



DPC: 25/30474081 DC

BSI Group Headquarters

389 Chiswick High Road London W4 4AL

Tel: +44 (0)20 8996 9000

Fax: +44 (0)20 8996 7400

www.bsigroup.com

Date: 19 February 2025

Origin: European

Latest date for receipt of comments: 14 April 2025

Project No. 2023/01024

Responsible committee: IST/33/2 Cryptography and Security Mechanisms

Interested committees:

Title: Draft BS ISO/IEC 18033-2 :2006/Amd 2 Information technology - Security techniques - Encryption algorithms. Part 2: Asymmetric ciphers. Amendment 2.

Please notify the secretary if you are aware of any keywords that might assist in classifying or identifying the standard or if the content of this standard

- i) has any issues related to 3rd party IPR, patent or copyright
- ii) affects other national standard(s)
- iii) requires additional national guidance or information

WARNING: THIS IS A DRAFT AND MUST NOT BE REGARDED OR USED AS A BRITISH STANDARD. THIS DRAFT IS NOT CURRENT BEYOND 14 April 2025

This draft is issued to allow comments from interested parties; all comments will be given consideration prior to publication. No acknowledgement will normally be sent. **See overleaf for information on the submission of comments.**

No copying is allowed, in any form, without prior written permission from BSI except as permitted under the Copyright, Designs and Patent Act 1988 or for circulation within a nominating organization for briefing purposes. Electronic circulation is limited to dissemination by e-mail within such an organization by committee members.

Further copies of this draft may be purchased from BSI Shop <http://shop.bsigroup.com> or from BSI Customer Services, Tel: +44(0) 20 8996 9001 or email cservices@bsigroup.com. British, International and foreign standards are also available from BSI Customer Services.

Information on the co-operating organizations represented on the committees referenced above may be obtained from <https://standardsdevelopment.bsigroup.com/>.

Responsible Committee Secretary: Ms Joy Tan (BSI)

Direct tel:

E-mail: joy.tan@bsigroup.com

Full document details: 25/30474081 DC

Introduction

This draft standard is based on national and international discussions. Your comments on this draft are invited and will assist in the preparation of the consequent standard.

For international standards, comments will be reviewed by the relevant UK national committee before sending the consensus UK vote and comments to the international committee, which will then decide appropriate action. If the international standard is approved, it is usual for the text to be published as a British Standard.

For national standards, comments will be reviewed by the relevant UK national committee and the resulting standards published as a British Standard.

UK Vote

Please indicate whether you consider the UK should submit a negative (with supporting technical reasons) or positive vote on this draft. Please indicate if you are aware of any reason why this draft standard should not be published as a British Standard.

Submission of Comments

- **Annotated drafts are not acceptable and will be rejected.**
- All comments should be submitted online at <https://standardsdevelopment.bsigroup.com/>. You will need to register in order to comment.

Template for comments and secretariat observations

Date: xx/xx/20xx	Document: ISO/DIS xxxx
------------------	------------------------

1	2	(3)	4	5	(6)	7
MB	Clause No./ Subclause No./Annex (e.g. 3.1)	Paragraph/ Figure/Table/Note	Type of comment	Comment (justification for change) by the MB	Proposed change by the MB	Secretariat observations on each comment submitted
	3.1	Definition 1	ed	Definition is ambiguous and needs clarifying.	Amend to read '...so that the mains connector to which no connection...'	
	6.4	Paragraph 2	te	The use of the UV photometer as an alternative cannot be supported as serious problems have been encountered in its use in the UK.	Delete reference to UV photometer.	



DRAFT Amendment

ISO/IEC 18033-2:2006/ DAM 2

Information technology — Security techniques — Encryption algorithms —

Part 2: Asymmetric ciphers

AMENDMENT 2

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

Partie 2: Chiffres asymétriques

AMENDEMENT 2

ICS: 35.030

ISO/IEC JTC 1/SC 27

Secretariat: **DIN**

Voting begins on:
2025-02-19

Voting terminates on:
2025-05-14

This document is circulated as received from the committee secretariat.

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENTS AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Reference number
ISO/IEC 18033-2:2006/DAM 2:2025(en)

Full document details: 25/30474081 DC

© ISO/IEC 2025  **Documents**
Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2025

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

Information technology — Security techniques — Encryption algorithms —

Part 2: Asymmetric ciphers

AMENDMENT 2

Replace text in the following clause:

2 Normative references

Replace

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

by

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

Replace

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

by

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

8 Generic hybrid ciphers

Replace the following in Clause 8.1.2:

Replace

- ECIES-KEM (described in 10.2),
- PSEC-KEM (described in 10.3),
- ACE-KEM (described in 10.4),
- FACE-KEM (described in 10.5), and
- RSA-KEM (described in 11.5).

by

- 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
- ML-KEM (described in Clause 15).

Add the following clause:

13 The Classic McEliece KEM

13.1 General

Clause 13 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 Clause 13.16. `CM.Encrypt` is a synonym for `CM.Encap`, and `CM.Decrypt` is a synonym for `CM.Decap`.

The definitions for each selected parameter set are unified into a single definition for a broader parameter space specified in Clause 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 below 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 ; and $F(I, R)$ 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 Beware that conventions in the mathematical literature sometimes agree with this but sometimes do not: for example, polynomial exponents are conventionally numbered from 0, while most vectors not related to 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. Beware that 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 Clause 13.16 (not another parameter set from Clause 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 Clause 13.16.

Conformant implementations shall use the encodings specified in Clause 13.

NOTE 1 Clause 13 does not allow the format flexibility described in Clause 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 $v \geq \mu \geq 0$ with $v \leq k + \mu$. Parameter sets that do not mention these parameters define them as $(0,0)$ by default.

- 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 $c_0 < c_1 < \dots < c_{r-1}$ 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.

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 $(\mu, \nu) = (32, 64)$.

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

13.5 Matrix generation for Goppa codes

13.5.1 General

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

- g is a monic irreducible polynomial in $\mathbb{F}_q[x]$ of degree t and
- $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$ are distinct elements of $\mathbb{F}_q$.

The algorithm 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 $(T, \dots)$, 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 algorithm 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 $i = 0, \dots, t-1$ and $j = 0, \dots, n-1$.
2. Form an $mt \times n$ matrix N over $\mathbb{F}_2$ by replacing each entry $u_0 + u_1z + \dots + u_{m-1}z^{m-1}$ of M with a column of m bits $u_0, u_1, \dots, u_{m-1}$.
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 $\perp$.
4. Return (T, Γ) .

13.5.3 Semi-systematic form

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

- T is an $mt \times k$ matrix over $\mathbb{F}_2$;
- $c_{mt-\mu}, \dots, c_{mt-1}$ are integers with $mt - \mu \leq c_{mt-\mu} < c_{mt-\mu+1} < \dots < c_{mt-1} < mt - \mu + \nu$;
- $\Gamma' = (g, \alpha'_0, \alpha'_1, \dots, \alpha'_{n-1})$;
- g is the same as in the input; and

— $\alpha'_0, \alpha'_1, \dots, \alpha'_{n-1}$ 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 $i = 0, \dots, t-1$ and $j = 0, \dots, n-1$.
2. Form an $mt \times n$ matrix N over $\mathbb{F}_2$ by replacing each entry $u_0 + u_1z + \dots + u_{m-1}z^{m-1}$ of M with a column of m bits $u_0, u_1, \dots, u_{m-1}$.
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 $mt - \mu \leq c_{mt-\mu} < c_{mt-\mu+1} < \dots < c_{mt-1} < mt - \mu + \nu$. The matrix H is a variable that can change later.)
4. Set $(\alpha'_0, \alpha'_1, \dots, \alpha'_{n-1}) \leftarrow (\alpha_0, \alpha_1, \dots, \alpha_{n-1})$. (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 $(T, c_{mt-\mu}, \dots, c_{mt-1}, \Gamma')$ where $\Gamma' = (g, \alpha'_0, \alpha'_1, \dots, \alpha'_{n-1})$.

NOTE In the special case $(\mu, \nu) = (0, 0)$, the $c_{mt-\mu}, \dots, c_{mt-1}$ 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 weight- t column vector $e \in \mathbb{F}_2^n$; 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 $C = He \in \mathbb{F}_2^{mt}$.

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 $(g, \alpha'_0, \alpha'_1, \dots, \alpha'_{n-1})$;
- g is a monic irreducible polynomial in $\mathbb{F}_q[x]$ of degree t ; and
- $\alpha'_0, \alpha'_1, \dots, \alpha'_{n-1}$ are distinct elements of $\mathbb{F}_q$.

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

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

The algorithm is as follows:

1. Extend C to $v = (C, 0, \dots, 0) \in \mathbb{F}_2^n$ 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 reference to a guide for implementors.)
3. Set $e = v + c$.
4. If $\text{wt}(e) = t$ and $C = He$, return e . Otherwise return $\perp$.

13.8 Irreducible-polynomial generation

The following algorithm CM.Irreducible takes a string of $\sigma_1 t$ input bits $d_0, d_1, \dots, d_{\sigma_1 t-1}$. 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 $j \in \{0, 1, \dots, t-1\}$. (Within each group of σ_1 input bits, this uses only the first m bits. The algorithm ignores the remaining bits.)
2. Define $\beta = \beta_0 + \beta_1 y + \dots + \beta_{t-1} y^{t-1} \in \mathbb{F}_q[y]/F(y)$.
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 reference to 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 $(\alpha_0, \alpha_1, \dots, \alpha_{q-1})$ of q distinct elements of $\mathbb{F}_q$. The algorithm is as follows:

1. Take the first σ_2 input bits $b_0, b_1, \dots, b_{\sigma_2-1}$ as a σ_2 -bit integer $a_0 = b_0 + 2b_1 + \dots + 2^{\sigma_2-1} b_{\sigma_2-1}$, take the next σ_2 bits as a σ_2 -bit integer a_1 , and so on through a_{q-1} .
2. If $a_0, a_1, \dots, a_{q-1}$ are not distinct, return $\perp$.
3. Sort the pairs (a_i, i) in lexicographic order to obtain pairs $(a_{\pi(i)}, \pi(i))$ where π is a permutation of $\{0, 1, \dots, q-1\}$.
4. Define $\alpha_i = \sum_{j=0}^{m-1} \pi(i)_j \cdot z^{m-1-j}$ where $\pi(i)_j$ denotes the j th 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 $(\alpha_0, \alpha_1, \dots, \alpha_{q-1})$.

13.10 Key generation

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).

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 $E = \text{PRG}(\delta)$, 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 $\alpha_0, \dots, \alpha_{q-1}$ 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 $\Gamma = (g, \alpha_0, \alpha_1, \dots, \alpha_{n-1})$. (Note that $\alpha_n, \dots, \alpha_{q-1}$ are not used in Γ .)
7. Compute $(T, c_{mt-\mu}, \dots, c_{mt-1}, \Gamma') \leftarrow \text{CM.MatGen}(\Gamma)$. If this fails, set $\delta \leftarrow \delta'$ and restart the algorithm.
8. Write Γ' as $(g, \alpha'_0, \alpha'_1, \dots, \alpha'_{n-1})$.
9. Output T as public key and $(\delta, c, g, \alpha, s)$ as private key, where $c = (c_{mt-\mu}, \dots, c_{mt-1})$ and $\alpha = (\alpha'_0, \dots, \alpha'_{n-1}, \alpha_n, \dots, \alpha_{q-1})$.

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

The following randomized algorithm CM.Encap takes as input a public key T . It outputs a ciphertext C and a session key K . For non-pc parameter sets, the 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 Clause 13.15.5 for Hash input encodings.
4. Output ciphertext C and session key K .

For pc parameter sets, the 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 Clause 13.15.6 for Hash input encodings.
4. Compute $K = \text{Hash}(1, e, C)$.
5. Output ciphertext C and session key K .

13.13 Decapsulation

The following algorithm CM.Decap takes as input a ciphertext C and a private key, and outputs a session key K . For non-pc parameter sets, the algorithm is as follows:

1. Set $b \leftarrow 1$.
2. Extract $s \in \mathbb{F}_2^n$ and $\Gamma' = (g, \alpha'_0, \alpha'_1, \dots, \alpha'_{n-1})$ 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 Clause 13.15.5 for Hash input encodings.
5. Output session key K .

For pc parameter sets, the algorithm is as follows:

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 $b \leftarrow 1$.
3. Extract $s \in \mathbb{F}_2^n$ and $\Gamma' = (g, \alpha'_0, \alpha'_1, \dots, \alpha'_{n-1})$ 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 Clause 13.15.6 for Hash input encodings.
6. If $C'_1 \neq C_1$, set $e \leftarrow s$ and $b \leftarrow 0$.
7. Compute $K = \text{Hash}(b, e, C)$.
8. Output session key K .

13.14 Choices of symmetric-cryptography parameters

In Clauses 13.14 and 13.15, Uint8 means the set $\{0, 1, \dots, 255\}$; a Uint8 string means a string of elements of Uint8; and 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. Clause 13.15 specifies how various objects are represented as Uint8 strings.

NOTE 1 The usage of Uint8 in Clauses 13.14 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").

In Clause 13.14, $\text{SHAKE256}(x, \ell)$ takes a Uint8 string x as input and an integer ℓ , which is a multiple of 8 and produces an $(\ell/8)$ -Uint8 string. SHAKE256 is the Uint8-string ("sequence of bytes") function defined in Annex C.2 of ISO/IEC 10118-3:2018, not the bit-string function defined in that annex.

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

NOTE 3 Clause 13.15 specifies how Clause 13 represents ℓ -bit strings as $(\ell/8)$ -Uint8 strings. That representation covers all $(\ell/8)$ -Uint8 strings, allowing those strings to be viewed as ℓ -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 $\text{Hash}(x)$ is defined as $\text{SHAKE256}(x, \ell)$.
- 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 $\text{PRG}(\delta)$ is defined as $\text{SHAKE256}((64, \delta), n + \sigma_2 q + \sigma_1 t + \ell)$. 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 Clause 13.15). Thus, the hash inputs do not overlap the SHAKE256 inputs used in PRG.

13.15 Representation of objects as Uint8 strings

13.15.1 Bit vectors

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

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

If r is not a multiple of 8 then an r -bit vector $v = (v_0, v_1, \dots, v_{r-1}) \in \mathbb{F}_2^r$ 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.

13.15.3 Ciphertexts for non-pc parameter sets

For non-pc parameter sets: A ciphertext C is an element of $\mathbb{F}_2^{mt}$. It is represented as a $\lceil mt/8 \rceil$ -Uint8 string.

13.15.4 Ciphertexts for pc parameter sets

For pc parameter sets, a ciphertext C has two components: $C_0 \in \mathbb{F}_2^{mt}$ and $C_1 \in \mathbb{F}_2^\ell$. The ciphertext is represented as the concatenation of the $\lceil mt/8 \rceil$ -Uint8 string representing C_0 and the $\lceil \ell/8 \rceil$ -Uint8 string representing C_1 .

13.15.5 Hash inputs for non-pc parameter sets

For non-pc parameter sets, there are two types of hash inputs: $(1, v, C)$, and $(0, v, C)$. Here $v \in \mathbb{F}_2^n$, and C is a ciphertext.

The initial 0 or 1 is represented as a Uint8. The vector v is represented as the next $\lceil n/8 \rceil$ Uint8s. The ciphertext is represented as the next $\lceil mt/8 \rceil$ Uint8s.

NOTE All hash inputs thus begin with Uint8 0 or 1, as mentioned earlier.

13.15.6 Hash inputs for pc parameter sets

For pc parameter sets, there are three types of hash inputs: $(2, v)$; $(1, v, C)$; and $(0, v, C)$. Here $v \in \mathbb{F}_2^n$, and C is a ciphertext.

The initial 0, 1, or 2 is represented as a Uint8. The vector v is represented as the next $\lceil n/8 \rceil$ Uint8s. The ciphertext, if present, is represented as the next $\lceil mt/8 \rceil + \lceil \ell/8 \rceil$ Uint8s.

NOTE All hash inputs thus begin with Uint8 0, 1, or 2, as mentioned earlier.

13.15.7 Public keys

The public key T , which is an $mt \times k$ matrix, is represented in a row-major fashion. Each row of T is represented as a $\lceil k/8 \rceil$ -Uint8 string, and the public key is represented as the $m \lceil tk/8 \rceil$ -Uint8 concatenation of these strings.

13.15.8 Field elements

Each element of $\mathbb{F}_q \cong \mathbb{F}_2[z]/f(z)$ has the form $\sum_{i=0}^{m-1} c_i z^i$ where $c_i \in \mathbb{F}_2$. The representation of the field element is the representation of the vector $(c_0, c_1, \dots, c_{m-1}) \in \mathbb{F}_2^m$.

13.15.9 Monic irreducible polynomials

The monic irreducible degree- t polynomial $g = g_0 + g_1 x + \dots + g_{t-1} x^{t-1} + x^t$ is represented as $t \lceil m/8 \rceil$ Uint8s, namely the concatenation of the representations of the field elements $g_0, g_1, \dots, g_{t-1}$.

13.15.10 Field orderings

The obvious representation of a sequence $(\alpha_0, \dots, \alpha_{q-1})$ of q distinct elements of $\mathbb{F}_q$ would be as a sequence of q field elements. This clause instead specifies the following representation.

An “in-place Beneš network” is a series of $2m-1$ stages of swaps applied to an array of $q=2^m$ objects $(a_0, a_1, \dots, a_{q-1})$. The first stage conditionally swaps a_0 and a_1 , conditionally swaps a_2 and a_3 , conditionally swaps a_4 and a_5 , etc., as specified by a sequence of $q/2$ control bits (1 meaning swap, 0 meaning leave in place). The second stage conditionally swaps a_0 and a_2 , conditionally swaps a_1 and a_3 , conditionally swaps a_4 and a_6 , etc., as specified by the next $q/2$ control bits. This continues through the m th stage, which conditionally swaps a_0 and $a_{q/2}$, conditionally swaps a_1 and $a_{q/2+1}$, etc. The $(m+1)$ st stage is just like the $(m-1)$ st stage (with new control bits), the $(m+2)$ nd stage is just like the $(m-2)$ nd stage, and so on through the $(2m-1)$ st stage.

Define π as the permutation of $\{0, 1, \dots, q-1\}$ such that $\alpha_i = \sum_{j=0}^{m-1} \pi(i)_j \cdot z^{m-1-j}$ for all $i \in \{0, 1, \dots, q-1\}$.

Here, $\pi(i)_j$ denotes the coefficient of 2^j in the binary expansion of the integer $\pi(i)$, i.e. $\pi(i) \in \{0, 1\}$ for all

i and we have $\pi(i) = \sum_{j=0}^{m-1} \pi(i)_j 2^j$. The ordering is represented as a sequence of $(2m-1)2^{m-1}$ control bits for an in-place Beneš network for π . This vector is represented as $\lceil (2m-1)2^{m-4} \rceil$ Uint8s as above.

Mathematically, each permutation has multiple choices of control-bit vectors. However, for conformance to this document regarding Classic McEliece, a permutation π shall be converted to the control bits defined by controlbits in the following Python 3.9.2 script. This is not a requirement for the decapsulation algorithm *reading* control bits to check uniqueness.

```
def composeinv(c,pi):
    return [y for x,y in sorted(zip(pi,c))]

def controlbits(pi):
    n = len(pi)
    m = 1
    while 1<<m < n: m += 1
    assert 1<<m == n

    if m == 1: return [pi[0]]
    p = [pi[x^1] for x in range(n)]
    q = [pi[x]^1 for x in range(n)]

    piinv = composeinv(range(n),pi)
    p,q = composeinv(p,q),composeinv(q,p)

    c = [min(x,p[x]) for x in range(n)]
    p,q = composeinv(p,q),composeinv(q,p)
    for i in range(1,m-1):
        cp,p,q = composeinv(c,q),composeinv(p,q),composeinv(q,p)
        c = [min(c[x],cp[x]) for x in range(n)]

    f = [c[2*j]%2 for j in range(n//2)]
    F = [x^f[x//2] for x in range(n)]
    Fpi = composeinv(F,piinv)
    l = [Fpi[2*k]%2 for k in range(n//2)]
    L = [y^l[y//2] for y in range(n)]
    M = composeinv(Fpi,L)
    subM = [[M[2*j+e]//2 for j in range(n//2)] for e in range(2)]
    subz = map(controlbits,subM)
    z = [s for s0s1 in zip(*subz) for s in s0s1]
    return f+z+l
```

13.15.11 Column selections

Part of the private key generated by CM.KeyGen is a sequence $c = (c_{mt-\mu}, \dots, c_{mt-1})$ of μ integers in increasing order between $mt-\mu$ and $mt-\mu+v-1$.

This sequence c is represented as the $\lceil v/8 \rceil$ -Uint8 string $(u_0, u_1, \dots, u_{\lceil v/8 \rceil-1})$, such that $u_0 + u_1 2^8 + \dots + u_{\lceil v/8 \rceil-1} 2^{8(\lceil v/8 \rceil-1)}$ is equal to

$$\sum_{i=0}^{\mu-1} 2^{c_{mt-\mu+i}-(mt-\mu)}.$$

However, for $(\mu, v) = (0, 0)$, the sequence c is instead represented as the 8-Uint8 string starting with 4 Uint8s of value 255 followed by 4 Uint8s of value 0.

13.15.12 Private keys

A private key $(\delta, c, g, \alpha, s)$ is represented as the concatenation of five parts:

- The $\lceil \ell/8 \rceil$ -Uint8 string representing $\delta \in \mathbb{F}_2^\ell$.

- The string representing the column selections c . This string has $\lceil v/8 \rceil$ Uint8s, or 8 Uint8s if $(\mu, v) = (0, 0)$.
- The $t \lceil m/8 \rceil$ -Uint8 string representing the polynomial g .
- The $\lceil (2m-1)2^{m-4} \rceil$ Uint8s representing the field ordering α .
- The $\lceil n/8 \rceil$ -Uint8 string representing $s \in \mathbb{F}_2^n$.

13.16 Selected parameter sets

13.16.1 General

Table 13.16.1.1 lists the names of the selected parameter sets, and the sizes in bytes of private keys, public keys, ciphertexts, and session keys. The definitions of the selected parameter sets appear in Clause 13.16.2 through Clause 13.16.17. For the specification of symmetric-cryptography parameters, see Clause 13.14. See Annex D for references to literature describing further characteristics of these parameter sets, including security characteristics.

Table 13.16.1.1 - Sizes (in bytes) of Classic McEliece inputs and outputs

parameter set	private key	public key	ciphertext	session key
mceliece460896	13 608	524 160	156	32
mceliece6688128	13 932	1 044 992	208	32
mceliece6960119	13 948	1 047 319	194	32
mceliece8192128	14 120	1 357 824	208	32
any of the above + f	same	same	same	same
any of the above + pc	same	same	same + 32	same

13.16.2 Parameter set mceliece460896

KEM with $m=13$, $n=4\,608$, $t=96$. Field polynomials $f(z)=z^{13}+z^4+z^3+z+1$ and $F(y)=y^{96}+y^{10}+y^9+y^6+1$. This is a non-pc parameter set.

13.16.3 Parameter set mceliece460896f

KEM with $m=13$, $n=4\,608$, $t=96$. Field polynomials $f(z)=z^{13}+z^4+z^3+z+1$ and $F(y)=y^{96}+y^{10}+y^9+y^6+1$. Semi-systematic parameters $(\mu, v)=(32, 64)$. This is a non-pc parameter set.

13.16.4 Parameter set mceliece460896pc

KEM with $m=13$, $n=4\,608$, $t=96$. Field polynomials $f(z)=z^{13}+z^4+z^3+z+1$ and $F(y)=y^{96}+y^{10}+y^9+y^6+1$. This is a pc parameter set.

13.16.5 Parameter set mceliece460896pcf

KEM with $m=13$, $n=4\,608$, $t=96$. Field polynomials $f(z)=z^{13}+z^4+z^3+z+1$ and $F(y)=y^{96}+y^{10}+y^9+y^6+1$. Semi-systematic parameters $(\mu, v)=(32, 64)$. This is a pc parameter set.

13.16.6 Parameter set mceliece6688128

KEM with $m=13$, $n=6\,688$, $t=128$. Field polynomials $f(z)=z^{13}+z^4+z^3+z+1$ and $F(y)=y^{128}+y^7+y^2+y+1$. This is a non-pc parameter set.

13.16.7 Parameter set mceliece6688128f