ETSI TS 103 718 V1.1.1 (2020-10)

CYBER;
External encodings for the Advanced Encryption Standard

Reference
DTS/CYBER-0044

Keywords
algorithm, encryption, security

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

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables 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 (https://ipr.etsi.org/).

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.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Cyber Security (CYBER).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

Software security techniques can be used to protect software assets in the clear-box model. An overview of such techniques can be found in ETSI TR 103 642 [i.1].

As described in ETSI TR 103 642 [i.1], one class of security techniques is referred to as clear-box cryptography. Clear-box cryptography aims to protect the secret key of a keyed cryptographic algorithm in the clear-box model. The concept of clear-box cryptography and the first clear-box implementation were introduced in [i.2]. The cryptographic algorithm considered in [i.2] is AES [1], a well-known block cipher. Today, clear-box AES implementations are widely used in practice.

As also described in ETSI TR 103 642 [i.1], cryptanalysis, DCA, and DFA are threats to the security of a clear-box block cipher implementation against key extraction. To the current state of knowledge, the combination of external encodings and clear-box transformations can provide a good security measure against these threats. In particular, all published clear-box AES implementations make use of external encodings.

1 Scope

The present document specifies external encodings for AES. The external encodings specified in the present document can also be applied to other block ciphers; in particular, they can be applied to any block cipher with a block size of either 64 or 128 bits.

The present document does not define clear-box transformations that are applied to obfuscate the implementation of the block cipher and the external encodings. Requirements related to the clear-box implementation of the external encodings are also outside the scope of the present document.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] FIPS PUB 197 (November 2001): "Announcing the Advanced Encryption Standard (AES)".

NOTE: Available at: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TR 103 642 (V1.1.1): "CYBER; Security techniques for protecting software in a white box model".

NOTE: Available at: https://www.etsi.org/deliver/etsi_tr/103600_103699/103642/01.01.01_60/tr_103642v010101p.pdf.

[i.2] S. Chow, P. Eisen, H. Johnson, and P.C. van Oorschot: "White-box cryptography and an AES implementation", in Selected Areas in Cryptography 2002, LNCS 2595, K. Nyberg and H. Heys, Eds., pp. 250-270, Springer, 2003.

[i.3] A. Amadori, W. Michiels, and P. Roelse: "A DFA attack on white-box implementations of AES with external encodings", in Selected Areas in Cryptography 2019, LNCS 11959, K.G. Paterson and D. Stebila, Eds., pp. 591-617, Springer, 2020.

[i.4] D.E. Knuth: "The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Third Edition", Addison-Wesley, 1997.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

bit: binary digit having a value of zero or one

block cipher: function that maps a plaintext to a ciphertext using a cipher key

cipher key: secret cryptographic key, used as input to a block cipher

ciphertext: sequence of bits that is the output of a block cipher or an input of an inverse block cipher

clear-box model: model in which an adversary is assumed to have full access to a software binary and its execution environment

NOTE: Clear-box is also referred to as white-box in other documentation.

external encoding: input encoding or output encoding

external encoding key: input encoding key or output encoding key

input encoding: function that is performed before a block cipher, mapping an input vector to an output vector using an input encoding key

input encoding key: secret cryptographic key, used as input to an input encoding

matrix: collection of bits arranged in a number of rows and a number of columns

output encoding: function that is performed after a block cipher, mapping an input vector to an output vector using an output encoding key

output encoding key: secret cryptographic key, used as input to an output encoding

plaintext: sequence of bits that is an input to a block cipher or the output of an inverse block cipher

vector: sequence of bits

3.2 Symbols

For the purposes of the present document, the following symbols apply:

$\oplus$	Addition of two bits, addition of two vectors.
$\cdot$	Multiplication of two bits, multiplication of a bit with a vector, multiplication of a vector with a matrix, multiplication of two matrices.
$\|\|$	Concatenation of two vectors.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AES	Advanced Encryption Standard
DCA	Differential Computation Analysis
DFA	Differential Fault Analysis
RNG	Random Number Generator

4 Preliminaries

4.1 Basic mathematical concepts

4.1.1 Bits, vectors, and matrices

The addition of two bits is defined as: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, and $1 \oplus 1 = 0$. The multiplication of two bits is defined as: $0 \cdot 0 = 0$, $0 \cdot 1 = 0$, $1 \cdot 0 = 0$, and $1 \cdot 1 = 1$.

An m-bit vector is a sequence of m bits.

EXAMPLE 1: (1, 0, 1, 1) is a 4-bit vector.

The bits of an m-bit vector are numbered 1 to m from left to right in the present document.

EXAMPLE 2: $(a_1, a_2, \ldots, a_7)$ with a_i equal to 0 or 1 for i = 1, 2, …, 7 is a 7-bit vector.

If the value of every bit of a vector is zero, then the vector is referred to as a zero vector.

A bit can be multiplied with an m-bit vector $(a_1, a_2, \ldots, a_m)$. This multiplication is defined as: $0 \cdot (a_1, a_2, \ldots, a_m) = (0, 0, \ldots, 0)$ and $1 \cdot (a_1, a_2, \ldots, a_m) = (a_1, a_2, \ldots, a_m)$.

An m × m matrix is a matrix with m rows and m columns.

EXAMPLE 3: The following matrix is a 3 × 3 matrix:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

The entries of an m × m matrix A are denoted and numbered as follows in the present document:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,m} \end{bmatrix}$$

An m × m matrix A with $a_{i,j} = 1$ if i = j and $a_{i,j} = 0$ if $i \neq j$ is referred to as an identity matrix and is denoted by I_m. If $a_{i,j} = 0$ for all values of i and j, then A is called a zero matrix.

If $A^{(k)}$ for k = 1, 2 , …, r are m × m matrices, then the entries of the (rm) × (rm) matrix $A = \text{diag}(A^{(1)}, A^{(2)}, \ldots, A^{(r)})$ are defined as follows: for $1 \leq k \leq r$ and $1 \leq i, j \leq m$, $a_{(k-1)m+i,\,(k-1)m+j} = a^{(k)}_{i,j}$ if $|i - j| < m$, and $a_{i,j} = 0$ if $|i - j| \geq m$. In other words, if 0_m denotes an m × m zero matrix, then the matrix A is defined as:

$$A = \begin{bmatrix} A^{(1)} & 0_m & \cdots & 0_m \\ 0_m & A^{(2)} & \cdots & 0_m \\ \vdots & \vdots & \ddots & \vdots \\ 0_m & 0_m & & A^{(r)} \end{bmatrix}$$

EXAMPLE 4:

$$\text{diag}\left(\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

4.1.2 Concatenation, addition, and linear combination of vectors

The concatenation of the m-bit vector $(a_1, a_2, \ldots, a_m)$ and the k-bit vector $(b_1, b_2, \ldots, b_k)$ is defined as: $(a_1, a_2, \ldots, a_m) \| (b_1, b_2, \ldots, b_k) = (a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_k)$.

EXAMPLE 1: The concatenation of the 3-bit vector (0, 1, 0) and the 4-bit vector (1, 1, 1, 1) is a 7-bit vector and equals (0, 1, 0, 1, 1, 1, 1).

The addition of two m-bit vectors $(a_1, a_2, \ldots, a_m)$ and $(b_1, b_2, \ldots, b_m)$ is defined as: $(a_1, a_2, \ldots, a_m) \oplus (b_1, b_2, \ldots, b_m) = (a_1 \oplus b_1, a_2 \oplus b_2, \ldots, a_m \oplus b_m)$.

EXAMPLE 2: $(0, 1, 0, 1, 1) \oplus (0, 1, 1, 0, 1) = (0, 0, 1, 1, 0)$.

If $X_1, X_2, \ldots, X_i$ are m-bit vectors and if $b_1, b_2, \ldots, b_i$ are bits for some value of $i \geq 1$, then the linear combination of these vectors with these scalars is $b_1 \cdot X_1 \oplus b_2 \cdot X_2 \oplus \ldots \oplus b_i \cdot X_i$. If V is a set of m-bit vectors, then the span of V, denoted by span(V), is defined as the set of all linear combinations of the elements of V. The span of the empty set is the set containing only the zero vector.

EXAMPLE 3: If $V = \{(0, 1, 0), (1, 1, 0)\}$, then $\text{span}(V) = \{(0, 0, 0), (0, 1, 0), (1, 1, 0), (1, 0, 0)\}$.

4.1.3 Vector - matrix and matrix - matrix multiplication

If $X = (x_1, x_2, \ldots, x_m)$ is an m-bit vector and if the entries of the $m \times m$ matrix A are denoted as $a_{i,j}$ with $1 \leq i, j \leq m$ (see also clause 4.1.1), then the product of X and A is defined as:

$$X \cdot A = x_1 \cdot (a_{1,1}, a_{1,2}, \ldots, a_{1,m}) \oplus x_2 \cdot (a_{2,1}, a_{2,2}, \ldots, a_{2,m}) \oplus \ldots \oplus x_m \cdot (a_{m,1}, a_{m,2}, \ldots, a_{m,m}).$$

EXAMPLE 1:

$$(1, 1, 0) \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = 1 \cdot (1, 1, 1) \oplus 1 \cdot (0, 1, 0) \oplus 0 \cdot (1, 0, 0) = (1, 0, 1)$$

$X \cdot A$ is an m-bit vector and $X \cdot I_m = X$.

The product of two $m \times m$ matrices A and B is an $m \times m$ matrix. If $C = A \cdot B$ and if:

$$B = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,m} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,m} \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,m} \end{bmatrix},$$

then $c_{i,j} = a_{i,1} b_{1,j} \oplus a_{i,2} b_{2,j} \oplus \ldots \oplus a_{i,m} b_{m,j}$ for $1 \leq i, j \leq m$.

EXAMPLE 2:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

A is called an invertible matrix if there exists a unique matrix B such that $A \cdot B = I_m$. If A is an invertible matrix, then the rows of A span the set of all m-bit vectors. The inverse of A is denoted by A^{-1}. Efficient algorithms exist to test whether a matrix is invertible and to compute the inverse of an invertible matrix.

4.2 Block ciphers

A block cipher is an encryption function E_K that maps an n-bit plaintext to an n-bit ciphertext. The parameter n is referred to as the block size, and in the present document, its value is either 64 or 128 (in the case of AES, n = 128). A plaintext and a ciphertext are denoted by P and C respectively. The encryption function E_K is parameterized by a key K and E_K is an invertible function for each value of K. The inverse function, referred to as the decryption function, is also parameterized by K and is denoted by D_K. The input to D_K is an n-bit ciphertext and its output is an n-bit plaintext. The encryption function E_K and the decryption function D_K are depicted in Figure 1.

Figure 1: Encryption of a plaintext and decryption of a ciphertext using a block cipher

4.3 External encodings

In the present document an external encoding is a function that maps an n-bit input vector to an n-bit output vector. There are two types of external encoding:

1) The first type is referred to as an input encoding and is denoted by F_{K1};

2) The second type is referred to as an output encoding and is denoted by G_{K2}.

Both type of encoding are parameterized by a key and both type of encoding are invertible functions for each value of the key. K1 is referred to as an input encoding key and K2 is referred to as an output encoding key. In the present document, an external encoding key K' is either an input encoding key K1 or an output encoding key K2. The size of K' can be large compared to the size of a block cipher key; see also clause 5.4.

The external encodings are composed with the function E_K to yield a new function $E'_{K,K1,K2}$ that maps an n-bit input vector to an n-bit output vector. This function is parameterized by K, K1, and K2 and defined by:

$$E'_{K,K1,K2}(X) = G_{K2}(E_K(F_{K1}(X))).$$

That is, first the input encoding is applied to the n-bit input vector, then E_K is applied to the output vector of the input encoding, and finally the output encoding is applied to the output of E_K. If the input to E_K is denoted by P, and if the corresponding output is denoted by C (as in clause 4.1), then $X = F_{K1}^{-1}(P)$ and $E'_{K,K1,K2}(X) = G_{K2}(C)$.

The inverse of $E'_{K,K1,K2}$ is denoted by $D'_{K,K1,K2}$. If Y denotes the n-bit input vector to $D'_{K,K1,K2}$, then

$$D'_{K,K1,K2}(Y) = F_{K1}^{-1}(D_K(G_{K2}^{-1}(Y))).$$

Further, if the input to D_K is denoted by C, and if the corresponding output is denoted by P (as in clause 4.1), then $Y = G_{K2}(C)$ and $D'_{K,K1,K2}(Y) = F_{K1}^{-1}(P)$. The functions $E'_{K,K1,K2}$ and $D'_{K,K1,K2}$ are depicted in Figure 2.

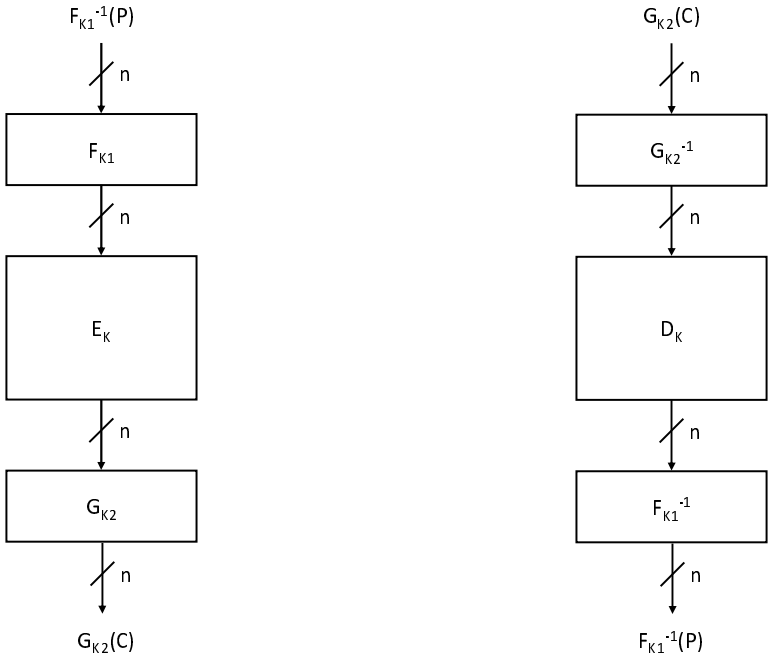

Figure 2: A block cipher with external encodings and its inverse

5 External encodings specification

5.1 Basic functions

The external encodings specified in the present document use two types of function. The first type of function maps an 8-bit input vector to an 8-bit output vector. The function is parameterized by the external encoding key K' (which is either K1 or K2, depending on the type of external encoding) and the function is invertible for each value of K'. As detailed in clauses 5.2 and 5.3, if the block size of the block cipher equals n, then one external encoding uses n/8 of these functions, denoted by $T_{K'}^{(i)}$ for i = 1, 2, …, n/8. The present document assumes that for given K' each $T_{K'}^{(i)}$ is represented by a look-up table.

The second type of function maps an n-bit input vector to an n-bit output vector. This function is also parameterized by the external encoding key K' and for every choice of K', the function defines an invertible affine mapping on the vector space consisting of all n-bit vectors. The function is denoted by $H_{K'}$ and the present document assumes that $H_{K'}$ is represented by an n × n invertible matrix $A_{K'}$ and an n-bit vector $b_{K'}$ such that $H_{K'}(X) = X \cdot A_{K'} \oplus b_{K'}$. As detailed in clauses 5.2 and 5.3, one external encoding uses one function $H_{K'}$.

5.2 Input encoding

If the n bits of the vector X input to the input encoding F_{K1} are denoted by x_i for i = 1, 2, …, n, and if the 8-bit vector X_j is defined as $X_j = (x_{8j-7}, x_{8j-6}, x_{8j-5}, x_{8j-4}, x_{8j-3}, x_{8j-2}, x_{8j-1}, x_{8j})$ for j = 1, 2, …, n/8, then F_{K1} shall be as follows:

$$F_{K1}(X) = H_{K1}(T_{K1}^{(1)}(X_1) \| T_{K1}^{(2)}(X_2) \| \ldots \| T_{K1}^{(n/8)}(X_{n/8}))$$

$$= (T_{K1}^{(1)}(X_1) \| T_{K1}^{(2)}(X_2) \| \ldots \| T_{K1}^{(n/8)}(X_{n/8})) \cdot A_{K1} \oplus b_{K1}.$$

This function is depicted in Figure 3.

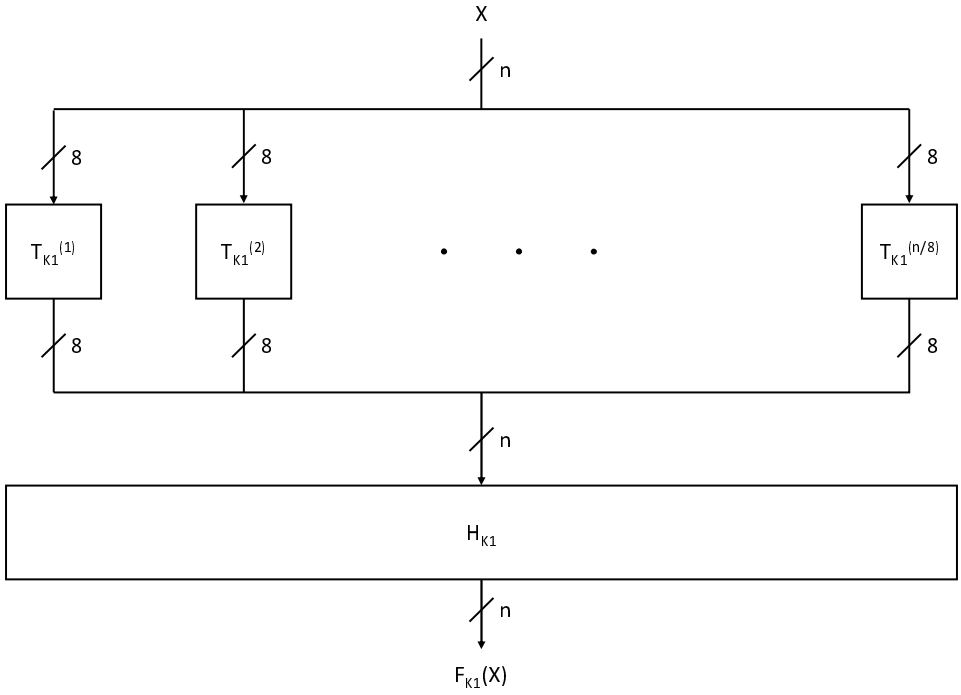

Figure 3: Input encoding

Inverse input encoding: let Y denote the n-bit input vector of the inverse input encoding F_{K1}^{-1} and define the n-bit vector Z as $Z = H_{K1}^{-1}(Y) = Y \cdot A_{K1}^{-1} \oplus b_{K1} \cdot A_{K1}^{-1}$. If the n bits of Z are denoted by z_i for i = 1, 2, …, n, and if the 8-bit vector Z_j is defined as $Z_j = (z_{8j-7}, z_{8j-6}, z_{8j-5}, z_{8j-4}, z_{8j-3}, z_{8j-2}, z_{8j-1}, z_{8j})$ for j = 1, 2, …, n/8, then F_{K1}^{-1} shall be as follows:

$$F_{K1}^{-1}(Y) = (T_{K1}^{(1)})^{-1}(Z_1) \| (T_{K1}^{(2)})^{-1}(Z_2) \| \ldots \| (T_{K1}^{(n/8)})^{-1}(Z_{n/8}).$$

5.3 Output encoding

Let X denote the n-bit input vector of the output encoding G_{K2} and define the n-bit vector Z as $Z = H_{K2}(X) = X \cdot A_{K2} \oplus b_{K2}$. If the n bits of Z are denoted by z_i for i = 1, 2, …, n, and if the 8-bit vector Z_j is defined as $Z_j = (z_{8j-7}, z_{8j-6}, z_{8j-5}, z_{8j-4}, z_{8j-3}, z_{8j-2}, z_{8j-1}, z_{8j})$ for j = 1, 2, …, n/8, then G_{K2} shall be as follows:

$$G_{K2}(X) = T_{K2}^{(1)}(Z_1) \| T_{K2}^{(2)}(Z_2) \| \ldots \| T_{K2}^{(n/8)}(Z_{n/8}).$$

This function is depicted in Figure 4.

Inverse output encoding: if the n bits of the vector Y input to the inverse output encoding G_{K2}^{-1} are denoted by y_i for i = 1, 2, …, n, and if the 8-bit vector Y_j is defined as $Y_j = (y_{8j-7}, y_{8j-6}, y_{8j-5}, y_{8j-4}, y_{8j-3}, y_{8j-2}, y_{8j-1}, y_{8j})$ for j = 1, 2, …, n/8, then G_{K2}^{-1} shall be as follows:

$$G_{K2}^{-1}(Y) = H_{K2}^{-1}((T_{K2}^{(1)})^{-1}(Y_1) \| (T_{K2}^{(2)})^{-1}(Y_2) \| \ldots \| (T_{K2}^{(n/8)})^{-1}(Y_{n/8}))$$

$$= ((T_{K2}^{(1)})^{-1}(Y_1) \| (T_{K2}^{(2)})^{-1}(Y_2) \| \ldots \| (T_{K2}^{(n/8)})^{-1}(Y_{n/8})) \cdot A_{K2}^{-1} \oplus b_{K2} \cdot A_{K2}^{-1}.$$

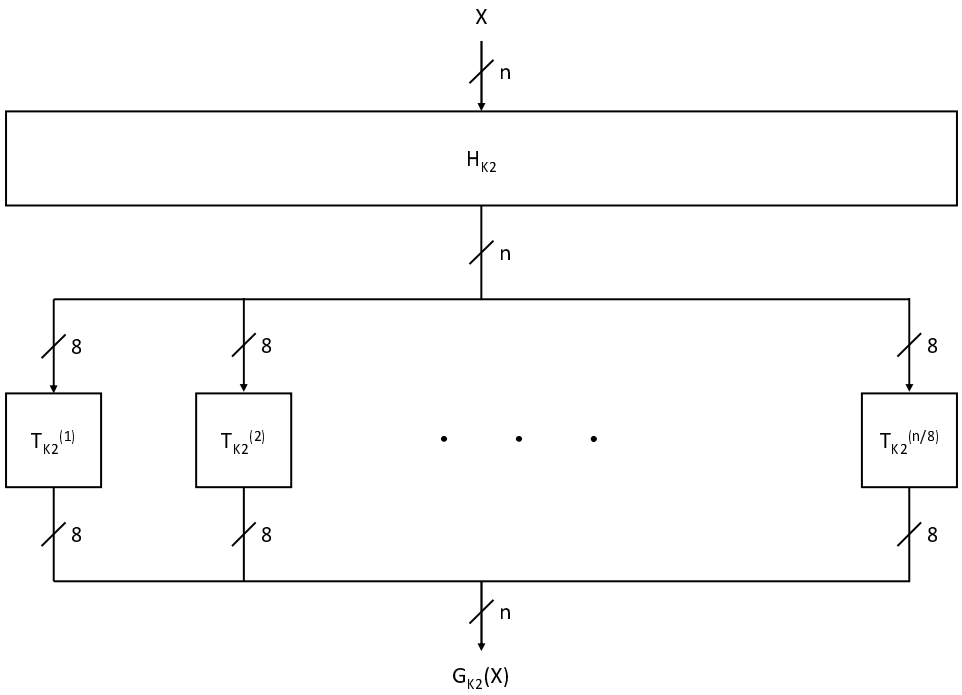

Figure 4: Output encoding

5.4 Key sizes

It is assumed that each function $T_{K'}^{(i)}$ with $1 \le i \le n/8$ is represented by a look-up table and that $H_{K'}$ is represented by an $n \times n$ invertible matrix $A_{K'}$ and an n-bit vector $b_{K'}$ (see clause 5.1). The external encoding key K' is defined by the contents of these look-up tables, the contents of $A_{K'}$, and the contents of $b_{K'}$. As a result, K' can be represented using $(256 \cdot 8 \cdot n/8) + n^2 + n$ bits = $n^2 + 257n$ bits. When n = 64, K' can be represented using 20,544 bits; when n = 128, K' can be represented using 49,280 bits. These key sizes can be too large for certain applications.

EXAMPLE: The function $E'_{K,K1,K2}(X)$ is implemented in a clear-box environment, and clear-box cryptography techniques are used to protect it. The function $E'_{K,K1,K2}(X)$ interfaces with an implementation of the function $D_K(G_{K2}^{-1}(Y))$. The function $D_K(G_{K2}^{-1}(Y))$ is implemented inside a security processor, and hardware security measures are used to protect the implementation of $D_K(G_{K2}^{-1}(Y))$. Furthermore, the amount of memory available in the security processor to store the output encoding key K2 is less than 20,544 bits.

This clause therefore defines different key sizes. A parameter t with $t \in \{1, 2, 4, 8\}$ if n = 64 and $t \in \{1, 2, 4, 8, 16\}$ if n = 128 is used to define one key size for each combination of t and n:

- $T_{K'}^{(i)}$ with $1 \le i \le t$ shall be randomly generated invertible functions, and $T_{K'}^{(i)}$ with $t < i \le n/8$ shall be as follows: $T_{K'}^{(j+t)} = T_{K'}^{(j)}$ for j = 1, 2, …, n/8 – t.
- If s = n/(8t), then $A_{K'}^{(i)}$ shall be randomly generated 8t × 8t invertible matrices for i = 1, 2, …, s, and $A_{K'}$ shall be as follows: $A_{K'} = \text{diag}(A_{K'}^{(1)}, A_{K'}^{(2)}, \ldots, A_{K'}^{(s)})$.
- The n-bit vector $b_{K'}$ shall be generated at random.

Tables 1 and 2 list the key sizes for each combination of t and n. The key size assumes that K' is defined as the contents of the look-up tables representing $T_{K'}^{(i)}$ for i = 1, 2, …, t, the contents of the matrices $A_{K'}^{(1)}, A_{K'}^{(2)}, \ldots, A_{K'}^{(s)}$, and the contents of $b_{K'}$.

An implementation of an external encoding shall support at least one of the key sizes listed in Table 1 or Table 2.

Table 1: Key sizes for n = 64

t	Key size
8	20,544 bits
4	10,304 bits
2	5,184 bits
1	2,624 bits

Table 2: Key sizes for n = 128

t	Key size
16	49,280 bits
8	24,704 bits
4	12,416 bits
2	6,272 bits
1	3,200 bits

NOTE 1: For a fixed value of n, the set of keys with a key size k is a subset of the set of keys with a key size k' > k. In other words, an implementation that supports a specific key size for a value of n also supports all smaller-sized keys for this value of n.

NOTE 2: For the largest key size there is redundancy in the representation of the key in that the addition with the n-bit vector $b_{K'}$ could also be implemented by adapting the contents of the look-up tables representing the functions $T_{K'}^{(i)}$ with $1 \leq i \leq n/8$ accordingly. For the reduced key sizes this is not necessarily the case. For consistency, and since the increase in key size is negligible, it is assumed that the n-bit vector $b_{K'}$ is also included separately in the representation of the key with the largest key size.

NOTE 3: An attack on a specific class of clear-box implementations of AES with external encodings was presented in [i.3]. In particular, this class of clear-box implementations uses the same output encoding as defined in the present document for n = 128 and t = 1. For details and example measures to prevent this attack, refer to [i.3].

6 Life cycle of an external encoding key

6.1 Example system

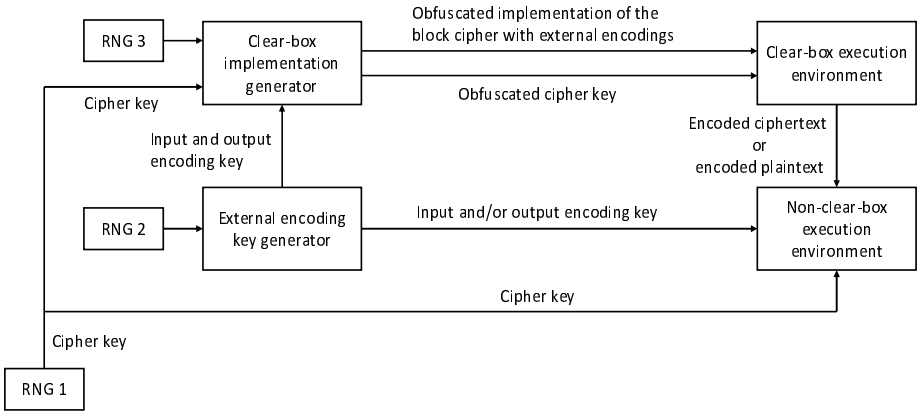

Figure 5: System components

Figure 5 depicts basic components of an example system that uses a clear-box implementation of a block cipher with external encodings. The example system is used to describe the life cycle stages of an external encoding key in clauses 6.2 and 6.3.

6.2 Key generation and key distribution

The key generation stage shown in Figure 5 uses RNG 1 to generate cipher keys, and it uses RNG_2 and an external encoding key generator to generate external encoding keys.

- The confidentiality of an external encoding key shall be protected during its generation.

EXAMPLE 1: Physical security measures, for instance offered by a stand-alone computer that is located in a physically protected area, can provide this protection.

An input encoding key, an output encoding key, and a cipher key are then input to a clear-box implementation generator.

- The confidentiality and authenticity of an external encoding key shall be protected during its distribution to the clear-box implementation generator.

EXAMPLE 2: Cryptographic techniques such as encryption, a message authentication code, and/or a digital signature can provide this protection. Alternatively, the distribution channel can be protected by physical security measures.

The example system depicted in the figure assumes that the block cipher and the external encodings are fixed and known to the clear-box generator. In addition to the keys, the clear-box generator takes a random number from RNG 3 as input to generate the obfuscated implementation of the block cipher with external encodings (including the external encoding keys) and an obfuscated cipher key. Next, these two outputs are distributed to the clear-box execution environment.

In the example system, it is assumed that multiple obfuscated cipher keys can be generated for one obfuscated implementation of the block cipher with the same external encoding keys. If this is done, then the obfuscated implementation of the block cipher with external encodings (including the external encoding keys) can be distributed once to the clear-box environment, and one obfuscated cipher key can be distributed to this environment for every cipher key. Such an implementation is referred to as a dynamic-key clear-box implementation.

If one or both external encoding keys need to be updated in the clear-box implementation, then the example system assumes that a new obfuscated implementation of the block cipher with external encodings is generated and distributed to the clear-box environment.

In Figure 5, the clear-box implementation interfaces with an implementation of the block cipher with one or two external encodings that is executed in a non-clear-box environment. The corresponding input and/or output encoding key, and the corresponding cipher key are therefore also distributed to this non-clear-box environment.

- The confidentiality and authenticity of an external encoding key shall be protected during its distribution to a non-clear-box execution environment.

EXAMPLE 3: Cryptographic techniques such as encryption, a message authentication code, and/or a digital signature can provide this protection. Alternatively, the distribution channel can be protected by physical security measures.

6.3 Key storage and key use

The obfuscated implementation of the block cipher, including the obfuscated external encoding keys, is stored and used in the clear-box environment. In addition, in the example system shown in Figure 5, the input and/or output encoding keys are stored and used in the non-clear-box execution environment.

- The confidentiality and integrity of an external encoding key shall be protected during its storage and use in a non-clear-box execution environment.

EXAMPLE 1: The execution environment of a security processor and its built-in security measures can provide this protection.

- An external encoding key shall be protected against key misuse to ensure that the key can be used only for its intended purpose. As a minimum, the purpose metadata shall include the associated block cipher, its operation (encrypt or decrypt), the size of the external encoding key, and the type of the encoding (input or output).

EXAMPLE 2: Physical protection (tamper-resistant hardware) and/or cryptographic techniques can provide this protection.

In a non-clear-box implementation of the block cipher with an input encoding and/or an output encoding, there are 2 or 3 different keys: a cipher key and an input encoding key and/or an output encoding key. In practice, these keys, or 2 out of 3 keys, can be securely linked together so that they can only be used simultaneously. In general, linking keys will reduce the number of keys of the block cipher with external encodings from 2 to 1 or from 3 to either 2 or 1 from the adversary's perspective. Reducing the number of keys can prevent certain types of attack.

Annex A (informative): External encoding key generator

This informative annex contains pseudo-code for an external encoding key generator. This generator can be used for generating external encoding keys. The pseudo-code comprises three algorithms. These algorithms assume that the following two routines are available for generating random numbers:

```
random-integer(i)
    INPUT:     an integer 1 ≤ i ≤ 255
    OUTPUT:    a random integer in the range [0,i]

random-m-bit-vector(m)
    INPUT:     m ∈ {8, 16, 32, 64, 128}
    OUTPUT:    a random m-bit vector
```

The `random` module of the programming language Python® contains two routines, referred to as `random.randint` and `random.getrandbits`, that can be used to define the above mentioned routines. The routine call `random.randint(0,i)` returns a random Python® integer in the range [0, i] and the routine call `random.getrandbits(m)` returns a Python® integer with m random bits. For details, refer to https://docs.python.org/3/library/random.html.

Similar but cryptographically stronger routines, referred to as `secrets.randbelow` and `secrets.randbits` are available in the `secrets` module of Python®. The routine call `secrets.randbelow(i+1)` returns a random Python® integer in the range [0, i] and the call `secrets.randbits(m)` returns a Python® integer with m random bits. For details, refer to https://docs.python.org/3/library/secrets.html.

The first algorithm constructs a random permutation on 8-bit vectors using a modern variant of the Fisher-Yates shuffle, also referred to as Durstenfeld's algorithm [i.4]. The routine `binary-vector(i)` in the first algorithm takes an integer i with $0 \le i \le 255$ as input and returns the vector $(a_1, a_2, \ldots, a_8)$ such that $i = \Sigma_{j=0,1,\ldots,7} a_{8-j} 2^j$.

```
ALGORITHM random-permutation()
    INPUT:     none
    OUTPUT:    a permutation on 8-bit vectors
FOR i FROM 0 TO 255 T[i] ← binary-vector(i)
FOR i FROM 255 DOWNTO 1 DO
    j ← random-integer(i)
    Exchange T[i] and T[j]
RETURN T
```

The second algorithm generates an m × m random invertible matrix with $m \in \{8, 16, 32, 64, 128\}$. It uses a brute force method to build up the matrix row by row. In the description of the algorithm, $C^{(i)}$ denotes the i^{th} row of the m × m matrix C for i = 1, 2, …, m.

```
ALGORITHM random-invertible-matrix(m)
    INPUT:     m ∈ {8, 16, 32, 64, 128}
    OUTPUT:    an m × m invertible matrix C
i ← 1
WHILE i ≤ m DO
    C(i) ← random-m-bit-vector(m)
    IF C(i) ∉ span({C(1), C(2), …, C(i-1)}) THEN i ← i + 1
RETURN C
```

Since the span contains 2^{i-1} vectors, the probability p_i that the assignment `i ← i + 1` is executed equals $(2^m - 2^{i-1}) / 2^m$ for i = 1, 2, …, m. The expected number of iterations of the `WHILE` loop therefore equals:

$$\Sigma_{i=1,2\ldots,m} 1 / p_i = \Sigma_{i=0,1\ldots,m-1} 2^m / (2^m - 2^i) = \Sigma_{i=0,1\ldots,m-1} (2^m - 2^i + 2^i) / (2^m - 2^i)$$

$$= m + \Sigma_{i=1,2\ldots,m} 1 / (2^i - 1) < m + 1 + \Sigma_{i=1,2,\ldots,m-1} 1 / 2^i < m + 2.$$

The associated variance equals:

$$\Sigma_{i=1,2\ldots,m} (1 - p_i) / p_i^2 = \Sigma_{i=0,1\ldots,m-1} 2^{m+i} / (2^m - 2^i)^2 = \Sigma_{i=1,2\ldots,m} 1 / (2^i + 2^{-i} - 2)$$

$$= 2 + \Sigma_{i=2\ldots,m} 1 / (2^i + 2^{-i} - 2) < 2 + \Sigma_{i=1\ldots,m-1} 2^{-i} < 3.$$

The third algorithm uses the first and second algorithms to generate one external encoding key. The inputs to the third algorithm are the block size $n \in \{64, 128\}$ and the number of independently generated functions $T_{K'}^{(i)}$, denoted by t with $t \in \{1, 2, 4, 8\}$ if n = 64 and $t \in \{1, 2, 4, 8, 16\}$ if n =128 (see also clause 5.4). The outputs are t random permutations on 8-bit vectors, s = n/(8t) random invertible binary matrices of dimension 8t × 8t, and a random n-bit vector. These permutations are $T_{K'}^{(i)}$ for i = 1, 2, …, t, the matrices are $A_{K'}^{(1)}$, $A_{K'}^{(2)}$, …, and $A_{K'}^{(s)}$, and the n-bit vector is $b_{K'}$ (see also clauses 5.1 and 5.4).

```
ALGORITHM random-external-encoding(n,t)
    INPUTS:    1) n ∈ {64, 128}
               2) t ∈ {1, 2, 4, 8} if n = 64 and t ∈ {1, 2, 4, 8, 16} if n = 128
    OUTPUTS:   1) t permutations on 8-bit vectors T(1), T(2), …, T(t)
               2) 8t × 8t invertible binary matrices A(1), A(2), …, A(n/(8t))
               3) an n-bit vector b

FOR i = 1 to t DO
    T(i) ← random-permutation()
s ← n/(8t)
FOR i = 1 to s DO
    A(i) ← random-invertible-matrix(8t)
b ← random-m-bit-vector(n)
RETURN (T(1), T(2), …, T(t), A(1), A(2), …, A(s), b)
```

For completeness, the routine calls associated with the different external encoding key sizes are listed in Tables A.1 and A.2 below.

Table A.1: Routine calls for n = 64

Key size	Routine call
20,544 bits	random-external-encoding-key(64,8)
10,304 bits	random-external-encoding-key(64,4)
5,184 bits	random-external-encoding-key(64,2)
2,624 bits	random-external-encoding-key(64,1)

Table A.2: Routine calls for n = 128

Key size	Routine call
49,280 bits	random-external-encoding-key(128,16)
24,704 bits	random-external-encoding-key(128,8)
12,416 bits	random-external-encoding-key(128,4)
6,272 bits	random-external-encoding-key(128,2)
3,200 bits	random-external-encoding-key(128,1)

Annex B (informative): Test data

Python® code that can be used to generate test data for the external encodings is available at:
https://forge.etsi.org/rep/cyber/103718_AES

This code uses the external encoding key generator defined in Annex A and the `random` module of Python® to define the routines `random-integer(i)` and `random-m-bit-vector(m)`. More precisely, as described in Annex A, `random.randint(0,i)` is called to generate a random Python® integer in the range [0, i] and `random.getrandbits(m)` is called to generate a Python® integer with m random bits.

The Python® code is intended to be run at the command line and provides the following help message:

```
$ python encoding.py -h
usage: encoding.py [-h] [-e {input,output}] [-n {64,128}] [-t {1,2,4,8,16}] [-r R] [-p P]

Generate test data sets for the external encodings specified in ETSI TS 103 718

optional arguments:
  -h, --help          show this help message and exit
  -e {input,output}   type of encoding (default = input)
  -n {64,128}         block size (default = 128)
  -t {1,2,4,8,16}     parameter defining the key size (default = 1)
  -r R                number of test data sets (default = 1)
  -p P                seed for initializing the pseudo-random number generator (default = 0)
```

NOTE 1: The parameter t is as defined in clause 5.4. Recall that the combination of n = 64 and t = 16 is invalid. The code will return an error message if this combination is selected.

NOTE 2: Test data generated by the code can be reproduced by selecting the same value of the seed.

A test data set contains the following data:

`Input vector:`	The n-bit input vector of the external encoding.
`Intermediate:`	The n-bit input vector of the function $H_{K'}$ in case of an input encoding and the n-bit output vector of $H_{K'}$ in case of an output encoding.
`Output vector:`	The n-bit output vector of the external encoding.
`External encoding key:`	The functions $T_{K'}^{(i)}$ for i = 1, 2, …, t; $T_{K'}^{(i)}$ is referred to as `T[i]` in a test data set. The matrices $A_{K'}^{(i)}$ for i = 1, 2, …, s; $A_{K'}^{(i)}$ is referred to as `A[i]` in a test data set. The n-bit vector $b_{K'}$, referred to as `b` in a test data set.
`Inverse of T[i]:`	The inverses of the functions $T_{K'}^{(i)}$ for i = 1, 2, …, t; the inverse of $T_{K'}^{(i)}$ is referred to as `Tinv[i]` in a test data set.
`Inverse of A[i]:`	The inverses of the matrices $A_{K'}^{(i)}$, for i = 1, 2, …, s; the inverse of $A_{K'}^{(i)}$ is referred to as `Ainv[i]` in a test data set.

The n-bit input vector of the external encoding and the external encoding key are generated at random for every test data set.

NOTE 3: A test data set of an external encoding is also a test data set for the inverse external encoding since the `Output vector` of the external encoding is the `Input vector` of the inverse external encoding and the `Input vector` of the external encoding is the `Output vector` of the inverse external encoding. Moreover, the `Intermediate` of the external encoding is also the `Intermediate` of its inverse. For completeness, the inverses of the functions $T_{K'}^{(i)}$ for i = 1, 2, …, t and the inverses of the matrices $A_{K'}^{(i)}$ for i = 1, 2, …, s are also included in a test data set since these inverses are used during the computations of the inverse external encoding.

All data in a test data set are displayed using hexadecimal notation:

- An n-bit vector (a_1, a_2, …, a_n), i.e. `Input vector`, `Intermediate`, `Output vector`, or `b` of an `External encoding key` in a test data set, is represented using n/4 hexadecimal characters, and displayed as $h_1\ h_2 \ldots h_{n/4}$ with $h_k = \Sigma_{j=0,1,2,3}\ a_{4k-j}\ 2^j$ for k = 1, 2, …, n/4.

- A function T mapping an 8-bit input vector to an 8-bit output vector, i.e. `T[i]` or `Tinv[i]` in a test data set, is displayed as 16 rows of 16 entries. If $X = (x_1, x_2, \ldots, x_8)$ denotes the input vector of T, then Entry $(1 + \Sigma_{j=0,1,2,3}\ x_{8-j}\ 2^j)$ of Row $(1 + \Sigma_{j=0,1,2,3}\ x_{4-j}\ 2^j)$ is T(X). T(X) is represented using 2 hexadecimal characters: if $T(X) = (y_1, y_2, \ldots, y_8)$, then T(X) is displayed as $h_1\ h_2$ with $h_k = \Sigma_{j=0,1,2,3}\ y_{4k-j}\ 2^j$ for k = 1, 2.

- An m × m matrix A, i.e. `A[i]` or `Ainv[i]` in a test data set, is displayed as m rows, and each row is represented using m/4 hexadecimal characters. If the entries of the i^{th} row with $1 \leq i \leq m$ are $a_{i,1}$, $a_{i,2}$, …, $a_{i,m}$, then the i^{th} row is displayed as $h_{i,1}\ h_{i,2} \ldots h_{i,m/4}$ with $h_{i,k} = \Sigma_{j=0,1,2,3}\ a_{i,4k-j}\ 2^j$ for k = 1, 2, …, m/4.

History

Document history		
V1.1.1	October 2020	Publication