Classic McEliece: conservative code-based cryptography

10 October 2020

Principal submitter

This submission is from the following team, listed in alphabetical order:

- Martin R. Albrecht, Information Security Group, Royal Holloway, University of London
- Daniel J. Bernstein, University of Illinois at Chicago and Ruhr University Bochum
- Tung Chou, Academia Sinica
- Carlos Cid, Royal Holloway, University of London and Simula UiB
- Jan Gilcher, ETH Zürich
- Tanja Lange, Eindhoven University of Technology
- Varun Maram, ETH Zürich
- Ingo von Maurich, self
- Rafael Misoczki, Google
- Ruben Niederhagen, University of Southern Denmark
- Kenneth G. Paterson, ETH Zürich
- Edoardo Persichetti, Florida Atlantic University
- Christiane Peters, self
- Peter Schwabe, Max Planck Institute for Security and Privacy & Radboud University
- 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

E-mail address (preferred): `authorcontact-mceliece-merged@box.cr.yp.to`

Telephone (if absolutely necessary): +1-312-996-3422. Postal address (if absolutely necessary): Daniel J. Bernstein, Department of Computer Science, University of Illinois at Chicago, 851 S. Morgan (M/C 152), Room 1120 SEO, Chicago, IL 60607–7053.

Auxiliary submitters: There are no auxiliary submitters. The principal submitter is the team listed above.

Inventors/developers: The inventors/developers of this submission are the same as the principal submitter. Relevant prior work is credited below where appropriate.

Owner: Same as submitter.

Signature: $\times$. See also printed version of "Statement by Each Submitter".

Document generated with the help of `pqskeleton` version 20190309.

Contents

1 Introduction

The first code-based public-key cryptosystem was introduced in 1978 by McEliece [51]. 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 40 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:[1] this includes a "dual" PKE proposed by Niederreiter [56], software speedups such as [9], and hardware speedups such as [74].

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 [15] 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.

This submission *Classic McEliece* (CM) brings all of this together. It presents 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.

[1]Other work includes McEliece variants whose security has not been studied as thoroughly. For example, many proposals replace binary Goppa codes with other families of codes, and lattice-based cryptography replaces "codeword plus random errors" with "lattice point plus random errors". Code-based cryptography and lattice-based cryptography are two of the main types of candidates identified in NIST's call for Post-Quantum Cryptography Standardization. This submission focuses on the classic McEliece system precisely because of how thoroughly it has been studied.

2 General algorithm specification (part of 2.B.1)

2.1 Notation and parameters

2.1.1 Notation

The list below introduces the notation used in this section. 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	$(g, \alpha_1, \ldots, \alpha_n)$	(part of the private key)
s	A bit string of length n	(part of the private key)
T	An $(n-k) \times k$ matrix over $\mathbb{F}_2$	(the CM public key)
e	A bit string of length n and Hamming weight t	
C	A ciphertext encapsulating a session key	
C_0	A bit string of length $n-k$	(part of the ciphertext)
C_1	A bit string of length ℓ	(part of the ciphertext)

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.

2.1.2 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 $n \le q$.
- A positive integer $t \ge 2$ 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 $\nu \ge \mu \ge 0$ with $\nu \le k + \mu$. 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 $\sigma_1 \ge m$.
- An integer $\sigma_2 \ge 2m$.
- A pseudorandom bit generator G mapping a string of ℓ bits to a string of $n+\sigma_2 q+\sigma_1 t+\ell$ bits.

2.2 The one-way function

2.2.1 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 $c_1 < c_2 < \cdots < c_r$ such that

- row 1 of R begins with a 1 in column c_1, and this is the only 1 in column c_1;
- row 2 of R begins with a 1 in column c_2, the only 1 in column c_2;
- row 3 of R begins with a 1 in column c_3, the only 1 in column c_3;
- etc.;
- row r of R begins with a 1 in column c_r, the only 1 in column c_r; 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
- $c_1 = 1$, $c_2 = 2$, $c_3 = 3$, and so on through $c_r = r$. (This second condition is equivalent to simply saying $c_r = r$, 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. One way to do this is as follows:

- Use Gaussian elimination to compute R in reduced row-echelon form.
- Return R if R is in systematic form, else $\perp$.

Implementors should note that Gaussian elimination can be streamlined in this context by using early aborts. One can begin by trying to reduce the initial columns to triangular form; if the answer is $\perp$ then one can skip reducing these columns to an identity matrix, and one can skip the operations on the remaining columns. There must always be a nonzero entry in column 1 (or else the answer is $\perp$), then after elimination there must always be a nonzero entry in column 2 (or else the answer is $\perp$), etc.

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 $1 \leq i \leq r - \mu$; and $c_i \leq i - \mu + \nu$ for $1 \leq i \leq r$. (The c_i conditions are equivalent to simply $c_{r-\mu} = r - \mu$ and $c_r \leq r - \mu + \nu$ 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 $(\mu, \nu) = (32, 64)$. See Section 4.2 for an explanation of why the $(\mu, \nu) = (32, 64)$ case is of interest.

As in the special case of systematic form, one way to compute the (μ, ν)-semi-systematic form is to compute the reduced row-echelon form R, and then output R if R is in (μ, ν)-semi-systematic form. A more streamlined computation requires a nonzero entry in the first column, then after elimination requires a nonzero entry in the second column, and so on for the first $r - \mu$ columns; then computes the reduced row-echelon form of the next ν columns of the bottom μ rows, and requires this submatrix to have rank μ; and then completes the computation of reduced row-echelon form of the entire matrix.

2.2.2 Matrix generation for Goppa codes

The following algorithm MatGen takes as input $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ where

- g is a monic irreducible polynomial in $\mathbb{F}_q[x]$ of degree t and
- $\alpha_1, \alpha_2, \ldots, \alpha_n$ 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 $(n-k) \times k$ 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 $(n-k) \times k$ 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 $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, \ldots, t$ and $j = 1, \ldots, n$.
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 $(I_{n-k} \mid T)$, where I_{n-k} is an $(n-k) \times (n-k)$ identity matrix. If this fails, return $\perp$.
4. Return (T, Γ).

The input $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$, also provided as output, describes a binary Goppa code of length n and dimension $k = n - mt$. The public key T is a binary $(n-k) \times k$ matrix such that $H = (I_{n-k} \mid T)$ is a parity-check matrix for the same Goppa code.

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

- T is an $(n-k) \times k$ matrix over $\mathbb{F}_2$;
- $c_{n-k-\mu+1}, \ldots, c_{n-k}$ are integers with $n - k - \mu < c_{n-k-\mu+1} < c_{n-k-\mu+2} < \cdots < c_{n-k} \leq n - k - \mu + \nu$;
- $\Gamma' = (g, \alpha'_1, \alpha'_2, \ldots, \alpha'_n)$;
- g is the same as in the input; and
- $\alpha'_1, \alpha'_2, \ldots, \alpha'_n$ 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 $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, \ldots, t$ and $j = 1, \ldots, n$.
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 the ith row has its leading 1 in column c_i. By definition of semi-systematic form, $c_1 = 1$, $c_2 = 2$, and so on through $c_{n-k-\mu} = n - k - \mu$; and $n - k - \mu < c_{n-k-\mu+1} < c_{n-k-\mu+2} < \cdots < c_{n-k} \leq n - k - \mu + \nu$. The matrix H is a variable that can change later.)
4. Set $(\alpha'_1, \alpha'_2, \ldots, \alpha'_n) \leftarrow (\alpha_1, \alpha_2, \ldots, \alpha_n)$. (Each α'_i is a variable that can change later.)
5. For $i = n-k-\mu+1$, then $i = n-k-\mu+2$, and so on through $i = n-k$, in this order: swap column i with column c_i in H, while swapping α'_i with α'_{c_i}. (After the swap, the ith row has its leading 1 in column i. The swap does nothing if $c_i = i$.)
6. The matrix H now has systematic form $(I_{n-k} \mid T)$, where I_{n-k} is an $(n-k) \times (n-k)$ identity matrix. Return $(T, c_{n-k-\mu+1}, \ldots, c_{n-k}, \Gamma')$ where $\Gamma' = (g, \alpha'_1, \alpha'_2, \ldots, \alpha'_n)$.

By construction $\{\alpha'_1, \alpha'_2, \ldots, \alpha'_n\} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, since $(\alpha'_1, \alpha'_2, \ldots, \alpha'_n)$ is obtained by a series of swaps from $(\alpha_1, \alpha_2, \ldots, \alpha_n)$. More precisely, swaps take place only on indices between $n-k-\mu+1$ and $n-k-\mu+\nu$, so $\alpha'_i = \alpha_i$ for $i \leq n-k-\mu$; $\alpha'_i = \alpha_i$ for $i > n-k-\mu+\nu$; and $\{\alpha'_{n-k-\mu+1}, \ldots, \alpha'_{n-k-\mu+\nu}\} = \{\alpha_{n-k-\mu+1}, \ldots, \alpha_{n-k-\mu+\nu}\}$.

As before, the input $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ describes a binary Goppa code of length n and dimension $k = n - mt$. The output $\Gamma' = (g, \alpha'_1, \alpha'_2, \ldots, \alpha'_n)$ also describes a binary Goppa code of length n and dimension k, a permuted version of the first code. The public key T is a binary $(n-k) \times k$ matrix such that $H = (I_{n-k} \mid T)$ is a parity-check matrix for the second code.

In the special case $(\mu, \nu) = (0, 0)$, the $c_{n-k-\mu+1}, \ldots, c_{n-k}$ 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.

2.2.3 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 $(n-k) \times k$ matrix over $\mathbb{F}_2$. The algorithm output Encode(e, T) is a vector $C_0 \in \mathbb{F}_2^{n-k}$. Here is the algorithm:

1. Define $H = (I_{n-k} \mid T)$.
2. Compute and return $C_0 = He \in \mathbb{F}_2^{n-k}$.

2.2.4 Decoding subroutine

The following algorithm DECODE decodes $C_0 \in \mathbb{F}_2^{n-k}$ to a word e of Hamming weight $\mathsf{wt}(e) = t$ with $C_0 = He$ if such a word exists; otherwise it returns failure.

Formally, DECODE takes two inputs: a vector $C_0 \in \mathbb{F}_2^{n-k}$; and Γ', the last component of MATGEN(Γ) for some Γ such that MATGEN(Γ) $\neq \perp$. Write T for the first component of MATGEN(Γ). By definition of MATGEN,

- T is an $(n-k) \times k$ matrix over $\mathbb{F}_2$;
- Γ' has the form $(g, \alpha'_1, \alpha'_2, \ldots, \alpha'_n)$;
- g is a monic irreducible polynomial in $\mathbb{F}_q[x]$; and
- $\alpha'_1, \alpha'_2, \ldots, \alpha'_n$ are distinct elements of $\mathbb{F}_q$.

There are two possibilities for DECODE(C_0, Γ'):

- If $C_0 =$ ENCODE(e, T) then DECODE(C_0, Γ') $= e$. In other words, if there exists a weight-t vector $e \in \mathbb{F}_2^n$ such that $C_0 = He$ with $H = (I_{n-k} \mid T)$, then DECODE(C_0, Γ') $= e$.
- If C_0 does not have the form He for any weight-t vector $e \in \mathbb{F}_2^n$, then DECODE(C_0, Γ') $= \perp$.

Here is the algorithm:

1. Extend C_0 to $v = (C_0, 0, \ldots, 0) \in \mathbb{F}_2^n$ by appending k zeros.
2. Find the unique codeword c in the Goppa code defined by Γ' that is at distance $\leq t$ from v. If there is no such codeword, return $\perp$.
3. Set $e = v + c$.
4. If $\mathsf{wt}(e) = t$ and $C_0 = He$, return e. Otherwise return $\perp$.

There are several well-known algorithms for Step 2. For references and speedups see generally [9] and [24].

To see why DECODE works, note first that the "syndrome" Hv is C_0, because the first $n-k$ positions of v are multiplied by the identity matrix and the remaining positions are zero. If C_0 has the form He where e has weight t then $Hv = He$, so $c = v + e$ is a codeword. This codeword has distance exactly t from v, and it is the unique codeword at distance $\leq t$ from v since the minimum distance of Γ' is at least $2t + 1$. Hence Step 2 finds c, Step 3 finds e, and Step 4 returns e. Conversely, if DECODE returns e in Step 4 then e has been verified to have weight t and to have $C_0 = He$, so if C_0 does not have this form then DECODE must return $\perp$.

The logic here relies on Step 2 always finding a codeword at distance t if one exists. It does not rely on Step 2 failing in the cases that a codeword does not exist: DECODE remains correct if, instead of returning $\perp$, Step 2 chooses some vector $c \in \mathbb{F}_2^n$ and continues on to

Step 3.

Implementors are cautioned that it is important to avoid leaking secret information through side channels, and that the distinction between success and failure of DECODE is secret in the context of the Classic McEliece KEM. In particular, immediately stopping the computation when Step 2 returns $\perp$ would reveal this distinction through timing, so it is recommended for implementors to have Step 2 always choose some $c \in \mathbb{F}_2^n$.

The DECODE definition refers to H, which one can compute via $\text{MATGEN}(\Gamma') = (T, \ldots)$. However, this recomputation is not necessary. In order to test $C_0 = He$, implementors can use any parity-check matrix H' for the same code. The computation uses $v = (C_0, 0, \ldots, 0)$ and compares $H'v$ to $H'e$. The results are equal if and only if $v + e = c$ is a codeword, which implies $He = H(v + c) = Hv + Hc = Hv = C_0$. There are various well-known choices of H' related to $\hat{H}$ that are recovered from Γ' much more efficiently than MATGEN, and that can be applied to vectors without using quadratic space.

2.3 The Model Classic McEliece KEM

We first introduce the Model Classic McEliece KEM, which is not the actual Classic McEliece KEM but is used as a stepping-stone in the analysis of Classic McEliece. The reasons for defining these two KEMs are explained in detail in Section 4.4.

Model Classic McEliece uses MODELKEYGEN from Section 2.3.1 for key generation and MODELENCAP from Section 2.3.2 for encapsulation. Classic McEliece uses KEYGEN from Section 2.4.3 for key generation and ENCAP from Section 2.4.5 for encapsulation. Model Classic McEliece and Classic McEliece both use DECAP from Section 2.3.3 for decapsulation.

2.3.1 Model key generation

The following randomized algorithm MODELKEYGEN takes no input (beyond the parameters). It outputs a public key and private key. Here is the algorithm:

1. Generate a uniform random n-bit string s.
2. Generate a uniform random monic irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t.
3. Generate a uniform random sequence $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n distinct elements of $\mathbb{F}_q$.
4. Define $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$.
5. Compute $(T, c_{n-k-\mu+1}, \ldots, c_{n-k}, \Gamma') = \text{MATGEN}(\Gamma)$. If this fails, restart the algorithm. (Recall that $\Gamma' = \Gamma$ in the case $(\mu, \nu) = (0, 0)$.)
6. Output T as the public key, and (Γ', s) as the private key.

2.3.2 Model encapsulation

The following randomized algorithm ModelEncap takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Generate a uniform random vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \textsc{Encode}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

2.3.3 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. Split the ciphertext C as (C_0, C_1) with $C_0 \in \mathbb{F}_2^{n-k}$ and $C_1 \in \mathbb{F}_2^{\ell}$.
2. Set $b \leftarrow 1$.
3. Extract $s \in \mathbb{F}_2^n$ and $\Gamma' = (g, \alpha_1', \alpha_2', \ldots, \alpha_n')$ from the private key.
4. Compute $e \leftarrow \textsc{Decode}(C_0, \Gamma')$. If $e = \perp$, set $e \leftarrow s$ and $b \leftarrow 0$.
5. Compute $C_1' = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings.
6. If $C_1' \neq C_1$, set $e \leftarrow s$ and $b \leftarrow 0$.
7. Compute $K = \mathsf{H}(b, e, C)$; see Section 2.5.2 for H input encodings.
8. Output session key K.

If C is a legitimate ciphertext then $C = (C_0, C_1)$ with $C_0 = He$ for some $e \in \mathbb{F}_2^n$ of weight t and $C_1 = \mathsf{H}(2, e)$. The decoding algorithm will return e as the unique weight-t vector and the $C_1' = C_1$ check will pass, thus $b = 1$ and K matches the session key computed in encapsulation.

As an implementation note, the output of decapsulation is unchanged if "$e \leftarrow s$" in Step 4 is changed to assign something else to e. Implementors may prefer, e.g., to set e to a fixed n-bit string, or a random n-bit string other than s. However, the definition of decapsulation does depend on e being set to s in Step 6.

Implementors are again cautioned that it is important to avoid leaking secret information through side channels. In particular, the distinction between failures in Step 4, failures in Step 6, and successes is secret information, and branching would leak this information through timing. It is recommended for implementors to always go through the same sequence of computations, using arithmetic to simulate tests and conditional assignments.

2.4 The Classic McEliece KEM

2.4.1 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 $\perp$.

2.4.2 Field-ordering generation

The following algorithm FieldOrdering takes a string of $\sigma_2 q$ input bits. It outputs either $\perp$ or a sequence $(\alpha_1, \alpha_2, \ldots, \alpha_q)$ 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 $\perp$. (Implementors can merge this test into the following sorting step.)
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
$$\alpha_{i+1} = \sum_{j=0}^{m-1} \pi(i)_j \cdot z^{m-1-j}$$
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 $(\alpha_1, \alpha_2, \ldots, \alpha_q)$.

2.4.3 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 $\alpha_1, \dots, \alpha_q$ 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 $\Gamma = (g, \alpha_1, \alpha_2, \dots, \alpha_n)$. (Note that $\alpha_{n+1}, \dots, \alpha_q$ are not used here.)
7. Compute $(T, c_{n-k-\mu+1}, \dots, c_{n-k}, \Gamma') \leftarrow$ MatGen(Γ). If this fails, set $\delta \leftarrow \delta'$ and restart the algorithm.
8. Write Γ' as $(g, \alpha'_1, \alpha'_2, \dots, \alpha'_n)$.
9. Output T as public key and $(\delta, c, g, \alpha, s)$ as private key, where $c = (c_{n-k-\mu+1}, \dots, c_{n-k})$ and $\alpha = (\alpha'_1, \dots, \alpha'_n, \alpha_{n+1}, \dots, \alpha_q)$.

2.4.4 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 $b_0, b_1, \dots, b_{\sigma_1 \tau - 1}$.
2. Define $d_j = \sum_{i=0}^{m-1} b_{\sigma_1 j + i} 2^i$ for each $j \in \{0, 1, \dots, \tau - 1\}$.
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 such entries, 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. (Implementors are cautioned to compute e through arithmetic rather than variable-time RAM lookups.)
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 our selected parameters have $q/2 \leq n \leq q$, so $\tau \in \{t, 2t\}$.

2.4.5 Encapsulation

The following randomized algorithm $\textsc{Encap}$ takes as input a public key T. It outputs a ciphertext C and a session key K. Here is the algorithm:

1. Use $\textsc{FixedWeight}$ to generate a vector $e \in \mathbb{F}_2^n$ of weight t.
2. Compute $C_0 = \textsc{Encode}(e, T)$.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.5.2 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.5.2 for H input encodings.
5. Output ciphertext C and session key K.

This is identical to $\textsc{ModelEncap}$ except in how the vector e is generated.

2.5 Bits and bytes

2.5.1 Choices of symmetric-cryptography parameters

All of our 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 2.5.2.
- The integer σ_1 is 16. (All of the parameter sets have $m \le 16$, so $\sigma_1 \ge m$.)
- 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 or 2 (see Section 2.5.2), and thus do not overlap the SHAKE256 inputs used in G.

2.5.2 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.

We define Simply Decoded Classic McEliece as ignoring padding bits on input, and Narrowly Decoded Classic McEliece as rejecting inputs (ciphertexts and public keys) where padding bits are nonzero. These are equivalent for some parameter sets but not for all; see Section 4.3. Our software implements Narrowly Decoded Classic McEliece.

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 has two components: $C_0 \in \mathbb{F}_2^{n-k}$ and $C_1 \in \mathbb{F}_2^\ell$. The ciphertext is represented as the concatenation of the $\lceil mt/8 \rceil$-byte string representing C_0 and the $\lceil \ell/8 \rceil$-byte string representing C_1.

Hash inputs. 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 byte. The vector v is represented as the next $\lceil n/8 \rceil$ bytes. The ciphertext, if present, is represented as the next $\lceil mt/8 \rceil + \lceil \ell/8 \rceil$ bytes.

All hash inputs thus begin with byte 0, 1, or 2, 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 $(\alpha_1, \ldots, \alpha_q)$ of q distinct elements of $\mathbb{F}_q$ would be as a sequence of q field elements. We specify a different representation that simplifies fast constant-time decoding algorithms.

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 $\alpha_{i+1} = \sum_{j=0}^{m-1} \pi(i)_j \cdot z^{m-1-j}$ for all $i \in \{0, 1, \ldots, q-1\}$. The ordering $(\alpha_1, \ldots, \alpha_q)$ 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. To simplify testing, we require that a permutation π be converted to specifically the control bits defined in [8] for π. Software *reading* control bits does not check uniqueness.

As low-cost protection against faults in computing the control bits for π, implementors are advised to check after the computation that applying the Beneš network produces π, and to restart key generation if this test fails.

Column selections. Part of the private key generated by KeyGen is a sequence $c = (c_{n-k-\mu+1}, \ldots, c_{n-k})$ of μ integers in increasing order between $n-k-\mu+1$ and $n-k-\mu+\nu$. (This is not used in the algorithms here, but supports compression as explained in Section 4.4.)

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

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

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.

The special handling of $(\mu, \nu) = (0, 0)$ is designed so that a private key using $(\mu, \nu) = (0, 0)$ is compatible with decapsulation software using $(\mu, \nu) = (32, 64)$, when all other parameters are the same.

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 $s \in \mathbb{F}_2^n$.

3 List of parameter sets (part of 2.B.1)

3.1 Parameter set `kem/mceliece348864`

KEM with $m = 12$, $n = 3488$, $t = 64$. Field polynomials $f(z) = z^{12} + z^3 + 1$ and $F(y) = y^{64} + y^3 + y + z$. This parameter set is **proposed and implemented** in this submission.

3.2 Parameter set `kem/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 $(\mu, \nu) = (32, 64)$. This parameter set is **proposed and implemented** in this submission.

3.3 Parameter set `kem/mceliece460896`

KEM with $m = 13$, $n = 4608$, $t = 96$. Field polynomials $f(z) = z^{13} + z^4 + z^3 + z + 1$ and $F(y) = y^{96} + y^{10} + y^9 + y^6 + 1$. This parameter set is **proposed and implemented** in this submission.

3.4 Parameter set `kem/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 $(\mu, \nu) = (32, 64)$. This parameter set is **proposed and implemented** in this submission.

3.5 Parameter set `kem/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 parameter set is **proposed and implemented** in this submission.

3.6 Parameter set `kem/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 $(\mu, \nu) = (32, 64)$. This parameter set is **proposed and implemented** in this submission.

3.7 Parameter set `kem/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 parameter set is **proposed and implemented** in this submission.

3.8 Parameter set `kem/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 $(\mu, \nu) = (32, 64)$. This parameter set is **proposed and implemented** in this submission.

3.9 Parameter set `kem/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 parameter set is **proposed and implemented** in this submission.

3.10 Parameter set `kem/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 $(\mu, \nu) = (32, 64)$. This parameter set is **proposed and implemented** in this submission.

4 Design rationale (part of 2.B.1)

4.1 One-wayness

There is a long history of trapdoor systems (in modern terminology: PKEs) that are designed to be one-way (in modern terminology: OW-CPA). One-wayness means that it is difficult to invert the map from input to ciphertext, given the public key, when the input is chosen uniformly at random.

The McEliece system is one of the oldest proposals, almost as old as RSA. RSA has suffered dramatic security losses, while the McEliece system has maintained a spectacular security track record unmatched by any other proposals for post-quantum encryption. This is the reason that we have chosen to submit the McEliece system.

Here is more detail to explain what we mean by “spectacular security track record”.

With the key-size optimizations discussed below, the McEliece system uses a key size of $(c_0 + o(1))b^2(\lg b)^2$ bits to achieve 2^b security against all inversion attacks that were known in 1978, when the system was introduced. Here lg means logarithm base 2, $o(1)$ means something that converges to 0 as $b \to \infty$, and $c_0 \approx 0.7418860694$.

The best attack at that time was from 1962 Prange [61]. After 1978 there were 27 publications studying the one-wayness of the system and introducing increasingly sophisticated non-quantum attack algorithms:

1. 1981 Clark–Cain [25], crediting Omura.
2. 1988 Lee–Brickell [46].
3. 1988 Leon [47].
4. 1989 Krouk [42].
5. 1989 Stern [67].
6. 1989 Dumer [31].
7. 1990 Coffey–Goodman [26].
8. 1990 van Tilburg [71].
9. 1991 Dumer [32].
10. 1991 Coffey–Goodman–Farrell [27].
11. 1993 Chabanne–Courteau [22].
12. 1993 Chabaud [23].
13. 1994 van Tilburg [72].
14. 1994 Canteaut–Chabanne [18].
15. 1998 Canteaut–Chabaud [19].
16. 1998 Canteaut–Sendrier [20].
17. 2008 Bernstein–Lange–Peters [11].
18. 2009 Bernstein–Lange–Peters–van Tilborg [13].
19. 2009 Finiasz–Sendrier [34].
20. 2011 Bernstein–Lange–Peters [12].
21. 2011 May–Meurer–Thomae [49].
22. 2012 Becker–Joux–May–Meurer [3].
23. 2013 Hamdaoui–Sendrier [37].
24. 2015 May–Ozerov [50].
25. 2016 Canto Torres–Sendrier [69].

26. 2017 Both–May [16].

27. 2018 Both–May [17].

What is the cumulative impact of all this work? Answer: With the same key-size optimizations, the McEliece system uses a key size of $(c_0 + o(1))b^2(\lg b)^2$ bits to achieve 2^b security against all non-quantum attacks known today, where c_0 is exactly the same constant. All of the improvements have disappeared into the $o(1)$.

This does not mean that the required key size is precisely the same—that dozens of attack papers over 40 years have not accomplished *anything*. What it means is that the required change in key size is below 1% once b is large enough; below 0.1% once b is large enough; etc. This is a remarkably stable security story.

What about quantum attacks? Grover's algorithm is applicable, reducing the attack cost to asymptotically its square root; see generally [5]. In other words, the key now needs $(4c_0 + o(1))b^2(\lg b)^2$ bits. As before, further papers on the topic have merely improved the $o(1)$.

All of the papers mentioned above are focusing on the most effective attack strategy known, namely "information-set decoding". This strategy does not exploit any particular structure of a generator matrix G: it recovers a low-weight error vector e given a *uniform random* matrix G and $Gm + e$ for some m. Experiments are consistent with the theory that McEliece's matrices G behave like uniform random matrices in this context.

There are also many papers studying attacks that instead recover McEliece's private key from the public key G. Recovering the private key also breaks one-wayness, since the attacker can then use the receiver's decryption algorithm. These attacks can be much faster than a brute-force search through private keys: for example, Sendrier's "support splitting" algorithm [63] quickly finds $\alpha_1, \ldots, \alpha_n$ given g provided that $n = q$. More generally, whether or not $n = q$, support splitting finds $\alpha_1, \ldots, \alpha_n$ given g and given the *set* $\{\alpha_1, \ldots, \alpha_n\}$. (This can be viewed as a reason to keep n somewhat smaller than q, since then there are many possibilities for the set, along with many possibilities for g; most of our proposed parameter sets provide this extra defense.) However, despite this and other interesting speedups, the state-of-the-art key-recovery attacks are vastly slower than information-set decoding.

Various authors have proposed replacing the binary Goppa codes in McEliece's system with other families of codes: see, e.g., [2, 4, 52, 56, 58, 53]. Often these replacements are advertised as allowing smaller public keys. Unfortunately, many of these proposals have turned out to allow unacceptably fast recovery of the private key (or of something equivalent to the private key, something that allows fast inversion of the supposedly one-way function). Some small-key proposals are unbroken, but in this submission we focus on binary Goppa codes as the traditional, conservative, well-studied choice.

Authors of attacks on other codes often study whether binary Goppa codes are affected by their attacks. These studies consistently show that McEliece's system is far beyond all known attacks. For example, 2013 Faugère–Gauthier-Umaña–Otmani–Perret–Tillich [33] showed that "high-rate" binary Goppa codes can be distinguished from random codes. The worst-case possibility is that this distinguisher somehow allows an inversion attack faster than attacks for random codes. However, the distinguisher stops working

- at 8 errors for $n = 1024$ (where McEliece's original parameters used 50 errors),
- at 20 errors for $n = 8192$ (where our proposed parameters use between 96 and 128 errors),

etc. As another example, the attack in [28] reaches degree $m = 2$ where McEliece's original parameters used degree $m = 10$ and where our proposed parameters use degree $m = 12$ or $m = 13$.

4.2 Better efficiency for the same one-wayness

The main focus of this submission is security, but we also take reasonable steps to improve efficiency when this clearly does not compromise security. In particular, we make the following two well-known modifications.

First modification. The goal of the public key in McEliece's system is to communicate an $[n, k]$ linear code C over $\mathbb{F}_2$: a k-dimensional linear subspace of $\mathbb{F}_2^n$. This means communicating the ability to generate uniform random elements of C. McEliece accomplished this by choosing the public key to be a uniform random generator matrix G for C: specifically, multiplying any generator matrix for C by a uniform random invertible matrix.

The first modification accomplishes this by instead choosing the public key to be the unique systematic-form generator matrix for C if one exists. This means a generator matrix of the form $\left(\frac{T}{I_k}\right)$ where T is some $(n-k) \times k$ matrix and I_k is the $k \times k$ identity matrix. Approximately 29% of choices of C have this form, so key generation requires about 3.4 attempts on average, but now the public key occupies only $k(n-k)$ bits instead of kn bits. Note that sending a systematic-form generator matrix also implies sending a parity-check matrix H for C, namely $(I_{n-k} \mid T)$.

Any attack against the limited set of codes allowed here implies an attack with probability 29% against the full set of codes allowed by McEliece; this is a security difference of at most 2 bits. Furthermore, any attack against the systematic-form public key can be used to attack any generator matrix for the same code, and in particular McEliece's public key, since anyone given any generator matrix can quickly compute the systematic-form public key by linear algebra.

Second modification. McEliece's ciphertext has the form $Ga + e$. Here G is a random $n \times k$ generator matrix for a code C as above; a is a column vector of length k; e is a weight-t column vector of length n; and the ciphertext is a column vector of length n. McEliece's inversion problem is to compute (a, e) given G and the ciphertext $Ga + e$, where a is a uniform random column vector of length k; e is a uniform random weight-t column vector of length n; and a, e are independent.

Niederreiter [56] instead suggested a ciphertext of the form He. Here H is a parity-check matrix for C used as a public key, and e is a weight-t column vector of length n, so the ciphertext is a column vector of length just $n - k$, shorter than McEliece's ciphertext. Niederreiter's inversion problem is to compute e given H and the ciphertext He, where e is a uniform random weight-t vector of length n.

For any distribution of parity-check matrices H publicly computable from the code C (e.g., the unique systematic-form parity-check matrix $(I_{n-k} \mid T)$ defined above), Niederreiter's inversion problem is equivalent to McEliece's inversion problem for the same code. In particular, any attack recovering e from Niederreiter's He and H can be used with negligible overhead to recover (a, e) from McEliece's $Ga + e$ and G. Specifically, given $Ga + e$ and G, compute a parity-check matrix H for the same code, multiply H by $Ga + e$ to obtain $HGa + He = He$, apply the attack to recover e from He, subtract e from $Ga + e$ to obtain Ga, and recover a by linear algebra.

Semi-systematic form, continued. As a generalization (introduced by Chou) of the idea of systematic form, we consider any key obtained as follows:

- Starting from the secret parity-check matrix for the code C, compute the unique parity-check matrix in reduced row-echelon form.
- Start over with a new code if this matrix is not acceptable. This generalization is parameterized by the definition of acceptability: e.g., one can define an acceptable matrix as a matrix in (μ, ν)-semi-systematic form.
- Permute the matrix columns to reach systematic form, while permuting the code accordingly. This requires all acceptable matrices to have full rank.

It is important here for the second and third steps to depend only on the reduced row-echelon form. This guarantees that any attack against the resulting public key can be converted into an attack against McEliece's public key: anyone can convert McEliece's public key into the parity-check matrix in reduced row-echelon form, and then follow the second and third steps.

Accepting only systematic-form matrices—i.e., $(0, 0)$-semi-systematic-form matrices—is the simplest possibility, making implementations as easy as possible to write and audit. One can argue that accepting more matrices produces a tighter security proof, but the original tightness loss was at most 2 bits. The primary argument for accepting more matrices is a performance argument, namely that this increases the success probability of each key-generation attempt.

Accepting *any* full-rank matrix maximizes the success probability. On the other hand, the analysis in [35] suggests that constant-time implementations of the first step will then be very slow. Presumably this means that the overall key-generation time will be slower on average, despite the improved success probability.

The concept of (μ, ν)-semi-systematic form is designed to take both the time and the success probability into account. Compared to $(\mu, \nu) = (0, 0)$, a small increase in μ and $\nu - \mu$ reduces and stabilizes the number of key-generation attempts. It is reasonable to estimate, for example, that $(\mu, \nu) = (32, 64)$ reduces the failure probability of each attempt below 2^{-30}, so most of the time one needs only 1 key-generation attempt. This attempt requires extra work for a constant-time echelon-form computation, but only within ν columns, which is not a large issue when ν is kept reasonably small.

We have three reasons for continuing to propose $(0, 0)$-semi-systematic-form computations. First, we also speed up these computations, skipping most of the work in Gaussian elimination in the failure cases and thus reducing the average key-generation time. Second, it is not clear how often users will generate new keys, and as a result it is not clear how much users will care about the speedups from $(32, 64)$-semi-systematic form. Third, there is value in simplicity.

4.3 Indistinguishability against chosen-ciphertext attacks

Assume that McEliece's system is one-way. Niederreiter's system is then also one-way: the attacker cannot efficiently compute a uniform random weight-t vector e given Niederreiter's public key H and the ciphertext He.

What the user actually needs is more than one-wayness. The user is normally sending a plaintext with structure, perhaps a plaintext that can simply be guessed. Furthermore, the attacker can try modifying ciphertexts to see how the receiver reacts. McEliece's original PKE was not designed to resist, and does not resist, such attacks. In modern terminology, the user needs IND-CCA2 security.

There is a long literature studying the IND-CCA2 security of various PKE constructions, and in particular constructions built from an initial PKE assumed to have OW-CPA security. An increasingly popular simplification here is to encrypt the user's plaintext with an authenticated cipher such as AES-GCM. The public-key problem is then simply to send an unpredictable session key to use as the cipher key. Formally, our design goal here is to build a KEM with IND-CCA2 security; "KEM-DEM" composition [29] then produces a PKE with IND-CCA2 security, assuming a secure DEM. More complicated PKE constructions can pack some plaintext bytes into the ciphertext but are more difficult to audit and would be contrary to our goal of producing high confidence in security.

For our KEM construction we follow the best practices established in the literature:

- We use a uniform random PKE input e. We compute the session key as a hash of e.
- Our ciphertext is the original ciphertext plus a "confirmation": another cryptographic hash of e.
- After using the private key to compute e from a ciphertext, we recompute the ciphertext (including the confirmation) and check that it matches.
- If decryption fails (i.e., if computing e fails or the recomputed ciphertext does not

match), we do not return a KEM failure: instead we return a pseudorandom function of the ciphertext, specifically a cryptographic hash of a separate private key and the ciphertext.

We use a standard, thoroughly studied cryptographic hash function. We ensure that the three hashes mentioned above are obtained by applying this function to input spaces that are visibly disjoint. We choose the input details to simplify implementations that run in constant time, in particular not leaking whether decryption failed.

There are intuitive arguments for these practices, and the same practices already had benefits for partial security proofs available at the time of our round-1 submission:

- A KEM construction published in a classic 2003 paper by Dent [30, Section 6] features a tight proof of security against ROM attacks, assuming OW-CPA security of the underlying PKE. This theorem relies on the first three items in the list above.
- A much more recent KEM construction by Saito, Xagawa, and Yamakawa [62] features a tight proof of security against the broader class of QROM attacks, under somewhat stronger assumptions. This theorem relies on the first, third, and fourth items.

Both theorems also rely on two PKE features that are provided by the PKE we use: the ciphertext is a *deterministic* function of the input e, and there are no decryption failures for legitimate ciphertexts. At the time of our round-1 submission, the theorems stated in the literature did not apply directly to our KEM construction, but we included a preliminary analysis indicating that the proof ideas do apply, and subsequent analysis confirmed this; see Section 6. The deterministic PKE, the fact that decryption always works for legitimate ciphertexts, and the overall simplicity of the KEM construction should make it possible to formally verify complete proofs, building further confidence.

IND-CCA2 for encodings. Ciphertexts are normally encoded as byte strings and then further encoded as objects in higher-level protocols. Encodings in network protocols and in other applications are, in general, not unique: there are many objects that will decode to the same ciphertext.

Shoup [66] refers to this non-uniqueness property as “benign malleability”. It is possible to construct applications where this property loses security. There is a debate in the literature as to whether this should be addressed

- in applications, by the rule of acting on encoded ciphertexts solely by decoding and decapsulating them, or
- in decoders, by the rule of enforcing unique encodings for ciphertexts.

A generic reencoding wrapper converts a deterministic decoder (and deterministic encoder) into a decoder following the second rule, by rejecting any encoded ciphertext $\underline{C}$ different from the encoding of the decoding of $\underline{C}$. A similar wrapper that rewrites ciphertexts, without rejecting them, converts an application into an application following the first rule.

Given any encoding, one can extend the definition of IND-CCA2 for ciphertexts into a definition of IND-CCA2 for encoded ciphertexts. The rule of enforcing unique encodings makes the second definition equivalent to the first. The rule of acting on encoded ciphertexts solely by decoding and decapsulating them makes the second definition unnecessary.

The literature contains many other extensions of IND-CCA2. There are arguments that the extended properties are easy for cryptosystems to achieve and can protect applications. There are counterarguments saying that the same security can be achieved in a simpler way by another layer of the system. Consider, e.g., Shoup's argument in [66, Section 3.3] that KEMs should avoid hashing "labels" such as identities since "it is easier to implement labels in the data encapsulation mechanism".

Encodings as byte strings are within the scope of Classic McEliece. To provide clarity for applications that want to enforce unique encodings as byte strings, we distinguish between

- Narrowly Decoded Classic McEliece, which requires padding bits (not just for encoded ciphertexts but also for encoded public keys) to be 0 on decoding, and
- Simply Decoded Classic McEliece, which ignores padding bits on decoding.

A wrapper that requires the padding bits to be 0 converts an implementation of Simply Decoded Classic McEliece into an implementation of Narrowly Decoded Classic McEliece. A wrapper that clears the padding bits converts an implementation of Narrowly Decoded Classic McEliece into an implementation of Simply Decoded Classic McEliece.

Simply Decoded Classic McEliece and Narrowly Decoded Classic McEliece are equivalent for our selected non-`6960` parameter sets, since only the `6960` parameter sets use padding bits. For the `6960` parameter sets, our software implements Narrowly Decoded Classic McEliece by checking the appropriate bits of public keys and ciphertexts. This check is handled in constant time, for applications where "public" keys and ciphertexts are actually confidential (e.g., obtained as outputs of another layer of decryption). In case applications fail to check return values, the encapsulation software sets all bits to 0 in its ciphertext and session-key output buffers in case of bad padding, and the decapsulation software sets all bits to 1 in its session-key output buffer in case of bad padding. The difference between 0 and 1 here is designed so that a cascade of several possible failures (bad padding in a public key, ignoring the encapsulation failure, bad padding in a ciphertext, and ignoring the decapsulation failure) will produce two different session keys that will not interoperate in typical DEMs, increasing the chance of the failures being caught by tests.

Further malleability is possible in private keys, not just because of padding bits but because there are multiple sequences of control bits that represent the same permutation. Also, modifying one bit in a public key has a significant chance of not affecting any particular ciphertext, and various linear-algebra operations on public keys have predictable effects on ciphertexts. These obvious properties of Classic McEliece have no effect on our IND-CCA2 security goal, but one can make other definitions that are affected by these properties. Application designers are encouraged to assume solely the standard IND-CCA2 property, and in any case to be clear regarding the properties that they assume.

4.4 Generation of random objects

A widely deployed RSA prime-generation algorithm was broken by ROCA [55]. Nothing was shown to be wrong with the underlying source of random bytes, or the primality of each output p, or the interval containing p, or the entropy of p, namely 256 bits. The problem was that, for efficiency, the algorithm generated primes with a special structure that could be exploited by the attacker.

This algorithm was compliant with RSA specifications that asked for "random" primes p in an interval. It was not compliant with RSA specifications that asked for "uniform random" primes p in an interval, since the distribution of p was not uniform, but formally such specifications prohibit almost all RSA implementations: any PRNG converting (say) 256 bits of entropy into a 1024-bit prime p is producing a non-uniform output distribution.

PRNGs are good for testability, so standards should allow cryptographic modules to generate a prime p from a PRNG. How does the cryptographic module validator assess whether the prime p is sufficiently random?

If RSA security reviewers have considered uniform random private keys (p, q), then there is no loss of security from any distribution of (p, q) that is indistinguishable from uniform. It therefore suffices for the validator to ask whether the algorithm to generate (p, q) is generating a distribution indistinguishable from uniform. A weaker divergence property (see, e.g., [7]) suffices for "search" security properties such as signature security and OW-CPA.

The standard PRNG objective is to generate a byte string indistinguishable from a uniform random byte string. This is analogous to generating primes indistinguishable from uniform random primes, but it is not the same. To close the gap, some standards specify algorithms to deterministically convert byte strings into private keys (p, q); see, e.g., [54, Appendix B.3.3]. The security goal for such an algorithm is to convert a uniform random byte string into (p, q) indistinguishable from uniform. If the algorithm meets this goal, and is applied to a byte string indistinguishable from uniform, then it produces (p, q) indistinguishable from uniform. A weaker divergence property again suffices for some applications.

This structure means that there is a process of specifying, reviewing, and approving algorithms to generate RSA keys: not just tests for primality, but complete algorithms to convert random bytes into private keys. The cryptographic module validator checks whether the implementation is using an approved private-key-generation algorithm and an approved source of random bytes.

Random objects in Classic McEliece. The same issues arise in post-quantum cryptography. In particular, key generation in the Model Classic McEliece KEM, defined using ModelKeyGen from Section 2.3.1, asks for a uniform random sequence $(\alpha_1, \ldots, \alpha_n)$ of n distinct elements of $\mathbb{F}_q$, and a uniform random monic irreducible polynomial g of degree t. Even if an implementation is using an approved source of random bytes, how does the cryptographic module validator assess whether an implementation is generating a sufficiently random sequence $(\alpha_1, \ldots, \alpha_n)$ and a sufficiently random g?

To answer these questions, we define deterministic algorithms Irreducible and FieldOrdering designed to convert uniform random byte strings into sufficiently random g and $(\alpha_1, \ldots, \alpha_q)$ respectively, and on top of this we define a deterministic algorithm SeededKeyGen that converts a 32-byte seed into a private key. We then define KeyGen for the Classic McEliece KEM to apply SeededKeyGen to a uniform random 32-byte seed.

Specifying ModelKeyGen gives a simple definition of the Model Classic McEliece KEM for review of the IND-CCA2 security property. A separate review of the relationship between KeyGen and ModelKeyGen then transports the IND-CCA2 security property from the Model Classic McEliece KEM to the Classic McEliece KEM. The cryptographic module validator then checks that an implementation is correctly implementing SeededKeyGen and is starting from an approved source of 32 random bytes for KeyGen.

This structure is compatible with specifying, reviewing, and approving future alternatives to SeededKeyGen, for example because performance analysis finds faster secure key-generation methods.

Compression of private keys. Classic McEliece private keys are much smaller than public keys, but there may be interest in compressing them further.

The KeyGen structure explained above, deterministically mapping a 32-byte seed to a private key, implies that a private key can be compressed to these 32 bytes. Uncompression then means running SeededKeyGen again. We have designed various details of SeededKeyGen, and of the private-key format, to support a slightly different compression mechanism for which uncompression is much faster than key generation.

The main bottleneck in key generation is reducing a parity-check matrix to systematic form (or semi-systematic form), and starting over with a new key-generation attempt if the matrix reduction fails. However, as explained earlier, the resulting public key is not needed for decapsulation. The data flow from the matrix reduction to the private key consists solely of (1) knowing whether $(g, \alpha_1, \ldots, \alpha_n)$ has been rejected and (2) a permutation of $(\alpha_1, \ldots, \alpha_n)$ into $(\alpha'_1, \ldots, \alpha'_n)$ for the generalization to semi-systematic form.

SeededKeyGen starts with a seed δ and deterministically maps δ to $(g, \alpha_1, \ldots, \alpha_n, \delta')$. If $(g, \alpha_1, \ldots, \alpha_n)$ is rejected, SeededKeyGen replaces δ with δ' and starts over. This structure supports a compression mechanism that stores the *final* seed, a seed known to pass the rejection-sampling process, rather than the initial seed. Uncompression then simply maps the final seed δ to the final $(g, \alpha_1, \ldots, \alpha_n)$, without any matrix operations.

For the generalization to semi-systematic form, uncompression needs to compute $(\alpha'_1, \ldots, \alpha'_n)$, not just $(\alpha_1, \ldots, \alpha_n)$. The permutation of $(\alpha_1, \ldots, \alpha_n)$ into $(\alpha'_1, \ldots, \alpha'_n)$ is fully specified by a ν-bit string of weight μ encoding $c = (c_{n-k-\mu+1}, \ldots, c_{n-k})$: e.g., a 64-bit string of weight 32 when $(\mu, \nu) = (32, 64)$.

We organize the objects $(\delta, c, g, \alpha, s)$ in a private key so that four natural compression mechanisms each consist of simple truncation:

- Truncation to (δ, c, g, α) saves $n/8$ bytes in the private key, and regenerating s from δ requires simply (the first) $n/8$ bytes of SHAKE256 output.
- Truncation to (δ, c, g) requires more work to regenerate α but saves much more space.
- Truncation to (δ, c) requires a minimal-polynomial computation to regenerate g but compresses to just 40 bytes.
- Truncation to the 32-byte seed δ suffices for systematic form.

We could encode c as 0 bytes for systematic form. We instead encode it as 8 bytes so that a systematic-form private key can also be used by implementations that expect a semi-systematic-form private key. This avoids the need for key-format specifications to distinguish systematic form from semi-systematic form when all other parameters are the same: systematic-form private keys are simply the special case of semi-systematic-form private keys in which these 8 bytes are $(255, 255, 255, 255, 0, 0, 0, 0)$.

The private key is specified to record an ordering $(\alpha'_1, \ldots, \alpha'_n, \alpha_{n+1}, \ldots, \alpha_q)$ of the field $\mathbb{F}_q$ as a sequence of control bits for a Beneš network. Most of our parameters have $n < q$, and no use is made of $(\alpha_{n+1}, \ldots, \alpha_q)$, so another way to save space is to list just $(\alpha'_1, \ldots, \alpha'_n)$. One can apply the corresponding permutation by sorting, which is slower than a Beneš network but avoids the need to compute control bits. One can also apply the corresponding permutation through RAM lookups, but implementors are cautioned that this leaks information through timing on many platforms. Specifying control bits as the default representation of α has the advantage of encouraging constant-time implementations. All implementations should be reviewed for timing leaks and other applicable side-channel leaks in any case.

Out of all 256-bit seeds, under 30% of seeds are accepted for systematic form. The final seed has only about 254 bits of entropy and, like most private keys in most cryptographic algorithms, is efficiently distinguishable from uniform. A search through seeds still costs more than AES-256 key search.

Field ordering. The FieldOrdering algorithm interprets a uniform random $4q$-byte input string as a uniform random sequence of 32-bit integers $a_0, a_1, \ldots, a_{q-1}$. This sequence is rejected if and only if it contains fewer than q distinct elements. The sequence is accepted with probability $(1 - 1/2^{32})(1 - 2/2^{32}) \cdots (1 - (q - 1)/2^{32})$, which is more than 0.99 if $q \leq 2^{13}$, and more than 0.6 if $q \leq 2^{16}$. (This description focuses on our choice $\sigma_2 = 32$. Parameters with $q > 2^{16}$ would take more than 32 bits in each integer by definition of σ_2, so the acceptance probability would still be more than 0.6.)

An accepted sequence is a uniform random sequence of distinct 32-bit integers $a_0, a_1, \ldots, a_{q-1}$. There is then a unique permutation that sorts $(a_0, a_1, \ldots, a_{q-1})$, and the output is the same permutation applied to an initial ordering of $\mathbb{F}_q$. The permutation is a uniform random permutation, so the output is a uniform random ordering of $\mathbb{F}_q$.

Omitting the rejection would produce a permutation distinguishable from uniform, but would still suffice for almost exactly the same OW-CPA security level by a divergence argument. See [7].

Irreducible polynomial. The IRREDUCIBLE algorithm extracts mt input bits from a uniform random $2t$-byte input string, and interprets the mt bits as a uniform random element β of $\mathbb{F}_q[y]/F(y)$. The algorithm returns the minimal polynomial g of β over $\mathbb{F}_q$ if g has degree t; otherwise it fails. (This description focuses on our choice $\sigma_1 = 16$, with $m \leq 16$.)

Any particular monic irreducible degree-t polynomial g has exactly t roots in $\mathbb{F}_q[y]/F(y)$, and is thus found by exactly t of the 2^{mt} possibilities for β, i.e., exactly $t2^{16t-mt}$ of the possibilities for the $16t$-bit input string. The distribution of g is thus uniform. Uniformity is again overkill here: having low divergence would suffice.

Well-known formulas for the number of irreducible polynomials (equivalently, the observation that the algorithm fails exactly when β is in a proper subfield of $\mathbb{F}_q[y]/F(y)$) imply that this algorithm succeeds with probability more than 99% when $t \geq 16$.

Fixed-weight vector in encapsulation. MODELENCAP begins by generating a uniform random n-bit vector of weight t. This again raises a question for cryptographic module validators regarding how these vectors are generated. ENCAP instead calls FIXEDWEIGHT, which calls a traditional RNG that produces a stream of bits.

This RNG can in turn generate output from a series of seeds as in key generation, allowing a ciphertext to be compressed to the final seed. See [10] for an application. The same compression approach works for rejection sampling in much more generality.

FIXEDWEIGHT generates a uniform random n-bit vector $e = (e_0, e_1, \ldots, e_{n-1})$ of weight t by generating a uniform random sequence $(a_0, a_1, \ldots, a_{t-1})$ of t distinct integers in $\{0, 1, \ldots, n-1\}$, and using those integers as the support of e.

FIXEDWEIGHT generates this uniform random sequence by generating a uniform random sequence $(a_0, a_1, \ldots, a_{t-1})$ of integers in $\{0, 1, \ldots, n-1\}$, and then starting over if the integers are not distinct. Each try succeeds with probability $(1 - 1/n)(1 - 2/n) \cdots (1 - (t-1)/n)$, which is above $1/4$ for each of our selected parameter sets. (The alternative e-generation method in [7] guarantees its run time, but FIXEDWEIGHT is essentially always faster.)

Generating a uniform random sequence $(a_0, a_1, \ldots, a_{t-1})$ of integers in $\{0, 1, \ldots, q-1\}$ is a simple matter of collecting uniform random bits. For $n < q$, one well-known way to generate a uniform random stream of integers in $\{0, 1, \ldots, n-1\}$ is by rejection sampling on a uniform random stream of integers in $\{0, 1, \ldots, q-1\}$. If n is below $q/2$ then it is more efficient to begin with integers in $\{0, 1, \ldots, q/2-1\}$, and similar comments apply if n is below $q/4$ etc., but we skip these refinements since all of our parameters have $n > q/2$.

FIXEDWEIGHT uses a batch of τ integers in $\{0, 1, \ldots, q-1\}$, and applies rejection sampling to generate a batch of integers in $\{0, 1, \ldots, n-1\}$, say u integers, where u is between 0 and τ. For each u, the integers in $\{0, 1, \ldots, n-1\}$ are uniform. Consequently, if $u \geq t$, the first t integers in $\{0, 1, \ldots, n-1\}$ are uniform as desired. Batch processing simplifies parallelization and vectorization, and some standard RNGs are much more efficient at generating a large batch of random bits than at generating the same volume of data in small chunks.

The probability of $u \geq t$ is the sum of the coefficients of $x^t, x^{t+1}, \ldots, x^\tau$ in $(1-n/q+(n/q)x)^\tau$. This probability is above 0.96 for $(m, n, t) = (13, 4608, 96)$, and much closer to 1 for our other selected parameters.

There would be a slight savings in time from reducing τ below $2t$. With smaller batch sizes it would also save time to handle the case $u < t$ differently: instead of discarding the u integers already found in $\{0, 1, \ldots, n-1\}$, keep those integers and use the next batch to extend the list of integers. This would reduce the number of iterations required, at the expense of tracking state between iterations. As long as the selection process sees only whether integers are in $\{0, 1, \ldots, n-1\}$ or not, the resulting integers in $\{0, 1, \ldots, n-1\}$ are uniform.

5 Detailed performance analysis (2.B.2)

5.1 Overview of implementations

We are supplying 40 software implementations as part of this submission:

- There are five proposed parameter sets using systematic form: `mceliece348864`, `mceliece460896`, `mceliece6960119`, `mceliece6688128`, and `mceliece8192128`.
- There are also five proposed parameter sets using semi-systematic form: `mceliece348864f`, `mceliece460896f`, `mceliece6960119f`, `mceliece6688128f`, and `mceliece8192128f`.
- Each of these ten parameter sets has a `ref` implementation (designed for clarity, not performance); a `vec` implementation (vectorizing across the 64 bits in a `long long`); an `sse` implementation (using the Intel/AMD 128-bit vector instructions); and an `avx` implementation (using the Intel/AMD 256-bit vector instructions). These four implementations are interoperable and produce identical test vectors.

All of the implementations are designed to avoid all data flow from secrets to timing,[2] stopping timing attacks such as [68]. Formally verified protection against timing attacks can be provided by a combination of architecture documentation as recommended in [6] and [38], and timing-aware compilation as in [1].

5.2 Time

Table 1 reports speeds of the `avx` implementations on an Intel Haswell CPU core described in more detail below. For comparison, the `mceliece8192128` software originally submitted for

[2]Each attempted key generation (for the non-`f` variants) succeeds with probability about 29%, as mentioned earlier, so the total time for key generation varies. However, the final *successful* key generation takes constant time, and it uses separate random numbers from the unsuccessful key-generation attempts. In other words, the information about secrets that is leaked through timing is information about secrets that are not used.

	operation	quartile	median	average	quartile
`mceliece348864`	`keypair`	35492688	46526112	58034411	68561244
`mceliece348864f`	`keypair`	36612936	36627388	36641040	36645716
`mceliece460896`	`keypair`	119530424	158155696	215785433	236056616
`mceliece460896f`	`keypair`	116896272	116914656	117067765	116938560
`mceliece6688128`	`keypair`	364883936	458561448	556495649	738637632
`mceliece6688128f`	`keypair`	284363176	284468140	284584602	284551052
`mceliece6960119`	`keypair`	249975756	330214944	438217685	490873872
`mceliece6960119f`	`keypair`	246252520	246291008	246508730	246336840
`mceliece8192128`	`keypair`	316082088	409854088	514489441	594965680
`mceliece8192128f`	`keypair`	316118712	316166640	316202817	316221040
`mceliece348864`	`enc`	42812	43832	44350	44872
`mceliece460896`	`enc`	111140	115540	117782	121460
`mceliece6688128`	`enc`	143540	149080	151721	154632
`mceliece6960119`	`enc`	156212	159116	161224	163412
`mceliece8192128`	`enc`	175840	177480	178093	178688
`mceliece348864`	`dec`	134056	134184	134745	134324
`mceliece460896`	`dec`	270444	270856	271694	271048
`mceliece6688128`	`dec`	322756	322988	323957	323148
`mceliece6960119`	`dec`	300448	300688	301480	301072
`mceliece8192128`	`dec`	325624	325744	326531	325904

Table 1: Time for complete cryptographic functions on an Intel Haswell CPU core. All times are expressed in CPU cycles. Statistics are computed across SUPERCOP's default 93 experiments. The `f` variants have different `keypair` algorithms but identical `enc` algorithms and identical `dec` algorithms.

round 1 took about 2 billion cycles for each key-generation attempt (and on average about 6 billion cycles for total key generation), slightly under 300000 cycles for encapsulation, and slightly over 450000 cycles for decapsulation. The round-2 submission reached approximately the current speeds for encapsulation and decapsulation, took 1.28 billion cycles on average for `mceliece8192128` key generation, and took 0.68 billion cycles for `mceliece8192128f` key generation. The software in this submission generates keys more than twice as quickly.

Table 2 reports measurements of a separate FPGA implementation of the core mathematical functions (not including, e.g., hashing). The computations in McEliece's cryptosystem are particularly well suited for hardware implementations; see [74] and `https://caslab.csl.yale.edu/code/niederreiter/`.

5.3 Space

Table 3 reports sizes of inputs and outputs. Note the small ciphertext sizes (including 32 bytes for confirmation), about half the size of compressed SIKE ciphertexts that claim the same security levels, and much smaller than lattice ciphertexts that claim the same security

	FPGA	`keypair` cycles	`enc` cycles	`dec` cycles	Fmax MHz	LUT	FF	BRAM
Area optimized								
`mceliece348864`	Artix-7	1599882	2720	15638	108.1	25327	49383	168
`mceliece460896`	Artix-7	5002044	3360	27645	107.0	38669	74858	303
`mceliece6688128`	Virtex-7	12389742	5024	47309	136.1	44345	83637	446
`mceliece6960119`	Virtex-7	11179636	5413	40728	140.5	44154	88963	563
`mceliece8192128`	Virtex-7	15185314	6528	48802	134.2	45150	88154	525
Area and time balanced								
`mceliece348864`	Artix-7	482893	2720	12036	104.8	39766	70453	213
`mceliece460896`	Artix-7	1383104	3360	18771	108.0	57134	97056	349
`mceliece6688128`	Virtex-7	3346231	5024	32145	143.4	66615	111299	492
`mceliece6960119`	Virtex-7	3086064	5413	26617	136.2	63629	115580	509
`mceliece8192128`	Virtex-7	4115427	6528	33640	131.3	67457	115819	572
Time optimized								
`mceliece348864`	Artix-7	202787	2720	10023	105.6	81339	132190	236
`mceliece460896`	Virtex-7	515806	3360	14571	130.8	109484	168939	446
`mceliece6688128`	Virtex-7	1046139	5024	24730	136.6	122624	186194	589
`mceliece6960119`	Virtex-7	974306	5413	19722	130.0	116928	188324	607
`mceliece8192128`	Virtex-7	1286179	6528	26237	129.9	123361	190707	589

Table 2: Performance of core mathematical operations on FPGAs. We provide numbers for three performance parameter sets: one for small area, one for small runtime, and one for balanced time and area.

levels. Classic McEliece is used in the PQ-WireGuard [40] VPN, which beyond security is "mainly concerned about ciphertext size".

Another aspect of space is RAM usage. McTiny [10] shows how clients can stream ephemeral Classic McEliece keys through a tiny network server to set up new sessions, with the server immediately handling each network packet and not allocating any RAM per client.

5.4 Description of platforms

The software measurements were collected using `supercop-20200906` running on a computer named `titan0`. The CPU on `titan0` is an Intel Xeon E3-1275 v3 (Haswell) running at 3.50GHz. This CPU does not support hyperthreading. It does support Turbo Boost but `/sys/devices/system/cpu/intel_pstate/no_turbo` was set to 1, disabling Turbo Boost. `titan0` has 32GB of RAM and runs Ubuntu 18.04. Benchmarks used `./do-part`, which ran on one core of the CPU. The compiler list was reduced to just `gcc -march=native -mtune=native -O3 -fomit-frame-pointer -fwrapv`.

NIST says that the "NIST PQC Reference Platform" is "an Intel x64 running Windows or Linux and supporting the GCC compiler." `titan0` is an Intel x64 running Linux and

	Public key	Private key	Ciphertext	Session key
`mceliece348864`	261120	6492	128	32
`mceliece348864f`	261120	6492	128	32
`mceliece460896`	524160	13608	188	32
`mceliece460896f`	524160	13608	188	32
`mceliece6688128`	1044992	13932	240	32
`mceliece6688128f`	1044992	13932	240	32
`mceliece6960119`	1047319	13948	226	32
`mceliece6960119f`	1047319	13948	226	32
`mceliece8192128`	1357824	14120	240	32
`mceliece8192128f`	1357824	14120	240	32

Table 3: Sizes of inputs and outputs to the complete cryptographic functions. All sizes are expressed in bytes.

supporting the GCC compiler. Beware, however, that different Intel CPUs have different cycle counts.

The hardware design was synthesized using Vivado v2018.3. Some of the hardware designs for smaller parameter sets fit into an Artix-7 XC7A200T FPGA, and in these cases performance numbers are reported on that FPGA. In the remaining cases, performance numbers on a Virtex-7 XC7V2000T FPGA are reported instead. BRAM is shown in the unit of RAMB36.

5.5 How parameters affect performance

The ciphertext size is $n - k$ bits. Normally the rate $R = k/n$ is chosen around 0.8 (see Section 8), so the ciphertext size is around $0.2n$ bits, i.e., $n/40$ bytes, plus 32 bytes for confirmation.

The public-key size is $k(n-k)$ bits. For $R \approx 0.8$ this is around $0.16n^2$ bits, i.e., $n^2/50$ bytes.

Generating the public key uses $n^{3+o(1)}$ operations with standard Gaussian elimination. There are asymptotically faster matrix algorithms. Private-key operations use just $n^{1+o(1)}$ operations with standard algorithms.

6 Expected strength (2.B.4) in general

This submission is designed and expected to provide IND-CCA2 security.

See Section 7 for the quantitative security of our proposed parameter sets, and Section 8 for analysis of known attacks. The rest of this section analyzes the KEM from a provable-security perspective.

6.1 Provable-security overview

In general, a security theorem for a cryptographic system C states that an attack $\mathcal{A}$ of type T against C implies an attack $\mathcal{A}'$ against an underlying problem P. Here are four important ways to measure the quality of a security theorem:

- The security of the underlying problem P. The theorem is useless if P is easy to break, and the value of the theorem is questionable if the security of P has not been thoroughly studied.
- The "tightness" of the theorem: i.e., the closeness of the efficiency of $\mathcal{A}'$ to the efficiency of $\mathcal{A}$. If $\mathcal{A}'$ is much less efficient than $\mathcal{A}$ then the theorem does not rule out the possibility that C is much easier to break than P.
- The type T of attacks covered by the theorem. The theorem does not rule out attacks of other types.
- The level of verification of the proof.

Our original plan in preparing the round-1 Classic McEliece submission was to present a KEM with a theorem of the following type:

- P is exactly the thoroughly studied inversion (OW-CPA) problem for McEliece's original 1978 system.
- The theorem is extremely tight.
- The theorem covers all IND-CCA2 "ROM" (Random-Oracle Model) attacks. Roughly, an attack of this type is an IND-CCA2 attack that works against any hash function H, given access to an oracle that computes H on any input.
- The proof was already published by Dent [30, Theorem 8] in 2003. The proof is not very complicated, and should be within the range of current techniques for computer verification of proofs.

Shortly before round-1 submissions were due, a paper by Saito, Xagawa and Yamakawa [62] indicated that it was possible—without sacrificing tightness—to expand the attack type T from all IND-CCA2 ROM attacks to all IND-CCA2 "QROM" (Quantum Random-Oracle Model) attacks. Roughly, an attack of this type is an IND-CCA2 attack that works against any hash function H, given access to an oracle that computes H on a *quantum superposition* of inputs.

In our round-1 submission we wrote the following:

> An obstacle here is that Dent's theorem and the Saito–Xagawa–Yamakawa theorem are stated for different KEMs. Another obstacle is that, while Dent's theorem is stated with OW-CPA as the sole assumption, the Saito–Xagawa–Yamakawa theorem is stated with additional assumptions.
>
> To obtain the best of both worlds, we have designed a KEM that combines Dent's framework with the Saito–Xagawa–Yamakawa framework, with the goal

of allowing *both* proof techniques to apply. This has created a temporary sacrifice in the level of verification, but we expect that complete proofs will be written and checked by the community in under a year.

Our round-1 submission included preliminary analyses of both frameworks, and then followup analyses in the literature gave complete proofs applicable to our KEM. We include the story here. Section 6.2 presents the abstract KEM design; Sections 6.3 and 6.4 repeat the preliminary analyses given in our round-1 submission regarding the two proof frameworks; Section 6.5 summarizes the followup analyses in the literature; and Section 6.6 explains how the abstract KEM design relates to the specification of Classic McEliece.

6.2 Abstract conversion

Abstractly, we are building a correct KEM given a correct deterministic PKE. We want the KEM to achieve IND-CCA2 security, and we want this to be proven to the extent possible, assuming that the PKE achieves OW-CPA security.

The PKE functionality is as follows. There is a set of public keys, a set of private keys, a set of plaintexts, and a set of ciphertexts. There is a key-generation algorithm KeyGen that produces a public key and a private key. There is a deterministic encryption algorithm $\mathsf{Encrypt}$ that, given a plaintext and a public key, produces a ciphertext. There is a decryption algorithm $\mathsf{Decrypt}$ that, given a ciphertext and a private key, produces a plaintext or a failure symbol $\perp$ (which is not a plaintext). We require that $\mathsf{Decrypt}(\mathsf{Encrypt}(p, K), k) = p$ for every (K, k) output by $\mathsf{KeyGen}()$ and every plaintext p.

We emphasize that $\mathsf{Encrypt}$ is not permitted to randomize its output: in other words, any randomness used to produce a ciphertext must be in the plaintext recovered by decryption. We also emphasize that $\mathsf{Decrypt}$ is not permitted to fail on valid ciphertexts; even a tiny failure probability is not permitted. These requirements are satisfied by the PKE in this submission, and the literature indicates that these requirements are helpful for security proofs.

In this level of generality, our KEM is defined in two modular layers as follows, using three hash functions H_0, H_1, H_2. These hash functions can be modeled in proofs as independent random oracles. If the hash output spaces are the same then this is equivalent to defining $\mathsf{H}_i(x) = \mathsf{H}(i, x)$ for a single random oracle H, since the input spaces are disjoint.

First layer. Write X for the original correct deterministic PKE. We define a modified PKE $X_2 = \textsc{ConfirmPlaintext}(X, \mathsf{H}_2)$ as follows. This modified PKE is also a correct deterministic PKE.

The modified key-generation algorithm KeyGen_2 is the same as the original key-generation algorithm KeyGen. The set of public keys is the same, and the set of private keys is the same.

The modified encryption algorithm $\mathsf{Encrypt}_2$ is defined by $\mathsf{Encrypt}_2(p, K) = (\mathsf{Encrypt}(p, K), \mathsf{H}_2(p))$. The set of plaintexts is the same, and the modified set of ciphertexts consists of pairs of original ciphertexts and hash values.

Finally, the modified decryption algorithm $\mathsf{Decrypt}_2$ is defined by $\mathsf{Decrypt}_2((C, h), k) = \mathsf{Decrypt}(C, k)$.

Note that $\mathsf{Decrypt}_2$ does not check hash values: changing (C, h) to a different (C, h') produces the same output from $\mathsf{Decrypt}_2$. There was also no requirement for the original PKE X to recognize invalid ciphertexts.

Second layer. We define a KEM RandomizeSessionKeys$(X_2, \mathsf{H}_1, \mathsf{H}_0)$ as follows, given a correct deterministic PKE X_2 with algorithms $\mathsf{KeyGen}_2, \mathsf{Encrypt}_2, \mathsf{Decrypt}_2$. This KEM is a correct KEM.

Key generation:

1. Compute $(K, k) \leftarrow \mathsf{KeyGen}_2()$.
2. Choose a uniform random plaintext s.
3. Output K as the public key, and (k, K, s) as the private key.

Encapsulation, given a public key K:

1. Choose a uniform random plaintext p.
2. Compute $C \leftarrow \mathsf{Encrypt}_2(p, K)$.
3. Output C as the ciphertext, and $\mathsf{H}_1(p, C)$ as the session key.

Decapsulation, given a ciphertext C and a private key (k, K, s):

1. Compute $p' \leftarrow \mathsf{Decrypt}_2(C, k)$.
2. If $p' = \perp$, set $p' \leftarrow s$ and $b \leftarrow 0$. Otherwise set $b \leftarrow 1$.
3. Compute $C' \leftarrow \mathsf{Encrypt}_2(p', K)$.
4. If $C \neq C'$, set $p' \leftarrow s$ and $b \leftarrow 0$.
5. Output $\mathsf{H}_b(p', C)$ as the session key.

In other words:

- If there exists a plaintext p such that $C = \mathsf{Encrypt}_2(p, K)$, then decapsulation outputs $\mathsf{H}_1(p, C)$. Indeed, $p' = \mathsf{Decrypt}_2(C, k) = p$ by correctness, so $C' = \mathsf{Encrypt}_2(p, K) = C$ and $b = 1$ throughout, so the output is $\mathsf{H}_1(p, C)$.
- If there does *not* exist a plaintext p such that $C = \mathsf{Encrypt}_2(p, K)$, then decapsulation outputs $\mathsf{H}_0(s, C)$. Indeed, the only way for b to avoid being set to 0 in Step 4 is to have $C' = \mathsf{Encrypt}_2(p', K)$, contradiction; so that step sets p' to s and sets b to 0, and decapsulation outputs $\mathsf{H}_0(s, C)$.

6.3 The Dent framework: preliminary analysis

The following text is repeated from our round-1 submission. Readers interested in the latest analyses can skip to Section 6.5.

The conversion by Dent requires nothing more than OW-CPA security for the underlying PKE, and has a tight IND-CCA2 ROM proof, but for a different KEM. Compared to Dent's KEM, the most significant change in our KEM is the replacement of the $\perp$ output for decapsulation errors with a pseudorandom value. This variant is not new and similar techniques have been used before for code-based schemes (e.g. [59, 60]). We expect that a theorem along the following lines can be proven for our KEM, showing that this difference does not have any sort of negative impact on the security proof.

Expected Theorem 1 *Let $\mathcal{A}$ be an IND-CCA2 adversary against the KEM, running in time t, with advantage ϵ, that performs at most q decapsulation queries and at most q_1 and q_2 queries to the independent uniform random oracles H_1 and H_2 respectively. Then there exists an OW-CPA adversary $\mathcal{A}'$ against the PKE, running in time t', which is successful with probability ϵ', where*

$$t' \leq t + (q + q_1 + q_2)T,$$
$$\epsilon' \geq \epsilon - \frac{q}{2^{\ell_2}} - \frac{q}{\#M},$$

where T is the running time of encapsulation, ℓ_2 is the number of bits of H_2 output, and $\#M$ is the size of the plaintext space.

We now indicate the modifications that need to be made in the proof of [30, Theorem 8]. First of all, the auxiliary table used by the algorithm simulating H_1 (called *KDFList* in [30]) now contains entries of the type (x_0, x_1, x_2, K) to reflect the different form of the input. The simulator works in exactly the same way, checking the table for previously queried values and outputting a randomly-generated value for K otherwise. Then, we have to modify the response to decapsulation queries. These receive the same input as in [30], and the simulator behaves similarly. It first checks if there exists a preimage p that was already queried by the hash simulator for H_2 and is consistent with the ciphertext. But now, the simulator has to output a value for K even if this check fails: it will simply call the key-generating simulator for $\mathsf{H}_0(s, C)$ rather than $\mathsf{H}_1(p, C)$, where s is an independently generated element as in an honest run of the key generation algorithm. This modification has no impact on the simulation and the adversary learns no more than if it would have received $\perp$ instead. Note that the game is still halted if the adversary attempts to query the simulator on the challenge ciphertext.

Apart from these modifications, the proof is expected to proceed in the same way, generating the same probability bound. The probability bound is a consequence of one of two events occurring, none of which are impacted by the above modifications: the probability of the adversary querying the decapsulation oracle on the challenge ciphertext before this is generated, or querying it on the encapsulation of a string for which the hash oracle hasn't been queried.

6.4 The SXY framework: preliminary analysis

The following text is repeated from our round-1 submission. Readers interested in the latest analyses can skip to Section 6.5.

As noted above, Saito, Xagawa, and Yamakawa very recently introduced a KEM construction "XYZ" with a tight QROM theorem [62, Theorem 5.2]. This theorem, like Dent's theorem, requires the underlying PKE to be correct (no decryption error) and deterministic. It also makes a stronger security assumption regarding the PKE: the PKE is required to satisfy a new notion of security called PR-CPA, which guarantees that encryption keys and ciphertexts can be indistinguishably replaced by "fake", randomly-generated equivalents. More precisely, to be considered PR-CPA secure, an encryption scheme needs to satisfy the following three requirements:

- *PR-key security*: adversary has negligible advantage to distinguish a real public key from a fake one.
- *PR-ciphertext security*: adversary has negligible advantage to distinguish a real ciphertext from a fake one when using a fake public key.
- *Statistical disjointness*: negligible probability that a fake ciphertext is in the range of a real ciphertext obtained via a fake key.

See [62, Definition 3.1].

Our KEM construction has two differences from XYZ. First, there is an extra hash value in the ciphertext. Second, the ciphertext is an extra input to the hash used to compute the session key. We expect that a QROM theorem can be proven for our KEM as a composition of the following two steps.

Step 1: Reduce to passive attacks. The proof in [62] can be decomposed into two parts. The first part shows that decapsulation does not reveal any additional information: i.e., all attacks are as difficult as passive attacks.

The original proof of the first part proceeds as follows. If decryption fails or reencryption produces a different ciphertext, XYZ decapsulation outputs $\mathsf{H}_0(s, C)$. The proof simulates $\mathsf{H}_0(s, C)$ with $\mathsf{H}_q(C)$, where H_q (using the notation from [62]) is a random oracle.

If decryption succeeds and reencryption produces the same ciphertext, XYZ decapsulation outputs $\mathsf{H}_1(p)$. The proof redefines $\mathsf{H}_1(p)$ as $\mathsf{H}_q(\mathsf{Encrypt}(p, K))$; this does not change the attack success probability, since H_1 is again a random oracle. It is crucial to understand that this is valid only since the attack doesn't have access to H_q—except via decapsulation failures, but those are disjoint inputs to H_q.

Now decapsulation outputs $\mathsf{H}_q(C)$ for *all* ciphertexts C, whether C itself is valid or invalid.

The attack using this decapsulation oracle has the same output as an attack that instead uses an oracle for its own randomly chosen H_q.

For our KEM construction, decapsulation outputs $\mathsf{H}_1(p, C)$ in the success case rather than $\mathsf{H}_1(p)$. We proceed analogously. First simulate $\mathsf{H}_0(s, C)$ with $\mathsf{H}_q(C, C)$, where H_q is a random oracle. Then redefine $\mathsf{H}_1(p, C)$ as $\mathsf{H}_q(\mathsf{Encrypt}(p, K), C)$; this is again a random oracle, and again the inputs to H_q are disjoint between the valid and invalid cases. Finally, decapsulation maps C to $\mathsf{H}_q(C, C)$ in all cases, regardless of the validity of C.

Step 2: Invoke the PR-CPA assumptions. The second part of the proof in [62] shows that, given the PR-CPA assumptions, passive attacks are infeasible. We expect this part of the proof to apply directly to our KEM construction, invoking the PR-CPA assumptions for the modified PKE.

We expect the PR-CPA assumptions for the modified PKE to be provable as follows from the same assumptions for the original PKE. PR-key security is the same property for the two PKEs, since $\mathsf{KeyGen}_2 = \mathsf{KeyGen}$. PR-ciphertext security for the modified PKE for a random oracle H_2 should follow from PR-ciphertext security for the original PKE. Statistical disjointness for the modified PKE is implied by statistical disjointness for the original PKE, since identical ciphertexts for the modified PKE begin with identical ciphertexts for the original PKE.

Plausibility of the PR-CPA assumptions for Classic McEliece. As noted in Section 4, there is a long literature on information-set decoding, the fastest inversion attack known against the McEliece PKE. This literature generally treats the problem of decoding *uniform random* codes, and frequently observes that—in experiments—the attacks behave the same way for uniform random binary Goppa codes. This behavior of attacks is sometimes formalized and generalized to a hypothesis about all fast algorithms: namely, the generator matrix (or parity-check matrix) for a uniform random binary Goppa code is hard to distinguish from the generator matrix (or parity-check matrix) for a uniform random code.

This hypothesis is the PR-key security assumption for this PKE. Cryptanalysis of this hypothesis has focused mainly on key-recovery attacks, although, as noted earlier, there is a paper [33] explicitly studying distinguishing attacks. None of these attacks threaten PR-key security for our proposed parameters. This is not the same as saying that PR-key security has been studied as thoroughly as OW-CPA security. Similarly, existing cryptanalysis of PR-ciphertext security has focused mainly on inversion attacks. Statistical disjointness, a statement about the sparsity of the range of the encryption function compared to the ciphertext space, may be provable: a similar property "γ-uniformity" was proved by Cayrel, Hoffmann, and Persichetti [21].

To summarize, there is already some work that can be viewed as studying the PR-CPA assumptions. On the other hand, the assumptions go beyond the thoroughly studied McEliece OW-CPA problem. A theorem assuming PR-CPA security, as in [62], is thus not a replacement for a theorem assuming merely OW-CPA security, as in [30, Theorem 8]. Note that the reduction to passive attacks is independent of this choice of assumption.

6.5 Followup analyses

In May 2018, an update of [62] gave the following two-layer proof for our two-layer KEM:

- First, ConfirmPlaintext (called "KC" in [62]) produces a "disjoint simulatable" PKE from an OW-CPA PKE. (Disjoint simulatability is a ciphertext-unrecognizability assumption.) This reduction is tight in the ROM but loose in the QROM.
- Second, RandomizeSessionKeys produces an IND-CCA2 KEM from a disjoint simulatable PKE. This reduction is tight in the ROM, and also tight in the QROM.

Another paper [14] in May 2018, from a subset of the Classic McEliece team, gave a two-layer proof that RandomizeSessionKeys produces a ROM IND-CCA2 KEM from an OW-CPA PKE. This paper also presented counterexamples to two theorems from [39], illustrating the importance of proof verification.

We summarized the situation in our round-2 submission as follows: "In short, as expected, the state-of-the-art proof techniques work for our KEM. The main open question is whether QROM IND-CCA2 security can be proven tightly from OW-CPA security of the underlying PKE."

This open question was then resolved positively by [15], which proves a bound ϵ on the probability of a QROM IND-CCA2 attack, assuming a bound on the scale of ϵ^2 on the probability of an OW-CPA attack against the underlying deterministic PKE.

(For comparison, the best QROM results [43] known for randomized PKEs need to assume a bound on the scale of ϵ/q where q is the number of hash queries. For values of ϵ of interest, ϵ/q is much smaller than ϵ^2. These results also assume IND-CPA security, which is a stronger assumption than OW-CPA security. There are other results for randomized PKEs assuming only OW-CPA security, but these results are even less tight.)

In the context of Classic McEliece, the success probabilities of the best OW-CPA attacks known have the following shape: the best probability-ϵ non-quantum attacks use about $\epsilon 2^\lambda$ operations, and the best probability-ϵ^2 quantum attacks use about $\epsilon 2^{\lambda/2}$ operations. If these are optimal then, by [15], a QROM IND-CCA2 attack has success probability at most ϵ using about $\epsilon 2^{\lambda/2}$ operations.

A natural next step is formal verification, covering both the general CCA conversion and its application to Classic McEliece. See [70] for formal verification of another QROM IND-CCA2 proof; this particular proof is too loose to be useful here, but it should be feasible to adapt the same verification techniques to [15].

6.6 Relating the abstract conversion to the specification

The general specification in Section 2 can be viewed as the result of the following four steps:

- Start with the McEliece PKE. This PKE is correct and deterministic, and its OW-CPA security has been thoroughly studied.
- Switch to Niederreiter's dual PKE. This PKE is correct and deterministic, and its OW-CPA security is tightly implied by the OW-CPA security of the McEliece PKE.
- Obtain a KEM by applying the ConfirmPlaintext conversion, followed by the RandomizeSessionKeys conversion. This KEM is correct, and its IND-CCA2 security is the topic of the previous subsections.
- Apply three further optimizations discussed below. These optimizations preserve correctness, and they do not affect the IND-CCA2 security analysis.

The first optimization is as follows. Checking whether $C = \mathsf{Encrypt}_2(p', K)$, with the knowledge that $p' = \mathsf{Decrypt}_2(C, k)$, does not necessarily require a full $\mathsf{Encrypt}_2$ computation. In particular, in Section 2, the decoding procedure is already guaranteed to output

- a weight-t vector whose syndrome is the input if such a vector exists, or
- $\perp$ otherwise.

Checking whether $C = \mathsf{Encrypt}_2(p', K)$ is thus a simple matter of checking $\mathsf{H}_2(p')$.

The second optimization is as follows. The KEM private key (k, K, s) does not necessarily need as much space as the space for k plus the space for K plus the space for s. For example, if K can be computed efficiently from k, then it can be recomputed on demand, or optionally cached. In Section 2, the situation is even simpler: decapsulation, with the first optimization, does not look at K, so K is simply eliminated from the KEM private key.

The third optimization is that s is generated from a larger space than the plaintext space: it is simpler to generate a uniform random n-bit string than to generate a uniform random weight-t n-bit string. The set of s enters into the security analysis solely for the indistinguishability of $\mathsf{H}_0(s, C)$ from uniform random.

7 Expected strength (2.B.4) for each parameter set

7.1 Parameter set `kem/mceliece348864`

IND-CCA2 KEM, Category 1.

7.2 Parameter set `kem/mceliece348864f`

IND-CCA2 KEM, Category 1.

7.3 Parameter set `kem/mceliece460896`

IND-CCA2 KEM, Category 3.

7.4 Parameter set `kem/mceliece460896f`

IND-CCA2 KEM, Category 3.

7.5 Parameter set `kem/mceliece6688128`

IND-CCA2 KEM, Category 5.

7.6 Parameter set `kem/mceliece6688128f`

IND-CCA2 KEM, Category 5.

7.7 Parameter set `kem/mceliece6960119`

IND-CCA2 KEM, Category 5.

7.8 Parameter set `kem/mceliece6960119f`

IND-CCA2 KEM, Category 5.

7.9 Parameter set `kem/mceliece8192128`

IND-CCA2 KEM, Category 5.

7.10 Parameter set `kem/mceliece8192128f`

IND-CCA2 KEM, Category 5.

8 Analysis of known attacks (2.B.5)

8.1 Information-set decoding, asymptotically

There is a long literature studying algorithms to invert the McEliece PKE. See Section 4.1.

The fastest attacks known use information-set decoding (ISD). The simplest form of ISD, from 1962 Prange [61], tries to guess an error-free information set in the code. An information set is, by definition, a set of positions that determines an entire codeword. The set is error-free, by definition, if it avoids all of the error positions in the "received word", i.e., the ciphertext; then the ciphertext at those positions is exactly the codeword at those positions. The attacker determines the rest of the codeword by linear algebra, and sees whether the attack succeeded by checking whether the error weight is t.

One expects a random set of k positions to be an information set with reasonable probability, the same 29% mentioned earlier. However, the chance of the set being error-free drops rapidly as the number of errors increases. The following asymptotic statement holds for any real number R with $0 < R < 1$: if the code dimension k is $(R + o(1))n$, and the number of errors t is $\Theta(n/\log n)$, then the chance of the set being error-free is $(1 - R + o(1))^t$ as $n \to \infty$. The cost of ISD is thus $(1/(1-R) + o(1))^t$.

Subsequent improvements to ISD have affected the $o(1)$ but have not changed the constant $1/(1-R)$. See generally [13] and [69].

In the McEliece system, t is asymptotically $(1 - R + o(1))n/\lg n$, so the assumption $t \in \Theta(n/\log n)$ holds.[3] To summarize, the (OW-CPA) security level of the McEliece system against all of these attacks is the $n/\lg n$ power of $1/(1-R)^{1-R} + o(1)$.

Meanwhile the ciphertext size is $(1 - R + o(1))n$ bits, and the key size is $(R(1-R) + o(1))n^2$ bits. Security level 2^b thus uses key size $(c_0 + o(1))b^2(\lg b)^2$ where $c_0 = R/(1-R)(\lg(1-R))^2$. This c_0 reaches its minimum value, approximately 0.7418860694, when R is approximately 0.7968121300.

8.2 Information-set decoding, concretely

We emphasize that $o(1)$ does not mean 0: it means something that converges to 0 as $n \to \infty$. More detailed attack-cost evaluation is therefore required for any particular parameters.

As an example, our parameter set `mceliece6960119` takes $m = 13$, $n = 6960$, and $t = 119$. This parameter set was proposed in the attack paper [11] that broke the original McEliece parameters $(10, 1024, 50)$.

[3]Beware that some ISD papers instead measure their results for much larger $t \in \Theta(n)$, such as "half of the GV distance". This dramatically increases cost from $2^{\Theta(n/\lg n)}$ to $2^{\Theta(n)}$. For example, [50] two years ago reports $\mathcal{O}(2^{0.0473n})$ when t is half of the GV distance, compared to $\mathcal{O}(2^{0.0576n})$ from Prange 55 years ago. As these numbers illustrate, this inflation of t also makes differences between algorithms more noticeable. Such large error rates are of interest in coding theory but are not relevant to the McEliece system.

That paper reported that its attack uses $2^{266.94}$ bit operations to break the $(13, 6960, 119)$ parameter set. Subsequent ISD variants have reduced the number of bit operations considerably below 2^{256}. However:

- None of these analyses took into account the costs of memory access. A closer look shows that the attack in [11] is bottlenecked by random access to a huge array (much larger than the public key being attacked), and that subsequent ISD variants use even more memory. The same amount of hardware allows much more parallelism in attacking, e.g., AES-256.
- Known quantum attacks multiply the security level of both ISD and AES by an asymptotic factor $0.5 + o(1)$, but a closer look shows that the application of Grover's method to ISD suffers much more overhead in the inner loop.

We expect that switching from a bit-operation analysis to a cost analysis will show that this parameter set is more expensive to break than AES-256 pre-quantum and much more expensive to break than AES-256 post-quantum.

8.3 Key recovery

A different inversion strategy is to find the private key $(g, \alpha_1, \ldots, \alpha_n)$. As noted earlier, one should not think that this is as difficult as a brute-force search: one can determine the sequence $(\alpha_1, \ldots, \alpha_n)$ from g and the *set* $\{\alpha_1, \ldots, \alpha_n\}$, or alternatively determine g from $(\alpha_1, \ldots, \alpha_n)$. See generally [48], [36], and [57]. However, for (e.g.) our `mceliece6960119` parameter set, the number of choices of g is more than 2^{1500}. Known symmetries provide only a small speedup. The number of choices of $(\alpha_1, \ldots, \alpha_n)$ is much larger. Most of our parameter sets have an extra defense here, namely that there are a huge number of possibilities for the set $\{\alpha_1, \ldots, \alpha_n\}$.

In a multi-message attack scenario, the cost of finding the private key is spread across many messages. There are also faster multi-message attacks that do not rely on finding the private key; see, e.g., [41] and [65]. Rather than analyzing multi-message security in detail, we rely on the general fact that attacking T targets cannot gain more than a factor T. For example, with our recommended `6688/6960/8192` parameter sets, one ciphertext is expected to be secure against an attacker without the resources to find an AES-256 key, and 2^{64} ciphertexts are expected to all be secure against an attacker without the resources to find an AES-192 key.

8.4 Chosen-ciphertext attacks

A traditional approach to chosen-ciphertext attacks against the McEliece system is to add (say) two errors to a ciphertext $Gm + e$. This is equivalent to adding two errors to e. Decryption succeeds if and only if the resulting error vector has weight t, i.e., exactly one of the two error positions was already in e. It is straightforward to find e from the pattern of decryption failures. See, e.g., [73]. For a Niederreiter ciphertext He, one similarly adds two errors to e by adjusting He appropriately.

There are two reasons that these attacks do not work against our submission. First, KEM decapsulation forces the ciphertext to include a hash of e as a confirmation, and the attacker has no way to compute the hash of a modified version of e without knowing e in the first place. Second, the KEM does not reveal decryption failures: the modified ciphertext will produce an unpredictable session key, whether or not the modified error vector has weight t.

The confirmation allows attackers to check possibilities for e by checking their hashes. However, this is much less efficient than ISD.

8.5 Side-channel attacks

As for any state-of-the-art implementation of cryptographic primitives, side-channel security needs to be taken into consideration. Any operation that handles secret data needs to be protected against side-channel attacks depending on the intended application and usage scenario of the implementation.

As a baseline, any implementation should be constant time, i.e., not exposing a varying runtime depending on secret data. The reference implementation of this submission is a constant time implementation.

In addition to protection against timing attacks, further protection against other side channels may be necessary depending on the intended use case. A smart-card implementation for example should be protected among others against power and EM side-channel attacks.

Side channels might in fact enable practical attacks for key recovery and chosen-ciphertext attacks and can amplify ISD by providing partial information (see [44] for an example).

9 Advantages and limitations (2.B.6)

The central advantage of this submission is security. See the design rationale.

Regarding efficiency, the use of random-looking linear codes with no visible structure forces public-key sizes to be on the scale of a megabyte for quantitatively high security: the public key is a full (generator/parity-check) matrix. Key-generation software is also not very fast. Applications must continue using each public key for long enough to handle the costs of generating and distributing the key.

There are, however, some compensating efficiency advantages. Encapsulation and decapsulation are reasonably fast in software, and impressively fast in hardware, due to the simple nature of the objects (binary vectors) and operations (such as binary matrix-vector multiplications). Key generation is also quite fast in hardware. The hardware speeds of key generation and decoding are already demonstrated by our FPGA implementation. Encapsulation takes only a single pass over a public key, allowing large public keys to be streamed through small coprocessors and small devices.

Furthermore, the ciphertexts are unusually small for post-quantum cryptography: under 256 bytes for our proposed high-security parameter sets. This allows ciphertexts to fit comfortably inside single network packets. The small ciphertext size can be much more important for total traffic than the large key size, depending on the ratio between how often keys are sent and how often ciphertexts are sent. System parameters can be adjusted for even smaller ciphertexts.

References

[1] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 53–70. USENIX Association, 2016. https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_almeida.pdf.

[2] Marco Baldi, Franco Chiaraluce, Roberto Garello, and Francesco Mininni. Quasi-cyclic low-density parity-check codes in the McEliece cryptosystem. In *Proceedings of IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007*, pages 951–956. IEEE, 2007.

[3] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1+1=0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/isd-extended.pdf.

[4] Thierry P. Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing key length of the McEliece cryptosystem. In Bart Preneel, editor, *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97. Springer, 2009. https://hal.archives-ouvertes.fr/hal-01081727/file/ACTI-BERGER-2009-2.pdf.

[5] Daniel J. Bernstein. Grover vs. McEliece. In Sendrier [64], pages 73–80. https://cr.yp.to/papers.html#grovercode.

[6] Daniel J. Bernstein. Some small suggestions for the Intel instruction set, 2014. https://blog.cr.yp.to/20140517-insns.html.

[7] Daniel J. Bernstein. Divergence bounds for random fixed-weight vectors obtained by sorting, 2018. https://cr.yp.to/papers.html#divergence.

[8] Daniel J. Bernstein. Verified fast formulas for control bits for permutation networks, 2020. https://cr.yp.to/papers.html#controlbits.

[9] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 250–272. Springer, 2013. https://tungchou.github.io/papers/mcbits.pdf.

[10] Daniel J. Bernstein and Tanja Lange. McTiny: Fast high-confidence post-quantum key erasure for tiny network servers. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1731–1748. USENIX Association, 2020. https://mctiny.org.

[11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In Johannes A. Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008. https://eprint.iacr.org/2008/318.

[12] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011. https://eprint.iacr.org/2010/585.

[13] Daniel J. Bernstein, Tanja Lange, Christiane Peters, and Henk C. A. van Tilborg. Explicit bounds for generic decoding algorithms for code-based cryptography. In *Pre-proceedings of WCC 2009*, pages 168–180, 2009.

[14] Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification, 2018. https://eprint.iacr.org/2018/526.

[15] Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 61–90. Springer, 2019. https://eprint.iacr.org/2019/590.

[16] Leif Both and Alexander May. Optimizing BJMM with nearest neighbors: Full decoding in $2^{2n/21}$ and McEliece security, 2017. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/bjmm+.pdf.

[17] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In Lange and Steinwandt [45], pages 25–46. https://eprint.iacr.org/2017/1139.

[18] Anne Canteaut and Herve Chabanne. A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem. In Pascale Charpin, editor, *Livre des résumés—EUROCODE 94, Abbaye de la Bussière sur Ouche, France, October 1994*, 1994. https://hal.inria.fr/inria-00074443.

[19] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Trans. Information Theory*, 44(1):367–378, 1998. https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/Canteaut_Chabaud98.pdf.

[20] Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, volume 1514 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 1998. https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/Canteaut_Sendrier98.pdf.

[21] Pierre-Louis Cayrel, Gerhard Hoffmann, and Edoardo Persichetti. Efficient implementation of a CCA2-secure variant of McEliece using generalized Srivastava codes. In Marc Fischlin, Johannes A. Buchmann, and Mark Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*, pages 138–155. Springer, 2012. https://hal-ujm.archives-ouvertes.fr/file/index/docid/712875/filename/2012_PKC_cayrel.pdf.

[22] Herve Chabanne and Bernard Courteau. Application de la méthode de décodage itérative d'Omura à la cryptanalyse du système de McEliece, 1993. Université de Sherbrooke, Rapport de Recherche, number 122.

[23] Florent Chabaud. Asymptotic analysis of probabilistic algorithms for finding short codewords. In Paul Camion, Pascale Charpin, and Sami Harari, editors, *Eurocode '92: proceedings of the international symposium on coding theory and applications held in Udine, October 23–30, 1992*, pages 175–183. Springer, 1993.

[24] Tung Chou. McBits revisited. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2017. https://tungchou.github.io/papers/mcbits_revisited.pdf.

[25] George C. Clark, Jr. and J. Bibb Cain. *Error-correcting coding for digital communication*. Plenum, 1981.

[26] John T. Coffey and Rodney M. Goodman. The complexity of information set decoding. *IEEE Transactions on Information Theory*, 35:1031–1037, 1990.

[27] John T. Coffey, Rodney M. Goodman, and P. Farrell. New approaches to reduced complexity decoding. *Discrete and Applied Mathematics*, 33:43–60, 1991. https://core.ac.uk/reader/81155220.

[28] Alain Couvreur, Ayoub Otmani, and Jean-Pierre Tillich. Polynomial time attack on Wild McEliece over quadratic extensions. *IEEE Trans. Information Theory*, 63(1):404–427, 2017. https://eprint.iacr.org/2014/112.

[29] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, January 2004. https://shoup.net/papers/cca2.pdf.

[30] Alexander W. Dent. A designer's guide to KEMs. In Kenneth G. Paterson, editor, *Cryptography and Coding, 9th IMA International Conference, Cirencester, UK, December 16-18, 2003, Proceedings*, volume 2898 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2003. https://eprint.iacr.org/2002/174.

[31] Ilya I. Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25:24–32, 1989. http://www.mathnet.ru/eng/ppi635.

[32] Ilya I. Dumer. On minimum distance decoding of linear codes. In Grigori A. Kabatianskii, editor, *Fifth joint Soviet-Swedish international workshop on information theory, Moscow, 1991*, pages 50–52, 1991.

[33] Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate McEliece cryptosystems. *IEEE Trans. Information Theory*, 59(10):6830–6844, 2013. https://eprint.iacr.org/2010/331.

[34] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009. https://eprint.iacr.org/2009/414.

[35] Classic McEliece Comparison Task Force. Classic McEliece vs. NTS-KEM. 2018. https://classic.mceliece.org/nist/vsntskem-20180629.pdf.

[36] J. Keith Gibson. Equivalent Goppa codes and trapdoors to McEliece's public key cryptosystem. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 517–521. Springer, 1991.

[37] Yann Hamdaoui and Nicolas Sendrier. A non asymptotic analysis of information set decoding. 2013. https://eprint.iacr.org/2013/162.

[38] Gernot Heiser. For safety's sake: We need a new hardware-software contract! *IEEE Des. Test*, 35(2):27–30, 2018. https://ts.data61.csiro.au/publications/csiro_full_text/Heiser_18.pdf.

[39] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017. https://eprint.iacr.org/2017/604.

[40] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Ralf Zimmermann. Post-quantum WireGuard. IEEE S&P 2021, to appear. https://eprint.iacr.org/2020/379.

[41] Thomas Johansson and Fredrik Jönsson. On the complexity of some cryptographic problems based on the general decoding problem. *IEEE Trans. Information Theory*, 48(10):2669–2678, 2002.

[42] Evgueni A. Krouk. Decoding complexity bound for linear block codes. *Problemy Peredachi Informatsii*, 25:103–107, 1989. http://www.mathnet.ru/eng/ppi665.

[43] Veronika Kuchta, Amin Sakzad, Damien Stehlé, Ron Steinfeld, and Shifeng Sun. Measure-rewind-measure: Tighter quantum random oracle model proofs for one-way to hiding and CCA security. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 703–728. Springer, 2020.

[44] Norman Lahr, Ruben Niederhagen, Richard Petri, and Simona Samardjiska. Side channel information set decoding using iterative chunking. In *Advances in Cryptology - ASIACRYPT 2020*, Lecture Notes in Computer Science. Springer, 2020, to appear. Preprint: https://eprint.iacr.org/2019/1459.

[45] Tanja Lange and Rainer Steinwandt, editors. *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*. Springer, 2018.

[46] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In Christoph G. Günther, editor, *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988.

[47] Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Information Theory*, 34(5):1354–1359, 1988.

[48] Pierre Loidreau and Nicolas Sendrier. Weak keys in the McEliece public-key cryptosystem. *IEEE Trans. Information Theory*, 47(3):1207–1211, 2001.

[49] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/decoding.pdf.

[50] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228. Springer, 2015. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/codes.pdf.

[51] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. Technical report, NASA, 1978. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[52] Rafael Misoczki and Paulo S. L. M. Barreto. Compact McEliece keys from Goppa codes. In Michael J. Jacobson Jr., Vincent Rijmen, and Rei Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2009. https://eprint.iacr.org/2009/187.

[53] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 2069–2073. IEEE, 2013. https://eprint.iacr.org/2012/409.

[54] National Institute for Standards and Technology (NIST). Digital signature standard (DSS) FIPS 186–4, 2013. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[55] Matús Nemec, Marek Sýs, Petr Svenda, Dusan Klinec, and Vashek Matyas. The return of Coppersmith's attack: Practical factorization of widely used RSA moduli. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1631–1648. ACM, 2017. https://crocs.fi.muni.cz/public/papers/rsa_ccs17.

[56] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.

[57] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 95–145. Springer Berlin Heidelberg, 2009.

[58] Edoardo Persichetti. Compact McEliece keys based on quasi-dyadic Srivastava codes. *J. Mathematical Cryptology*, 6(2):149–169, 2012. https://eprint.iacr.org/2011/179.

[59] Edoardo Persichetti. Secure and anonymous hybrid encryption from coding theory. In Philippe Gaborit, editor, *Post-Quantum Cryptography: 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, pages 174–187, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[60] Edoardo Persichetti. Code-based key encapsulation from McEliece's cryptosystem. In Johannes Blömer, Ilias S. Kotsireas, Temur Kutsia, and Dimitris E. Simos, editors, *Mathematical Aspects of Computer and Information Sciences - 7th International Conference, MACIS 2017, Vienna, Austria, November 15-17, 2017, Proceedings*, volume 10693 of *Lecture Notes in Computer Science*, pages 454–459. Springer, 2017. https://arxiv.org/abs/1706.06306.

[61] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, IT-8:S5–S9, 1962.

[62] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 520–551. Springer, 2018. https://eprint.iacr.org/2017/1005.pdf.

[63] Nicolas Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Trans. Information Theory*, 46(4):1193–1203, 2000.

[64] Nicolas Sendrier, editor. *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*, volume 6061 of *Lecture Notes in Computer Science*. Springer, 2010.

[65] Nicolas Sendrier. Decoding one out of many. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, volume 7071 of *Lecture Notes in Computer Science*, pages 51–67. Springer, 2011. https://eprint.iacr.org/2011/367.

[66] Victor Shoup. A proposal for an ISO standard for public key encryption. 2001. https://eprint.iacr.org/2001/112.

[67] Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.

[68] Falko Strenzke. A timing attack against the secret permutation in the McEliece PKC. In Sendrier [64], pages 95–107.

[69] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2016. https://hal.inria.fr/hal-01244886v1/document.

[70] Dominique Unruh. Post-quantum verification of Fujisaki-Okamoto, 2020. Asiacrypt 2020, to appear. https://eprint.iacr.org/2020/962.

[71] Johan van Tilburg. On the McEliece public-key cryptosystem. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, volume 403 of *Lecture Notes in Computer Science*, pages 119–131. Springer, 1988.

[72] Johan van Tilburg. *Security-analysis of a class of cryptosystems based on linear error-correcting codes*. PhD thesis, Technische Universiteit Eindhoven, 1994.

[73] Eric R. Verheul, Jeroen M. Doumen, and Henk C. A. van Tilborg. Sloppy Alice attacks! Adaptive chosen ciphertext attacks on the McEliece public-key cryptosystem. In Mario Blaum, Patrick G. Farrell, and Henk C. A. van Tilborg, editors, *Information, coding and mathematics*, volume 687 of *Kluwer International Series in Engineering and Computer Science*, pages 99–119. Kluwer, 2002.

[74] Wen Wang, Jakub Szefer, and Ruben Niederhagen. FPGA-based Niederreiter cryptosystem using binary Goppa codes. In Lange and Steinwandt [45], pages 77–98. https://eprint.iacr.org/2017/1180.