International Standard

ISO/IEC 18033-2

First edition
2006-05-01

AMENDMENT 2
2026-06

Information technology — Security techniques — Encryption algorithms —

Part 2: Asymmetric ciphers

AMENDMENT 2

Sample Document
get full document from standards.iteh.ai

Technologies de l'information — Techniques de sécurité — Algorithmes de chiffrement —

Partie 2: Chiffres asymétriques

AMENDEMENT 2

Reference number
ISO/IEC 18033-2:2006/Amd. 2:2026(en)

© ISO/IEC 2026

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

Sample Document

get full document from standards.iteh.ai

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2026

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Sample Document

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

Get your comment from standards.iteh.ai

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *Information security, cybersecurity and privacy protection*.

A list of all parts in the ISO/IEC 18033 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

Sample Document

get full document from standards.iteh.ai

Information technology — Security techniques — Encryption algorithms —

Part 2:
Asymmetric ciphers

AMENDMENT 2

Replace text in the following clause:

Clause 2 Normative references

Replace

ISO/IEC 10118-3:2004, *IT Security techniques — Hash-functions — Part 3: Dedicated hash-functions*

with

ISO/IEC 10118-3:2018, *IT Security techniques — Hash-functions — Part 3: Dedicated hash-functions*

Sample Document

Replace

ISO/IEC 18033-3:2005, *Information technology — Security techniques — Encryption algorithms — Part 3: Block ciphers*

get full document from standards.iteh.ai

with

ISO/IEC 18033-3:2010, *Information technology — Security techniques — Encryption algorithms — Part 3: Block ciphers*

Clause 8 Generic hybrid ciphers

Replace the first four bullet points in 8.1.2 with the following bullet points:

— ECIES-KEM (described in 10.2),

— PSEC-KEM (described in 10.3),

— ACE-KEM (described in 10.4),

— FACE-KEM (described in 10.5),

— RSA-KEM (described in 11.5),

— Classic McEliece KEM (described in Clause 13),

— FrodoKEM (described in Clause 14), and

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

— ML-KEM (described in Clause 15).

After Clause 12

Add the following new clauses after Clause 12:

13 The Classic McEliece KEM

13.1 General

This clause defines the Classic McEliece KEM. This KEM consists of three mathematical functions, namely CM.KeyGen, CM.Encap, and CM.Decap, for each of the "selected parameter sets" listed in 13.16. CM.Encrypt is a synonym for CM.Encap, and CM.Decrypt is a synonym for CM.Decap.

A broader parameter space for Classic McEliece is specified in 13.3. For each parameter set in that parameter space, subsequent subclauses define

— exactly which public key and private key are output by CM.KeyGen given random bits;

— exactly which ciphertext and session key are output by CM.Encap given a public key and random bits; and

— exactly which session key is output by CM.Decap given a ciphertext and a private key.

These subclauses define each mathematical function F by presenting an algorithm to compute F. Basic algorithms such as Gaussian elimination are not repeated here, but CM.MatGen, CM.Encode, CM.Decode, CM.Irreducible , CM.FieldOrdering , CM.SeededKeyGen , CM.FixedWeight , CM.KeyGen , CM.Encap , and CM.Decap are specified as numbered lists of steps. See Annex D for a reference to a guide for implementors.

Stating that an algorithm A computes a mathematical function F means the following: the domain of F is the set of pairs (I, R) such that I is the input consumed by a run of A and R is the string of random bits generated by that run of A; $F(I, R)$ is the output of that run of A, or the symbol ∞ if that run of A does not halt.

EXAMPLE 1 The CM.KeyGen algorithm reads exactly ℓ random bits, so the domain of the mathematical function CM.KeyGen is the set of ℓ-bit strings. Here ℓ, one of the Classic McEliece parameters, is 256 for each of the selected parameter sets.

EXAMPLE 2 The algorithm CM.FixedWeight is a rejection-sampling algorithm, and the number of random bits it uses can vary from one run of the algorithm to another. The pairs (I, R) in the domain of the function computed by this algorithm have exactly the same variations in the lengths of R. The algorithm halts with probability 1 for independent uniformly distributed random bits. In the non-halting cases, R has infinite length.

NOTE 1 If an algorithm A is deterministic, i.e., does not consume random bits, then the string R of random bits generated by a run of A is the empty string.

NOTE 2 Any change in the observed behaviour of an algorithm, where the observations consist of the input, the random bits used, and the output, is a change in the mathematical function computed by the algorithm. For example, if algorithms A and B are almost identical but B generates and discards some extra bits beyond what A uses, then the string of random bits used is longer for B than for A, so A and B are not computing the same mathematical function.

Clause 13 consistently uses indices numbered from 0, including row indices, column indices, and α indices.

NOTE 3 Conventions in the mathematical literature sometimes number indices from 0, but sometimes do not: for example, polynomial exponents are conventionally numbered from 0, while most vectors not related to

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

polynomial exponents are conventionally numbered from 1.

Throughout Clause 13, $\mathbb{F}_r$ means a finite field with r elements, where r is a power of 2.

Elements of $\mathbb{F}_2^n$, such as codewords and error vectors, are treated as column vectors.

NOTE 4 This convention avoids all transpositions. This differs from a common convention in coding theory, namely to write codewords as row vectors but to transpose the codewords for applying parity checks.

13.2 Requirements

To claim **conformance** to this document regarding Classic McEliece, an algorithm shall (1) name either CM.KeyGen or CM.Encap or CM.Decap; (2) identify a parameter set listed in 13.16 (not another parameter set from 13.3); and (3) compute exactly the corresponding mathematical function defined in this document for that parameter set.

For example, a CM.KeyGen implementation claimed to conform to this document for the mceliece6960119 parameter set shall compute the specified CM.KeyGen function for that parameter set: i.e., the implementation shall read exactly $\ell = 256$ bits of randomness, and shall produce the same output that the CM.KeyGen algorithm specified below produces given the same 256-bit string.

Conformance to this document for a tuple of three Classic McEliece algorithms, one for each of CM.KeyGen and CM.Encap and CM.Decap, is defined as conformance to this document for each algorithm, and again shall identify a parameter set listed in 13.16.

Conformant implementations shall use the encodings specified in Clause 13.

Sample Document

NOTE 1 Clause 13 does not allow the format flexibility described in 7.3.

NOTE 2 Users sometimes place further constraints on algorithms, for example to include various side-channel countermeasures (which could use their own random bits) or to achieve particular levels of performance. Such constraints are out of scope here. Clause 13 defines the mathematical functions to be computed; it does not constrain how these functions are computed.

13.3 Parameters

The *CM parameters* are implicit inputs to the CM algorithms defined below. A CM parameter set specifies the following:

— A positive integer m. This also defines a parameter $q = 2^m$.

— A positive integer n with $n \leq q$.

— A positive integer $t \geq 2$ with $mt < n$. This also defines a parameter $k = n - mt$.

— A monic irreducible polynomial $f(z) \in \mathbb{F}_2[z]$ of degree m. This defines a representation $\mathbb{F}_2[z]/f(z)$ of the field $\mathbb{F}_q$.

— A monic irreducible polynomial $F(y) \in \mathbb{F}_q[y]$ of degree t. This defines a representation $\mathbb{F}_q[y]/F(y)$ of the field $\mathbb{F}_{q^t} = \mathbb{F}_{2^{mt}}$.

— Integers $\nu \geq \mu \geq 0$ with $\nu \leq k + \mu$. Parameter sets that do not mention these parameters define them as (0,0) by default.

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

— The symmetric-cryptography parameters, which are the following:

  — A positive integer ℓ.

  — A cryptographic hash function Hash that outputs ℓ bits.

  — An integer $\sigma_1 \geq m$.

  — An integer $\sigma_2 \geq 2m$.

  — A pseudorandom bit generator PRG mapping a string of ℓ bits to a string of $n + \sigma_2 q + \sigma_1 t + \ell$ bits.

— Each parameter set is also labeled as either a pc parameter set or a non-pc parameter set.

NOTE pc is also referred to in the literature as "plaintext confirmation".

13.4 Matrix reduction

13.4.1 Reduced row-echelon form

Saying that a matrix R is in "reduced row-echelon form" means that there is a sequence of column indices $c_0 < c_1 < \cdots < c_{r-1}$ such that:

— row 0 of R begins with a 1 in column c_0, and this is the only nonzero entry in column c_0;

— row 1 of R begins with a 1 in column c_1, the only nonzero entry in column c_1;

— row 2 of R begins with a 1 in column c_2, the only nonzero entry in column c_2;

— etc;

— row $r-1$ of R begins with a 1 in column c_{r-1}, the only nonzero entry in column c_{r-1}; and

— all subsequent rows of R are 0.

NOTE 1 The rank of R is r.

NOTE 2 Gaussian elimination is a well-known algorithm that, given a matrix X, computes the unique matrix R in reduced row-echelon form having the same number of rows as X and the same row space as X.

13.4.2 Systematic form

As a special case of reduced row-echelon form, saying that a matrix R is in "systematic form" means that

— R has exactly r rows, i.e., there are no zero rows; and

— $c_i = i$ for $0 \leq i < r$.

NOTE 1 The second condition is equivalent to saying $c_{r-1} = r - 1$, except in the degenerate case $r = 0$.

NOTE 2 R is in systematic form if and only if R has the form $(I_r|T)$, where I_r is an $r \times r$ identity matrix.

"Row-reducing a matrix X to systematic form" means computing the unique systematic-form matrix having the same row space as X, if such a matrix exists.

Sample Document
get full document from standards.iteh.ai

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

See Annex D for a reference to a guide for implementors.

13.4.3 Semi-systematic form

The following generalization of the concept of systematic form uses two integer parameters μ, ν satisfying $\nu \geq \mu \geq 0$.

Let R be a rank-r matrix in reduced row-echelon form. Assume that $r \geq \mu$, and that there are at least $r - \mu + \nu$ columns.

We say that R is in "(μ, ν)-semi-systematic form" if R has r rows (i.e. no zero rows); $c_i = i$ for $0 \leq i < r - \mu$; and $c_i \leq i - \mu + \nu$ for $0 \leq i < r$.

NOTE 1 The c_i conditions are equivalent to $c_{r-\mu-1} = r - \mu - 1$ and $c_{r-1} \leq r - \mu + \nu - 1$ except in the degenerate case $r = \mu$.

NOTE 2 As a special case, (μ, ν)-semi-systematic form is equivalent to systematic form if $\mu = \nu$. However, if $\nu > \mu > 0$ then (μ, ν)-semi-systematic form allows more matrices than systematic form.

"Row-reducing a matrix X to (μ, ν)-semi-systematic form" means computing the unique (μ, ν)-semi-systematic-form matrix having the same row space as X, if such a matrix exists.

This specification gives various definitions first for the simpler case $(\mu, \nu) = (0,0)$ and then for the general case. The list of selected parameter sets provides, for each key size, one parameter set with $(\mu, \nu) = (0,0)$, and one parameter set labeled "f" with $(\mu, \nu) = (32,64)$.

Sample Document

See Annex D for a guide for implementors.

get full document from standards.iteh.ai

13.5 Matrix generation for Goppa codes

13.5.1 General

The following algorithm CM.MatGen takes as input $\Gamma = (g, \alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ where

— g is a monic irreducible polynomial in $\mathbb{F}_q[x]$ of degree t and

— $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ are distinct elements of $\mathbb{F}_q$.

The output CM.MatGen(Γ) is defined first in the simpler case of systematic form, and then in the general case of semi-systematic form. The output is either $\perp$ or a $(\mu + 2)$-tuple $(T, \ldots)$, where T is the *CM public key*, an $mt \times k$ matrix over $\mathbb{F}_2$.

13.5.2 Systematic form

For $(\mu, \nu) = (0,0)$, the output CM.MatGen(Γ) is either $\perp$ or of the form (T, Γ), where T is an $mt \times k$ matrix over $\mathbb{F}_2$. The algorithm is as follows:

1. Compute the $t \times n$ matrix $M = \{h_{i,j}\}$ over $\mathbb{F}_q$, where $h_{i,j} = \alpha_j^i / g(\alpha_j)$ for $i = 0, \ldots, t-1$ and $j = 0, \ldots, n-1$.

2. Form an $mt \times n$ matrix N over $\mathbb{F}_2$ by replacing each entry $u_0 + u_1 z + \cdots + u_{m-1} z^{m-1}$ of M with a column of m bits $u_0, u_1, \ldots, u_{m-1}$.

3. Row-reduce N to systematic form $(I_{mt}|T)$, where I_{mt} is the $mt \times mt$ identity matrix over $\mathbb{F}_2$. If this fails, return $\perp$.

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

4. Return (T, Γ).

13.5.3 Semi-systematic form

For general (μ, ν), the output CM.MatGen(Γ) is either $\perp$ or a $(\mu+2)$-tuple of the form $\left(T, c_{mt-\mu}, \ldots, c_{mt-1}, \Gamma'\right)$, where

— T is an $mt \times k$ matrix over $\mathbb{F}_2$;

— $c_{mt-\mu}, \ldots, c_{mt-1}$ are integers with $mt-\mu \leq c_{mt-\mu} < c_{mt-\mu+1} < \cdots < c_{mt-1} < mt-\mu+\nu$;

— $\Gamma' = (g, \alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1})$;

— g is the same as in the input; and

— $\alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1}$ are distinct elements of $\mathbb{F}_q$.

The algorithm is as follows:

1. Compute the $t \times n$ matrix $M = \{h_{i,j}\}$ over $\mathbb{F}_q$, where $h_{i,j} = \alpha_j^i / g(\alpha_j)$ for $i = 0, \ldots, t-1$ and $j = 0, \ldots, n-1$.

2. Form an $mt \times n$ matrix N over $\mathbb{F}_2$ by replacing each entry $u_0 + u_1 z + \cdots + u_{m-1} z^{m-1}$ of M with a column of m bits $u_0, u_1, \ldots, u_{m-1}$.

3. Row-reduce N to (μ, ν)-semi-systematic form, obtaining a matrix H; if this fails, return $\perp$. Define c_i such that row i has its leading 1 in column c_i. (By definition of semi-systematic form, $c_i = i$ for $0 \leq i < mt-\mu$; and $mt-\mu \leq c_{mt-\mu} < c_{mt-\mu+1} < \cdots < c_{mt-1} < mt-\mu+\nu$. The matrix H is a variable that can change later.)

4. Set $(\alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1}) \leftarrow (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$. (Each α'_i is a variable that can change later.)

5. For $i = mt-\mu$, then $i = mt-\mu+1$, and so on through $i = mt-1$, in this order: swap column i with column c_i in H, while swapping α'_i with α'_{c_i}. (After the swap, row i has its leading 1 in column i. The swap does nothing if $c_i = i$.)

6. The matrix H now has systematic form $(I_{mt}|T)$, where I_{mt} is the $mt \times mt$ identity matrix over $\mathbb{F}_2$. Return $\left(T, c_{mt-\mu}, \ldots, c_{mt-1}, \Gamma'\right)$ where $\Gamma' = (g, \alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1})$.

   NOTE In the special case $(\mu, \nu) = (0,0)$, the $c_{mt-\mu}, \ldots, c_{mt-1}$ portion of the output is empty, and the i loop is empty, so Γ' is guaranteed to be the same as Γ. The reduction to (0,0)-semi-systematic form is exactly reduction to systematic form. The general algorithm definition thus matches the (0,0) algorithm definition.

13.6 Encoding subroutine

The following algorithm CM.Encode takes two inputs: a column vector $e \in \mathbb{F}_2^n$ of Hamming weight t and a public key T, i.e. an $mt \times k$ matrix over $\mathbb{F}_2$. The algorithm output CM.Encode(e, T) is a vector $C \in \mathbb{F}_2^{mt}$. The algorithm is as follows:

1. Define $H = (I_{mt}|T)$.

2. Compute and return $C = He \in \mathbb{F}_2^{mt}$.

Sample Document
get full document from standards.iteh.ai

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

13.7 Decoding subroutine

The following algorithm CM.Decode takes two inputs: a vector $C \in \mathbb{F}_2^{mt}$; and Γ', the last component of CM.MatGen(Γ) for some Γ such that CM.MatGen(Γ) $\neq \perp$. Write T for the first component of CM.MatGen(Γ). By definition of CM.MatGen,

— T is an $mt \times k$ matrix over $\mathbb{F}_2$;

— Γ' has the form $(g, \alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1})$;

— g is a monic irreducible polynomial in $\mathbb{F}_q[x]$ of degree t; and

— $\alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1}$ are distinct elements of $\mathbb{F}_q$.

There are two possibilities for CM.Decode(C, Γ'):

— If C = CM.Encode(e, T) then CM.Decode$(C, \Gamma') = e$. In other words, if there exists a weight-t vector $e \in \mathbb{F}_2^n$ such that $C = He$, where H is defined as $(I_{mt}|T)$, then CM.Decode$(C, \Gamma') = e$.

— If C does not have the form He for any weight-t vector $e \in \mathbb{F}_2^n$, then CM.Decode$(C, \Gamma') = \perp$.

The algorithm is as follows:

1. Extend C to $v = (C, 0, \ldots, 0) \in \mathbb{F}_2^n$ by appending k zeros.
2. Find the unique $c \in \mathbb{F}_2^n$ such that (1) $Hc = 0$ and (2) c has Hamming distance $\leq t$ from v. If there is no such c, return $\perp$. (See Annex D for a guide for implementors.)
3. Set $e = v + c$.
4. If wt$(e) = t$ and $C = He$, return e. Otherwise return $\perp$.

Sample Document
get full document from standards.iteh.ai

13.8 Irreducible-polynomial generation

The following algorithm CM.Irreducible takes a string of $\sigma_1 t$ input bits $d_0, d_1, \ldots, d_{\sigma_1 t - 1}$. It outputs either $\perp$ or a monic irreducible degree-t polynomial $g \in \mathbb{F}_q[x]$. The algorithm is as follows:

1. Define $\beta_j = \sum_{i=0}^{m-1} d_{\sigma_1 j + i} z^i$ for each $j \in \{0,1, \ldots, t-1\}$. (Within each group of σ_1 input bits, this uses only the first m bits. The algorithm ignores the remaining bits.)
2. Define $\beta = \beta_0 + \beta_1 y + \cdots + \beta_{t-1} y^{t-1} \in \mathbb{F}_q[y]/F(y)$.
3. Compute the minimal polynomial g of β over $\mathbb{F}_q$. (By definition g is monic and irreducible, and $g(\beta) = 0$. See Annex D for a guide for implementors.)
4. Return g if g has degree t. Otherwise return $\perp$.

13.9 Field-ordering generation

The following algorithm CM.FieldOrdering takes a string of $\sigma_2 q$ input bits. It outputs either $\perp$ or a sequence $(\alpha_0, \alpha_1, \ldots, \alpha_{q-1})$ of q distinct elements of $\mathbb{F}_q$. The algorithm is as follows:

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

1. Take the first σ_2 input bits $b_0, b_1, \ldots, b_{\sigma_2-1}$ as a σ_2-bit integer $a_0 = b_0 + 2b_1 + \cdots + 2^{\sigma_2-1}b_{\sigma_2-1}$, take the next σ_2 bits as a σ_2-bit integer a_1, and so on through a_{q-1}.

2. If $a_0, a_1, \ldots, a_{q-1}$ are not distinct, return $\perp$.

3. Sort the pairs (a_i, i) in lexicographic order to obtain pairs $\left(a_{\pi(i)}, \pi(i)\right)$ where π is a permutation of $\{0,1, \ldots, q-1\}$.

4. Define $\alpha_i = \sum_{j=0}^{m-1} \pi(i)_j \cdot z^{m-1-j}$ where $\pi(i)_j$ denotes the jth least significant bit of $\pi(i)$. (Recall that the finite field $\mathbb{F}_q$ is constructed as $\mathbb{F}_2[z]/f(z)$.)

5. Output $\left(\alpha_0, \alpha_1, \ldots, \alpha_{q-1}\right)$.

13.10 Key generation

13.10.1 CM.KeyGen

The following randomized algorithm CM.KeyGen takes no input (beyond the parameters). It outputs a public key and private key. The algorithm is as follows, using a subroutine CM.SeededKeyGen defined below:

1. Generate a uniform random ℓ-bit string δ. (This is called a *seed*.)

2. Output CM.SeededKeyGen(δ).

Sample Document

13.10.2 CM.SeededKeyGen

get full document from standards.iteh.ai

The following algorithm CM.SeededKeyGen takes an ℓ-bit input δ. It outputs a public key and private key. The algorithm is as follows:

1. Compute $E = \mathrm{PRG}(\delta)$, a string of $n + \sigma_2 q + \sigma_1 t + \ell$ bits.

2. Define δ' as the last ℓ bits of E.

3. Define s as the first n bits of E.

4. Compute $\alpha_0, \ldots, \alpha_{q-1}$ from the next $\sigma_2 q$ bits of E by the CM.FieldOrdering algorithm. If this fails, set $\delta \leftarrow \delta'$ and restart the algorithm.

5. Compute g from the next $\sigma_1 t$ bits of E by the CM.Irreducible algorithm. If this fails, set $\delta \leftarrow \delta'$ and restart the algorithm.

6. Define $\Gamma = (g, \alpha_0, \alpha_1, \ldots, \alpha_{n-1})$. (Note that $\alpha_n, \ldots, \alpha_{q-1}$ are not used in Γ.)

7. Compute $\left(T, c_{mt-\mu}, \ldots, c_{mt-1}, \Gamma'\right) \leftarrow \mathrm{CM.MatGen}(\Gamma)$. If this fails, set $\delta \leftarrow \delta'$ and restart the algorithm.

8. Write Γ' as $(g, \alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1})$.

9. Output T as public key and $(\delta, c, g, \alpha, s)$ as private key, where $c = \left(c_{mt-\mu}, \ldots, c_{mt-1}\right)$ and $\alpha = \left(\alpha'_0, \ldots, \alpha'_{n-1}, \alpha_n, \ldots, \alpha_{q-1}\right)$.

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

13.11 Fixed-weight-vector generation

The following randomized algorithm CM.FixedWeight takes no input. It outputs a vector $e \in \mathbb{F}_2^n$ of weight t. The algorithm uses a precomputed integer $\tau \geq t$ defined below. The algorithm is as follows:

1. Generate $\sigma_1\tau$ uniform random bits $b_0, b_1, \ldots, b_{\sigma_1\tau-1}$.
2. Define $d_j = \sum_{i=0}^{m-1} b_{\sigma_1 j+i} 2^i$ for each $j \in \{0,1,\ldots,\tau-1\}$. (Within each group of σ_1 random bits, this uses only the first m bits. The algorithm ignores the remaining bits.)
3. Define $a_0, a_1, \ldots, a_{t-1}$ as the first t entries in $d_0, d_1, \ldots, d_{\tau-1}$ in the range $\{0,1,\ldots,n-1\}$. If there are fewer than t entries in $d_0, d_1, \ldots, d_{\tau-1}$ in the range $\{0,1,\ldots,n-1\}$, restart the algorithm.
4. If $a_0, a_1, \ldots, a_{t-1}$ are not all distinct, restart the algorithm.
5. Define $e = (e_0, e_1, \ldots, e_{n-1}) \in \mathbb{F}_2^n$ as the weight-t vector such that $e_{a_i} = 1$ for each $i \in 0,1,\ldots,t-1$ and all other entries of e are 0.
6. Return e.

The integer τ is defined as t if $n = q$; as $2t$ if $q/2 \leq n < q$; as $4t$ if $q/4 \leq n < q/2$; etc.

NOTE All of the selected parameter sets have $q/2 \leq n \leq q$, so $\tau \in \{t, 2t\}$.

13.12 Encapsulation

Sample Document

13.12.1 General

The randomized algorithm CM.Encap takes as input a public key T. It outputs a ciphertext C and a session key K.

get full document from standards.iteh.ai

13.12.2 CM.Encap for non-pc parameters

For non-pc parameter sets, the CM.Encap algorithm is as follows:

1. Use CM.FixedWeight to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C = \text{CM.Encode}(e, T)$.
3. Compute $K = \text{Hash}(1, e, C)$; see 13.15.5 for Hash input encodings.
4. Output ciphertext C and session key K.

13.12.3 CM.Encap for pc parameters

For pc parameter sets, the CM.Encap algorithm is as follows:

1. Use CM.FixedWeight to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \text{CM.Encode}(e, T)$.
3. Compute $C_1 = \text{Hash}(2, e)$. Put $C = (C_0, C_1)$; see 13.15.6 for Hash input encodings.
4. Compute $K = \text{Hash}(1, e, C)$.

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

5. Output ciphertext C and session key K.

13.13 Decapsulation

13.13.1 General

The algorithm CM.Decap takes as input a ciphertext C and a private key, and outputs a session key K.

13.13.2 CM.Decap for non-pc parameters

For non-pc parameter sets, the algorithm CM.Decap is as follows:

1. Set $b \leftarrow 1$.
2. Extract $s \in \mathbb{F}_2^n$ and $\Gamma' = (g, \alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1})$ from the private key.
3. Compute $e \leftarrow \text{CM.Decode}(C, \Gamma')$. If $e = \perp$, set $e \leftarrow s$ and $b \leftarrow 0$.
4. Compute $K = \text{Hash}(b, e, C)$; see 13.15.5 for Hash input encodings.
5. Output K as session key.

13.13.3 CM.Decap for pc parameters

For pc parameter sets, the algorithm CM.Decap is as follows:

Sample Document

1. Split the ciphertext C as (C_0, C_1) with $C_0 \in \mathbb{F}_2^{mt}$ and $C_1 \in \mathbb{F}_2^{\ell}$.
2. Set $b \leftarrow 1$.

get full document from standards.iteh.ai

3. Extract $s \in \mathbb{F}_2^n$ and $\Gamma' = (g, \alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1})$ from the private key.
4. Compute $e \leftarrow \text{CM.Decode}(C_0, \Gamma')$. If $e = \perp$, set $e \leftarrow s$ and $b \leftarrow 0$.
5. Compute $C'_1 = \text{Hash}(2, e)$; see 13.15.6 for Hash input encodings.
6. If $C'_1 \neq C_1$, set $e \leftarrow s$ and $b \leftarrow 0$.
7. Compute $K = \text{Hash}(b, e, C)$; see 13.15.6 for Hash input encodings.
8. Output K as session key.

13.14 Choices of symmetric-cryptography parameters

In this subclause and 13.15, Uint8 means the set $\{0,1,\ldots,255\}$; a Uint8 string means a string of elements of Uint8; a b-Uint8 string means a string of b elements of Uint8, i.e. a Uint8 string of length b; b Uint8s mean b elements of Uint8. 13.15 specifies how various objects are represented as Uint8 strings.

NOTE 1 The usage of Uint8 in this subclause and 13.15 is designed to fit Classic McEliece into the conventional interface used in cryptographic software, in which private keys, public keys, ciphertexts, and session keys are Uint8 strings (often expressed in programming languages as arrays of "uint8_t" or arrays of "unsigned char").

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026

In this subclause, SHAKE256(x, i) takes as input a Uint8 string x and an integer i, which is a multiple of 8, and produces an $(i/8)$-Uint8 string. SHAKE256 shall be implemented in accordance with ISO/IEC 10118-3:2018, C.2. Specifically, it is the Uint8-string ("sequence of bytes") function defined in that annex, not the bit-string function.

NOTE 2 The most common interface to SHAKE256 software is that the inputs are a Uint8 string and $i/8$, and the output is an $(i/8)$-Uint8 string. However, in specifications, it is more common to define a SHAKE256 function that takes i rather than $i/8$ as an input. This difference in SHAKE256 notation can be confusing.

NOTE 3 13.15 specifies how Clause 13 represents i-bit strings as $(i/8)$-Uint8 strings. That representation covers all $(i/8)$-Uint8 strings, allowing those strings to be viewed as i-bit strings without further comment.

All of the selected parameter sets use the following symmetric-cryptography parameters.

— The integer ℓ is 256.

— The ℓ-bit string Hash(x) is defined as SHAKE256$(x,\ \ell)$.

— The integer σ_1 is 16. (All of the selected parameter sets have $m \leq 16$, so $\sigma_1 \geq m$.)

— The integer σ_2 is 32.

— The $(n + \sigma_2 q + \sigma_1 t + \ell)$-bit string PRG$(\delta)$ is defined as SHAKE256$\big((64, \delta), n + \sigma_2 q + \sigma_1 t + \ell\big)$. Here $64, \delta$ means the 33-Uint8 string that begins with the element 64 of Uint8 and continues with δ.

NOTE 4 For each hash input used in Classic McEliece, the first Uint8 entry of the input is 0 or 1 (or 2 for pc) (see 13.15). Thus, the hash inputs do not overlap the SHAKE256 inputs used in PRG.

Sample Document

13.15 Representation of objects as Uint8 strings

get full document from standards.iteh.ai

13.15.1 Bit vectors

If r is a multiple of 8 then an r-bit vector $v = (v_0, v_1, \ldots, v_{r-1}) \in \mathbb{F}_2^r$ is represented as the following $(r/8)$-Uint8 string:

$$(v_0 + 2v_1 + 4v_2 + \cdots + 128v_7, v_8 + 2v_9 + 4v_{10} + \cdots + 128v_{15}, \ldots, v_{r-8} + 2v_{r-7} + 4v_{r-6} + \cdots + 128v_{r-1}).$$

If r is not a multiple of 8 then an r-bit vector $v = (v_0, v_1, \ldots, v_{r-1}) \in \mathbb{F}_2^r$ is zero-padded on the right to length between $r + 1$ and $r + 7$, whichever is a multiple of 8, and then represented as above.

Here are two definitions: "Simply Decoded Classic McEliece" ignores padding bits on input, while "Narrowly Decoded Classic McEliece" rejects inputs (ciphertexts and public keys) where padding bits are nonzero; rejection means returning $\perp$. For some parameter sets (but not all), r is always a multiple of 8, so there are no padding bits, so Simply Decoded Classic McEliece and Narrowly Decoded Classic McEliece are identical.

The definitions of Simply Decoded and Narrowly Decoded are provided for convenience in discussions of situations where the distinction is potentially relevant. Applications should avoid relying on whether non-zero padding bits are always allowed, always rejected, or some intermediate option. Conformance to this document for Classic McEliece does not require a Simply Decoded or Narrowly Decoded label.

13.15.2 Session keys

A session key K is an element of $\mathbb{F}_2^\ell$. It is represented as a $\lceil \ell/8 \rceil$-Uint8 string.

© ISO/IEC 2026 – All rights reserved

ISO/IEC 18033-2:2006/Amd 2:2026 • https://standards.iteh.ai/catalog/standards/iso/6a9f6cc8-516f-4ce4-a220-6543fc525b82/iso-iec-18033-2-2006-amd-2-2026