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

2.1 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)
H	A cryptographic hash function	(part of the CM parameters)
ℓ	Length of a hash digest	(part of the CM parameters)
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)
(s, Γ)	A CM private key	
T	A 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.

2.2 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 $n \leq q$.

- A positive integer $t \geq 2$ with $mt < n$. This also defines a parameter $k = n - mt$.
- A monic irreducible polynomial $f(z) \in \mathbb{F}_2[z]$ of degree m. This defines a representation $\mathbb{F}_2[z]/f(z)$ of the field $\mathbb{F}_q$.
- A positive integer ℓ, and a cryptographic hash function H that outputs ℓ bits.

2.3 Key generation

Given a set of CM parameters, a user generates a *CM key pair* as follows:

1. Generate a uniform random monic irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t.
2. Select a uniform random sequence $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n distinct elements of $\mathbb{F}_q$.
3. 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 $j = 1, \ldots, n$.
4. Form an $mt \times n$ matrix $\hat{H}$ over $\mathbb{F}_2$ by replacing each entry $c_0 + c_1 z + \cdots + c_{m-1}z^{m-1}$ of $\tilde{H}$ with a column of t bits $c_0, c_1, \ldots, c_{m-1}$.
5. Apply Gaussian elimination to $\hat{H}$ to 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 Gaussian elimination does not produce I_{n-k} (i.e., $\hat{H}$ cannot be transformed to systematic form), go back to Step 1.
6. Generate a uniform random n-bit string s.
7. Put $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ and output (s, Γ) as private key and T as public key.

The second part of the private key, $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$, 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.

2.4 Encoding subroutine

The encoding subroutine 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 subroutine returns a vector $C_0 \in \mathbb{F}_2^{n-k}$ defined as follows:

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

2.5 Decoding subroutine

The decoding subroutine 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, this subroutine takes two inputs: a vector $C_0 \in \mathbb{F}_2^{n-k}$; and a private key (s, Γ). The subroutine has two possible return values, defined in terms of the public key T that corresponds to (s, Γ):

- If C_0 was returned by the encoding subroutine on input e and T, then the decoding subroutine returns 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 the decoding subroutine returns e.
- If C_0 does not have the form He for any weight-t vector $e \in \mathbb{F}_2^n$, then the decoding subroutine returns $\perp$ (failure).

The subroutine works as follows:

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 standard algorithms for Step 2 of this subroutine. For references and speedups see generally [7] and [17].

To see why the subroutine 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 the subroutine 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 the subroutine 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: the subroutine 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 in this subroutine 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 $c \in \mathbb{F}_2^n$.

As a further implementation note: In order to test $C_0 = He$, the decoding subroutine does not need to recompute H from Γ as in key generation. Instead it 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 standard choices of H' related to $\hat{H}$ that are easily recovered from Γ, and that can be applied to vectors without using quadratic space.

Remark. Note that the triple of algorithms (Key Generation, Encoding, Decoding) is essentially Niederreiter's "dual" version [42] of the McEliece cryptosystem (plus a private string s not used in decoding; s is used in decapsulation below). We use the binary Goppa code family, as in McEliece's original proposal [39], rather than variants such as the GRS family considered by Niederreiter. See Section 4 for further history.

2.6 Encapsulation

The sender generates a session key K and its ciphertext C as follows:

1. Generate a uniform random vector $e \in \mathbb{F}_2^n$ of weight t.
2. Use the encoding subroutine on e and public key T to compute C_0.
3. Compute $C_1 = \mathsf{H}(2, e)$; see Section 2.8 for H input encodings. Put $C = (C_0, C_1)$.
4. Compute $K = \mathsf{H}(1, e, C)$; see Section 2.8 for H input encodings.
5. Output session key K and ciphertext C.

2.7 Decapsulation

The receiver decapsulates the session key K from ciphertext C as follows:

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. Use the decoding subroutine on C_0 and private key Γ to compute e. If the subroutine returns $\perp$, set $e \leftarrow s$ and $b \leftarrow 0$.
4. Compute $C_1' = \mathsf{H}(2, e)$; see Section 2.8 for H input encodings.
5. If $C_1' \neq C_1$, set $e \leftarrow s$ and $b \leftarrow 0$.

6. Compute $K = \mathsf{H}(b, e, C)$; see Section 2.8 for H input encodings.
7. 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 3 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 5.

Implementors are again cautioned that it is important to avoid leaking secret information through side channels. In particular, the distinction between failures in Step 3, failures in Step 5, 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.

2.8 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 to length between $r+1$ and $r+7$, whichever is a multiple of 8, and then represented as above.

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.

Public keys. The public key T, which is essentially a $mt \times (n - mt)$ 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 $(c_0, c_1, \ldots, c_{m-1}) \in \mathbb{F}_2^m$.

Private keys. A private key has the form $(s, g, \alpha_1, \alpha_2, \ldots, \alpha_n)$. This is represented as the concatenation of three parts:

- The $\lceil n/8 \rceil$-byte string representing $s \in \mathbb{F}_2^n$.
- The $t\lceil m/8 \rceil$-byte string representing $g = g_0 + g_1 x + \cdots + g_{t-1} x^{t-1} + x^t$, namely the concatenation of the representations of the field elements $g_0, g_1, \ldots, g_{t-1}$.
- The representation defined below of the sequence $(\alpha_1, \ldots, \alpha_n)$.

The obvious representation of $(\alpha_1, \ldots, \alpha_n)$ would be as a sequence of n field elements. We specify a different representation that simplifies fast constant-time decoding algorithms: $(\alpha_1, \ldots, \alpha_n)$ are converted into a $(2m-1)2^{m-1}$-bit vector of "control bits" defined below, and then this vector is represented as $\lceil (2m-1)2^{m-4} \rceil$ bytes as above.

Recall that a "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.

Finally, $(\alpha_1, \ldots, \alpha_n)$ are represented as the control bits for a Beneš network that, when applied to all q field elements $(0, z^{12}, z^{11}, z^{12} + z^{11}, z^{10}, z^{12} + z^{10}, \ldots)$ in reverse lexicographic order, produces an array that begins $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and continues with the remaining field elements in some order. An algorithm by Lev, Pippenger, and Valiant [35] computes these control bits at reasonably high speed given the target array.

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

3.1 Parameter set `kem/mceliece6960119`

KEM with $m = 13$, $n = 6960$, $t = 119$, $\ell = 256$. Field polynomial $f(z) = z^{13}+z^4+z^3+z+1$. Hash function: SHAKE256 with 32-byte output.

3.2 Parameter set `kem/mceliece8192128`

KEM with $m = 13$, $n = 8192$, $t = 128$, $\ell = 256$. Field polynomial $f(z) = z^{13}+z^4+z^3+z+1$. Hash function: SHAKE256 with 32-byte output.

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 [47]. After 1978 there were 25 publications studying the one-wayness of the system and introducing increasingly sophisticated non-quantum attack algorithms:

1. 1981 Clark–Cain [18], crediting Omura.
2. 1988 Lee–Brickell [33].
3. 1988 Leon [34].
4. 1989 Krouk [32].

5. 1989 Stern [52].
6. 1989 Dumer [24].
7. 1990 Coffey–Goodman [19].
8. 1990 van Tilburg [55].
9. 1991 Dumer [25].
10. 1991 Coffey–Goodman–Farrell [20].
11. 1993 Chabanne–Courteau [15].
12. 1993 Chabaud [16].
13. 1994 van Tilburg [56].
14. 1994 Canteaut–Chabanne [11].
15. 1998 Canteaut–Chabaud [12].
16. 1998 Canteaut–Sendrier [13].
17. 2008 Bernstein–Lange–Peters [8].
18. 2009 Bernstein–Lange–Peters–van Tilborg [10].
19. 2009 Finiasz–Sendrier [27].
20. 2011 Bernstein–Lange–Peters [9].
21. 2011 May–Meurer–Thomae [37].
22. 2012 Becker–Joux–May–Meurer [3].
23. 2013 Hamdaoui–Sendrier [29].
24. 2015 May–Ozerov [38].
25. 2016 Canto Torres–Sendrier [54].

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 [49] quickly finds $\alpha_1, \ldots, \alpha_n$ given g provided that $n = 2^q$. More generally, whether or not $n = 2^q$, 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 2^q, since then there are many possibilities for the set, along with many possibilities for g; one of our suggested parameter sets provides 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, 40, 42, 44, 41]. 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 [26] 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 suggested parameters use more than 100 errors),

etc. As another example, the attack in [21] reaches degree $m = 2$ where McEliece's original parameters used degree $m = 10$ and where our suggested parameters use degree $m = 13$.

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 modifications suggested by Niederreiter [42].

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.

Niederreiter accomplished 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 becomes about 3.4× slower 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 by Niederreiter 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 Niederreiter's 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 Niederreiter's 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-w column vector of length n; and the ciphertext is a column vector of length n. McEliece's inversion problem is to compute a uniform random input (a, e) given G and the ciphertext $Ga + e$.

Niederreiter's ciphertext instead has the form He. Here H is the unique systematic-form $(n-k) \times n$ parity-check matrix for C, and e is a weight-w 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 a uniform random input e given H and the ciphertext He.

Niederreiter's inversion problem is equivalent to McEliece's inversion problem for the same code. In particular, any attack recovering a random e from Niederreiter's He and H can be used with negligible overhead to recover a random (a, e) from McEliece's $Ga + e$ and G. Specifically, compute H from G, 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.

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-w 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 [22] 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 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 to some extent there are also proofs. Specifically, a KEM construction 15 years ago from Dent [23, Section 6] features a tight proof of security against ROM attacks, assuming OW-CPA security of the underlying PKE; and a very recent KEM construction by Saito, Xagawa, and Yamakawa [48, Theorem 5.2] features a tight proof of security against the broader class of QROM attacks, under somewhat stronger texts, and 32-byte session keys.

5.5 Area

On the medium-sized Altera Stratix V FPGA described above, the `mceliece8192128` hardware design takes 227,750 registers (flip-flops), 129,059 ALMs (55% of available logic resources), 1,126 RAM blocks (44% of available on-chip RAM), and 4 DSP blocks (1.6% of available DSPs). The `mceliece6960119` hardware design takes 223,232 registers (flip-flops), 121,806 ALMs (52% of logic resources), 961 RAM blocks (38% of available on-chip RAM), and 6 DSP blocks (2.3% of available DSPs). Note that this includes only key generation and decoding; full decapsulation and encapsulation will use more space, for example for hashing.

5.6 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 two suggested 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 was to present a KEM with a theorem of the folowing 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 [23, Theorem 8] fourteen years ago. The proof is not very complicated, and should be within the range of current techniques for computer verification of proofs.

However, a very recent paper by Saito, Xagawa and Yamakawa [48] indicates that it is 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.

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.

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 $\textsc{RandomizeSessionKeys}(X_2, \mathsf{H}_1, \mathsf{H}_0)$ 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 Non-quantum reduction

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. [45, 46]). 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 [23, Theorem 8]. First of all, the auxiliary table used by the algorithm simulating H_1 (called $KDFList$ in [23]) 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 [23], 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 Quantum reduction

As noted above, Saito, Xagawa, and Yamakawa very recently introduced a KEM construction "XYZ" with a tight QROM theorem [48, 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 [48, 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 [48] 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 [48]) 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 [48] 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 [26] explicitly studying distinguishing attacks. None of these attacks threaten PR-key security for our suggested 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 [14].

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 [48], is thus not a replacement for a theorem assuming merely OW-CPA security, as in [23, Theorem 8]. Note that the reduction to passive attacks is independent of this choice of assumption.

6.5 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/mceliece6960119`

IND-CCA2 KEM, Category 5.

7.2 Parameter set `kem/mceliece8192128`

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 [47], 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 a 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 [10] and [54].

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.

Our smaller parameter set `mceliece6960119` takes $m = 13$, $n = 6960$, and $t = 119$. This parameter set was proposed in the attack paper [8] that broke the original McEliece parameters $(10, 1024, 50)$.

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 [8] is bottlenecked by random access to a huge array (much

[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, [38] 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.

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 [36], [28], and [43]. The number of choices of g is more than 2^{1500} for our smaller parameter set and more than 2^{1600} for our larger parameter set. Known symmetries provide only a small speedup. The number of choices of $(\alpha_1, \ldots, \alpha_n)$ is much larger. Our smaller parameter set has 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., [31] and [51]. 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. Our expected security levels are so high that this is not a concern for any foreseeable value of T.

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., [57]. 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.

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 suggested 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.

[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.

[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.

[5] Daniel J. Bernstein. Grover vs. McEliece. In Sendrier [50], pages 73–80.

[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, 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Herve Chabanne and B. 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.

[16] 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.

[17] 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.

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

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

[20] John T. Coffey, Rodney M. Goodman, and P. Farrell. New approaches to reduced complexity decoding. *Discrete and Applied Mathematics*, 33:43–60, 1991.

[21] 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.

[22] 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.

[23] 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.

[24] Ilya I. Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25:24–32, 1989.

[25] 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.

[26] 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.

[27] 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.

[28] J. K. 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.

[29] Yann Hamdaoui and Nicolas Sendrier. A non asymptotic analysis of information set decoding. *IACR Cryptology ePrint Archive*, 2013:162, 2013. https://eprint.iacr.org/2013/162.

[30] Gernot Heiser. For safety's sake: we need a new hardware-software contract! *IEEE Design and Test*, 2017. To appear.

[31] 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.

[32] Evgueni A. Krouk. Decoding complexity bound for linear block codes. *Problemy Peredachi Informatsii*, 25:103–107, 1989.

[33] 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.

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

[35] Gavriela Freund Lev, Nicholas Pippenger, and Leslie G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Trans. Computers*, 30(2):93–100, 1981.

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

[37] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $2^{0.054n}$. 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.

[38] 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.

[39] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. Technical report, NASA, 1978. http://ipnpr.jpl.nasa.gov/progressreport2/42-44/44N.PDF.

[40] 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.

[41] 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.

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

[43] 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.

[44] Edoardo Persichetti. Compact McEliece keys based on quasi-dyadic Srivastava codes. *J. Mathematical Cryptology*, 6(2):149–169, 2012.

[45] 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.

[46] Edoardo Persichetti. Code-based key encapsulation from McEliece's cryptosystem. 2017. http://arxiv.org/abs/1706.06306.

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

[48] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. https://eprint.iacr.org/2017/1005.

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

[50] 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.

[51] 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.

[52] 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.

[53] Falko Strenzke. A timing attack against the secret permutation in the McEliece PKC. In Sendrier [50], pages 95–107.

[54] 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.

[55] 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.

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

[57] 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.

[58] Wen Wang, Jakub Szefer, and Ruben Niederhagen. FPGA-based Niederreiter cryptosystem using binary Goppa codes. Paper in submission, http://caslab.csl.yale.edu/code/niederreiter/.