardware size and speed.

d) *Changing of the substitution table S7.*
This is not a significant change, and is in fact equivalent to just rearranging the bit order before and after the original *S*7. We have not found a better table from the viewpoint of hardware implementation.

e) *Changing of the substitution table S9.*
This makes hardware smaller (and possibly faster). The total number of "terms" of the new *S*9 in its algebraic normal form is smaller than that of the original *S*9. We searched all polynomials and normal bases, all powers whose hamming weight is two, and all linear combinations of t_j's for shorter y_i's (see [5]), where the length of y_i is defined as the number of terms (except a constant value) in its algebraic normal form. For the new *S*9, the average length of yi's is 11.2, while for the original version it is 11.7.

f) *Adding another S7 in the FI function*
This makes the security level significantly higher but hardware bigger. We expect that this increase will be compensated with the reduction of the key scheduling part. Note that the penalty on hardware speed is not particularly significant because *S*9 and *S*7 can be performed in parallel.

9.3.2 Key Scheduling Part

a) Removing all FI functions in the key scheduling part.
This makes hardware smaller and/or reduces key set-up time. We expect that related key attacks do not work for this structure.

b) Adding the constant values Ci and rotate shift operations.
This avoids using the same subkey values in different rounds.

9.4 Rationale for the A5/3 and GEA3 design

The construction of the KGCORE confidentiality function is quite similar to the one used for the 3G confidentiality function f8. The two main distinctive features of this construction are the following:

(1) the precomputation of a prewhitening constant A (based upon the initial values CA, CB, CC and CD and upon the modified key value (CK ⊕ KM)

(2) the technique used to produce a sequence of keystream blocks is **neither a mere counter mode nor a mere OFB mode, but a mixture of both techniques.**

The main reason for choice (1), i.e. the precomputation of a prewhitening constant A, was to provide some protection of the underlying blockcipher (KASUMI) by preventing the access by an adversary to blockcipher outputs corresponding to known or chosen inputs. Moreover, as will be shown in section 10.4 , the fact that the prewhitening constant A depends upon the initial values CA, CB, CC and CD provides extra protection against some distinguishing attacks.

The main reasons for choice (2) were to avoid some undesirable properties resulting from a mere OFB mode or a mere counter mode. As a matter of fact:

- **A mere OFB mode, whether or not combined with the precomputation of a prewhitening constant A**, would have allowed for short cycles in the keystream. Though short cycles would not occur with a very high probability, they represent a more serious practical threat than most other distinguishing properties (because they may lead to the disclosure of a much larger amount of information), and are therefore to be avoided.

- **A mere counter mode, if combined with the precomputation of a prewhitening constant A (choice 1), would lead to distinguishing attacks of substantial probability requiring only about 2^{32} keystream blocks.** Let us indeed assume that two keystream blocks, corresponding to blockcounter values i and j and prewhitening values A and A' are equal, i.e. A ⊕ i = A' ⊕ j. Then, other pairs of colliding keystream blocks corresponding to pairs of blockcounter values of the form (i+d, j+d) are likely to exist. For instance if i and j are both even, then two subsequent keystream blocks (corresponding to blockcounter values i+1 and j+1 respectively) are also equal, since i+1 = i ⊕ 1 and j+1 = j ⊕ 1, and therefore A ⊕ (i+1) = A' ⊕ (j+1).

- **In the case of a mere counter mode not combined with the precomputation of a prewhitening constant A (choice 1)**, the former distinguishing attack could be avoided, if the KASUMI input block in the computation of each keystream block consisted of CA, CB, CC, CD, the blockcounter value, and of some filling bits. However, the advantage resulting from choice (1) mentioned before would be lost and moreover, another kind of distinguishing property requiring about 2^{32} keystream blocks would exist, namely the fact that the keystream blocks corresponding to distinct (CA,CB,CC,CD, Blockcounter) values would necessarily be pairwise distinct, whereas some collisions of pairs of keystream blocks would be likely to occur in the case of a perfectly random keystream generator.

10 Algorithm evaluation

In this section we summarise the results of the algorithm evaluation. Much of this work was done as a part of the 3GPP Standard Confidentiality and Integrity algorithms project and for more details we refer to the report ref. [8]. The scope for this latter evaluation work was:

- Analysis of the various components of KASUMI;

- Analysis of KASUMI as a generic 64-bit block cipher;

- Analysis of the encryption mode used for A5/3 and GEA3.

All of these aspects have been exposed to mathematical and statistical evaluation by the task force and by external evaluators. No attacks that threatens the use of A5/3 and GEA3 within GSM/ECSD~~DGE~~/GPRS~~/EGPRS~~ systems have been identified and the general conclusion is that the algorithm is well suited for its intended use.

10.1 Properties of KASUMI components

Each functional component of KASUMI has been carefully studied to reveal any weakness that could be used as a basis for an attack on the entire algorithm. The following are the main results from this work.

10.1.1 FL function

The FL function is a linear function, and the security of the algorithm is not meant to depend on this function. Its main purpose is to be a low cost additional scrambling, making individual bits harder to track through the rounds.

The FL function has the property that for any key KL, an input of $0^{16}1^{16}$ always gives an output of 1^{32}. Hence for some round inputs, some of the key bits in KL can be changed without having any effect on the output of that round. This property can be used to guarantee a zero difference at the end of the first round, thus effectively removing the first round. More generally, small changes to the input to FL only make small output changes, and this can be useful going either forwards or backwards through FL.

The fixed point is used in some of the differential attacks mentioned later, but no attack exploiting this property that extends beyond 5 rounds of KASUMI has been found.

10.1.2 FI function

This is the basic randomising function of KASUMI with 16 bits input and 16 bits output. It is again composed of a four-round structure using two non-linear substitution boxes S7 and S9. Using theorem 4 of ref. [21], we can show that the average linear and differential probability of FI is less than $(2^{-9+1})(2^{-7+1}) = 2^{-14}$ assuming uniform distribution of the subkeys in use. S7 and S9 have been designed in a way that avoids linear structures in FI. This fact has been confirmed by statistical testing.

10.1.3 The S7 box

The S7 box in KASUMI is essentially the same as S7 in MISTY1 (see [21]); the KASUMI S7 was made by rearranging the bit order before and after the original S7. The S7 box is specially designed to be easy to implement in hardware using combinatorial logic, and as a consequence the non-linear order is 3. The algebraic normal form of this function can be found in ref. [7].

10.1.4 The S9 box

The S9 box is different from the S9 box in MISTY1[21], but it has been constructed in much the same way. That is, it is easy to implement in hardware (actually easier than the original S9), and has non-linear order 2. The algebraic normal form of this function can be found in ref. [7]. S9 can been seen as a composition of the power function $x \# x^5$ and a linear output transformation defined over $GF(2)^9$, it is known that it achieves almost perfect non-linearity.

10.1.5 Key schedule

The key schedule of KASUMI is very simple, but this fact has not been found to constitute any real weakness, and there seems to be no gain in practice by making it more complicated. Each of the 128 bits of secret key is used once and only once in every round. They are used in different ways in different rounds, and also at different parts within those rounds, and at times the values are altered using key modifications constants.

Due to the use of the constants C1 to C8 in the key schedule, there is no fixed recurrence relation between consecutive round keys. This property is required to prevent chosen plaintext attacks that are faster than exhaustive search. Further, there exists no equivalent, more compact representation of the expanded key.

Even if regularity and symmetry in the key scheduling do not introduce weaknesses in the algorithm, care should be taken such that shorter keys e.g. 64 bit keys are not extended to a full-length key in a very symmetric way. Just padding with zeroes could give some advantage to an attacker and should not be recommended.

In his analysis of MISTY1 ref.[21] Matsui shows that if the subkey bits are independent, the average differential and linear probabilities are less than 2^{-56}. Some concern has been expressed that with the simple key schedule in KASUMI, the assumption of subkey independence might be too optimistic. However, we have no indications in this direction.

10.1.6 Statistical testing of components

The two S-boxes S7 and S9 are Almost Perfect Non-linear (APN) bijective Boolean Mappings. It is known from the literature (e.g. [14] and [24]) that those functions have specific properties. Some results from the calculations and tests made are due to the construction of the S-boxes. The statistical tests confirmed the design principles of the actual constructions.

The linear approximation test showed that in the case of S7 the maximal Hamming distance of each linear combination of the output components of S7 from the set of affine functions is equal to 64 + 8 = 72, i.e. each linear combination of the output components can be approximated by at least one affine function up to 64 + 8 = 72 values. For S9 the value for the Hamming distance described above is equal to 256 + 16 = 272. That is because the Walsh transform of each linear combination of the output components of the S7 and S9 mappings are three-valued (see [14]).

S7 has no linear factors. But for each linear combination of the output components of S9 one can find one linear factor. That is due to the fact that the component functions of S9 are quadratic (see [24]), i.e. the algebraic normal form of the component function has quadratic terms at the most.

Concerning the cycle structure, S7 and S9 have no obvious deficiencies, e.g. a lot of transpositions.

S7 and S9 are not random S-boxes. The dependence test showed that each output bit of both mappings is dependent on every input bit. But for S9 there are output bits which always change when one input bit is toggled. This is because of the linear structures of S9. S7 satisfies the Avalanche effect, S9 does not.

For the FI and the FO functions no linear structures were found. The dependence test showed that each output bit of both functions is dependent on every input bit. Both functions satisfy the Avalanche effect. But a closer look at the FI function shows that it doesn't behave like a random function according to the dependence test.

10.2 Analysis of KASUMI as a generic 64-bits block cipher

10.2.1 Differential cryptanalysis

From its construction it is clear that, provided that subkeys are independent, three rounds of KASUMI have no differential or linear characteristics with probability larger than 2^{-56}. It should be noted that the upper bounds on FI, see 10.1.2, is tight. It is possible to find differential characteristics for FI with probability 2^{-14}. It is also important to note that the differential effect of FL is low.

In this section we review some of the differential attacks that have been found on reduced versions of KASUMI.

A differential chosen plaintext attack

A chosen plaintext attack on 5 rounds of KASUMI that can be used to recover the key is described in ref. [8]. The attack requires roughly 2^{38} chosen plaintexts, and 2^{80} small operations. It might be possible to extend this attack to 6 rounds, but not to the full 8 rounds of KASUMI.

Differential related key attacks

Though related key attacks seem not to be a threat within the A5/3 and GEA3 use of context, such attacks were considered. It was concluded that it is possible to perform differential related key attacks on four and five rounds of KASUMI. The four round attack requires the encryptions of approximately 2^9 chosen plaintext pairs X and X^* under keys K and K^* respectively, where K and K^* differ in only one bit. The average complexity of this attack is approximately 2^{41}. The five round attack, which is an extension of the four round attack, requires the encryptions of on average $3 \cdot 2^{17}$ chosen plaintext pairs, and has an average complexity of approximately 2^{36}.

Impossible differentials

In the FI function there are no impossible differentials, because of its four-round structure. In the three round FO function, however, several impossible differentials occur since the round function FI is bijective. These lead to impossible differentials over 2 and 3 rounds of KASUMI without the FL function.

The FL functions seem to destroy most of these impossible differentials, or more precisely, make their existence key dependent. We were not able to derive any impossible differentials for the true KASUMI from those known to exist for the FO function.

Hence we are not aware of other impossible key-independent differentials for the KASUMI cipher, than the well-known five-round impossible differential of the form:

$$(0, A) \rightarrow (A, 0) \rightarrow (*, A) \rightarrow (A, *) \rightarrow (0, A) \rightarrow (A, 0)$$

where A is a non-zero 32-bit block, 0 is a 32-bit block of all zeros and each occurrence of $*$ can be replaced by any (possibly different) non-zero block.

This differential can however be used to distinguish 5 round KASUMI from a truly random function, see [8] for details.

An attack on KASUMI reduced to 6 rounds has been found that requires 2^{55} chosen plaintexts and computation of approximately 2^{119} FI values. Another attack against 6 rounds of KASUMI has been found requiring $2^{53.3}$ chosen plaintexts with a complexity of the order of 2^{100} encryptions. Both attacks exploit impossible differentials and the structure of the FO function.

No similar attack on the full 8 rounds of KASUMI has been found, and in the actual context these attacks are not applicable.

Truncated differentials

The best way that has been found to exploit truncated differentials for KASUMI leads to an attack on 3 or 4 rounds of KASUMI without the FL function. This attack uses the fact that the function FO restricted to the 16 leftmost input bits, is bijective onto the leftmost 16 bits in the output.

3 rounds can be broken using about 2^{35} plaintext pairs derived from 2^{18} chosen plaintexts. The 4 round attack requires 2^{48} chosen plaintexts. The FL function will complicate the attack, and in any case, KASUMI with 5 rounds or more is secure against this attack.

10.2.2 Linear cryptanalysis

The validity of the proofs of security given by Matsui in [21] has been examined. That is, how average is the behaviour of fixed keys with respect to linear approximations over the FI function. Mathematical calculations using the Walsh-Hadamard transform and experimental calculations were carried out independently and reached the same conclusions.

LP^{FI} was estimated to be on average smaller than 2^{-14} for any linear hull over FI, but there are specific key values and linear hulls for which $LP^{FI} \approx 2^{-12}$. Of course there will also be key values for which the actual bias is much less than the average case. The maximal amounts of correlation are not high enough to make it possible to chain them to a useful linear approximation path over rounds of KASUMI. For construction of overall approximations one needs to consider all possible paths, and not only the ones which give large biases (correlations).

One attack on five rounds of KASUMI might be possible, but it would require a work effort of at least 2^{95}, around 2^{58} known plaintexts and only be applicable to a fraction of 2^{-3} of the key space. A variant may potentially reduce the work effort to 2^{93} and require around 2^{49} known plaintexts, but will only be applicable to a fraction of 2^{-41} of the key space.

We conclude that, for the full 8 round KASUMI, all keys of the FI function behave pretty much like an average key with respect to the studied linear approximation relations.

10.2.3 Higher order differential attacks

Quite a lot of analysis has been conducted in Japan concerning the strength of the Misty algorithms. Tanaka et al. shows in [31] that 5 round Misty1 without the FL function can be attacked using 1,408 chosen plaintexts, with a method using 6^{th} and 7^{th} order differentials.

It can be shown that the differential property leading to this attack is actually due to the choice of the *S*7 box. Further, it can be shown that it is actually not possible to find an *S*7 box coming from a mapping $x \# x^{e_3}$ with an exponent e_3 of Hamming weight 3, that is at the same time an optimum from the points of view of average differential/linear probabilities and of the 7^{th} order differential property. Finally, the product of any two output bits from S7 will have an algebraic degree bounded by 5.

However, we do not believe that this 7^{th} order differential property still holds for KASUMI, due to the modification of the FI function. Further, we are convinced that traditional attacks based on higher order differentials will work for at most 5 rounds of KASUMI, and no other variants have been found that work for more than 5 rounds of KASUMI.

In [30] Sugita shows the relation between inputs and outputs of 6 rounds of a "Misty-like" transformation, and proves it is not a locally random function. The relation is used in a higher order differential attack to guess the key of 5 round Misty1 without the FL function.

10.3 Implementation attacks

KASUMI has also been analysed with respect to differential attacks like *timing attacks, simple power analysis and differential power analysis.* This investigation did not reveal any properties of KASUMI that would make it particularly vulnerable to these type of attacks. Specifically KASUMI has a favourable key scheduling with respect to power attack methods that try to derive information about the Hamming weight of subkey bytes. The restricted use of KASUMI in the 3GPP environment will also reduce the possibilities for such attacks. In an application where an attacker can do measurements of time of execution and/or power consumption, specific care should be taken to guarantee resistance against implementation attacks.

10.3.1 Statistical evaluation of KASUMI

The block cipher KASUMI itself was tested by statistical methods. We used the dependence test to see if KASUMI satisfies the plaintext-ciphertext Avalanche effect and the key-ciphertext Avalanche effect. The Avalanche effect demands that about 32 bits of the output block shall change if one bit of the 64-bit input block is toggled if the key is fixed, or if one bit of the 128-bit key is toggled provided the same input block is used. We performed the dependence test on KASUMI reduced to two rounds, reduced to four rounds and on the full round KASUMI.

KASUMI reduced to four rounds already satisfies the key-ciphertext and the plaintext-ciphertext Avalanche effect.

To check how good KASUMI destroys redundancy in the input data, we generated a sequence of 16384 blocks of 64 bits by consecutive applications of the block cipher algorithm in ECB-mode, where between two encryption operations the input block is increased by one, starting with the all-zero block. The key was randomly chosen and the same for all calls of KASUMI. For the first sequence the output blocks were concatenated to a sequence of 1,048,576 bits. For the second sequence we built 64 sequences of 16384 bits each out of the i^{th} bits of the 16384 blocks, i.e. one sequence consisting of the first bits of all 16384 blocks, one sequence consisting of the second bits of all blocks, …, one sequence consisting of the 64^{th} bits of all blocks. These 64 sequences were then concatenated again to a sequence of 1,048,576 bits.

The following statistical tests (stream cipher tests) were applied on the two sequences (for a description of most of the tests see for example ref.[17]):

- Frequency test
- Overlapping *m*-tuple test
- Gap test
- Run test
- Coupon-Collector's test
- Universal Maurer test
- Poker test
- Correlation test
- Rank test
- Linear-complexity test
- Ziv-Lempel complexity test
- Maximum-order-complexity test

The two sequences generated to verify that KASUMI destroys redundancy in the input passed all stream cipher tests, i.e. there is no indication that these sequences deviate from random behaviour.

10.4 Analysis of the encryption mode used for A5/3 and GEA3

10.4.1 Distinguishing attacks on A5/3 and GEA3

If we take into account the A5/3 and GEA3 context of operation, namely the fact that the maximum length (in 64-bit blocks) of any keystream sequence is equal to $N = 2^{13}$ blocks in the worst case, i.e. in the GPRS ~~and EGPRS~~ case, we are not aware of any distinguishing attack on A5/3 and GEA3 requiring only 2^{32} keystream blocks, due to the following facts:

- Given any fixed initial value IV = (CA,CB,CC,CD), collisions on two keystream blocks of the keystream sequence associated with IV are predictable and correspond to (i,j) pairs of blockcountervalues such that XOR of the two feedback blocks B and B' involved in the computation of keystream blocks i and j **[1]** be equal to $i \oplus j$. But the probability for such collisions to occur among the at most N blocks of the keystream associated with IV is less than $N^2/2.2^{-64} \approx 2^{-39}$. Thus the number of N-blocks keystream sequences required in order for such a distinguishing event to occur with a probability close to 1 is about 2^{39} - which represents more than the at most 2^{33} distinct IV values available in the GPRS ~~and EGPRS~~ case. Therefore even if an adversary is provided with the keystream sequences associated with all possible count values (i.e. 2^{33}. $2^{13} = 2^{46}$ keystream blocks), the probability of such a distinguishing event remains low (about $2^{33}.2^{-39} = 2^{-6}$)

- Given any two distinct initial values IV = (CA,CB,CC,CD) and IV'= (CA',CB',CC',CD'), the prewhitening constants A and A' differ, and it becomes difficult to predict for which IV and IV' values the corresponding keystream blocks are equal. On the other hand, the observation of two equal keystream blocks (of numbers i and j) associated with two distinct IV values IV and IV' provides an adversary with an equation in the A and A' unknowns, namely the $A \oplus A'$ value: $A \oplus A' = B \oplus B' \oplus i \oplus j$ (where B and B' represent the feedback blocks associated with the keystream blocks corresponding to IV and i and IV' and j respectively). This does not represent by itself a distinguishing information, but if a sufficiently large system of such equations can be collected by an adversary, a "linear consistency test" can be applied, and the fact that inconsistencies are never detected represents a distinguishing information. An order of magnitude of the K number of distinct keystream sequences required in order for the resulting distinguishing probability to become close to 1 can be computed using the following heuristic argument. The number of pairs of colliding blocks is about $(K \cdot N)^2/2.\ 2^{-64} = K^2.2^{-39}$, and each pair of colliding blocks provides an equation of the form $A \oplus A' = B \oplus B' \oplus i \oplus j$, i.e. an equation with GF(2) coefficients in the K unknown A prewhitening constants. A consistency test on such a system is likely to provide a distinguishing information when linear dependencies among the equations are likely to exist, which can be expected to happen when the number of equations is close to the number of unknowns, i.e. when $K \approx K^2.2^{-39}$, i.e. when $K \approx 2^{39}$. Thus even if an adversary is provided with the keystream sequences associated with all possible count values ($K = 2^{33} << 2^{39}$) the probability of this second kind of distinguishing event can be expected to be low.

This analysis shows that typical types of collision-based distinguisheres seems not to produce any distinguishing attacks on the A5/3 and GEA3 keystream generator taking into account the limitations of the number of keystream blocks per keystream sequence ($N = 2^{13}$ in the worst case, i.e. GPRS ~~and EGPRS)~~ and of the number of distinct keystream sequences for the whole system ($K < 2^{33}$ for GPRS ~~and EGPRS).~~

10.5 External evaluation

The KASUMI cipher and the 3GPP f8 and f9 algorithms were analysed by three independent evaluators. Reports from these groups are included in ref. [8] together with comments the task force made to given recommendations from the groups.

The external evaluation did not reveal any flaws or security weaknesses in KASUMI or the f8 and f9 algorithms. The design was found to resist known attacks against block ciphers and the use in the f8 and f9 modes provide the necessary level of security for the intended systems.

Since A5/3 and GEA3 make use of the same underlying block cipher and the chaining mode is very similar to f8, the task force did not recommend to spend additional time and money on a formal external evaluation for this design.

---

1 The feedback block B corresponding to the blockcounter value i is defined as the zero block if i = 0, and as the i-1th keystream block if i>0.

10.6 Complexity evaluation

Independent manufacturers have tested the implementation complexity of the 3GPP confidentiality and integrity algorithms. Their finding concludes that the proposed algorithms fall within the requirements specified in section 6. A high-level realisation of KASUMI has been conducted. The conclusion is that the circuit size is manageable and below 3000 gates. The similarity between the use of KASUMI in 3GPP and in GSM/ECSD~~DGE~~/GPRS~~/EGPRS~~ means that A5/3 and GEA3 will meet the required implementation complexity as well.

11 Quality control

11.1 Algorithm approval

Prior to the release of the A5/3 and GEA3 algorithm specification and test data, the following approvals were gained:

- All members of the ETSI SAGE A5/3 Task Force stated that they have reviewed the technical and security related aspects of the A5/3 and GEA3 algorithms and confirmed that the specifications meet all requirements found in "Requirements Specification for the GSM A5/3 Encryption Algorithm (Version 2.0 final)", ref.[2].

- All members of the ETSI SAGE A5/3 Task Force approved the release of the A5/3 and GEA3 algorithm specifications and the test data to the Algorithm Design Authority (ETSI MCC).

11.2 Specification testing

In addition to peer reviews by the ETSI SAGE A5/3 Task Force, one of the companies involved has been assigned a dedicated task of "Specification Testing". This process involves careful review of the specifications by experts outside the Task Force with regard to the technical content, readability and the editorial presentations. The written contribution from this activity has been incorporated in the final documents.

11.3 Independent implementations

Based on the Algorithm Specification documents, ref.[3] and [7], two different companies represented in the Task Force have produced two independent implementations of the A5/3 and GEA3 algorithms. Both parties have used their implementation for simulations confirming the test data found in Document 2: Implementors' Test Data, ref. [4] and Document 3: "Design Conformance Test data", ref.[5].

11.4 Test data documents

The Task Force has produced two test data documents:

- Document 2: Implementors' Test Data, ref. [4]. This document contains detail simulations for the blockcipher KASUMI , A5/3 for GSM, A5/3 for ECSD~~DGE~~ and GEA3 for GPRS~~ and EGPRS~~. The document gives corresponding input/output values for all the component functions of KASUMI during the 8 rounds. A specific repetition test is included to check that all entries in both S-boxes have been tested. For each of the three applications the document provide detail simulations corresponding to the actual parameters and block sizes of data. This document is intended for implementors who need intermediate values to verify their implementation and correct errors.

- Document 3: "Design Conformance Test data", ref.[5]. This document provides sets of input/output test data for 'black box' testing of self-contained implementations of A5/3 (GSM and ECSD~~DGE~~) and GEA3 (GPRS~~ and EGPRS~~). These tests could be used to develop power-up and maintenance tests proving correctness of the algorithm operation.

11.5 Algorithms distribution procedures

In this case the distribution procedures and related documents were not produced by the ETSI SAGE Task Force; they are the joint responsibility of the ETSI Mobile Competence Centre and the GSM Association. The ETSI SAGE A5/3 Task Force did not draft nor reviewed any distribution procedures or related documents.

Annex A:
External references

NOTE: Not all references in this list are directly referred by this report. However, they contain information that are of relevance to the reading and understanding of the details. They are referred by the more detailed 3GPP report ref. [8].

[11] 3G TS 25.321 V3.0.0: 3rd Generation Partnership Project; Technical Specification Group (TSG) RAN; Working Group 2 (WG2); MAC protocol specification.

[12] S. Ar, R.J. Lipton, R. Rubinfeld, and M. Sudan, "Reconstructing algebraic functions from mixed data", *SIAM J. of Comput.*, Vol. 28, No. 2, 1998, pp 487-510.

[13] E. Biham, A. Biryukov, and A. Shamir, *Cryptanalysis of Skipjack Reduced to 31 Rounds using Impossible Differentials*, Technical Reports of the Computer Science Department in the Technion, 0947.

[14] Hans Dobbertin, Almost Perfect nonlinear Power Functions on GF(2^n): The Welch case, *IEEE Transactions on Information Theory*, Vol. 45, NO.4, May 1999

[15] T. Jakobsen, *High-Order Cryptanalysis of Block Ciphers*,PhDThesis, Department of Math., Technical University of Denmark, 1999.

[16] L. Knudsen, *Truncated and Higher Order Differentials*, Fast Software Encryption - Second International Workshop, Leuven, Belgium, LNCS 1008, Springer Verlag, 1995, pp. 196-211.

[17] Donald E. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, Addision-Wesley, Third Edition, 1998

[18] A.G. Konheim, Cryptography, a primer , NY, John Wiley & Sons, 1981.

[19] X. Lai. *Higher order derivatives and differential cryptanalysis*, In Proc. "Symposium on Communication, Coding and Cryptography" in honour of James L. Masseyon the occasion of his 60'th birthday, Feb. 10-13, 1994, Monte – Verita, Ascona, Switzerland, 1994.

[20] S. K. Langford and M. E. Hellman, *Differential – Linear Cryptanalysis*, Advances in Cryptology – CRYPTO '94, in Lecture Notes in Computer Science 839, Springer, pp. 17-25.

[21] Mitsuru Matsui: *New Block Encryption Algorithm MISTY*, Proceedings of Fast Software Encryption '97 conference, in Lecture Notes in Computer Science 1267, Springer, pp. 54-68.

[22] Willi Meier, Othmar Staffelbach, *Nonlinearity Criteria for Cryptographic Functions*, EUROCRYPT' 89

[23] A. Menezes, P.C. van Oorschot, S.A Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[24] Kaisa Nyberg, *Differential uniform mappings for cryptography*, EUROCRYPT' 93. Lecture Notes in Computer Science 765, Springer Verlag, 1993

[25] K.Nyberg and L. Knudsen, *Provable Security Against a Differential Attack*, Journal of Cryptology Vol 8 Nr 1, 1995

[26] S. Luck, *On the Security of the 128-Bit Block Cipher DEAL*, http://th.informatik.uni-mannheim.de/m/lucks/papers/deal.ps.gz

[27] Rainer A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer, 1986

[28] SSLeay source, ftp://ftp.psy.uq.oz.au/pub/Crypto/SSL (1999), see also http://www.cryptsoft.com/ssleay/faq.html (1999)

[29] M. Sudan, *Decoding of Reed-Solomon codes beyond the error-correction bound*, Journal of Complexity, Vol. 13, 1997, pp 180-193.

[30] M. Sugita: *Higher Order Differential Attack on Block Cipher MISTY1*, 2". Technical Report of IEICE, ISEC98-4, May 1998.

[31] Hidema Tanaka, Kazuyuki Hisamatsu and Toshinobu Kaneko*: "Strength of MISTY1 without FL function for Higher Order Differential attack*". The 3rd World Multiconference on Systemic Cybernetics and Informatics and the International Conference on Information Systems Analysis and Synthesis SCI'99/ISAS'99, Orlando, USA, August 1999.

[32] TSG SA WG3#5: Liaison Statement to SA3 on Ciphering Algorithm Requirements by RAN WG2

[33] D. Wagner, *The Boomerang Attack*, FSE '99, Lecture Notes in Computer Science 1636, Springer Verlag, 1999

[34] Wassenaar Arrangement, December 1998.

Annex B:
Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
2002-05	-	-	-	-	ETSI SAGE first publication		SAGE V1.0
2002-07	-	-	-	-	Agreed at SA WG3 #24 for presentation to TSG SA #17 for approval. Converted into 3GPP TR format (TR 55.919) (Technically equivalent to SAGE V1.0)	SAGE V1.0	1.0.0
2002-09	SP-17	SP-020506	-	-	Approved for Release 6 - version 6.0.0	1.0.0	6.0.0