3GPP TSG SA WG3 Security — S3#29 **S3-030389**
15 - 18 July 2003, San Francisco, USA

CR-Form-v7

CHANGE REQUEST

⌘	**55.216**	**CR**	**CRNum**	⌘ **rev**	**-**	⌘	Current version:	6.1.0 ⌘

*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: ⌘	Clarification on the usage of the Key length.			
Source: ⌘	Siemens			
Work item code: ⌘	Security		***Date:*** ⌘	08/07/2003
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: ⌘	1) Currently the value of the parameter KLEN within this specification is fixed to 64-bit which value the implementations derive from several other 3GPP specifications (Kc). 2) The current MAP specifications only allow Kc to be a multiple of 8-bit which does not fit the full KLEN flexibility. 3) SA3 have decided that only two key lengths will be possible (SA3#28): 64-bit or 128-bit. CN1 was contacted, and it was confirmed that they preferred another algorithm-Identifier (e.g. GEA4, A5/4) when a longer key length would be applicable in future. So according to (1) and (2) full KLEN flexibility is not used and not possible; and according to (3) is not intended in future for GEA3 and A5/3.
Summary of change: ⌘	Remove the unnecessary KLEN flexibility.
Consequences if not approved: ⌘	Future doubt about KLEN flexibility applicable to the algorithms described in this specification, which will not be in accordance with the MAP-interface restrictions.

Clauses affected: ⌘	4,5,6			
	Y	**N**		
Other specs affected: ⌘		**N**	Other core specifications	⌘
		N	Test specifications	
		N	O&M Specifications	

Other comments: ⌘	

***** Begin of Change ****

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:

Table 3: GSM A5/3 inputs

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

Table 4. GSM A5/3 outputs

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

NOTE 1: ~~At the time of writing, the standards specify that K_C is 64 bits long.~~ The~~is~~ specification of the **A5/3** algorithm only allows KLEN to be of value 64~~for possible future enhancements to support longer keys~~.

NOTE 2: It must be assumed that **K_C** is unstructured data — it must not be assumed, for instance, that any bits of **K_C** have predetermined values.

4.3 Function Definition

(See figure B.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 encryption

5.1 Introduction

The ECSD **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:

Table 5: ECSD A5/3 inputs

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

Table 6: ECSD A5/3 outputs

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

NOTE 1: At~~ the time of writing, the standards specify that K_C is 64 bits long. ~~The~~is~~ specification of the **A5/3** algorithm only allows KLEN to be of value 64. ~~for possible future enhancements to support longer keys.~~

NOTE 2: It must be assumed that **K_C** is unstructured data — it must not be assumed, for instance, that any bits of **K_C** have predetermined values.

5.3 Function Definition

(See figure B.3, Annex B).

We define the function by mapping the ECSD **A5/3** inputs onto the inputs of the core function **KGCORE**, and mapping the output of **KGCORE** onto the outputs of ECSD **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] = $\mathbf{K_C}$**[0]…**$\mathbf{K_C}$**[KLEN-1]**

If **KLEN** < 128 then

CK[KLEN]…CK[127] = $\mathbf{K_C}$**[0]…**$\mathbf{K_C}$**[127 – KLEN]**

(So in particular if **KLEN** = 64 then **CK =** $\mathbf{K_C}$ **||** $\mathbf{K_C}$)

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:

Table 7: GEA3 inputs

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~~KLEN	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 8: GEA3 outputs

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

NOTE 1: ~~At the time of writing, the standards specify that $\mathbf{K_C}$ is 64 bits long.~~ Th~~e~~is specification of the **GEA3** algorithm only allows KLEN to be of value 64.~~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 figure B.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 ≤ ***i*** ≤ **M-1** define:

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

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

******End of Change ***