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 analsysis 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.

Kirk

From: Kirk Fleming <kpfleming@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 >=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.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.

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->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.yp.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.yp.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 >=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 bottle-necked 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->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 eveyone 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
--

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.yp.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 Theta 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.

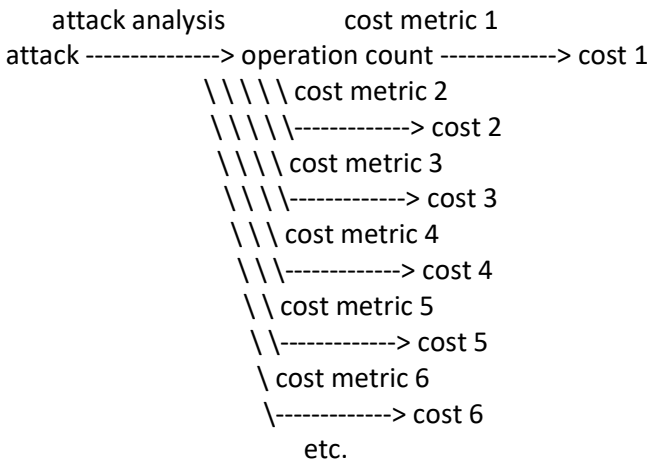
This was already answered: "The multipliers in an Intel CPU core carry out millions of bit operations in the same time that it takes the core to retrieve data from (say) 6GB of RAM, never mind RAM contention across cores."

> 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,...,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 earier 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 eveyone 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...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 (C0,C1).
2. Set b:=1.
3. Extract s and Gamma' from the private key.
4. Compute e := Decode(C0,Gamma').
   If e = Fail set e:=s and b:=0.
5. Compute C'1 := H(2,e).
6. If C'1 != C1 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 <-- 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 := Decode(C0,Gamma').
> If e = Fail set e:=0 and b:=0.

Let C := (C0,C1) be a ciphertext chosen so that C0 is not a codeword and C1 := H(2,0).

If decoding succeeds in Step 4' then e := Decode(C0,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 (C0,C1).
> 2. Set b:=1.
> 3. Extract s and Gamma' from the private key.
> 4. Compute e := Decode(C0,Gamma').
> If e = Fail set e:=s and b:=0.
> 5. Compute C'1 := H(2,e).
> 6. If C'1 != C1 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 <-- 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 := Decode(C0,Gamma').
   If e = Fail set e:=s and b:=0.
5. Compute C'1 := H(2,e).
6. If C'1 != C1 set e:=s and b:=0.

"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."

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".

(Notably, both "e := ..." 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...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.)

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 imples 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 := Decode(C0,Gamma').
> If e = Fail set e:=s and b:=0.
> 5. Compute C'1 := H(2,e).
> 6. If C'1 != C1 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←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...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←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←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 := Decode(C0,Gamma').
> > If e = Fail set e:=s and b:=0.
> > 5. Compute C'1 := H(2,e).
> > 6. If C'1 != C1 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←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 := Decode(C0,Gamma').
> > > If e = Fail set e:=s and b:=0.
> > > 5. Compute C'1 := H(2,e).
> > > 6. If C'1 != C1 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@shiftleft.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, 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 <- something else would indeed leave the decapsulation result unchanged, because when e = bot, since Decrypt is perfectly correct, there is no value of "something else" such that Encrypt(pk, 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 <- s, if indeed any value would suffice. But this isn't a valid objection either, because within that branch e=bot. Even if Encrypt(pk, 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 imples 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 C1' !=
> C1 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' != 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←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."

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 != 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 != 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 <kpfleming@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^ that are recovered from Gamma' 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

--

From: Kirk Fleming <kpfleming@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^ that are recovered from Gamma' 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,...,0) by apppending k zeros.

2. Find the unique codeword c in the Goppa code defined by Gamma' that is at distance <= 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 C0 = 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 <kpfleming@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 C0 = 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

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 C0 = 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.

OK, the question why the C0 = He 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 C0 = 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.
>
> OK, the question why the C0 = He 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 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.

I 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 C0 = He 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 C0 = He 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 C0 = He. The wt(e) = t check will amost 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 C0 = He 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 C0 = He
> 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 C0 = He. The wt(e) = t check will amost
> 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 C0 =
> He 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 C0 = He anyway...

Thus, your observation is relevant only for a decoding algorithm that 'cheaply' detects its own failure; otherwise, the (C0=He)-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 (C0=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 C0 = He will likely be required and efficient and if it isn't required you are allowed to leave it out.

Ruben

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Friday, May 21, 2021 12:55 AM
To: Kirk Fleming
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Was there ever any response (from the Classic McEliece team or otherwise) to this comment from 30 Dec 2020? More specifically:

On Wed, Dec 30, 2020 at 7:49 PM Kirk Fleming <kpfleming@mail.com> wrote:

> 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.

Does anyone know what effect this has on security?

> 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.

Has there since been any such security analysis or justification provided for the proposed parameter sets?

Sincerely yours in cryptography,
Chris

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Friday, May 21, 2021 5:09 AM
To: Christopher J Peikert; Kirk Fleming
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

On 5/21/21 6:54 AM, Christopher J Peikert wrote:
> On Wed, Dec 30, 2020 at 7:49 PM Kirk Fleming <kpfleming@mail.com> wrote:
>
>> 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.
>>
>
> Does anyone know what effect this has on security?

According to the analysis in the quoted paper [Bernstein, Lange and Peters. 2008], the difference in the security level when going from 121 errors to 119 errors while keeping the other parameters unchanged is about 4.

Ruben
(speaking for myself)

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Saturday, May 22, 2021 4:38 PM
To: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: CRYSTALS-KYBER

For the record, here is a direct response to the following framework and case study of the ciphertext-compression methods used by some NIST finalists/alternates, versus Ding's 2012 patent.

On Fri, Jan 1, 2021 at 7:19 AM D. J. Bernstein <djb@cr.yp.to> wrote:

> Main goal of this message: Unify notation and terminology for the
> systems under discussion, to provide a framework for evaluating claims
> regarding what's the same, what's different, etc. This message is
> organized around a case study, namely a puzzling claim of a dividing
> line between Ding's patented work and newer compressed-LPR proposals.

The evidence, viewed within the proposed framework, clearly shows that there is nothing at all puzzling about the claimed dividing line.

The finalist/alternate NIST PQC proposals use a compression technique that by mid-2011 -- well before Ding's 2012 patent application -- was widely known and used across several (Ring-/Module-)LWE/LWR cryptographic constructions.

This prior art describes the technique's general applicability to the broad family of such cryptographic systems. Moreover, it _explicitly_ describes the technique's applicability to "structured" lattice systems defined over polynomial rings for efficiency, including "LPR ciphertexts" (which are functionally equivalent to "Ring-LWE samples," up to the added plaintext/encapsulated key).

Ding's patent describes a different technique that, in some contexts, can yield more compression than the prior-art techniques can. The mathematical, algorithmic, and functional distinctions are easy to see from the framework's own examples, as we will show next.

> Vadim Lyubashevksy writes (email dated 2 Dec 2020 20:47:11 +0100):
> > No version of the LPR paper presents "reconciliation".
>
> Let's say Alice sends A = aG+e, and Bob sends B = Gb+d, where lowercase
> letters are small quantities. Alice computes aB = aGb+ad. Bob computes
> Ab = aGb+eb. Can we agree that "noisy DH" is a good name for this?
>
> At this point Alice and Bob have computed something that's _similar_ but
> not _identical_. People who say "reconciliation" in the noisy-DH context
> are referring to an extra step of Alice and Bob computing the _same_
> secret with the help of further communication.

[NB: the term "reconciliation" does not have a consensus precise definition. Some people use it to refer to the general goal of getting Alice and Bob to compute exactly the same value. Some people use it to refer to a specific method of accomplishing this, like the one in Ding's patent, and use different words for other methods. The choice of word doesn't matter for the substance of the facts and analysis here. I'll refrain from using the term wherever possible to avoid any ambiguity/confusion.]

The normal reconciliation pattern is that Bob sends C = Ab+M+c, where M is in a sufficiently limited set to allow error correction: for example, each position in M is restricted to the set {0,floor(q/2)}. Alice now finds C-aB = M+c+eb-ad, and corrects errors to find M. Bob also finds M.

Example 1: Take the variables in R_q = F_q[x]/(x^n+1) where n is a power of 2, and take C as Ab+M+c.

This cryptosystem, with some further restrictions on distributions etc., appeared in the revised version of the LPR paper (not with the name "reconciliation", I agree) and in talk slides as early as April 2010.
This is pretty much always what people mean by "the LPR cryptosystem", modulo the question of which number fields are considered; I'm focusing here on the commonly used case of power-of-2 cyclotomic fields.

Example 2: Special case of Example 1, with the extra requirement of C being in a more restricted set, so that C can be sent in less space.

This is what I call "compressed LPR". The LPR cryptosystem was in the revised version of the LPR paper, but compressed LPR wasn't; see below for further discussion of the history.

Example 3: Special case of Example 2, where C is a deterministic function of Ab+M: e.g., C is obtained by rounding Ab+M to a more restricted set.

Examples 2 and 3 are how Kyber and SABER work*, respectively. Note that M has no dependence on A, b, or c, and rounding down to a single bit per polynomial coefficient would not work -- decryption would be incorrect. (E.g., Kyber512 compresses the integer coefficients of C to 10 bits each, and can't go much further.)

The methods and distinguishing features of Example 2 (i.e., rounding C to a more restricted set) and Example 3 (i.e., replace small error c with deterministic rounding) are well represented in the prior art from 2009-2011, as shown by the references at the end of this message.

(For reference, Example 2 belongs to a class known in the literature as "rounded (Ring-/Module-)LWE", and Example 3 belongs to the class "(Ring-/Module-)Learning With Rounding.")

Importantly, Ding's patent doesn't claim (or even describe) the compression methods of Examples 2/3, and for good reason -- one can't patent prior art. So, the patent's limited claims don't cover Example 2/3. At this point one can already conclude that Kyber/SABER's compression is non-infringing, but let's continue.

*(but with the variables being vectors or matrices over R_q; this distinction isn't important for the analysis of the compression techniques, and in any case, this setting is also explicitly covered by the prior art.)

Example 4: Special case of Example 3, where M is the output of rounding -Ab to have each position in {0,floor(q/2)}. This puts each position of Ab+M between about -q/4 and q/4.

For definiteness let's say that C is then obtained by rounding each position of Ab+M to {-ceil(q/8),ceil(q/8)}. The difference c = C-Ab-M is then between about -q/4 and about q/4.

> Notice that each position of the shared secret M is now communicating
> one bit of information about Ab, namely whether that position of Ab is
> closer to 0 or to q/2.

Example 4 closely corresponds to what Ding's patent describes. Note the following features that distinguish it from Examples 2 and 3: M now depends on Ab (perhaps even deterministically), and every polynomial coefficient can be compressed to a *single* bit (as opposed to the necessary several bits in Example 2/3).

The patent must be limited to these specific methods and (potential) advantages; it does not apply to any form of "compressed LPR" in general.

No NIST finalist/alternate has any of these distinguishing features (again, they use only the compression method from Example 2/3). So, Example 4's/Ding's compression method is irrelevant to the NIST proposals.

> Example 5: ...
>
> (Readers coming at all this from an FHE background will recognize the
> difference between Example 5 and Example 4 as being analogous to the
> difference between 2009 DGHV and 2000 Cohen/2003 Regev. But pointing to
> inefficient prior art won't win a patent case; see below.)

It's true that long before the patent, Examples 4 and 5 were understood to have a distinction without a difference. But we shouldn't limit our view to literature that doesn't use polynomial rings for efficiency. For example, the prior art BGV'11 cited below explicitly describes Example 2's compression on "LPR ciphertexts" (a.k.a., Ring-/Module-LWE samples).

> All of the compressed-LPR examples, everything starting from Example 2,
> also have the following feature:
>
> (S) Bob squeezes C into less space than an R_q element.

Based on the abundant prior art, any person of ordinary skill working in the area (or even someone just following it closely) would have easily seen that feature (S) -- obtained using the methods of Examples 2 and 3 -- is generally applicable to any combination of A, B, C, D, P (and even X, Y, Z to be named later) in the broad family of (Ring-/Module-)LWE/LWR cryptosystems.

Indeed, the prior art has multiple explicit examples of various combinations (including most and possibly all of A, B, C, D, P), along with remarks that the techniques apply generally to all manner of "LWE samples" (including LPR ciphertexts). There would be no novelty or inventive step involved in applying them to yet-another-combination; they were by then a widely used tool in the lattice-crypto toolbox.

> The plaintiff's lawyer will pull out one expert
> witness after another saying that efficiency improvements now
> claimed in retrospect to be obvious weren't obvious at the time;
> and will then pull out the big gun, 2014 Peikert.

Like Ding's patent, 2014 Peikert falls under Example 4. None of the NIST proposals use any of those methods or have any of their distinguishing features, so 2014 Peikert is not even a starter pistol here.

> How is the
> defendant's lawyer supposed to argue that saving space compared to
> LPR was obvious in 2012 to people of ordinary skill in the art,

> given that 2014 Peikert claimed that saving space compared to LPR
> was new, the result of an "innovation" in 2014 Peikert?

This is not "given," it is a blatant misrepresentation of 2014 Peikert's claims.

It did not claim that "saving [any] space compared to uncompressed LPR" was novel -- indeed, saving a small amount of space was already explicit from the cited prior art.

It did claim (among other things) a particular, rigorous method for saving *almost 2x space* in an LPR-like key exchange, by working similarly to Example 4. Again, this class of technique is unused by, and irrelevant to, the NIST proposals.

> * The non-infringement argument. This time it's the defendant's
> lawyer trying to argue that there's a "substantial" difference
> between the patent claims and what the defendant is doing.

A competent non-infringement argument ought to win easily. To summarize:

The claims in Ding's patent are limited to (something very close to) Example 4, and definitely don't touch Example 2/3. But the latter are all that Kyber/SABER use for compression, and they are well represented by prior art. Frankly, this argument should be a slam dunk. But if you want more:

Another possible substantial difference is that SABER additionally compresses B (the ciphertext "preamble") by a decent amount; e.g., level-3 SABER compresses each 13-bit integer to 10 bits, a 23% savings. I don't see Ding's patent as describing any compression of B.

Another possible substantial difference is Example 2/3's ability to choose M freely (or have it be specified as an input) -- which Kyber/SABER do -- versus Example 4's restriction that M must depend on Ab. I understand this to be the "public-key encryption" versus "key exchange" distinction that Vadim Lyubashevsky has brought up in prior messages. To turn Example 4 into a PKE, one would need to somehow combine M with the intended message and send the result -- but this costs more communication, undoing some of the benefits of the high compression.

In addition, there may be other limitations of the patent or substantial differences leading to good non-infringement arguments (whether related to compression or not), and good attacks on the patent's validity in the first place.

Sincerely yours in cryptography,
Chris

Here is a (likely incomplete) list of prior art describing the compression techniques described in Example 2/3. These works explicitly construct KEMs, public-key encryption, and other lattice-based cryptosystems, but the techniques' general applicability to the entire LWE/LWR family is a recurring explicit theme. This list originally appeared in a previous message of mine [link].

* 2009 https://web.eecs.umich.edu/~cpeikert/pubs/svpcrypto.pdf

  -- Section 4.2: "When using a large value of q... the efficiency of the prior schemes is suboptimal... Fortunately, it is possible to improve their efficiency (without sacrificing correctness) by discretizing the LWE distribution more ‘coarsely’ using a relatively small modulus q'..." The subsequent definition of KEM.Encaps uses rounding to compress the ciphertext, just as described in the text.

The above text mentions only LWE, because it predates the publication of the more "structured" and efficient Ring-LWE and Module-LWE (cf. Kyber, SABER, NTRU LPRime). But upon their introduction in 2010 and 2011, a skilled person would

easily recognize that rounding works just as well for those too. Indeed, the following prior art does just that, often explicitly for compression purposes:

* 2011 https://eprint.iacr.org/2011/401 and https://web.eecs.umich.edu/~cpeikert/pubs/slides-prf.pdf introducing (Ring-)LWR (cf. SABER):

-- "Our derandomization technique for LWE is very simple: instead of adding a small random error term to each inner product, we just deterministically round it to the nearest element of a sufficiently 'coarse' public subset of p << q well-separated values..." and

-- "In the ring setting, the derandomization technique and hardness proof based on ring-LWE all go through without difficulty as well" and

-- "We believe that this technique should be useful in many other settings" and "LWE: deterministic errors also gives more practical PRGs, GGM-type PRFs, encryption, ..."

* 2011 https://eprint.iacr.org/2011/344 :

-- "Our dimension-modulus reduction idea enables us to take a ciphertext with parameters (n, log q) as above, and convert it into a ciphertext of the same message, but with parameters (k, log p) which are much smaller than (n, log q)..." and

-- "the underlying intuition is that Z_p can 'approximate' Z_q by simple scaling, up to a small error..." and

-- "As a nice byproduct of this technique, the ciphertexts of the resulting fully homomorphic scheme become very short!"

* 2011 https://eprint.iacr.org/2011/277 : more on modulus reduction (a.k.a. rounding), this time explicitly for Ring- and Module-LWE:

-- "This is an abstract scheme that can be instantiated with either LWE or Ring LWE..." and

-- "The transformation from c to c' involves simply scaling by (p/q) and rounding..." and

-- "while they use modulus switching in 'one shot' to obtain a small ciphertext (to which they then apply Gentry's bootstrapping procedure), we will use it (iteratively, gradually) to keep the noise level essentially constant, while stingily sacrificing modulus size..." and

-- defining what's now known as Module-LWE: "LWE is simply GLWE instantiated with d = 1. RLWE is GLWE instantiated with n = 1. Interestingly, as far as we know, instances of GLWE between these extremes have not been explored. One would suspect that GLWE is hard for any (n, d) such that ... "

From: Kirk Fleming <kpfleming@mail.com>
Sent: Monday, June 21, 2021 7:17 PM
To: Ruben Niederhagen
Cc: Christopher J Peikert; pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

In the Classic McEliece presentation at the recent NIST post-quantum conference Tanja Lange gave estimates for the security of mceliece-1024-50 and mceliece-1223-23 using the Markov chain analysis from BLP2010. She failed to do the same for the parameter sets proposed in the Round 3 submission.

For completeness, a limited search using the isdfq code by Peters (github.com/christianepeters/isdfq) gave:

Parameter Set	Bit Ops
mceliece-3488-064	2^145.2
mceliece-4608-096	2^187.0
mceliece-6688-128	2^261.9
mceliece-6960-119	2^262.3
mceliece-8192-128	2^297.1

The 2^262.3 estimate for mceliece-6960-119 corresponds to the 4-bit reduction in security cited by Ruben.

Kirk

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Monday, June 21, 2021 10:42 PM
To: Kirk Fleming
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

For the record: in nearly six months there has been no response from the Classic McEliece team, nor update to the specification, to this:

On Wed, Dec 30, 2020 at 7:49 PM Kirk Fleming <kpfleming@mail.com> wrote:

> 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.

A few hours ago, Dan Bernstein (one of the Classic McEliece submitters) wrote:

"Security claims regarding NISTPQC submissions have to be stated clearly for public review. When a reviewer disproves a security claim, the claim has to be withdrawn, and the withdrawal has to properly credit the disproof. Any other approach disincentivizes security review."

Shortly thereafter, Kirk Fleming wrote: "a limited search using the isdfq code by Peters (github.com/christianepeters/isdfq)" gave an estimate for mceliece-6960-119 of 2^262.3 bit operations. This is less than the 2^266.94 bit operations claimed in the submission.

Is this a "disproof" of the submission's security claim? If the code's output represents an actual attack, then yes, it is a disproof.

In any case, the submission certainly contains a disproved *description of* BLP08's clear security claim. By the same rationale from the above quote (do not disincentivize security review), the same remedy is required: withdraw the claim and properly credit the disproof.

Sincerely yours in cryptography,
Chris

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Tuesday, June 22, 2021 2:10 AM
To: Kirk Fleming
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Hi Kirk!

On 6/22/21 1:16 AM, Kirk Fleming wrote:
> In the Classic McEliece presentation at the recent NIST post-quantum
> conference Tanja Lange gave estimates for the security of
> mceliece-1024-50 and mceliece-1223-23 using the Markov chain analysis
> from BLP2010. She failed to do the same for the parameter sets
> proposed in the Round 3 submission.

Tanja did, however, on the very same slide link to the paper "A Finite Regime Analysis of Information Set Decoding Algorithms" by Baldi, Barenghi, Chiaraluce, Pelosi, and Santini, advertising that it provides "bit operations & memory for all Classic McEliece parameters".

> For completeness, a limited search using the isdfq code by Peters
> (github.com/christianepeters/isdfq) gave:
> Parameter Set Bit Ops
> mceliece-3488-064 2^145.2
> mceliece-4608-096 2^187.0
> mceliece-6688-128 2^261.9
> mceliece-6960-119 2^262.3
> mceliece-8192-128 2^297.1
> The 2^262.3 estimate for mceliece-6960-119 corresponds to the 4-bit
> reduction in security cited by Ruben.

Thank you for doing the math!

I am happy that your calculations fit well to the proposed security levels of the Classic McEliece submission.

Ruben
(speaking for myself)

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Tuesday, June 22, 2021 5:53 AM
To: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

On 6/22/21 4:42 AM, Christopher J Peikert wrote:
> A few hours ago, Dan Bernstein (one of the Classic McEliece
> submitters) wrote
> <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Yx0wZuZP6ag/m
> /rjmo2mYeCQAJ>
> :
>
> "Security claims regarding NISTPQC submissions have to be stated
> clearly for public review. When a reviewer disproves a security claim,
> the claim has to be withdrawn, and the withdrawal has to properly
> credit the disproof. Any other approach disincentivizes security review."

I fail to see a point in the fact that Dan Bernstein is one of the Classic McEliece submitters (as am I); quite clearly his individual statements are not a joint position of the Classic McEliece submitters.

> Shortly thereafter, Kirk Fleming wrote
> <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/EiwxGnfQgec/m/PyJCsL4iCQAJ>:
> "a limited search using the isdfq code by Peters (
> github.com/christianepeters/isdfq)" gave an estimate for
> mceliece-6960-119 of 2^262.3 bit operations. This is less than the
> 2^266.94 bit operations claimed in the submission.
>
> Is this a "disproof" of the submission's security claim? If the code's
> output represents an actual attack, then yes, it is a disproof.
>
> In any case, the submission certainly contains a disproved
> *description of* BLP08's clear security claim. By the same rationale
> from the above quote (do not disincentivize security review), the same remedy is required:
> withdraw the claim and properly credit the disproof.

Just for clarification and to avoid misunderstandings by readers of this thread: The Classic McEliece submission proposes the mceliece-6960-119 parameter set (13,6960,119) for Category 5. The estimated cost of 2^262.3 bit operations according to this specific reference as computed by Kirk Fleming are well within Category 5 - probably without any dispute also by Christopher Peikert.

Chris refers to a different (though interesting) discussion in the thread "Looseness, security risks, and LWR vs. LWE".

Best regards
Ruben (speaking for myself)

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Tuesday, June 22, 2021 6:14 AM
To: Ruben Niederhagen
Cc: pqc-comments; pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

On Tue, Jun 22, 2021 at 5:52 AM Ruben Niederhagen <ruben@polycephaly.org> wrote:

> On 6/22/21 4:42 AM, Christopher J Peikert wrote:
> > A few hours ago, Dan Bernstein (one of the Classic McEliece submitters)
> > wrote
> > <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Yx0wZuZP6ag/m/rjmo2mYeCQAJ>
> > :
> >
> > "Security claims regarding NISTPQC submissions have to be stated clearly
> > for public review. When a reviewer disproves a security claim, the claim
> > has to be withdrawn, and the withdrawal has to properly credit the
> > disproof. Any other approach disincentivizes security review."
>
> I fail to see a point in the fact that Dan Bernstein is one of the
> Classic McEliece submitters (as am I); quite clearly his individual
> statements are not a joint position of the Classic McEliece submitters.

Fair enough. My message can be taken as a request to know what *is* the joint position of the Classic McEliece team on the highlighted discrepancy, which has been pending for almost six months, and seems easy to resolve. Moreover, there are several other substantive but unaddressed points made by Kirk Fleming in this thread that deserve some response.

> Just for clarification and to avoid misunderstandings by readers of this
> thread: The Classic McEliece submission proposes the mceliece-6960-119
> parameter set (13,6960,119) for Category 5. The estimated cost of
> 2^262.3 bit operations according to this specific reference as computed
> by Kirk Fleming are well within Category 5 - probably without any
> dispute also by Christopher Peikert.

Yes, I don't dispute that this is within Category 5. My prior message concerned the more precise security claims.

Sincerely yours in cryptography,
Chris

From: Kirk Fleming <kpfleming@mail.com>
Sent: Tuesday, June 22, 2021 8:19 PM
To: Ruben Niederhagen
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Hi Ruben,

Ruben Niederhagen wrote:
> Thank you for doing the math!

I did very little. Thank Christiane Peters for writing the code!

> I am happy that your calculations fit well to the proposed security
> levels of the Classic McEliece submission.

I disagree. I still believe mceliece-4608-096 is less secure than it needs to be for Category 3.

This should not be a surprise when you realise the parameter sets were chosen based on the size of the public key. The loose rule of thumb is that increasing the size of the public key by a factor of 4 doubles the security so if a 2^18 byte public key is equivalent to AES-128 then you might reasonably expect a 2^20 byte public key to be equivalent to AES-256. However, doubling the size of the public key only increases the security by a factor of Sqrt(2) so you should only expect a 2^19 byte public key to be equivalent to AES-181.

This is not a proof, of course, but it does suggest why the estimates from the Markov chain analysis are lower for mceliece-4608-096, relatively speaking, than the other parameter sets.

Kirk

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Wednesday, June 23, 2021 7:25 AM
To: Kirk Fleming
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Hi Kirk!

On 6/23/21 2:18 AM, Kirk Fleming wrote:
> Ruben Niederhagen wrote:
>> I am happy that your calculations fit well to the proposed security
>> levels of the Classic McEliece submission.
>
> I disagree. I still believe mceliece-4608-096 is less secure than it
> needs to be for Category 3.
>
> This should not be a surprise when you realise the parameter sets were
> chosen based on the size of the public key. The loose rule of thumb is
> that increasing the size of the public key by a factor of 4 doubles
> the security so if a 2^18 byte public key is equivalent to
> AES-128 then you might reasonably expect a 2^20 byte public key to be
> equivalent to AES-256.
How do you get to this rule of thumb? Be aware that there are several parameters in code-based crypto that are contributing to the key size on the one hand and to the attack cost on the other hand; there are many parameter sets with a public key size of, e.g., around 512 kB but varying attack costs - as there are many parameter sets for a cost of around 2^192 operations but with varying key sizes.

For example, the (not proposed) parameter sets (m=13, n=4750, t=90) and (m=13, n=7900, t=44) have a very similar public key size of about 512 kB
- but significantly different attack costs.

> However, doubling the size of the public key only increases the
> security by a factor of Sqrt(2) so you should only expect a 2^19 byte
> public key to be equivalent to AES-181. This is not a proof, of
> course, but it does suggest why the estimates from the Markov chain
> analysis are lower for mceliece-4608-096, relatively speaking, than
> the other parameter sets.
Well, applying well established formulas to assess the cost of an attack for specific parameter sets (as you did using Christiane's script) sounds good enough to me; I hope you are not suggesting that the cost analysis itself is flawed.

Picking parameter sets according to key sizes is not uncommon; recommended RSA key sizes tend to be a power of two or at least a multiple of 1024 bit. Also common security levels themselves like 128, 192, and 256 (also 160, 224, 320, ...) are somewhat arbitrary and probably indeed were motivated by the corresponding key sizes of AES, ECC, etc. However, you are of course entitled to your own opinion on whether or not parameter sets (as long as they provide the desired security level) should be picked targeting a specific key size.

Ruben
(speaking for myself)

From: Kirk Fleming <kpfleming@mail.com>
Sent: Wednesday, June 23, 2021 8:17 AM
To: Ruben Niederhagen
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Hi Ruben,

Ruben Niederhagen wrote:
>>> I am happy that your calculations fit well to the proposed
>>> security levels of the Classic McEliece submission.

Kirk Fleming wrote:
>> I disagree. I still believe mceliece-4608-096 is less secure than it
>> needs to be for Category 3.

>> This should not be a surprise when you realise the parameter sets
>> were chosen based on the size of the public key. The loose rule of
>> thumb is that increasing the size of the public key by a factor of 4
>> doubles the security so if a 2^18 byte public key is equivalent to
>> AES-128 then you might reasonably expect a 2^20 byte public key to be
>> equivalent to AES-256.

Ruben replied:
> How do you get to this rule of thumb?

See for example Section 8.1 of the Classic McEliece submission:

"Security level 2^b thus uses key size (c_0+o(1))b^2 (lg b)^2
where c0 = R/(1−R)(lg(1−R))^2 ."

If we ignore log factors, when R is fixed the size of the public key
is asymptotically quadratic in the security parameter.

> Be aware that there are several
> parameters in code-based crypto that are contributing to the key size on
> the one hand and to the attack cost on the other hand; there are many
> parameter sets with a public key size of, e.g., around 512 kB but
> varying attack costs - as there are many parameter sets for a cost of
> around 2^192 operations but with varying key sizes.

> For example, the (not proposed) parameter sets (m=13, n=4750, t=90) and
> (m=13, n=7900, t=44) have a very similar public key size of about 512 kB
> - but significantly different attack costs.

Of course it is possible to choose parameters with similarly sized
public keys but varying security levels. However, the Classic McEliece
parameter sets were chosen to maximize security given an upper bound
on the public key size. This will tend to give parameter sets with similar
values for R.

>> However, doubling the size of the public key only increases the
>> security by a factor of Sqrt(2) so you should only expect a 2^19 byte
>> public key to be equivalent to AES-181. This is not a proof, of
>> course, but it does suggest why the estimates from the Markov chain
>> analysis are lower for mceliece-4608-096, relatively speaking, than
>> the other parameter sets.

> Well, applying well established formulas to assess the cost of an attack
> for specific parameter sets (as you did using Christiane's script)
> sounds good enough to me; I hope you are not suggesting that the cost
> analysis itself is flawed.

No, I'm not suggesting that the analysis is flawed. I'm suggesting that the analysis doesn't support the claimed security category for this parameter set.

> Picking parameter sets according to key sizes is not uncommon;
> recommended RSA key sizes tend to be a power of two or at least a
> multiple of 1024 bit. Also common security levels themselves like 128,
> 192, and 256 (also 160, 224, 320, ...) are somewhat arbitrary and
> probably indeed were motivated by the corresponding key sizes of AES,
> ECC, etc. However, you are of course entitled to your own opinion on
> whether or not parameter sets (as long as they provide the desired
> security level) should be picked targeting a specific key size.

I have no objection to choosing parameter sets that target specific key sizes provided that they also meet the claimed security categories.

Kirk

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Wednesday, June 23, 2021 10:17 AM
To: Kirk Fleming
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Hi Kirk!

On 6/23/21 2:16 PM, Kirk Fleming wrote:> Ruben replied:
>> How do you get to this rule of thumb?
>
> See for example Section 8.1 of the Classic McEliece submission: > "Security level 2^b thus uses key size (c_0+o(1))b^2 (lg b)^2
> where c0 = R/(1−R)(lg(1−R))^2 ."
> If we ignore log factors, when R is fixed the size of the public key
> is asymptotically quadratic in the security parameter.

Well - an asymptotic rule of thumb might be quite a bit off for concrete parameter sets - as illustrated by my example parameter sets (m=13, n=4750, t=90) and (m=13, n=7900, t=44).

>> Well, applying well established formulas to assess the cost of an
>> attack for specific parameter sets (as you did using Christiane's
>> script) sounds good enough to me; I hope you are not suggesting that
>> the cost analysis itself is flawed.
>
> No, I'm not suggesting that the analysis is flawed. I'm suggesting
> that the analysis doesn't support the claimed security category for
> this parameter set.

I am confused; the very attack costs that you reported in one of your earlier mails for the Category-3 parameter sets seems to support the claimed security category. Could you be more specific on why you consider this not to be the case?

I'd suggest to take this offline so we don't continue to 'spam' the mailing list; this discussion might be off-topic for many readers. Maybe we can just report back with the results of our offline discussion afterwards.

Ruben

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Wednesday, June 23, 2021 7:30 PM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

> "Security level 2^b thus uses key size (c_0+o(1))b^2 (lg b)^2
> where c0 = R/(1−R)(lg(1−R))^2 ."
> If we ignore log factors, when R is fixed the size of the public key
> is asymptotically quadratic in the security parameter.

The argument here, together with the context, is

(1) claiming that the 1MB parameters are cutting things close for 256,
(2) claiming that o(1) can be treated as 0,
(3) claiming that log factors can be ignored, giving a simple b^2,
(4) observing that sqrt(1/2) is 6% below sqrt(192/256), and
(5) concluding that the 1/2 MB parameters are 6% off for 192.

But if one _doesn't_ ignore log factors then the b^2 (lg b)^2 ratio between b=256 and b=192 is 1.98, which turns the 6% into under 1%. The argument never claims that #1 is cutting things _that_ close.

Bigger picture: #3 is pointless oversimplification; the submission already specifically disclaims #2; #1 is ill-defined, and, for people who understand what the literature says about the attacks, it's hard to imagine a realistic cost metric that would make #1 correct. In general, basing such a minor quantitative complaint on such a large pile of assumptions can simply be dismissed as too error-prone to consider.

The submission team officially requested last year that NIST "fully define the cost metric to be used for 'categories'" so that all submission teams can evaluate costs in this metric. NIST's failure to do this---despite the transparency rule in the call for proposals---is the primary reason that we all keep wasting time on arguments founded upon ill-defined claims. This is not how NISTPQC should work.

---Dan (speaking for myself)

From: 'David A. Cooper' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Thursday, June 24, 2021 11:06 AM
To: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

I do not believe that Kirk attempted to make the argument that you ascribe to him at all. I believe that he was very clear that he was not using what he described as a "loose rule of thumb" as the basis for questioning the security level of mceliece-4608-096.

Kirk provided a table of security estimates and suggested that the estimate for one of the parameter sets seemed to be lower than required for the claimed security level. I have not seen any response to these emails expressing disagreement with the security estimates from the table or attempting to explain why mceliece-4608-096 should be considered category 3 despite the security estimate of 2^187.0 bit ops.

David (speaking only from myself)

On 6/21/21 7:16 PM, Kirk Fleming wrote:

> For completeness, a limited search using the isdfq code
> by Peters (github.com/christianepeters/isdfq) gave:
>
> | Parameter Set | Bit Ops |
> |---|---|
> | mceliece-3488-064 | 2^145.2 |
> | mceliece-4608-096 | 2^187.0 |
> | mceliece-6688-128 | 2^261.9 |
> | mceliece-6960-119 | 2^262.3 |
> | mceliece-8192-128 | 2^297.1 |

On 6/22/21 8:18 PM, Kirk Fleming wrote:

> I still believe mceliece-4608-096 is less secure than it
> needs to be for Category 3.

On 6/23/21 7:30 PM, D. J. Bernstein wrote:

```
              "Security level 2^b thus uses key size (c_0+o(1))b^2 (lg b)^2
                where c0 = R/(1-R)(lg(1-R))^2 ."
            If we ignore log factors, when R is fixed the size of the public
            key
            is asymptotically quadratic in the security parameter.


        The argument here, together with the context, is

           (1) claiming that the 1MB parameters are cutting things close for 256,
           (2) claiming that o(1) can be treated as 0,
           (3) claiming that log factors can be ignored, giving a simple b^2,
           (4) observing that sqrt(1/2) is 6% below sqrt(192/256), and
           (5) concluding that the 1/2 MB parameters are 6% off for 192.

        But if one _doesn't_ ignore log factors then the b^2 (lg b)^2 ratio
        between b=256 and b=192 is 1.98, which turns the 6% into under 1%. The
        argument never claims that #1 is cutting things _that_ close.
```

Bigger picture: #3 is pointless oversimplification; the submission already specifically disclaims #2; #1 is ill-defined, and, for people who understand what the literature says about the attacks, it's hard to imagine a realistic cost metric that would make #1 correct. In general, basing such a minor quantitative complaint on such a large pile of assumptions can simply be dismissed as too error-prone to consider.

The submission team officially requested last year that NIST "fully define the cost metric to be used for 'categories'" so that all submission teams can evaluate costs in this metric. NIST's failure to do this---despite the transparency rule in the call for proposals---is the primary reason that we all keep wasting time on arguments founded upon ill-defined claims. This is not how NISTPQC should work.

---Dan (speaking for myself)

From: pqc-forum@list.nist.gov on behalf of Ruben Niederhagen <ruben@polycephaly.org>
Sent: Thursday, June 24, 2021 12:55 PM
To: Cooper, David A. (Fed); pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

On 6/24/21 5:06 PM, 'David A. Cooper' via pqc-forum wrote:
> Kirk provided a table of security estimates and suggested that the
> estimate for one of the parameter sets seemed to be lower than
> required for the claimed security level. I have not seen any response
> to these emails expressing disagreement with the security estimates
> from the table or attempting to explain why mceliece-4608-096 should
> be considered category 3 despite the security estimate of 2^187.0 bit ops.

Let me quote from page 46 of the Classic McEliece submission document:

"A closer look shows that the attack in [Bernstein, Lange and Peters.
2008] 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."

Hence, I am feeling very comfortable with the quoted cost of 2^187 bit operations for Category 3; the script that Kirk is using counts only bit operations as well and does not take memory cost into consideration.

Let me refer you again to the paper "A Finite Regime Analysis of Information Set Decoding Algorithms" by Baldi, Barenghi, Chiaraluce, Pelosi, and Santini, which covers several ISD attack variants and also reports memory cost. In that paper, the interesting algorithms for the Category 3 parameter set are the MMT and BJMM variants of ISD, which both come with significant memory cost.

As Dan mentioned, an official definition or general agreement on how to include the memory cost into the NIST security categories is still open.
I'd argue that, including memory cost, all security categories are well covered by the Classic McEliece submission.

Nevertheless, being a code-based scheme, Classic McEliece allows to choose parameter sets quite flexibly - so if eventually there was a definition or general agreement of how to handle memory cost to which the currently proposed parameters did not fit, then suitable parameter sets could be chosen quite easily.

Ruben
(speaking for myself)

--

From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Thursday, June 24, 2021 7:16 PM
To: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

Some procedural objections.

'David A. Cooper' (NIST) via pqc-forum writes:
> I have not seen any response to these emails expressing disagreement
> with the security estimates from the table or attempting to explain
> why mceliece-4608-096 should be considered category 3 despite the
> security estimate of 2^187.0 bit ops.

In fact, the Classic McEliece team filed an OFFICIAL COMMENT dated 19 Nov 2020 13:37:46 +0100 responding in detail to the first iteration of this claim regarding bit operations. Procedurally, it's not reasonable to have every subsequent minor variation of the argument forcing the team to prepare a new response. The big problem pointed out in the original response hasn't gone away.

If NIST didn't understand something in the team response, then NIST should have promptly asked for clarification. I understand that you're speaking for yourself, but doesn't NIST have a central issue tracker making it easy for NIST people to see what has been addressed already?

> I do not believe that Kirk attempted to make the argument that you
> ascribe to him at all.

Let's look at the facts. He

* quoted the "(c_0+o(1))b^2 (lg b)^2" asymptotic;
* explicitly ignored "log factors"---that's #3 in my list---and
* ignored the o(1)---that's #2 in my list.

He used this to justify previously claiming a quadratic "rule of thumb".
He used this "rule of thumb", in turn, to say

* "you should only expect a 2^19 byte public key to be equivalent to AES-181"---that's #4 and #5 in my list.

Here 181 arises as sqrt(1/2)*256. The logic here has a gaping hole when the 1MB key costs _much more_ to break than AES-256, so he instead

* claimed "a 2^20 byte public key to be equivalent to AES-256"---that's #1 in my list.

The starting point before this is actually a claim about 128, which is equivalent to the claim about 256 for anyone who believes the quadratic from #2 and #3; sqrt(2)*128 is the same 181.

So his argument for "you should only expect a 2^19 byte public key to be equivalent to AES-181" is the argument that I attributed to him. Please withdraw your statement to the contrary.

> I believe that he was very clear that he was not using what he
> described as a "loose rule of thumb" as the basis for questioning the
> security level of mceliece-4608-096.

Readers will understand his original 194.36 argument (from a script simplifying costs in one way), and his latest 187.0 argument (from a script simplifying costs in another way), to be claiming that 460896 is slightly less secure than AES-192. The big problem with this claim was already covered in the team message from last year.

Readers will _also_ understand his new argument that "you should only expect a 2^19 byte public key to be equivalent to AES-181" as claiming that 460896 is slightly less secure than AES-192. Within this argument,
#1 starts from the same big problem as the other argument, but #2 and #3 aren't shared by the other argument.

I quoted #3, correctly summarized the whole argument, and then pointed out how the inaccuracy of #3 destroys the argument (while also noting the problems with #1 and #2). I expected to see the argument withdrawn.
I didn't expect to have to contend with claims that the argument won't affect readers and claims that the argument wasn't even presented.

---Dan

From: Kirk Fleming <kpfleming@mail.com>
Sent: Friday, June 25, 2021 8:38 PM
To: Ruben Niederhagen
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Ruben wrote:
> I'd suggest to take this offline so we don't continue
> to 'spam' the mailing list; this discussion might be
> off-topic for many readers.

I was planning to agree but I now find myself being asked to withdraw claims that I don't believe I made based on objections that distort what I said.

Let's recap.

The Classic McEliece update included security estimates for two challenge problems but did not extend the analysis to the parameter sets proposed in the submission. I posted those estimates as they illustrate Ruben's 4-bit gap in the mceliece-6960-119 claim from the submission.

Ruben replied he was happy the estimates fit the claimed security categories. I disagreed. I don't believe the Category 3 claim for mceliece-4608-094 is supported by the estimates.

So far so good.

I then made the observation that the parameter selection criteria would not necessarily lead to a parameter set meeting to Category 3. The rough heuristic I used was that the public keys are quadratic in the size of the security parameter. I thought I made it clear this wasn't a rigorous argument. I certainly didn't use it to make any claims about the security of specific parameter sets. Heuristics like this can be useful guides for designers when choosing parameters but they are no substitute for proper analysis.

Am I being asked to withdraw the heuristic? I don't think it should be controversial. When challenged by Ruben I quoted a formula from the Classic McEliece submission as that seemed particularly apt. I could equally have quoted Philippe's rank metric talk from the NIST 2015 workshop. None of the objections about ignoring o(1) or log terms are relevant. I have never claimed that asymptotic results give meaningful security estimates for finite parameters.

The wider objection is that the 187.0 estimate given for mceliece-4608-094 doesn't include memory costs. This is true but the Classic McEliece team won't commit to a metric that does include memory costs. Fortunately, the NTRUPrime team do and it's now being used by some of the other submissions. I leave it as an exercise for the reader to work out if mceliece-4608-094 meets Category 3 in that metric.

Kirk

From: pqc-forum@list.nist.gov on behalf of Kirk Fleming <kpfleming@mail.com>
Sent: Saturday, June 26, 2021 11:12 AM
To: Kirk Fleming
Cc: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

I wrote:
> I don't believe the Category 3 claim for mceliece-4608-094
> is supported by the estimates.

> The wider objection is that the 187.0 estimate given for
> mceliece-4608-094 doesn't include memory costs.

> I leave it as an exercise for the reader to work out if
> mceliece-4608-094 meets Category 3 in that metric.

Those should all have been mceliece-4608-096.

Kirk

> **Sent:** Saturday, June 26, 2021 at 12:37 AM
> **From:** "Kirk Fleming" <kpfleming@mail.com>
> **To:** "Ruben Niederhagen" <ruben@polycephaly.org>
> **Cc:** "pqc-forum" <pqc-forum@list.nist.gov>, "pqc-comments" <pqc-comments@nist.gov>
> **Subject:** Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
>
> Ruben wrote:
> > I'd suggest to take this offline so we don't continue
> > to 'spam' the mailing list; this discussion might be
> > off-topic for many readers.
>
> I was planning to agree but I now find myself being asked
> to withdraw claims that I don't believe I made based on
> objections that distort what I said.
>
> Let's recap.
>
> The Classic McEliece update included security estimates for
> two challenge problems but did not extend the analysis to
> the parameter sets proposed in the submission. I posted
> those estimates as they illustrate Ruben's 4-bit gap in
> the mceliece-6960-119 claim from the submission.
>
> Ruben replied he was happy the estimates fit the claimed
> security categories. I disagreed. I don't believe the
> Category 3 claim for mceliece-4608-094 is supported by
> the estimates.
>
> So far so good.
>
> I then made the observation that the parameter selection
> criteria would not necessarily lead to a parameter set
> meeting to Category 3. The rough heuristic I used was
> that the public keys are quadratic in the size of the
> security parameter. I thought I made it clear this wasn't
> a rigorous argument. I certainly didn't use it to make
> any claims about the security of specific parameter sets.
> Heuristics like this can be useful guides for designers

when choosing parameters but they are no substitute for proper analysis.

Am I being asked to withdraw the heuristic? I don't think it should be controversial. When challenged by Ruben I quoted a formula from the Classic McEliece submission as that seemed particularly apt. I could equally have quoted Philippe's rank metric talk from the NIST 2015 workshop. None of the objections about ignoring o(1) or log terms are relevant. I have never claimed that asymptotic results give meaningful security estimates for finite parameters.

The wider objection is that the 187.0 estimate given for mceliece-4608-094 doesn't include memory costs. This is true but the Classic McEliece team won't commit to a metric that does include memory costs. Fortunately, the NTRUPrime team do and it's now being used by some of the other submissions. I leave it as an exercise for the reader to work out if mceliece-4608-094 meets Category 3 in that metric.

Kirk

From: pqc-forum@list.nist.gov on behalf of Blumenthal, Uri - 0553 - MITLL <uri@ll.mit.edu>
Sent: Saturday, June 26, 2021 10:08 PM
To: Kirk Fleming
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

> >> I'd suggest to take this offline so we don't continue
> >> to 'spam' the mailing list; this discussion might be
> >> off-topic for many readers.
> >
> >I was planning to agree but I now find myself being asked
> >to withdraw claims that I don't believe I made based on
> >objections that distort what I said.

Evaluation of the security claims of one of the PQC Round 3 finalists can hardly be off-topic for the PQC-forum readers.

Thus, I would prefer that this discussion stays open – aka, on the list rather than turning “private”.

From: Ruben Niederhagen <ruben@polycephaly.org>
Sent: Tuesday, June 29, 2021 9:22 AM
To: pqc-forum; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Statement by the Classic McEliece team:

The Classic McEliece submission document states on page 46:

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

However, Kirk Fleming brought to our attention that this quoted attack cost is for a similar parameter set with the same length, dimension, and degree - but for adding 121 errors during encryption and not 119 errors as in our case. We correct the corresponding sentence to:

> "[Bernstein, Lange, and Peters. 2008] reported that its attack uses
> 2^266.94 bit operations to break a related (13,6960,119) parameter
> set with 121 errors."

From: Freja Elbro <davdavsen@gmail.com>
Sent: Friday, August 13, 2021 10:37 AM
To: pqc-forum
Cc: D. J. Bernstein; pqc-forum; pqc-comments; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Dear Dan and others,

I know that you say it is a thankless job, but I am considering being Caroline the cryptoanalyst. I know quite a bit about the ISD algorithms, but I am a complete novice to different cost metrics, so I hope you can help by giving some direction here.

1) I did a Google search on Ray Perlner's slides, but did not find anything. Can you be more specific in your reference? Or give a link? Regardless of whether I choose to be Caroline the Cryptoanalyst, the slides sound mighty interesting.

2) Are there other resources on different cost metrics, which might be a good place to start?

3) Where do I find analyses of AES in different cost metrics? It only makes sense to do an analysis of McEliece in a specific metric if there exists an analysis of AES in that metric, so I need to know which metrics AES has been analysed in (and what the result was...)

4) You say that the attack is "bottlenecked by random access to a huge array". When analysing the complexity of the algorithm, is this different from just "calls to memory"? In other words, why can we not "just" count the number of operations and the number of accesses to memory? What makes this type of call to memory special?

I hope to hear from some of you, who have more experience with different cost metrics.

Best regards,
Freja Elbro

mandag den 7. december 2020 kl. 11.21.36 UTC+1 skrev D. J. Bernstein:

> 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.

From: Michael Hamburg <mike@shiftleft.org>
Sent: Friday, August 13, 2021 11:16 AM
To: Freja Elbro; pqc-forum
Cc: D. J. Bernstein; pqc-forum; pqc-comments
Subject: RE: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Hi Freja,

On the subject of RAM, the cost of a memory read or write depends on size of the memory being accessed. The signal indicating what to read, and the signal bringing the data back, must travel to and from the location where the data is stored, and this takes time and dissipates energy along the way. If memory were organized as a 3-dimensional structure, then this would take O(cbrt(n)) time and energy. But a large 3d structure of active electronics is difficult to power and cool, so a 2d model and a sqrt(n) cost model is probably more accurate, or perhaps somewhere in between. This holds even on the scale of individual CPUs: it's why they have multi-level caches. The exact costs are debatable, but the fact that they increase as some power of the memory being accesses is important for even rough estimation of real-world attack costs.

Note that if the attack is run on many parallel machines, and each machine accesses only its own RAM, then this is much cheaper even if the total amount of memory is the same, because they don't have to communicate as far. That is, each lookup is into a smaller, local RAM, so it's cheaper.

Taking a naïve attack and multiplying its cost by sqrt(memory) isn't optimal, of course. The cost can be balanced for a meet-in-the-middle attack, and thus for ISD: you need to trade more computation for fewer memory accesses into smaller memories. See eg the famous van Oorschot-Wiener paper [1]. This makes the attack significantly more expensive than it would be if memory accesses cost O(1).

Brute-force attacks on AES don't use very much memory, so they cost about the same in this model.

Cheers,

- Mike

[1] Parallel Collision Search with Cryptanalytic Applications. Paul van Oorschot and Michael Wiener, Journal of Cryptology, 1999. https://link.springer.com/content/pdf/10.1007/PL00003816.pdf

---

From: Freja Elbro
Sent: Friday, August 13, 2021 2:36 PM
To: pqc-forum
Cc: D. J. Bernstein; pqc-...@list.nist.gov; pqc-co...@nist.gov; pqc-co...@nist.gov
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Dear Dan and others,

I know that you say it is a thankless job, but I am considering being Caroline the cryptoanalyst. I know quite a bit about the ISD algorithms, but I am a complete novice to different cost metrics, so I hope you can help by giving some direction here.

1) I did a Google search on Ray Perlner's slides, but did not find anything. Can you be more specific in your reference? Or give a link? Regardless of whether I choose to be Caroline the Cryptoanalyst, the slides sound mighty interesting.

From: Perlner, Ray A. (Fed)
Sent: Friday, August 13, 2021 1:37 PM
To: Michael Hamburg; Freja Elbro; pqc-forum
Cc: D. J. Bernstein; pqc-forum; pqc-comments
Subject: RE: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Hi Freja and Mike

Just briefly adding. I think the slides of mine Dan is referring to are the ones linked in my August 17 email here: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/GR3plWpSV4U/m/SChfTyNvBAAJ

Best,
Ray

From: Michael Hamburg <mike@shiftleft.org>
Sent: Friday, August 13, 2021 11:16 AM
To: Freja Elbro <davdavsen@gmail.com>; pqc-forum <pqc-forum@list.nist.gov>
Cc: D. J. Bernstein <djb@cr.yp.to>; pqc-forum <pqc-forum@list.nist.gov>; pqc-comments <pqc-comments@nist.gov>
Subject: RE: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Hi Freja,

On the subject of RAM, the cost of a memory read or write depends on size of the memory being accessed. The signal indicating what to read, and the signal bringing the data back, must travel to and from the location where the data is stored, and this takes time and dissipates energy along the way. If memory were organized as a 3-dimensional structure, then this would take O(cbrt(n)) time and energy. But a large 3d structure of active electronics is difficult to power and cool, so a 2d model and a sqrt(n) cost model is probably more accurate, or perhaps somewhere in between. This holds even on the scale of individual CPUs: it's why they have multi-level caches. The exact costs are debatable, but the fact that they increase as some power of the memory being accesses is important for even rough estimation of real-world attack costs.

Note that if the attack is run on many parallel machines, and each machine accesses only its own RAM, then this is much cheaper even if the total amount of memory is the same, because they don't have to communicate as far. That is, each lookup is into a smaller, local RAM, so it's cheaper.

Taking a naïve attack and multiplying its cost by sqrt(memory) isn't optimal, of course. The cost can be balanced for a meet-in-the-middle attack, and thus for ISD: you need to trade more computation for fewer memory accesses into smaller memories. See eg the famous van Oorschot-Wiener paper [1]. This makes the attack significantly more expensive than it would be if memory accesses cost O(1).

Brute-force attacks on AES don't use very much memory, so they cost about the same in this model.

Cheers,

- Mike

[1] Parallel Collision Search with Cryptanalytic Applications. Paul van Oorschot and Michael Wiener, Journal of Cryptology, 1999. https://link.springer.com/content/pdf/10.1007/PL00003816.pdf

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, August 17, 2021 6:36 AM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

Freja Elbro writes:
> 2) Are there other resources on different cost metrics, which might be
> a good place to start?

A great starting point is Knuth's Art of Computer Programming, which (1) defines clear metrics for the cost of computation, with (2) some efforts at realism, and then (3) precisely and (4) accurately analyzes the cost in these metrics of (5) many important algorithms. For _some_ algorithms Knuth also (6) compares different metrics, including (7) metrics that account for communication cost (see, e.g., the analysis of external sorting) and (8) parallel metrics (see, e.g., the analysis of sorting-network depth and the systolic array for multiplying integers).

The literature on computational complexity theory has more of #6, usually backed by #1 and #4, sometimes with #7, sometimes with #8. There are many pathways into this; the best starting point I've seen is https://eprints.illc.uva.nl/id/eprint/1008/1/CT-1989-02.text.pdf (well, I've read the printed version, but I assume the preprint is similar).
However, this side of the literature generally doesn't bother with #5, and deliberately sacrifices #3 in favor of simplicity, which also compromises #2 and puts serious limits on the value of #7 and #8.

https://maths-people.anu.edu.au/~brent/pd/rpb055.pdf is a typical example of a cloud of papers that aim to do better at #2 under the constraints of #1, #7, and #8. There's a split here between papers aiming for #3 for optimizing simple algorithms (see, e.g., the AES attack papers in https://2012.sharcs.org/accepted.html and various other papers under https://www.hyperelliptic.org/tanja/SHARCS/) and papers aiming to optimize more complicated algorithms at the expense of #3 (see, e.g., https://cr.yp.to/papers.html#batchnfs).

There are many papers reporting speeds of computations on existing CPUs and networks. This includes, e.g., papers reporting speed records for various cryptanalytic computations, scoring well on #5 and usually #3, sometimes with #7 and/or #8. The idea of using existing machines might sound great for #2 and #4, but for security analysis someone has to extrapolate to large-scale attacks. If data-access time isn't obviously an issue in academic speed records, then people will usually fall into the trap of assuming that it won't be an issue in much larger attacks, simply ignoring the research that has gone into analyzing #1+#2+#7+#8.

---Dan

From: Freja Elbro <davdavsen@gmail.com>
Sent: Tuesday, August 31, 2021 11:23 AM
To: pqc-forum
Cc: D. J. Bernstein; pqc-comments; pqc-comments; pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Dear NIST and others,

(and thank you so much, Mike, Ray and Dan for comprehensive answers!)

Sabine Pircher and I would like to do the security analysis of McEliece that NIST asked for at the PQCrypto2021: Count the concrete number of bit operations needed to perform an ISD attack. However, as Dan and others point out, it is not clear in which metric to do the analysis. Therefore, before we put in the work, we would like some feedback on whether it will be appreciated.

Several authors have already done the count that NIST asks for:

- Christiane Peters code: https://github.com/christianepeters/isdfq Uses the algorithm presented in https://eprint.iacr.org/2009/589 to estimate the security of McEliece. I am not sure how that algorithm compares to BJMM below, but expect it to be slower. Note that the script does not automatically optimize parameters, so to use the script, you have to manually search.
- "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. Provides security estimates for McEliece based on seven ISD-algorithms, but does not provide code to check their numbers. They also restrict BJMM to three levels, where two levels would be better in this case. See answer By Elena below.
- Answer by Elena Kirshanova https://crypto.stackexchange.com/questions/92074/number-of-bit-operations-required-for-information-set-decoding-attacks-on-code-b She estimates security of McEliece based on MMT and BJMM and provides source code, but she excludes polynomial factors.

What is missing? All the work above assumes that lookup takes constant time.

What we can do: Optimize BJMM parameters in the different models

- Random access to memory costs square root of memory size
- Random access to memory costs cubic root of memory size

To find the security level of McEliece in those two metrics.

What do you think? Would this be helpful to the discussion?

(Thank you Dan by the way for your very comprehensive answer! We read some of the material, but had to stop at some level, as was a bit deeper than we were hoping to go. We unfortunately do not have the time to understand models for how computers work, but have to rely on superficial models like the two above to be able to estimate running time.)

tirsdag den 17. august 2021 kl. 12.36.29 UTC+2 skrev D. J. Bernstein:

From: Perlner, Ray A. (Fed)
Sent: Wednesday, September 1, 2021 2:51 PM
To: Freja Elbro; pqc-forum
Cc: D. J. Bernstein; pqc-comments; pqc-comments; pqc-forum
Subject: RE: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Dear Freja,

From our (NIST) perspective, better estimates for the cost of BJMM would be very helpful. You mention models where random access memory costs are proportional to the square root and cube root of memory size, and these would be helpful to have, but we would also like some confirmation that we have accurate estimates of the cost of BJMM, when the cost of RAM queries is constant or logarithmic in the size of the memory. As you note, Elena's estimates indicate that BJMM depth 2 is best when determining cost in the RAM model. However, depth 3 seems likely to be better if the cost of memory access is taken into account, unless the number of random accesses to the memory is very small relative to the number of bit operations performed.

My guess is that a cube root memory cost is going to be the closest to the real world cost of an attack, but only because departures from reality in both directions will roughly cancel. It's also possible that the real cost of memory will be less than suggested by models that don't distinguish between true random access queries, and memory queries that can be made local using strategies similar to Van-Oorschot Weiner, and it might be helpful to investigate that possibility when looking at models that assume queries to large memories are significantly more expensive than queries to small memories.

In any event, thank you for volunteering to help us in our evaluations.

Ray Perlner

From: Edoardo Persichetti <epersichetti@fau.edu>
Sent: Friday, November 19, 2021 9:25 AM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 3 OFFICIAL COMMENT: Classic McEliece

Dear all

NIST has sent email to the Classic McEliece team requesting information about "the concrete security of Classic McEliece's parameter sets." This is a reply from the Classic McEliece team.

An optimized version of the latest BJMM/MMT variants in 2021 by Esser, May, and Zweydinger, running on real hardware, is barely faster than a 2008 Stern-type attack. See the round-3 Classic McEliece talk:

https://www.nist.gov/video/third-pqc-standardization-conference-session-vii-performance-and-candidate-updates
https://csrc.nist.gov/CSRC/media/Presentations/classic-mceliece-round-3-presentation/images-media/session-7-classic-mceliece-lange.pdf

As for cryptographic sizes, the state-of-the-art cost analyses are from a 2021 script

https://github.com/Crypto-TII/syndrome_decoding_estimator.git

by Bellini and Esser. See also https://eprint.iacr.org/2021/1243, the accompanying paper by Esser and Bellini. Running

```
import sys
from sd_estimator.estimator import sd_estimate_display
for n,w in (1024,50),(3488,64),(4608,96),(6688,128),(6960,119),(8192,128):
    m = 10
    while 2**m < n: m += 1
    k = n-w*m
    sd_estimate_display(n=n,k=k,w=w,memory_access=2)
    sys.stdout.flush()
```

outputs the tables shown below. The tables again show how small the improvements have been after Stern's 1989 attack. The smallest costs in the tables (not counting quantum attacks) are

2^158.6 for mceliece348864,
2^201.0 for mceliece460896,
2^278.2 for mceliece6688128,
2^279.2 for mceliece6960119, and
2^315.6 for mceliece8192128,

and footnote 11 of the paper indicates that the actual cost is higher.

Changing "memory_access" to "0" gives bit-operation counts that much more obviously underestimate actual costs, such as 2^65.1 bit operations for a Stern attack (using cost-0 access to megabytes of RAM) against McEliece's original parameters, 2^141 bit operations (using cost-0 access to >2^80 RAM) for mceliece348864, and 2^246 bit operations for mceliece6960119. (See Table 2 of https://eprint.iacr.org/2021/1243 for more of these bit-operation counts.) This matches what Section 8.2 of the Classic McEliece submission has always said regarding the originally proposed mceliece6960119 size:

> Subsequent ISD variants have reduced the number of bit operations considerably below 2^256. ... 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.

This 2021 script supersedes a 2019 paper by Baldi et al. The numbers in the 2019 paper for MMT in particular were much better than for BJMM (and Stern)---which is impossible since BJMM structurally includes MMT, so it was always clear that the MMT numbers had to be disregarded. NIST stated that BJMM "is widely considered to be an improvement over MMT" and asked "Is this analysis correct? If so, it seems like this could threaten not just some of the McEliece parameters, but also some of the parameters of the other code-based schemes." Kirshanova explained in

https://crypto.stackexchange.com/questions/92074/number-of-bit-operations-required-for-information-set-decoding-attacks-on-code-b

the "conclusion that BJMM algorithm is worse than MMT is incorrect because MMT is a special case of BJMM." An MMT/BJMM calculation error in the 2019 paper is pinpointed in https://eprint.iacr.org/2021/1243, Appendix A. The error would also have been caught by simple consistency checks against experiments.

The tables output by the script appear below.

```
================================================================================
Complexity estimation to solve the (1024,524,50) syndrome decoding problem
================================================================================
The following table states bit complexity estimates of the corresponding algorithms including an approximation of the polynomial factors inherent to the algorithm.
The quantum estimate gives a very optimistic estimation of the cost for a quantum aided attack with a circuit of limitted depth (should be understood as a lowerbound).
+---------------+--------------+---------+
|               |   estimate   | quantum |
+---------------+------+-------+---------+
| algorithm     | time | memory |    time |
+---------------+------+-------+---------+
| Prange        | 84.8 |  19.3 |   49.2  |
| Stern         | 73.2 |  26.1 |    --   |
| Dumer         | 74.1 |  26.2 |    --   |
```

Ball Collision	74.2	26.1	--
BJMM (MMT)	73.1	24.2	--
BJMM-pdw	71.8	21.1	--
May-Ozerov	70.6	21.1	--
Both-May	71.5	21.2	--

+----------+------+--------+---------+

==========

Complexity estimation to solve the (3488,2720,64) syndrome decoding problem

==========

The following table states bit complexity estimates of the corresponding algorithms including an approximation of the polynomial factors inherent to the algorithm.

The quantum estimate gives a very optimistic estimation of the cost for a quantum aided attack with a circuit of limitted depth (should be understood as a lowerbound).

	estimate		quantum
algorithm	time	memory	time
Prange	178.3	21.6	95.4
Stern	162.8	32.6	--
Dumer	163.2	32.6	--
Ball Collision	163.2	32.6	--
BJMM (MMT)	162.2	30.6	--
BJMM-pdw	160.3	26.8	--
May-Ozerov	158.6	24.6	--
Both-May	159.8	24.9	--

==========

Complexity estimation to solve the (4608,3360,96) syndrome decoding problem

==========

The following table states bit complexity estimates of the corresponding algorithms including an approximation of the polynomial factors inherent to the algorithm.

The quantum estimate gives a very optimistic estimation of the cost for a quantum aided attack with a circuit of limitted depth (should be understood as a lowerbound).

	estimate		quantum
algorithm	time	memory	time
Prange	222.1	22.7	140.3
Stern	205.3	33.6	--
Dumer	205.8	33.6	--
Ball Collision	205.8	33.6	--
BJMM (MMT)	204.8	31.6	--
BJMM-pdw	203.4	28.7	--
May-Ozerov	201.0	25.5	--
Both-May	202.2	26.5	--

==========

Complexity estimation to solve the (6688,5024,128) syndrome decoding problem

==========

The following table states bit complexity estimates of the corresponding algorithms including an approximation of the

polynomial factors inherent to the algorithm.
The quantum estimate gives a very optimistic estimation of the cost for a quantum aided attack with a circuit of limitted depth (should be understood as a lowerbound).

	estimate		quantum	
algorithm	time	memory	time	
Prange	301.2	23.6	219.9	
Stern	283.0	35.3	--	
Dumer	283.3	35.3	--	
Ball Collision	283.3	35.3	--	
BJMM (MMT)	282.3	33.3	--	
BJMM-pdw	280.4	29.4	--	
May-Ozerov	278.2	26.4	--	
Both-May	279.4	26.8	--	

==========

Complexity estimation to solve the (6960,5413,119) syndrome decoding problem

==========

The following table states bit complexity estimates of the corresponding algorithms including an approximation of the polynomial factors inherent to the algorithm.
The quantum estimate gives a very optimistic estimation of the cost for a quantum aided attack with a circuit of limitted depth (should be understood as a lowerbound).

	estimate		quantum	
algorithm	time	memory	time	
Prange	302.2	23.6	220.4	
Stern	283.9	35.6	--	
Dumer	284.3	35.6	--	
Ball Collision	284.3	35.6	--	
BJMM (MMT)	283.3	33.6	--	
BJMM-pdw	281.4	29.7	--	
May-Ozerov	279.2	26.6	--	
Both-May	280.3	27.0	--	

==========

Complexity estimation to solve the (8192,6528,128) syndrome decoding problem

==========

The following table states bit complexity estimates of the corresponding algorithms including an approximation of the polynomial factors inherent to the algorithm.
The quantum estimate gives a very optimistic estimation of the cost for a quantum aided attack with a circuit of limitted depth (should be understood as a lowerbound).

	estimate		quantum	
algorithm	time	memory	time	
Prange	339.5	23.9	257.6	
Stern	320.7	36.3	--	

Dumer	321.0	36.4	--
Ball Collision	321.0	36.3	--
BJMM (MMT)	320.0	34.4	--
BJMM-pdw	318.6	31.4	--
May-Ozerov	315.6	27.2	--
Both-May	317.0	27.6	--
+----------------+-------+--------+---------+

Best,
Edoardo

From: pqc-forum@list.nist.gov on behalf of Kirk Fleming <kpfleming@mail.com>
Sent: Saturday, November 20, 2021 10:15 PM
To: pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece

Edoardo wrote:
> The smallest costs in the tables (not counting
> quantum attacks) are
>
> 2^158.6 for mceliece348864,
> 2^201.0 for mceliece460896,
> 2^278.2 for mceliece6688128,
> 2^279.2 for mceliece6960119, and
> 2^315.6 for mceliece8192128,
>
> and footnote 11 of the paper indicates that the
> actual cost is higher.

Footnote 11 explains that the memory access cost uses the number of vectors instead of the number of bits. The difference between Sqrt(vectors) and Sqrt(memory)/2^5, the memory access cost used by the NTRUPrime submission, is at most 1.5 bits for May-Ozerov with these parameters.

In fact, the actual costs will be lower than the figures given. The memory access cost is applied as a fixed penalty but not all gates need a memory access and not all memory accesses need the full amount of memory.

This shows mceliece-4608-96 is 5 to 12 bits below Category 3 when attacking a single ciphertext. If you include the DOOM attack it's even worse.

Kirk
--

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Monday, June 6, 2022 12:21 PM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

This is an official comment from the Classic McEliece team regarding a recent talk at Eurocrypt on https://eprint.iacr.org/2021/1634, a paper reporting implementations of the MMT/BJMM algorithms using about 2^60.7 cycles in total on 256 AMD EPYC 7742 CPU cores to break miniature versions of McEliece with length 1284.

For comparison, when the 2021 Bellini--Esser estimator

https://github.com/Crypto-TII/syndrome_decoding_estimator.git

is told to ignore the cost of memory access ("memory_access=0"), it says
2^66.3 bit operations for 1989 Stern, followed by minor improvements from newer memory-intensive algorithms, such as 2^65.8 bit operations for BJMM and 2^64.6 bit operations for May--Ozerov. These numbers are larger than 2^60.7, but the units are also different: bit operations rather than CPU cycles.

These algorithms were already accounted for in the Classic McEliece submission in 2017, which, regarding the recommended 6960119 parameter set, wrote "ISD variants have reduced the number of bit operations considerably below 2^256". The submission also pointed out that this ignores the cost of memory access, and stated an expectation 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."

This expectation was then confirmed by subsequent analysis. For example, the Bellini--Esser estimator with "memory_access=2" reports cost 2^279.2, as noted in the Classic McEliece team email to pqc-forum in November 2021.

Anyone who wants such a high security level _without_ relying on the cost of memory access can instead use the 8192128 parameter set, where the Bellini--Esser estimator with "memory_access=0" says 2^275.6 bit operations for May--Ozerov, assigning zero cost to 2^194.4 memory. The exponent here is about 10% better than attacks from the 1980s (e.g., the estimator says 2^307.4 bit operations for Dumer), but the changes in costs are much smaller if one accounts for the real cost of memory.

The new paper and talk claim that some of the Classic McEliece parameter sets "fail to reach their security level by roughly 20 and 10 bit", that "McEliece Slightly Overestimates Security", etc., giving the impression of challenging the Classic McEliece security analysis and parameter selection. However, these claims are based on three serious errors.

The first error is comparing the cost of a memory-intensive attack to the cost of an AES attack "on our hardware," while ignoring the fact that the machine puts a large volume of hardware into optimizing memory access and only a tiny fraction of the hardware into AES circuits. It is easy to make an AES attack sound much more expensive than it actually is if one uses a highly suboptimal AES attack machine.

The second error is modeling random access to an array of size 2^80 as being as cheap as 80 bit operations. The way the paper arrives at this physically implausible model is by arguing that logarithmic cost "most accurately models our experimental data". The data points were selected from a limited range for which memory access has essentially constant cost on that machine; this is selection bias, not a valid basis for extrapolation.

The third error is jumping from a comparison in this cheap-RAM model to a claim of a Classic McEliece "overestimate". The submission has always said that the number of bit operations is "considerably below 2^256"; obviously this does not reach 2^272 if one assigns merely logarithmic cost to RAM (changing the exponent by at most 8 bits). What the paper says here is matching what the submission says, not disputing it. The Classic McEliece assignment of category 5 to the 6960119 parameter set has always been explicitly based on the real cost of memory access.

From: pqc-forum@list.nist.gov on behalf of Andre Esser <andre.r.esser@gmail.com>
Sent: Wednesday, June 8, 2022 5:20 AM
To: pqc-forum
Cc: D. J. Bernstein; pqc-forum; pqc-comments; pqc-comments
Subject: [pqc-forum] Re: ROUND 3 OFFICIAL COMMENT: Classic McEliece

The parameter selection process of Classic McEliece takes the asymptotic runtime exponent of Prange and chooses the code parameters such that it approximately matches the respective AES-keysize, i.e., 128, 192 or 256. The security analysis then relies on the not quantified statement that no algorithmic improvement over Prange needs to be considered because in a real attack the memory access costs outweigh the improvement.

It does not need much to "challenge this security analysis and parameter selection". And without a doubt this can not be sufficient for a final standardization of parameter sets. For this, the McEliece team has to extend its submission (in the forth round) by the necessary formalisms that support their security arguments.

Instead the team attacks the credibility of the authors of one of the first formal treatments, arriving at the conclusion of a **slight** security overestimation. It seems unclear, how the team can rule out a security deficit of a few bits with such certainty, but is failing to give a convincing argument / formalism in the submission.

> The first error is comparing the cost of a memory-intensive attack to
> the cost of an AES attack "on our hardware," while ignoring the fact
> that the machine puts a large volume of hardware into optimizing memory
> access and only a tiny fraction of the hardware into AES circuits. It is
> easy to make an AES attack sound much more expensive than it actually is
> if one uses a highly suboptimal AES attack machine.

While we would not agree that a processor with AES hardware acceleration is a particularly suboptimal way of attacking AES, there are of course more specialized systems, but so are for ISD attacks. This goes probably back to the vague security definitions in form of category I, III and V, which do not specify any benchmarking platform, model or metrics. But here you are asking **us** to provide the formalisms on which you seem to have based the security of the parameter sets.
We are happy to have a dialogue about it, but the "error" of missing formalisms for such a comparison is not on us. Also note that the effect of the exact machine we used in comparison to a completely theoretical treatment is insignificant.
We arrive at almost the same security-deficits as Esser-Bellini.

> The second error is modeling random access to an array of size 2^80 as
> being as cheap as 80 bit operations. The way the paper arrives at this
> physically implausible model is by arguing that logarithmic cost "most
> accurately models our experimental data". The data points were selected
> from a limited range for which memory access has essentially constant
> cost on that machine; this is selection bias, not a valid basis for
> extrapolation.

We are modeling the **amortized** memory access cost. This accounts for the fact that not every memory access is completely

random. Moreover, our data structures are specifically designed to allow for good access patterns.
Of course, by the inherent limitation given by our hardware constraints the data points for extrapolation are selected from a
limited range. However, the high memory experiments were run using about 1.6TB of RAM (which gives significant access times).
And still we find the point where high memory beats low memory at code length 1400. If we theoretically model higher memory
access costs this break even point should not exist.

> The third error is jumping from a comparison in this cheap-RAM model to
> a claim of a Classic McEliece “overestimate”. The submission has always
> said that the number of bit operations is “considerably below 2^256";
> obviously this does not reach 2^272 if one assigns merely logarithmic
> cost to RAM (changing the exponent by at most 8 bits). What the paper
> says here is matching what the submission says, not disputing it. The
> Classic McEliece assignment of category 5 to the 6960119 parameter set
> has always been explicitly based on the real cost of memory access.

The McEliece submission does never specify what the “real cost of memory access” is and to what extend it influences the security
level. Making it easy to rule out any access cost that contradicts the security as “not real”. Moreover, the category 3 set has now
been found repeatedly to not match the security guarantees even under higher memory access costs.

We do not claim that our work is the holy grail in terms of formalizing security arguments for McEliece, but the introduction of
formalisms were overdue. Admittedly, the memory access models are quite idealized, but we are convinced that a cube-root modeling
of the amortized cost overestimates the effort.

We are happy to have discussions about it and to see other models evolving that do a better job in capturing all real world factors.
And finally, of course, we hope to see the results embedded in the Classic McEliece specification and security analysis.

However, we would like all discussions to be peace- and respectful. And preferably in person or in papers, to avoid the time consuming
crafting of such long monologues.

Best Regards,
Andre, Alex and Floyd

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, June 14, 2022 12:06 AM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] Re: ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

Andre Esser writes:
> The parameter selection process of Classic McEliece takes the
> asymptotic runtime exponent of Prange and chooses the code parameters
> such that it approximately matches the respective AES-keysize, i.e.,
> 128, 192 or 256.

No, that's not at all how the Classic McEliece parameters were chosen.
The parameter sets were explicitly selected as follows:

* 348864: "optimal security within 2^18 bytes if n and t are required to be multiples of 32".
* 460896: "optimal security within 2^19 bytes if n and t are required to be multiples of 32".
* 6688128: "optimal security within 2^20 bytes if n and t are required to be multiples of 32".
* 6960119: same without the multiples-of-32 restriction.
* 8192128: "taking both n and t to be powers of 2".

Parameter optimization for any specified key size is simpler and more robust than trying to work backwards from comparisons between McEliece and unrelated cryptosystems. There's no inherent power-of-2 requirement for the key sizes, but 1MB, 0.5MB, 0.25MB are easy to remember.

The quotes above are from the submission document that presented the current list of proposed parameters:

https://classic.mceliece.org/nist/mceliece-20190331-mods.pdf

Scientifically, this parameter-selection strategy for McEliece goes back at least to https://eprint.iacr.org/2008/318. See the paragraph in that paper beginning "For keys limited to", in particular obtaining n = 6960 from a 1MB limit.

The underlying security evaluations have always been explicitly based on concrete analyses, _not_ asymptotics (even though asymptotics are helpful for assessing security stability). This was already emphasized in Section 8.2 of the original submission document:

https://classic.mceliece.org/nist/mceliece-20171129.pdf

The section begins as follows:

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

That section continues by pointing to the 2008 paper mentioned above, https://eprint.iacr.org/2008/318, as the source of n = 6960. Anyone checking that paper sees that the paper obtained this value of n via a concrete analysis of that paper's attack, not from asymptotics.

Furthermore, that attack is noticeably faster than Prange's algorithm.
Asymptotically, this cost difference disappears into a 1+o(1) factor in the exponent, but the parameter selection has never relied on any such simplification.

To be clear, it's not that all of the parameter details are from 2008.
For example, to simplify KEM implementations, the Classic McEliece parameter choices avoid list decoding (which would otherwise allow 1 or
2 extra errors). More importantly, the smaller parameter sets were not in the original 2017 proposal; they were added in 2019, in response to NIST making clear in 2019 that it wanted smaller options.

For any particular parameter set, evaluations of the Classic McEliece parameter proposals using the 2008 scripts and various post-2008 scripts show some differences, mostly minor differences regarding exactly which attack overheads are counted and exactly which attacks are covered. The largest quantitative differences come from the gaps between free memory and realistic models.

The Classic McEliece team filed an OFFICIAL COMMENT years ago requesting that NIST "fully define the cost metric to be used" for NISTPQC, so that all submission teams could evaluate costs in this metric:

https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/EiwxGnfQgec/m/LqckEVciAQAJ

In the absence of action by NIST to settle on a metric for NISTPQC, the Classic McEliece team filed another OFFICIAL COMMENT in November 2021 with numbers from a recent estimator for all proposed parameter sets:

https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/dN_O0rvsLV4/m/QZ4UjtxnAwAJ

For example, that estimator says 2^279.2 for the 6960119 parameter set, in line with the expectations stated in the original Classic McEliece submission in 2017. This is not a surprise, given how stable the landscape of attack algorithms has been.

> The security analysis then relies on the not quantified statement that
> no algorithmic improvement over Prange needs to be considered because
> in a real attack the memory access costs outweigh the improvement.

No. Section 8.2 of the 2017 submission document

https://classic.mceliece.org/nist/mceliece-20171129.pdf

started from the 2008 numbers (which are already better than Prange), explicitly considered subsequent algorithms (see also Section 4.1 for references), observed that the 2008 algorithm and subsequent algorithms were bottlenecked by memory access, and stated the following regarding the recommended 6960119 parameter set:

> 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.

Adequate cost quantification wasn't in the literature at that point, but is now readily available from the November 2021 OFFICIAL COMMENT:

https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/dN_O0rvsLV4/m/QZ4UjtxnAwAJ

The submission has always stated that "ISD variants have reduced the number of bit operations considerably below 2^256" for 6960119, so the
category-5 assignment for this parameter set relies on accounting for the costs of memory. For people who want category 5 while ignoring memory costs, the 8192128 parameter set has always been fully specified and implemented as part of the Classic McEliece proposal.

> While we would not agree that a processor with AES hardware
> acceleration is a particularly suboptimal way of attacking AES

Here is one way to see that an AES attacker using the same 7nm chip technology can do _five orders of magnitude_ better than the paper's AES
attack:

* https://www.ant-miner.store/product/antminer-s17-56th/ describes real Bitcoin mining hardware carrying out 56 terahashes/second at 2520 watts, i.e., 45*10^-12 joules per hash. If these hashes are full Bitcoin hashes, 2xSHA-256, then they are roughly the same cost as 16xAES-128, so a similar AES attack box would use roughly 3*10^-12 joules per AES-128.

* The paper in question, https://eprint.iacr.org/2021/1634, reports (in Section 7) 2.16*10^9 "AES encryptions per second performed by our cluster". The cluster has four EPYC 7742 CPUs (according to Section 4). The power consumption of the cluster isn't reported, but presumably is on the scale of 1000 watts, i.e., on the scale of 500000*10^-12 joules per AES-128.

An easier analysis considers AT rather than energy. Each 64-core EPYC
7742 CPU has 32 billion transistors, meaning that each CPU core has half a billion transistors:

https://hexus.net/tech/reviews/cpu/133244-amd-epyc-7742-2p-rome-server/?page=2

The AES attack in question uses the AES instructions, which, on each of these cores, can at best carry out two parallel AES rounds per cycle according to the Zen2 AESENC figures in Agner Fog's tables:

https://agner.org/optimize/instruction_tables.pdf

Each round uses a few thousand bit operations, with a small number of transistors per bit operation, accounting for only a tiny fraction of the transistors in the CPU. Standard key-search circuits instead have almost all transistors performing cipher operations at each moment, with minor overheads for key selection and comparison.

> but so are for ISD attacks.

Quantitatively, compared to their AES hardware, Intel and AMD put _much_ more of their hardware into optimizing memory access---a serious chunk of each core, plus extensive off-chip resources---for obvious reasons.

The point here isn't that it's impossible to use special-purpose hardware to streamline memory-intensive attacks. The point is that a claim regarding the quantitative costs of two attacks, namely AES key search and a memory-intensive ISD

attack, was comparing time but neglecting to account for vast differences in the hardware resources used by the attacks. This is not a meaningful security comparison; it does not correctly predict what large-scale attackers can do.

> But here you are asking *us* to provide the formalisms on which you
> seem to have based the security of the parameter sets.

No. The Classic McEliece team asked "NIST to fully define the cost metric to be used for 'categories'". This is not something that should be decided ad-hoc for particular attacks.

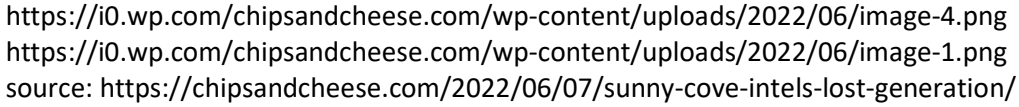
The submission has always explicitly advocated accounting for the real cost of memory ("switching from a bit-operation analysis to a cost analysis"). If it turns out that NIST asks for category 5 with free
memory: as noted above, the 8192128 parameter set has always been available.

> We are modeling the *amortized* memory access cost.

All of these algorithms are bottlenecked by large-scale data motion for, e.g., sorting b-bit chunks of data, where b is small. It is physically implausible that moving b bits of data large distances through an array of size 2^80 can be as cheap as 80b bit operations.

> Of course, by the inherent limitation given by our hardware
> constraints the data points for extrapolation are selected from a limited range.

The hardware does not require this selection: the same machine offers lower-cost caches (and higher-cost storage beyond DRAM). Real graphs of memory-access cost such as


https://i0.wp.com/chipsandcheese.com/wp-content/uploads/2022/06/image-4.png
https://i0.wp.com/chipsandcheese.com/wp-content/uploads/2022/06/image-1.png
source: https://chipsandcheese.com/2022/06/07/sunny-cove-intels-lost-generation/

are forced by distance constraints to be worse than square-root lines in log space, and are then bumped to locally horizontal line segments (L1 cache, L2 cache, etc.) to simplify the engineering. Taking experiments within one of those horizontal line segments, and then extrapolating horizontally from this selection of data (or logarithmically, which on such a small scale is difficult to robustly distinguish from horizontally), gives incorrect predictions for larger sizes.

---D. J. Bernstein (speaking for myself, beyond the quotes)

From: pqc-forum@list.nist.gov on behalf of D. J. Bernstein <djb@cr.yp.to>
Sent: Thursday, June 16, 2022 2:23 PM
To: pqc-comments
Cc: pqc-forum
Subject: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

The following observation may be of interest for people quantifying the stability of state-of-the-art attacks against Classic McEliece.

Our PQCrypto 2008 ISD algorithm is faster than the Eurocrypt 2022 ISD algorithm, on the CPUs selected in the new paper, for the challenges selected in the new paper, according to a direct comparison of (1) our measurements of the 2008 software (the 2+2 case of the 2008 algorithm) and (2) the speeds reported in the new paper for that paper's software.

Instructions for reproducing our measurements appear in README in the following package, along with a review of the known opportunities for further speedups: https://cr.yp.to/software/lowweight-20220616.tar.gz

The new paper's comparison to previous work does not appear to account for various speedups described in the 2008 paper, such as the usage of 2^l-bit tables and the "c" parameter. These speedups are particularly important for the 2+2 case, also influencing comparisons of the 2+2 case to other cases.

---D. J. Bernstein, T. Lange, and C. Peters

From: Andre Esser <andre.r.esser@gmail.com>
Sent: Tuesday, June 21, 2022 7:17 AM
To: pqc-forum
Cc: D. J. Bernstein; pqc-forum; pqc-comments; pqc-comments
Subject: Re: ROUND 3 OFFICIAL COMMENT: Classic McEliece

We never claimed that advanced ISD procedures in the *low-memory regime* (2+2 case) would yield significant improvements over Stern- or Dumer-like decoding.

Citation from our Eurocrypt 2022 paper:
"We find that our implementation performs 12.46 and 17.85 times faster on the McEliece-1284 challenge and 9.56 and 20.36 times faster on the McEliece-1223 instance than [other *Dumer* implementations]"

>>The following observation may be of interest for people quantifying the stability of state-of-the-
>>art attacks against Classic McEliece.

Our work already shows "stability" in the sense that even though McEliece parameters were selected according to a 60 year old algorithm, they miss the security levels only by a few bits. However, the claimed stagnation makes the assumption that the 2+2 case would be the relevant one for McEliece security estimates. Where it is the *high-memory regime* which yields the speedups. Our results show that already for code length of about 1400 the high memory regime yields improvements in practice. Hence, it cannot be neglected entirely in the security analysis.

>>Our PQCrypto 2008 ISD algorithm is faster than the Eurocrypt 2022 ISD algorithm […]

According to the uploaded benchmarks by not even 20%. However, inspecting the code reveals that it is highly specialized and optimized for the *low-memory regime*. Opposed to our code, which allows for high-memory instantiations.

A tuning to the special case together with the mentioned (only 2+2 relevant) improvements is likely to bring us back to the above speedup range. Especially, since we already improved our implementation by a factor of about 2.5. However, as said, an optimization for the high-memory case is the way to go when focussing on McEliece security.

-Andre (speaking for myself)

D. J. Bernstein schrieb am Donnerstag, 16. Juni 2022 um 22:23:59 UTC+4:

> The following observation may be of interest for people quantifying the
> stability of state-of-the-art attacks against Classic McEliece.

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Monday, July 4, 2022 5:57 PM
To: pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
Attachments: signature.asc

Follow Up Flag: Follow up
Flag Status: Completed

D. J. Bernstein, T. Lange, and C. Peters wrote:
> Our PQCrypto 2008 ISD algorithm is faster than the Eurocrypt 2022 ISD
> algorithm, on the CPUs selected in the new paper, for the challenges
> selected in the new paper, according to a direct comparison of (1) our
> measurements of the 2008 software (the 2+2 case of the 2008 algorithm)
> and (2) the speeds reported in the new paper for that paper's software.

As further confirmation, I used the 2008 software to run full attacks on two EPYC 7742 CPUs (the 2022 paper selected these CPUs and used four of them) against all three dimension-1223 Goppa challenges available from https://decodingchallenge.org. (The 2022 paper covered one of the dimension-1223 challenges and a dimension-1284 challenge.) All three runs completed successfully without any surprises.

Thanks to the Fast Crypto Lab cluster at the Institute of Information Science at Academia Sinica, Taiwan, for providing the dual EPYC 7742.

The exact time for any particular attack run is well known to have considerable randomness, just like searching randomly for an AES key.
Because of the variance in run times, individual run times are not statistically meaningful and should never be used to compare algorithms of this type. However, I monitored wall-clock times, CPU times, cycle counts (using RDTSC, which runs at 2.245GHz on these CPUs), and iteration counts to check for any discrepancies across these numbers.

All three of the runs happened to be lucky, using, respectively, just 83%, 1.7%, and 83% of the average number of iterations (184 billion):
more precisely, 153022933873, 3111365348, 153926141900 iterations. Each iteration took 1.68 million cycles (average; there was some variance, as expected from the cache structure). The wall-clock times using 128 cores were about 248 hours, 5 hours, and 250 hours. Here are the CPU times and wall-clock times in more detail:

```
114454917.40user 497.65system 248:23:46elapsed 12799%CPU (0avgtext+0avgdata 15380maxresident)k
2327597.87user 21.17system 5:03:05elapsed 12799%CPU (0avgtext+0avgdata 15488maxresident)k
115167358.18user 200.05system 249:56:30elapsed 12799%CPU (0avgtext+0avgdata 15496maxresident)k
```

The output vectors for the three runs appear below, in the same order as the "providers" on https://decodingchallenge.org.

It is natural to wonder whether lucky runs are actually an indication that average iteration counts have been miscalculated. Here are three independent ways to check the calculations.

The first is to run more experiments, meaning smaller runs for any given CPU budget. For example, here are the iteration counts observed in running the same code (with the same l=22, m=1, c=8) against a range of smaller challenges from https://decodingchallenge.org:

```
431 3.8% 154 4060 4033
431 52.8% 2144 4060 4033
431 60.0% 2435 4060 4033
482 131.1% 18482 14101 14029
482 220.7% 31120 14101 14029
482 16.6% 2338 14101 14029
534 45.2% 5341 11820 11748
534 47.5% 5616 11820 11748
534 2.6% 302 11820 11748
587 33.2% 13600 40969 40766
587 73.6% 30168 40969 40766
587 9.6% 3937 40969 40766
640 154.1% 225989 146688 146077
640 51.2% 75132 146688 146077
640 137.4% 201604 146688 146077
695 131.9% 730398 553830 551850
695 43.9% 243069 553830 551850
695 1.1% 6030 553830 551850
751 64.3% 6500820 10111752 10087366
751 67.3% 6803082 10111752 10087366
751 134.4% 13586235 10111752 10087366
808 133.9% 55699927 41583562 41492753
808 11.4% 4748656 41583562 41492753
808 3.5% 1460929 41583562 41492753
865 60.7% 96347036 158848155 158526591
865 53.9% 85624754 158848155 158526591
865 11.6% 18492422 158848155 158526591
923 269.7% 1847405002 684909178 683634590
923 174.9% 1197860601 684909178 683634590
923 6.3% 43134954 684909178 683634590
982 35.1% 974255720 2776326652 2771487282
982 128.7% 3572455007 2776326652 2771487282
982 24.1% 668935675 2776326652 2771487282
1041 47.4% 236327577 498978858 497804860
1041 248.1% 1237981333 498978858 497804860
1041 170.7% 851782527 498978858 497804860
```

The first column is the dimension. The last two columns are the average iteration counts calculated for type-1 and type-3 iterations, always very close together. (The attack code uses type-2 iterations, which are intermediate.) The third column is the observed iteration count for one attack. The second column is the observed iteration count as a percentage of the calculated average type-1 iteration count.

A graph of this distribution of percentages shows no evident anomalies compared to graphs of percentages sampled from the calculated distribution. The usual statistics do not show surprising p-values. One can, of course, carry out many more experiments to pin down the experimental average to within, e.g., 1% and check for a match with the calculated average. The 2008 paper already reported doing this across millions of experiments for small sizes.

The second way to check the calculations is to review the calculation software, including (1) the formulas and (2) examples checked by hand.
This work was already done in 2008 for the C calculator available from https://github.com/christianepeters/isdf2. I've further checked output of the C calculator against a new Sage calculator for both "type 1" and "type 3" iterations, and checked the Sage calculator against the Markov chains described in the paper.

The third way to check the calculations is to do a much simpler calculation of the number of _independent_ iterations, equivalent to taking the c parameter much larger. Structurally, it's obvious that this will be smaller than the actual number of iterations for (say) c=8, but not _much_ smaller, since randomizing 8 columns has a considerable chance of changing the error weight in the information set. For dimension 1223, this calculation is an unsurprising 7% smaller than the
185 billion calculated for c=8.

The average 185 billion iterations at 1.68 million cycles/iteration for the 2008 software match the 6.28 days reported in README in

https://cr.yp.to/software/lowweight-20220616.tar.gz

for the dimension-1223 challenges on four EPYC 7742 CPUs. This is faster than the 8.22 days reported at the bottom of page 12 of the 2022 paper for the calculated average time for that paper's AVX2-specific software attacking those challenges on those CPUs.

The 2022 paper claims to "demonstrate that these algorithms lead to significant speedups for practical cryptanalysis on medium-sized instances (around 60 bit)"; concretely, it says "12.46 and 17.85 times faster on the McEliece-1284 challenge and 9.56 and 20.36 times faster on the McEliece-1223 instance than [14] and [24]". But [14] and [24] are missing various speedups from the 2008 paper. The 2022 paper failed to compare the speed of its algorithm to the speed of the 2008 algorithm.

The README from https://cr.yp.to/software/lowweight-20220616.tar.gz also runs through known speedups not included in the 2008 software, and concludes as follows:

> Overall it would not be surprising if at least half of the attack cycles
> can be removed on current CPUs compared to the 2008 code running on
> current CPUs. A much larger reduction in attack cost from 2008 to 2022
> has come from changes in hardware: computers do more per cycle and cost
> less per cycle. Accounting for continued improvements in technology is
> an important part of selecting cryptosystem parameters.

---D. J. Bernstein

positions 606 518 167 959 461 163 190 302 1204 206 812 668 947 240 368 837 187 1117 829 723 1160 1014 913 sorted
163 167 187 190 206 240 302 368 461 518 606 668 723 812 829 837 913 947 959 1014 1117 1160 1204 vector
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000010001000000000000000000010010
000000000000001000000000000000000000000000000000010000000000000000000000000000000000000000000000
000000000000001000000000000000000000000000000000000000000000000000000000000000001000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000001000000000000000000
000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000001000
000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000
000000000000000000000000000000000000000000001000000000000000010000000100000000000000000000000000
000000000000000000000000000000000000000000000000010000000000000000000000000000000100000000000001

00000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000
00000001000000000000000000000000000000000000000010000000000000000
positions 101 1073 350 529 519 827 1210 410 1038 797 479 906 418 403 342 575 146 397 805 226 716 704 102 sorted
101 102 146 226 342 350 397 403 410 418 479 519 529 575 704 716 797 805 827 906 1038 1073 1210 vector
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000110000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000
00000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000010000001000000000000000000000000000000000000000000
00000000000010000010000001000000100000000000000000000000000000000000000000000000000000000000000001
00000000000000000000000000000000000010000000010000000000000000000000000000000000000000000000000001
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000010000000001000000000000000000000000000000000000000000000000000000000
00000000000000000000000000010000000100000000000000000000010000000000000000000000000000000000000000
00000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000
00000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000001000000000000
positions 797 450 705 1107 1005 658 1068 345 866 952 335 849 134 987 943 900 553 16 577 407 904 631 1162 sorted
16 134 335 345 407 450 553 577 631 658 705 797 849 866 900 904 943 952 987 1005 1068 1107 1162 vector
00000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000010000000001000000000000000000000000000000000000000000
00000000000000000010000000000000000000000000000000000000000000000010000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000010000000000000000000000000
01000000000000000000000000000000000000000000000000000010000000000000000000000010000000000000000000
00000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000100000000000000000000000000000000000000000000000000010000000000000000000
00100000000000000000000000000000000010001000000000000000000000000000000000000010000000010000000000
00000000000000000000000000010000000000000000100000000000000000000000000000000000000000000000000000
00000000000010000000000000000000000000000000000000010000000000000000000000000000000000000000000000
00000000010000000000000000000000000000000000000000000000000000000000000000000000