Network Working Group
Internet-Draft
Intended status: Informational
Expires: 8 January 2026

S. Josefsson, Ed.

7 July 2025

Classic McEliece
draft-josefsson-mceliece-03

Abstract

This document specifies Classic McEliece, a Key Encapsulation Method (KEM) designed for IND-CCA2 security, even against quantum computers.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-josefsson-mceliece/.

Source for this draft and an issue tracker can be found at https://gitlab.com/jas/ietf-mceliece.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

Table of Contents

1. Introduction 3
2. Terms and definitions 4
3. Symbols and abbreviated terms 5
3.1. Guide to notation 5
3.2. Column vectors vs. row vectors 6
3.3. 0-numbering vs. 1-numbering 6
4. Requirements 6
5. Parameters 8
6. The one-way function 9
6.1. Matrix reduction 9
6.1.1. Reduced row-echelon form 9
6.1.2. Systematic form 9
6.1.3. Semi-systematic form 9
6.2. Matrix generation for Goppa codes 10
6.2.1. General 10
6.2.2. Systematic form 10
6.2.3. Semi-systematic form 11
6.3. Encoding subroutine 12
6.4. Decoding subroutine 12
7. The Classic McEliece KEM 13
7.1. Irreducible-polynomial generation 13
7.2. Field-ordering generation 13
7.3. Key generation 14
7.4. Fixed-weight-vector generation 15
7.5. Encapsulation 15
7.6. Decapsulation 16
8. Bits and bytes 17
8.1. Choices of symmetric-cryptography parameters 17
8.2. Representation of objects as byte strings 17
8.2.1. Bit vectors 17
8.2.2. Session keys 18
8.2.3. Ciphertexts for non-pc parameter sets 18
8.2.4. Ciphertexts for pc parameter sets 18
8.2.5. Hash inputs for non-pc parameter sets 18
8.2.6. Hash inputs for pc parameter sets 18
8.2.7. Public keys 19
8.2.8. Field elements 19

8.2.9. Monic irreducible polynomials 19
8.2.10. Field orderings 19
8.2.11. Column selections 21
8.2.12. Private keys 22
9. Selected parameter sets 22
9.1. Parameter set mceliece6688128 22
9.2. Parameter set mceliece6688128f 22
9.3. Parameter set mceliece6688128pc 22
9.4. Parameter set mceliece6688128pcf 22
9.5. Parameter set mceliece6960119 23
9.6. Parameter set mceliece6960119f 23
9.7. Parameter set mceliece6960119pc 23
9.8. Parameter set mceliece6960119pcf 23
9.9. Parameter set mceliece8192128 23
9.10. Parameter set mceliece8192128f 23
9.11. Parameter set mceliece8192128pc 23
9.12. Parameter set mceliece8192128pcf 23
10. Security Considerations 24
11. Acknowledgments 24
12. IANA Considerations 25
13. References 25
13.1. Normative References 25
13.2. Informative References 25
Appendix A. Overview of Classic McEliece resources (informative) 26
Author's Address 27

1. Introduction

The first code-based public-key encryption system (PKE) was introduced in 1978 [McEliece]. The public key specifies a random binary Goppa code. A ciphertext is a codeword plus random errors. The private key allows efficient decoding: extracting the codeword from the ciphertext, identifying and removing the errors.

The McEliece system was designed to be one-way (OW-CPA), meaning that an attacker cannot efficiently find the codeword from a ciphertext and public key, when the codeword is chosen randomly. The security level of the McEliece system has remained remarkably stable, despite dozens of attack papers over 45 years. The original McEliece parameters were designed for only 2^64 security, but the system easily scales up to "overkill" parameters that provide ample security margin against advances in computer technology, including quantum computers.

The McEliece system has prompted a tremendous amount of followup work. Some of this work improves efficiency while clearly preserving security: this includes a "dual" PKE proposed by Niederreiter, software speedups, and hardware speedups.

Furthermore, it is now well known how to efficiently convert an OW-CPA PKE into a KEM that is IND-CCA2 secure against all ROM attacks. This conversion is tight, preserving the security level, under two assumptions that are satisfied by the McEliece PKE: first, the PKE is deterministic (i.e., decryption recovers all randomness that was used); second, the PKE has no decryption failures for valid ciphertexts. Even better, recent work achieves similar tightness for a broader class of attacks, namely QROM attacks. The risk that a hash-function- specific attack could be faster than a ROM or QROM attack is addressed by the standard practice of selecting a well-studied, high-security, "unstructured" hash function.

Classic McEliece brings all of this together. It is a KEM designed for IND-CCA2 security at a very high security level, even against quantum computers. The KEM is built conservatively from a PKE designed for OW-CPA security, namely Niederreiter’s dual version of McEliece’s PKE using binary Goppa codes. Every level of the construction is designed so that future cryptographic auditors can be confident in the long-term security of post-quantum public-key encryption.

2. Terms and definitions

For the purposes of this document, the following terms and definitions apply.

* SHAKE256: see [NIST.FIPS.202], the sole symmetric primitive used in Classic McEliece with the selected parameters

* IND-CCA2: indistinguishability against adaptive chosen-ciphertext attacks

* KEM: key-encapsulation mechanism

* OW-CPA: one-wayness against chosen-plaintext attacks

* PKE: public-key encryption system

* ROM: random-oracle model

* QROM: quantum random-oracle model

* F_q: finite field of q

* :=: member of a set

* A_b, A_{b}: entity A subscripted with expression b

* A^b, A^{b}: entity A superscripted with expression b

* A_b^c, A_{b}^{c}, A^{c}_{b}: entity A subscripted with expression b and superscripted with expression c

* =>: larger than or equal

* <=: less than or equal

* CEILING(a): function mapping a to the least integer greater than or equal to a

* p * q: matrix multiplication

3. Symbols and abbreviated terms

3.1. Guide to notation

The list below introduces the notation used in this specification. It is meant as a reference guide only; for complete definitions of the terms listed, refer to the appropriate text. Some other symbols are also used occasionally; they are introduced in the text where appropriate.

* n: The code length (part of the CM parameters)

* k: The code dimension (part of the CM parameters)

* t: The guaranteed error-correction capability (part of the CM parameters)

* q: The size of the field used (part of the CM parameters)

* m: logarithm base 2 of q (part of the CM parameters)

* u: A nonnegative integer (part of the CM parameters)

* v: A nonnegative integer (part of the CM parameters)

* Hash: A cryptographic hash function (symmetric-cryptography parameter)

* HashLen: Length of an output of Hash (symmetric-cryptography parameter)

* Sigma_1 : A nonnegative integer (symmetric-cryptography parameter)

* Sigma_2 : A nonnegative integer (symmetric-cryptography parameter)

* PRG: A pseudorandom bit generator (symmetric-cryptography parameter)

* g: A polynomial in F_q[x] (part of the private key)

* alpha_i: An element of the finite field F_q (part of the private key)

* Gamma: (g, alpha_0, ..., alpha_{n-1}) (part of the private key)

* s: A bit string of length n (part of the private key)

* T: An mt * k matrix over F_2 (the CM public key)

* e: A bit string of length n and Hamming weight t

* C: A ciphertext encapsulating a session key

3.2. Column vectors vs. row vectors

Elements of F_2^n, such as codewords and error vectors, are always viewed as column vectors. 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.

3.3. 0-numbering vs. 1-numbering

To simplify comparisons to software in most programming languages, this specification consistently uses indices numbered from 0, including row indices, column indices, and alpha indices. 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.

4. Requirements

This document defines the Classic McEliece KEM. The KEM consists of three mathematical functions, namely KeyGen, Encap, and Decap, for each of the "selected parameter sets" listed in Clause 10.

The definitions for each selected parameter set are unified into a single definition for a broader parameter space specified in Clause 6. For each parameter set in that parameter space, subsequent clauses in this document define

* exactly which public key and private key are output by KeyGen given random bits;

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

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

This document defines each mathematical function F by presenting an algorithm to compute F. Basic algorithms such as Gaussian elimination are not repeated here, but MatGen, Encode, Decode, Irreducible, FieldOrdering, SeededKeyGen, FixedWeight, KeyGen, Encap, and Decap are specified below as numbered lists of steps.

Three of these algorithms, namely FixedWeight, KeyGen, and Encap, are randomized, generating random bits at specified moments. The set of strings of random bits allowed as input for the corresponding mathematical functions is defined as the set of strings of random bits consumed by these algorithms. For example, the KeyGen algorithm reads exactly HashLen random bits, so the domain of the mathematical function KeyGen is the set of HashLen-bit strings. Here HashLen, one of the Classic McEliece parameters, is 256 for each of the selected parameter sets.

To claim conformance to this document, an algorithm shall (1) name either KeyGen or Encap or Decap; (2) identify a parameter set listed in Clause 10 (not another parameter set from Clause 6); and (3) compute exactly the corresponding mathematical function defined in this document for that parameter set. For example, a KeyGen implementation claimed to conform to this document for the mceliece6960119 parameter set shall compute the specified KeyGen function for that parameter set: i.e., the implementation shall read exactly HashLen = 256 bits of randomness, and shall produce the same output that the KeyGen algorithm specified below produces given the same 256-bit string.

Conformance to this document for a tuple of three algorithms, one for each of KeyGen and Encap and Decap, is defined as conformance to this document for each algorithm, and again shall identify a parameter set listed in Clause 10.

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 for this document. This document defines the mathematical functions that shall be computed by any conformant algorithms; this document does not constrain how these functions are computed.

5. 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 <= q.

* A positive integer t => 2 with mt < n. This also defines a parameter k = n - mt.

* A monic irreducible polynomial f(z) := F_2[z] of degree m. This defines a representation F_2[z]/f(z) of the field F_q.

* A monic irreducible polynomial F(y) := F_q[y] of degree t. This defines a representation F_q[y]/F(y) of the field F_{q^t} = F_{2^mt}.

* Integers v => u => 0 with v <= k + u. Parameter sets that do not mention these parameters define them as (0,0) by default.

* The symmetric-cryptography parameters listed below.

The symmetric-cryptography parameters are the following:

* A positive integer HashLen.

* A cryptographic hash function Hash that outputs HashLen bits.

* An integer Sigma_1 => m.

* An integer Sigma_2 => 2m.

* A pseudorandom bit generator PRG mapping a string of HashLen bits to a string of n + Sigma_2 q + Sigma_1 t + HashLen bits.

6. The one-way function

6.1. Matrix reduction

6.1.1. Reduced row-echelon form

Given a matrix X, Gaussian elimination 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. Being in reduced row-echelon form means that there is a sequence c_0 < c_1 < ... < 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 that the rank of R is r.

6.1.2. Systematic form

As a special case, R is in systematic form if

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

* c_i = i for 0 <= i < r. (This second condition is equivalent to simply saying c_{r-1} = r - 1, except in the degenerate case r = 0.)

In other words, R has the form (I_r|T), where I is an r * r identity matrix. 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.

6.1.3. Semi-systematic form

The following generalization of the concept of systematic form uses two integer parameters u,v satisfying v => u => 0.

Let R be a rank-r matrix in reduced row-echelon form. Assume that r => u, and that there are at least r - u + v columns.

We say that R is in (u,v)-semi-systematic form if R has r rows (i.e., no zero rows); c_i = i for 0 <= i < r - u; and c_i <= i - u + v for 0 <= i < r. (The c_i conditions are equivalent to simply c_{r-u-1} = r - u - 1 and c_{r-1} <= r - u + v - 1 except in the degenerate case r = u.)

As a special case, (u,v)-semi-systematic form is equivalent to systematic form if u = v. However, if v > u then (u,v)-semi-systematic form allows more matrices than systematic form.

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

6.2. Matrix generation for Goppa codes

6.2.1. General

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

* g is a monic irreducible polynomial in F_q[x] of degree t and

* alpha_0, alpha_1, ..., alpha_{n-1} are distinct elements of F_q.

The algorithm output MatGen(Gamma) is defined first in the simpler case of systematic form, and then in the general case of semi-systematic form. The output is either NIL or of the form (T, ...), where T is the CM public key, an mt * k matrix over F_2.

6.2.2. Systematic form

For (u,v) = (0,0), the algorithm output MatGen(Gamma) is either NIL or of the form (T, Gamma), where T is an mt * k matrix over F_2. Here is the algorithm:

1. Compute the t * n matrix M = {h_{i,j}} over F_q, where h_{i,j} = alpha_j^i / g(alpha_j) for i = 0, ..., t - 1 and j = 0, ..., n - 1.

2. Form an mt * n matrix N over F_2 by replacing each entry u_0 + u_1 z + ... + u_{m-1} z^{m-1} of M with a column of m bits u_0, u_1, ..., u_{m-1}.

3. Reduce N to systematic form (I_{mt}|T), where I_{mt} is an mt * mt identity matrix. If this fails, return NIL.

4. Return (T, Gamma).

6.2.3. Semi-systematic form

For general u,v, the algorithm output MatGen(Gamma) is either NIL or of the form (T, c_{mt-u}, ..., c_{mt-1}, Gamma'), where

* T is an mt * k matrix over F_2;

* c_{mt-u}, ..., c_{mt-1} are integers with mt - u <= c_{mt-u} < c_{mt-u+1} < ... < c_{mt-1} < mt - u + v;

* Gamma' = (g, alpha'_0, alpha'_1, ..., alpha'_{n-1});

* g is the same as in the input; and

* alpha'_0, alpha'_1, ..., alpha'_{n-1} are distinct elements of F_q.

Here is the algorithm:

1. Compute the t * n matrix M = {h_{i,j}} over F_q, where h_{i,j} = alpha_j^i / g(alpha_j) for i = 0, ..., t - 1 and j = 0, ..., n - 1.

2. Form an mt * n matrix N over F_2 by replacing each entry u_0 + u_1 z + ... + u_{m-1} z^{m-1} of M with a column of m bits u_0, u_1, ..., u_{m-1}.

3. Reduce N to (u,v)-semi-systematic form, obtaining a matrix H. If this fails, return NIL. (Now row i has its leading 1 in column c_i. By definition of semi-systematic form, c_i = i for 0 <= i < mt - u; and mt - u <= c_{mt-u} < c_{mt-u+1} < ... < c_{mt-1} < mt - u + v. The matrix H is a variable that can change later.)

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

5. For i = mt - u, then i = mt - u + 1, and so on through i = mt - 1, in this order: swap column i with column c_i in H, while swapping alpha'_i with alpha'_{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 an mt * mt identity matrix. Return (T, c_{mt-u}, ..., c_{mt-1}, Gamma') where Gamma' = (g, alpha'_0, alpha'_1, ..., alpha'_{n-1}).

In the special case (u,v) = (0,0), the c_{mt-u}, ..., c_{mt-1} portion of the output is empty, and the i loop is empty, so Gamma' is guaranteed to be the same as Gamma. 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.

6.3. Encoding subroutine

The following algorithm Encode takes two inputs: a weight-t column vector e := F_2^n; and a public key T, i.e., an mt * k matrix over F_2. The algorithm output Encode(e, T) is a vector C := F_2^{mt}. Here is the algorithm:

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

2. Compute and return C = He := F_2^{mt}.

6.4. Decoding subroutine

The following algorithm Decode decodes C := F_2^{mt} to a word e of Hamming weight wt(e) = t with C = He if such a word exists; otherwise it returns failure.

Formally, Decode takes two inputs: a vector C := F_2^{mt}; and Gamma', the last component of MatGen(Gamma) for some Gamma such that MatGen(Gamma) != NIL. Write T for the first component of MatGen(Gamma). By definition of MatGen,

* T is an mt * k matrix over F_2;

* Gamma' has the form (g, alpha'_0, alpha'_1, ..., alpha'_{n-1});

* g is a monic irreducible polynomial in F_q[x] of degree t; and

* alpha'_0, alpha'_1, ..., alpha'_{n-1} are distinct elements of F_q.

There are two possibilities for Decode(C, Gamma'):

* If C = Encode(e, T) then Decode(C, Gamma') = e. In other words, if there exists a weight-t vector e := F_2^n such that C = He with H = (I_{mt}|T), then Decode(C, Gamma') = e.

* If C does not have the form He for any weight-t vector e := F_2^n, then Decode(C, Gamma') = NIL.

Here is the algorithm:

1. Extend C to v = (C, 0, ..., 0) := F_2^n by appending k zeros.

2. Find the unique c := F_2^n such that (1) Hc = 0 and (2) c has Hamming distance <= t from v. If there is no such c, return NIL.

3. Set e = v + c.

4. If wt(e) = t and C = He, return e. Otherwise return NIL.

7. The Classic McEliece KEM

7.1. Irreducible-polynomial generation

The following algorithm Irreducible takes a string of Sigma_1 t input bits d_0, d_1, ..., d_{Sigma_1 t - 1}. It outputs either NIL or a monic irreducible degree-t polynomial g := F_q[x]. Here is the algorithm:

1. Define beta_j = SUM^{m-1}_{i=0}(d_{Sigma_1 j + i} z^i) for each j := {0, 1, ..., t - 1}. (Within each group of Sigma_1 input bits, this uses only the first m bits. The algorithm ignores the remaining bits.)

2. Define beta = beta_0 + beta_1 y + ... + beta_{t-1} y^{t-1} := F_q[y] / F(y).

3. Compute the minimal polynomial g of beta over F_q. (By definition g is monic and irreducible, and g(beta) = 0.)

4. Return g if g has degree t. Otherwise return NIL.

7.2. Field-ordering generation

The following algorithm FieldOrdering takes a string of Sigma_2 q input bits. It outputs either NIL or a sequence (alpha_0, alpha_1, ..., alpha_{q-1}) of q distinct elements of F_q. Here is the algorithm:

1. Take the first Sigma_2 input bits b_0, b_1, ..., b_{Sigma_2-1} as a Sigma_2-bit integer a_0 = b_0 + 2b_1 + ... + 2^{Sigma_2 -1} b_{Sigma_2-1}, take the next Sigma_2 bits as a Sigma_2-bit integer a_1, and so on through a_{q-1}.

2. If a_0, a_1, ..., a_q-1 are not distinct, return NIL.

3. Sort the pairs (a_i, i) in lexicographic order to obtain pairs (a_pi(i), pi(i)) where pi is a permutation of {0, 1, ..., q - 1}.

4. Define alpha_i = SUM^{m-1}_{j=0}(pi(i)_j z^{m-1-j})

5. where pi(i)_j denotes the j:th least significant bit of pi(i). (Recall that the finite field F_q is constructed as F_2[z] / f(z).)

6. Output (alpha_0, alpha_1, ..., alpha_{q-1}).

7.3. Key generation

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

1. Generate a uniform random HashLen-bit string Delta. (This is called a seed.)

2. Output SeededKeyGen(Delta).

The following algorithm SeededKeyGen takes an HashLen-bit input Delta. It outputs a public key and private key. Here is the algorithm:

1. Compute E = PRG(Delta), a string of n + Sigma_2 q + Sigma_1 t + HashLen bits.

2. Define Delta’ as the last HashLen bits of E.

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

4. Compute alpha_0, ..., alpha_{q-1} from the next Sigma_2 q bits of E by the FieldOrdering algorithm. If this fails, set Delta = Delta’ and restart the algorithm.

5. Compute g from the next Sigma_1 t bits of E by the Irreducible algorithm. If this fails, set Delta = Delta’ and restart the algorithm.

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

7. Compute (T, c_{mt-u}, ..., c_{mt-1}, Gamma’) = MatGen(Gamma). If this fails, set Delta = Delta’ and restart the algorithm.

8. Write Gamma’ as (g, alpha’_0, alpha’_1, ..., alpha’_{n-1}).

9. Output T as public key and (Delta, c, g, alpha, s) as private key, where c = (c_{mt-u}, ..., c_{mt-1}) and alpha = (alpha’_0, ..., alpha’_{n-1}, alpha_n, ..., alpha_{q-1}).

7.4. Fixed-weight-vector generation

The following randomized algorithm FixedWeight takes no input. It outputs a vector e := F_2^n of weight t. The algorithm uses a precomputed integer tau => t defined below. Here is the algorithm:

1. Generate Sigma_1 tau uniform random bits b_0, b_1, ..., b_{Sigma_1 tau-1}.

2. Define d_j = SUM^{m-1}_{i=0}(b_{Sigma_1 j + i} 2^i) for each j := {0, 1, ..., tau - 1}. (Within each group of Sigma_1 random bits, this uses only the first m bits. The algorithm ignores the remaining bits.)

3. Define a_0, a_1, ..., a_{t-1} as the first t entries in d_0, d_1, ..., d_{tau-1} in the range {0, 1, ..., n - 1}. If there are fewer than t such entries, restart the algorithm.

4. If a_0, a_1, ..., a_{t-1} are not all distinct, restart the algorithm.

5. Define e = (e_0, e_1, ..., e_{n-1}) := F_2^n as the weight-t vector such that e_{a_i} = 1 for each i.

6. Return e.

The integer tau is defined as t if n = q; as 2t if q/2 <= n < q; as 4t if q/4 <= n < q/2; etc. All of the selected parameter sets have q / 2 <= n <= q, so tau := {t, 2t}.

7.5. Encapsulation

The following randomized algorithm Encap takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm for non-pc parameter sets:

1. Use FixedWeight to generate a vector e := F_2^n of weight t.

2. Compute C = Encode(e, T).

3. Compute K = Hash(1, e, C); see Clause 9.2 for Hash input encodings.

4. Output ciphertext C and session key K.

Here is the algorithm for pc parameter sets:

1. Use FixedWeight to generate a vector e := F_2^n of weight t.

2. Compute C_0 = Encode(e, T).

3. Compute C_1 = Hash(2, e). Put C = (C_0, C_1).

4. Compute K = Hash(1, e, C).

5. Output ciphertext C and session key K.

7.6. Decapsulation

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

1. Set b = 1.

2. Extract s := F_2^n and Gamma' = (g, alpha'_0, alpha'_1, ..., alpha'_{n-1}) from the private key.

3. Compute e = Decode(C, Gamma'). If e = NIL, set e = s and b = 0.

4. Compute K = Hash(b, e, C); see Clause 9.2 for Hash input encodings.

5. Output session key K.

Here is the algorithm for pc parameter sets:

1. Split the ciphertext C as (C_0, C_1) with C_0 := F_2^{mt} and C_1 := F_2^HashLen.

2. Set b = 1.

3. Extract s := F_2^n and Gamma' = (g, alpha'_0, alpha'_1, ..., alpha'_{n-1}) from the private key.

4. Compute e = Decode(C_0, Gamma'). If e = NIL, set e = s and b = 0.

5. Compute C'_1 = Hash(2, e).

6. If C'_1 != C_1, set e = s and b = 0.

7. Compute K = Hash(b, e, C).

8. Output session key K.

8. Bits and bytes

8.1. Choices of symmetric-cryptography parameters

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

* The integer HashLen is 256.

* The HashLen-bit string Hash(x) is defined as the first HashLen bits of output of SHAKE256(x). Byte strings here are viewed as bit strings in little-endian form; see Clause 9.2. The set of bytes is defined as {0, 1, ..., 255}.

* The integer Sigma_1 is 16. (All of the selected parameter sets have m <= 16, so Sigma_1 => m.)

* The integer Sigma_2 is 32.

* The (n + Sigma_2 q + Sigma_1 t + HashLen)-bit string PRG(Delta) is defined as the first n + Sigma_2 q + Sigma_1 t + HashLen bits of output of SHAKE256(64, Delta). Here 64, Delta means the 33-byte string that begins with byte 64 and continues with Delta.

All Hash inputs used in Classic McEliece begin with byte 0 or 1 (or 2 for pc) (see Clause 9.2), and thus do not overlap the SHAKE256 inputs used in PRG.

8.2. Representation of objects as byte strings

8.2.1. Bit vectors

If r is a multiple of 8 then an r-bit vector v = (v_0, v_1, ..., v_{r-1}) := F_2^r is represented as the following sequence of r/8 bytes:

(v0 + 2v_1 + 4v_2 + ... + 128v_7, v_8 + 2v_9 + 4v_10 + ... + 128v_15, ..., v_{r-8} + 2v_{r-7} + 4v_{r-6} + ... + 128v_{r-1}).

If r is not a multiple of 8 then an r-bit vector v = (v_0, v_1, ..., v_{r-1}) := 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.

By definition, 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 NIL. 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 the distinction. Conformance to this document does not require a Simply Decoded or Narrowly Decoded label.

8.2.2. Session keys

A session key K is an element of F_2^HashLen. It is represented as a CEILING(HashLen/8)-byte string.

8.2.3. Ciphertexts for non-pc parameter sets

For non-pc parameter sets: A ciphertext C is an element of F_2^{mt}. It is represented as a CEILING(mt/8)-byte string.

8.2.4. Ciphertexts for pc parameter sets

For pc parameter sets, a ciphertext C has two components: C_0 := F_2^mt and C_1 := F_2^HashLen. The ciphertext is represented as the concatenation of the CEILING(mt/8)-byte string representing C_0 and the CEILING(HashLen/8)-byte string representing C_1.

8.2.5. 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 := F_2^n, and C is a ciphertext.

The initial 0 or 1 is represented as a byte. The vector v is represented as the next CEILING(n/8) bytes. The ciphertext is represented as the next CEILING(mt/8) bytes. All hash inputs thus begin with byte 0 or 1, as mentioned earlier.

8.2.6. 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 := F_2^n, and C is a ciphertext.

The initial 0, 1, or 2 is represented as a byte. The vector v is represented as the next CEILING(n/8) bytes. The ciphertext, if present, is represented as the next CEILING(mt/8) + CEILING(HashLen/8) bytes.

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

8.2.7. Public keys

The public key T, which is an mt * k matrix, is represented in a row-major fashion. Each row of T is represented as a CEILING(k/8)-byte string, and the public key is represented as the mt CEILING(k/8)-byte concatenation of these strings.

8.2.8. Field elements

Each element of F_q congruent F_2[z] / f(z) has the form SUM^{m-1}_{i=0}(c_i z) where c_i := F_2. The representation of the field element is the representation of the vector (c_0, c_1, ..., c_{m-1}) := F_2^m.

8.2.9. Monic irreducible polynomials

The monic irreducible degree-t polynomial g = g_0 + g_1 x + ... + g_{t-1} x^{t-1} + x t is represented as t CEILING(m/8) bytes, namely the concatenation of the representations of the field elements g_0, g_1, ..., g_{t-1}.

8.2.10. Field orderings

The obvious representation of a sequence (alpha_0, ..., alpha_{q-1}) of q distinct elements of F_q would be as a sequence of q field elements. This document instead specifies the following representation.

An "in-place Benes network" is a series of 2m - 1 stages of swaps applied to an array of q = 2^m objects (a_0, a_1, ..., 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 pi as the permutation of {0, 1, ..., q - 1} such that alpha_i = SUM^{m-1}_{j=0}(pi(i)_j z^{m-1-j} for all i := {0, 1, ..., q - 1}. The ordering (alpha_0, ..., alpha_{q-1}) is represented as a sequence of (2m - 1)2^{m-1} control bits for an in-place Benes network for pi. This vector is represented as CEILING((2m - 1)2^{m-4}) bytes as above.

Mathemtically, each permutation has multiple choices of control-bit vectors. For conformance to this document, a permutation pi shall be converted to specifically the control bits defined by controlbits in the following Python 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
```

8.2.11. Column selections

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

This sequence c is represented as a CEILING(v/8)-byte string, the little-endian format of the integer SUM^{u-1}_{i=0}(2^{c_{mt-u+i}-(mt-u).

However, for (u,v) = (0,0), the sequence c is instead represented as the 8-byte string which is the little- endian format of 2^32 - 1, i.e., 4 bytes of value 255 followed by 4 bytes of value 0.

8.2.12. Private keys

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

* The CEILING(HashLen/8)-byte string representing Delta := F_2^HashLen.
* The string representing the column selections c. This string has CEILING(v/8) bytes, or 8 bytes if (u,v) = (0,0).
* The tCEILING(m/8)-byte string representing the polynomial g.
* The CEILING((2m - 1)2^{m-4}) bytes representing the field ordering alpha.
* The CEILING(n/8)-byte string representing s := F_2^n.

9. Selected parameter sets

9.1. Parameter set mceliece6688128

KEM with m = 13, n = 6688, 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.

9.2. Parameter set mceliece6688128f

KEM with m = 13, n = 6688, 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. Semi-systematic parameters (u,v) = (32,64). This is a non-pc parameter set.

9.3. Parameter set mceliece6688128pc

KEM with m = 13, n = 6688, 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 pc parameter set.

9.4. Parameter set mceliece6688128pcf

KEM with m = 13, n = 6688, 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. Semi-systematic parameters (u,v) = (32,64). This is a pc parameter set.

9.5. Parameter set mceliece6960119

KEM with m = 13, n = 6960, t = 119. Field polynomials f(z) = z^13 + z^4 + z^3 + z + 1 and F(y) = y 119 + y^8 + 1. This is a non-pc parameter set.

9.6. Parameter set mceliece6960119f

KEM with m = 13, n = 6960, t = 119. Field polynomials f(z) = z^13 + z^4 + z^3 + z + 1 and F(y) = y 119 + y^8 + 1. Semi-systematic parameters (u,v) = (32,64). This is a non-pc parameter set.

9.7. Parameter set mceliece6960119pc

KEM with m = 13, n = 6960, t = 119. Field polynomials f(z) = z^13 + z^4 + z^3 + z + 1 and F(y) = y 119 + y^8 + 1. This is a pc parameter set.

9.8. Parameter set mceliece6960119pcf

KEM with m = 13, n = 6960, t = 119. Field polynomials f(z) = z^13 + z^4 + z^3 + z + 1 and F(y) = y 119 + y^8 + 1. Semi-systematic parameters (u,v) = (32,64). This is a pc parameter set.

9.9. Parameter set mceliece8192128

KEM with m = 13, n = 8192, 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.

9.10. Parameter set mceliece8192128f

KEM with m = 13, n = 8192, 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. Semi-systematic parameters (u,v) = (32,64). This is a non-pc parameter set.

9.11. Parameter set mceliece8192128pc

KEM with m = 13, n = 8192, 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 pc parameter set.

9.12. Parameter set mceliece8192128pcf

KEM with m = 13, n = 8192, 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. Semi-systematic parameters (u,v) = (32,64). This is a pc parameter set.

10. Security Considerations

Classic McEliece is a Key Encapsulation Mechanism designed to achieve IND-CCA2 security at a very high security level, against conventional and quantum computers.

The security of Classic McEliece depends on the availability and proper use of cryptographically secure random data.

Implementation should be designed to minimize leaking of security sensitive material, including protecting against side-channel attacks.

New research results on the security of Classic McEliece may be published at any time that may warrant implementation or deployment reconsiderations.

To hedge against new research findings, Classic McEliece may be combined with other algorithms (e.g., Curve25519) in a "hybrid" mode intended to be no weaker than any one of the individual algorithms used and the way the algorithms are combined. We recommend to use the Chempat [I-D.josefsson-chempat] combiner.

11. Acknowledgments

This document is a transcribed version of the proposed ISO McEliece standard (the "20230419" [CM-iso] version).

The Classic McEliece team is to be considered the author and owner of the text in this document, and consists of (in alphabetical order):

* Daniel J. Bernstein, University of Illinois at Chicago and Ruhr University Bochum
* Tung Chou, Academia Sinica
* Carlos Cid, Simula UiB and Okinawa Institute of Science and Technology
* Jan Gilcher, ETH Zurich
* Tanja Lange, Eindhoven University of Technology
* Varun Maram, ETH Zurich
* Ingo von Maurich, self
* Rafael Misoczki, Google

* Ruben Niederhagen, Academia Sinica and University of Southern Denmark

* Edoardo Persichetti, Florida Atlantic University

* Christiane Peters, self

* Nicolas Sendrier, Inria

* Jakub Szefer, Yale University

* Cen Jung Tjhai, PQ Solutions Ltd.

* Martin Tomlinson, PQ Solutions Ltd. and University of Plymouth

* Wen Wang, Yale University

The editor would like to thank John Mattsson, Peter C and Loganaden Velvindron for comments that improved the document.

12. IANA Considerations

This document has no IANA actions.

13. References

13.1. Normative References

[NIST.FIPS.202]
Dworkin, M., Dworkin, M. J., and NIST, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", FIPS PUB 202, NIST Federal Information Processing Standards Publications 202, DOI 10.6028/nist.fips.202, DOI 10.6028/NIST.FIPS.202, August 2015, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.

13.2. Informative References

[CM-impl] Classic McEliece Team, "Classic McEliece: conservative code-based cryptography: guide for implementors", October 2022, <https://classic.mceliece.org/mceliece-impl-20221023.pdf>.

[CM-iso] Classic McEliece Team, "Information security - Encryption algorithms - Part 1978: Classic McEliece", April 2023, <https://classic.mceliece.org/iso-mceliece-20230419.pdf>.

[CM-papers]
Classic McEliece Team, "Classic McEliece: papers", October 2022, <https://classic.mceliece.org/papers.html>.

[CM-pc] Classic McEliece Team, "Classic McEliece: conservative code-based cryptography: what plaintext confirmation means", October 2022, <https://classic.mceliece.org/mceliece-pc-20221023.pdf>.

[CM-rationale]
Classic McEliece Team, "Classic McEliece: conservative code-based cryptography: design rationale", October 2022, <https://classic.mceliece.org/mceliece-rationale-20221023.pdf>.

[CM-sage] Classic McEliece Team, "Classic McEliece: Sage package", October 2022, <https://classic.mceliece.org/spec.html>.

[CM-security]
Classic McEliece Team, "Classic McEliece: conservative code-based cryptography: guide for security reviewers", October 2022, <https://classic.mceliece.org/mceliece-security-20221023.pdf>.

[CM-spec] Classic McEliece Team, "Classic McEliece: conservative code-based cryptography: cryptosystem specification", October 2022, <https://classic.mceliece.org/mceliece-spec-20221023.pdf>.

[I-D.josefsson-chempat]
Josefsson, S., "Chempat: Generic Instantiated PQ/T Hybrid Key Encapsulation Mechanisms", Work in Progress, Internet-Draft, draft-josefsson-chempat-03, 18 March 2025, <https://datatracker.ietf.org/doc/html/draft-josefsson-chempat-03>.

[McEliece] McEliece, R. J., "A public-key cryptosystem based on algebraic coding theory", 1978, <https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF>.

Appendix A. Overview of Classic McEliece resources (informative)

Classic McEliece is specified in [CM-spec] and, for the pc options, [CM-pc]. The specification in this document is compatible with [CM-spec] and [CM-pc]. For the design rationale, see [CM-rationale].

[CM-sage] presents algorithms for the Classic McEliece functions in the Sage language. Subject to being computer-executable, this package is designed for the algorithms to be as readable as possible, including detailed comments matching the algorithms to [CM-spec] (and [CM-pc]).

[CM-impl] provides guidance to implementors. For example, it covers security against side-channel attacks, considerations in picking a parameter set, engineering cryptographic network applications for efficiency, existing implementations, and how to build new implementations.

[CM-security] provides guidance to security reviewers. As a preliminary matter, [CM-security] covers correctness of the cryptosystem: for example, c in Step 2 of Decode is unique if it exists, and c always exists when C is output by Encap. [CM-security] then reviews the stability of attacks against the original 1978 McEliece cryptosystem introduced in [McEliece], and reviews the tight relationship between the OW-CPA security of that cryptosystem and the QROM IND-CCA2 security of Classic McEliece.

Given the analysis in [CM-security], all of the parameters selected in this document meet ISO's requirement of 2^128 post-quantum security against known attacks. This is true even if one counts merely qubit operations, ignoring (1) qubit overheads and (2) the costs of memory access inside attacks. (This document does not comment on whether parameters not listed here also meet this requirement.) For comparison, 128-bit ciphers such as AES-128 provide only slightly more than 2^64 security in this metric.

Many further references can be found in the documents cited above and in [CM-papers].

Author's Address

Simon Josefsson (editor)
Email: simon@josefsson.org