Classic McEliece: conservative code-based cryptography: cryptosystem specification

23 October 2022

Contents

1 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 Section 7.

The definitions for each selected parameter set are unified into a single definition for a broader parameter space specified in Section 3. For each parameter set in that parameter space, subsequent sections of 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 ℓ random bits, so the domain of the mathematical function 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.

To claim **compliance** with this document, an algorithm must (1) name either KeyGen or Encap or Decap; (2) identify a parameter set listed in Section 7 (not another parameter set from Section 3); and (3) compute exactly the corresponding mathematical function defined in this document for that parameter set. For example, a KeyGen implementation claimed to comply with this document for the `mceliece6960119` parameter set is required to compute the specified KeyGen function for that parameter set: i.e., the implementation is required to read exactly $\ell = 256$ bits of randomness, and to produce the same output that the KeyGen algorithm specified below produces given the same 256-bit string.

Compliance for a tuple of three algorithms, one for each of KeyGen and Encap and Decap, is defined as compliance for each algorithm, and again must identify a parameter set listed in Section 7.

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 must be computed by any compliant algorithms; this document does not constrain how these functions are computed.

2 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	$\log_2 q$	(part of the CM parameters)
μ	A nonnegative integer	(part of the CM parameters)
ν	A nonnegative integer	(part of the CM parameters)
H	A cryptographic hash function	(symmetric-cryptography parameter)
ℓ	Length of an output of H	(symmetric-cryptography parameter)
σ_1	A nonnegative integer	(symmetric-cryptography parameter)
σ_2	A nonnegative integer	(symmetric-cryptography parameter)
G	A pseudorandom bit generator	(symmetric-cryptography parameter)
g	A polynomial in $\mathbb{F}_q[x]$	(part of the private key)
α_i	An element of the finite field $\mathbb{F}_q$	(part of the private key)
Γ	$(g, \alpha_0, \ldots, \alpha_{n-1})$	(part of the private key)
s	A bit string of length n	(part of the private key)
T	An $mt \times k$ matrix over $\mathbb{F}_2$	(the CM public key)
e	A bit string of length n and Hamming weight t	
C	A ciphertext of length mt encapsulating a session key	

Column vectors vs. row vectors. Elements of $\mathbb{F}_2^n$, 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.

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 α 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.

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.
- The symmetric-cryptography parameters listed below.

The symmetric-cryptography parameters are the following:

- A positive integer ℓ.
- A cryptographic hash function H that outputs ℓ bits.
- An integer $\sigma_1 \geq m$.
- An integer $\sigma_2 \geq 2m$.
- A pseudorandom bit generator G mapping a string of ℓ bits to a string of $n+\sigma_2 q+\sigma_1 t+\ell$ bits.

4 The one-way function

4.1 Matrix reduction

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 $c_0 < c_1 < \cdots < c_{r-1}$ 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 that the rank of R is r.

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 $0 \leq i < r$. (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 $(I_r \mid T)$, where I is an $r \times r$ 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.

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 $0 \leq i < r$. (The c_i conditions are equivalent to simply $c_{r-\mu-1} = r - \mu - 1$ and $c_{r-1} \leq r - \mu + \nu - 1$ except in the degenerate case $r = \mu$.)

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

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

4.2 Matrix generation for Goppa codes

The following algorithm 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 algorithm output 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 of the form $(T, \ldots)$, where T is the *CM public key*, an $mt \times k$ matrix over $\mathbb{F}_2$.

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

1. Compute the $t \times n$ matrix $\tilde{H} = \{h_{i,j}\}$ 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 $\hat{H}$ over $\mathbb{F}_2$ by replacing each entry $u_0 + u_1 z + \cdots + u_{m-1} z^{m-1}$ of $\tilde{H}$ with a column of m bits $u_0, u_1, \ldots, u_{m-1}$.
3. Reduce $\hat{H}$ to systematic form $(I_{mt} \mid T)$, where I_{mt} is an $mt \times mt$ identity matrix. If this fails, return $\perp$.
4. Return (T, Γ).

Semi-systematic form. For general μ, ν, the algorithm output MatGen(Γ) is either $\perp$ or of the form $(T, c_{mt-\mu}, \ldots, c_{mt-1}, \Gamma')$, where

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

Here is the algorithm:

1. Compute the $t \times n$ matrix $\tilde{H} = \{h_{i,j}\}$ 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 $\hat{H}$ over $\mathbb{F}_2$ by replacing each entry $u_0 + u_1 z + \cdots + u_{m-1} z^{m-1}$ of $\tilde{H}$ with a column of m bits $u_0, u_1, \ldots, u_{m-1}$.
3. Reduce $\hat{H}$ to (μ, ν)-semi-systematic form, obtaining a matrix H. If this fails, return $\perp$. (Now row i has its leading 1 in column c_i. By definition of semi-systematic form, $c_i = i$ for $0 \le i < mt - \mu$; and $mt - \mu \le c_{mt-\mu} < c_{mt-\mu+1} < \cdots < c_{mt-1} < mt - \mu + \nu$. 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 $(I_{mt} \mid T)$, where I_{mt} is an $mt \times mt$ identity matrix. Return $(T, c_{mt-\mu}, \ldots, c_{mt-1}, \Gamma')$ where $\Gamma' = (g, \alpha'_0, \alpha'_1, \ldots, \alpha'_{n-1})$.

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.

4.3 Encoding subroutine

The following algorithm 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 $\text{ENCODE}(e, T)$ is a vector $C \in \mathbb{F}_2^{mt}$. Here is the algorithm:

1. Define $H = (I_{mt} \mid T)$.
2. Compute and return $C = He \in \mathbb{F}_2^{mt}$.

4.4 Decoding subroutine

The following algorithm DECODE decodes $C \in \mathbb{F}_2^{mt}$ to a word e of Hamming weight $\mathsf{wt}(e) = t$ with $C = He$ if such a word exists; otherwise it returns failure.

Formally, DECODE takes two inputs: a vector $C \in \mathbb{F}_2^{mt}$; and Γ', the last component of $\text{MATGEN}(\Gamma)$ for some Γ such that $\text{MATGEN}(\Gamma) \neq \perp$. Write T for the first component of $\text{MATGEN}(\Gamma)$. By definition of 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 $\text{DECODE}(C, \Gamma')$:

- If $C = \text{ENCODE}(e, T)$ then $\text{DECODE}(C, \Gamma') = e$. In other words, if there exists a weight-t vector $e \in \mathbb{F}_2^n$ such that $C = He$ with $H = (I_{mt} \mid T)$, then $\text{DECODE}(C, \Gamma') = e$.
- If C does not have the form He for any weight-t vector $e \in \mathbb{F}_2^n$, then $\text{DECODE}(C, \Gamma') = \perp$.

Here is the algorithm:

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 $\perp$. (For the fact that c is unique if it exists, and the fact that c always exists when C is output by ENCAP, see the separate "guide for security reviewers" document.)
3. Set $e = v + c$.
4. If $\mathsf{wt}(e) = t$ and $C = He$, return e. Otherwise return $\perp$.

5 The Classic McEliece KEM

5.1 Irreducible-polynomial generation

The following algorithm 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 $g \in \mathbb{F}_q[x]$. Here is the algorithm:

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 $g(\beta) = 0$.)
4. Return g if g has degree t. Otherwise return $\perp$.

5.2 Field-ordering generation

The following algorithm 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 $\mathbb{F}_q$. Here is the algorithm:

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 $(a_{\pi(i)}, \pi(i))$ 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 $(\alpha_0, \alpha_1, \ldots, \alpha_{q-1})$.

5.3 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 ℓ-bit string δ. (This is called a *seed*.)
2. Output SEEDEDKEYGEN(δ).

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

1. Compute $E = \mathsf{G}(\delta)$, 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 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 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 $(T, c_{mt-\mu}, \ldots, c_{mt-1}, \Gamma') \leftarrow \text{MatGen}(\Gamma)$. 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 $c = (c_{mt-\mu}, \ldots, c_{mt-1})$ and $\alpha = (\alpha'_0, \ldots, \alpha'_{n-1}, \alpha_n, \ldots, \alpha_{q-1})$.

5.4 Fixed-weight-vector generation

The following randomized algorithm 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. Here is the algorithm:

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

5.5 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:

1. Use FixedWeight to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C = \text{Encode}(e, T)$.
3. Compute $K = \mathsf{H}(1, e, C)$; see Section 6.2 for H input encodings.
4. Output ciphertext C and session key K.

5.6 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:

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 $e \leftarrow \text{Decode}(C, \Gamma')$. If $e = \perp$, set $e \leftarrow s$ and $b \leftarrow 0$.
4. Compute $K = \mathsf{H}(b, e, C)$; see Section 6.2 for H input encodings.
5. Output session key K.

6 Bits and bytes

6.1 Choices of symmetric-cryptography parameters

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

- The integer ℓ is 256.
- The ℓ-bit string $\mathsf{H}(x)$ is defined as the first ℓ bits of output of SHAKE256(x). Byte strings here are viewed as bit strings in little-endian form; see Section 6.2. The set of bytes is defined as $\{0, 1, \ldots, 255\}$.
- 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 $\mathsf{G}(\delta)$ is defined as the first $n + \sigma_2 q + \sigma_1 t + \ell$ bits of output of SHAKE256$(64, \delta)$. Here $64, \delta$ means the 33-byte string that begins with byte 64 and continues with δ.

All H inputs used in Classic McEliece begin with byte 0 or 1 (see Section 6.2), and thus do

not overlap the SHAKE256 inputs used in G.

6.2 Representation of objects as byte strings

Vectors over $\mathbb{F}_2$. 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 sequence of $r/8$ bytes:

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

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.

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 $\perp$. 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.

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

Ciphertexts. A ciphertext C is an element of $\mathbb{F}_2^{mt}$. It is represented as a $\lceil mt/8 \rceil$-byte string.

Hash inputs. 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 byte. The vector v is represented as the next $\lceil n/8 \rceil$ bytes. The ciphertext is represented as the next $\lceil mt/8 \rceil$ bytes. All hash inputs thus begin with byte 0 or 1, as mentioned earlier.

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 $\lceil k/8 \rceil$-byte string, and the public key is represented as the $mt\lceil k/8 \rceil$-byte concatenation of these strings.

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 $(c_0, c_1, \ldots, c_{m-1}) \in \mathbb{F}_2^m$.

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

Field orderings. The obvious representation of a sequence $(\alpha_0, \ldots, \alpha_{q-1})$ of q distinct elements of $\mathbb{F}_q$ would be as a sequence of q field elements. This document 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 $(a_0, a_1, \ldots, a_{q-1})$. 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 mth 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 $\{0, 1, \ldots, q-1\}$ such that $\alpha_i = \sum_{j=0}^{m-1} \pi(i)_j \cdot z^{m-1-j}$ for all $i \in \{0, 1, \ldots, q-1\}$. The ordering $(\alpha_0, \ldots, \alpha_{q-1})$ 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 $\lceil (2m-1)2^{m-4} \rceil$ bytes as above.

Each permutation has multiple choices of control-bit vectors. This document requires that a permutation π be converted to specifically the control bits defined by `controlbits` in Figure 1. The decapsulation algorithm *reading* control bits does not check uniqueness.

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

This sequence c is represented as a $\lceil \nu/8 \rceil$-byte string, the little-endian format of the integer

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

However, for $(\mu, \nu) = (0, 0)$, 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.

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

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

```
def permutation(c):
  m = 1
  while (2*m-1)<<(m-1) < len(c): m += 1
  assert (2*m-1)<<(m-1) == len(c)

  n = 1<<m
  pi = list(range(n))
  for i in range(2*m-1):
    gap = 1<<min(i,2*m-2-i)
    for j in range(n//2):
      if c[i*n//2+j]:
        pos = (j%gap)+2*gap*(j//gap)
        pi[pos],pi[pos+gap] = pi[pos+gap],pi[pos]

  return pi

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
```

Figure 1: Python functions to compute the permutation for an in-place Beneš network given control bits, and to compute control bits given a permutation.

7 Selected parameter sets

7.1 Parameter set `mceliece348864`

KEM with $m = 12$, $n = 3488$, $t = 64$. Field polynomials $f(z) = z^{12} + z^3 + 1$ and $F(y) = y^{64} + y^3 + y + z$.

7.2 Parameter set `mceliece348864f`

KEM with $m = 12$, $n = 3488$, $t = 64$. Field polynomials $f(z) = z^{12} + z^3 + 1$ and $F(y) = y^{64} + y^3 + y + z$. Semi-systematic parameters $(\mu, \nu) = (32, 64)$.

7.3 Parameter set `mceliece460896`

KEM with $m = 13$, $n = 4608$, $t = 96$. Field polynomials $f(z) = z^{13} + z^4 + z^3 + z + 1$ and $F(y) = y^{96} + y^{10} + y^9 + y^6 + 1$.

7.4 Parameter set `mceliece460896f`

KEM with $m = 13$, $n = 4608$, $t = 96$. 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 $(\mu, \nu) = (32, 64)$.

7.5 Parameter set `mceliece6688128`

KEM with $m = 13$, $n = 6688$, $t = 128$. Field polynomials $f(z) = z^{13} + z^4 + z^3 + z + 1$ and $F(y) = y^{128} + y^7 + y^2 + y + 1$.

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

7.7 Parameter set `mceliece6960119`

KEM with $m = 13$, $n = 6960$, $t = 119$. Field polynomials $f(z) = z^{13} + z^4 + z^3 + z + 1$ and $F(y) = y^{119} + y^8 + 1$.

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

7.9 Parameter set `mceliece8192128`

KEM with $m = 13$, $n = 8192$, $t = 128$. Field polynomials $f(z) = z^{13} + z^4 + z^3 + z + 1$ and $F(y) = y^{128} + y^7 + y^2 + y + 1$.

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