

3GPP TS 35.201 V9.0.0 (2009-12)

Technical Specification

**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
(Release 9)**



The present document has been developed within the 3rd Generation Partnership Project (3GPP™) and may be further elaborated for the purposes of 3GPP.

The present document has not been subject to any approval process by the 3GPP Organizational Partners and shall not be implemented. This Specification is provided for future development work within 3GPP only. The Organizational Partners accept no liability for any use of this Specification. Specifications and reports for implementation of the 3GPP™ system should be obtained via the 3GPP Organizational Partners' Publications Offices.

Keywords

UMTS, algorithm, KASUMI

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>

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

©2009, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TTA, TTC).
All rights reserved.

UMTS™ is a Trade Mark of ETSI registered for the benefit of its members
3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners
LTE™ is a Trade Mark of ETSI currently being registered for the benefit of its Members and of the 3GPP Organizational Partners
GSM® and the GSM logo are registered and owned by the GSM Association

Contents

Foreword	4
Introduction	4
0 Scope	6
NORMATIVE SECTION	7
1 Outline of the normative part	8
1.1 References	8
2 Introductory information	9
2.1 Introduction	9
2.2 Notation	9
2.2.1 Radix	9
2.2.2 Conventions	9
2.2.3 Bit/Byte ordering	9
2.2.4 List of Symbols	10
2.3 List of Variables	10
3 Confidentiality algorithm f_8	11
3.1 Introduction	11
3.2 Inputs and Outputs	11
3.3 Components and Architecture	11
3.4 Initialisation	11
3.5 Keystream Generation	12
3.6 Encryption/Decryption	12
4 Integrity algorithm f_9	12
4.1 Introduction	12
4.2 Inputs and Outputs	12
4.3 Components and Architecture	13
4.4 Initialisation	13
4.5 Calculation	13
INFORMATIVE SECTION	15
Annex 1 (informative): Figures of the f_8 and f_9 Algorithms	16
Annex 2 (informative): Simulation Program Listing	18
Annex 3 (informative): Change history	22

Foreword

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

The 3GPP Confidentiality and Integrity Algorithms f8 & f9 have been developed through the collaborative efforts of the European Telecommunications Standards Institute (ETSI), the Association of Radio Industries and Businesses (ARIB), the Telecommunications Technology Association (TTA), the T1 Committee.

The f8 & f9 Algorithms Specifications may be used only for the development and operation of 3G Mobile Communications and services. Every Beneficiary must sign a Restricted Usage Undertaking with the Custodian and demonstrate that he fulfills the approval criteria specified in the Restricted Usage Undertaking.

Furthermore, Mitsubishi Electric Corporation holds essential patents on the Algorithms. The Beneficiary must get a separate IPR License Agreement from Mitsubishi Electronic Corporation Japan.

For details of licensing procedures, contact ETSI, ARIB, TTA or T1.

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 specification has been prepared by the 3GPP Task Force, and gives a detailed specification of the 3GPP confidentiality algorithm *f8*, and the 3GPP integrity algorithm *f9*.

This document is the first of four, which between them form the entire specification of the 3GPP Confidentiality and Integrity Algorithms:

- **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* Specification"**.
- 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".
- 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".
- 3GPP TS 35.204: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 4: Design Conformance Test Data".

The normative part of the specification of the f_8 (confidentiality) and f_9 (integrity) algorithms is in the main body of this document. The annexes to this document are purely informative. Annex 1 contains illustrations of functional elements of the algorithm, while Annex 2 contains an implementation program listing of the cryptographic algorithm specified in the main body of this document, written in the programming language C.

The normative part of the specification of the block cipher (**KASUMI**) on which they are based is in the main body of Document 2. The annexes of that document, and Documents 3 and 4 above, are purely informative.

0 Scope

This specification gives a detailed specification of the 3GPP confidentiality algorithm f_8 , and the 3GPP integrity algorithm f_9 .

NORMATIVE SECTION

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

1 Outline of the normative part

Section 1 introduces the algorithms and describes the notation used in the subsequent sections.

Section 3 specifies the confidentiality algorithm *f8*.

Section 4 specifies the integrity algorithm *f9*.

1.1 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] 3GPP TS 33.102 version 3.2.0: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Security Architecture".
- [2] 3GPP TS 33.105 version 3.1.0: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Cryptographic Algorithm Requirements".
- [3] 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 Specification".
- [4] 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".
- [5] 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".
- [6] 3GPP TS 35.204: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 4: Design Conformance Test Data".
- [7] ISO/IEC 9797-1:1999: "Information technology – Security techniques – Message Authentication Codes (MACs)".

2 Introductory information

2.1 Introduction

Within the security architecture of the 3GPP system there are two standardised algorithms: A confidentiality algorithm f_8 , and an integrity algorithm f_9 . These algorithms are fully specified here. Each of these algorithms is based on the **KASUMI** algorithm that is specified in a companion document[4]. **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 confidentiality algorithm f_8 is a stream cipher that is used to encrypt/decrypt blocks of data under a confidentiality key **CK**. The block of data may be between 1 and 20000 bits long. The algorithm uses **KASUMI** in a form of output-feedback mode as a keystream generator.

The integrity algorithm f_9 computes a 32-bit MAC (Message Authentication Code) of a given input message using an integrity key **IK**. The approach adopted uses **KASUMI** in a form of CBC-MAC mode.

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

$$\langle \text{variable} \rangle = \langle \text{expression} \rangle$$

we mean that $\langle \text{variable} \rangle$ assumes the value that $\langle \text{expression} \rangle$ 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 **MESSAGE** is subdivided into 64-bit substrings **MB₀,MB₁...MB_i** so if we have a message:

0x0123456789ABCDEFFEDCBA987654321086545381AB594FC28786404C50A37...

we have:

MB₀ = 0x0123456789ABCDEF

MB₁ = 0xFEDCBA9876543210

MB₂ = 0x86545381AB594FC2

MB₃ = 0x8786404C50A37...

In binary this would be:

```
00000001001000110100010101100111100010011010101111001101111011111111110...
```

with $\mathbf{MB}_0 = 0000000100100011010001010110011110001001101010111100110111101111$
 $\mathbf{MB}_1 = 1111111011011100101110101001100001110110010101000011001000010000$
 $\mathbf{MB}_2 = 1000011001010100010100111000000110101011010110010100111111000010$
 $\mathbf{MB}_3 = 1000011110000110010000000100110001010000101000110111...$

2.2.4 List of Symbols

=	The assignment operator.
\oplus	The bitwise exclusive-OR operation.
	The concatenation of the two operands.
$\mathbf{KASUMI}[x]_k$	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 . ($X = X[0] X[1] X[2] \dots$).
Y_i	The i^{th} block of the variable Y . ($Y = Y_0 Y_1 Y_2 \dots$).

2.3 List of Variables

A, B	are 64-bit registers that are used within the f_8 and f_9 functions to hold intermediate values.
BEARER	a 5-bit input to the f_8 function.
BLKCNT	a 64-bit counter used in the f_8 function.
BLOCKS	an integer variable indicating the number of successive applications of KASUMI that need to be performed, for both the f_8 and f_9 functions.
CK	a 128-bit confidentiality key.
COUNT	a 32-bit time variant input to both the f_8 and f_9 functions.
DIRECTION	a 1-bit input to both the f_8 and f_9 functions indicating the direction of transmission (uplink or downlink).
FRESH	a 32-bit random input to the f_9 function.
IBS	the input bit stream to the f_8 function.
IK	a 128-bit integrity key.
KM	a 128-bit constant that is used to modify a key. This is used in both the f_8 and f_9 functions. (It takes a different value in each function).
$\mathbf{KS}[i]$	is the i^{th} bit of keystream produced by the keystream generator.
\mathbf{KSB}_i	is the i^{th} block of keystream produced by the keystream generator. Each block of keystream comprises 64 bits.
LENGTH	is an input to the f_8 and f_9 functions. It specifies the number of bits in the input bitstream.
MAC-I	is the 32-bit message authentication code (MAC) produced by the integrity function f_9 .
MESSAGE	is the input bitstream of LENGTH bits that is to be processed by the f_9 function.
OBS	the output bit streams from the f_8 function.
PS	is the input padded string processed by the f_9 function.
REGISTER	is a 64-bit value that is used within the f_8 function.

3 Confidentiality algorithm *f8*

3.1 Introduction

The confidentiality algorithm *f8* is a stream cipher that encrypts/decrypts blocks of data between 1 and 20000 bits in length.

3.2 Inputs and Outputs

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

Table 1: *f8* inputs

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	$\times 18^1$	The number of bits to be encrypted/decrypted (1-20000)
IBS	1-20000	Input bit stream IBS[0]....IBS[LENGTH-1]

Table 2: *f8* output

Parameter	Size (bits)	Comment
OBS	1-20000	Output bit stream OBS[0]....OBS[LENGTH-1]

3.3 Components and Architecture

(See fig 1 Annex A)

The keystream generator is based on the block cipher **KASUMI** that is specified in [4]. **KASUMI** is used in a form of output-feedback mode and generates the output keystream 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 key variables before the generation of keystream bits.

We set the 64-bit register **A** to **COUNT || BEARER || DIRECTION || 0...0**

(left justified with the right most 26 bits set to 0).

i.e. **A = COUNT[0]...COUNT[31] BEARER[0]...BEARER[4] DIRECTION[0] 0...0**

We set counter **BLKCNT** to zero.

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

We set **KSB₀** to zero.

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

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

¹ In the sample C-code we treat LENGTH as a 32-bit integer.

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 plaintext/ciphertext to be encrypted/decrypted consists of **LENGTH** bits (1-20000) whilst the keystream generator produces keystream bits in multiples of 64 bits. Between 0 and 63 of the least significant bits are discarded from the last block depending on the total number of bits required by **LENGTH**.

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

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

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

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

where $\mathbf{BLKCNT} = n-1$

The individual bits of the keystream are extracted from **KSB**₁ to **KSB**_{**BLOCKS**} in turn, most significant bit first, by applying the operation:

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

$$\mathbf{KS}[(n-1)*64+i] = \mathbf{KSB}_n[i]$$

3.6 Encryption/Decryption

Encryption/decryption operations are identical and are performed by the exclusive-OR of the input data (**IBS**) with the generated keystream (**KS**).

For each integer **i** with $0 \leq i \leq \mathbf{LENGTH}-1$ we define:

$$\mathbf{OBS}[i] = \mathbf{IBS}[i] \oplus \mathbf{KS}[i]$$

4 Integrity algorithm *f9*

4.1 Introduction

The integrity algorithm *f9* computes a Message Authentication Code (MAC) on an input message under an integrity key **IK**. There is no limitation on the input message length of the *f9* algorithm.

For ease of implementation the algorithm is based on the same block cipher (**KASUMI**) as is used by the confidentiality algorithm *f8*.

4.2 Inputs and Outputs

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

Table 3: *f9* inputs

Parameter	Size (bits)	Comment
COUNT-I	32	Frame dependent input COUNT-I[0]...COUNT-I[31]
FRESH	32	Random number FRESH[0]...FRESH[31]
DIRECTION	1	Direction of transmission DIRECTION[0]
IK	128	Integrity key IK[0]...IK[127]
LENGTH	$\times 19^2$	The number of bits to be 'MAC'd
MESSAGE	LENGTH	Input bit stream

Table 4: *f9* output

Parameter	Size (bits)	Comment
MAC-I	32	Message authentication code MAC-I[0]...MAC-I[31]

4.3 Components and Architecture

(See fig 2 Annex A)

The integrity function is based on the block cipher **KASUMI** that is specified in [4]. **KASUMI** is used in a chained mode to generate a 64-bit digest of the message input. Finally the leftmost 32-bits of the digest are taken as the output value **MAC-I**.

4.4 Initialisation

In this section we define how the integrity function is initialised with the key variables before the calculation commences.

We set the working variables: $\mathbf{A} = \mathbf{0}$
and $\mathbf{B} = \mathbf{0}$

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

We concatenate **COUNT**, **FRESH**, **MESSAGE** and **DIRECTION**. We then append a single '1' bit, followed by between 0 and 63 '0' bits so that the total length of the resulting string **PS** (padded string) is an integral multiple of 64 bits, i.e.:

$$\mathbf{PS} = \text{COUNT}[0] \dots \text{COUNT}[31] \text{ FRESH}[0] \dots \text{FRESH}[31] \text{ MESSAGE}[0] \dots \text{MESSAGE}[\text{LENGTH}-1] \text{ DIRECTION}[0] \text{ 1 } \mathbf{0}^*$$

Where $\mathbf{0}^*$ indicates between 0 and 63 '0' bits.

4.5 Calculation

We split the padded string **PS** into 64-bit blocks \mathbf{PS}_i where:

$$\mathbf{PS} = \mathbf{PS}_0 \parallel \mathbf{PS}_1 \parallel \mathbf{PS}_2 \parallel \dots \parallel \mathbf{PS}_{\text{BLOCKS}-1}$$

We perform the following operations for each integer **n** with $0 \leq \mathbf{n} \leq \text{BLOCKS}-1$:

$$\begin{aligned} \mathbf{A} &= \text{KASUMI}[\mathbf{A} \oplus \mathbf{PS}_n]_{\text{IK}} \\ \mathbf{B} &= \mathbf{B} \oplus \mathbf{A} \end{aligned}$$

Finally we perform one more application of **KASUMI** using a modified form of the integrity key **IK**.

$$\mathbf{B} = \text{KASUMI}[\mathbf{B}]_{\text{IK} \oplus \text{KM}}$$

The 32-bit **MAC-I** comprises the left-most 32 bits of the result.

$$\mathbf{MAC-I} = \text{lefthalf}[\mathbf{B}]$$

² In the sample C-code we treat LENGTH as a 32-bit integer.

i.e. For each integer i with $0 \leq i \leq 31$ we define:

$$\mathbf{MAC-I}[i] = \mathbf{B}[i].$$

Bits $\mathbf{B}[32] \dots \mathbf{B}[63]$ are discarded.

INFORMATIVE SECTION

This part of the document is purely informative and does not form part of the normative specification of KASUMI.

Annex 1 (informative): Figures of the *f8* and *f9* Algorithms

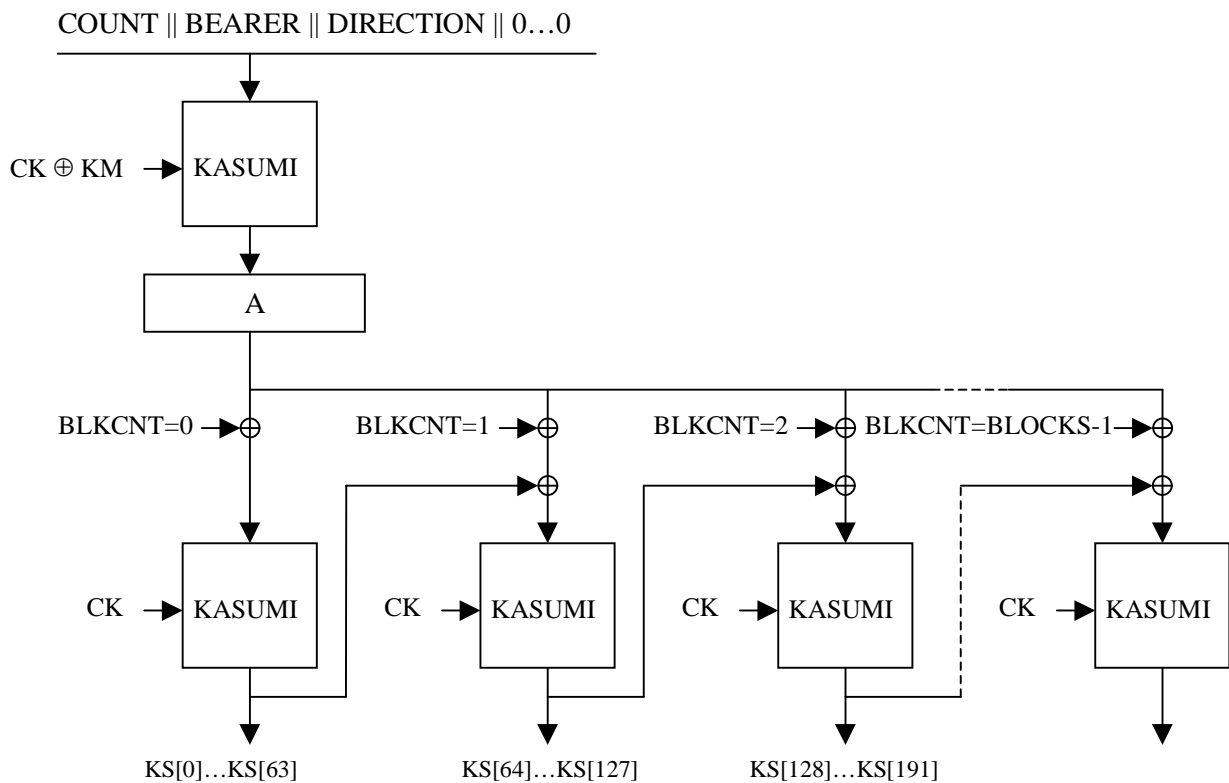


Figure 1: *f8* Keystream Generator

Note: **BLKCNT** is specified as a 64-bit counter so there is no ambiguity in the expression $A \oplus \text{BLKCNT} \oplus \text{KS}_{n-1}$ where all operands are of the same size. In a practical implementation where the key stream generator is required to produce no more than 5114 bits (80 keystream blocks) only the least significant 7 bits of the counter need to be realised.

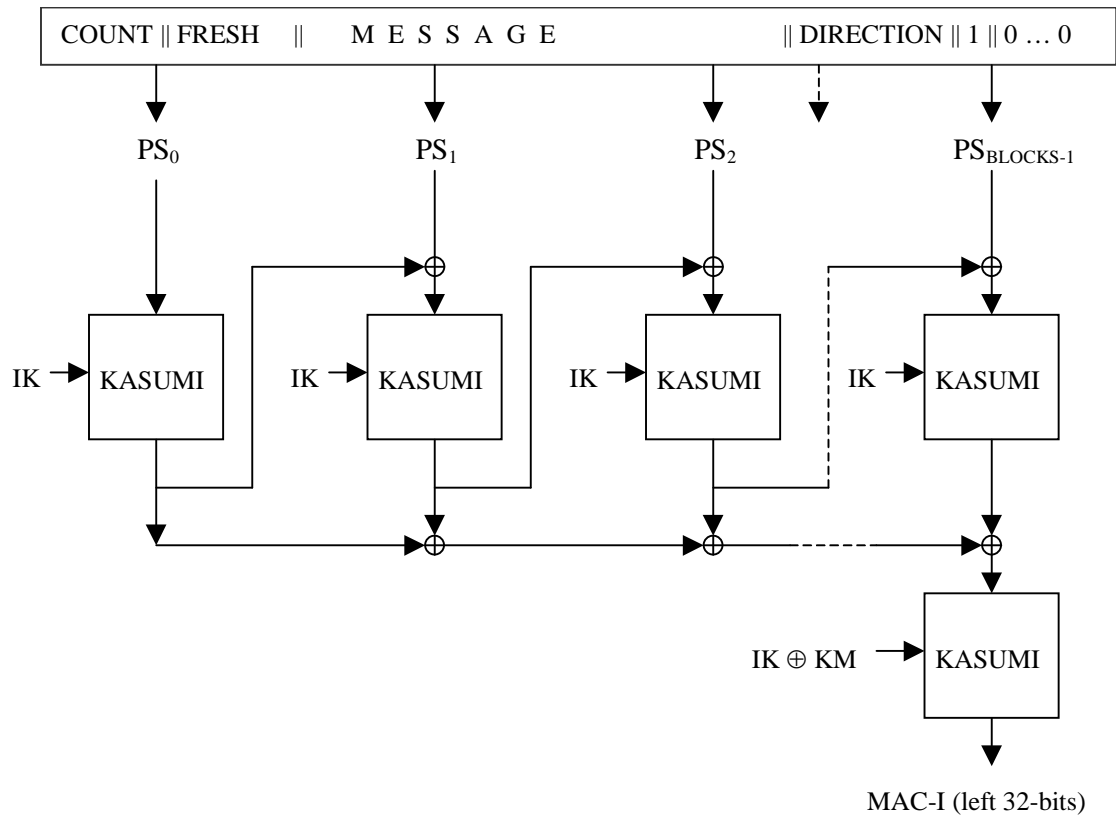


Figure 2: f_9 Integrity function

Annex 2 (informative): Simulation Program Listing

Header file

```

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

typedef unsigned char  u8;
typedef unsigned short u16;
typedef unsigned long  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 );
u8 * f9( u8 *key,int count,int fresh, int dir,u8 *data,int length );
void f8( u8 *key,int count,int bearer,int dir,u8 *data,int length );

```

Function f8

```

/*-----
 *                               F8 - Confidentiality Algorithm
 *-----
 *
 * A sample implementation of f8, the 3GPP Confidentiality algorithm.
 * 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 1.0      05 November 1999
 *
 *-----*/

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

/*-----
 * f8()
 * Given key, count, bearer, direction, data,
 * and bit length encrypt the bit stream
 *-----*/
void f8( u8 *key, int count, int bearer, int dir, u8 *data, int length )
{
    REGISTER64 A;      /* the modifier          */
    REGISTER64 temp;   /* The working register */
    int i, n;
    u8 ModKey[16];     /* Modified key          */
    u16 blkcnt;        /* The block counter    */

    /* 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) (count>>24);
    A.b8[1] = (u8) (count>>16);
    A.b8[2] = (u8) (count>>8);
    A.b8[3] = (u8) (count);

```

```

A.b8[4] = (u8) (bearer<<3);
A.b8[4] |= (u8) (dir<<2);

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

for( n=0; n<16; ++n )
    ModKey[n] = (u8)(key[n] ^ 0x55);
KeySchedule( ModKey );

Kasumi( A.b8 ); /* First encryption to create modifier */

/* Final initialisation steps */

blkcnt = 0;
KeySchedule( key );

/* Now run the block cipher */

while( length > 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] ^= (u8) blkcnt;
    temp.b8[6] ^= (u8) (blkcnt>>8);

    /* 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( length >= 64 )
        n = 8;
    else
        n = (length+7)/8;

    /* XOR the keystream with the input data stream */

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

/*-----
 *           e n d   o f   f 8 . c
 *-----*/

```

Function f9

```

/*-----
 *           F9 - Integrity Algorithm
 *-----
 *
 * A sample implementation of f9, the 3GPP Integrity algorithm.
 *
 * 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      05 September 2000
 *
 *-----*/

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

/*-----
 * f9()
 * Given key, count, fresh, direction, data,
 * and message length, calculate the hash value

```

```

*-----*/
u8 *f9( u8 *key, int count, int fresh, int dir, u8 *data, int length )
{
    REGISTER64 A; /* Holds the CBC chained data */
    REGISTER64 B; /* Holds the XOR of all KASUMI outputs */
    u8 FinalBit[8] = {0x80, 0x40, 0x20, 0x10, 8,4,2,1};
    u8 ModKey[16];
    static u8 mac_i[4]; /* static memory for the result */
    int i, n;

    /* Start by initialising the block cipher */

    KeySchedule( key );

    /* Next initialise the MAC chain. Make sure we *
     * have the data in the right byte order. *
     * <A> holds our chaining value... *
     * <B> is the running XOR of all KASUMI o/ps */

    for( n=0; n<4; ++n )
    {
        A.b8[n] = (u8)(count>>(24-(n*8)));
        A.b8[n+4] = (u8)(fresh>>(24-(n*8)));
    }
    Kasumi( A.b8 );
    B.b32[0] = A.b32[0];
    B.b32[1] = A.b32[1];

    /* Now run the blocks until we reach the last block */

    while( length >= 64 )
    {
        for( n=0; n<8; ++n )
            A.b8[n] ^= *data++;
        Kasumi( A.b8 );
        length -= 64;
        B.b32[0] ^= A.b32[0]; /* running XOR across */
        B.b32[1] ^= A.b32[1]; /* the block outputs */
    }

    /* Process whole bytes in the last block */

    n = 0;
    while( length >=8 )
    {
        A.b8[n++] ^= *data++;
        length -= 8;
    }

    /* Now add the direction bit to the input bit stream *
     * If length (which holds the # of data bits in the *
     * last byte) is non-zero we add it in, otherwise *
     * it has to start a new byte. */

    if( length )
    {
        i = *data;
        if( dir )
            i |= FinalBit[length];
    }
    else
        i = dir ? 0x80 : 0;

    A.b8[n++] ^= (u8)i;

    /* Now add in the final '1' bit. The problem here *
     * is if the message length happens to be n*64-1. *
     * If so we need to process this block and then *
     * create a new input block of 0x8000000000000000. */

    if( (length==7) && (n==8) ) /* then we've filled the block */
    {
        Kasumi( A.b8 );
        B.b32[0] ^= A.b32[0]; /* running XOR across */
        B.b32[1] ^= A.b32[1]; /* the block outputs */

        A.b8[0] ^= 0x80; /* toggle first bit */
    }
}

```

```
        i = 0x80;
        n = 1;
    }
    else
    {
        if( length == 7 )          /* we finished off the last byte */
            A.b8[n] ^= 0x80;      /* so start a new one..... */
        else
            A.b8[n-1] ^= FinalBit[length+1];
    }

    Kasumi( A.b8 );
    B.b32[0] ^= A.b32[0]; /* running XOR across */
    B.b32[1] ^= A.b32[1]; /* the block outputs */

    /* Final step is to KASUMI what we have using the *
    * key XORd with 0xAAAA..... */

    for( n=0; n<16; ++n )
        ModKey[n] = (u8)*key++ ^ 0xAA;
    KeySchedule( ModKey );
    Kasumi( B.b8 );

    /* We return the left-most 32-bits of the result */

    for( n=0; n<4; ++n )
        mac_i[n] = B.b8[n];

    return( mac_i );
}

/*-----
 *           e n d   o f   f 9 . c
 *-----*/
```

Annex 3 (informative): Change history

Change history							
Date	TSG #	TSG Doc.	CR	Rev	Subject/Comment	Old	New
12-1999	-	-	-	-	ETSI SAGE Publication (restricted)	-	SAGE v1.0
05-2000	-	-	-	-	ETSI SAGE update: Small change to sample code (16-bit portability issue)	SAGE v1.0	SAGE v1.1
09-2000	-	-	-	-	ETSI SAGE update: Small change to sample f9 code (boundary condition)	SAGE v1.1	SAGE v1.2
09-2000	SA_07				Approved by TSG SA and placed under change control	SAGE v1.2	3.1.0
07-2001	-	-	-	-	Word version received: Re-formatted into 3GPP TS format (MCC) No technical change from version 3.1.0.	3.1.0	3.1.1
08-2001	-				Addition of Mitsubishi IPR information in Foreword and correction of reference titles. No technical change from version 3.1.0.	3.1.1	3.1.2
08-2001	-	-	-	-	Release 4 version created.	3.1.2	4.0.0
12-2001	SP-14	SP-010620	002		Correct the maximum input message length for f8 and f9	4.0.0	4.1.0
06-2002	SP-16	-	-	-	Upgrade to Release 5	4.1.0	5.0.0
12-2004	SP-26	-	-	-	Upgrade to Release 6	5.0.0	6.0.0
2005-09	SP-29	SP-050563	0003	-	Correction of sample code	6.0.0	6.1.0
06-2007	SP-36	-	-	-	Upgrade to Release 7	6.1.0	7.0.0
12-2008	SP-42	--	-	-	Upgrade to Release 8	7.0.0	8.0.0
2009-12	-	-	-	-	Upgrade to Release 9	8.0.0	9.0.0