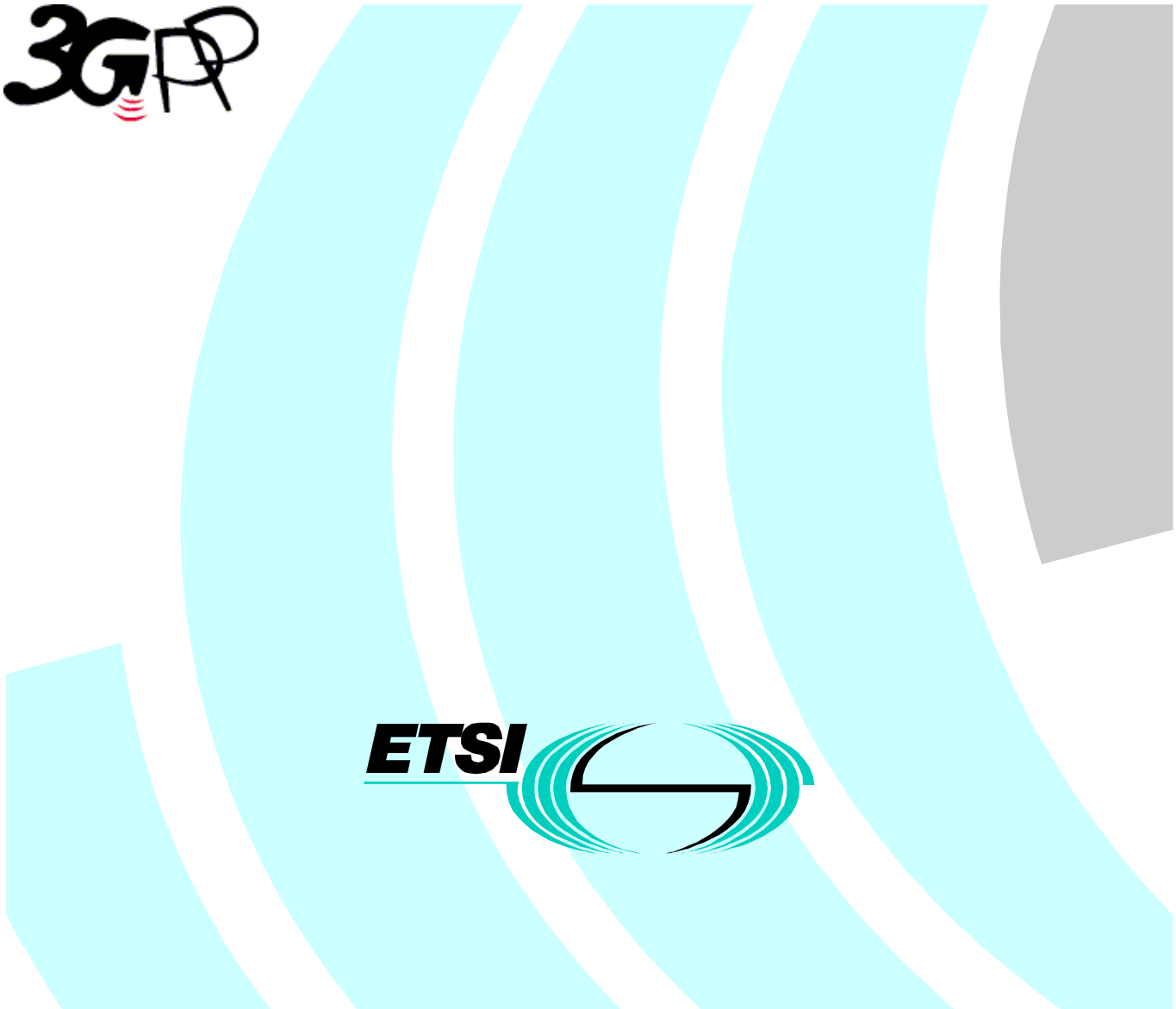
ETSI TS 135 206 V4.0.0 (2001-04)

Technical Specification

Universal Mobile Telecommunications System (UMTS);
3G Security;
Specification of the MILENAGE algorithm set:
An example algorithm Set for the 3GPP Authentication and Key Generation functions f1, f1*, f2, f3, f4, f5 and f5*;
Document 2: Algorithm Specification
(3GPP TS 35.206 version 4.0.0 Release 4)



Reference
RTS/TSGS-0335206Uv4

Keywords
UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://www.etsi.org/ipr).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

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

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under www.etsi.org/key .

Contents

Foreword

This Technical Specification (TS) 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 document has been prepared by the 3GPP Task Force, and contains an example set of algorithms which may be used as the authentication and key generation functions ***f1***, ***f1****, ***f2***, ***f3***, ***f4***, ***f5*** and ***f5****. (It is not mandatory that the particular algorithms specified in this document are used — all seven functions are operator-specifiable rather than being fully standardised). This document is one five, which between them form the entire specification of the example algorithms, entitled:

- 3GPP TS 35.205: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*;
  Document 1: General".

- 3GPP TS 35.206: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*;
  Document 2: Algorithm Specification".

- 3GPP TS 35.207: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*;
  Document 3: Implementors' Test Data".

- 3GPP TS 35.208: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*;
  Document 4: Design Conformance Test Data".

- 3GPP TR 35.909: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*;
  Document 5: Summary and results of design and evaluation".

0 The name "MILENAGE"

The name of this algorithm set is "MILENAGE". It should be pronounced like a French word — something like "**mi-le-nahj**".

1 Outline of the document

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

Section 3 explains how the algorithms are designed as a framework in such a way that various "customising components" can be selected in order to customise the algorithm for a particular operator.

Section 4 defines the example algorithms. The algorithm framework is defined in section 4.1; in section 4.2, specific instances of the components are selected to define the specific example algorithm set.

Section 5 explains various options and considerations for implementation of the algorithms, including considerations to be borne in mind when modifying the customising components.

Illustrative pictures are given in Annex 1. Annex 2 gives a specification of the block cipher algorithm which is used as a cryptographic kernel in the definition of the example algorithms. Annexes 3 and 4 contain source code in the C programming language: Annex 3 gives a complete and straightforward implementation of the algorithm set, while Annex 4 gives an example of an alternative high-performance implementation just of the kernel function.

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 v3.5.0: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Security Architecture".

[2] 3GPP TS 33.105 v3.4.0: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Cryptographic Algorithm Requirements".

[3] 3GPP TS 35.206: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm Specification" (this document).

[4] 3GPP TS 35.207: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 3: Implementors' Test Data".

[5] 3GPP TS 35.208: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 4: Design Conformance Test Data".

[6] Joan Daemen and Vincent Rijmen: "AES Proposal: Rijndael", available at http://csrc.nist.gov/encryption/aes/round2/AESAlgs/Rijndael/Rijndael.pdf or http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaeldocV2.zip

[7] http://csrc.nist.gov/encryption/aes/

[8] Thomas S. Messerges, "Securing the AES finalists against Power Analysis Attacks", in FSE 2000, Seventh Fast Software Encryption Workshop, ed. Schneier, Springer Verlag, 2000.

[9] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", in CRYPTO'96, Lecture Notes in Computer Science #1109, Springer Verlag, 1996.

[10] J. Kelsey, B. Schneier, D. Wagner, C. Hall, "Side Channel Cryptanalysis of Product Ciphers", in ESORICS'98, Lecture Notes in Computer Science #1485, Springer Verlag, 1998.

[11] L. Goubin, J. Patarin, "DES and differential power analysis", in CHES'99, Lecture Notes in Computer Science #1717, Springer Verlag, 1999.

[12] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis", in CRYPTO'99, Lecture Notes in Computer Science #1666, Springer Verlag, 1999.

[13] L. Goubin, J.-S. Coron, "On boolean and arithmetic masking against differential power analysis", in CHES'00, Lecture Notes in Computer Science series, Springer Verlag (to appear).

2 INTRODUCTORY INFORMATION

2.1 Introduction

Within the security architecture of the 3GPP system there are seven security functions ***f1***, ***f1****, ***f2***, ***f3***, ***f4***, ***f5*** and ***f5****. The operation of these functions falls within the domain of one operator, and the functions are therefore to be specified by each operator rather than being fully standardised. The algorithms specified in this document are examples that may be used by an operator who does not wish to design his own.

The inputs and outputs of all seven algorithms are defined in section 2.4.

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 substrings, the

leftmost (most significant) substring is numbered 0, the next most significant is numbered 1, and so on through to the least significant.

2.2.4 List of Symbols

=	The assignment operator.
⊕	The bitwise exclusive-OR operation
||	The concatenation of the two operands.
$E[x]_k$	The result of applying a block cipher encryption to the input value **x** using the key **k**.
rot(x,r)	The result of cyclically rotating the 128-bit value **x** by **r** bit positions towards the most significant bit. If **x** = **x[0]** || **x[1]** || … **x[127]**, and **y** = rot(**x**,**r**), then **y** = **x[r]** || **x[r+1]** || … **x[127]** || **x[0]** || **x[1]** || **x[r-1]**.
X[i]	The i^{th} bit of the variable **X**. (**X = X[0] || X[1] || X[2] || …..**).

2.3 List of Variables

AK	a 48-bit anonymity key that is the output of either of the functions ***f5*** and ***f5****.
AMF	a 16-bit authentication management field that is an input to the functions ***f1*** and ***f1****.
c1,c2,c3,c4,c5	128-bit constants, which are XORed onto intermediate variables.
CK	a 128-bit confidentiality key that is the output of the function ***f3***.
IK	a 128-bit integrity key that is the output of the function ***f4***.
IN1	a 128-bit value constructed from **SQN** and **AMF** and used in the computation of the functions ***f1*** and ***f1****.
K	a 128-bit subscriber key that is an input to the functions ***f1***, ***f1****, ***f2***, ***f3***, ***f4***, ***f5*** and ***f5****.
MAC-A	a 64-bit network authentication code that is the output of the function ***f1***.
MAC-S	a 64-bit resynchronisation authentication code that is the output of the function ***f1****.
OP	a 128-bit Operator Variant Algorithm Configuration Field that is a component of the functions ***f1***, ***f1****, ***f2***, ***f3***, ***f4***, ***f5*** and ***f5****.
OP_C	a 128-bit value derived from **OP** and **K** and used within the computation of the functions.
OUT1,OUT2,OUT3,OUT4,OUT5	128-bit computed values from which the outputs of the functions ***f1***, ***f1****, ***f2***, ***f3***, ***f4***, ***f5*** and ***f5**** are obtained.
r1,r2,r3,r4,r5	integers in the range 0–127 inclusive, which define amounts by which intermediate variables are cyclically rotated.
RAND	a 128-bit random challenge that is an input to the functions ***f1***, ***f1****, ***f2***, ***f3***, ***f4***, ***f5*** and ***f5****.
RES	a 64-bit signed response that is the output of the function ***f2***.
SQN	a 48-bit sequence number that is an input to either of the functions ***f1*** and ***f1****. (For ***f1**** this input is more precisely called SQN_{MS}.)
TEMP	a 128-bit value used within the computation of the functions.

2.4 Algorithm Inputs and Outputs

The inputs to the algorithms are given in tables 1 and 2, the outputs in tables 3–9 below.

Table 1: inputs to *f1* and *f1**

Parameter	Size (bits)	Comment
K	128	Subscriber key K[0]…K[127]
RAND	128	Random challenge RAND[0]…RAND[127]
SQN	48	Sequence number SQN[0]…SQN[47]. (For ***f1**** this input is more precisely called SQN_{MS}.)
AMF	16	Authentication management field AMF[0]…AMF[15]

Table 2: inputs to *f2*, *f3*, *f4*, *f5* and *f5**

Parameter	Size (bits)	Comment
K	128	Subscriber key K[0]…K[127]
RAND	128	Random challenge RAND[0]…RAND[127]

Table 3: *f1* **output**

Parameter	Size (bits)	Comment
MAC-A	64	Network authentication code MAC-A[0]…MAC-A[63]

Table 4: *f1** **output**

Parameter	Size (bits)	Comment
MAC-S	64	Resynch authentication code MAC-S[0]…MAC-S[63]

Table 5. *f2* **output**

Parameter	Size (bits)	Comment
RES	64	Response RES[0]…RES[63]

Table 6. *f3* **output**

Parameter	Size (bits)	Comment
CK	128	Confidentiality key CK[0]…CK[127]

Table 7. *f4* **output**

Parameter	Size (bits)	Comment
IK	128	Integrity key IK[0]…IK[127]

Table 8. *f5* **output**

Parameter	Size (bits)	Comment
AK	48	Anonymity key AK[0]…AK[47]

Table 9. *f5** **output**

Parameter	Size (bits)	Comment
AK	48	Resynch anonymity key AK[0]…AK[47]

Note: Both f5 and f5* outputs are called AK according to reference [2]. In practice only one of them will be calculated in each instance of the authentication and key agreement procedure.

3 The algorithm framework and the specific example algorithms

The example algorithm set makes use of the following components:

- A block cipher encryption function, which takes a 128-bit input and a 128-bit key and returns a 128-bit output. If the input is **x**, the key is **k** and the output is **y**, we write $\mathbf{y} = E[\mathbf{x}]_{\mathbf{k}}$.

- A 128-bit value **OP**. This is an Operator Variant Algorithm Configuration Field, which the Task Force was asked to include as a simple means to provide separation between the functionality of the algorithms when used by different operators. It is left to each operator to select a value for **OP**. The algorithm set is designed to be secure whether or not **OP** is publicly known; however, operators may see some advantage in keeping their value of **OP** secret. This and other aspects of the use of **OP** are discussed further in section 5.

In the specific example algorithm set, a particular block cipher is used. But the algorithms have been designed so that this component can be replaced by any operator who wishes to create his own customised algorithm set. In that sense this document defines an algorithm framework, and the example algorithm set is one that fits within the framework. This is how the algorithm set is defined in section 4: in section 4.1 the framework is defined in terms of the block cipher, and then in section 4.2 a block cipher is selected to give a fully specified algorithm set.

4 Definition of the example algorithms

4.1 Algorithm Framework

A 128-bit value $\mathbf{OP_C}$ is derived from **OP** and **K** as follows:

$\mathbf{OP_C} = \mathbf{OP} \oplus E[\mathbf{OP}]_{\mathbf{K}}$.

An intermediate 128-bit value **TEMP** is computed as follows:

$\mathbf{TEMP} = E[\mathbf{RAND} \oplus \mathbf{OP_C}]_{\mathbf{K}}$.
A 128-bit value **IN1** is constructed as follows:
IN1[0] .. **IN1**[47] = **SQN**[0] .. **SQN**[47]
IN1[48] .. **IN1**[63] = **AMF**[0] .. **AMF**[15]
IN1[64] .. **IN1**[111] = **SQN**[0] .. **SQN**[47]
IN1[112] .. **IN1**[127] = **AMF**[0] .. **AMF**[15]

Five 128-bit constants **c1**, **c2**, **c3**, **c4**, **c5** are defined as follows:

c1[i] = 0 for $0 \le i \le 127$
c2[i] = 0 for $0 \le i \le 127$, except that **c2**[127] = 1
c3[i] = 0 for $0 \le i \le 127$, except that **c3**[126] = 1
c4[i] = 0 for $0 \le i \le 127$, except that **c4**[125] = 1
c5[i] = 0 for $0 \le i \le 127$, except that **c5**[124] = 1
Five integers **r1**, **r2**, **r3**, **r4**, **r5** are defined as follows:
r1 = 64; **r2** = 0; **r3** = 32; **r4** = 64; **r5** = 96

Five 128-bit blocks **OUT1**, **OUT2**, **OUT3**, **OUT4**, **OUT5** are computed as follows:

$\mathbf{OUT1} = E[\mathbf{TEMP} \oplus \text{rot}(\mathbf{IN1} \oplus \mathbf{OP_C}, \mathbf{r1}) \oplus \mathbf{c1}]_{\mathbf{K}} \oplus \mathbf{OP_C}$
$\mathbf{OUT2} = E[\text{rot}(\mathbf{TEMP} \oplus \mathbf{OP_C}, \mathbf{r2}) \oplus \mathbf{c2}]_{\mathbf{K}} \oplus \mathbf{OP_C}$
$\mathbf{OUT3} = E[\text{rot}(\mathbf{TEMP} \oplus \mathbf{OP_C}, \mathbf{r3}) \oplus \mathbf{c3}]_{\mathbf{K}} \oplus \mathbf{OP_C}$
$\mathbf{OUT4} = E[\text{rot}(\mathbf{TEMP} \oplus \mathbf{OP_C}, \mathbf{r4}) \oplus \mathbf{c4}]_{\mathbf{K}} \oplus \mathbf{OP_C}$
$\mathbf{OUT5} = E[\text{rot}(\mathbf{TEMP} \oplus \mathbf{OP_C}, \mathbf{r5}) \oplus \mathbf{c5}]_{\mathbf{K}} \oplus \mathbf{OP_C}$

The outputs of the various functions are then defined as follows:

Output of ***f1*** = MAC-A, where MAC-A[0] .. MAC-A[63] = **OUT1**[0] .. **OUT1**[63]
Output of ***f1**** = MAC-S, where MAC-S[0] .. MAC-S[63] = **OUT1**[64] .. **OUT1**[127]
Output of ***f2*** = RES, where RES[0] .. RES[63] = **OUT2**[64] .. **OUT2**[127]
Output of ***f3*** = CK, where CK[0] .. CK[127] = **OUT3**[0] .. **OUT3**[127]
Output of ***f4*** = IK, where IK[0] .. IK[127] = **OUT4**[0] .. **OUT4**[127]
Output of ***f5*** = AK, where AK[0] .. AK[47] = **OUT2**[0] .. **OUT2**[47]
Output of ***f5**** = AK, where AK[0] .. AK[47] = **OUT5**[0] .. **OUT5**[47]

(The repeated reference to AK is not a mistake: AK is the name of the output of either ***f5*** or ***f5****, and these two functions will not in practice be computed simultaneously.)

4.2 Specific Example Algorithms

The specific example algorithm set is defined by specifying the block cipher encryption function E[], which we do in this section. (It is left to each operator to specify the Operator Variant Algorithm Configuration Field **OP**.)

The block cipher selected is Rijndael [6]. This is the algorithm proposed as the Advanced Encryption Standard [7]. More precisely, it is Rijndael with 128-bit key and 128-bit block size.

$E[\mathbf{x}]_{\mathbf{k}}$ = the result of applying the Rijndael encryption algorithm
to the 128-bit value **x** under the 128-bit key **k**.

Although the definitive specification of Rijndael is in [6], a complete specification of Rijndael with 128-bit key and 128-bit block size is also given in Annex 2 of this document.

The inputs to and output of Rijndael are defined as strings of bytes. The 128-bit string **x** = **x[0]** || **x[1]** || … **x[127]** is treated as a string of bytes by taking **x[0]** || **x[1]** || … **x[7]** as the first byte, **x[8]** || **x[9]** || … **x[15]** as the second byte, and so on. The key and output string are converted in the same way.

Note that the following patent statement has been made publicly (and included in [6]) by the authors of the Rijndael algorithm: "Rijndael or any of its implementations is not and will not be subject to patents."

5 Implementation considerations

5.1 OP_C computed on or off the USIM?

Recall that **OP** is an Operator Variant Algorithm Configuration Field. It is expected that each operator will define a value of **OP** which will then be used for all its subscribers. (It is up to operators to decide how to manage **OP**. The value of **OP** used for new batches of USIMs could be changed occasionally; or perhaps a different value could be given to each different USIM supplier. **OP** could even be given a different value for every subscriber if desired, but that is not really the intention.)

It will be seen in section 4.1 that $\mathbf{OP_C}$ is computed from **OP** and **K**, and that it is only $\mathbf{OP_C}$, not **OP**, that is ever used in subsequent computations. This gives two alternative options for implementation of the algorithms on the USIM:

(a) $\mathbf{OP_C}$ computed off the USIM: $\mathbf{OP_C}$ is computed as part of the USIM prepersonalisation process, and $\mathbf{OP_C}$ is stored on the USIM. **OP** itself is not stored on the USIM.

(b) $\mathbf{OP_C}$ computed on the USIM: **OP** is stored on the USIM (it may be considered as a hard-coded part of the algorithm if preferred). $\mathbf{OP_C}$ is recomputed each time the algorithms are called.

The SAGE Task Force recommends that $\mathbf{OP_C}$ be computed off the USIM if possible, since this gives the following benefits:

- The complexity of the algorithms run on the USIM is reduced.

- It is more likely that **OP** can be kept secret. (If **OP** is stored on the USIM, it only takes one USIM to be reverse engineered for **OP** to be discovered and published. But it should be difficult for someone who has discovered even a large number of ($\mathbf{OP_C}$, **K**) pairs to deduce **OP**. That means that the $\mathbf{OP_C}$ associated with any other value of **K** will be unknown, which may make it harder to mount some kinds of cryptanalytic and forgery attacks. The algorithms are designed to be secure whether or not **OP** is known to the attacker, but a secret **OP** is one more hurdle in the attacker's path.)

5.2 Customising the choice of block cipher

It was explained in section 3 that an operator may create a variant algorithm set by selecting a block cipher other than Rijndael. It is vitally important that whatever block cipher is chosen is one that has been extensively analysed and is still believed to be secure. The security of the authentication and key generation functions is crucially dependent on the strength of the block cipher.

Strictly speaking, in fact, the kernel function does not have to be a block cipher; it just has to be a keyed function (with 128-bit input, key and output) satisfying the following cryptographic requirement:

- Let the key be fixed. Without initial knowledge of the key, but with a large number of pairs of chosen input and resulting output, it must be infeasible to determine the key, and also infeasible to predict the output for any other chosen input with probability significantly greater than 2^{-128}.

See also section 5.4 about protecting against side channel attacks; this will need to be borne in mind when selecting/implementing a replacement kernel function.

5.3 Further customisation

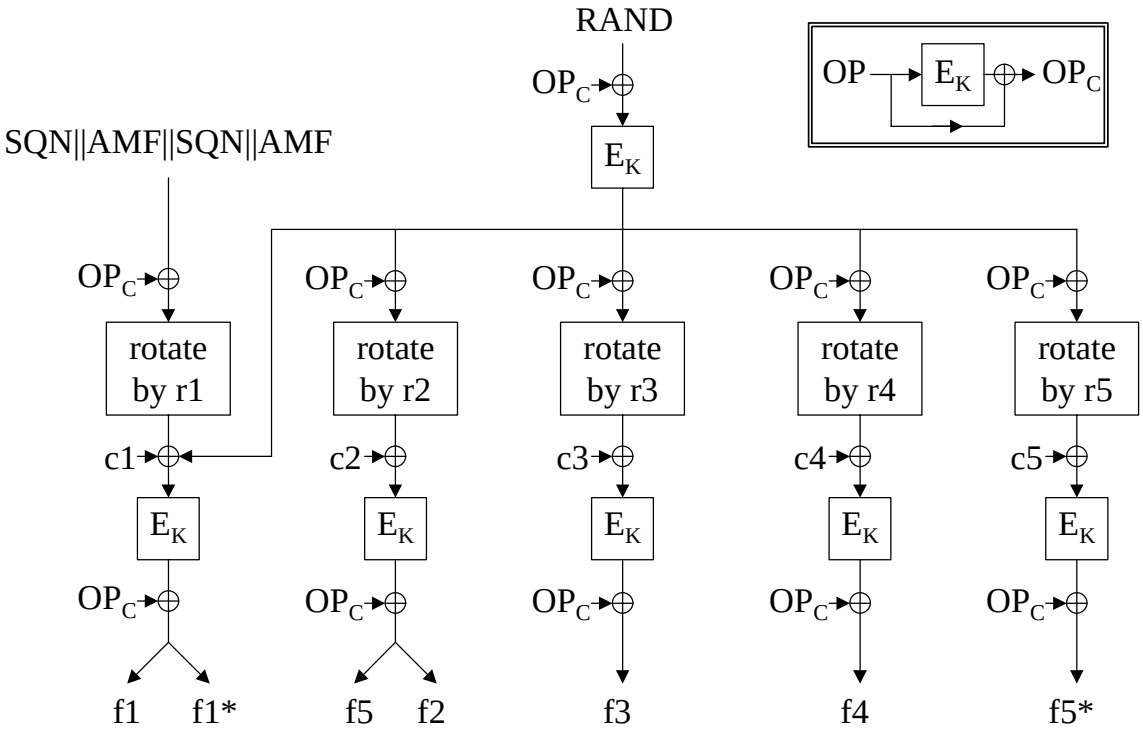
If an operator wishes to customise the algorithms still further, a simple approach is to select different values for the constants **c1**–**c5** and **r1**–**r5**. If this is done, the pairs (**ci**, **ri**) must all be different. It must not be the case that both **ci** = **cj** and **ri** = **rj** for **i** ≠ **j**. For instance, it must not be the case that both **c2** = **c4** and **r2** = **r4**. Additionally it is recommended that the following restrictions are applied:

- **c1** has even parity. (A 128-bit value has even parity if the number of '1's in its binary representation is even.)
- **c2**–**c5** all have odd parity.

5.4 Resistance to side channel attacks

When these algorithms are implemented on a USIM, consideration should be given to protecting them against side channel attacks such as differential power analysis (DPA). [8, 9, 10, 11, 12, 13] may be useful references.

Annex 1:
Figure of the Algorithms



Annex 2:
Specification of the Block Cipher Algorithm Rijndael

A2.1 Introduction

This section provides a specification for the example kernel function. The block cipher used is Rijndael. The complete specification of Rijndael is given elsewhere [6]. To quote from [6]:

> "Rijndael is an iterated block cipher with a variable block length and a variable key length. The block length and the key length can be independently specified to 128, 192 or 256 bits."

For 3GPP purpose, Rijndael is used only in encryption mode and has the block and key length both set to 128 bits. For the rest of this section, when we refer to Rijndael we mean Rijndael with 128-bit block and key length. This document describes a simple byte oriented implementation of the encryption mode of Rijndael. Readers wishing more detail on the design of the cipher or implementation speed-ups are referred to the original document [6].

A2.2 The State and External Interfaces of Rijndael

Rijndael is composed of a series of rounds that transform the input into the output. An intermediate result is called the **State**. The **State** can be pictured as a 4x4 rectangular array of bytes (128 bits in total). The Cipher Key is similarly pictured as a 4x4 rectangular array. These representations are illustrated in Figure 1.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Figure 1: Example of State and Cipher Key layout

Rijndael takes plaintext bytes $\mathbf{P_0}$, $\mathbf{P_1}$, …,$\mathbf{P_{15}}$ and key bytes $\mathbf{K_0}$, $\mathbf{K_1}$, …$\mathbf{K_{15}}$ as input and ciphertext bytes $\mathbf{C_0}$, $\mathbf{C_1}$, …, $\mathbf{C_{15}}$ as output. The plaintext bytes are mapped onto the state bytes in the order $\mathbf{a_{0,0}}$, $\mathbf{a_{1,0}}$, $\mathbf{a_{2,0}}$, $\mathbf{a_{3,0}}$, $\mathbf{a_{0,1}}$, $\mathbf{a_{1,1}}$, $\mathbf{a_{2,1}}$, $\mathbf{a_{3,1}}$,… and the key bytes in the order $\mathbf{k_{0,0}}$, $\mathbf{k_{1,0}}$, $\mathbf{k_{2,0}}$, $\mathbf{k_{3,0}}$, $\mathbf{k_{0,1}}$, $\mathbf{k_{1,1}}$, $\mathbf{k_{2,1}}$, $\mathbf{k_{3,1}}$,…. At the end of the cipher operation, the ciphertext is extracted from the **State** by taking the **State** bytes in the same order.

Hence if the one-dimensional index of a byte within a block is **n** and the two dimensional index is (**i, j**), we have:

$$\mathbf{i} = \mathbf{n} \bmod 4; \quad \mathbf{j} = \lfloor \mathbf{n}/4 \rfloor; \quad \mathbf{n} = \mathbf{i} + 4 * \mathbf{j}$$

A2.3 Internal Structure

Rijndael consists of the following operations:

- an initial Round Key addition
- 9 rounds, numbered 1-9, each consisting of
  - a byte substitution transformation
  - a shift row transformation
  - a mix column transformation
  - a Round Key addition
- A final round (round 10) consisting of
  - a byte substitution transformation
  - a shift row transformation
  - a Round Key addition

The component transformations and how the Round Keys are derived from the cipher keys are specified in the following subsections.

A2.4 The Byte Substitution Transformation

The byte substitution transformation is a non-linear byte substitution, operating on each of the **State** bytes independently. The substitution table (S-box) is given in section A2.9.

Figure 2 illustrates the effect of the byte substitution transformation on the **State**.

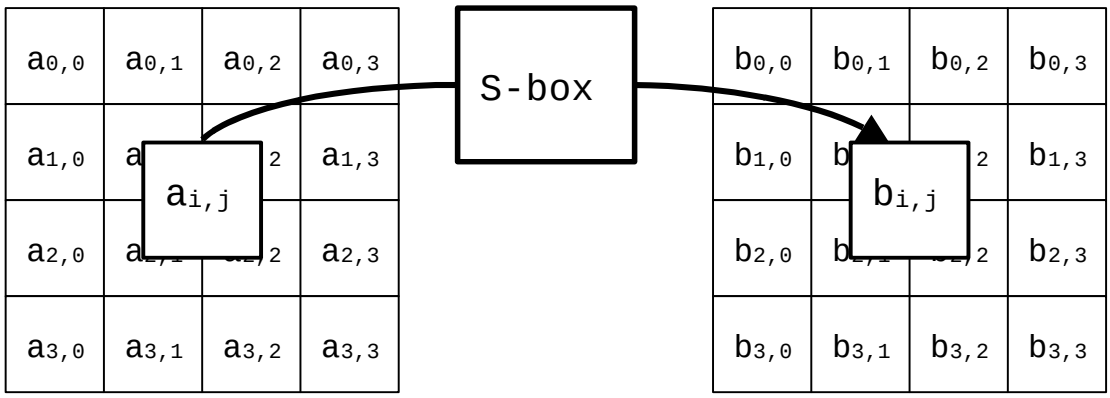


Figure 2: Byte substitution acts on the individual bytes of the State

So, for every element of **State,** we apply the transformation:

$$\mathbf{b}_{i,j} = \text{S-box}[\ \mathbf{a}_{i,j}\ ]$$

Where $\mathbf{a}_{i,j}$ is the initial value of the element in **State**,
and $\mathbf{b}_{i,j}$ is the output value of the element in **State**.

A2.5 The Shift Row Transformation

In the shift row transformation, the rows of the **State** are cyclically left shifted by different amounts. Row 0 is not shifted, row 1 is shifted by 1 byte, row 2 by 2 bytes and row 3 by 3 bytes.

Figure 3 illustrates the effect of the shift row transformation on the **State**.

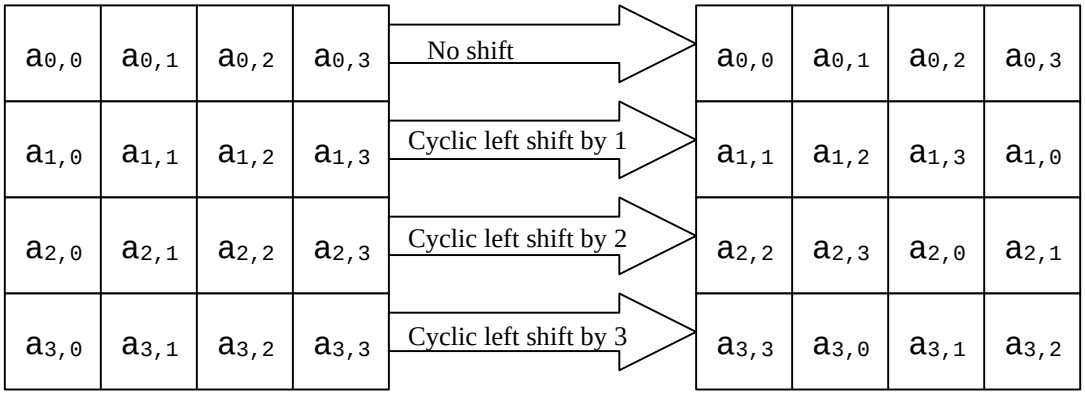


Figure 3: Shift Row operates on the rows of the State

A2.6 The Mix Column Transformation

The mix column transformation operates on each column of the **State** independently. For column `j`, we have

$$\mathbf{b}_{0,j} = T_2(\mathbf{a}_{0,j}) \oplus T_3(\mathbf{a}_{1,j}) \oplus \mathbf{a}_{2,j} \oplus \mathbf{a}_{3,j}$$
$$\mathbf{b}_{1,j} = \mathbf{a}_{0,j} \oplus T_2(\mathbf{a}_{1,j}) \oplus T_3(\mathbf{a}_{2,j}) \oplus \mathbf{a}_{3,j}$$
$$\mathbf{b}_{2,j} = \mathbf{a}_{0,j} \oplus \mathbf{a}_{1,j} \oplus T_2(\mathbf{a}_{2,j}) \oplus T_3(\mathbf{a}_{3,j})$$
$$\mathbf{b}_{3,j} = T_3(\mathbf{a}_{0,j}) \oplus \mathbf{a}_{1,j} \oplus \mathbf{a}_{2,j} \oplus T_2(\mathbf{a}_{3,j})$$

where:

$T_2(\mathbf{a}) = 2*\mathbf{a}$ if $\mathbf{a} < 128$
or $T_2(\mathbf{a}) = (2*\mathbf{a}) \oplus 283$ if $\mathbf{a} \geq 128$
and $T_3(\mathbf{a}) = T_2(\mathbf{a}) \oplus \mathbf{a}$.

For example:

If $\mathbf{a} = 63$ then $T_2(63) = 126$; $T_3(63) = T_2(63) \oplus 63 = 65$
If $\mathbf{a} = 143$ then $T_2(143) = 5$; $T_3(143) = T_2(143) \oplus 143 = 138$.

Figure 4 illustrates the effect of the mix column transformation on the **State**.

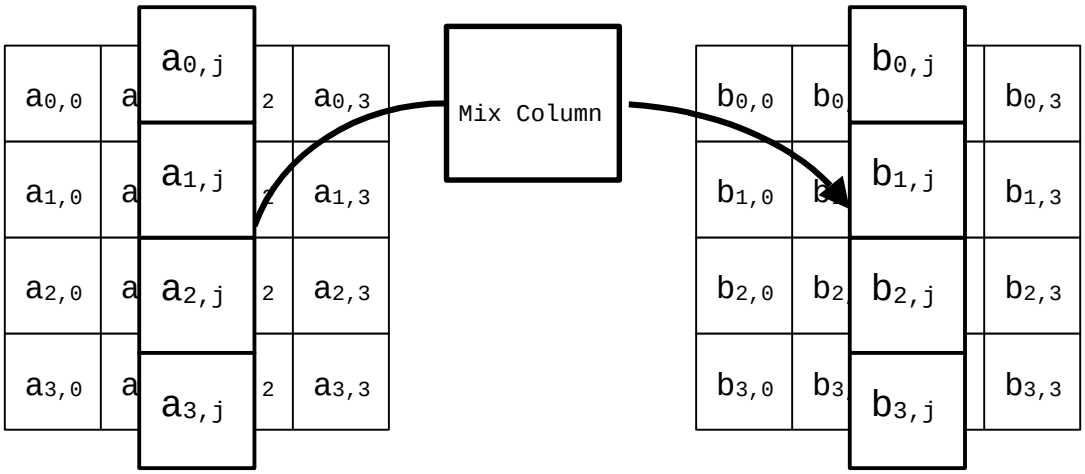


Figure 4: Mix Column operates on the columns of the State

A2.7 The Round Key addition

In this operation, a Round Key is applied to the **State** by a simple bitwise exclusive-or. The Round Key is derived from the Cipher Key by means of the key schedule. The Round Key length is equal to the block length.

This transformation is illustrated in Figure 5.

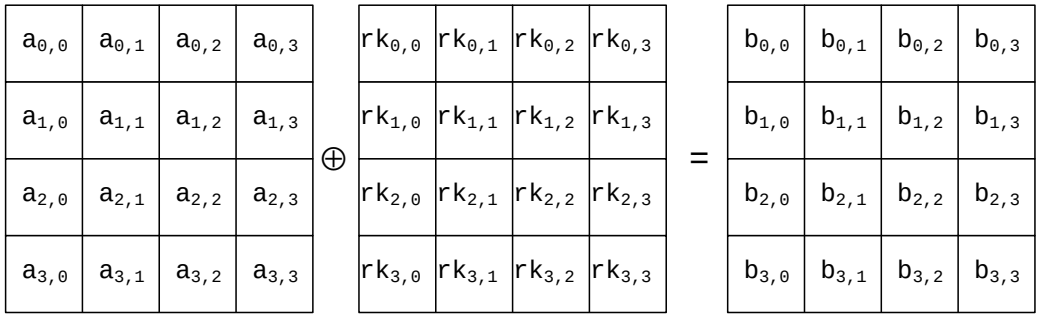


Figure 5: In the key addition the Round Key is bitwise XORed to the State

So, for every element of **State**, we have:

$$\mathbf{b_{i,j}} = \mathbf{a_{i,j}} \oplus \mathbf{rk_{i,j}}$$

Where $\mathbf{a_{i,j}}$ is the initial value of the element in **State**,
$\mathbf{b_{i,j}}$ is the output value of the element in **State**, and
$\mathbf{rk_{i,j}}$ is the round key byte.

A2.8 Key schedule

Rijndael has 11 Round Keys, numbered 0-10, that are each 4x4 rectangular arrays of bytes. The Round Keys are derived from the Cipher Key by means of the key schedule. The initial Round Key (thought of as the zeroth Round Key) is formed directly from the cipher key. This zeroth Round Key is used unaltered for the initial key addition. The remaining Round Keys are used in the ten rounds. Each new round key is derived from the previous round key. Note: It is possible to run the key schedule round by round on an "as required" basis and so only use a total of 16 bytes to store the Round Key.

Let $\mathbf{rk_{r,i,j}}$ be the value of the $\mathbf{r}^{th}$ Round Key at position $(\mathbf{i, j})$ in the array and $\mathbf{k_{i,j}}$ be the cipher key loaded into a 4x4 array.

Intialisation: $\mathbf{rk_{0,i,j}} = \mathbf{k_{i,j}}$ for all **i** and **j**.

The other Round Keys (**r**=1 to 10 inclusive) are calculated from the previous one as follows. First the 0^{th} column is constructed:

$$\mathbf{rk}_{r,0,0} = \mathbf{rk}_{r-1,0,0} \oplus \text{S-box}[\mathbf{rk}_{r-1,1,3}] \oplus \text{round\_const}[\mathbf{r}]$$
$$\mathbf{rk}_{r,1,0} = \mathbf{rk}_{r-1,1,0} \oplus \text{S-box}[\mathbf{rk}_{r-1,2,3}]$$
$$\mathbf{rk}_{r,2,0} = \mathbf{rk}_{r-1,2,0} \oplus \text{S-box}[\mathbf{rk}_{r-1,3,3}]$$
$$\mathbf{rk}_{r,3,0} = \mathbf{rk}_{r-1,3,0} \oplus \text{S-box}[\mathbf{rk}_{r-1,0,3}]$$

where round_const[1] = 1 and round_const[**r**] = T_2(round_const[**r**-1]), and S-box is the previously mentioned byte substitution.

Then the remaining three columns are constructed in turn from the corresponding column of the previous Round Key and the previous column of the current Round Key:

$$\mathbf{rk}_{r,i,j} = \mathbf{rk}_{r-1,i,j} \oplus \mathbf{rk}_{r,i,j-1} \text{ for } \mathbf{i}=0,1,2,3 \text{ and } \mathbf{j}=1,2,3.$$

Note: The ten round constants computed from the equations:

$$\text{round\_const}[1] = 1$$
$$\text{round\_const}[\mathbf{r}] = T_2(\text{round\_const}[\mathbf{r}-1]) \quad r=2,3\ldots,10$$

are: 1, 2, 4, 8, 16, 32, 64, 128, 27, 54.

A2.9 The Rijndael S-box

```
Sbox[256] = {
 99,124,119,123,242,107,111,197, 48,  1,103, 43,254,215,171,118,
202,130,201,125,250, 89, 71,240,173,212,162,175,156,164,114,192,
183,253,147, 38, 54, 63,247,204, 52,165,229,241,113,216, 49, 21,
  4,199, 35,195, 24,150,  5,154,  7, 18,128,226,235, 39,178,117,
  9,131, 44, 26, 27,110, 90,160, 82, 59,214,179, 41,227, 47,132,
 83,209,  0,237, 32,252,177, 91,106,203,190, 57, 74, 76, 88,207,
208,239,170,251, 67, 77, 51,133, 69,249,  2,127, 80, 60,159,168,
 81,163, 64,143,146,157, 56,245,188,182,218, 33, 16,255,243,210,
 96,129, 79,220, 34, 42,144,136, 70,238,184, 20,222, 94, 11,219,
224, 50, 58, 10, 73,  6, 36, 92,194,211,172, 98,145,149,228,121,
231,200, 55,109,141,213, 78,169,108, 86,244,234,101,122,174,  8,
186,120, 37, 46, 28,166,180,198,232,221,116, 31, 75,189,139,138,
112, 62,181,102, 72,  3,246, 14, 97, 53, 87,185,134,193, 29,158,
225,248,152, 17,105,217,142,148,155, 30,135,233,206, 85, 40,223,
140,161,137, 13,191,230, 66,104, 65,153, 45, 15,176, 84,187, 22};
```

Annex 3:
Simulation Program Listing - Byte Oriented

```
/*-----------------------------------------------------------------
 *          Example algorithms f1, f1*, f2, f3, f4, f5, f5*
 *-----------------------------------------------------------------
 *
 *  A sample implementation of the example 3GPP authentication and
 *  key agreement functions f1, f1*, f2, f3, f4, f5 and f5*.  This is
 *  a byte-oriented implementation of the functions, and of the block
 *  cipher kernel function Rijndael.
 *
 *  This has been coded for clarity, not necessarily for efficiency.
 *
 *  The functions f2, f3, f4 and f5 share the same inputs and have
 *  been coded together as a single function.  f1, f1* and f5* are
 *  all coded separately.
 *
 *-----------------------------------------------------------------*/

typedef unsigned char u8;


/*--------- Operator Variant Algorithm Configuration Field --------*/

            /*------- Insert your value of OP here -------*/
u8 OP[16] = {0x63, 0xbf, 0xa5, 0x0e, 0xe6, 0x52, 0x33, 0x65,
             0xff, 0x14, 0xc1, 0xf4, 0x5f, 0x88, 0x73, 0x7d};
            /*------- Insert your value of OP here -------*/


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

void f1    ( u8 k[16], u8 rand[16], u8 sqn[6], u8 amf[2],
             u8 mac_a[8] );
void f2345 ( u8 k[16], u8 rand[16],
             u8 res[8], u8 ck[16], u8 ik[16], u8 ak[6] );
void f1star( u8 k[16], u8 rand[16], u8 sqn[6], u8 amf[2],
             u8 mac_s[8] );
void f5star( u8 k[16], u8 rand[16],
             u8 ak[6] );
void ComputeOPc( u8 op_c[16] );
void RijndaelKeySchedule( u8 key[16] );
void RijndaelEncrypt( u8 input[16], u8 output[16] );


/*-----------------------------------------------------------------
 *                            Algorithm f1
 *-----------------------------------------------------------------
 *
 *  Computes network authentication code MAC-A from key K, random
 *  challenge RAND, sequence number SQN and authentication management
 *  field AMF.
 *
 *-----------------------------------------------------------------*/

void f1    ( u8 k[16], u8 rand[16], u8 sqn[6], u8 amf[2],
             u8 mac_a[8] )
{
  u8 op_c[16];
  u8 temp[16];
  u8 in1[16];
  u8 out1[16];
  u8 rijndaelInput[16];
  u8 i;

  RijndaelKeySchedule( k );

  ComputeOPc( op_c );

  for (i=0; i<16; i++)
    rijndaelInput[i] = rand[i] ^ op_c[i];
  RijndaelEncrypt( rijndaelInput, temp );
```

```
  for (i=0; i<6; i++)
  {
    in1[i]    = sqn[i];
    in1[i+8]  = sqn[i];
  }
  for (i=0; i<2; i++)
  {
    in1[i+6]  = amf[i];
    in1[i+14] = amf[i];
  }

  /* XOR op_c and in1, rotate by r1=64, and XOR *
   * on the constant c1 (which is all zeroes)   */

  for (i=0; i<16; i++)
    rijndaelInput[(i+8) % 16] = in1[i] ^ op_c[i];

  /* XOR on the value temp computed before */

  for (i=0; i<16; i++)
    rijndaelInput[i] ^= temp[i];

  RijndaelEncrypt( rijndaelInput, out1 );
  for (i=0; i<16; i++)
    out1[i] ^= op_c[i];

  for (i=0; i<8; i++)
    mac_a[i] = out1[i];

  return;
} /* end of function f1 */



/*-----------------------------------------------------------------
 *                          Algorithms f2-f5                       *
 *-----------------------------------------------------------------
 *
 *  Takes key K and random challenge RAND, and returns response RES,
 *  confidentiality key CK, integrity key IK and anonymity key AK.
 *
 *-----------------------------------------------------------------*/

void f2345 ( u8 k[16], u8 rand[16],
             u8 res[8], u8 ck[16], u8 ik[16], u8 ak[6] )
{
  u8 op_c[16];
  u8 temp[16];
  u8 out[16];
  u8 rijndaelInput[16];
  u8 i;

  RijndaelKeySchedule( k );

  ComputeOPc( op_c );

  for (i=0; i<16; i++)
    rijndaelInput[i] = rand[i] ^ op_c[i];
  RijndaelEncrypt( rijndaelInput, temp );

  /* To obtain output block OUT2: XOR OPc and TEMP,    *
   * rotate by r2=0, and XOR on the constant c2 (which *
   * is all zeroes except that the last bit is 1).     */

  for (i=0; i<16; i++)
    rijndaelInput[i] = temp[i] ^ op_c[i];
  rijndaelInput[15] ^= 1;

  RijndaelEncrypt( rijndaelInput, out );
  for (i=0; i<16; i++)
    out[i] ^= op_c[i];

  for (i=0; i<8; i++)
    res[i] = out[i+8];
  for (i=0; i<6; i++)
    ak[i]  = out[i];

  /* To obtain output block OUT3: XOR OPc and TEMP,        *
```

```
   * rotate by r3=32, and XOR on the constant c3 (which    *
   * is all zeroes except that the next to last bit is 1). */

  for (i=0; i<16; i++)
    rijndaelInput[(i+12) % 16] = temp[i] ^ op_c[i];
  rijndaelInput[15] ^= 2;

  RijndaelEncrypt( rijndaelInput, out );
  for (i=0; i<16; i++)
    out[i] ^= op_c[i];

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

  /* To obtain output block OUT4: XOR OPc and TEMP,         *
   * rotate by r4=64, and XOR on the constant c4 (which     *
   * is all zeroes except that the 2nd from last bit is 1). */

  for (i=0; i<16; i++)
    rijndaelInput[(i+8) % 16] = temp[i] ^ op_c[i];
  rijndaelInput[15] ^= 4;

  RijndaelEncrypt( rijndaelInput, out );
  for (i=0; i<16; i++)
    out[i] ^= op_c[i];

  for (i=0; i<16; i++)
    ik[i] = out[i];

  return;
} /* end of function f2345 */


/*-----------------------------------------------------------------
 *                           Algorithm f1*
 *-----------------------------------------------------------------
 *
 *  Computes resynch authentication code MAC-S from key K, random
 *  challenge RAND, sequence number SQN and authentication management
 *  field AMF.
 *
 *-----------------------------------------------------------------*/

void f1star( u8 k[16], u8 rand[16], u8 sqn[6], u8 amf[2],
             u8 mac_s[8] )
{
  u8 op_c[16];
  u8 temp[16];
  u8 in1[16];
  u8 out1[16];
  u8 rijndaelInput[16];
  u8 i;

  RijndaelKeySchedule( k );

  ComputeOPc( op_c );

  for (i=0; i<16; i++)
    rijndaelInput[i] = rand[i] ^ op_c[i];
  RijndaelEncrypt( rijndaelInput, temp );

  for (i=0; i<6; i++)
  {
    in1[i]    = sqn[i];
    in1[i+8]  = sqn[i];
  }
  for (i=0; i<2; i++)
  {
    in1[i+6]  = amf[i];
    in1[i+14] = amf[i];
  }

  /* XOR op_c and in1, rotate by r1=64, and XOR *
   * on the constant c1 (which is all zeroes)   */

  for (i=0; i<16; i++)
    rijndaelInput[(i+8) % 16] = in1[i] ^ op_c[i];
```

```
  /* XOR on the value temp computed before */

  for (i=0; i<16; i++)
    rijndaelInput[i] ^= temp[i];

  RijndaelEncrypt( rijndaelInput, out1 );
  for (i=0; i<16; i++)
    out1[i] ^= op_c[i];

  for (i=0; i<8; i++)
    mac_s[i] = out1[i+8];

  return;
} /* end of function f1star */


/*-----------------------------------------------------------------
 *                           Algorithm f5*                         
 *-----------------------------------------------------------------
 *
 *  Takes key K and random challenge RAND, and returns resynch
 *  anonymity key AK.
 *
 *-----------------------------------------------------------------*/

void f5star( u8 k[16], u8 rand[16],
             u8 ak[6] )
{
  u8 op_c[16];
  u8 temp[16];
  u8 out[16];
  u8 rijndaelInput[16];
  u8 i;

  RijndaelKeySchedule( k );

  ComputeOPc( op_c );

  for (i=0; i<16; i++)
    rijndaelInput[i] = rand[i] ^ op_c[i];
  RijndaelEncrypt( rijndaelInput, temp );

  /* To obtain output block OUT5: XOR OPc and TEMP,         *
   * rotate by r5=96, and XOR on the constant c5 (which     *
   * is all zeroes except that the 3rd from last bit is 1). */

  for (i=0; i<16; i++)
    rijndaelInput[(i+4) % 16] = temp[i] ^ op_c[i];
  rijndaelInput[15] ^= 8;

  RijndaelEncrypt( rijndaelInput, out );
  for (i=0; i<16; i++)
    out[i] ^= op_c[i];

  for (i=0; i<6; i++)
    ak[i] = out[i];

  return;
} /* end of function f5star */


/*-----------------------------------------------------------------
 *  Function to compute OPc from OP and K.  Assumes key schedule has
    already been performed.
 *-----------------------------------------------------------------*/

void ComputeOPc( u8 op_c[16] )
{
  u8 i;

  RijndaelEncrypt( OP, op_c );
  for (i=0; i<16; i++)
    op_c[i] ^= OP[i];

  return;
} /* end of function ComputeOPc */
```

```
/*-------------------- Rijndael round subkeys ---------------------*/
u8 roundKeys[11][4][4];

/*---------------------- Rijndael S box table ----------------------*/
u8 S[256] = {
 99,124,119,123,242,107,111,197, 48,  1,103, 43,254,215,171,118,
202,130,201,125,250, 89, 71,240,173,212,162,175,156,164,114,192,
183,253,147, 38, 54, 63,247,204, 52,165,229,241,113,216, 49, 21,
  4,199, 35,195, 24,150,  5,154,  7, 18,128,226,235, 39,178,117,
  9,131, 44, 26, 27,110, 90,160, 82, 59,214,179, 41,227, 47,132,
 83,209,  0,237, 32,252,177, 91,106,203,190, 57, 74, 76, 88,207,
208,239,170,251, 67, 77, 51,133, 69,249,  2,127, 80, 60,159,168,
 81,163, 64,143,146,157, 56,245,188,182,218, 33, 16,255,243,210,
205, 12, 19,236, 95,151, 68, 23,196,167,126, 61,100, 93, 25,115,
 96,129, 79,220, 34, 42,144,136, 70,238,184, 20,222, 94, 11,219,
224, 50, 58, 10, 73,  6, 36, 92,194,211,172, 98,145,149,228,121,
231,200, 55,109,141,213, 78,169,108, 86,244,234,101,122,174,  8,
186,120, 37, 46, 28,166,180,198,232,221,116, 31, 75,189,139,138,
112, 62,181,102, 72,  3,246, 14, 97, 53, 87,185,134,193, 29,158,
225,248,152, 17,105,217,142,148,155, 30,135,233,206, 85, 40,223,
140,161,137, 13,191,230, 66,104, 65,153, 45, 15,176, 84,187, 22,
};

/*------- This array does the multiplication by x in GF(2^8) ------*/
u8 Xtime[256] = {
  0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30,
 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62,
 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94,
 96, 98,100,102,104,106,108,110,112,114,116,118,120,122,124,126,
128,130,132,134,136,138,140,142,144,146,148,150,152,154,156,158,
160,162,164,166,168,170,172,174,176,178,180,182,184,186,188,190,
192,194,196,198,200,202,204,206,208,210,212,214,216,218,220,222,
224,226,228,230,232,234,236,238,240,242,244,246,248,250,252,254,
 27, 25, 31, 29, 19, 17, 23, 21, 11,  9, 15, 13,  3,  1,  7,  5,
 59, 57, 63, 61, 51, 49, 55, 53, 43, 41, 47, 45, 35, 33, 39, 37,
 91, 89, 95, 93, 83, 81, 87, 85, 75, 73, 79, 77, 67, 65, 71, 69,
123,121,127,125,115,113,119,117,107,105,111,109, 99, 97,103,101,
155,153,159,157,147,145,151,149,139,137,143,141,131,129,135,133,
187,185,191,189,179,177,183,181,171,169,175,173,163,161,167,165,
219,217,223,221,211,209,215,213,203,201,207,205,195,193,199,197,
251,249,255,253,243,241,247,245,235,233,239,237,227,225,231,229
};


/*---------------------------------------------------------------
 *  Rijndael key schedule function.  Takes 16-byte key and creates
 *  all Rijndael's internal subkeys ready for encryption.
 *---------------------------------------------------------------*/

void RijndaelKeySchedule( u8 key[16] )
{
  u8 roundConst;
  int i, j;

  /* first round key equals key */
  for (i=0; i<16; i++)
    roundKeys[0][i & 0x03][i>>2] = key[i];

  roundConst = 1;

  /* now calculate round keys */
  for (i=1; i<11; i++)
  {
    roundKeys[i][0][0] = S[roundKeys[i-1][1][3]]
                         ^ roundKeys[i-1][0][0] ^ roundConst;
    roundKeys[i][1][0] = S[roundKeys[i-1][2][3]]
                         ^ roundKeys[i-1][1][0];
    roundKeys[i][2][0] = S[roundKeys[i-1][3][3]]
                         ^ roundKeys[i-1][2][0];
    roundKeys[i][3][0] = S[roundKeys[i-1][0][3]]
                         ^ roundKeys[i-1][3][0];

    for (j=0; j<4; j++)
    {
      roundKeys[i][j][1] = roundKeys[i-1][j][1] ^ roundKeys[i][j][0];
      roundKeys[i][j][2] = roundKeys[i-1][j][2] ^ roundKeys[i][j][1];
      roundKeys[i][j][3] = roundKeys[i-1][j][3] ^ roundKeys[i][j][2];
```

```
    }

    /* update round constant */
    roundConst = Xtime[roundConst];
  }

  return;
} /* end of function RijndaelKeySchedule */


/* Round key addition function */
void KeyAdd(u8 state[4][4], u8 roundKeys[11][4][4], int round)
{
  int i, j;

  for (i=0; i<4; i++)
    for (j=0; j<4; j++)
      state[i][j] ^= roundKeys[round][i][j];

  return;
}


/* Byte substitution transformation */
int ByteSub(u8 state[4][4])
{
  int i, j;

  for (i=0; i<4; i++)
    for (j=0; j<4; j++)
      state[i][j] = S[state[i][j]];

  return 0;
}


/* Row shift transformation */
void ShiftRow(u8 state[4][4])
{
  u8 temp;

  /* left rotate row 1 by 1 */
  temp = state[1][0];
  state[1][0] = state[1][1];
  state[1][1] = state[1][2];
  state[1][2] = state[1][3];
  state[1][3] = temp;

  /* left rotate row 2 by 2 */
  temp = state[2][0];
  state[2][0] = state[2][2];
  state[2][2] = temp;
  temp = state[2][1];
  state[2][1] = state[2][3];
  state[2][3] = temp;

  /* left rotate row 3 by 3 */
  temp = state[3][0];
  state[3][0] = state[3][3];
  state[3][3] = state[3][2];
  state[3][2] = state[3][1];
  state[3][1] = temp;

  return;
}


/* MixColumn transformation*/
void MixColumn(u8 state[4][4])
{
  u8 temp, tmp, tmp0;
  int i;

  /* do one column at a time */
  for (i=0; i<4;i++)
  {
    temp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i];
    tmp0 = state[0][i];
```

```
    /* Xtime array does multiply by x in GF2^8 */
    tmp = Xtime[state[0][i] ^ state[1][i]];
    state[0][i] ^= temp ^ tmp;

    tmp = Xtime[state[1][i] ^ state[2][i]];
    state[1][i] ^= temp ^ tmp;

    tmp = Xtime[state[2][i] ^ state[3][i]];
    state[2][i] ^= temp ^ tmp;

    tmp = Xtime[state[3][i] ^ tmp0];
    state[3][i] ^= temp ^ tmp;
  }

  return;
}


/*-------------------------------------------------------------------
 *  Rijndael encryption function.  Takes 16-byte input and creates
 *  16-byte output (using round keys already derived from 16-byte
 *  key).
 *-------------------------------------------------------------------*/

void RijndaelEncrypt( u8 input[16], u8 output[16] )
{
  u8 state[4][4];
  int i, r;

  /* initialise state array from input byte string */
  for (i=0; i<16; i++)
    state[i & 0x3][i>>2] = input[i];

  /* add first round_key */
  KeyAdd(state, roundKeys, 0);

  /* do lots of full rounds */
  for (r=1; r<=9; r++)
  {
    ByteSub(state);
    ShiftRow(state);
    MixColumn(state);
    KeyAdd(state, roundKeys, r);
  }

  /* final round */
  ByteSub(state);
  ShiftRow(state);
  KeyAdd(state, roundKeys, r);

  /* produce output byte string from state array */
  for (i=0; i<16; i++)
  {
    output[i] = state[i & 0x3][i>>2];
  }

  return;
} /* end of function RijndaelEncrypt */
```

Annex 4:
Rijndael Listing - 32-Bit Word Oriented

```
/*-----------------------------------------------------------------
 *                     Rijndael Implementation
 *-----------------------------------------------------------------
 *
 *  A sample 32-bit orientated implementation of Rijndael, the
 *  suggested kernel for the example 3GPP authentication and key
 *  agreement functions.
 *
 *  This implementation draws on the description in section 5.2 of
 *  the AES proposal and also on the implementation by
 *  Dr B. R. Gladman <brg@gladman.uk.net> 9th October 2000.
 *  It uses a number of large (4k) lookup tables to implement the
 *  algorithm in an efficient manner.
 *
 *  Note: in this implementation the State is stored in four 32-bit
 *  words, one per column of the State, with the top byte of the
 *  column being the _least_ significant byte of the word.
 *
*-----------------------------------------------------------------*/

#define LITTLE_ENDIAN   /* For INTEL architecture */

typedef unsigned char   u8;
typedef unsigned int     u32;

/* Circular byte rotates of 32 bit values */

#define rot1(x) ((x <<  8) | (x >> 24))
#define rot2(x) ((x << 16) | (x >> 16))
#define rot3(x) ((x << 24) | (x >>  8))

/* Extract a byte from a 32-bit u32 */

#define byte0(x)    ((u8)(x))
#define byte1(x)    ((u8)(x >>  8))
#define byte2(x)    ((u8)(x >> 16))
#define byte3(x)    ((u8)(x >> 24))


/* Put or get a 32 bit u32 (v) in machine order from a byte *
 * address in (x)                                           */

#ifdef  LITTLE_ENDIAN

#define u32_in(x)     (*(u32*)(x))
#define u32_out(x,y)  (*(u32*)(x) = y)

#else

/* Invert byte order in a 32 bit variable */

__inline u32 byte_swap(const u32 x)
{
    return rot1(x) & 0x00ff00ff | rot3(x) & 0xff00ff00;
}
__inline u32 u32_in(const u8 x[])
{
  return byte_swap(*(u32*)x);
};
__inline void u32_out(u8 x[], const u32 v)
{
  *(u32*)x = byte_swap(v);
};

#endif

/*--------------- The lookup tables ----------------------------*/

static u32 rnd_con[10] =
{
 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36
```

```
};

static u32 ft_tab[4][256] =
{
 {
 0xA56363C6,0x847C7CF8,0x997777EE,0x8D7B7BF6,0x0DF2F2FF,0xBD6B6BD6,0xB16F6FDE,0x54C5C591,
 0x50303060,0x03010102,0xA96767CE,0x7D2B2B56,0x19FEFEE7,0x62D7D7B5,0xE6ABAB4D,0x9A7676EC,
 0x45CACA8F,0x9D82821F,0x40C9C989,0x877D7DFA,0x15FAFAEF,0xEB5959B2,0xC947478E,0x0BF0F0FB,
 0xECADAD41,0x67D4D4B3,0xFDA2A25F,0xEAAFAF45,0xBF9C9C23,0xF7A4A453,0x967272E4,0x5BC0C09B,
 0xC2B7B775,0x1CFDFDE1,0xAE93933D,0x6A26264C,0x5A36366C,0x413F3F7E,0x02F7F7F5,0x4FCCCC83,
 0x5C343468,0xF4A5A551,0x34E5E5D1,0x08F1F1F9,0x937171E2,0x73D8D8AB,0x53313162,0x3F15152A,
 0x0C040408,0x52C7C795,0x65232346,0x5EC3C39D,0x28181830,0xA1969637,0x0F05050A,0xB59A9A2F,
 0x0907070E,0x36121224,0x9B80801B,0x3DE2E2DF,0x26EBEBCD,0x6927274E,0xCDB2B27F,0x9F7575EA,
 0x1B090912,0x9E83831D,0x742C2C58,0x2E1A1A34,0x2D1B1B36,0xB26E6EDC,0xEE5A5AB4,0xFBA0A05B,
 0xF65252A4,0x4D3B3B76,0x61D6D6B7,0xCEB3B37D,0x7B292952,0x3EE3E3DD,0x712F2F5E,0x97848413,
 0xF55353A6,0x68D1D1B9,0000000000,0x2CEDEDC1,0x60202040,0x1FFCFCE3,0xC8B1B179,0xED5B5BB6,
 0xBE6A6AD4,0x46CBCB8D,0xD9BEBE67,0x4B393972,0xDE4A4A94,0xD44C4C98,0xE85858B0,0x4ACFCF85,
 0x6BD0D0BB,0x2AEFEFC5,0xE5AAAA4F,0x16FBFBED,0xC5434386,0xD74D4D9A,0x55333366,0x94858511,
 0xCF45458A,0x10F9F9E9,0x06020204,0x817F7FFE,0xF05050A0,0x443C3C78,0xBA9F9F25,0xE3A8A84B,
 0xF35151A2,0xFEA3A35D,0xC0404080,0x8A8F8F05,0xAD92923F,0xBC9D9D21,0x48383870,0x04F5F5F1,
 0xDFBCBC63,0xC1B6B677,0x75DADAAF,0x63212142,0x30101020,0x1AFFFFE5,0x0EF3F3FD,0x6DD2D2BF,
 0x4CCDCD81,0x140C0C18,0x35131326,0x2FECECC3,0xE15F5FBE,0xA2979735,0xCC444488,0x3917172E,
 0x57C4C493,0xF2A7A755,0x827E7EFC,0x473D3D7A,0xAC6464C8,0xE75D5DBA,0x2B191932,0x957373E6,
 0xA06060C0,0x98818119,0xD14F4F9E,0x7FDCDCA3,0x66222244,0x7E2A2A54,0xAB90903B,0x8388880B,
 0xCA46468C,0x29EEEEC7,0xD3B8B86B,0x3C141428,0x79DEDEA7,0xE25E5EBC,0x1D0B0B16,0x76DBDBAD,
 0x3BE0E0DB,0x56323264,0x4E3A3A74,0x1E0A0A14,0xDB494992,0x0A06060C,0x6C242448,0xE45C5CB8,
 0x5DC2C29F,0x6ED3D3BD,0xEFACAC43,0xA66262C4,0xA8919139,0xA4959531,0x37E4E4D3,0x8B7979F2,
 0x32E7E7D5,0x43C8C88B,0x5937376E,0xB76D6DDA,0x8C8D8D01,0x64D5D5B1,0xD24E4E9C,0xE0A9A949,
 0xB46C6CD8,0xFA5656AC,0x07F4F4F3,0x25EAEACF,0xAF6565CA,0x8E7A7AF4,0xE9AEAE47,0x18080810,
 0xD5BABA6F,0x887878F0,0x6F25254A,0x722E2E5C,0x241C1C38,0xF1A6A657,0xC7B4B473,0x51C6C697,
 0x23E8E8CB,0x7CDDDDA1,0x9C7474E8,0x211F1F3E,0xDD4B4B96,0xDCBDBD61,0x868B8B0D,0x858A8A0F,
 0x907070E0,0x423E3E7C,0xC4B5B571,0xAA6666CC,0xD8484890,0x05030306,0x01F6F6F7,0x120E0E1C,
 0xA36161C2,0x5F35356A,0xF95757AE,0xD0B9B969,0x91868617,0x58C1C199,0x271D1D3A,0xB99E9E27,
 0x38E1E1D9,0x13F8F8EB,0xB398982B,0x33111122,0xBB6969D2,0x70D9D9A9,0x898E8E07,0xA7949433,
 0xB69B9B2D,0x221E1E3C,0x92878715,0x20E9E9C9,0x49CECE87,0xFF5555AA,0x78282850,0x7ADFDFA5,
 0x8F8C8C03,0xF8A1A159,0x80898909,0x170D0D1A,0xDABFBF65,0x31E6E6D7,0xC6424284,0xB86868D0,
 0xC3414182,0xB0999929,0x772D2D5A,0x110F0F1E,0xCBB0B07B,0xFC5454A8,0xD6BBBB6D,0x3A16162C
 },
 {
 0x6363C6A5,0x7C7CF884,0x7777EE99,0x7B7BF68D,0xF2F2FF0D,0x6B6BD6BD,0x6F6FDEB1,0xC5C59154,
 0x30306050,0x01010203,0x6767CEA9,0x2B2B567D,0xFEFEE719,0xD7D7B562,0xABAB4DE6,0x7676EC9A,
 0xCACA8F45,0x82821F9D,0xC9C98940,0x7D7DFA87,0xFAFAEF15,0x5959B2EB,0x47478EC9,0xF0F0FB0B,
 0xADAD41EC,0xD4D4B367,0xA2A25FFD,0xAFAF45EA,0x9C9C23BF,0xA4A453F7,0x7272E496,0xC0C09B5B,
 0xB7B775C2,0xFDFDE11C,0x93933DAE,0x26264C6A,0x36366C5A,0x3F3F7E41,0xF7F7F502,0xCCCC834F,
 0x3434685C,0xA5A551F4,0xE5E5D134,0xF1F1F908,0x7171E293,0xD8D8AB73,0x31316253,0x15152A3F,
 0x0404080C,0xC7C79552,0x23234665,0xC3C39D5E,0x18183028,0x969637A1,0x05050A0F,0x9A9A2FB5,
 0x07070E09,0x12122436,0x80801B9B,0xE2E2DF3D,0xEBEBCD26,0x27274E69,0xB2B27FCD,0x7575EA9F,
 0x0909121B,0x83831D9E,0x2C2C5874,0x1A1A342E,0x1B1B362D,0x6E6EDCB2,0x5A5AB4EE,0xA0A05BFB,
 0x5252A4F6,0x3B3B764D,0xD6D6B761,0xB3B37DCE,0x2929527B,0xE3E3DD3E,0x2F2F5E71,0x84841397,
 0x5353A6F5,0xD1D1B968,0000000000,0xEDEDC12C,0x20204060,0xFCFCE31F,0xB1B179C8,0x5B5BB6ED,
 0x6A6AD4BE,0xCBCB8D46,0xBEBE67D9,0x3939724B,0x4A4A94DE,0x4C4C98D4,0x5858B0E8,0xCFCF854A,
 0xD0D0BB6B,0xEFEFC52A,0xAAAA4FE5,0xFBFBED16,0x434386C5,0x4D4D9AD7,0x33336655,0x85851194,
 0x45458ACF,0xF9F9E910,0x02020406,0x7F7FFE81,0x5050A0F0,0x3C3C7844,0x9F9F25BA,0xA8A84BE3,
 0x5151A2F3,0xA3A35DFE,0x404080C0,0x8F8F058A,0x92923FAD,0x9D9D21BC,0x38387048,0xF5F5F104,
 0xBCBC63DF,0xB6B677C1,0xDADAAF75,0x21214263,0x10102030,0xFFFFE51A,0xF3F3FD0E,0xD2D2BF6D,
 0xCDCD814C,0x0C0C1814,0x13132635,0xECECC32F,0x5F5FBEE1,0x979735A2,0x444488CC,0x17172E39,
 0xC4C49357,0xA7A755F2,0x7E7EFC82,0x3D3D7A47,0x6464C8AC,0x5D5DBAE7,0x1919322B,0x7373E695,
 0x6060C0A0,0x81811998,0x4F4F9ED1,0xDCDCA37F,0x22224466,0x2A2A547E,0x90903BAB,0x88880B83,
 0x46468CCA,0xEEEEC729,0xB8B86BD3,0x1414283C,0xDEDEA779,0x5E5EBCE2,0x0B0B161D,0xDBDBAD76,
 0xE0E0DB3B,0x32326456,0x3A3A744E,0x0A0A141E,0x494992DB,0x06060C0A,0x2424486C,0x5C5CB8E4,
 0xC2C29F5D,0xD3D3BD6E,0xACAC43EF,0x6262C4A6,0x919139A8,0x959531A4,0xE4E4D337,0x7979F28B,
 0xE7E7D532,0xC8C88B43,0x37376E59,0x6D6DDAB7,0x8D8D018C,0xD5D5B164,0x4E4E9CD2,0xA9A949E0,
 0x6C6CD8B4,0x5656ACFA,0xF4F4F307,0xEAEACF25,0x6565CAAF,0x7A7AF48E,0xAEAE47E9,0x08081018,
 0xBABA6FD5,0x7878F088,0x25254A6F,0x2E2E5C72,0x1C1C3824,0xA6A657F1,0xB4B473C7,0xC6C69751,
 0xE8E8CB23,0xDDDDA17C,0x7474E89C,0x1F1F3E21,0x4B4B96DD,0xBDBD61DC,0x8B8B0D86,0x8A8A0F85,
 0x7070E090,0x3E3E7C42,0xB5B571C4,0x6666CCAA,0x484890D8,0x03030605,0xF6F6F701,0x0E0E1C12,
 0x6161C2A3,0x35356A5F,0x5757AEF9,0xB9B969D0,0x86861791,0xC1C19958,0x1D1D3A27,0x9E9E27B9,
 0xE1E1D938,0xF8F8EB13,0x98982BB3,0x11112233,0x6969D2BB,0xD9D9A970,0x8E8E0789,0x949433A7,
 0x9B9B2DB6,0x1E1E3C22,0x87871592,0xE9E9C920,0xCECE8749,0x5555AAFF,0x28285078,0xDFDFA57A,
 0x8C8C038F,0xA1A159F8,0x89890980,0x0D0D1A17,0xBFBF65DA,0xE6E6D731,0x424284C6,0x6868D0B8,
 0x414182C3,0x999929B0,0x2D2D5A77,0x0F0F1E11,0xB0B07BCB,0x5454A8FC,0xBBBB6DD6,0x16162C3A
 },
 {
 0x63C6A563,0x7CF8847C,0x77EE9977,0x7BF68D7B,0xF2FF0DF2,0x6BD6BD6B,0x6FDEB16F,0xC59154C5,
 0x30605030,0x01020301,0x67CEA967,0x2B567D2B,0xFEE719FE,0xD7B562D7,0xAB4DE6AB,0x76EC9A76,
 0xCA8F45CA,0x821F9D82,0xC98940C9,0x7DFA877D,0xFAEF15FA,0x59B2EB59,0x478EC947,0xF0FB0BF0,
 0xAD41ECAD,0xD4B367D4,0xA25FFDA2,0xAF45EAAF,0x9C23BF9C,0xA453F7A4,0x72E49672,0xC09B5BC0,
 0xB775C2B7,0xFDE11CFD,0x933DAE93,0x264C6A26,0x366C5A36,0x3F7E413F,0xF7F502F7,0xCC834FCC,
```

```
 0x34685C34,0xA551F4A5,0xE5D134E5,0xF1F908F1,0x71E29371,0xD8AB73D8,0x31625331,0x152A3F15,
 0x04080C04,0xC79552C7,0x23466523,0xC39D5EC3,0x18302818,0x9637A196,0x050A0F05,0x9A2FB59A,
 0x070E0907,0x12243612,0x801B9B80,0xE2DF3DE2,0xEBCD26EB,0x274E6927,0xB27FCDB2,0x75EA9F75,
 0x09121B09,0x831D9E83,0x2C58742C,0x1A342E1A,0x1B362D1B,0x6EDCB26E,0x5AB4EE5A,0xA05BFBA0,
 0x52A4F652,0x3B764D3B,0xD6B761D6,0xB37DCEB3,0x29527B29,0xE3DD3EE3,0x2F5E712F,0x84139784,
 0x53A6F553,0xD1B968D1,0000000000,0xEDC12CED,0x20406020,0xFCE31FFC,0xB179C8B1,0x5BB6ED5B,
 0x6AD4BE6A,0xCB8D46CB,0xBE67D9BE,0x39724B39,0x4A94DE4A,0x4C98D44C,0x58B0E858,0xCF854ACF,
 0xD0BB6BD0,0xEFC52AEF,0xAA4FE5AA,0xFBED16FB,0x4386C543,0x4D9AD74D,0x33665533,0x85119485,
 0x458ACF45,0xF9E910F9,0x02040602,0x7FFE817F,0x50A0F050,0x3C78443C,0x9F25BA9F,0xA84BE3A8,
 0x51A2F351,0xA35DFEA3,0x4080C040,0x8F058A8F,0x923FAD92,0x9D21BC9D,0x38704838,0xF5F104F5,
 0xBC63DFBC,0xB677C1B6,0xDAAF75DA,0x21426321,0x10203010,0xFFE51AFF,0xF3FD0EF3,0xD2BF6DD2,
 0xCD814CCD,0x0C18140C,0x13263513,0xECC32FEC,0x5FBEE15F,0x9735A297,0x4488CC44,0x172E3917,
 0xC49357C4,0xA755F2A7,0x7EFC827E,0x3D7A473D,0x64C8AC64,0x5DBAE75D,0x19322B19,0x73E69573,
 0x60C0A060,0x81199881,0x4F9ED14F,0xDCA37FDC,0x22446622,0x2A547E2A,0x903BAB90,0x880B8388,
 0x468CCA46,0xEEC729EE,0xB86BD3B8,0x14283C14,0xDEA779DE,0x5EBCE25E,0x0B161D0B,0xDBAD76DB,
 0xE0DB3BE0,0x32645632,0x3A744E3A,0x0A141E0A,0x4992DB49,0x060C0A06,0x24486C24,0x5CB8E45C,
 0xC29F5DC2,0xD3BD6ED3,0xAC43EFAC,0x62C4A662,0x9139A891,0x9531A495,0xE4D337E4,0x79F28B79,
 0xE7D532E7,0xC88B43C8,0x376E5937,0x6DDAB76D,0x8D018C8D,0xD5B164D5,0x4E9CD24E,0xA949E0A9,
 0x6CD8B46C,0x56ACFA56,0xF4F307F4,0xEACF25EA,0x65CAAF65,0x7AF48E7A,0xAE47E9AE,0x08101808,
 0xBA6FD5BA,0x78F08878,0x254A6F25,0x2E5C722E,0x1C38241C,0xA657F1A6,0xB473C7B4,0xC69751C6,
 0xE8CB23E8,0xDDA17CDD,0x74E89C74,0x1F3E211F,0x4B96DD4B,0xBD61DCBD,0x8B0D868B,0x8A0F858A,
 0x70E09070,0x3E7C423E,0xB571C4B5,0x66CCAA66,0x4890D848,0x03060503,0xF6F701F6,0x0E1C120E,
 0x61C2A361,0x356A5F35,0x57AEF957,0xB969D0B9,0x86179186,0xC19958C1,0x1D3A271D,0x9E27B99E,
 0xE1D938E1,0xF8EB13F8,0x982BB398,0x11223311,0x69D2BB69,0xD9A970D9,0x8E07898E,0x9433A794,
 0x9B2DB69B,0x1E3C221E,0x87159287,0xE9C920E9,0xCE8749CE,0x55AAFF55,0x28507828,0xDFA57ADF,
 0x8C038F8C,0xA159F8A1,0x89098089,0x0D1A170D,0xBF65DABF,0xE6D731E6,0x4284C642,0x68D0B868,
 0x4182C341,0x9929B099,0x2D5A772D,0x0F1E110F,0xB07BCBB0,0x54A8FC54,0xBB6DD6BB,0x162C3A16
 },
 {
 0xC6A56363,0xF8847C7C,0xEE997777,0xF68D7B7B,0xFF0DF2F2,0xD6BD6B6B,0xDEB16F6F,0x9154C5C5,
 0x60503030,0x02030101,0xCEA96767,0x567D2B2B,0xE719FEFE,0xB562D7D7,0x4DE6ABAB,0xEC9A7676,
 0x8F45CACA,0x1F9D8282,0x8940C9C9,0xFA877D7D,0xEF15FAFA,0xB2EB5959,0x8EC94747,0xFB0BF0F0,
 0x41ECADAD,0xB367D4D4,0x5FFDA2A2,0x45EAAFAF,0x23BF9C9C,0x53F7A4A4,0xE4967272,0x9B5BC0C0,
 0x75C2B7B7,0xE11CFDFD,0x3DAE9393,0x4C6A2626,0x6C5A3636,0x7E413F3F,0xF502F7F7,0x834FCCCC,
 0x685C3434,0x51F4A5A5,0xD134E5E5,0xF908F1F1,0xE2937171,0xAB73D8D8,0x62533131,0x2A3F1515,
 0x080C0404,0x9552C7C7,0x46652323,0x9D5EC3C3,0x30281818,0x37A19696,0x0A0F0505,0x2FB59A9A,
 0x0E090707,0x24361212,0x1B9B8080,0xDF3DE2E2,0xCD26EBEB,0x4E692727,0x7FCDB2B2,0xEA9F7575,
 0x121B0909,0x1D9E8383,0x58742C2C,0x342E1A1A,0x362D1B1B,0xDCB26E6E,0xB4EE5A5A,0x5BFBA0A0,
 0xA4F65252,0x764D3B3B,0xB761D6D6,0x7DCEB3B3,0x527B2929,0xDD3EE3E3,0x5E712F2F,0x13978484,
 0xA6F55353,0xB968D1D1,0000000000,0xC12CEDED,0x40602020,0xE31FFCFC,0x79C8B1B1,0xB6ED5B5B,
 0xD4BE6A6A,0x8D46CBCB,0x67D9BEBE,0x724B3939,0x94DE4A4A,0x98D44C4C,0xB0E85858,0x854ACFCF,
 0xBB6BD0D0,0xC52AEFEF,0x4FE5AAAA,0xED16FBFB,0x86C54343,0x9AD74D4D,0x66553333,0x11948585,
 0x8ACF4545,0xE910F9F9,0x04060202,0xFE817F7F,0xA0F05050,0x78443C3C,0x25BA9F9F,0x4BE3A8A8,
 0xA2F35151,0x5DFEA3A3,0x80C04040,0x058A8F8F,0x3FAD9292,0x21BC9D9D,0x70483838,0xF104F5F5,
 0x63DFBCBC,0x77C1B6B6,0xAF75DADA,0x42632121,0x20301010,0xE51AFFFF,0xFD0EF3F3,0xBF6DD2D2,
 0x814CCDCD,0x18140C0C,0x26351313,0xC32FECEC,0xBEE15F5F,0x35A29797,0x88CC4444,0x2E391717,
 0x9357C4C4,0x55F2A7A7,0xFC827E7E,0x7A473D3D,0xC8AC6464,0xBAE75D5D,0x322B1919,0xE6957373,
 0xC0A06060,0x19988181,0x9ED14F4F,0xA37FDCDC,0x44662222,0x547E2A2A,0x3BAB9090,0x0B838888,
 0x8CCA4646,0xC729EEEE,0x6BD3B8B8,0x283C1414,0xA779DEDE,0xBCE25E5E,0x161D0B0B,0xAD76DBDB,
 0xDB3BE0E0,0x64563232,0x744E3A3A,0x141E0A0A,0x92DB4949,0x0C0A0606,0x486C2424,0xB8E45C5C,
 0x9F5DC2C2,0xBD6ED3D3,0x43EFACAC,0xC4A66262,0x39A89191,0x31A49595,0xD337E4E4,0xF28B7979,
 0xD532E7E7,0x8B43C8C8,0x6E593737,0xDAB76D6D,0x018C8D8D,0xB164D5D5,0x9CD24E4E,0x49E0A9A9,
 0xD8B46C6C,0xACFA5656,0xF307F4F4,0xCF25EAEA,0xCAAF6565,0xF48E7A7A,0x47E9AEAE,0x10180808,
 0x6FD5BABA,0xF0887878,0x4A6F2525,0x5C722E2E,0x38241C1C,0x57F1A6A6,0x73C7B4B4,0x9751C6C6,
 0xCB23E8E8,0xA17CDDDD,0xE89C7474,0x3E211F1F,0x96DD4B4B,0x61DCBDBD,0x0D868B8B,0x0F858A8A,
 0xE0907070,0x7C423E3E,0x71C4B5B5,0xCCAA6666,0x90D84848,0x06050303,0xF701F6F6,0x1C120E0E,
 0xC2A36161,0x6A5F3535,0xAEF95757,0x69D0B9B9,0x17918686,0x9958C1C1,0x3A271D1D,0x27B99E9E,
 0xD938E1E1,0xEB13F8F8,0x2BB39898,0x22331111,0xD2BB6969,0xA970D9D9,0x07898E8E,0x33A79494,
 0x2DB69B9B,0x3C221E1E,0x15928787,0xC920E9E9,0x8749CECE,0xAAFF5555,0x50782828,0xA57ADFDF,
 0x038F8C8C,0x59F8A1A1,0x09808989,0x1A170D0D,0x65DABFBF,0xD731E6E6,0x84C64242,0xD0B86868,
 0x82C34141,0x29B09999,0x5A772D2D,0x1E110F0F,0x7BCBB0B0,0xA8FC5454,0x6DD6BBBB,0x2C3A1616
 }
};

static u32 fl_tab[4][256] =
{
 {
 0x00000063,0x0000007C,0x00000077,0x0000007B,0x000000F2,0x0000006B,0x0000006F,0x000000C5,
 0x00000030,0x00000001,0x00000067,0x0000002B,0x000000FE,0x000000D7,0x000000AB,0x00000076,
 0x000000CA,0x00000082,0x000000C9,0x0000007D,0x000000FA,0x00000059,0x00000047,0x000000F0,
 0x000000AD,0x000000D4,0x000000A2,0x000000AF,0x0000009C,0x000000A4,0x00000072,0x000000C0,
 0x000000B7,0x000000FD,0x00000093,0x00000026,0x00000036,0x0000003F,0x000000F7,0x000000CC,
 0x00000034,0x000000A5,0x000000E5,0x000000F1,0x00000071,0x000000D8,0x00000031,0x00000015,
 0x00000004,0x000000C7,0x00000023,0x000000C3,0x00000018,0x00000096,0x00000005,0x0000009A,
 0x00000007,0x00000012,0x00000080,0x000000E2,0x000000EB,0x00000027,0x000000B2,0x00000075,
 0x00000009,0x00000083,0x0000002C,0x0000001A,0x0000001B,0x0000006E,0x0000005A,0x000000A0,
 0x00000052,0x0000003B,0x000000D6,0x000000B3,0x00000029,0x000000E3,0x0000002F,0x00000084,
 0x00000053,0x000000D1,0x00000000,0x000000ED,0x00000020,0x000000FC,0x000000B1,0x0000005B,
```

```
0x0000006A,0x000000CB,0x000000BE,0x00000039,0x0000004A,0x0000004C,0x00000058,0x000000CF,
0x000000D0,0x000000EF,0x000000AA,0x000000FB,0x00000043,0x0000004D,0x00000033,0x00000085,
0x00000045,0x000000F9,0x00000002,0x0000007F,0x00000050,0x0000003C,0x0000009F,0x000000A8,
0x00000051,0x000000A3,0x00000040,0x0000008F,0x00000092,0x0000009D,0x00000038,0x000000F5,
0x000000BC,0x000000B6,0x000000DA,0x00000021,0x00000010,0x000000FF,0x000000F3,0x000000D2,
0x000000CD,0x0000000C,0x00000013,0x000000EC,0x0000005F,0x00000097,0x00000044,0x00000017,
0x000000C4,0x000000A7,0x0000007E,0x0000003D,0x00000064,0x0000005D,0x00000019,0x00000073,
0x00000060,0x00000081,0x0000004F,0x000000DC,0x00000022,0x0000002A,0x00000090,0x00000088,
0x00000046,0x000000EE,0x000000B8,0x00000014,0x000000DE,0x0000005E,0x0000000B,0x000000DB,
0x000000E0,0x00000032,0x0000003A,0x0000000A,0x00000049,0x00000006,0x00000024,0x0000005C,
0x000000C2,0x000000D3,0x000000AC,0x00000062,0x00000091,0x00000095,0x000000E4,0x00000079,
0x000000E7,0x000000C8,0x00000037,0x0000006D,0x0000008D,0x000000D5,0x0000004E,0x000000A9,
0x0000006C,0x00000056,0x000000F4,0x000000EA,0x00000065,0x0000007A,0x000000AE,0x00000008,
0x000000BA,0x00000078,0x00000025,0x0000002E,0x0000001C,0x000000A6,0x000000B4,0x000000C6,
0x000000E8,0x000000DD,0x00000074,0x0000001F,0x0000004B,0x000000BD,0x0000008B,0x0000008A,
0x00000070,0x0000003E,0x000000B5,0x00000066,0x00000048,0x00000003,0x000000F6,0x0000000E,
0x00000061,0x00000035,0x00000057,0x000000B9,0x00000086,0x000000C1,0x0000001D,0x0000009E,
0x000000E1,0x000000F8,0x00000098,0x00000011,0x00000069,0x000000D9,0x0000008E,0x00000094,
0x0000009B,0x0000001E,0x00000087,0x000000E9,0x000000CE,0x00000055,0x00000028,0x000000DF,
0x0000008C,0x000000A1,0x00000089,0x0000000D,0x000000BF,0x000000E6,0x00000042,0x00000068,
0x00000041,0x00000099,0x0000002D,0x0000000F,0x000000B0,0x00000054,0x000000BB,0x00000016
},
{
0x00006300,0x00007C00,0x00007700,0x00007B00,0x0000F200,0x00006B00,0x00006F00,0x0000C500,
0x00003000,0x00000100,0x00006700,0x00002B00,0x0000FE00,0x0000D700,0x0000AB00,0x00007600,
0x0000CA00,0x00008200,0x0000C900,0x00007D00,0x0000FA00,0x00005900,0x00004700,0x0000F000,
0x0000AD00,0x0000D400,0x0000A200,0x0000AF00,0x00009C00,0x0000A400,0x00007200,0x0000C000,
0x0000B700,0x0000FD00,0x00009300,0x00002600,0x00003600,0x00003F00,0x0000F700,0x0000CC00,
0x00003400,0x0000A500,0x0000E500,0x0000F100,0x00007100,0x0000D800,0x00003100,0x00001500,
0x00000400,0x0000C700,0x00002300,0x0000C300,0x00001800,0x00009600,0x00000500,0x00009A00,
0x00000700,0x00001200,0x00008000,0x0000E200,0x0000EB00,0x00002700,0x0000B200,0x00007500,
0x00000900,0x00008300,0x00002C00,0x00001A00,0x00001B00,0x00006E00,0x00005A00,0x0000A000,
0x00005200,0x00003B00,0x0000D600,0x0000B300,0x00002900,0x0000E300,0x00002F00,0x00008400,
0x00005300,0x0000D100,0000000000,0x0000ED00,0x00002000,0x0000FC00,0x0000B100,0x00005B00,
0x00006A00,0x0000CB00,0x0000BE00,0x00003900,0x00004A00,0x00004C00,0x00005800,0x0000CF00,
0x0000D000,0x0000EF00,0x0000AA00,0x0000FB00,0x00004300,0x00004D00,0x00003300,0x00008500,
0x00004500,0x0000F900,0x00000200,0x00007F00,0x00005000,0x00003C00,0x00009F00,0x0000A800,
0x00005100,0x0000A300,0x00004000,0x00008F00,0x00009200,0x00009D00,0x00003800,0x0000F500,
0x0000BC00,0x0000B600,0x0000DA00,0x00002100,0x00001000,0x0000FF00,0x0000F300,0x0000D200,
0x0000CD00,0x00000C00,0x00001300,0x0000EC00,0x00005F00,0x00009700,0x00004400,0x00001700,
0x0000C400,0x0000A700,0x00007E00,0x00003D00,0x00006400,0x00005D00,0x00001900,0x00007300,
0x00006000,0x00008100,0x00004F00,0x0000DC00,0x00002200,0x00002A00,0x00009000,0x00008800,
0x00004600,0x0000EE00,0x0000B800,0x00001400,0x0000DE00,0x00005E00,0x00000B00,0x0000DB00,
0x0000E000,0x00003200,0x00003A00,0x00000A00,0x00004900,0x00000600,0x00002400,0x00005C00,
0x0000C200,0x0000D300,0x0000AC00,0x00006200,0x00009100,0x00009500,0x0000E400,0x00007900,
0x0000E700,0x0000C800,0x00003700,0x00006D00,0x00008D00,0x0000D500,0x00004E00,0x0000A900,
0x00006C00,0x00005600,0x0000F400,0x0000EA00,0x00006500,0x00007A00,0x0000AE00,0x00000800,
0x0000BA00,0x00007800,0x00002500,0x00002E00,0x00001C00,0x0000A600,0x0000B400,0x0000C600,
0x0000E800,0x0000DD00,0x00007400,0x00001F00,0x00004B00,0x0000BD00,0x00008B00,0x00008A00,
0x00007000,0x00003E00,0x0000B500,0x00006600,0x00004800,0x00000300,0x0000F600,0x00000E00,
0x00006100,0x00003500,0x00005700,0x0000B900,0x00008600,0x0000C100,0x00001D00,0x00009E00,
0x0000E100,0x0000F800,0x00009800,0x00001100,0x00006900,0x0000D900,0x00008E00,0x00009400,
0x00009B00,0x00001E00,0x00008700,0x0000E900,0x0000CE00,0x00005500,0x00002800,0x0000DF00,
0x00008C00,0x0000A100,0x00008900,0x00000D00,0x0000BF00,0x0000E600,0x00004200,0x00006800,
0x00004100,0x00009900,0x00002D00,0x00000F00,0x0000B000,0x00005400,0x0000BB00,0x00001600
},
{
0x00630000,0x007C0000,0x00770000,0x007B0000,0x00F20000,0x006B0000,0x006F0000,0x00C50000,
0x00300000,0x00010000,0x00670000,0x002B0000,0x00FE0000,0x00D70000,0x00AB0000,0x00760000,
0x00CA0000,0x00820000,0x00C90000,0x007D0000,0x00FA0000,0x00590000,0x00470000,0x00F00000,
0x00AD0000,0x00D40000,0x00A20000,0x00AF0000,0x009C0000,0x00A40000,0x00720000,0x00C00000,
0x00B70000,0x00FD0000,0x00930000,0x00260000,0x00360000,0x003F0000,0x00F70000,0x00CC0000,
0x00340000,0x00A50000,0x00E50000,0x00F10000,0x00710000,0x00D80000,0x00310000,0x00150000,
0x00040000,0x00C70000,0x00230000,0x00C30000,0x00180000,0x00960000,0x00050000,0x009A0000,
0x00070000,0x00120000,0x00800000,0x00E20000,0x00EB0000,0x00270000,0x00B20000,0x00750000,
0x00090000,0x00830000,0x002C0000,0x001A0000,0x001B0000,0x006E0000,0x005A0000,0x00A00000,
0x00520000,0x003B0000,0x00D60000,0x00B30000,0x00290000,0x00E30000,0x002F0000,0x00840000,
0x00530000,0x00D10000,0000000000,0x00ED0000,0x00200000,0x00FC0000,0x00B10000,0x005B0000,
0x006A0000,0x00CB0000,0x00BE0000,0x00390000,0x004A0000,0x004C0000,0x00580000,0x00CF0000,
0x00D00000,0x00EF0000,0x00AA0000,0x00FB0000,0x00430000,0x004D0000,0x00330000,0x00850000,
0x00450000,0x00F90000,0x00020000,0x007F0000,0x00500000,0x003C0000,0x009F0000,0x00A80000,
0x00510000,0x00A30000,0x00400000,0x008F0000,0x00920000,0x009D0000,0x00380000,0x00F50000,
0x00BC0000,0x00B60000,0x00DA0000,0x00210000,0x00100000,0x00FF0000,0x00F30000,0x00D20000,
0x00CD0000,0x000C0000,0x00130000,0x00EC0000,0x005F0000,0x00970000,0x00440000,0x00170000,
0x00C40000,0x00A70000,0x007E0000,0x003D0000,0x00640000,0x005D0000,0x00190000,0x00730000,
0x00600000,0x00810000,0x004F0000,0x00DC0000,0x00220000,0x002A0000,0x00900000,0x00880000,
0x00460000,0x00EE0000,0x00B80000,0x00140000,0x00DE0000,0x005E0000,0x000B0000,0x00DB0000,
0x00E00000,0x00320000,0x003A0000,0x000A0000,0x00490000,0x00060000,0x00240000,0x005C0000,
```

```
 0x00C20000,0x00D30000,0x00AC0000,0x00620000,0x00910000,0x00950000,0x00E40000,0x00790000,
 0x00E70000,0x00C80000,0x00370000,0x006D0000,0x008D0000,0x00D50000,0x004E0000,0x00A90000,
 0x006C0000,0x00560000,0x00F40000,0x00EA0000,0x00650000,0x007A0000,0x00AE0000,0x00080000,
 0x00BA0000,0x00780000,0x00250000,0x002E0000,0x001C0000,0x00A60000,0x00B40000,0x00C60000,
 0x00E80000,0x00DD0000,0x00740000,0x001F0000,0x004B0000,0x00BD0000,0x008B0000,0x008A0000,
 0x00700000,0x003E0000,0x00B50000,0x00660000,0x00480000,0x00030000,0x00F60000,0x000E0000,
 0x00610000,0x00350000,0x00570000,0x00B90000,0x00860000,0x00C10000,0x001D0000,0x009E0000,
 0x00E10000,0x00F80000,0x00980000,0x00110000,0x00690000,0x00D90000,0x008E0000,0x00940000,
 0x009B0000,0x001E0000,0x00870000,0x00E90000,0x00CE0000,0x00550000,0x00280000,0x00DF0000,
 0x008C0000,0x00A10000,0x00890000,0x000D0000,0x00BF0000,0x00E60000,0x00420000,0x00680000,
 0x00410000,0x00990000,0x002D0000,0x000F0000,0x00B00000,0x00540000,0x00BB0000,0x00160000
 },
 {
 0x63000000,0x7C000000,0x77000000,0x7B000000,0xF2000000,0x6B000000,0x6F000000,0xC5000000,
 0x30000000,0x01000000,0x67000000,0x2B000000,0xFE000000,0xD7000000,0xAB000000,0x76000000,
 0xCA000000,0x82000000,0xC9000000,0x7D000000,0xFA000000,0x59000000,0x47000000,0xF0000000,
 0xAD000000,0xD4000000,0xA2000000,0xAF000000,0x9C000000,0xA4000000,0x72000000,0xC0000000,
 0xB7000000,0xFD000000,0x93000000,0x26000000,0x36000000,0x3F000000,0xF7000000,0xCC000000,
 0x34000000,0xA5000000,0xE5000000,0xF1000000,0x71000000,0xD8000000,0x31000000,0x15000000,
 0x04000000,0xC7000000,0x23000000,0xC3000000,0x18000000,0x96000000,0x05000000,0x9A000000,
 0x07000000,0x12000000,0x80000000,0xE2000000,0xEB000000,0x27000000,0xB2000000,0x75000000,
 0x09000000,0x83000000,0x2C000000,0x1A000000,0x1B000000,0x6E000000,0x5A000000,0xA0000000,
 0x52000000,0x3B000000,0xD6000000,0xB3000000,0x29000000,0xE3000000,0x2F000000,0x84000000,
 0x53000000,0xD1000000,0000000000,0xED000000,0x20000000,0xFC000000,0xB1000000,0x5B000000,
 0x6A000000,0xCB000000,0xBE000000,0x39000000,0x4A000000,0x4C000000,0x58000000,0xCF000000,
 0xD0000000,0xEF000000,0xAA000000,0xFB000000,0x43000000,0x4D000000,0x33000000,0x85000000,
 0x45000000,0xF9000000,0x02000000,0x7F000000,0x50000000,0x3C000000,0x9F000000,0xA8000000,
 0x51000000,0xA3000000,0x40000000,0x8F000000,0x92000000,0x9D000000,0x38000000,0xF5000000,
 0xBC000000,0xB6000000,0xDA000000,0x21000000,0x10000000,0xFF000000,0xF3000000,0xD2000000,
 0xCD000000,0x0C000000,0x13000000,0xEC000000,0x5F000000,0x97000000,0x44000000,0x17000000,
 0xC4000000,0xA7000000,0x7E000000,0x3D000000,0x64000000,0x5D000000,0x19000000,0x73000000,
 0x60000000,0x81000000,0x4F000000,0xDC000000,0x22000000,0x2A000000,0x90000000,0x88000000,
 0x46000000,0xEE000000,0xB8000000,0x14000000,0xDE000000,0x5E000000,0x0B000000,0xDB000000,
 0xE0000000,0x32000000,0x3A000000,0x0A000000,0x49000000,0x06000000,0x24000000,0x5C000000,
 0xC2000000,0xD3000000,0xAC000000,0x62000000,0x91000000,0x95000000,0xE4000000,0x79000000,
 0xE7000000,0xC8000000,0x37000000,0x6D000000,0x8D000000,0xD5000000,0x4E000000,0xA9000000,
 0x6C000000,0x56000000,0xF4000000,0xEA000000,0x65000000,0x7A000000,0xAE000000,0x08000000,
 0xBA000000,0x78000000,0x25000000,0x2E000000,0x1C000000,0xA6000000,0xB4000000,0xC6000000,
 0xE8000000,0xDD000000,0x74000000,0x1F000000,0x4B000000,0xBD000000,0x8B000000,0x8A000000,
 0x70000000,0x3E000000,0xB5000000,0x66000000,0x48000000,0x03000000,0xF6000000,0x0E000000,
 0x61000000,0x35000000,0x57000000,0xB9000000,0x86000000,0xC1000000,0x1D000000,0x9E000000,
 0xE1000000,0xF8000000,0x98000000,0x11000000,0x69000000,0xD9000000,0x8E000000,0x94000000,
 0x9B000000,0x1E000000,0x87000000,0xE9000000,0xCE000000,0x55000000,0x28000000,0xDF000000,
 0x8C000000,0xA1000000,0x89000000,0x0D000000,0xBF000000,0xE6000000,0x42000000,0x68000000,
 0x41000000,0x99000000,0x2D000000,0x0F000000,0xB0000000,0x54000000,0xBB000000,0x16000000
 }
};

/*----------------- The workspace ------------------------------*/

static u32 Ekey[44];    /* The expanded key */

/*------ The round Function.  4 table lookups and 4 Exors ------*/
#define f_rnd(x, n)                     \
  ( ft_tab[0][byte0(x[n])]              \
  ^ ft_tab[1][byte1(x[(n + 1) & 3])]    \
  ^ ft_tab[2][byte2(x[(n + 2) & 3])]    \
  ^ ft_tab[3][byte3(x[(n + 3) & 3])] )

#define f_round(bo, bi, k)          \
    bo[0] = f_rnd(bi, 0) ^ k[0];    \
    bo[1] = f_rnd(bi, 1) ^ k[1];    \
    bo[2] = f_rnd(bi, 2) ^ k[2];    \
    bo[3] = f_rnd(bi, 3) ^ k[3];    \
    k += 4

/*--- The S Box lookup used in constructing the Key schedule ---*/
#define ls_box(x)       \
 (  fl_tab[0][byte0(x)] \
  ^ fl_tab[1][byte1(x)] \
  ^ fl_tab[2][byte2(x)] \
  ^ fl_tab[3][byte3(x)] )

/*------------ The last round function (no MixColumn) ----------*/
#define lf_rnd(x, n)                    \
  ( fl_tab[0][byte0(x[n])]              \
  ^ fl_tab[1][byte1(x[(n + 1) & 3])]    \
  ^ fl_tab[2][byte2(x[(n + 2) & 3])]    \
```

```
  ^ fl_tab[3][byte3(x[(n + 3) & 3])] )


/*-------------------------------------------------------------
 * RijndaelKeySchedule
 *   Initialise the key schedule from a supplied key
 */
void RijndaelKeySchedule(u8 key[16])
{
    u32  t;
    u32  *ek=Ekey,      /* pointer to the expanded key   */
         *rc=rnd_con;   /* pointer to the round constant */

    Ekey[0] = u32_in(key     );
    Ekey[1] = u32_in(key +  4);
    Ekey[2] = u32_in(key +  8);
    Ekey[3] = u32_in(key + 12);

    while(ek < Ekey + 40)
    {
        t = rot3(ek[3]);
        ek[4] = ek[0] ^ ls_box(t) ^ *rc++;
        ek[5] = ek[1] ^ ek[4];
        ek[6] = ek[2] ^ ek[5];
        ek[7] = ek[3] ^ ek[6];
        ek += 4;
    }
}

/*-------------------------------------------------------------
 * RijndaelEncrypt
 *   Encrypt an input block
 */
void RijndaelEncrypt(u8 in[16], u8 out[16])
{
    u32    b0[4], b1[4], *kp = Ekey;

    b0[0] = u32_in(in     ) ^ *kp++;
    b0[1] = u32_in(in +  4) ^ *kp++;
    b0[2] = u32_in(in +  8) ^ *kp++;
    b0[3] = u32_in(in + 12) ^ *kp++;

    f_round(b1, b0, kp);
    f_round(b0, b1, kp);
    f_round(b1, b0, kp);
    f_round(b0, b1, kp);
    f_round(b1, b0, kp);
    f_round(b0, b1, kp);
    f_round(b1, b0, kp);
    f_round(b0, b1, kp);
    f_round(b1, b0, kp);

    u32_out(out,      lf_rnd(b1, 0) ^ kp[0]);
    u32_out(out +  4, lf_rnd(b1, 1) ^ kp[1]);
    u32_out(out +  8, lf_rnd(b1, 2) ^ kp[2]);
    u32_out(out + 12, lf_rnd(b1, 3) ^ kp[3]);
}
```

Annex A (informative): Change history

Change history					
TSG SA #	**Version**	**CR**	**Tdoc SA**	**New Version**	**Subject/Comment**
SP-10	SAGE v 1.1	-	SP-010673	3.0.0	Approved as Release 1999
SP-11	3.0.0	-	-	4.0.0	Updated to Release 4

History

Document history		
V4.0.0	April 2001	Publication