
From: Kirk Fleming <kpfleming@mail.com>
Sent: Monday, November 9, 2020 8:35 PM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 3 OFFICIAL COMMENT: Classic McEliece

There is a strong possibility that NIST intend to standardize Classic McEliece at the end of Round 3 as a conservative option. Their Status Report on the Second Round expressed confidence in its construction and said that they believed it could be ready for standardization if selected. Standardizing Classic McEliece, however, also means standardizing at least some of the parameter sets proposed by the submitters.

I am filing this comment to request that the Classic McEliece submitters justify the claimed security categories for their parameters.

The Round 3 submission does not include any concrete analysis of the security provided by the proposed parameter sets. There is a lot of hype about the asymptotic result from Canto Torres and Sendrier but this is ultimately irrelevant for any finite choice of parameters. The only firm statement contained in the submission is:

"[Bernstein, Lange and Peters. 2008] 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} ."

The submission argues that any reduction in classical security from improved ISD attacks will be offset by the increased memory requirements. While this may be true for some ISD variants such as BJMM, they provide no analysis to back this up. It also ignores other ISD variants such as Stern that need significantly less memory. In this case the finite regime analysis from the LEDACrypt team gives the following estimates of computational and memory costs for the pre-merger Classic McEliece and NTS-KEM parameters:

Parameter Set	Compute	Memory	Target
mceliece-3488-064	152.51	34.68	143
mceliece-4608-096	194.36	35.66	207
mceliece-6688-128	270.46	37.48	272
mceliece-6960-119	271.18	47.58	272
mceliece-8192-128	306.63	67.64	272
nts-kem-4096-064	166.76	35.60	143
nts-kem-8192-080	248.01	77.63	207
nts-kem-8192-136	313.52	67.50	272

By this analysis:

- The mceliece-4608-096 parameters are about 13 bits below Category 3 with an attack that needs slightly over 6 GiB of memory.
- The mceliece-6688-128 parameters are borderline Category 5 with an attack that needs slightly over 22 GiB of memory.
- The mceliece-6960-119 parameters are borderline Category 5 with an attack that needs around 24 TiB of memory.

If Classic McEliece is to be standardized as a conservative option then the parameter sets that are standardized with it should also be chosen conservatively. The NTS-KEM parameters were. Three out of the five Classic McEliece parameters were not.

From: Kirk Fleming <kpffleming@mail.com>
Sent: Wednesday, November 11, 2020 12:20 PM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Kirk Fleming wrote:

> In this case the finite regime analysis from the LEDACrypt team gives the following
> estimates of computational and memory costs for the pre-merger Classic McEliece
> and NTS-KEM parameters:

It was pointed out that my finite regime analysis reference was unclear. The figures were taken from Tables 4 and 5 in Baldi, Barenghu, Chiaraluce, Pelosi and Santini, "A finite regime analysis of Information Set Decoding algorithms", Algorithms 12, no. 10 (2019). The paper can be found at <https://www.mdpi.com/1999-4893/12/10/209/pdf>.

Kirk

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Thursday, November 19, 2020 7:38 AM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

Kirk Fleming writes:

> mceliece-3488-064	152.51	34.68	143
> mceliece-4608-096	194.36	35.66	207
> mceliece-6688-128	270.46	37.48	272
> mceliece-6960-119	271.18	47.58	272
> mceliece-8192-128	306.63	67.64	272

We agree that the second column is sometimes less than the fourth column.

However, the second column ignores the huge overheads measured by the third column. As stated in the submission:

A closer look shows that the attack in [11] is bottlenecked by random access to a huge array (much larger than the public key being attacked), and that subsequent ISD variants use even more memory. The same amount of hardware allows much more parallelism in attacking, e.g., AES-256.

The submission goes on to state expectations regarding (e.g.) 6960119 being more expensive to break than AES-256. The numbers you quote (e.g., $2^{271.18}$ operations on $2^{47.58}$ bits of RAM) are consistent with these expectations.

The literature shows that optimized hardware for AES attacks has similar efficiency to multipliers. The multipliers in an Intel CPU core carry out millions of bit operations in the same time that it takes the core to retrieve data from (say) 6GB of RAM, never mind RAM contention across cores. GPUs and TPUs show the costs of RAM even more clearly. On a larger scale, decades of supercomputing literature consistently fit a square-root scaling of RAM costs, and science-fiction 3D computers (which seem much harder to build than quantum computers) would still have cube-root costs.

> I am filing this comment to request that the Classic McEliece
> submitters justify the claimed security categories for their parameters.

Can you please clarify how you think that the numbers already in the literature don't already justify the assignments in the submission?

Our best guess is that you're assuming that NIST requires $\geq 2^{272}$ operations in a specified metric for "category 5", and that $2^{271.18}$ is an evaluation in this metric. But the assumption here isn't correct.

NIST has not issued clear definitions of the cost metrics for its "categories". If at some point NIST does issue clear, stable definitions of its cost metrics, then it should also allow all submitters to set their "category" assignments accordingly.

Note that one can make any cryptosystem sound much easier to break by choosing a cost metric that assigns unrealistically low cost to operations used in attacks; RAM access is just one example. If the same operations are not bottlenecks in attacks against AES-m or SHA-n then this can reverse comparisons to AES-m or SHA-n respectively.

- > The Round 3 submission does not include any concrete analysis of the
- > security provided by the proposed parameter sets.

Not true. For example, the stated expectation to be "more expensive to break than AES-256" is concrete, and the rationale is stated. A more detailed analysis depends on the cost metric.

- > There is a lot of hype about the asymptotic result from Canto Torres
- > and Sendrier

The graph in <https://video.cr.yt.to/2020/0813/video.html> shows that asymptotic lattice security levels against attacks known 10 years ago were 42% higher than lattice security levels against attacks known today (for otherwise unbroken lattice systems), while for McEliece's system the change is 0%. Any thorough evaluation of security risks will take these facts into account, along with other risk indicators.

- > but this is ultimately irrelevant for any finite choice of parameters.

The submission already says, at the top of the "Information-set decoding, concretely" section:

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

The section goes on to summarize what the literature says about concrete cost analyses, and ends with the AES-256 comparison reviewed above.

- > The submission argues that any reduction in classical security from
- > improved ISD attacks will be offset by the increased memory
- > requirements. While this may be true for some ISD variants such as
- > BJMM, they provide no analysis to back this up.

Again, here's the critical quote from the submission:

A closer look shows that the attack in [11] is bottlenecked by random access to a huge array (much larger than the public key being attacked), and that subsequent ISD variants use even more memory. The same amount of hardware allows much more parallelism in attacking, e.g., AES-256.

The exact impact depends on the cost metric.

- > It also ignores other
- > ISD variants such as Stern that need significantly less memory.

No, the submission doesn't ignore this. The "attack in [11]" (see quote above) is a Stern-type attack. The submission's comment that "subsequent ISD variants use even more memory" (see quote above) is pretty much the same as your comment that Stern-type attacks "need significantly less memory".

- > - The mceliece-4608-096 parameters are about 13 bits below Category 3 with
- > an attack that needs slightly over 6 GiB of memory.

Evaluating such a claim would require a clear, stable definition of "Category 3". If NIST settles on a definition to be used in NISTPQC then submitters will be able to make "category" assignments accordingly.

See above regarding the actual costs of accessing 6GB of RAM.

> If Classic McEliece is to be standardized as a conservative option
> then the parameter sets that are standardized with it should also be
> chosen conservatively. The NTS-KEM parameters were. Three out of the
> five Classic McEliece parameters were not.

Please clarify. Are you saying anything here beyond the 2^{194} vs. 2^{207} ,
 2^{270} vs. 2^{272} , and 2^{271} vs. 2^{272} discussed above?

If you're concerned about 2^{270} RAM operations vs. 2^{272} bit operations then you should be far more concerned about NIST dropping the targets from 2^{256} to 2^{128} :

* Multi-target attacks turn 2^{128} into something already feasible for large-scale attackers today. ECC is an exception to this (provably, via a tight self-reduction), but codes and lattices and AES aren't exceptions. See <https://blog.cr.y.p.to/20151120-batchattacks.html>.

* Both round-1 Classic McEliece options were at the 2^{256} level. However, NIST then praised NTS-KEM for offering lower-security options, and asked Classic McEliece for "parameter sets for other security categories".

From this perspective, NTS-KEM and NIST were far less conservative than Classic McEliece was. Note that general trends in networking and in CPUs are making the high-security Classic McEliece parameter sets affordable for more and more users.

There's also a tremendous amount to say about the dangers of unnecessary complexity. One aspect of this is specifying more parameter sets than necessary. Of course, NIST's request for lower-security parameter sets was an offer that Classic McEliece couldn't refuse, but we continue to recommend the 2^{256} parameter sets. Within those parameter sets:

* The 6960119 parameter set goes back to 2008, maximizing security for 1MB keys. This has held up just fine, and can reasonably be viewed as the default choice.

* The new 6688128 parameter set is a slight variant that requires n and t to be multiples of 32, which *might* be slightly better for formal verification, but the jury is still out on this. (Components are being verified---see <https://cr.y.p.to/papers.html#controlbits> for the latest news---but this work hasn't yet reached the components that could interact with the multiple-of-32 question.)

* The 8192128 parameter set is bigger, but the 6960119 and 6688128 parameter sets include an extra defense explained in the submission. People paranoid enough to imagine 2^{306} vs. 2^{270} making a difference should also be paranoid enough to imagine this defense making a difference.

One can easily write down even larger parameter sets. Also, the ISD attack analyses in the literature are precise and straightforward, and it should be easy to evaluate the security of any desired parameters in any well-defined metric.

However, the right order of events is, first, for NIST to fully define the cost metric to be used for "categories", and, second, for all submission teams to evaluate costs in this metric.

---Dan (on behalf of the Classic McEliece team)

From: pqc-forum@list.nist.gov on behalf of Kirk Fleming <kpfleming@mail.com>
Sent: Monday, November 23, 2020 8:01 PM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Dan Bernstein wrote:

> Kirk Fleming writes:

>>

>> mceliece-3488-064 152.51 34.68 143

>> mceliece-4608-096 194.36 35.66 207

>> mceliece-6688-128 270.46 37.48 272

>> mceliece-6960-119 271.18 47.58 272

>> mceliece-8192-128 306.63 67.64 272

>

> We agree that the second column is sometimes less than the fourth column.

> However, the second column ignores the huge overheads measured by the

> third column.

No. The second column includes a logarithmic memory overhead. This is not an accurate analysis of the cost of memory accesses but it does not ignore it.

> As stated in the submission:

>

> A closer look shows that the attack in [11] is bottlenecked by random

> access to a huge array (much larger than the public key being

> attacked), and that subsequent ISD variants use even more memory. The

> same amount of hardware allows much more parallelism in attacking,

> e.g., AES-256.

This is the second time you've used the word "huge". Let's give it some perspective. $2^{35.66}$ bits is less RAM than the laptop I'm using to write this. $2^{47.58}$ bits is the same as Amazon EC2's u-24tb1.metal instance. Maybe the "u" stands for 'uge'.

> The submission goes on to state expectations regarding (e.g.) 6960119

> being more expensive to break than AES-256. The numbers you quote (e.g.,

> $2^{271.18}$ operations on $2^{47.58}$ bits of RAM) are consistent with these

> expectations.

Your submission and this reply focus almost exclusively on the mceliece-6960-119 parameter set. What about the other four?

> The literature shows that optimized hardware for AES attacks has similar

> efficiency to multipliers. The multipliers in an Intel CPU core carry

> out millions of bit operations in the same time that it takes the core

> to retrieve data from (say) 6GB of RAM, never mind RAM contention across

> cores. GPUs and TPUs show the costs of RAM even more clearly. On a

> larger scale, decades of supercomputing literature consistently fit a

> square-root scaling of RAM costs, and science-fiction 3D computers

> (which seem much harder to build than quantum computers) would still

> have cube-root costs.

No. It's not as simple as applying a square-root overhead for memory across the board. For an interesting (but not very rigorous) example see http://www.ilikebigbits.com/2014_04_21_myth_of_ram_1.html. The blog argues for a square-root memory cost. If you look at the graphs this is a poor fit in the 10 MiB to 6 GiB range.

Not all memory accesses are the same. The memory overhead depends on the number of random accesses to high-latency memory. One cost model in Ray Perlner's slides assumes that up to 2^{50} bits of memory is local with unit cost memory accesses.

>> I am filing this comment to request that the Classic McEliece submitters
>> justify the claimed security categories for their parameters.

>
> Can you please clarify how you think that the numbers already in the
> literature don't already justify the assignments in the submission?

Let's try an easier question. Can you point to the paper cited by your submission that gives numbers for the mceliece-4608-096 parameter set which clearly justify its assignment as category 3?

> Our best guess is that you're assuming that NIST requires $\geq 2^{272}$
> operations in a specified metric for "category 5", and that $2^{271.18}$ is
> an evaluation in this metric. But the assumption here isn't correct.
> NIST has not issued clear definitions of the cost metrics for its
> "categories". If at some point NIST does issue clear, stable definitions
> of its cost metrics, then it should also allow all submitters to set
> their "category" assignments accordingly.
>
> Note that one can make any cryptosystem sound much easier to break by
> choosing a cost metric that assigns unrealistically low cost to
> operations used in attacks; RAM access is just one example. If the same
> operations are not bottlenecks in attacks against AES-m or SHA-n then
> this can reverse comparisons to AES-m or SHA-n respectively.

No. You are arguing that because NIST has not specified a single metric no evaluation in any metric can be valid. The Call for Proposals actually says (NIST's emphasis):

"In order for a cryptosystem to satisfy one of the above security requirements, any attack must require computational resources comparable to or greater than the stated threshold, with respect to **all** metrics that NIST deems to be potentially relevant to practical security."

NIST gives circuit size as an example of a metric and says that AES-192 key recovery is equivalent to 2^{207} classical gates in this metric. The finite regime analysis estimates that breaking mceliece-4608-096 with Stern takes $2^{194.4}$ classical gates including a logarithmic overhead for memory accesses to 6 GiB of RAM. Whether or not NIST deems this to be potentially relevant to practical security is up to them.

>> The Round 3 submission does not include any concrete analysis of the
>> security provided by the proposed parameter sets.

>
> Not true. For example, the stated expectation to be "more expensive to
> break than AES-256" is concrete, and the rationale is stated. A more
> detailed analysis depends on the cost metric.

We have different ideas of what constitutes concrete analysis. Let's take a look at what section 8.2 of your submission actually says. Don't worry. It's not that long.

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

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

However:

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

That's it. No mention of the other parameter sets. No attempt to quantify the impact of memory accesses. Just a vague statement that something that is "considerably below 2^{256} " bit operations will be fine because memory.

- >> There is a lot of hype about the asymptotic result from Canto Torres
- >> and Sendrier
- >
- > The graph in <https://video.cr.yp.to/2020/0813/video.html> shows that
- > asymptotic lattice security levels against attacks known 10 years ago
- > were 42% higher than lattice security levels against attacks known today
- > (for otherwise unbroken lattice systems), while for McEliece's system
- > the change is 0%. Any thorough evaluation of security risks will take
- > these facts into account, along with other risk indicators.
- >
- >> but this is ultimately irrelevant for any finite choice of parameters.

This is the irrelevant hype I mean. You are ignoring asymptotic improvements to half-distance decoding because McEliece uses errors with sublinear weight yet compare it with asymptotic improvements to the exact short vector problem even though lattice schemes depend on different variants of the problem.

- > The submission already says, at the top of the "Information-set
- > decoding, concretely" section:
- >
- > We emphasize that $o(1)$ does not mean 0: it means something that
- > converges to 0 as $n \rightarrow \infty$. More detailed attack-cost evaluation is
- > therefore required for any particular parameters.
- >
- > The section goes on to summarize what the literature says about concrete
- > cost analyses, and ends with the AES-256 comparison reviewed above.

The rest of section 8.2 is quoted above in its entirety. The literature says much more about concrete cost analyses than this "summary".

- >> The submission argues that any reduction in classical security from
- >> improved ISD attacks will be offset by the increased memory
- >> requirements. While this may be true for some ISD variants such as
- >> BJMM, they provide no analysis to back this up.
- >
- > Again, here's the critical quote from the submission:

>
 > A closer look shows that the attack in [11] is bottlenecked by random
 > access to a huge array (much larger than the public key being
 > attacked), and that subsequent ISD variants use even more memory. The
 > same amount of hardware allows much more parallelism in attacking,
 > e.g., AES-256.
 >
 > The exact impact depends on the cost metric.
 >
 >> It also ignores other
 >> ISD variants such as Stern that need significantly less memory.
 >
 > No, the submission doesn't ignore this. The "attack in [11]" (see quote
 > above) is a Stern-type attack. The submission's comment that "subsequent
 > ISD variants use even more memory" (see quote above) is pretty much the
 > same as your comment that Stern-type attacks "need significantly less
 > memory".
 >
 >> - The mceliece-4608-096 parameters are about 13 bits below Category 3 with
 >> an attack that needs slightly over 6 GiB of memory.
 >
 > Evaluating such a claim would require a clear, stable definition of
 > "Category 3". If NIST settles on a definition to be used in NISTPQC then
 > submitters will be able to make "category" assignments accordingly.
 >
 > See above regarding the actual costs of accessing 6GB of RAM.

No. The definition of category 3 is stable. Your argument is that NIST has not specified a single metric. Evaluating the claim only needs a metric that NIST deems practically relevant.

For example, the cost of breaking mceliece-4608-096 using Stern with no, logarithmic, cube-root, or square-root memory overhead is:

Parameter Set	Free	Log	Cbrt	Sqrt
mceliece-4608-096:	189.2	194.4	200.3	204.6

Ray Perlner's cost model slides include almost all of these metrics. None of them give an estimate greater than 2^{207} classical gates.

>> If Classic McEliece is to be standardized as a conservative option
 >> then the parameter sets that are standardized with it should also be
 >> chosen conservatively. The NTS-KEM parameters were. Three out of the
 >> five Classic McEliece parameters were not.
 >
 > Please clarify. Are you saying anything here beyond the 2^{194} vs. 2^{207} ,
 > 2^{270} vs. 2^{272} , and 2^{271} vs. 2^{272} discussed above?
 >
 > If you're concerned about 2^{270} RAM operations vs. 2^{272} bit operations

No, no, no. $2^{189.2}$ classical gates does not mean $2^{189.2}$ random memory accesses. Not all gates involve a memory access. Not all memory accesses are to high-latency memory. Not all high-latency memory accesses are random. My point is that you can't just say that everything will be fine because memory. You actually need to do the analysis.

> There's also a tremendous amount to say about the dangers of unnecessary
 > complexity. One aspect of this is specifying more parameter sets than
 > necessary. Of course, NIST's request for lower-security parameter sets
 > was an offer that Classic McEliece couldn't refuse, but we continue to
 > recommend the 2^{256} parameter sets.

I assume that if Classic McEliece is chosen for standardization you will be withdrawing the parameter sets that you don't recommend for use.

- > One can easily write down even larger parameter sets. Also, the ISD
- > attack analyses in the literature are precise and straightforward, and
- > it should be easy to evaluate the security of any desired parameters in
- > any well-defined metric. However, the right order of events is, first,
- > for NIST to fully define the cost metric to be used for "categories",
- > and, second, for all submission teams to evaluate costs in this metric.

No. The right order of events is, first, submit parameter sets with security estimates in a metric that you think is practically relevant and, second, let everyone else spend three rounds of the NIST process evaluating those claims. That's what most of other submissions have done.

> ---Dan (on behalf of the Classic McEliece team)

Kirk

--

You received this message because you are subscribed to the Google Groups "pqc-forum" group.

To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.

To view this discussion on the web visit <https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/trinity-2fd8b511-ef73-4d23-80f9-f60c63ff91a9-1606179667109%403c-app-mailcom-lxa05>.

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Monday, December 7, 2020 5:21 AM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

Speaking for myself in this message. As far as I can tell, the questions specific to Classic McEliece have already been answered. There are some broader NISTPQC process points here for NIST to address (as shown by the inconsistent handling of different cryptosystems), and there are some factual errors that I'll correct for the record.

As a starting point, there appears to be a misconception that NIST has already defined metrics to be used for the NISTPQC "categories": namely, (non-quantum and quantum) "gates" metrics, ignoring the costs of RAM.

If this were true then I would agree that all submissions have to report security in these metrics. But it isn't true. NIST has never issued any such definition. There are many different definitions of (non-quantum and quantum) "gates" in the literature, with many variations (e.g., sometimes RAM gates, sometimes not), often producing radically different cost numbers; NIST has never said which "gates" we're supposed to use.

The hard way to check this is to dig through the complete NISTPQC archives and look at what NIST has written. The easy way, for people who know some quantum algorithms, is to consider the following two facts:

- * The NISTPQC call for proposals claims that known SHA3-256 collision attacks need 2^{146} "gates", with no quantum speedups.
- * It's easy to find SHA3-256 collisions in only about 2^{85} oracle queries, which is in the ballpark of 2^{100} quantum "gates" with the "gates" defined in, e.g., Ambainis's famous distinctness paper and used in many other quantum-walk papers.

To see the 2^{85} , apply Ambainis's algorithm to 128-bit strings after a random prefix. Or, simpler and presumably faster, apply the BHT (Brassard--Hoyer--Tapp) algorithm to 256-bit strings. This means a Grover search through 170-bit inputs for hashes appearing in a table of 2^{86} hashes of other inputs.

To see the 2^{100} , start from the 2^{85} and look at the cost of SHA3-256 evaluation. Maybe ends up slightly over 100, but maybe not: one can handle a large oracle cost by adjusting 170 down and 86 up. The exact number doesn't matter for anything I'll say here.

Given the gap between 2^{100} and 2^{146} , why does the call for proposals ignore BHT, Ambainis, etc.? Anyone who thinks that NIST has defined "gates", and that NIST's "gates" allow free memory, will have trouble explaining this.

It's clear how many `_operations_` are used by (e.g.) BHT, but turning the operation count into a cost number to compare to other attacks requires defining a cost metric. Novices might think that the choice of cost metric is a matter of "low-order details", but the gap between 2^{146} and

2^{100} is bigger than most of the attack advances we've seen in NISTPQC, including advances that NIST has used to publicly justify its decisions.

It's almost as big as the gap between Kyber-768 and what the New Hope paper calls "JarJar". It's bigger than the pre-quantum gap between RSA-2048 and RSA-1024.

Once upon a time, NIST cited <https://cr.yp.to/papers.html#collisioncost> as justification for ignoring BHT. But that paper doesn't use a "gates" metric, so this doesn't salvage the idea that NIST has defined a "gates" metric. That paper also takes RAM costs into account in exponents, which is the opposite direction from treating RAM as free.

Ray Perlner claimed in August 2020 that quantum RAM gates are "much less realistic" than non-quantum RAM gates. This sounds very wrong to me--- cost 1 for an unlimited-size array lookup is not reality, and will never be reality, whether quantum or not---but in any case it doesn't resolve the more basic problem of not having a definition.

Where's the definition that says exactly which gates are included in NIST's "gates"? If the situation is supposed to be that NIST's "quantum gates" exclude RAM gates while, bizarrely, NIST's "classical gates" include RAM gates, then we should all be able to confidently see this by looking at NIST's definition. But there is no definition!

The fact that NIST has never defined a cost metric to be used in NISTPQC is a disincentive for detailed cost evaluations. Consider the quandary facing Caroline the Cryptanalyst:

- * If Caroline does the work to analyze system S in a realistic cost metric (e.g., 1981 Brent-Kung with appropriate parameter choices, or a quantum variant of the same thing) then people who don't like S will complain that Caroline is cheating and overestimating security through the choice of metric.
- * If Caroline instead picks a simpler cost metric that ignores the costs of RAM etc. then maybe Caroline will have less work, but people who like S will (correctly) complain that Caroline is underestimating security.

Caroline would be able to justify an analysis by pointing to NIST's request for a particular metric---but NIST hasn't defined a metric to use. Is it a surprise, then, that detailed cost evaluations in NISTPQC have been so rare? Is it a surprise that, to the extent they exist, they don't agree on which cost metrics to use?

Simply counting the higher-level operations that naturally appear in attacks doesn't run into these problems, but how are we supposed to compare attacks against different systems? Or attacks against the same system, when the underlying operations are different? (For example, within published lattice attacks, quantum enumeration looks more efficient than sieving in some cost metrics, although in this case there are also many unresolved questions about the higher-level operation counts.)

Even worse, NIST has managed to make many casual observers (not the careful observers such as Caroline!) believe that NIST has defined metrics to be used in NISTPQC. The unfortunate reality is that there's a definitional void filled with unjustified and conflicting assumptions.

For example:

- * Some people think that NIST "gates" assign low costs to memory (so BHT uses 2^{100} NIST "gates" and the call for proposals is wrong).

* Some people think that NIST "gates" assign high costs to memory (which is why BHT doesn't beat 2^{146} NIST "gates").

These incompatible beliefs end up being applied inconsistently to different candidates, as we're seeing in discussions right now about multiple finalists. The same lack of definitions means that nobody can figure out the "category" boundaries, even for the small number of submissions where operation counts in attacks are thoroughly understood. This is a cross-cutting failure in the NISTPQC evaluation procedures.

NIST has the unique power to fix this mess by issuing a complete definition of a metric (or multiple metrics if that's really needed) for all NISTPQC submissions to use. This means not just waving around ambiguous words such as "gates", but clearly specifying which gates. We can use, say, vOW and BHT cost analyses as test cases for the clarity of the definition, and then we can all go ahead with evaluating all submissions in the same metric.

Will this be the "best" metric? Unlikely. Maybe it will be so poorly chosen as to noticeably damage parameter selection. But we're starting from major ongoing damage to the evaluation process as a direct result of NIST's "category" boundaries not having clear definitions.

>> We agree that the second column is sometimes less than the fourth column.
>> However, the second column ignores the huge overheads measured by
>> the third column.
> No. The second column includes a logarithmic memory overhead.

Counting cost 1 for each bit of an index i is simply reading i ; it is not covering the far larger costs of retrieving $x[i]$. The attack is, as the submission states, "bottlenecked by random access to a huge array"; it is not bottlenecked by inspecting the indices used for this access.

The picture to have in mind here is a RAM request moving a huge distance down wires (or fiber or any other physical communication technology), compared to inspecting bits locally.

In algorithm analysis, pointing to overhead beyond cost analysis X means pointing to a cost that isn't included in X . A reader who redefines "overhead" to cover something that is included in X is not correctly presenting the original statement. This redefinition is also missing the point in this case: what we care about are "the huge overheads" of continual RAM access, since those are the bottlenecks in the attack.

>> As stated in the submission:
>> A closer look shows that the attack in [11] is bottlenecked by random
>> access to a huge array (much larger than the public key being
>> attacked), and that subsequent ISD variants use even more memory. The
>> same amount of hardware allows much more parallelism in attacking,
>> e.g., AES-256.
> This is the second time you've used the word "huge". Let's give it
> some perspective. $2^{35.66}$ bits is less RAM than the laptop I'm using
> to write this.

It's also 10000 times larger than the public key. This is "a huge array (much larger than the public key being attacked)", as the submission says. The same circuit size "allows much more parallelism in attacking, e.g., AES-256", as the submission says. See

<https://www.sites.google.com/site/michaeljameswiener/dessearch.pdf>

for an example (predating AES) of the efficiency of special-purpose hardware for cipher attacks.

The laptop "perspective" wildly overestimates the costs of attacks, as already established in the literature on attack hardware. The amount of overestimation varies from one cryptosystem to another. This makes the laptop "perspective" useless for serious security evaluation---so why are we seeing this "perspective" appearing in NISTPQC discussions? Answer: it's yet another random attempt to fill the void left by NIST's failure to define the NISTPQC security requirements. All sorts of people interested in NISTPQC are being put in the position of figuring out for themselves what the NISTPQC security levels mean, because the only organization in a position to do the job centrally hasn't done it.

>> The submission goes on to state expectations regarding (e.g.)
>> 6960119 being more expensive to break than AES-256. The numbers you
>> quote (e.g.,
>> $2^{271.18}$ operations on $2^{47.58}$ bits of RAM) are consistent with
>> these expectations.
> Your submission and this reply focus almost exclusively on the
> mceliece-6960-119 parameter set. What about the other four?

The submission takes this example (using the words "as an example") to explain the most important effects separating cost metrics in this case:

namely, the state-of-the-art attacks are "bottlenecked by random access to a huge array (much larger than the public key being attacked)", and quantum attacks suffer "much more overhead in the inner loop". Repeating the same points for each parameter set, *mutatis mutandis*, would be unilluminating.

Because of the unlimited variation in cost metrics, any parameter set can be ranked anywhere from "overkill" to "too small" depending on which cost metric is selected, as the submission spells out for this example. The incorrect idea that this is specific to Classic McEliece was already answered: "Note that one can make any cryptosystem sound much easier to break by choosing a cost metric that assigns unrealistically low cost to operations used in attacks; RAM access is just one example. If the same operations are not bottlenecks in attacks against AES-m or SHA-n then this can reverse comparisons to AES-m or SHA-n respectively."

I'm very much in favor of building tables of publicly verified cost evaluations for attacks across all parameter sets in all submissions, with AES-128, SHA-256, etc. as baselines---including how the evaluations are changing over time, so that we can recognize attack improvements, evaluate security stability, and evaluate maturity of analysis. However, this process doesn't work until NIST defines a cost metric for all NISTPQC submissions to use. Allowing cost metrics to be selected ad-hoc for particular submissions is error-prone and subject to abuse.

>> The literature shows that optimized hardware for AES attacks has
>> similar efficiency to multipliers. The multipliers in an Intel CPU
>> core carry out millions of bit operations in the same time that it
>> takes the core to retrieve data from (say) 6GB of RAM, never mind
>> RAM contention across cores. GPUs and TPUs show the costs of RAM
>> even more clearly. On a larger scale, decades of supercomputing
>> literature consistently fit a square-root scaling of RAM costs, and
>> science-fiction 3D computers (which seem much harder to build than
>> quantum computers) would still have cube-root costs.
> No. It's not as simple as applying a square-root overhead for memory
> across the board. For an interesting (but not very rigorous) example
> see http://www.ilikebigbits.com/2014_04_21_myth_of_ram_1.html. The
> blog argues for a square-root memory cost. If you look at the graphs
> this is a poor fit in the 10 MiB to 6 GiB range.

Across seven orders of magnitude of array sizes, the graphs in that blog post are always within one order of magnitude of the square-root prediction (and people who understand the microarchitecture already know where the deviations

come from). If you look at the supercomputing literature you'll see the same basic scaling across further orders of magnitude. (See, e.g., <https://cr.yip.to/papers.html#nonuniform>, Appendix Q.6, comparing the records from <https://sortbenchmark.org> at two different sizes for the number of 100-byte items sorted per joule.)

There's also a clear physical mechanism producing the square-root overhead: namely, moving a bit across distance \sqrt{N} consists of \sqrt{N} motions through distance 1. Each motion through distance 1 occupies a constant amount of hardware for a constant amount of time, the same way that a bit operation occupies a constant amount of hardware for a constant amount of time. Overall the motion through distance \sqrt{N} is occupying $\Theta(\sqrt{N})$ times more hardware (per unit time) than a bit operation is, where Θ covers the ratios between the relevant constants.

This basic throughput barrier is exactly what one sees in the 1981 Brent--Kung theorem, which obtains the same square root for a broad class of plausible machine models:

<https://maths-people.anu.edu.au/~brent/pd/rpb055.pdf>

Fiber doesn't magically avoid this throughput barrier: the hardware is still filled up by the data being communicated. Free-space lasers might seem to avoid the cost of equipment in the middle, but they spread out linearly with distance and don't end up improving scalability.

The "holographic principle" in physics says that the information in a volume of space is actually two-dimensional. (This won't surprise physics students who have already practiced converting three-dimensional to two-dimensional integrals.) The numerical limits don't say that existing technology is close to optimal, but they do say that anyone claiming better than square-root scaling is wrong for large enough sizes and thus has a ton of explaining to do regarding cryptographic sizes.

An easy mistake to make here is to cherry-pick some news story about a technology improvement at some medium size, and compare it to unimproved technology at a small size (e.g., the laptop "perspective"), while failing to ask whether technology improvements are also following the expected square-root curve down to small sizes.

> Not all memory accesses are the same. The memory overhead depends on
> the number of random accesses to high-latency memory.

I agree that memory accesses differ in general. However, the attacks we're talking about here are bottlenecked by random access to a huge array ("much larger than the public key being attacked").

Note that one can replace parallel random access with, e.g., bitonic sorting, at which point the word "random" might seem misleading since at a low level all accesses are through a predefined network. What really matters, as the Brent--Kung proof clearly shows, is simply how much data is moving across long distances.

> One cost model in Ray Perlner's slides assumes that up to 2^{50} bits of
> memory is local with unit cost memory accesses.

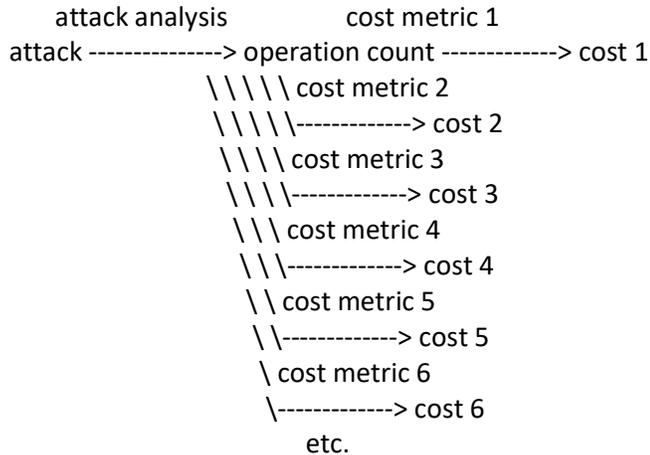
This was already answered: "The multipliers in an Intel CPU core carry out millions of bit operations in the same time that it takes the core to retrieve data from (say) 6GB of RAM, never mind RAM contention across cores."

> Can you point to the paper cited by your submission that gives numbers
> for the mceliece-4608-096 parameter set which clearly justify its
> assignment as category 3?

NIST has failed to define the metrics to be used, so there's no way for anyone to be sure what's included in "category 3".

This isn't specific to Classic McEliece. Every submission is in the same position of uncertainty regarding what NIST's security requirements mean. Beyond actual security risks, this creates an artificial risk of being criticized for allegedly not meeting someone's interpretation of NIST's security requirements. There is no way to avoid this risk: any "category" assignment of any parameter set in any submission can be targeted by a metric-selection attack. The way that these criticisms are allocated to submissions has nothing to do with security; it is a political game of ad-hoc marketing of cost metrics.

Regarding citations, the Classic McEliece submission cites the original attack papers. Those papers, in turn, convincingly count the operations that they use. Each operation count corresponds to many different cost numbers for many different choices of cost metric:



This correspondence is conceptually straightforward, but which cost metric are submitters supposed to use for NISTPQC? Nobody knows.

There are other submissions where the operations in attacks are much harder to count, but if we optimistically imagine this problem being fixed then we're still faced with the same basic cost-metric question for all submissions, as noted above.

Submitters could start from Perlner's recent "preliminary" slides, which run through various possible features of metrics. How many metrics are clearly defined in those slides, with a NIST commitment to treat attacks in those metrics as being relevant? Zero. So what exactly are submitters supposed to do? For each feature listed by Perlner, try to guess the sanest-sounding definition of a specific metric that seems consistent with that feature, and then work through a pile of cost evaluations in all of these metrics, while being pretty sure that NIST will disregard most, perhaps all, of the resulting numbers? See how crazy this is?

- > You are arguing that because NIST has not specified a single metric no
- > evaluation in any metric can be valid. The Call for Proposals actually
- > says (NIST's emphasis):
- > "In order for a cryptosystem to satisfy one of the above security
- > requirements, any attack must require computational resources
- > comparable to or greater than the stated threshold, with respect
- > to **all** metrics that NIST deems to be potentially relevant to
- > practical security."
- > NIST gives circuit size as an example of a metric

No.

As a baseline, I think everyone agrees that the structure specified in the call for proposals is as follows:

- * There's a set M of metrics to be used for NISTPQC (the "metrics that NIST deems to be potentially relevant to practical security").
- * Cryptosystem X is eliminated from category 1 (and thus from NISTPQC) if there's even one metric in M saying that attacking X costs less than AES-128 key search.
- * Cryptosystem X is eliminated from category 2 if one metric in M says that attacking X costs less than SHA-256 collision search.
- * Similarly for other categories.

But what's M? Can we even point to one clear example of something in M? Unfortunately not. If you look for a NIST document that clearly defines a metric and specifies that the metric is in this set M, you won't find any such document!

For example, it's not true that there's some clearly defined "circuit size" metric specified to be in M, and it's not true that there's some clearly defined "gates" metric specified to be in M. A clear definition of NIST "gates" would let the community understand and verify the claim in the call for proposals that known SHA3-256 attacks, such as BHT, don't beat 2^{146} NIST "gates". Right now this is unverifiable, unfalsifiable, mathematically meaningless, since there's no definition.

> and says that AES-192 key
> recovery is equivalent to 2^{207} classical gates in this metric

Unfortunately NIST never defined "this metric". See above.

> That's it. No mention of the other parameter sets. No attempt to
> quantify the impact of memory accesses.

The literature already pinpoints the number of memory accesses for arbitrary input sizes. As soon as NIST defines a cost metric, an analysis in that metric will be straightforward.

> Just a vague statement that something that is "considerably below
> 2^{256} " bit operations will be fine because memory.

The actual "will be fine" statement is "more expensive to break than AES-256". This is not vague. It has the cost metric as a parameter, and of course one can define cost metrics that reverse the comparison by ignoring important components of costs, but, again, having to allow every possible metric would undermine every category assignment in every submission.

Quantifying "considerably below 2^{256} " depends on the cost metric. The minimum "time" in the literature is achieved by algorithms using absurd amounts of memory, and those algorithms are optimal in metrics that allow $x[i]$ with cost 1, but gradually increasing the RAM cost in the metrics (compared to other costs) shifts the optimum to different algorithms.

> This is the irrelevant hype I mean. You are ignoring asymptotic
> improvements to half-distance decoding because McEliece uses errors
> with sublinear weight

Carefully quantifying the application of attack ideas to a cryptosystem, and correctly concluding that the asymptotic change in security levels is exactly 0%, is not "ignoring" anything.

The 0% change has always been clearly and correctly labeled as being asymptotic. This means that there's <10% change once the security level is large enough, <1% change once the security level is large enough, etc. Regarding the question of what "large enough" means, the submission cites the relevant attack papers and emphasizes the importance of concrete cost analysis.

Seeing that lattices have a worse picture here---e.g., the asymptotic lattice security levels were 42% higher just ten years ago---is a risk indicator. Nobody says it's the only risk indicator! Classic McEliece has established a strong track record of observing many other risk indicators and acting accordingly.

This is important. History suggests that identifying and monitoring risk indicators is one of the most effective ways that system designers can avoid disasters. There's ample evidence for this in the general risk-management literature, and there's quite a bit of evidence in cryptography.

Consider, for example, pre-quantum discrete-logarithm-based cryptography. There was already a long history by 2005 of

- * multiplicative groups suffering percentage losses (and sometimes larger losses) of asymptotic security levels, including losses avoided by elliptic curves, and
- * fields with extra automorphisms suffering percentage losses of asymptotic security levels, including losses avoided by prime fields.

Public recommendations from cryptographers monitoring the literature concluded that multiplicative groups were higher risk than elliptic curves, and that fields with extra automorphisms were higher risk than prime fields. (Of course, people desperate to paint the opposite picture could always come up with reasons: the RSA company was still claiming that elliptic curves hadn't been studied, for example, and there was also an interesting story regarding "medium" fields.) Unsurprisingly, we saw big further security losses after that for multiplicative groups (for some highlights see <https://blog.cr.yp.to/20161030-pqnist.html>), especially in the case of fields with extra automorphisms.

What does this tell us about lattices vs. McEliece? Should we be on the alert for percentage losses of asymptotic security levels (many losses for lattices, nothing for McEliece)? Larger losses (yes for lattices, no for McEliece)? Unnecessary automorphisms? All of the above?

If there's a rational argument that a particular risk indicator isn't helpful, the argument should be published for further analysis. Posting one message after another labeling a risk indicator as "irrelevant hype" doesn't move the analysis forward, and seems hard to reconcile with the thoroughly documented discrete-log history.

- > yet compare it with asymptotic improvements to the exact short vector
- > problem even though lattice schemes depend on different variants of the problem.

Actually, the SVP security levels being asymptotically 42% higher in 2010 than today translates directly into the security levels of (otherwise unbroken) lattice systems being asymptotically 42% higher in 2010 than today. Basically, the lattice attacks are bottlenecked by BKZ, and BKZ is bottlenecked by a relatively short series of expensive shortest-vector searches in a lower dimension (beta, the "block size"), so if the SVP solution loses P% of its asymptotic bits of security then the whole attack loses P% of its asymptotic bits of security.

(BKZ is one of many algorithms, and ongoing research indicates that SVP isn't really the right interface, but these issues don't seem to affect the exponent. A bigger issue is hybrid attacks: my message on that topic in July outlines a way to remove another percentage from the security level for any system with errors in, say, $\{-3, \dots, 3\}$, so the loss for lattice security compared to 2010 will be even larger than the SVP loss. Some hybrid-attack details still need verification, and the percentage hasn't been calculated yet.)

For comparison, consider <https://eprint.iacr.org/2017/1139.pdf> reporting exponent 0.0465 for half-distance decoding, where Prange's algorithm from 51 years earlier was exponent 0.0576; see the paper for a survey of intermediate results. This asymptotic 24% change corresponds to an asymptotic 0% change in the asymptotic security level of McEliece's system. The situation for McEliece's system has always been that the asymptotic bits of security come primarily from a gigantic combinatorial search through information sets.

To summarize: The graph in <https://video.cr.yp.to/2020/0813/video.html> correctly compares asymptotic attack costs, as the label indicates. In particular, the asymptotic security level of otherwise unbroken lattice cryptosystems has dropped as dramatically as shown in the graph. (Also, recent work, once quantified, should produce further lattice losses.)

> > The section goes on to summarize what the literature says about
> > concrete cost analyses, and ends with the AES-256 comparison reviewed above.
> The rest of section 8.2 is quoted above in its entirety. The
> literature says much more about concrete cost analyses than this "summary".

"Summary, noun: a brief statement or account of the main points of something." I agree that the literature says much more than the summary does.

> > Evaluating such a claim would require a clear, stable definition of
> > "Category 3". If NIST settles on a definition to be used in NISTPQC
> > then submitters will be able to make "category" assignments accordingly.
> > See above regarding the actual costs of accessing 6GB of RAM.
> No. The definition of category 3 is stable.

No, it isn't. NIST has never issued a clear, stable definition of "Category 3". NIST has given a pseudo-definition--- something that looks at first glance like a definition but that turns out to rely on undefined metrics. How are submitters supposed to evaluate costs in cost metrics that haven't been defined? See above.

> $2^{189.2}$ classical gates does not mean $2^{189.2}$ random memory accesses.
> Not all gates involve a memory access. Not all memory accesses are to
> high-latency memory. Not all high-latency memory accesses are random.

It's of course true across the field of cryptanalysis that attacks vary in how often they're randomly accessing memory. However, all of the attacks we're discussing here are bottlenecked by random access.

> My point is that you can't just say that everything will be fine
> because memory. You actually need to do the analysis.

The attack paper [11] cited in exactly this section of the submission already includes a detailed cost analysis of Stern's attack with many further optimizations. The inner loop consists of picking up a number that looks random, xor'ing it into an index i , randomly accessing $x[i]$, and going back to the top of the loop.

As the submission says, this attack is "bottlenecked by random access". Newer attacks are similarly memory-bound, using even more memory.

The operation counts are clear (for this submission; again, there are other submissions where attacks are much harder to analyze). The notion that there's some missing analysis of how many RAM accesses there are is simply not true. What's missing is a definition from NIST of how NISTPQC assigns costs to $x[i]$ and other operations.

>> There's also a tremendous amount to say about the dangers of
>> unnecessary complexity. One aspect of this is specifying more
>> parameter sets than necessary. Of course, NIST's request for
>> lower-security parameter sets was an offer that Classic McEliece
>> couldn't refuse, but we continue to recommend the 2^{256} parameter sets.
> I assume that if Classic McEliece is chosen for standardization you
> will be withdrawing the parameter sets that you don't recommend for use.

Procedurally, NIST is in charge, and has always specifically reserved the right to make its own parameter-set decisions after it consults with the submitters and the community. See the call for proposals. Obviously submission teams and other interested parties will provide feedback to NIST regarding the wisdom of their decisions.

>> One can easily write down even larger parameter sets. Also, the ISD
>> attack analyses in the literature are precise and straightforward,
>> and it should be easy to evaluate the security of any desired
>> parameters in any well-defined metric. However, the right order of
>> events is, first, for NIST to fully define the cost metric to be
>> used for "categories", and, second, for all submission teams to evaluate costs in this metric.
> No. The right order of events is, first, submit parameter sets with
> security estimates in a metric that you think is practically relevant
> and, second, let everyone else spend three rounds of the NIST process
> evaluating those claims. That's what most of other submissions have done.

No, it isn't. Let's take the Kyber submission as a case study of what other submissions have in fact done.

The round-3 Kyber submission has an analysis concluding that attacks against round-3 Kyber-512 are probably--- accounting for the "known unknowns"---somewhere between 2^{135} and 2^{167} "gates", a remarkably wide range for a cryptosystem that claims its security is well understood.

The submission hopes that the "known unknowns" will be pinned down, and that the final answer will be above 2^{143} . The submission also claims to use a "conservative lower bound" on attack costs. Can NIST see whether this "bound" is above or below 2^{143} ? Can anyone else see this? Nope.

Could someone have been reviewing the analysis since the first round of the NIST process? No. The parameter set has changed. The analysis has changed. The problem that cryptanalysts are being asked to study has changed! Kyber-512 used to claim that its security was based on the MLWE problem, but, as far as I can tell, this has been dropped in round 3. Concretely, my understanding is that if someone claims an attack against round-3 Kyber-512 by showing that MLWE for that size is easier to break than AES-128, the official response will be that the round-3 submission does not claim MLWE attacks are so hard---its security levels are claimed only for direct IND-CPA/IND-CCA2 attacks. (I've asked for confirmation of this in a separate thread on Kyber.)

So much for spending "three rounds of the NIST process evaluating those claims", or, in the words of the call for proposals, "maturity of analysis". Now let's move on to the "metric" question.

Let's say Caroline the Cryptanalyst is looking at round-3 Kyber-512. What would be required, structurally, for Caroline to show that round-3 Kyber-512 is broken? Let's assume for simplicity that Kyber "gates" are clearly defined and that AES-128 key search needs 2^{143} Kyber "gates".

Here are two possibilities for Caroline:

- * Strategy 1: Show that the best attacks considered in the submission actually use fewer than 2^{143} "gates". This is compatible with the $2^{135} \dots 2^{167}$ range claimed in the submission.
- * Strategy 2: Show that attacks not considered in the submission, such as hybrid attacks, use fewer than 2^{143} "gates".

These would both be breaks, since they'd violate the minimum for category 1, right? Nope. The submission protects itself against this by leaving wiggle room in the cost metric. Here's the critical quote:

We do not think that [2^{135}] would be catastrophic, in particular given the massive memory requirements ...

The submission is saying that 2^{135} "gates" wouldn't actually be a problem (assuming the attacks use tons of memory, which most, although not all, lattice attacks do), since of course memory costs much more than a "gate" count indicates.

Should Caroline expect NIST to downplay the memory requirements and accept the attack as a break? Why?

NIST has repeatedly asked for "confidence" that the NISTPQC security requirements are reached. I challenged the security of round-2 Kyber-512, pointing out errors in the security analysis and pointing out reasons to believe that known attacks don't use more "gates" than AES-128. Did NIST claim "confidence" that the attacks actually use more "gates" than AES-128? No. Did NIST kick out that parameter set? No. Instead NIST initiated a "preliminary", open-ended project to start incorporating overheads into cost metrics.

Does this sound like NIST automatically kicking out any parameter set when there's an attack below 2^{143} "gates", with free RAM "gates"? No.

It sounds like NIST shifting the boundaries of "category 1" as far as necessary to declare that Kyber-512 isn't broken, by playing games with the cost metric. Maybe NIST has another reason for not specifying a metric, but in any case the moving target deters cryptanalysis.

Part of recognizing security risks is recognizing the procedures that are actually being used for designing, evaluating, and standardizing cryptography. Cryptanalytic work is a critical part of this, and NISTPQC is raising far more security questions than cryptanalysts can answer in the time available. It's not as if the attack picture was stable in 2020, so why should we expect it to suddenly be stable in 2021, or in 2025? This denial-of-service attack against cryptanalysts starts from the inherent cryptanalytic complexity of post-quantum cryptography, but the attack is exacerbated by NIST's continued failure to clarify its pseudo-definition of the minimum allowed NISTPQC security level.

---Dan

From: Kirk Fleming <kpfleming@mail.com>
Sent: Friday, December 11, 2020 8:29 PM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Dan Bernstein (speaking for himself) wrote:

> As far as I can tell, the questions specific to Classic McEliece
> have already been answered.

In the formal sense that you responded, I guess. It's like asking the time and being told that it depends on the observer's frame of reference. Thanks Einstein. It is an answer but I still don't know if I'm late for class.

Kirk Fleming wrote:

>>>> I am filing this comment to request that the Classic McEliece
>>>> submitters justify the claimed security categories for their
>>>> parameters.

Dan Bernstein (on behalf of the Classic McEliece team) wrote:

>>> Can you please clarify how you think that the numbers already
>>> in the literature don't already justify the assignments in the
>>> submission?

Kirk Fleming wrote:

>> Let's try an easier question. Can you point to the paper cited
>> by your submission that gives numbers for the mceliece-4608-096
>> parameter set which clearly justify its assignment as category 3?

Dan Bernstein (speaking for himself) wrote:

> NIST has failed to define the metrics to be used, so there's no
> way for anyone to be sure what's included in "category 3".

To be absolutely clear, you are stating that the numbers in the literature already justify the assignments in the submission but that you can't point to any specific numbers because there is no way for anyone to be sure what's included in each category.

Okaaaay.

Let's try an even easier question. Why mceliece-4608-96?

The mceliece-6960-119 parameter set was taken from "Attacking and defending the McEliece cryptosystem". You could have chosen the mceliece-4624-95 parameter set from the same paper. Instead you chose a parameter set which had "optimal security within 2^{19} bytes if n and t are required to be multiples of 32". (mceliece-4624-95 was chosen to be optimal without the restrictions on n and t)

mceliece-5856-64 and mceliece-4160-128 are also within 2^{19} bytes. Both of these have smaller public keys than mceliece-4608-96 and mceliece-5856-64 has smaller ciphertexts. Please explain the metric you used to decide which of the three parameter sets had "optimal security".

Kirk

From: Kirk Fleming <kpfleming@mail.com>
Sent: Wednesday, December 30, 2020 7:50 PM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

I promise this is my last message on the topic. Dan has stopped responding but I wanted to make one final point.

Dan Bernstein (on behalf of the Classic McEliece team) asked:
> Can you please clarify how you think that the numbers already
> in the literature don't already justify the assignments in the
> submission?

The only concrete security estimate given in the entire Classic McEliece submission is:

"[Bernstein, Lange and Peters. 2008] reported that its attack uses $2^{266.94}$ bit operations to break the (13,6960,119) parameter set."

This is not true. Bernstein, Lange and Peters actually say:

"For keys limited to 2^{16} , 2^{17} , 2^{18} , 2^{19} , 2^{20} bytes, we propose Goppa codes of lengths 1744, 2480, 3408, 4624, 6960 and degrees 35, 45, 67, 95, 119 respectively, with 36, 46, 68, 97, 121 errors added by the sender. These codes achieve security levels 84.88, 107.41, 147.94, 191.18, 266.94 against our attack."

The $2^{266.94}$ estimate is not for mceliece-6960-119. It's for a different parameter set that uses list decoding to increase the number of errors that can be corrected.

The submission's lack of any serious security analysis for the five parameter sets being proposed should be cause for concern. The team's refusal to provide any justification for their claimed security categories when challenged should be disqualifying.

Kirk

From: Kirk Fleming <kpfleming@mail.com>
Sent: Monday, January 4, 2021 8:57 PM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

The Classic McEliece submission explicitly allows a change to decapsulation which breaks the IND-CCA2 security proof and leaves the scheme vulnerable to decoding failure attacks.

The decapsulation algorithm described in Section 2.3.3 is:

1. Split the ciphertext C as (C_0, C_1) .
2. Set $b:=1$.
3. Extract s and Gamma' from the private key.
4. Compute $e := \text{Decode}(C_0, \text{Gamma}')$.
If $e = \text{Fail}$ set $e:=s$ and $b:=0$.
5. Compute $C'1 := H(2, e)$.
6. If $C'1 \neq C_1$ set $e:=s$ and $b:=0$.
7. Compute $K := H(b, e, C)$.
8. Output session key K .

However, the discussion in the paragraphs following the algorithm description suggests a change to Step 4 for ease of implementation.

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

This is not true. Using a known fixed string trivially breaks the scheme.

To see why, consider decapsulation where Step 4 is changed to:

- 4'. Compute $e := \text{Decode}(C_0, \text{Gamma}')$.
If $e = \text{Fail}$ set $e:=0$ and $b:=0$.

Let $C := (C_0, C_1)$ be a ciphertext chosen so that C_0 is not a codeword and $C_1 := H(2, 0)$.

If decoding succeeds in Step 4' then $e := \text{Decode}(C_0, \text{Gamma}')$ and $b:=1$, Step 5 gives $C'1 := H(2, e)$, the confirmation check fails in Step 6 so gives $e:=s$ and $b:=0$, and Step 7 gives the unpredictable session key $K := H(0, s, C)$.

If decoding fails in Step 4' then $e:=0$ and $b:=0$, Step 5 gives $C'1 := H(2, 0)$, the confirmation check passes in Step 6 so keeps $e=0$ and $b=0$, and Step 7 gives the predictable session key $K := H(0, 0, C)$.

Distinguishing between these two cases gives enough information to recover messages using a standard reaction attack.

IND-CCA2 transforms are fragile. Any changes that deviate from the security proof for ease of implementation or efficiency reasons are potentially dangerous.

Kirk

From: daniel.apon <daniel.apon@nist.gov>
Sent: Monday, January 4, 2021 9:48 PM
To: pqc-forum
Cc: Kirk Fleming; pqc-forum; pqc-comments
Subject: Re: ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Hi Kirk,

"Distinguishing between these two cases gives enough information to recover messages using a standard reaction attack."

Would you describe what you mean in more detail?

Off hand, this seems like an implicit rejection vs explicit rejection issue, which raises various important issues, but doesn't seem to directly lead to a "standard reaction attack" in a straightforward manner -- to my knowledge. In particular, I don't see how to distinguish between decryption failures (of honestly generated ciphertexts) and decryption failures (of intentionally malformed ciphertexts).

Hope I'm not misunderstanding,
--Daniel

On Monday, January 4, 2021 at 8:57:08 PM UTC-5 Kirk Fleming wrote:

The Classic McEliece submission explicitly allows a change to decapsulation which breaks the IND-CCA2 security proof and leaves the scheme vulnerable to decoding failure attacks.

The decapsulation algorithm described in Section 2.3.3 is:

1. Split the ciphertext C as (C_0, C_1) .
2. Set $b := 1$.
3. Extract s and Γ' from the private key.
4. Compute $e := \text{Decode}(C_0, \Gamma')$.
If $e = \text{Fail}$ set $e := s$ and $b := 0$.
5. Compute $C'_1 := H(2, e)$.
6. If $C'_1 \neq C_1$ set $e := s$ and $b := 0$.
7. Compute $K := H(b, e, C)$.
8. Output session key K .

However, the discussion in the paragraphs following the algorithm description suggests a change to Step 4 for ease of implementation.

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

This is not true. Using a known fixed string trivially breaks the scheme.

To see why, consider decapsulation where Step 4 is changed to:

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, January 5, 2021 1:28 AM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack
Attachments: signature.asc

Kirk Fleming writes:

> "As an implementation note, the output of decapsulation
> is unchanged if 'e <-- s' in Step 4 is changed to assign
> something else to e. Implementors may prefer, e.g., to
> set e to a fixed n-bit string or a random n-bit string
> other than s."
> This is not true. Using a known fixed string trivially breaks the
> scheme.

For some reason Mr. Fleming omitted the next sentence in the paragraph, which says "However, the definition of decapsulation does depend on e being set to s in Step 6."

---Dan

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Tuesday, January 5, 2021 2:35 AM
To: pqc-comments; pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

On Tue, Jan 5, 2021 at 1:28 AM D. J. Bernstein <djb@cr.yp.to> wrote:

Kirk Fleming writes:

- > "As an implementation note, the output of decapsulation
- > is unchanged if 'e <-- s' in Step 4 is changed to assign
- > something else to e. Implementors may prefer, e.g., to
- > set e to a fixed n-bit string or a random n-bit string
- > other than s."
- > This is not true. Using a known fixed string trivially breaks the
- > scheme.

For some reason Mr. Fleming omitted the next sentence in the paragraph, which says "However, the definition of decapsulation does depend on e being set to s in Step 6."

This is an irrelevant and totally inadequate response to a serious issue. Fleming's example leaves Step 6 as-is, including the "e := s" part (as required by the sentence you quoted). And it demonstrates that the output of decapsulation *does* change if Step 4 is changed as suggested by the submission, contradicting the associated claim and even apparently breaking CCA security.

Sincerely yours in cryptography,
Chris

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, January 5, 2021 4:29 AM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack
Attachments: signature.asc

Christopher J Peikert writes:

> This is an irrelevant and totally inadequate response to a serious issue.
> Fleming's example leaves Step 6 as-is, including the "e := s" part (as
> required by the sentence you quoted). And it demonstrates that the
> output of decapsulation **does** change if Step 4 is changed as
> suggested by the submission, contradicting the associated claim and
> even apparently breaking CCA security.

Huh?

The specification indisputably requires e to be set to s in both failure cases, Step 4 and Step 6.

In the case of a Step 4 failure, the computation using e in Step 5 is irrelevant to the output, since the result is thrown away in Step 6.

This fact opens up more implementation options, using whatever e you want in Step 5 in case of failure, but you still have to set e to s in Step 6. This is exactly what the implementation note says:

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

If you maliciously misinterpret this by omitting the "but"/"However" part then of course you get different results and a fake contradiction.

---Dan

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Tuesday, January 5, 2021 9:50 AM
To: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Let's put the relevant lines of pseudocode and implementation note together, and consider the potential interpretations:

4. Compute $e := \text{Decode}(C_0, \text{Gamma}')$.
If $e = \text{Fail}$ set $e := s$ and $b := 0$.
5. Compute $C'1 := H(2, e)$.
6. If $C'1 \neq C1$ set $e := s$ and $b := 0$.

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

A natural interpretation of this -- in my opinion, the most natural one -- is:

- (a) in Step 4 you can replace " $e := s$ " with " $e := 00\dots 0$ " (n bits);
- (b) but in Step 6 you must preserve the " $e := s$ ".

(Notably, both " $e := \dots$ " assignments are behind conditionals, and the note says nothing about changing the logical control flow of the code. In particular, there's no instruction that the outcome of the test in Step 4 should influence the action taken in Step 6.)

Reader, do you find this to be a reasonable interpretation of the note? If so, Dan seems to think that you have "maliciously misinterpret[ed]" it. In any case, this interpretation leads to a total break of CCA security, as Fleming has demonstrated.

Now let's consider the interpretation that Dan appears to have:

- (a) in Step 4 you can replace " $e := s$ " with " $e := 00\dots 0$ ";
- (b) but for Step 6 you must change the control flow. Specifically:
 - (b1) test whether e was 'Fail' in Step 4 (e has since been overwritten, but we can check if $b=0$);
 - (b2) if so, then *unconditionally* set $e := s$, regardless of whether $C'1 \neq C1$ or not;
 - (b3) otherwise, do Step 6 as written above.

(It's plausible that this interpretation preserves CCA security, but I may have missed something, so no promises.)

I'll freely grant that if you know why the tests in Steps 4 and 6 are there, and why those steps set $e := s$ --- perhaps because you know the subtleties of the CCA security proof --- and if you intelligently use this knowledge to extrapolate from the text of the implementation note, then you can see this interpretation as being consistent with it.

But is this really the most natural interpretation? More to the point, is it one that implementors --- who may not have the above knowledge --- are likely to arrive at on their own from reading the note? I strongly doubt it. (No offense intended, implementors.)

At the very least, the note is ambiguous, and a perfectly reasonable reading of it leads to a devastating security failure. So, the specification should be updated to ensure that there is no ambiguity.

Sincerely yours in cryptography,
Chris

From: Kirk Fleming <kpfleming@mail.com>
Sent: Tuesday, January 5, 2021 3:00 PM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Hi Daniel,

Daniel Apon wrote:

>
> "Distinguishing between these two cases gives enough
> information to recover messages using a standard
> reaction attack."
>
> Would you describe what you mean in more detail?

Let $C = He$ for an unknown error e of weight t . Construct

$$C' = C + H_i + H_j$$

where H_i and H_j are a pair of columns of H . This corresponds to $C' = He'$ where the error e' is obtained from e by flipping the bits in positions i and j . In other words $e' = e + e_i + e_j$ where e_i and e_j are standard basis vectors.

If C' decodes successfully then you know that e' must also have weight t . This implies that e has a bit set in exactly one of the positions i or j . You can recover the full error e with at most $n-1$ spoofs of this form.

> Off hand, this seems like an implicit rejection vs
> explicit rejection issue, which raises various
> important issues, but doesn't seem to directly lead
> to a "standard reaction attack" in a straightforward
> manner -- to my knowledge
>
> In particular, I don't see how to distinguish between
> decryption failures (of honestly generated ciphertexts)
> and decryption failures (of intentionally malformed
> ciphertexts)

All the ciphertexts are malformed. The attack works because you learn whether rejection happens at Step 4 because of a decoding failure (decapsulation outputs the predictable session key) or at Step 6 because of a confirmation failure (decapsulation outputs a different session key).

Kirk

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, January 5, 2021 3:18 PM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack
Attachments: signature.asc

Christopher J Peikert writes:

- > 4. Compute $e := \text{Decode}(C_0, \Gamma)$.
- > If $e = \text{Fail}$ set $e := s$ and $b := 0$.
- > 5. Compute $C'_1 := H(2, e)$.
- > 6. If $C'_1 \neq C_1$ set $e := s$ and $b := 0$.

This is an excerpt from the specification of the decapsulation function.

The specification requires e to be set to s (and b to be set to 0; otherwise b is 1) in both failure cases, Step 4 and Step 6, producing output $H(0, s, C)$.

If there's a Step 4 failure then Step 6 is a no-op, since e is already s (and b is already 0). The rest of the algorithm shows that there is no use of C'_1 after Step 6. Consequently, in the case of a Step 4 failure, the computation using e in Step 5 is irrelevant to the output.

This fact opens up more implementation options, using whatever e you want in Step 5 in case of failure, but you still have to set e to s in Step 6. This, once again, is exactly what the implementation note says:

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

I already wrote essentially all of this in my previous message, the message that Dr. Peikert (1) claims to be replying to, and (2) posts various characterizations of, but (3) does not quote. I request that he follow the traditional email-handling practice of quoting and replying to each point, to reduce the risk of error and to assist readers tracking the discussions.

- > A natural interpretation of this -- in my opinion, the most natural one -- is:
- > (a) in Step 4 you can replace " $e := s$ " with " $e := 00\dots 0$ " (n bits);
- > (b) but in Step 6 you must preserve the " $e := s$ ".

This "interpretation" would replace the specified mathematical function with a different function, contradicting

(1) the paragraph's explicit statement that "the output of decapsulation is unchanged" and

(2) the paragraph's explicit label as an "implementation note",

not to mention that this "interpretation" would make the paragraph have the same meaning as the paragraph `_without_` the final sentence (the sentence that Mr. Fleming omitted), begging the question of why that sentence is there.

Even for readers who manage to miss the final sentence, this is two direct contradictions with what's stated at the beginning of the same paragraph. This "interpretation" is simply not correct.

> (Notably, both "e := ..." assignments are behind conditionals, and the
> note says nothing about changing the logical control flow of the code.

I don't know what Dr. Peikert thinks he means by "the logical control flow", so the claim of a change is hard to evaluate, never mind the question of why this is supposed to be relevant. In the end everything is fed through standard transformations to constant-time code, so the data flow ends up being independent of inputs.

> In
> particular, there's no instruction that the outcome of the test in
> Step 4 should influence the action taken in Step 6.)

The specification indisputably requires e to be set to s in both of the failure cases, producing a final output of H(0,s,C) in those cases. The implementation note correctly points out the implementor's freedom to vary an intermediate computation without affecting the final output.

> Reader, do you find this to be a reasonable interpretation of the note?

An "interpretation" that contradicts what's stated in the text is incorrect. Repeatedly claiming that an incorrect "interpretation" is "natural" and "reasonable" does not change the fact that it's wrong.

> If so, Dan seems to think that you have "maliciously misinterpret[ed]"
> it.

What I stated was "If you maliciously misinterpret this by omitting the 'but'/'However' part then of course you get different results and a fake contradiction." It's of course also possible for one person maliciously misinterpreting it to deceive other, non-malicious, people, which is an example of why something as important as standardizing post-quantum crypto should have evaluation procedures designed to resist attack.

> In any case, this interpretation leads to a total break of CCA
> security, as Fleming has demonstrated.

"Doctor, doctor, it hurts when I don't do what the documentation tells me to do."

There are some types of bugs in post-quantum software that are hard to catch with standard verification techniques. This isn't one of them.

> Now let's consider the interpretation that Dan appears to have:

Many people expect the word "interpretation" to mean that there's an ambiguity. There's no ambiguity here.

Furthermore, most people expect the word "appears" to indicate some lack of clarity. I see nothing unclear in the message that Dr. Peikert claims to be replying to here. If he thinks there's a lack of clarity in something I wrote, the normal way forward would be to quote it and ask for clarification.

> (a) in Step 4 you can replace "e := s" with "e := 00...0";

- > (b) but for Step 6 you must change the control flow. Specifically:
- > (b1) test whether e was 'Fail' in Step 4 (e has since been overwritten, but we can check if b=0);
- > (b2) if so, then *unconditionally* set e:=s, regardless of whether C'1 != C1 or not;
- > (b3) otherwise, do Step 6 as written above.
- > (It's plausible that this interpretation preserves CCA security, but I may have missed something, so no promises.)

Consider the following line of pseudocode:

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

Notice that this line is much more concise, and much easier to read, than the above series of several b-b1-b2-b3 lines.

Why, one wonders, was something so short and simple stated in such an obfuscated b-b1-b2-b3 way, using vastly more space than necessary, with a claim of a "change" in the "control flow" etc., a scary-sounding comment about possible losses of CCA security, etc.?

Most readers seeing this obfuscated presentation of X won't take the time to strip away the obfuscation---they'll simply assume, incorrectly, that X really is so complicated, and they'll tend to wonder whether a short implementation note could really have been saying something so complicated. This obfuscation thus contributes to the effort to convince readers to accept a wrong "interpretation" of the implementation note, rather than taking the implementation note to mean exactly what it says.

- > I'll freely grant that if you know why the tests in Steps 4 and 6 are there, and why those steps set e:=s --- perhaps because you know the subtleties of the CCA security proof --- and if you intelligently use this knowledge to extrapolate from the text of the implementation note, then you can see this interpretation as being consistent with it.

I don't see any overt errors in the statement being "freely" granted here. However, the paragraph tries to make readers believe that one needs to go to all this work to understand a simple implementation note. This belief is simply wrong.

I already spelled out the data-flow details that allow implementors to vary e in this way without changing the function being computed. Those details are entirely about the structure of the function being computed.

Again, the function is unchanged, so the question of why the function is designed this way is irrelevant, and the question made no appearance in my explanation.

More broadly, there's an impressive circularity between

- (1) repeatedly talking about CCA security to make the reader think that the function being computed is changing, and
- (2) repeatedly claiming that the function being computed is changing to make the reader think that there's a CCA security question,

all of which is completely out of whack with the fact that this is explicitly an implementation note, correctly and explicitly saying that it computes the same function.

- > But is this really the most natural interpretation? More to the point,

- > is it one that implementors --- who may not have the above knowledge
- > --- are likely to arrive at on their own from reading the note? I
- > strongly doubt it. (No offense intended, implementors.)

Again biased wording ("natural", "strongly doubt", etc.) is being used to push an "interpretation" of a paragraph that directly contradicts what the paragraph says.

- > At the very least, the note is ambiguous, and a perfectly reasonable
- > reading of it leads to a devastating security failure.

No. The claimed ambiguity is resolved by what the note says. Calling an incorrect interpretation "perfectly reasonable" does not make it so.

- > So, the specification should be updated to ensure that there is no ambiguity.

We are talking about an explicitly labeled `_implementation note_` that explicitly and correctly says that it does `_not_` change the function being computed. The explicit text is not consistent with your claimed "interpretation". Furthermore, it's simply wrong to label a request for a change in the note as requesting a change in the specification; we should all be clear about the roles of specifications, reference implementations, further implementations, and further text.

---Dan

P.S. Side question for NIST: Is it okay to be putting extra text into the subject line of OFFICIAL COMMENTS, as in this thread? This isn't what the official links show, but it would generally be helpful for tracking other discussions, for example to split the discussions of Kyber's patent issues from the discussions of Kyber-512's security.

From: Greg Maxwell <gmaxwell@gmail.com>
Sent: Tuesday, January 5, 2021 4:36 PM
To: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

On Tue, Jan 5, 2021 at 8:18 PM D. J. Bernstein <djb@cr.yp.to> wrote:

>
> This fact opens up more implementation options, using whatever e you
> want in Step 5 in case of failure, but you still have to set e to s
> in Step 6. This, once again, is exactly what the implementation note says:
>
> As an implementation note, the output of decapsulation is unchanged
> if “ $e \leftarrow s$ ” in Step 4 is changed to assign something else to e.
> Implementors may prefer, e.g., to set e to a fixed n-bit string, or a
> random n-bit string other than s. However, the definition of
> decapsulation does depend on e being set to s in Step 6.

I believe the implementation note is a foot-gun and should be revised.

The following "are cautioned" text does a laudable job of making the point that the two causes of failure must not be distinguishable. But it is easily read as only raising the concern of distinguishability only in terms of timing, leaving open the possibility of the cases being distinguishable by K taking on an attacker predictable value.

The note's statement about step 6 could be misread as saying "even though you're allowed to fill e with whatever in step 4, when C1' !=

C1 you must still follow step 6 and set e=s before step 7" in other words, only if the hash check fails. This way of reading it is vulnerable to the attack described in Kirk's post.

In particular, if an implementation follows that logic and in step 4 sets e to some implementation specific non-secret constant, this implementation flaw would be impossible to detect with generic test vectors, though it could be caught by review.

I believe the note could be changed to "As an implementation note, the output of decapsulation is unchanged if “ $e \leftarrow s$ ” in Step 4 is changed to assign something else to e, so long as the comparison in Step 6. is forced to consider C1' and C1 not equal when Decode fails."

Would that be correct?

Though even that is still a little opaque without any discussion of WHY you might want to implement it that way.

Clearly any implementation is free to implement it in whatever way computes exactly the same function (particularly if doing so doesn't create sidechannel vulnerabilities...), so an implementation note that simply reminds you of that without even suggesting a concrete optimization isn't necessarily the most useful. But my understanding of the note is that it is actually telling you that you're allowed to change the function so that it always fails if decoding fails, rather than happening to pass with $1:2^{256}$ probability when decoding fails should the invalid input happen to guess right $H(2,s)$. Is that correct?

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Tuesday, January 5, 2021 5:04 PM
To: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Dan, your latest reply contains a good deal of code analysis, mixed with the text of the implementation note, mixed with the words of the authors in this thread.

But the central issue is what the specification says on its own, and what implementors will do with it.

So, for the sake of concision, clarity, and precision, I suggest the following test. Please provide a modification of the original pseudocode in which:

- (a) Steps 1-3 are unchanged;
- (b) In Step 4, "e:=s" is changed to "e:=00...0", as allowed by the implementation note;
- (c) Steps 5-?? are the result of whatever you think the note instructs the implementor to do here.

Readers can then judge for themselves whether implementors would arrive at your modified pseudocode, on their own, by reading the spec.

Kirk Fleming's original message provided such modified code based on his reading of the spec -- namely, for (c) he retained the "e:=s" in Step 6, presumably believing that to follow the instruction "However, the definition of decapsulation does depend on e being set to s in Step 6."

I believe it's likely that implementors would naturally arrive at Fleming's pseudocode, even though it is completely insecure (under CCA). That indicates an ambiguity problem with the spec as currently written.

To conclude, I would like to point out one piece of logical sleight-of-hand to readers:

On Tue, Jan 5, 2021 at 3:18 PM D. J. Bernstein <djb@cr.yp.to> wrote:

Christopher J Peikert writes:

- > 4. Compute $e := \text{Decode}(C_0, \Gamma)$.
- > If $e = \text{Fail}$ set $e:=s$ and $b:=0$.
- > 5. Compute $C'_1 := H(2, e)$.
- > 6. If $C'_1 \neq C_1$ set $e:=s$ and $b:=0$.

This is an excerpt from the specification of the decapsulation function. The specification requires e to be set to s (and b to be set to 0; otherwise b is 1) in both failure cases, Step 4 and Step 6, producing output $H(0, s, C)$.

If there's a Step 4 failure then Step 6 is a no-op, since e is already s (and b is already 0). The rest of the algorithm shows that there is no use of C'_1 after Step 6. Consequently, in the case of a Step 4 failure, the computation using e in Step 5 is irrelevant to the output.

This fact opens up more implementation options, using whatever e you

want in Step 5 in case of failure, but you still have to set e to s in Step 6. This, once again, is exactly what the implementation note says:

As an implementation note, the output of decapsulation is unchanged if " $e \leftarrow s$ " in Step 4 is changed to assign something else to e .

As Kirk Fleming pointed out in his original message, the spec's claim "the output of decapsulation is unchanged...", which refers to the original pseudocode, is simply not true. Specifically, if we replace $e:=s$ with $e:=0$ in Step 4, then the decapsulation output can be changed by letting $C'1 = H(2,0)$. I don't think Dan has acknowledged this error.

I emphasize: in the spec, the "output is unchanged" claim is made *before* any suggested modifications to the code, and is used to motivate the possibility of such changes, so it must be referring to the original code.

Here is the sleight of hand: in his argument quoted above, Dan has *first* changed Step 6 to include an unconditional assignment $e:=s$ in the case of a Step 4 failure, *then* quotes the "output is unchanged" claim as being "exactly what the implementation note says." With that change to Step 6, the claim becomes plausible -- but the claim in the spec refers to the original code, and is false.

Sincerely yours in cryptography,
Chris

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Tuesday, January 5, 2021 5:09 PM
To: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

On Tue, Jan 5, 2021 at 5:04 PM Christopher J Peikert <cpeikert@alum.mit.edu> wrote:

As Kirk Fleming pointed out in his original message, the spec's claim "the output of decapsulation is unchanged...", which refers to the original pseudocode, is simply not true. Specifically, if we replace $e:=s$ with $e:=0$ in Step 4, then the decapsulation output can be changed by letting $C'1 = H(2,0)$. I don't think Dan has acknowledged this error.

Please excuse a typo here: it should be $C1 = H(2,0)$, not $C'1$.

Sincerely yours in cryptography,
Chris

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Tuesday, January 5, 2021 5:22 PM
To: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

I now realize that I used the term "spec" or "specification" in some places where Dan would use the separate term "implementation note." Feel free to replace each occurrence with whatever term is best in context, or replace them all with "submission"; it won't affect the substance of my comments.

Sincerely your in cryptography,
Chris

On Tue, Jan 5, 2021 at 5:04 PM Christopher J Peikert <cpeikert@alum.mit.edu> wrote:

Dan, your latest reply contains a good deal of code analysis, mixed with the text of the implementation note, mixed with the words of the authors in this thread.

But the central issue is what the specification says on its own, and what implementors will do with it.

So, for the sake of concision, clarity, and precision, I suggest the following test. Please provide a modification of the original pseudocode in which:

- (a) Steps 1-3 are unchanged;
- (b) In Step 4, "e:=s" is changed to "e:=00...0", as allowed by the implementation note;
- (c) Steps 5-?? are the result of whatever you think the note instructs the implementor to do here.

Readers can then judge for themselves whether implementors would arrive at your modified pseudocode, on their own, by reading the spec.

Kirk Fleming's original message provided such modified code based on his reading of the spec -- namely, for (c) he retained the "e:=s" in Step 6, presumably believing that to follow the instruction "However, the definition of decapsulation does depend on e being set to s in Step 6."

I believe it's likely that implementors would naturally arrive at Fleming's pseudocode, even though it is completely insecure (under CCA). That indicates an ambiguity problem with the spec as currently written.

To conclude, I would like to point out one piece of logical sleight-of-hand to readers:

On Tue, Jan 5, 2021 at 3:18 PM D. J. Bernstein <djb@cr.yp.to> wrote:

Christopher J Peikert writes:

- > 4. Compute $e := \text{Decode}(C_0, \text{Gamma})$.
- > If $e = \text{Fail}$ set $e:=s$ and $b:=0$.
- > 5. Compute $C'_1 := H(2, e)$.
- > 6. If $C'_1 \neq C_1$ set $e:=s$ and $b:=0$.

This is an excerpt from the specification of the decapsulation function. The specification requires e to be set to s (and b to be set to 0; otherwise b is 1) in both failure cases, Step 4 and Step 6, producing

From: Mike Hamburg <mike@shiftright.org>
Sent: Tuesday, January 5, 2021 5:27 PM
To: D. J. Bernstein
Cc: pqc-comments; pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Hi Dan,

On Jan 5, 2021, at 4:18 PM, D. J. Bernstein <djb@cr.yp.to> wrote:

Christopher J Peikert writes:

A natural interpretation of this -- in my opinion, the most natural one -- is:
(a) in Step 4 you can replace "e := s" with "e := 00...0" (n bits);
(b) but in Step 6 you must preserve the "e := s".

This "interpretation" would replace the specified mathematical function with a different function, contradicting

(1) the paragraph's explicit statement that "the output of decapsulation is unchanged" and

(2) the paragraph's explicit label as an "implementation note",

not to mention that this "interpretation" would make the paragraph have the same meaning as the paragraph `_without_` the final sentence (the sentence that Mr. Fleming omitted), begging the question of why that sentence is there.

Even for readers who manage to miss the final sentence, this is two direct contradictions with what's stated at the beginning of the same paragraph. This "interpretation" is simply not correct.

(Notably, both "e := ..." assignments are behind conditionals, and the note says nothing about changing the logical control flow of the code.

I don't know what Dr. Peikert thinks he means by "the logical control flow", so the claim of a change is hard to evaluate, never mind the question of why this is supposed to be relevant. In the end everything is fed through standard transformations to constant-time code, so the data flow ends up being independent of inputs.

In particular, there's no instruction that the outcome of the test in Step 4 should influence the action taken in Step 6.)

The specification indisputably requires e to be set to s in both of the failure cases, producing a final output of $H(0,s,C)$ in those cases. The implementation note correctly points out the implementor's freedom to vary an intermediate computation without affecting the final output.

Reader, do you find this to be a reasonable interpretation of the note?

An "interpretation" that contradicts what's stated in the text is incorrect. Repeatedly claiming that an incorrect "interpretation" is "natural" and "reasonable" does not change the fact that it's wrong.

If so, Dan seems to think that you have "maliciously misinterpret[ed]" it.

What I stated was "If you maliciously misinterpret this by omitting the 'but'/'However' part then of course you get different results and a fake contradiction." It's of course also possible for one person maliciously misinterpreting it to deceive other, non-malicious, people, which is an example of why something as important as standardizing post-quantum crypto should have evaluation procedures designed to resist attack.

...

Many people expect the word "interpretation" to mean that there's an ambiguity. There's no ambiguity here.

...

More broadly, there's an impressive circularity between

- (1) repeatedly talking about CCA security to make the reader think that the function being computed is changing, and
- (2) repeatedly claiming that the function being computed is changing to make the reader think that there's a CCA security question,

all of which is completely out of whack with the fact that this is explicitly an `_implementation_note_`, correctly and explicitly saying that it computes the `_same_` function.

But is this really the most natural interpretation? More to the point, is it one that implementors --- who may not have the above knowledge --- are likely to arrive at on their own from reading the note? I strongly doubt it. (No offense intended, implementors.)

Again biased wording ("natural", "strongly doubt", etc.) is being used to push an "interpretation" of a paragraph that directly contradicts what the paragraph says.

At the very least, the note is ambiguous, and a perfectly reasonable reading of it leads to a devastating security failure.

No. The claimed ambiguity is resolved by what the note says. Calling an incorrect interpretation "perfectly reasonable" does not make it so.

So, the specification should be updated to ensure that there is no ambiguity.

We are talking about an explicitly labeled `_implementation note_` that explicitly and correctly says that it does `_not_` change the function being computed.

With all due respect, horseshit. The implementation note begins with a perfectly clear and demonstrably false statement:

""

As an implementation note, the output of decapsulation is unchanged if "`e←s`" in Step 4 is changed to assign something else to `e`.

""

This sentence is perfectly clear, self-contained, and ends with a full stop. And as Kirk pointed out, it is false: if `e` is assigned a fixed string in step 4, instead of being assigned `s`, then there are `(C_0, C_1)` for which the output of decapsulation does change. What's more, they are easy to compute.

There is no other plausible interpretation of your statement. If you meant to say that decapsulation is unchanged if, when `e=bot`, step 5 is replaced with

$K = H(b, \text{something else}, C)$

while leaving steps 4 and 6 alone, then you could have written this instead. Demanding that the reader replace your false statement with a different, true statement (in which we somehow additionally branch on `b` in step 6, or something?) is entirely unreasonable. Even more unreasonable is to call the straightforward reading of your false statement a "fake contradiction". It's not a fake contradiction. It's a real contradiction. False statements are contradictions, more or less by definition.

You might reasonably object that the false statement is transparently false, that it is essentially a typo or an imprecision, that any reasonable reader would understand it to mean some other, true statement. In that case all your critics in this thread would be picking nits. But several people in this thread have stated that they read your statement as written, and I have also read it this way.

I will also support my claim that the false statement is not transparently false or transparently a typo. There are similar flows in in CCA transforms where this sort of statement would be true. Consider a generic CCA transform from a perfectly-correct OW-CPA encryption algorithm, which performs in part:

```
3: e <- Decrypt(sk, C)
4: if e = bot: e <- s and b <- 0
5: C' <- Encrypt(pk, e)
6: if C' != C: e <- s and b <- 0
```

In that case, setting $e \leftarrow$ something else would indeed leave the decapsulation result unchanged, because when $e = \text{bot}$, since Decrypt is perfectly correct, there is no value of “something else” such that $\text{Encrypt}(\text{pk}, \text{something else}) = C$. Thus the branch on line 6 will always be taken. So not only is the implementation note wrong, there are other, naturally analogous contexts in which a similar note would not be wrong.

And indeed, Classic McEliece is based on a perfectly-correct OW-CPA decryption algorithm. In this case, of course, the same logic does not apply because C_1 is only part of the ciphertext.

Furthermore, one might say that the natural interpretation is unreasonable because the specification wouldn't demand $e \leftarrow s$, if indeed any value would suffice. But this isn't a valid objection either, because within that branch $e = \text{bot}$. Even if $\text{Encrypt}(\text{pk}, \text{bot})$ is somehow defined, it will not typically be implemented. So in a typical implementation, e would need to be replaced with something on that branch. The specification says to replace it with s , but the implementation note claims, falsely, that any other choice would give the same result.

The implementation note continues:

“”””

Implementors may prefer, e.g., to set e to a fixed n -bit string, or a random n -bit string other than s . However, the definition of decapsulation does depend on e being set to s in Step 6.

'''

These statements are both true. It is also true that implementors may prefer to use variable-time decoding algorithms, or to skip Step 7 entirely, or to pass attacker-controlled data in backticks to a shell, or to do any number of other insecure things. Mentioning these in an implementation note is transparently content-free, however, so the only reasonable reading is to recommend that changing line 4 to “e <- something else” would be acceptable, subject to the following that line 6 must be intact and that side-channel leaks must be prevented.

But as Kirk has demonstrated, if an implementor changes line 4 while leaving line 6 intact, exactly as this implementation note details, then the resulting system is not CCA-secure. So this statement, while technically content-free, constitutes a footgun at the very least.

Coming back to another objection you made:

not to mention that this "interpretation" would make the paragraph have the same meaning as the paragraph `_without_` the final sentence (the sentence that Mr. Fleming omitted), begging the question of why that sentence is there.

The sentence is clearly there to warn implementors that the analogous change in line 6 is forbidden and would be dangerous. This would be a perfectly reasonable warning, except that changing line 4 is also dangerous. But the reader presumably doesn't know that because you've asserted the opposite.

Your documentation (Look! I didn't say "specification"!) is wrong, and you really ought to fix it.

Grumble,
— Mike

From: daniel.apon <daniel.apon@nist.gov>
Sent: Tuesday, January 5, 2021 7:44 PM
To: pqc-forum
Cc: Kirk Fleming; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Hi Kirk,

Yes-- your attack seems correct. Well done. =)

On Tuesday, January 5, 2021 at 3:00:01 PM UTC-5 Kirk Fleming wrote:

Hi Daniel,

Daniel Apon wrote:

>
> "Distinguishing between these two cases gives enough
> information to recover messages using a standard
> reaction attack."
>
> Would you describe what you mean in more detail?

Let $C = He$ for an unknown error e of weight t . Construct

$$C' = C + H_i + H_j$$

where H_i and H_j are a pair of columns of H . This corresponds to $C' = He'$ where the error e' is obtained from e by flipping the bits in positions i and j . In other words $e' = e + e_i + e_j$ where e_i and e_j are standard basis vectors.

If C' decodes successfully then you know that e' must also have weight t . This implies that e has a bit set in exactly one of the positions i or j . You can recover the full error e with at most $n-1$ spoofs of this form.

> Off hand, this seems like an implicit rejection vs
> explicit rejection issue, which raises various
> important issues, but doesn't seem to directly lead
> to a "standard reaction attack" in a straightforward
> manner -- to my knowledge
>
> In particular, I don't see how to distinguish between
> decryption failures (of honestly generated ciphertexts)
> and decryption failures (of intentionally malformed
> ciphertexts)

All the ciphertexts are malformed. The attack works because you learn whether rejection happens at Step 4 because of a decoding failure (decapsulation outputs the predictable session key) or at Step 6 because of a confirmation failure (decapsulation outputs a different session key).

Kirk

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Wednesday, January 6, 2021 12:51 AM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack
Attachments: signature.asc

Greg Maxwell writes:

> The note's statement about step 6 could be misread as saying "even
> though you're allowed to fill e with whatever in step 4, when $C_1' \neq$
> C_1 you must still follow step 6 and set $e=s$ before step 7"

The note about e being set to s is not limited to "when $C_1' \neq C_1$ ".

That's an added constraint that changes the meaning, contradicts this being an "implementation note", and contradicts this leaving the output "unchanged". So this is indeed a misreading.

> I believe the note could be changed to "As an implementation note, the
> output of decapsulation is unchanged if " $e \leftarrow s$ " in Step 4 is changed to
> assign something else to e, so long as the comparison in Step 6. is
> forced to consider C_1' and C_1 not equal when Decode fails."

Sure, and then we'll have people again trying to score cheap political points by pretending that this means something other than what it says:

e.g., pretending that "forced" is making some sort of security claim about a modified function (rather than pointing out to the implementor a different way to compute the same function) and then attacking this (fabricated) security claim.

Pseudocode, such as the line

6. If $C_1' \neq C_1$, set $b:=0$. If $b=0$, set $e:=s$.

in my previous message, makes misinterpretation harder but doesn't make it impossible. The ultimate answer is automated verification, but this whole discussion makes zero contributions to ongoing verification work, while actively corrupting novice evaluations of what's safest.

> Though even that is still a little opaque without any discussion of
> WHY you might want to implement it that way.

The original text says "Implementors may prefer, e.g., to set e to a fixed n-bit string, or a random n-bit string other than s." The first part simplifies the data flow in some types of implementations, and the second part is worth considering as a component of side-channel countermeasures beyond timing-attack protection.

> But my understanding
> of the note is that it is actually telling you that you're allowed to
> change the function so that it always fails if decoding fails,

If by "fails" you mean computes and outputs $H(0,s,C)$, this isn't a change to the specified function.

> rather than happening to pass with $1:2^{256}$ probability when decoding
> fails should the invalid input happen to guess right $H(2,s)$.

No, Step 6 is a no-op in the "guess right" case. There's nothing probabilistic here.

---Dan

From: Quan Thoi Minh Nguyen <msuntmquan@gmail.com>
Sent: Wednesday, January 6, 2021 2:10 AM
To: pqc-forum
Cc: D. J. Bernstein; pqc-forum; pqc-comments; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

It's puzzled me why the discussion in this thread is heated. Can we all take it easy? It's clear that all the people (Classic McEliece's authors and critics) are knowledgeable and reputable and know what they're doing. Is it worth arguing about the "implementation note" (which no implementers used or implemented yet) at this phase of competition?

If we're worried about implementations' security then I would be more worried about multiple security bugs that Markku-Juhani O. Saarinen found in real implementations.

After the competition is concluded, is it *NIST's job* to rewrite everything in a clearer manner in the standards so that implementers can follow? A similar process for IETF's RFC standard which takes months (if not years) and multiple rounds of edit to become standard.

Cheers,
- Quan

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Wednesday, January 6, 2021 2:33 AM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack
Attachments: signature.asc

Christopher J Peikert writes:

> Please provide a modification of the original pseudocode in which:

I already did, in exactly the message you claim to be replying to:

6. If $C'_1 \neq C_1$, set $b:=0$. If $b=0$, set $e:=s$.

This meets the stated requirements of `_preserving the output_` ("output of decapsulation is unchanged") and "e being set to s in Step 6" on failure, along with everything else that this implementation note says.

> Readers can then judge for themselves whether implementors would
> arrive at your modified pseudocode, on their own, by reading the spec.

What I wrote above is fully compliant with every detail of this `_implementation note_`, correctly computing the `_specified_ function`.

> But the central issue is what the specification says on its own,

The `_specification_` produces output $H(0,s,C)$ in both failure cases.

The `_implementation note_` correctly describes different computations that reach the specified output in all cases.

> and what implementors will do with it.

Analyzing the ecosystem of implementations is indisputably important, including taking account of possible errors and measures taken to protect against errors---e.g., NIST's validation process for its current standards, and more advanced processes that have been developed.

> Kirk Fleming's original message provided such modified code based on
> his reading of the spec -- namely, for (c) he retained the "e:=s" in
> Step 6, presumably believing that to follow the instruction "However,
> the definition of decapsulation does depend on e being set to s in Step 6."

This "interpretation" contradicts the text of the implementation note in question. This isn't disputed.

> That indicates an ambiguity problem with the spec as currently written.

No, it doesn't.

> As Kirk Fleming pointed out in his original message, the spec's claim
> "the output of decapsulation is unchanged...", which refers to the

> original pseudocode, is simply not true.

There are multiple levels of errors in what Dr. Peikert claims here.

First, the discussion is about an explicitly labeled "implementation note", not the specification. One of the reasons this structure is important is that it constrains the meaning of the note: the note is not changing the specified function. (Dr. Peikert posts a followup message claiming, without analysis, that this difference "won't affect the substance of my comments".)

Second, the claim of something being "not true" rests entirely on pushing an "interpretation" that contradicts what the text explicitly says, while refusing to acknowledge the existence of a statement that's perfectly correct and matches every single detail of the text.

> Specifically, if we replace $e:=s$ with $e:=0$ in Step 4, then the
> decapsulation output can be changed by letting $C^1 = H(2,0)$.

Making only that change would flunk "However, the definition of decapsulation does depend on e being set to s in Step 6", would contradict this being an "implementation note", and would contradict "the output of decapsulation is unchanged".

> I don't think Dan has acknowledged this error.

There's no error.

> I emphasize: in the spec,

Wrong again. It's an implementation note.

> the "output is unchanged" claim is made
> *before* any suggested modifications to the code,

The claim is correct. This is, as it says, an implementation note about different ways to compute exactly the same function.

> and is used to
> motivate the possibility of such changes, so it must be referring to
> the original code.

Not exactly. The implementation note is starting from the pseudocode used in the spec, but it is then pointing out some different ways to compute the same function.

> Here is the sleight of hand: in his argument quoted above, Dan has
> *first* changed Step 6 to include an unconditional assignment $e:=s$ in
> the case of a Step 4 failure, *then* quotes the "output is unchanged"
> claim as being "exactly what the implementation note says." With that
> change to Step 6, the claim becomes plausible

When the biased presentation ("sleight of hand" etc.) is stripped away, this paragraph almost becomes a clear admission of what the implementation note was saying in the first place.

> -- but the claim in the spec refers to the original code, and is false.

No.

---Dan

From: pqc-forum@list.nist.gov on behalf of Kirk Fleming <kpffleming@mail.com>
Sent: Wednesday, January 6, 2021 7:54 PM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Quan Thoi Minh Nguyen wrote:

- > It's puzzled me why the discussion in this thread is
- > heated. Can we all take it easy? It's clear that all
- > the people (Classic McEliece's authors and critics)
- > are knowledgeable and reputable and know what they're
- > doing. Is it worth arguing about the "implementation
- > note" (which no implementers used or implemented yet)
- > at this phase of competition?
- >
- > If we're worried about implementations' security then
- > I would be more worried about multiple security bugs
- > that Markku-Juhani O. Saarinen found in real
- > implementations.
- >
- > After the competition is concluded, is it NIST's job
- > to rewrite everything in a clearer manner in the
- > standards so that implementers can follow? A similar
- > process for IETF's RFC standard which takes months
- > (if not years) and multiple rounds of edit to become
- > standard.

You're correct that it is NIST's responsibility to write a clear and unambiguous standard but they will use the submission as a basis for that. For Classic McEliece it's not always easy to tell which parts constitute the specification and which parts are just commentary. There are lots of notes for implementers. Some or all of these could be picked up by NIST and included in the standard.

Let's look at a different example. Section 2.2.4 of the submission suggests that one of the checks in the decoding subroutine can be made more efficient:

"In order to test $C0 = He$, implementors can use any parity-check matrix H' for the same code."

"There are various well-known choices of H' related to H^\wedge that are recovered from Γ much more efficiently than MatGen, and that can be applied to vectors without using quadratic space."

What the submission fails to mention is that while H is public other choices of H' may need to remain private. An implementer could miss this if they're not an expert so NIST would need to add a warning to the standard.

I'm not saying that things like this will necessarily lead to problems in the standard. I'd expect NIST to weed most of them out and hope the rest get spotted in the review process. I'm saying that when a submission offers so many different implementation options without

being sufficiently clear about them it makes NIST's job much harder. I think it should be our responsibility to help by surfacing issues as early as possible.

Kirk

--

You received this message because you are subscribed to the Google Groups "pqc-forum" group.

To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.

To view this discussion on the web visit <https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/trinity-dc2c9b65-e8c4-43af-873c-e4d932c80ef0-1609980867163%403c-app-mailcom-lxa02>.

From: Kirk Fleming <kpffleming@mail.com>
Sent: Wednesday, January 6, 2021 9:49 PM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Kirk Fleming wrote:

> Let's look at a different example. Section 2.2.4 of the
> submission suggests that one of the checks in the
> decoding subroutine can be made more efficient:
>
> "In order to test $C0 = He$, implementors can use any
> parity-check matrix H' for the same code."
>
> "There are various well-known choices of H' related
> to H^\wedge that are recovered from Γ' much more
> efficiently than MatGen, and that can be applied to
> vectors without using quadratic space."

I just realized that I don't understand why the $C0 = He$ check is there in the first place. The pseudocode for the decoding subroutine is:

1. Extend $C0$ to $v = (C0, 0, \dots, 0)$ 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 Fail.
3. Set $e = v + c$.
4. If $wt(e) = t$ and $C0 = He$, return e . Otherwise return Fail.

Let's skip over the fact that the submission doesn't actually describe the decoder and just points to the McBits papers. I think this expects too much of the implementer but whatever.

If Step 2 finds a codeword c and Step 3 sets $e = v + c$ then $He = H(v+c) = Hv + Hc = Hv = C0$ in Step 4 by construction. The only reason you'd need to check $C0 = He$ is if the decoder in Step 2 wasn't guaranteed to either return a codeword or fail but that contradicts the specification.

Kirk

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Thursday, January 7, 2021 4:08 AM
To: Kirk Fleming; pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Hi Kirk!

On 1/7/21 3:49 AM, Kirk Fleming wrote:

> I just realized that I don't understand why the C_0 = He check is there
> in the first place.

Maybe the following paragraph helps you to understand this:

"Implementors are cautioned that it is important to avoid leaking secret information through side channels, and that the distinction between success and failure of Decode is secret in the context of the Classic McEliece KEM. In particular, immediately stopping the computation when Step 2 returns Fail would reveal this distinction through timing, so it is recommended for implementors to have Step 2 always choose some c ."
(Small edits due to math symbols.)

Ruben

From: pqc-forum@list.nist.gov on behalf of Kirk Fleming <kpffleming@mail.com>
Sent: Thursday, January 7, 2021 6:20 AM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Hi Ruben,

Ruben Niederhagen wrote:

>
> On 1/7/21 3:49 AM, Kirk Fleming wrote:
>>
>> I just realized that I don't understand why the CO = He
>> check is there in the first place.
>
> Maybe the following paragraph helps you to understand this:
>
> "Implementors are cautioned that it is important to avoid
> leaking secret information through side channels, and that
> the distinction between success and failure of Decode is
> secret in the context of the Classic McEliece KEM. In
> particular, immediately stopping the computation when Step 2
> returns Fail would reveal this distinction through timing,
> so it is recommended for implementors to have Step 2 always
> choose some c."
> (Small edits due to math symbols.)

Choosing a random c is presumably not part of the specification since it's only a recommendation. It also doesn't explain the need for the check if we interpret the paragraph the way we've been told to interpret the implementation note for decapsulation. According to that logic if decoding fails in Step 2 and the implementation chooses a random c then the computation of e in Step 3 and both checks in Step 4 are irrelevant because Step 4 must always return Fail.

Kirk

--

You received this message because you are subscribed to the Google Groups "pqc-forum" group.

To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.

To view this discussion on the web visit <https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/trinity-80898fb3-8621-4386-8e63-02739cca02ca-1610018396712%403c-app-mailcom-lxa01>.

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Thursday, January 7, 2021 6:38 AM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

On 1/7/21 12:19 PM, Kirk Fleming wrote:

> Ruben Niederhagen wrote:

> >

> > On 1/7/21 3:49 AM, Kirk Fleming wrote:

> >>

> >> I just realized that I don't understand why the $C_0 = H_e$

> >> check is there in the first place.

> >

> > Maybe the following paragraph helps you to understand this:

> >

> > "Implementors are cautioned that it is important to avoid

> > leaking secret information through side channels, and that

> > the distinction between success and failure of Decode is

> > secret in the context of the Classic McEliece KEM. In

> > particular, immediately stopping the computation when Step 2

> > returns Fail would reveal this distinction through timing,

> > so it is recommended for implementors to have Step 2 always

> > choose some c ."

> > (Small edits due to math symbols.) Choosing a random c is

> > presumably not part of the specification since it's only a

> > recommendation. It also doesn't explain the need for the check if we

> > interpret the paragraph the way we've been told to interpret the

> > implementation note for decapsulation.

> > According to that logic if decoding fails in Step 2 and the

> > implementation chooses a random c then the computation of e in Step 3

> > and both checks in Step 4 are irrelevant because Step 4 must always

> > return Fail.

OK, the question why the $C_0 = H_e$ check is there seems to be answered.

I'll file this under 'criticism of the writing style', then.

Ruben

From: Paul Hoffman <paul.hoffman@icann.org>
Sent: Thursday, January 7, 2021 10:58 AM
To: Ruben Niederhagen
Cc: pqc-forum; pqc-comments
Subject: Re: [Ext] [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

<lurker decloaking>

> On Jan 7, 2021, at 3:37 AM, Ruben Niederhagen <ruben@polycephaly.org> wrote:

>

> On 1/7/21 12:19 PM, Kirk Fleming wrote:

>> Ruben Niederhagen wrote:

>> >

>> > On 1/7/21 3:49 AM, Kirk Fleming wrote:

>> >>

>> >> I just realized that I don't understand why the $C_0 = H_e$

>> >> check is there in the first place.

>> >

>> > Maybe the following paragraph helps you to understand this:

>> >

>> > "Implementors are cautioned that it is important to avoid

>> > leaking secret information through side channels, and that

>> > the distinction between success and failure of Decode is

>> > secret in the context of the Classic McEliece KEM. In

>> > particular, immediately stopping the computation when Step 2

>> > returns Fail would reveal this distinction through timing,

>> > so it is recommended for implementors to have Step 2 always

>> > choose some c ."

>> > (Small edits due to math symbols.)

>> Choosing a random c is presumably not part of the specification

>> since it's only a recommendation. It also doesn't explain the

>> need for the check if we interpret the paragraph the way we've

>> been told to interpret the implementation note for decapsulation.

>> According to that logic if decoding fails in Step 2 and the

>> implementation chooses a random c then the computation of e in

>> Step 3 and both checks in Step 4 are irrelevant because Step 4

>> must always return Fail.

>

> OK, the question why the $C_0 = H_e$ check is there seems to be answered.

>

> I'll file this under 'criticism of the writing style', then.

Speaking as a specification writer: please don't do such filing.

The phrase "so it is recommended" leaves the implementer with three choices:

a) do what comes after the recommendation

b) do some other thing that the implementer makes up themselves

c) do what they thought was implied by the text preceding the recommendation

If you mean (a), the text needs to be updated to say "so implementers must".

If you mean (b), then the specification is both incomplete and dangerous. Some implementers will pick something really smart-looking that is wrong.

If you mean (c), then the specification is both poorly-written and dangerous. Some implementers will think that what you were hinting at is something that turns out to be wrong.

I hope you meant (a); if so, you need to update the specification to say so. As far as I can tell from my mild skimming of this thread, such updates are needed in other places in this specification in order to make it clear.

It would be good for the safety of everyone if the writers ****of all the specifications here**** would review their submissions and, wherever there is implementation recommendations, turn them into requirements. The crypto world has the same track record as the Internet protocol world of unclear specifications that lead to security disasters.

--Paul Hoffman, going back to lurking

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Thursday, January 7, 2021 11:48 AM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [Ext] [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

On 1/7/21 4:57 PM, Paul Hoffman wrote:

> It would be good for the safety of everyone if the writers **of all
> the specifications here** would review their submissions and, wherever
> there is implementation recommendations, turn them into requirements.
I do not entirely agree (although I do cherish the sentiment).

The specific recommendation here (talking about the "decoding discussion" on the C0 = He check) is about side channels.

Now, is it the task of a specification or of an implementation to provide security against side-channel attacks?

Let's take the definition of 'side-channel attack' from Wikipedia:

"[...] a side-channel attack is any attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself [...]"

A specification provides the algorithm and tells you what to implement, but not explicitly how to implement it; the specification is not the implementation.

Furthermore, a specification could hardly cover all sources of side channels. (Is it a hardware implementation? Is it a software implementation? What is the application, what is the use case - which side channels are relevant? ..?)

Thus, I'd say it is hardly the task for the specification to provide side-channel protection; this burden is with the implementer.

Should a specification provide recommendations to implementers in regards to side channels? "Yes, sure"? "No, never"? "Yes" if it helps to increase security, "no" if it does not..? Sorry, I have no proper answer here.

Ruben, also going back to lurking

From: Rainer Urian <rainer.urian@googlemail.com>
Sent: Thursday, January 7, 2021 1:19 PM
To: Ruben Niederhagen
Cc: pqc-forum; pqc-comments
Subject: Re: [Ext] [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

> Furthermore, a specification could hardly cover all sources of side
> channels. (Is it a hardware implementation? Is it a software implementation? What is the application, what is the use
case - which side channels are relevant? ..?) Thus, I'd say it is hardly the task for the specification to provide side-channel
protection; this burden is with the implementer.

I totally agree. Side-channel and fault-attack countermeasures are strongly related to the device and the environment
where the device is used.

For instance, they are rather different on PC and smart card devices.

Also, timing attacks cover only a very limited part of all possible side-channel attack scenarios.

So, there is never one-size-fits-all solution.

> Should a specification provide recommendations to implementers in regards to side channels? "Yes, sure"? "No,
never"? "Yes" if it helps to increase security, "no" if it does not..? Sorry, I have no proper answer here.

In my opinion, the specification should take care about all points related to crypto-analytic attacks. All points related to
side-channel and fault attacks shall be informative only.

According to my understanding, this is also the philosophy behind the current FIPS/NIST SP specifications.

BR,
Rainer

From: 'daniel.apon' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Thursday, January 7, 2021 2:16 PM
To: pqc-forum
Cc: rainer...@googlemail.com; pqc-forum; pqc-comments; Ruben Niederhagen
Subject: Re: [Ext] [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Hi Rainer and all,

Rainer's recent message (copied below) prompts me to re-highlight some brief informational content on NIST PQC's attitude/philosophy toward side-channel attacks.

As I haven't spoken with the team yet, I should highlight this as "*speaking for myself*," although I believe it's representative of the general view.

First, NIST has stated the importance of side-channel attacks and countermeasures a few times before.

For example, in the original PQC call for proposals in 2016:

"Schemes that can be made resistant to side-channel attacks at minimal cost are more desirable than those whose performance is severely hampered by any attempt to resist side-channel attacks."

For a more recent example, in the latest PQC 2nd Round Report (NISTIR 8309) in 2020:

"NIST hopes to see more and better data for performance in the third round. This performance data will hopefully include implementations that protect against side-channel attacks, such as timing attacks, power monitoring attacks, fault attacks, etc."

Second, as Rainer points out, the philosophy behind current FIPS/NIST SP specifications indeed only makes informative comments about side-channel attacks (passive or active). That is, while I suppose it's possible that NIST would choose to standardize some type of side-channel resistant PQC specification directly, this has not been the case historically, and I find it unlikely to be the case for the eventual PQC standards as well.

Nonetheless, we are certainly paying attention to side-channel attacks and their analysis, and it *could* be the case (all else equal) that this information directly factors into our eventual choice of this algorithm vs. that algorithm to standardize -- particularly if one algorithm/system seems to clearly lend itself to "side-channel resistance friendly implementations" more readily than another that is an otherwise close / near-indistinguishable competitor. Additionally, we certainly would not (or at least, should not) standardize an algorithm/system for which we believe side channel resistance is critical to its real-world adoption/application but where side-channel resistance appears inherently overly burdensome to come by.

So -- echoing the NIST team's prior statements on the issue: This is an important issue to think about, especially during the third round.

I sincerely appreciate everyone's earnest input to this discussion (especially, you lurkers out there :); it's very helpful.

Cheers,
--Daniel

From: Kirk Fleming <kpfleming@mail.com>
Sent: Friday, January 8, 2021 9:55 AM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

Ruben Niederhagen wrote:

>
> OK, the question why the $C_0 = H_e$ check is there seems to be
> answered.
>
> I'll file this under 'criticism of the writing style', then.

I confess. It was a leading question intended to let me complain about interpretations of implementation notes. That was mean and I apologize. There's still a serious point behind the question, though. The $C_0 = H_e$ check really isn't needed if you trust the decoder to either return a codeword or fail.

The submission recommends that when the decoder fails to find a codeword "Step 2 chooses some vector c in $(F_2)^n$ and continues on to Step 3."

What does the submission mean by "some vector c "? Does it mean that the implementation must choose a random c or can it set c to a fixed value?

If it's intended to be read as "choose a random c " then you're correct that you do need to check $C_0 = H_e$. The $wt(e) = t$ check will almost surely fail but the IND-CCA2 security proof requires the PKE scheme to be perfect. No decryption failures are allowed no matter how negligible the probability.

What about using a fixed value? It's actually better to set $c = 0$ when the decoder fails. In this case Step 3 sets $e = v$ and the check $C_0 = H_e$ is trivially true so can be skipped. On the other hand the check $wt(e) = t$ always fails since otherwise $c = 0$ is the nearest codeword and would have been found by the decoder.

Poorly written implementation notes are not just a problem for security. They can also be a problem for efficiency if they stop implementers from making changes that are genuinely safe.

Kirk

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Friday, January 8, 2021 11:36 AM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece decoding failure attack

On 1/8/21 3:55 PM, Kirk Fleming wrote:

> I confess. It was a leading question intended to let me complain about
> interpretations of implementation notes. That was mean and I
> apologize.

Apology accepted.

> There's still a serious point behind the question, though. The $C_0 = H_e$
> check really isn't needed if you trust the decoder to either return a
> codeword or fail.
> The submission recommends that when the decoder fails to find a
> codeword "Step 2 chooses some vector c in $(F_2)^n$ and continues on to
> Step 3."
> What does the submission mean by "some vector c "? Does it mean that
> the implementation must choose a random c or can it set c to a fixed
> value?
> If it's intended to be read as "choose a random c " then you're correct
> that you do need to check $C_0 = H_e$. The $wt(e) = t$ check will almost
> surely fail but the IND-CCA2 security proof requires the PKE scheme to
> be perfect. No decryption failures are allowed no matter how
> negligible the probability.
> What about using a fixed value? It's actually better to set $c = 0$ when
> the decoder fails. In this case Step 3 sets $e = v$ and the check $C_0 =$
> H_e is trivially true so can be skipped. On the other hand the check
> $wt(e) = t$ always fails since otherwise $c = 0$ is the nearest codeword
> and would have been found by the decoder.
> Poorly written implementation notes are not just a problem for
> security. They can also be a problem for efficiency if they stop
> implementers from making changes that are genuinely safe.

I don't want to chime in here on the discussion about the quality of the implementation notes (I am clearly biased in my opinion that the quality of the implementation notes is impeccable ;)).

Just as a note: The decoding algorithm used in Step 2 probably per se computes either the unique codeword c if it exists or 'some c in $(F_2)^n$ '. So, in order to know if it computed the unique codeword or some other vector, one then will need to compute something like $C_0 = H_e$ anyway...

Thus, your observation is relevant only for a decoding algorithm that 'cheaply' detects its own failure; otherwise, the $(C_0=H_e)$ -check probably is an efficient solution for this detection.

And maybe yet another statement: I claim that for functional correctness, an implementer of an algorithm from a specification is expected to ensure that his or her implementation gives the exact same output for all possible inputs as the specified algorithm. Thus, it is (functionally) alright to leave out steps in the algorithm that do not alter the result (e.g., some unnecessary (CO=He)-check). (Caution is required as usual if side-channel security comes into play.)

Basically what I am saying is that in an implementation, the CO = He will likely be required and efficient and if it isn't required you are allowed to leave it out.

Ruben