Classic McEliece vs. NTS-KEM

Classic McEliece Comparison Task Force

2018.06.29

Contents

1 Introduction 2

2 Ciphertext size: identical 3

3 Ciphertext details: Classic McEliece is better 4

4 Patent status: identical? 6

5 Chosen-ciphertext attacks: Classic McEliece is better 7

6 Systematic form: Classic McEliece is better 8

7 Permutation security: Classic McEliece is better 12

8 Compressed secret keys: Classic McEliece is better 14

9 Polynomials: Classic McEliece is better 15

10 Public keys: Classic McEliece is better 18

11 Allowing reduced n: Classic McEliece is better 19

12 Specific parameter sets: can argue either way 22

13 Miscellaneous sloppiness 23

14 Current software 24

Chapter 1

Introduction

The original McEliece code-based cryptosystem aimed for OW-CPA security (hardness of inversion). It has a strong security history: dozens of papers over 40 years have tried, with only marginal success, to attack this system.

Two submissions to the NIST post-quantum submission, Classic McEliece and NTS-KEM, are based directly on the McEliece cryptosystem. These submissions do extra work, using a modern hash function (SHAKE256 for Classic McEliece, SHA3-256 for NTS-KEM), to build key-encapsulation mechanisms (KEMs) aiming for IND-CCA2 security.

This document analyzes various differences between Classic McEliece and NTS-KEM.

Chapter 2

Ciphertext size: identical

Size comparison, with the usual notation of n for the code length and k for the code dimension:

ciphertext bits	system
n	original McEliece (not IND-CCA2 secure)
$n-k$	Niederreiter cryptosystem (not IND-CCA2 secure)
$n-k+256$	Classic McEliece submission
$n-k+256$	NTS-KEM submission

The two submissions thus have identical ciphertext sizes.

A dispute about $n-k$ vs. $n-k+256$. Our notes of the NTS-KEM talk at the First PQC Standardization Conference indicate that the speaker was saying

- that Dent's IND-CCA2 conversion (used in Classic McEliece), and every other standard tight conversion, requires extra space, while
- NTS-KEM achieves IND-CCA2 without extra space.

Together these statements imply that NTS-KEM has smaller ciphertexts than Classic McEliece.

However, NTS-KEM actually uses an extra 256-bit hash, i.e., total ciphertext size $n-k+256$ bits, and thus the same ciphertext size as Classic McEliece. See Chapter 3. Furthermore, "implicit rejection" achieves IND-CCA2 security with total ciphertext size $n-k$ bits, i.e., *without* using extra space. See Chapter 5.

We have not found a recording of the NTS-KEM talk, and it is possible that our notes do not accurately reflect what the speaker was saying.

Chapter 3

Ciphertext details: Classic McEliece is better

Notation: sizes and matrices. The following ciphertext description considers a code with parity-check matrix $H = (I_{n-k}|Q)$ and generator matrix $G = \begin{pmatrix} Q \\ I_k \end{pmatrix}$. The public matrix Q is $(n-k) \times k$.

Classic McEliece ciphertexts. Classic McEliece uses Niederreiter ciphertexts. A Niederreiter ciphertext has the form He. The vector e has n bits and specified weight; the vector He has $n-k$ bits.

Classic McEliece extends ciphertexts to include "plaintext confirmation": a 256-bit hash of e (not the hash used as a session key). This was proposed (in much more generality) by Dent in 2002.

NTS-KEM ciphertexts. NTS-KEM uses the following obfuscated form of Niederreiter ciphertexts. Generate a specified-weight n-bit vector

$$e = \begin{pmatrix} e_c \\ e_a \\ e_b \end{pmatrix}$$

where e_c has $n-k$ bits, e_a has $k-256$ bits, and e_b has 256 bits. Compute a 256-bit hash K of e. Send (B, C) as a ciphertext, where

$$B = K + e_b \qquad \text{and}$$

$$C = e_c + Q \begin{pmatrix} e_a \\ K \end{pmatrix}.$$

Note that B has 256 bits, and C has $n-k$ bits.

Details of the obfuscation. Here is how to see that the NTS-KEM ciphertext is an obfuscated Niederreiter ciphertext.

Start with a Niederreiter ciphertext He. Also include Dent's 256-bit plaintext confirmation, a hash of e.

Give names to the following parts of the n-bit vector e: the first $n-k$ bits are e_c; the next $k-256$ bits are e_a; the last 256 bits are e_b. The Niederreiter ciphertext He is then

$$e_c + Q\begin{pmatrix} e_a \\ e_b \end{pmatrix}$$

since $H = (I_{n-k}|Q)$.

Tweak the 256-bit plaintext-confirmation hash by adding the e_b part of the input to the output. In other words, replace Dent's 256-bit plaintext confirmation K with $K + e_b$. This tweak converts a uniform random hash function to a uniform random hash function, so Dent's ROM proof with the original hash function trivially implies a ROM proof with the tweaked hash function.

To review, the ciphertext is now

$$\left(K + e_b, e_c + Q\begin{pmatrix} e_a \\ e_b \end{pmatrix}\right).$$

Define

$$\text{Obfuscate}(B, C) = \left(B, C + Q\begin{pmatrix} 0 \\ B \end{pmatrix}\right)$$

where the 0 has $k-256$ bits. Anyone can compute this Obfuscate function, since Q is public; furthermore, Obfuscate is its own inverse. Apply this Obfuscate function to the ciphertext:

$$\begin{aligned} \text{Obfuscate}\left(K + e_b, e_c + Q\begin{pmatrix} e_a \\ e_b \end{pmatrix}\right) &= \left(K + e_b, e_c + Q\begin{pmatrix} e_a \\ e_b \end{pmatrix} + Q\begin{pmatrix} 0 \\ K + e_b \end{pmatrix}\right) \\ &= \left(K + e_b, e_c + Q\begin{pmatrix} e_a \\ K \end{pmatrix}\right). \end{aligned}$$

This obfuscated ciphertext is exactly the NTS-KEM ciphertext (B, C).

Comparison. The obfuscation embedded into NTS-KEM complicates the system description and has no compensating advantages.

Chapter 4

Patent status: identical?

Originally NTS-KEM claimed patents: U.S. patent application 20150163060 and G.B. patent application 2532242. It was never clear exactly which parts of NTS-KEM were claimed to be covered.

The NTS-KEM submitters wrote that they had "a US patent application and a granted UK patent describing a method by which a McEliece ciphertext may be shortened and have the same security as the full length McEliece ciphertext" and that this was used "in no other PQC submission as far as we can tell". A request for clarification went unanswered. It *seems* that what was patented is the obfuscation described in Chapter 3. Note again that Niederreiter's compression already achieved this ciphertext length in 1986; as shown in the previous section, NTS-KEM does *not* have shorter ciphertexts.

The NTS-KEM submitters also wrote that they had "no wish to place any constraints whatsoever" on people wishing to use this method. A request for clarification ("Are you saying that you're happy for people to use NTS-KEM as long as they pay you?") went unanswered.

In an email to the `pqc-forum`, dated 27 April 2018, the NTS-KEM team wrote "We have decided to eliminate any uncertainty by abandoning the patent with immediate effect. Our submission will no longer be subject to any patents and is free for anyone to experiment with." On the other hand, as of 17 June 2018,

https://www.ipo.gov.uk/p-ipsum/Case/PublicationNumber/GB2532242

shows that the UK patent was granted with granting date of 5 June 2018, and does not show any abandonment of the patent. Also, the "Image File Wrapper" available from https://portal.uspto.gov/pair/PublicPair for application 14/596098 does not show any abandonment of the U.S. patent application.

The Classic McEliece team never had, and never will have, any patents on Classic McEliece.

Chapter 5

Chosen-ciphertext attacks: Classic McEliece is better

Classic McEliece has two layers of defense against chosen-ciphertext attacks: plaintext confirmation as per Dent; and implicit rejection as per Persichetti's thesis. NTS-KEM has only the first defense.

The NTS-KEM presentation emphasized having a tight ROM IND-CCA2 security theorem from OW-CPA. The new Bernstein–Persichetti paper "Towards KEM Unification" (`https://eprint.iacr.org/2018/526`) presents full details of a tight ROM security theorem for Classic McEliece which achieves IND-CCA2 security from OW-CPA of the underlying cryptosystem.

A proof along the lines of Dent was already outlined in the Classic McEliece submission, but it turns out that, thanks to the implicit-rejection technique, it is possible to obtain a much simpler proof.

The same proof strategy should also allow QROM IND-CCA2 security to be obtained tightly from the Bernstein–Persichetti "IND-Hash" notion, which is very close to OW-CPA. There don't seem to be any tight QROM proof strategies applicable to NTS-KEM.

Any ROM or QROM proof for NTS-KEM can be converted into a ROM or QROM proof for Classic McEliece: strip away the obfuscation explained in Chapter 3, and replace decapsulation failures with pseudorandom results. A conversion the other way doesn't work, since it would have to figure out the pattern of decapsulation failures.

Chapter 6

Systematic form: Classic McEliece is better

Classic McEliece. The original McEliece system generates, independently, a uniform random polynomial and a uniform random permutation.

Classic McEliece, following Niederreiter, converts the public key to systematic form. This works about 29% of the time. If this does not work, Classic McEliece starts over with a new polynomial and permutation.

NTS-KEM. NTS-KEM claims an advantage here (page 24 of the submission, last sentence): a key-generation attempt works 100% of the time instead of 29%. The point is that NTS-KEM reorders columns "if necessary" during the conversion to systematic form: i.e., it applies a further permutation to ensure that the matrix can be converted to systematic form.

A non-full-rank matrix will still fail. It seems likely that the probability of encountering a non-full-rank matrix is below 2^{-256}, since NTS-KEM requires k to be larger than 256; this failure probability does not sound like a problematic deviation from NIST's submission requirements. On the other hand, there is no proof, and having a key-generation algorithm not always work is an extra complication for auditors. This is easy to fix: restart key generation in the failure cases.

To avoid confusion, the following description says "permuted systematic form" for the NTS-KEM output, and "unpermuted systematic form" for the Classic McEliece output.

Comparison. Of course reducing the number of key-generation attempts by a factor 3.5 sounds good, and maybe there are applications where this makes

a difference between acceptable and unacceptable costs. However, this analysis presumes that an NTS-KEM key-generation attempt, reducing the matrix to permuted systematic form, has the same cost as a Classic McEliece key-generation attempt, reducing the matrix to unpermuted systematic form.

NTS-KEM uses a variable-time algorithm to reduce to permuted systematic form. This is not (or at least should not be!) acceptable for auditors. Sometimes people argue that timing attacks against key generation are difficult since key generation is done only once; but the Aldaya–García–Tapia–Brumley paper "Cache-Timing Attacks on RSA Key Generation" (https://eprint.iacr.org/2018/367) shows that this argument is flawed.

It is of course theoretically possible to write down a constant-time algorithm to reduce to permuted systematic form. But how expensive is this, compared to a constant-time algorithm for unpermuted systematic form? If the answer is that it is 10× more expensive then there is *more* overall time spent on key generation. Even worse, this sets a bad precedent for implementors, who will be tempted to use variable-time algorithms (as illustrated by the NTS-KEM submission), creating unnecessary conflicts between simplicity, security, and speed.

Algorithm analysis: unpermuted systematic form. Consider a matrix with $n - k$ rows and n columns; e.g., 1664 rows and 8192 columns. The first step in converting to unpermuted systematic form is as follows:

- Search the first column for its "pivot", which by definition is the first nonzero entry. (If there is no pivot then the unpermuted-systematic-form computation fails.)
- Swap the pivot row with the first row.
- XOR the first row into other rows that have nonzero entries in this column.

The McBits paper instead suggests the following constant-time strategy:

- Conditionally XOR the second row into the first row, where the condition is that the first row has leading entry 0.
- Conditionally XOR the third row into the first row, where the condition is that the first row (now) has leading entry 0.
- Et cetera.
- Conditionally XOR the first row into the second row, where the condition is that the second row has leading entry 1.
- Conditionally XOR the first row into the third row, where the condition is that the third row has leading entry 1.

- Et cetera.

The next step works through the second column in a similar way, searching for a pivot starting from the second row. The same procedure continues for $n - k$ steps.

Overall the number of positions searched for pivots is about $(n-k)^2/2$, while there are about $3n(n-k)^2/2$ XORs of entries across rows. It is important to understand that the time ratio between these two operations is far below the ratio $3n$ of bits handled: rows are stored as sequences of words (or CPU vectors), so (conditionally) XOR'ing one row into another typically handles 32 or 64 (or even more) entries at a time.

It is possible to save more time by merging the vector operations here into matrix multiplications and using fast matrix-multiplication techniques, although the current software does not do this. It is also possible to save time by handling the first $n-k$ columns, checking for systematic form, aborting in the failure case, and then carrying out the corresponding operations on the remaining k columns.

Algorithm analysis: permuted systematic form. Converting to *permuted* systematic form means that if there is no pivot in the first column then there will be a permutation of columns. To find the first pivot row, one must then search the second column, possibly the third column, etc. Doing this in constant time means always searching $k+1$ columns (assuming the matrix has full rank), and also doing k conditional swaps of columns, i.e., $k(n-k)$ conditional swaps of entries across columns. Similar comments apply to subsequent pivots.

Overall there are about $k(n-k)^2/2$ positions searched for pivots, and about $k(n-k)^2$ swaps of entries across columns, on top of the previous $3n(n-k)^2/2$ XORs of entries across rows. The operation count is only about twice what it was before, but the time is much worse, since it is much more difficult to pack the pivot searches and column swaps into word operations. Storing the matrix by columns instead of by rows simply shifts the problem to the row operations.

This analysis suggests that a constant-time permuted-systematic-form algorithm (under the full-rank assumption) is much slower than a constant-time unpermuted-systematic-form algorithm. It might be possible to do better, for example with a constant-time version of the techniques of "Matrix inversion made difficult", but it is certainly not obvious that this would save enough time to be competitive.

To summarize, the current situation is that constant-time permuted-systematic-form software seems unlikely to be competitive in key-generation speed.

A different possibility. What happens if we try p positions for each pivot in the constant-time permuted-systematic-form algorithm sketched above, and restart key generation if the positions are all 0?

Presumably each key-generation attempt has about an $(n-k)/2^p$ chance of failure. The point here is that the input bits seem reasonably random. This can be checked experimentally for moderate values of p, and p can be chosen to optimize the average key-generation time; presumably p is then below 30. Or one could take, say, $p = 128$ for applications that are willing to use slightly more time in exchange for more confidence that the first key-generation attempt will work. "Real-time" applications can safely budget the time for (say) four key-generation attempts, if the auditor checks experimentally that each attempt has failure probability below (say) 2^{-32}.

For most of the pivots, the p positions can be, say, the last p positions in the column checked for the constant-time unpermuted-systematic-form algorithm, saving time. There is a slowdown only for the last $p-1$ pivots, where the positions will have to stretch across two or more columns.

As explained in Chapter 7, it is important to check that the output is defined entirely by the code, and not by any extra information in the input matrix.

Chapter 7

Permutation security: Classic McEliece is better

Classic McEliece. Classic McEliece is designed so that any inversion attack (OW-CPA attack) against Classic McEliece implies an inversion attack against the original McEliece system with probability about 29% and with almost identical speed.

To prove this implication, simply convert the McEliece public key and ciphertext into a public key and ciphertext for Classic McEliece, by reducing the McEliece public key to (unpermuted) systematic form. This reduction works with probability about 29%, the same probability that a key-generation attempt works. When it works, this process produces the same distribution of Classic McEliece public keys (and corresponding ciphertexts) that are produced by the Classic McEliece key-generation (and encryption) algorithms.

This is not a two-way implication: it's conceivable that Classic McEliece is *more* secure than original McEliece. However, there are no known attacks separating these. More importantly, the whole point is to obtain confidence in the security of Classic McEliece from the long study of original McEliece.

NTS-KEM. NTS-KEM claims to have the same feature: namely, that any inversion attack against NTS-KEM implies an inversion attack against the original McEliece system with high probability and with almost identical speed.

There is, however, a security deficiency at this point in the NTS-KEM specification: namely, the permutation-update mechanism (see Chapter 6) is not specified. Consequently, there is no reason to believe that the permutation update is defined entirely by the *code* provided as input. To see that this is important, note that a malicious permutation-update mechanism can easily leak secrets through entries of the matrix that it produces as output. The security

theorem claimed for NTS-KEM does not make any assumptions excluding such a mechanism, so the theorem cannot be correct as stated.

If the permutation update is defined entirely by the code provided as input, then any inversion attack against NTS-KEM implies an inversion attack against the original McEliece system with almost identical speed: simply convert the McEliece public key and ciphertext into a public key and ciphertext for NTS-KEM, permuting the ciphertext along with the public key. It is important here that converting the original secret Goppa matrix directly into the NTS-KEM public key produces the same result as converting it first into the McEliece public key and then into the NTS-KEM public key; this is where the "defined entirely by the code" assumption is used.

The NTS-KEM reference implementation appears to *first* compute reduced row-echelon form, and *then* compute a permutation update as a function of the reduced row-echelon form. However, the specification ("transform $\mathbf{H}$ to reduced row echelon form, re-ordering its columns if necessary, such that the identity matrix $\mathbf{I}_{n-k}$ occupies the last $n - k$ columns of $\mathbf{H}$") does not require this order of operations.

Chapter 8

Compressed secret keys: Classic McEliece is better

It is straightforward to generate the secret permutation in Classic McEliece from a 256-bit seed: use the seed with a stream cipher to generate random numbers, and sort the random numbers to determine the permutation. Similar comments apply to the secret polynomial.

As discussed earlier, the permutation and polynomial are acceptable with probability 29%. Key generation writes down a parity-check matrix and tries to reduce the matrix to (unpermuted) systematic form. If this fails, key generation starts over with a new seed (e.g., a hash of the previous seed).

A user has the option of storing only the seed—the final 256-bit seed; this is smaller and faster than storing the original seed and the number of hashes required. The user then expands the seed into the permutation and polynomial whenever desired. Note that this does not require the space and time for the matrix computations that took place during key generation.

For NTS-KEM, as discussed earlier, a further permutation is applied during key generation. This permutation is determined by a matrix computation. If the user stores only the 256-bit seed, then expanding the seed into the permutation and polynomial seems to require the same matrix computations—much more time and space than for Classic McEliece. There does not seem to be any way to avoid these costs without storing additional information about the further permutation.

One way to represent the further permutation would be as a sequence of swaps. The number of swaps required for the further permutation is not very large on average. However, variable-length secret keys raise questions regarding timing attacks. It is not clear how many extra bits are required for constant-length compressed keys that work with acceptable probability.

Chapter 9

Polynomials: Classic McEliece is better

Classic McEliece uses monic irreducible Goppa polynomials. NTS-KEM uses monic squarefree (i.e., separable) Goppa polynomials without linear factors.

Mathematical background: the number of polynomials. Assume $t \geq 2$. Then the set of monic degree-t irreducible polynomials over the finite field $\mathbf{F}_q$ is a subset of the set of monic degree-t squarefree polynomials over $\mathbf{F}_q$ without linear factors.

Define δ as the probability that a monic degree-t squarefree polynomial without linear factors is irreducible. The point of the following calculations is that $\delta \approx \exp(1)/t$.

The number of monic degree-t irreducible polynomials is $(1/t)\sum_{d|t}\mu(d)q^{t/d} \approx q^t/t$, where μ is the Möbius function. For example, for $t = 64$, this number is $(q^{64} - q^{32})/64 \approx q^{64}/64$; i.e., about 2^{762} for $q = 4096$.

The number of monic degree-t squarefree polynomials without linear factors is the coefficient of x^t in the power series $(1-qx^2)/((1-qx)(1+x)^q)$. For example, for $t = 64$ and $q = 4096$, this number is $0.36783451848291\ldots \cdot q^{64} \approx 1.5 \cdot 2^{766}$. To understand the factor $0.36783451848291\ldots$ here, note that there are

- q monic linear polynomials, each of which divides a uniform random monic degree-t polynomial with probability $1/q$ since $t \geq 1$;
- $(q^2 - q)/2$ monic irreducible degree-2 polynomials, each of which has a *square* dividing a uniform random monic degree-t polynomial with probability $1/q^4$ if $t \geq 4$;

- $(q^3 - q)/3$ monic irreducible degree-3 polynomials, each of which has a *square* dividing a uniform random monic degree-t polynomial with probability $1/q^6$ if $t \geq 6$;
- etc.

If these probabilities were independent then the overall chance of non-divisibility would be $(1 - 1/q)^q(1 - 1/q^4)^{(q^2-q)/2}(1 - 1/q^6)^{(q^3-q)/3} \cdots$. In particular, for $q = 4096$, the first factor $(1 - 1/q)^q$ is $0.36783452944434\ldots$; the second factor is $0.99999997020495\ldots$; the third factor is $0.99999999999514\ldots$; etc. The probabilities are not exactly independent, but they are close to independent when t is not very small. As q increases, the first factor converges to $1/\exp(1) = 0.36787944117144\ldots$, and the remaining factors converge to 1.

The number of keys. Consider a McEliece key generated using a monic degree-t squarefree polynomial without linear factors. Define δ' as the probability that the polynomial is irreducible. Experiments show that the squarefree and irreducible cases have practically identical probabilities of producing keys (see Chapter 6), so $\delta' \approx \delta \approx \exp(1)/t$.

Relative security. In the original McEliece paper, the user "randomly selects an irreducible polynomial of degree t"; i.e., irreducible polynomials were the original choice in the McEliece cryptosystem. Preserving this choice trivially preserves the security level, whereas changing to squarefree polynomials might change the security level.

The central argument to preserve McEliece's original choice is that changing to squarefree polynomials could conceivably produce a large drop in security level. Perhaps the only secure cases are the irreducible cases, while all other cases are much more efficiently breakable. There are many other areas of cryptography where the existence (and/or knowledge) of nontrivial factors helps the attacker; there has been little study of the impact of factors in the McEliece context.

A counterargument is that, compared to original McEliece, changing to squarefree polynomials could conceivably *increase* the security level. There are two reasons that this counterargument is less convincing than the argument.

First, as noted earlier, the whole point is to obtain confidence in security from the long study of the original McEliece system. Having much *less* security than the original system would be a serious flaw in the system design. Having much *more* security than the original system would be nice but is not necessary.

Second, changing to squarefree polynomials cannot increase the security level by more than $\log_2(1/\delta') \approx \log_2 t - 1.44$ bits, assuming $t \geq 2$. If $\log_2 t$ is small (which it is for both Classic McEliece and NTS-KEM), the potential gain in security level brought by switching from irreducible to squarefree is also small.

Specifically, consider an inversion algorithm with success probability ϵ at breaking monic degree-t irreducible polynomials. Run the same algorithm against a monic degree-t squarefree polynomial without linear factors. The polynomial has probability δ' of being irreducible, so the algorithm now has success probability *at least* $\delta'\epsilon$.

Entropy. An argument for squarefree is that there are more squarefree polynomials, i.e., there is a larger space of possible secrets. However, this expansion cannot gain more than about $\log_2 t - 1.44$ bits of security, while it could lose much more security; see above.

Algorithms to generate polynomials. One can generate a random irreducible polynomial with probability extremely close to 1 by computing the minimal polynomial of a random element of the field defined by a standard irreducible polynomial of that degree. Computing minimal polynomials is a simple matter of linear algebra.

Recognizing a squarefree polynomial is a simple matter of checking for common factors between the polynomial and its derivative; checking for linear factors is also simple.

Further analysis is required of implementation security (e.g., checking for common factors is typically done by a variable-time gcd computation) and performance.

Per-user choices. The choice between squarefree and irreducible is visible only to the secret-key holder, so it would be possible to let different secret-key holders make different choices. However, settling on one choice is better for security of the ecosystem. A similar comment applies in Chapter 6.

Chapter 10

Public keys: Classic McEliece is better

Size comparison:

public-key bits	system
$k(n-k)$	Classic McEliece submission
$k(n-k)$	NTS-KEM software
$k(n-k)+32$	NTS-KEM definition in NTS-KEM specification

Classic McEliece public keys are marginally shorter and marginally simpler than public keys defined in the NTS-KEM specification, since NTS-KEM spends 4 extra bytes in public keys to communicate the parameters $(t, 256)$. See the definition of "public key" on the top of page 12 of the specification, and footnote 7 of the specification ("We assume that the public key values τ and ℓ are stored in two-byte strings each").

However, there appears to be a discrepancy between (1) the definition in the NTS-KEM specification, (2) the specific key sizes in the NTS-KEM specification, and (3) the NTS-KEM software. The specific key sizes and software appear to skip the 4 bytes for τ and ℓ.

Chapter 11

Allowing reduced n: Classic McEliece is better

Classic McEliece allows n below 2^m. NTS-KEM describes "a high degree of flexibility in the setting of parameters" as an advantage, but requires specifically $n = 2^m$.

The best tradeoffs between key size and security often require n to be below 2^m. See Figure 11.1. For example, within the constraint of keeping keys below a fixed size (specifically a megabyte), Classic McEliece obtains several bits more security with n below 2^m than can be obtained under the restriction $n = 2^m$. Similarly, the 200-bit security level is too high to be reached with $n = 4096$; the restriction $n = 2^m$ forces keys to be 5694975 bits, while allowing other values of n reduces keys below 4800000 bits. The 2008 Bernstein–Lange–Peters paper already describes this effect and has a numerical example of it.

Classic McEliece also points to n below 2^m as creating an additional obstacle to support splitting. In other words, $n < 2^m$ is more paranoid than $n = 2^m$.

NTS-KEM argues that "implementations are cleaner and usually faster" with the restriction $n = 2^m$. There is a little bit of truth to this, but not much, and people who want the best tradeoff between security and key size will be willing to spend the implementation effort.

NTS-KEM argues that taking $n = 2^m$ "minimises the length of the ciphertext" at any particular security level, and claims that "We chose to make shortening the ciphertext length our top design priority". However, NTS-KEM parameters aren't actually chosen for minimum ciphertext length (even if one adds a requirement for parameters to avoid high-rate distinguishers). For example, NTS-KEM selects $(n, w) = (8192, 136)$ with 1768-bit ciphertexts (and 11357632-bit keys), but at approximately the same security level it could instead have selected $(n, w) = (16384, 81)$ with 1134-bit ciphertexts (and 13672764-bit keys),

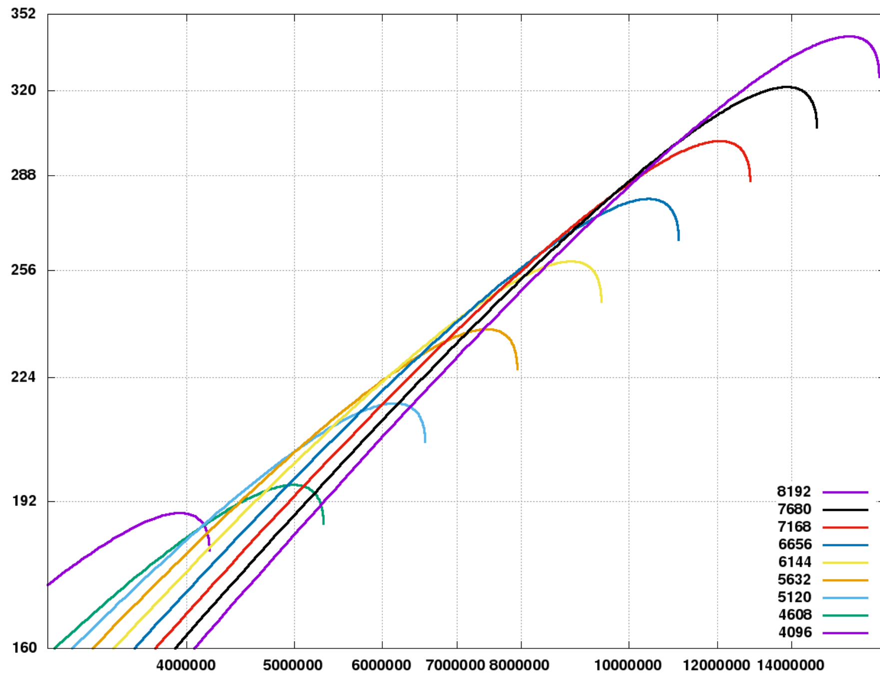

Figure 11.1: Horizontal axis: Key size $k(n-k)$ in bits. Vertical axis: Security level according to the BLP formulas. Purple curve on the left: $n = 4096$ with various choices of error weight. Purple curve on the right: $n = 8192$ with various choices of error weight. Each curve limits k to the range $[n/2, n]$.

or $(n, w) = (24576, 70)$ with 1050-bit ciphertexts (and 24702300-bit keys).

NTS-KEM argues that "setting parameters is also simpler" with the restriction $n = 2^m$. However, the main work in setting parameters is choosing a function that maps parameters to a score, and, unless the function is extremely slow, it will easily finish for the whole range of Classic McEliece parameters.

Chapter 12

Specific parameter sets: can argue either way

Classic McEliece proposes two level-5 options:

- `mceliece6960119`: $n = 6960$ with $t = 119$.
- `mceliece8192128`: $n = 8192$ with $t = 128$.

For the same level NTS-KEM has $n = 8192$ with $t = 136$. NTS-KEM also proposes $n = 8192$ with $t = 80$ for level 3, and $n = 4096$ with $t = 64$ for level 1.

An argument for supporting multiple levels is that (maybe) some applications will care about the performance differences between level 1 and level 5. An argument against supporting multiple levels is that the whole point here is to be conservative.

NTS-KEM always takes t as a multiple of 8 and claims that this helps avoid software bugs.

Chapter 13

Miscellaneous sloppiness

NTS-KEM allows t ("τ"), the number of errors, to be 1, in which case key generation loops forever. Classic McEliece requires t to be at least 2, avoiding this mistake. This doesn't really matter since t is always chosen much larger.

NTS-KEM claims incorrectly that list decoding creates decryption failures. For the moment no submission is recommending list decoding.

Chapter 14

Current software

Classic McEliece already has constant-time software. This is a major advantage.

For encapsulation and decapsulation, the Classic McEliece software is generally faster than the NTS-KEM software. One big slowdown in the NTS-KEM software is a naive syndrome computation rather than a transposed additive FFT.

The NTS-KEM software uses M4RI for key generation, saving time, but this isn't constant-time. See Chapter 6.

The NTS-KEM software uses variable-time shuffling methods. These won't beat constant-time sorting in speed.