From: 4akolzinaolga@gmail.com
Sent: Friday, March 02, 2018 10:22 AM
To: pqc-forum
Cc: 4akolzinaolga@gmail.com
Subject: Re: [pqc-forum] Re: NTRU optimization

Yes, we will publish this as an official comment.

вторник, 27 февраля 2018 г., 23:20:18 UTC+2 пользователь Alperin-Sheriff, Jacob (Fed) написал:

> Is this intended to be an official comment to ntruprime?

From: "4akolz...@gmail.com" <4akolz...@gmail.com>
Date: Tuesday, February 27, 2018 at 2:03 AM
To: pqc-forum <pqc-...@list.nist.gov>
Subject: [pqc-forum] Re: NTRU optimization

Polynomials multiplication

NTRU Prime proposes the use of combined method for multiplying polynomials (Toom and Karatsuba), which doesn't take into account a special form of one of the polynomials (1, -1, 0), which is implemented in the function rq_mult. The authors use AVX2 commands to optimize, the critical code is written in assembler.

We suggest using polynomials of special type for multiplication. We also use AVX2 commands, the critical code is written in assembler.

For parameters and keys that are generated by the NTRUPrime algorithm (q = 4591; p = 761; t = 125) for Linux, we got an acceleration of about 1.5 times while performing the polynomial multiplication operation.

--
You received this message because you are subscribed to the Google Groups "pqc-forum" group.
To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.
Visit this group at https://groups.google.com/a/list.nist.gov/group/pqc-forum/.

From: 4akolzinaolga@gmail.com
Sent: Wednesday, March 14, 2018 7:43 AM
To: pqc-forum
Subject: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Dear NTRU Prime submitters!

In this comment we suggest some improvement for multiplication operation.

NTRU Prime algorithm uses the combined method of multiplication (Toom and Karatsuba) to multiply polynomials, which doesn't take into account the special form of polynomials (1, -1, 0). This method is realized in function rq_mult.

A special type of polynomial may be used for multiplication, as we've investigated. Two arrays may be defined for a polynomial: an array with idices of positive elements and an array with indices of negative elements. The processing of arrays is parallelized.

Our method, like NTRU Prime, uses AVX2 commands, the critical code is written in assembler.

For parameters and keys that were generated by the NTRU Prime algorithm (q = 4591; p = 761; t = 125) on Linux, we got an acceleration of about 1.5 times when executing the operation of multiplying polynomials, compared to the function rq_mult.

Processor: Intel (R) Core (TM) i5-4440 CPU @3.1 GHz, Memory: 8GB

Best regards,

I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina

Ukraine

From: William Whyte <wwhyte@onboardsecurity.com>
Sent: Wednesday, March 14, 2018 8:11 AM
To: 4akolzinaolga@gmail.com
Cc: pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Hi researchers,

Is your implementation constant time? We've found (in the context of "original" NTRU) that multiplication methods that use the trinary form of the private key are hard to make constant time and in general violate the principle that there should be no control flow from secret information to operations.

If you've found a way to make this constant time, then there are additional speedups to be had from taking f (and r) to be of the form (f1*f2) +f3, as described in
J. Hoffstein, J.H. Silverman**Random Small Hamming Weight Products with applications to cryptography**
Discrete Appl. Math., 130 (1) (2003), pp. 37-49

... though you have to be a little careful with the parameters.

Cheers,

William

On Wed, Mar 14, 2018 at 7:42 AM, <4akolzinaolga@gmail.com> wrote:

> Dear NTRU Prime submitters!
>
> In this comment we suggest some improvement for multiplication operation.
>
> NTRU Prime algorithm uses the combined method of multiplication (Toom and Karatsuba) to multiply polynomials, which doesn't take into account the special form of polynomials (1, -1, 0). This method is realized in function rq_mult.
>
> A special type of polynomial may be used for multiplication, as we've investigated. Two arrays may be defined for a polynomial: an array with idices of positive elements and an array with indices of negative elements. The processing of arrays is parallelized.
>
> Our method, like NTRU Prime, uses AVX2 commands, the critical code is written in assembler.
>
> For parameters and keys that were generated by the NTRU Prime algorithm (q = 4591; p = 761; t = 125) on Linux, we got an acceleration of about 1.5 times when executing the operation of multiplying polynomials, compared to the function rq_mult.
>
> Processor: Intel (R) Core (TM) i5-4440 CPU @3.1 GHz, Memory: 8GB
>
> Best regards,
>
> I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina
>
> Ukraine

From: William Whyte <wwhyte@onboardsecurity.com>
Sent: Wednesday, March 14, 2018 10:51 AM
To: Olga Akolzina; pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Hi Olga,

>> Execution time is defined by total number of non-zero elements, this quantity is set by algorithm (t parameter).

This is true to a first order, but in our experience there was additional variation.

>> Time doesn't depend on coefficients indices.

We observed some dependency in our experiments. Do you have experimental results showing that there's no dependency? Can you share those?

Cheers,

William

On Wed, Mar 14, 2018 at 9:34 AM, Olga Akolzina <4akolzinaolga@gmail.com> wrote:

> Execution time is defined by total number of non-zero elements, this quantity is set by algorithm (t parameter). Time doesn't depend on coefficients indices.
>
> We didn't investigate the form (f1*f2)+f3.
>
> Best regards,
>
> I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina

2018-03-14 14:11 GMT+02:00 William Whyte <wwhyte@onboardsecurity.com>:

> Hi researchers,
>
> Is your implementation constant time? We've found (in the context of "original" NTRU) that multiplication methods that use the trinary form of the private key are hard to make constant time and in general violate the principle that there should be no control flow from secret information to operations.
>
> If you've found a way to make this constant time, then there are additional speedups to be had from taking f (and r) to be of the form (f1*f2) +f3, as described in
> J. Hoffstein, J.H. Silverman**Random Small Hamming Weight Products with applications to cryptography**
> Discrete Appl. Math., 130 (1) (2003), pp. 37-49
>
> ... though you have to be a little careful with the parameters.
>
> Cheers,

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Wednesday, March 14, 2018 11:13 AM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

The central claim from Gorbenko, Kachko, Yesina, and Akolzina is that it "makes no sense" to switch from the traditional NTRU multiplication algorithms (using the sparsity of one input) to "complex" multiplication algorithms (Karatsuba, Toom, etc.). From a performance perspective, the evidence presented for this claim has at least two serious flaws:

* The speeds claimed here for "complex" multiplication algorithms are worse than previously published software. My understanding is that the authors measured their own implementation of these algorithms, not using state-of-the-art implementation techniques for this CPU.

* My understanding is that the claimed bottom line, "acceleration of about 1.5 times", is actually comparing a 4-core implementation to a 1-core implementation. This use of 4 cores has worse throughput, energy consumption, etc. than simply running separate computations on separate cores.

More importantly, from a security perspective, we require constant-time algorithms. Even if sparse techniques can be competitive in speed (which is unproven), I agree with William's assessment that those techniques are hard to make constant time.

The bigger picture is that the constant-time cycle counts reported on https://na01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fntruprime.cr.yp.to&data=02%7C01%7Csara.kerman%40nist.gov%7C980b212c6f2e4882159608d589be1681%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C1%7C636566371878471392&sdata=%2FKSgNVO8Q4YyYrLrgJhXqEt8T7PT46fnkWeIk35mkz4%3D&reserved=0 are already so fast that it's hard to find any applications that can't afford them. Obviously even more speed is nice if we can get it (which is why new sorting code is coming soon!), but speed should be measured properly, and it shouldn't come at the expense of security.

---Dan

P.S. To be clear: I'm _not_ saying that applications can always handle the sizes of lattice-based ciphertexts, typically around a kilobyte.

From: Olga Akolzina <4akolzinaolga@gmail.com>
Sent: Thursday, March 15, 2018 9:26 AM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Hello!

>> My understanding is that the authors measured their own implementation of these algorithms, not using state-of-the-art implementation techniques for this CPU.

We used NTRU Prime optimized code (AVX + assembler inserts) for speed measurement.

>> My understanding is that the claimed bottom line, "acceleration of about 1.5 times", is actually comparing a 4-core implementation to a 1-core implementation. This use of 4 cores has worse throughput, energy consumption, etc. than simply running separate computations on separate cores.

Yes, but both functions were performed on a 4-core processor, we do not see the possibility of effective vectorizing your algorithm, as the size of data being multiplied is gradually decreasing.

Thank you for recommendation, we'll do an experiment on time and indices independence.

Best regards,

I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina

2018-03-14 17:12 GMT+02:00 D. J. Bernstein <djb@cr.yp.to>:

> The central claim from Gorbenko, Kachko, Yesina, and Akolzina is that it
> "makes no sense" to switch from the traditional NTRU multiplication
> algorithms (using the sparsity of one input) to "complex" multiplication
> algorithms (Karatsuba, Toom, etc.). From a performance perspective, the
> evidence presented for this claim has at least two serious flaws:
>
> * The speeds claimed here for "complex" multiplication algorithms are
> worse than previously published software. My understanding is that
> the authors measured their own implementation of these algorithms,
> not using state-of-the-art implementation techniques for this CPU.
>
> * My understanding is that the claimed bottom line, "acceleration of
> about 1.5 times", is actually comparing a 4-core implementation to
> a 1-core implementation. This use of 4 cores has worse throughput,
> energy consumption, etc. than simply running separate computations
> on separate cores.
>
> More importantly, from a security perspective, we require constant-time
> algorithms. Even if sparse techniques can be competitive in speed (which
> is unproven), I agree with William's assessment that those techniques
> are hard to make constant time.

From: Olga Akolzina <4akolzinaolga@gmail.com>
Sent: Thursday, March 15, 2018 10:22 AM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

We've done 2000 tests with different keys, time dispersion doesn't exceed 12%.

2018-03-15 15:26 GMT+02:00 Olga Akolzina <4akolzinaolga@gmail.com>:

> Hello!
>
> >> My understanding is that the authors measured their own implementation of these algorithms, not using state-of-the-art implementation techniques for this CPU.
>
> We used NTRU Prime optimized code (AVX + assembler inserts) for speed measurement.
>
> >> My understanding is that the claimed bottom line, "acceleration of about 1.5 times", is actually comparing a 4-core implementation to a 1-core implementation. This use of 4 cores has worse throughput, energy consumption, etc. than simply running separate computations on separate cores.
>
> Yes, but both functions were performed on a 4-core processor, we do not see the possibility of effective vectorizing your algorithm, as the size of data being multiplied is gradually decreasing.
>
> Thank you for recommendation, we'll do an experiment on time and indices independence.
>
> Best regards,
>
> I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina

2018-03-14 17:12 GMT+02:00 D. J. Bernstein <djb@cr.yp.to>:

> The central claim from Gorbenko, Kachko, Yesina, and Akolzina is that it
> "makes no sense" to switch from the traditional NTRU multiplication
> algorithms (using the sparsity of one input) to "complex" multiplication
> algorithms (Karatsuba, Toom, etc.). From a performance perspective, the
> evidence presented for this claim has at least two serious flaws:
>
> * The speeds claimed here for "complex" multiplication algorithms are
> worse than previously published software. My understanding is that
> the authors measured their own implementation of these algorithms,
> not using state-of-the-art implementation techniques for this CPU.
>
> * My understanding is that the claimed bottom line, "acceleration of
> about 1.5 times", is actually comparing a 4-core implementation to
> a 1-core implementation. This use of 4 cores has worse throughput,
> energy consumption, etc. than simply running separate computations
> on separate cores.
>
> More importantly, from a security perspective, we require constant-time

From: William Whyte <wwhyte@onboardsecurity.com>
Sent: Thursday, March 15, 2018 10:29 AM
To: Olga Akolzina
Cc: pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Right, the time dispersion isn't huge, but it is observable. It would be great if there was a way to make it go away, as naively it seems that the index-based version should be faster, especially with the f = f1*f2+f3 trick, but we couldn't find a way to get rid of it.

Cheers,

William

On Thu, Mar 15, 2018 at 10:21 AM, Olga Akolzina <4akolzinaolga@gmail.com> wrote:

> We've done 2000 tests with different keys, time dispersion doesn't exceed 12%.
>
> 2018-03-15 15:26 GMT+02:00 Olga Akolzina <4akolzinaolga@gmail.com>:
>
> > Hello!
> >
> > >> My understanding is that the authors measured their own implementation of these algorithms, not using state-of-the-art implementation techniques for this CPU.
> >
> > We used NTRU Prime optimized code (AVX + assembler inserts) for speed measurement.
> >
> > >> My understanding is that the claimed bottom line, "acceleration of about 1.5 times", is actually comparing a 4-core implementation to a 1-core implementation. This use of 4 cores has worse throughput, energy consumption, etc. than simply running separate computations on separate cores.
> >
> > Yes, but both functions were performed on a 4-core processor, we do not see the possibility of effective vectorizing your algorithm, as the size of data being multiplied is gradually decreasing.
> >
> > Thank you for recommendation, we'll do an experiment on time and indices independence.
> >
> > Best regards,
> >
> > I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina
> >
> > 2018-03-14 17:12 GMT+02:00 D. J. Bernstein <djb@cr.yp.to>:
> >
> > > The central claim from Gorbenko, Kachko, Yesina, and Akolzina is that it
> > > "makes no sense" to switch from the traditional NTRU multiplication
> > > algorithms (using the sparsity of one input) to "complex" multiplication
> > > algorithms (Karatsuba, Toom, etc.). From a performance perspective, the
> > > evidence presented for this claim has at least two serious flaws:
> > >
> > > * The speeds claimed here for "complex" multiplication algorithms are
> > > worse than previously published software. My understanding is that

From: Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent: Tuesday, March 27, 2018 9:24 AM
To: pqc-forum
Subject: [pqc-forum] NTRU Prime Code is Imcomplete

Hi,

I started working through the submissions recently (got about 50% working at the moment). but the NTRU Prime code seems to be really missing bits. For example modq.h tries to include

#include "crypto_int16.h"
#include "crypto_int32.h"
#include "crypto_uint16.h"
#include "crypto_uint32.h"

These files are not contained in the package. It's easy enough to work around this (why not use standard stdint.h btw?), but I have no clue what "crypto_hash_sha512.h" is -- it does not appear to be part of any of the libraries given in the "standard evaluation platform" defined by NIST, and not defined by NTL, GMP, or OpenSSL include files.

Furthermore, the submission is bizarrely dated after the submission deadline, directory being named "ntruprime-20171214".

Cheers,
- markku

Dr. Markku-Juhani O. Saarinen <mjos@iki.fi>
--

From: EL HASSANE LAAJI <e.laaji@ump.ac.ma>
Sent: Tuesday, March 27, 2018 11:17 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: NTRU Prime

Hi researchers NTRUprime .
I'm very interested to continuous my researche on NTRU releases , I just finished benchmarking between NTRUprime and NewHope.
In my openeen , the NTRU prime is the best schem. but jour implementation is not realy professional .
I have some remarks about this:
- there is a lot of repetetions functions ,rather than use c++ template technique. for examples: modq_minusproduct(,,) and mod3_minusproduct(,,) the same for other functions in files mod3.h and modq.h...
- another remark is you don't allocate the memory for pointers variables like in files rq.c and small.c and others in your implementations.
it must allocate memory and freeze it after used.
it is possible to reduce your size code until 30% to 50%.
you will have bugues because memory allocations for pointers
....
best regards

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, March 27, 2018 2:41 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: NTRU Prime
Attachments: signature.asc

Four comments in reply to recent questions about NTRU Prime software.

1. This month PQCRYPTO released https://libpqcrypto.org, which includes
77 cryptographic systems from 19 submissions, one of those submissions being NTRU Prime. Compiling (and using) libpqcrypto is much simpler than compiling one NIST submission after another, and the libpqcrypto tests are much more comprehensive than NIST's KAT tests.

2. Regarding C++: The NTRU Prime software is in C, simplifying usage as a library from a wide range of languages. NIST said that submissions "should only use C++ functionality where absolutely required in order to use NTL".

C++ would make this code slightly shorter, but not much, and only in
superficial ways that don't have much to do with code readability. What makes much more of a difference in readability is switching to Sage. See the Sage reference implementations of sntrup4591761 and ntrulpr4591761 available from https://ntruprime.cr.yp.to/software.html.

3. Regarding memory allocation: The NTRU Prime software avoids malloc(), alloca(), large stack arrays, etc. These rules are essential for deployment in some small environments, and improve reliability in many more environments. The use of pointers follows normal C conventions, and beyond this follows a more restricted discipline that is intended to assist ongoing verification projects. The same discipline has been used for some previous crypto software that has already been successfully verified.

I see no basis for the claims that the software "must allocate memory"
and that the software will have bugs "because memory allocations for pointers" (whatever exactly this means). Perhaps this is based on some
C++ coding guide that makes exaggerated claims regarding pointers and
encourages use of C++ references instead. I'm skeptical about the notion that rewriting code according to such guides will simplify verification.

4. Regarding completeness of the submitted code: The steps shown below compile the originally submitted NTRU Prime code (and check the KATs) using the SUPERCOP version that was available at the time of submission.
These steps were tested in under a minute on an Intel E3-1275 v3
(Haswell) running Ubuntu 16.04 with standard development tools (apt install build-essential).

The originally submitted code had C reference implementations for
sntrup4591761 and ntrulpr4591761, and also a fast but non-portable sntrup4591761/avx implementation. Shortly after submission we released ntrulpr4591761/avx and (as requested by NIST) documentation of internal software details. See https://ntruprime.cr.yp.to/software.html.

---Dan

```
cd $HOME
wget https://bench.cr.yp.to/supercop/supercop-20171020.tar.xz
wget https://ntruprime.cr.yp.to/nist/ntruprime-20171130.tar.gz

tar -xf supercop-20171020.tar.xz
( cd supercop-20171020
  sed -i 1q okcompilers/c
  sed -i 1q okcompilers/cpp
  ./do-part init
  ./do-part crypto_verify 32
  ./do-part crypto_hash sha512
  ./do-part crypto_stream aes256ctr
)

tar -xf ntruprime-20171130.tar.gz
( cd ntruprime-20171130/Reference_Implementation/kem/sntrup4591761
  make
  cmp kat_kem.req ../../../KAT/kem/sntrup4591761/kat_kem.req
  cmp kat_kem.rsp ../../../KAT/kem/sntrup4591761/kat_kem.rsp
  cmp kat_kem.int ../../../KAT/kem/sntrup4591761/kat_kem.int
)
( cd ntruprime-20171130/Reference_Implementation/kem/ntrulpr4591761
  make
  cmp kat_kem.req ../../../KAT/kem/ntrulpr4591761/kat_kem.req
  cmp kat_kem.rsp ../../../KAT/kem/ntrulpr4591761/kat_kem.rsp
  cmp kat_kem.int ../../../KAT/kem/ntrulpr4591761/kat_kem.int
)
( cd ntruprime-20171130/Additional_Implementations/kem/sntrup4591761/avx
  make
  cmp kat_kem.req ../../../../KAT/kem/sntrup4591761/kat_kem.req
  cmp kat_kem.rsp ../../../../KAT/kem/sntrup4591761/kat_kem.rsp
  cmp kat_kem.int ../../../../KAT/kem/sntrup4591761/kat_kem.int
)
```

From: Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent: Tuesday, March 27, 2018 5:47 PM
To: pqc-forum
Cc: pqc-comments; djb@cr.yp.to
Subject: Re: OFFICIAL COMMENT: NTRU Prime

Hi,

In addition to candidates dependent on NTL and its C++ interfaces, NTRU Prime certainly required a bit more work than most other candidates to get tested due to these unannounced external dependencies that Dan mentioned in his post. I should have guessed that they were to some library or other thing coming from Dan's group.

A bit of a headscratcher was that the standard API seemed not to be available. This was because crypto_kem.h redefines the function names, e.g.
[..]
#define crypto_kem_keypair crypto_kem_ntrulpr4591761_keypair
#define crypto_kem_enc crypto_kem_ntrulpr4591761_enc
#define crypto_kem_dec crypto_kem_ntrulpr4591761_dec
[..]
This was included by keypair.c etc, so crypto_kem_keypair() was actually not externally available, but crypto_kem_ntrulpr4591761_keypair() was. This is noted in the external web page that Dan mentioned, but not in the submission itself.

Rather than using the standard "seed expander interface" of the NIST API, NTRU Prime used its own crypto_stream_aes256ctr(), which is not part of the submission but had to be implemented with libcrypto. The order of parameters for crypto_hash_sha512() was easy enough to guess and convert to SHA512() which is part the reference platform (libcrypto).

Recent versions of GCC apper to refuse to accept single-letter lower case macro definitions from param.h:

#define q 4591
#define p 761
#define w 250

These were a nightmare to replace to more descriptive upper case macro names in dozens of locations where they were used (params.h was included from about 9 source code modules but was fortunately not part of the external interface).

But anyhow, it IS possible to get NTRU Prime running without SUPERCOP, which is, in my opinion, is super cumbersome (475MB right after untar, and probably gigabytes if you run it).

Cheers,
- markku

On Tuesday, March 27, 2018 at 7:41:20 PM UTC+1, D. J. Bernstein wrote:
Four comments in reply to recent questions about NTRU Prime software.

1. This month PQCRYPTO released https://libpqcrypto.org, which includes
77 cryptographic systems from 19 submissions, one of those submissions
being NTRU Prime. Compiling (and using) libpqcrypto is much simpler than
compiling one NIST submission after another, and the libpqcrypto tests

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Tuesday, March 27, 2018 9:27 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: NTRU Prime
Attachments: signature.asc

My previous message included a short script to test the originally submitted NTRU Prime software and KATs. I mentioned that the script takes under a minute on an Intel E3-1275 v3 (Haswell) running Ubuntu 16.04 with standard development tools (apt install build-essential).

Let me emphasize that this _includes_ the time for the script to build all relevant subroutines from supercop-20171020. This uses SUPERCOP's "do-part" tool, based on John Schanck's "supercop-fastbuild".

A newer release, supercop-20171218, internally includes and tests both sntrup4591761 and ntrulpr4591761; i.e., the software was already tested months ago and works fine. These and many more KEMs are also tested by libpqcrypto-20180314.

Regarding .h files: In the NaCl/SUPERCOP/.../libpqcrypto API, obviously there's a difference between (e.g.)

* the crypto_hash_sha256() provided by crypto_hash_sha256.h and
* the crypto_hash_sha512() provided by crypto_hash_sha512.h,

but the people implementing these functions have always been allowed to simply define crypto_hash() after including crypto_hash.h. The central library-compilation tools automatically create the crypto_hash.h file to make this work, with all necessary function declarations and macros renaming crypto_hash as (e.g.) crypto_hash_sha512.

As discussed on the list in September, NIST has skipped this automatic setup of crypto_*.h, instead requiring each implementation to provide similar work as part of its own .h files. Most primitives don't rename functions in their .h files, but this means that those primitives can't be linked together into a single library. The NTRU Prime software includes appropriate function renaming.

More broadly, the big picture of building cryptographic libraries for use in production implies extra goals for implementors. There are, for example, a huge number of different SHA-3 implementations included in the software submitted to NIST, often not exactly matching the functions provided by the Keccak Code Package. Getting rid of this redundancy--- figuring out the right SHA-3 functions to provide centrally for use by all of these submissions---will be a win for optimization, auditing, verification, etc., even though it will produce extra software layers.

Regarding the idea of replacing calls to crypto_stream_aes256ctr() with calls to NIST's "seed expander": This would produce different outputs, in violation of the NTRU Prime specification, and would not interoperate with correct implementations.

Finally, if there's some surprising portability issue with "recent versions of gcc" then an appropriate report can be expected to produce appropriate software updates. However, it's important for reports to answer the standard debugging questions (what exactly did you do? what exactly did you expect the computer to do? what exactly did the computer do differently?). If it turns out that the problem is actually with someone's modified version of the software and not with the original software then the obvious solution is to use the original software.

---Dan

From: 4akolzinaolga@gmail.com
Sent: Wednesday, March 28, 2018 6:26 AM
To: pqc-forum
Subject: [pqc-forum] Re: OFFICIAL COMMENT: NTRU Prime

More precise research results

1. For 10,000 keys' maximum dispersion of all measurements is

-5.19676<=e<=6.62797(%).

The key numbers for which the minimum and maximum values were obtained when the measurements were repeated didn't match.

2. For 100 keys, the maximum dispersion is -4.142 <= e <= 6.06054 (%).

3. For 100 keys and 100 times measurements, the maximum dispersion of minima, min: -0.745778 <= e <= 0.732426 (%), maxima dispersion from the average maximum, max: -0.891563 <= e <= 0.977177 (%). Thus, the measurement error can be up to 3%.

4 At the moment, the function rand (standard C library) was used to generate the numbers of polynomials coefficients with nonzero values. In future we suppose to use for this purpose more reliable generator and perform a full statistical analysis of the results in order to identify the possibility of obtaining a key, taking into account the time difference in measurements.

Thank You!

Best regards,

I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina

From: Alperin-Sheriff, Jacob (Fed)
Sent: Monday, April 23, 2018 11:27 AM
To: pqc-comments
Subject: OFFICIAL COMMENT: NTRU Prime

NTRU Prime team:

I was going through the “Estimate all the {LWE, NTRU} Schemes” site
https://estimate-all-the-lwe-ntru-schemes.github.io/docs/

And notice that the estimates there for sieving algorithms NTRU prime are well below the claimed security for most of the algorithms.

I notice that your team claims that achieving sieving algorithms in practice that outperform enumeration algorithms is not realistic in practice, but the security level also falls significantly below the claimed Category 5 for the Q-Core-Enum and Lotus models, which doesn’t seem to be covered in the supporting documentation.

—Jacob Alperin-Sheriff

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Wednesday, April 25, 2018 11:02 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime
Attachments: signature.asc

Speaking for myself here, with the objective of highlighting a severe quantitative error by NIST, along with the procedures that led to this error. I don't think an NTRU Prime team comment is needed.

Let me begin with the rules specified in the call for proposals. Each submission is required to contain "an analysis of the algorithm with respect to known attacks". This is separate from the "description of the expected security strength".

The NTRU Prime documentation accordingly does a ton of work to analyze known attacks. I'm not aware of any part of the analysis that requires a correction or update. There's one part that should be worked out in more detail, namely the hybrid attack actually running somewhat more slowly, for reasons identified in the cited Wunderer paper.

Separately from this, the NTRU Prime documentation describes the expected security strength, and the measures taken to protect users against various potential future advances in attacks.

For comparison, most lattice-based submissions skimp on the requirement to analyze known attacks. They instead use massive underestimates of the costs of known attacks---underestimates designed for simplicity rather than accuracy. This might sound safe (even "conservative") but it causes problems in at least four directions:

* Absolute security: These massive underestimates are _constantly_ being misinterpreted and mislabeled as stating actual security levels. Unbroken systems are incorrectly described as broken. Potential users end up thinking that post-quantum crypto is less practical than it actually is.

* Relative security: Different proposals are underestimated by different amounts. I've already given the example of EMBLEM-611 vs. uRound2.KEM-500:

  - One "estimate" says that EMBLEM-611 is hundreds of times easier to break than uRound2.KEM-500 (2^76 vs. 2^84).
  - Another "estimate" says that, no, uRound2.KEM-500 is thousands of times easier to break than EMBLEM-611 (2^126 vs. 2^142).

  Which of these two systems is more secure? Which of the "estimates" is giving the wrong answer to this question?

* Risk assessment: NIST's rules correctly point to "the complexity of the best known attack has recently decreased significantly" as an example of attacks being "poorly understood". Massive underestimates have the general effect of hiding this history, making attacks seem better understood than they actually are.

* Cryptanalytic incentives: Promotion of underestimates---especially if they're not clearly labeled as such---makes it unnecessarily difficult for researchers to publish _actual_ speedups in attacks.

Here's a concrete example of the problems created in one of these directions. NIST

* refers to "Q-Core-Enum" as a "model" of the "security level";
* alludes to the 2^187 claimed by this formula for sntrup4591761;
* says that this is "significantly below the claimed Category 5"; and
* files this as a formal comment, evidently requesting explanation.

Wow, sounds like a serious conflict between the claimed security level and this well-known "Q-Core-Enum model"! But the picture looks totally different for anyone who digs into the details:

(1) References indicate that the "Q-Core-Enum" formula was introduced in the NTRU-HRSS-KEM analysis. This analysis, in turn, says that this formula assumes a "purely hypothetical" (their words!) quantum square-root speedup in enumeration compared to a quasilinear extrapolation of the Chen--Nguyen experiments.

Similarly, Laarhoven, Mosca, and van de Pol had written "There seems to be no simple way to apply quantum search to the enumeration algorithms that are currently used in practice", which evidently disclaims knowledge of such an attack.

Formally, I could stop at this point, since we're not talking about a known attack. But I've noticed that some people don't seem to understand the difference between analyzing known attacks and making predictions about future attacks. If I report the simple fact that Q-Core-Enum is talking about an algorithm that isn't known to exist, will I be misrepresented as endorsing what the literature says regarding the difficulty of finding the algorithm?

My actual objective here is to highlight a severe quantitative error (see below), so I'm going to explicitly _skip_ the "purely hypothetical" argument. Instead I'll make two observations about the performance of an algorithm of this type.

First, the reported 2^187 is (supposedly---I haven't checked) calculated as the square root of the (extrapolated) number of non-quantum operations. Evidently this number of non-quantum operations is around (2^187)^2 = 2^374. Keep this 2^374 in mind; I'll come back to it later.

Second, presumably a quantum enumeration algorithm will actually have a ton of overhead beyond the square root. I don't mean the general overhead of quantum computation; I mean the cost of the node evaluation (if the 2^374 total is actually 2^354 nodes then 2^187 turns into 2^197), the cost of reversibility, the cost of reversibly managing the search-space irregularities discussed by Laarhoven--Mosca--van de Pol, etc.

(2) Following further references shows that the "Core" part of "Q-Core-Enum" refers to https://eprint.iacr.org/2015/1092.pdf, which explicitly says that it ignores a polynomial factor and that this is "clearly a pessimistic estimation (from the defender's point of view)".

(3) The 2^187 produced by the formula is a count of ill-defined "operations" that, presumably, are much larger than single gates.

(4) The comparison is to Category 5, the (conjectured) difficulty of finding an AES-256 key. Traditionally this would be estimated around 2^150 or 2^160 quantum gates, so 2^187 gates (never mind 2^187 "operations") clearly wouldn't be an improvement.

NIST recognizes, correctly, that the actual Grover speedup here is limited by MAXDEPTH. So, depending on MAXDEPTH, the cost of breaking AES is going to be more like 2^234 quantum gates; and perhaps Grover won't be of _any_ use. But then _why doesn't NIST recognize exactly the same obstacle for the supposed application of Grover to enumeration_?

Issues #1, #2, and #3 aren't _necessarily_ fatal for NIST's comparison between the 2^187 and the 2^234. If a quantum enumeration algorithm exists, it's not inconceivable that

* the quantum-enumeration overhead from #1,
* times the polynomial factor ignored in "Core" in #2,
* times the number of gates per "operation" in #3,
* times the reported 2^187,

is below 2^234.

Issue #4, on the other hand, is clearly fatal. There's no way that a quantum search through (2^187)^2 = 2^374 enumeration nodes (with a very small number of target nodes) is going to beat a quantum search through 2^256 AES keys. At the risk of belaboring the obvious, let me spell out the details in three post-quantum metrics:

* In the traditional, rather vague, "operations" metric without latency limits, searching through 2^374 nodes costs 2^187 "operations". This obviously doesn't beat searching through 2^256 AES keys, which costs only 2^128 "operations".

* If latency is limited to, say, 2^64 "operations", then searching through 2^374 nodes costs 2^310 "operations". This obviously doesn't beat searching through 2^256 AES keys, which costs only 2^192 "operations".

* In a limited-latency gate-count metric, searching through 2^374 nodes will cost many more than 2^310 gates---of course one would need to see an actual algorithm to work out the details---and this is again obviously not competitive with 2^234 gates to search through 2^256 AES keys.

In short, saying that the 2^187 is below the 2^234 is nonsense. This comparison requires either

* misunderstanding the 2^234 as being in the first metric, which is omitting a factor around 2^106, or

* misunderstanding the 2^187 as being in the third metric, which is omitting a factor above 2^123.

This omitted factor above 2^100 is the severe quantitative error that I mentioned above---never mind the further factors from #1, #2, and #3.

I think it's useful to contemplate some ways that this error could have been caught _before_ it became the foundation of a formal comment:

* The error would have been clear if NIST had insisted on accurate labeling. For months I've been asking on this list for clear and realistic cost metrics, hypothetical attacks being clearly distinguished from known attacks, etc.:

  https://groups.google.com/a/list.nist.gov/forum/message/raw?msg=pqc-forum/UFxDg9TenNE/y_xKN7S3CwAJ
  https://groups.google.com/a/list.nist.gov/forum/message/raw?msg=pqc-forum/UFxDg9TenNE/hmFEfo88CwAJ
  https://groups.google.com/a/list.nist.gov/forum/message/raw?msg=pqc-forum/h4_LCVNejCI/FyV5hgnqBAAJ
  https://groups.google.com/a/list.nist.gov/forum/message/raw?msg=pqc-forum/xFxHuxPXTO4/1LEWsQ8eAQAJ

  Making the metric explicit shows that it's nonsense to compare the 2^187 to the 2^234. Focusing on a realistic metric prevents the 2^187 from showing up in the first place.

* The error is also clear from a glance at what the (hypothetical) attack is supposed to be doing---a quantum search through 2^374 enumeration nodes. However, even this minimal amount of information about the algorithm ends up being suppressed in giant tables full of numbers such as the above 187.

  I think the people spreading numbers such as this 187 have to take responsibility for the misinformation that they're planting in the minds of the readers. The responsibility is even larger when these numbers are collected into large tables and graphs---readers are attracted to, and easily fooled by errors in, such collections, as we've seen from how NIST handled the 187 in this case.

  Let me propose a simple first step that would certainly improve accuracy, and that should help reduce confusion: Rename "Estimate all the schemes" as "Underestimate all the schemes".

More broadly, I would like to see more attention to ways in which evaluation can be scaled to a large volume of submissions _without_ compromising the quality of the evaluation.

---Dan

P.S. Looking at the "all the schemes" tables, I see another problem regarding NTRU Prime in particular.

The NTRU Prime submission presents a family tree of four NTRU options beyond the choice of ring, with terminology designed to aid in comparisons (while giving appropriate credit to the original NTRU idea):

* "Noisy Product NTRU" (typical "Ring-LWE-based" systems fit here);
* "Rounded Product NTRU" (typical "Ring-LWR-based" systems fit here);
* "Noisy Quotient NTRU" (original NTRU);
* "Rounded Quotient NTRU" (harder to find in the literature).

The main point of the submission is the NTRU Prime ring choice, and beyond this the submission recommends Rounded NTRU over Noisy NTRU, but the submission is explicitly agnostic regarding Product NTRU vs. Quotient NTRU. There are two KEMs in the submission:

* an example of Rounded Product NTRU (ntrulpr4591761) and

* an example of Rounded Quotient NTRU (sntrup4591761).

I'm puzzled to see that the "all the schemes" tables, while listing various options as "RLWE" and "RLWR", list both ntrulpr4591761 and
sntrup4591761 as "NTRU", along with various other "NTRU" variants. This classification looks wrong in two dimensions:

* If Noisy NTRU and Rounded NTRU are supposed to have different security, as suggested by the separate listings of Noisy Product NTRU ("RLWE") and Rounded Product NTRU ("RLWR"), then why are Noisy Quotient NTRU (e.g., original NTRU) and Rounded Quotient NTRU (e.g., sntrup4591761) lumped together?

* There are certainly some small differences (with no clear winner!) in attacks against Product NTRU and Quotient NTRU, so it's quite reasonable to separate "RLWE"/"RLWR" from "NTRU"---but then I don't see any justification for putting ntrulpr4591761 on the "NTRU" side rather than the "RLWR" side.

I wouldn't be surprised if this misclassification has also produced quantitative errors, although the general pattern of underestimation strikes me as a much bigger problem.

From: Perlner, Ray (Fed)
Sent: Wednesday, April 25, 2018 1:48 PM
To: D. J. Bernstein; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: RE: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

You're absolutely right that the complexity of q-core-enum is not relevant to security strength 5, since it's a quantum attack. Unless there's a better than Grover speedup, security strength 5 pretty much amounts to 256 bits of security against classical attacks. That said, at least on cursory inspection, I didn't see any evidence that LOTUS was estimating the cost of a quantum attack. What's wrong with the Lotus-Enum estimate in your opinion?

-----Original Message-----
From: D. J. Bernstein [mailto:djb@cr.yp.to]
Sent: Wednesday, April 25, 2018 11:02 AM
To: pqc-comments <pqc-comments@nist.gov>
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Speaking for myself here, with the objective of highlighting a severe quantitative error by NIST, along with the procedures that led to this error. I don't think an NTRU Prime team comment is needed.

Let me begin with the rules specified in the call for proposals. Each submission is required to contain "an analysis of the algorithm with respect to known attacks". This is separate from the "description of the expected security strength".

The NTRU Prime documentation accordingly does a ton of work to analyze known attacks. I'm not aware of any part of the analysis that requires a correction or update. There's one part that should be worked out in more detail, namely the hybrid attack actually running somewhat more slowly, for reasons identified in the cited Wunderer paper.

Separately from this, the NTRU Prime documentation describes the expected security strength, and the measures taken to protect users against various potential future advances in attacks.

For comparison, most lattice-based submissions skimp on the requirement to analyze known attacks. They instead use massive underestimates of the costs of known attacks---underestimates designed for simplicity rather than accuracy. This might sound safe (even "conservative") but it causes problems in at least four directions:

* Absolute security: These massive underestimates are _constantly_ being misinterpreted and mislabeled as stating actual security levels. Unbroken systems are incorrectly described as broken. Potential users end up thinking that post-quantum crypto is less practical than it actually is.

* Relative security: Different proposals are underestimated by different amounts. I've already given the example of EMBLEM-611 vs. uRound2.KEM-500:

  - One "estimate" says that EMBLEM-611 is hundreds of times easier to break than uRound2.KEM-500 (2^76 vs. 2^84).
  - Another "estimate" says that, no, uRound2.KEM-500 is thousands of times easier to break than EMBLEM-611 (2^126 vs. 2^142).

  Which of these two systems is more secure? Which of the "estimates"

From: Moody, Dustin (Fed)
Sent: Wednesday, April 25, 2018 2:12 PM
To: D. J. Bernstein; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: RE: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Dan,

One other thing to keep in mind. Jacob's question wasn't meant to be an official NIST position. We hope that anything said by a NIST employee in the forum isn't automatically regarded as speaking for all of NIST. We want to be able to ask questions and discuss ideas in the forum, without having to have the consensus of the entire NIST team before we post. If in doubt - ask us.

Dustin

-----Original Message-----
From: D. J. Bernstein [mailto:djb@cr.yp.to]
Sent: Wednesday, April 25, 2018 11:02 AM
To: pqc-comments <pqc-comments@nist.gov>
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Speaking for myself here, with the objective of highlighting a severe quantitative error by NIST, along with the procedures that led to this error. I don't think an NTRU Prime team comment is needed.

Let me begin with the rules specified in the call for proposals. Each submission is required to contain "an analysis of the algorithm with respect to known attacks". This is separate from the "description of the expected security strength".

The NTRU Prime documentation accordingly does a ton of work to analyze known attacks. I'm not aware of any part of the analysis that requires a correction or update. There's one part that should be worked out in more detail, namely the hybrid attack actually running somewhat more slowly, for reasons identified in the cited Wunderer paper.

Separately from this, the NTRU Prime documentation describes the expected security strength, and the measures taken to protect users against various potential future advances in attacks.

For comparison, most lattice-based submissions skimp on the requirement to analyze known attacks. They instead use massive underestimates of the costs of known attacks---underestimates designed for simplicity rather than accuracy. This might sound safe (even "conservative") but it causes problems in at least four directions:

* Absolute security: These massive underestimates are _constantly_ being misinterpreted and mislabeled as stating actual security levels. Unbroken systems are incorrectly described as broken. Potential users end up thinking that post-quantum crypto is less practical than it actually is.

* Relative security: Different proposals are underestimated by different amounts. I've already given the example of EMBLEM-611 vs. uRound2.KEM-500:

  - One "estimate" says that EMBLEM-611 is hundreds of times easier to break than uRound2.KEM-500 (2^76 vs. 2^84).
  - Another "estimate" says that, no, uRound2.KEM-500 is thousands of

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Thursday, April 26, 2018 8:48 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime
Attachments: signature.asc

Perlner, Ray (Fed) writes:
> You're absolutely right that the complexity of q-core-enum is not
> relevant to security strength 5, since it's a quantum attack.

Even worse: it's a tiny piece of a hypothetical quantum attack. See my previous message for details.

> What's wrong with the Lotus-Enum estimate in your opinion?

Let's skip opinions and focus on the facts.

Look at the LOTUS submission. You'll see that it cites and uses a paper called "A theoretical cost lower bound of lattice vector enumeration", reviewed as pages 17--41 of the submission. The submission says again and again that it is trying to compute _lower bounds_. This means that, when there's a conflict between

* being able to prove that the cost is at least L and
* the accuracy of L as an estimate of the cost,

accuracy is sacrificed. The resulting lower bounds are much smaller than the actual attack costs. The LOTUS submission contains some of these comparisons, along with reminders such as the following:

> Since they are lower bounds, they are much smaller than the other
> works for average complexities such as [28].

The lack of accuracy doesn't prohibit this from being added to the "Underestimate all the LWE schemes!" project, joining various "Core"
formulas (explicit underestimates), various "Q" formulas (ignoring huge overheads), and various "Sieve" formulas (explicit underestimates and ignoring huge overheads).

But, hmmm, the formulas in the LOTUS submission are complicated, so someone instead decided to reuse the name "Lotus" for a simple curve that's claimed to be a fit to the "estimated cost model" in the LOTUS submission. At this point I was expecting

* a pinpoint reference to this "model" in the submission;
* an identification of the curve-fitting mechanism; and
* an analysis of the accuracy of the fit across the parameter range.

As far as I can tell, all of this information is missing. Is the public supposed to blindly trust that "Lotus" is related in some way to what the LOTUS submission says? What exactly is this relationship? Why do the "Lotus" numbers for the LOTUS parameters not match what the LOTUS submission says about those parameters?

The information presented about "Lotus" is so limited that I can't even tell whether it's supposed to be quantum or non-quantum. You say "I didn't see any evidence that LOTUS was estimating the cost of a quantum attack"; I agree that the LOTUS submission focuses on non-quantum analysis; but the web page lists "Lotus" under "quantum enumeration". The paper says that "Lotus" seems to be a "form of Core-Enum", but does this wording exclude "Q-Core-Enum"?

The "Lotus" numbers are between the "Q-Core-Enum" numbers and the "Core-Enum" numbers. If "Lotus" is meant to be non-quantum then the gap below "Core-Enum" is further evidence of the inaccuracy of "Lotus"--- nobody claims that "Core-Enum" is _higher_ than actual costs for this type of algorithm. If "Lotus" is meant to be quantum then it provides similar evidence against the accuracy of "Q-Core-Enum", and at the same time the concrete numbers that it gives are above category 5.

Whatever exactly "Lotus" means, numbers computed by "Lotus" then appear on a web page. Does the web page mention the discrepancies between "Lotus" and the LOTUS submission? No. Does the web page mention that the LOTUS submission was aiming for a _lower bound_ and repeatedly disclaiming the notion that this was the cost of an attack? No.

Instead the web page presents these numbers as "complexity estimates"
for running "attacks". Readers are led to believe, incorrectly, that each of these numbers is the cost of a known attack---that if a number _isn't_ the cost of a known attack then there must be some identifiable error in the analysis producing this number. Readers are also led to believe, incorrectly, that the 14 different numbers are talking about 14 different attacks, meaning that the smallest number is the security level, and that discrepancies don't indicate inaccuracies.

As an example of these effects, when sntrup4591761 shows up as only
2^200 in the "Lotus" estimate, we see two NIST people asking for an explanation. As far as I can tell, these requests rely critically on the belief that there's a contradiction between

* the NTRU Prime documentation saying that the best non-quantum attacks known are in category 5 and

* "Lotus" saying "cost" only 2^200 for sntrup4591761, significantly below category 5 (assuming it's non-quantum and assuming "cost" isn't too many gates).

Is the NTRU Prime team going to drop sntrup4591761 from category 5 to, say, category 3, which according to the vox-populi mechanism of security evaluation is the right security category for dimensions around 768? Or is the NTRU Prime team going to say what's "wrong" with "Lotus"?

Meanwhile the LOTUS submission is saying "lower bound" again and again.
The analysis is claiming that the cost of known attacks _cannot be smaller_ than L. It isn't claiming (and it's repeatedly disclaiming!) the notion that the costs are _as small as_ L. There's no contradiction in the first place; there's nothing for the NTRU Prime team to answer.

Almost three months ago I asked for clear statements of how the "estimates" from "estimate all the LWE schemes!" are supposed to be related to the costs of known attcks:

I see the 2^68.90 "estimate" regarding EMBLEM-611, but I don't see a clear statement of how the thing being estimated is supposed to be related to the security of EMBLEM-611. Is there an attack that's
  * claimed to break EMBLEM-611 and
  * estimated to use 2^68.90 operations?
What exactly are the operations being counted, and what information is available to justify using the estimate?

There's still no answer. In particular, even though I've checked that almost all of the numbers were actually computed as _underestimates_ in the original sources, explicitly sacrificing accuracy---i.e., the original sources do _not_ claim that there are known attacks that run at these speeds---these numbers are being presented and repeated

* without the information that they are underestimates and
* without any explanation of why this information is hidden.

Let me again propose renaming "Estimate all the schemes" as "Underestimate all the schemes". This new name would certainly improve accuracy compared to the old name. A separate table can report the occasional numbers that _weren't_ designed as underestimates.

---Dan

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Monday, April 30, 2018 4:07 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: NTRU Prime
Attachments: signature.asc

I've posted a paper quantifying and justifying a security claim in a small part of the original NTRU Prime paper. This analysis is relevant to

* the implementation of Streamlined NTRU Prime,
* the implementation of NTRU LPRime, and
* the specification of NTRU LPRime.

The same analysis is also relevant to the implementation of Classic McEliece (but I think filing this note as two formal comments would be overkill). It might also be relevant to various other implementations and specifications that randomly permute vectors---in particular, that generate random weight-w vectors---but I see two important caveats here:

* The analysis is specific to "search" problems such as OW-CPA. This is fine for CCA conversions that start from OW-CPA, but some submissions start from IND-CPA instead, and I would expect much larger random numbers to be required for proofs in that context. (I'm not saying that I know corresponding attacks.)

* Each claimed application needs to be checked.

The reason that this analysis ties to the specification for NTRU LPRime, but merely the implementation for Streamlined NTRU Prime, is that

* the "Product NTRU"/LPR approach isn't naturally a deterministic PKE, and typical CCA conversions end up specifying how error vectors are generated, whereas

* the "Quotient NTRU" approach is naturally a deterministic PKE.

The paper is called "Divergence bounds for random fixed-weight vectors obtained by sorting". It's available from the NTRU Prime web page:

https://ntruprime.cr.yp.to/papers.html#divergence

The context is the following. One standard way to randomly permute vectors---e.g., to obtain a random weight-w vector from a standard weight-w vector---is to sort random numbers together with the vectors.
This is easy to implement in constant time with a sorting network: see

https://ntruprime.cr.yp.to/papers.html#ntruprime-paper

specifically Figure S.1.

(For comparison, the NTRU-HRSS-KEM submission claims that fixed-weight vectors are "more difficult to implement in constant time" than a tower of ad-hoc subroutines in that submission. I'm skeptical. Constant-time sorting code is quite straightforward, even if it has to be implemented from scratch and isn't shared with any other applications.)

A vectorized version of a sorting network is very fast when the numbers aren't very big. For example, the two KEMs included in the NTRU Prime submission, sntrup4591761 and ntrulpr4591761, sort 761 30-bit numbers together with a random element of {-1,1}^w + {0}^(761-w) to obtain a random weight-w ternary vector. This takes only about 20000 Haswell cycles with the current software, and only about 10000 cycles with software coming soon.

(In contrast, the NTRU-HRSS-KEM paper claims that its variable-weight vectors are "much more efficient without significantly impacting security". Maybe the cycle counts are below 10000 cycles, but the text "without significantly impacting" suggests that there's some penalty in ciphertext size, and I suspect that the added network delays for this penalty will be far above 10000 cycles. Interestingly, NTRU-HRSS-KEM seems to make the opposite ciphertext-size-vs.-speed decision in its "inverses mod p" discussion.)

Sorting a uniform random vector of _distinct_ numbers produces a uniform random permutation, but it's simpler to skip checking whether the numbers are distinct. There's a noticeable collision probability for 761 30-bit numbers; do the occasional collisions produce an exploitable bias in the output vectors? The NTRU Prime paper says the following:

> We could check for these collisions and restart if they occur, but
> the information leak is negligible.

This is what is quantified and justified in the "divergence" paper. In short, for both sntrup4591761 and ntrulpr4591761, the divergence of all relevant distributions from uniform is shown to be below 1.001, meaning that any search (e.g., an OW-CPA attack) is <1.001 times more likely to be successful.

There isn't supposed to be any particular novelty in this divergence analysis, but I think it's important to write down details of this sort of thing to support auditing.

---Dan

From: John Schanck <jschanck@uwaterloo.ca>
Sent: Monday, May 07, 2018 1:11 PM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime

Dear Dan,

Thank you for this analysis of fixed weight vector sampling.
We have several comments on your comparison with NTRU-HRSS-KEM.

D. J. Bernstein wrote:
> (In contrast, the NTRU-HRSS-KEM paper claims that its variable-weight
> vectors are "much more efficient without significantly impacting
> security". Maybe the cycle counts are below 10000 cycles, but the text
> "without significantly impacting" suggests that there's some penalty
> in ciphertext size, and I suspect that the added network delays for
> this penalty will be far above 10000 cycles. [...])

That is not the reading of "without significantly impacting security"
that we had in mind [1], but your point is well taken. One can use fixed weight sampling to reduce ciphertext size, and this reduction in size could offset the cost of fixed weight sampling. To identify a meaningful "penalty in ciphertext size" one needs to fix the weight parameter, since low weight vectors are a security risk.

If we had considered fixed weight vectors for ntruhrss701 we would have fixed the weight close to 2n/3 (the expected weight of uniform ternary vectors). Our correctness condition would then require q > 5279, and we would have two options:

- Continue to use q=8192, with no change in ciphertext size.
- Switch to prime q, take q=5303, and save < 55 bytes.

Experiments would be needed to say whether a 55 byte savings in ciphertext size would offset the added cost of fixed weight sampling, prime q, and Z/q-encoding routines.

Going further afield from our proposal, we could follow Streamlined NTRU Prime and optimize for ciphertext size instead of combinatorial security. We might then choose one of the following parameter sets

- q=4591, weight 3n/5 (matching q of sntrup4591761),
- q=4096, weight n/2,
- q=2999, weight 3n/8 (matching weight of sntrup4591761).

These save < 73, 88, and < 127 bytes respectively. They are listed in order of decreasing security with respect to an enumeration based hybrid attack.

The point, in listing these alternatives, is that this discussion must include the weight parameter and an analysis of its impact on combinatorial security. We would be happy to hear that fixed weight sampling is efficient on a variety of platforms, and that there are no serious risks in using weight n/2 vectors. We have not yet been convinced that this is the case.

All the best,
John (on behalf of the NTRU-HRSS-KEM team).

[1] The security loss implied by "without significantly impacting security" is relative to NTRU-HRSS with n=701, q=8192, and uniform ternary (or fixed weight 2n/3) f, g, r and m. We quantify this loss in our security analysis: the Core-SVP analysis suggests BKZ-466 when f, g, r, and m are sampled using our proposed sampler; it suggests BKZ-470 when f, g, r, and m are uniform ternary.

--

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Saturday, May 12, 2018 1:00 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: NTRU Prime
Attachments: graph.pdf; signature.asc

John Schanck writes:
> we could follow Streamlined NTRU Prime and optimize for ciphertext
> size instead of combinatorial security.

Actually, what the NTRU Prime submission says is that its recommended parameters for sntrup "provide an excellent tradeoff between size and security level" (and that the analysis of ntrulpr parameter sets "works the same way").

See the graph that I've attached to this message, showing size (specifically p log_2 q) vs. security level for some of the sntrup parameter choices listed in the NTRU Prime paper. One can immediately see from this graph that certain parameter choices are particularly good---even better than Pareto-optimal.

It's wrong to describe this as optimizing size "instead of" security.
Both size and security are taken into account.

(As a side note, the parameter-selection strategy used here is the same as the strategy used to identify, e.g., Curve25519. Of course the primary performance metric is different: ECC cost is dominated by CPU time, while lattice cost is dominated by size.)

In light of this picture, here are some comments on the switch from variable weights to fixed weights. We all seem to agree that this improves each size-vs-security-level data point (for zero decryption failures; maybe this isn't true for other failure levels, but both of the submissions under discussion here deliberately eliminate all failures). On the other hand, maybe generating variable-weight vectors is cheaper than generating fixed-weight vectors. So there are two possibly competing effects to quantify:

* What is the exact effect on CPU time?

* What is the exact effect on the size-vs-security-level data points?

The limited evidence available today suggests that both of these effects are fairly small:

* I've been giving numbers supporting the idea that the effect on CPU time is small. This is where the NTRU-HRSS-KEM paper claims that variable weight is "much" more efficient, without giving any numbers or any other justification.

* John has been giving numbers supporting the idea that the effect on the size-vs-security-level data points is small. This is where the NTRU-HRSS-KEM paper says that the security impact is not "significant", again without giving any numbers.

Perhaps in the end the conclusion will be that both effects are too small for any users to care. However, as long as there are claims to the contrary (e.g., "much more efficient"), it seems necessary to pursue quantification. Also, experience with ECC shows that there's value in optimizing very small choices as a principled mechanism to limit the potential influence of malicious parties on the standardization process.

One way to put size and CPU time on the same scale (as in the NTRU Prime paper) is to consider a quad-core 3GHz CPU handling a 100Mbps Internet connection:

* In 1 millisecond, each core runs 3 million cycles, for a total of 12 million cycles across the 4 cores---enough time for more than 100 of the cryptographic operations that we're talking about.

* In the same 1 millisecond, the Internet connection transmits only 12500 bytes, an order of magnitude fewer ciphertexts.

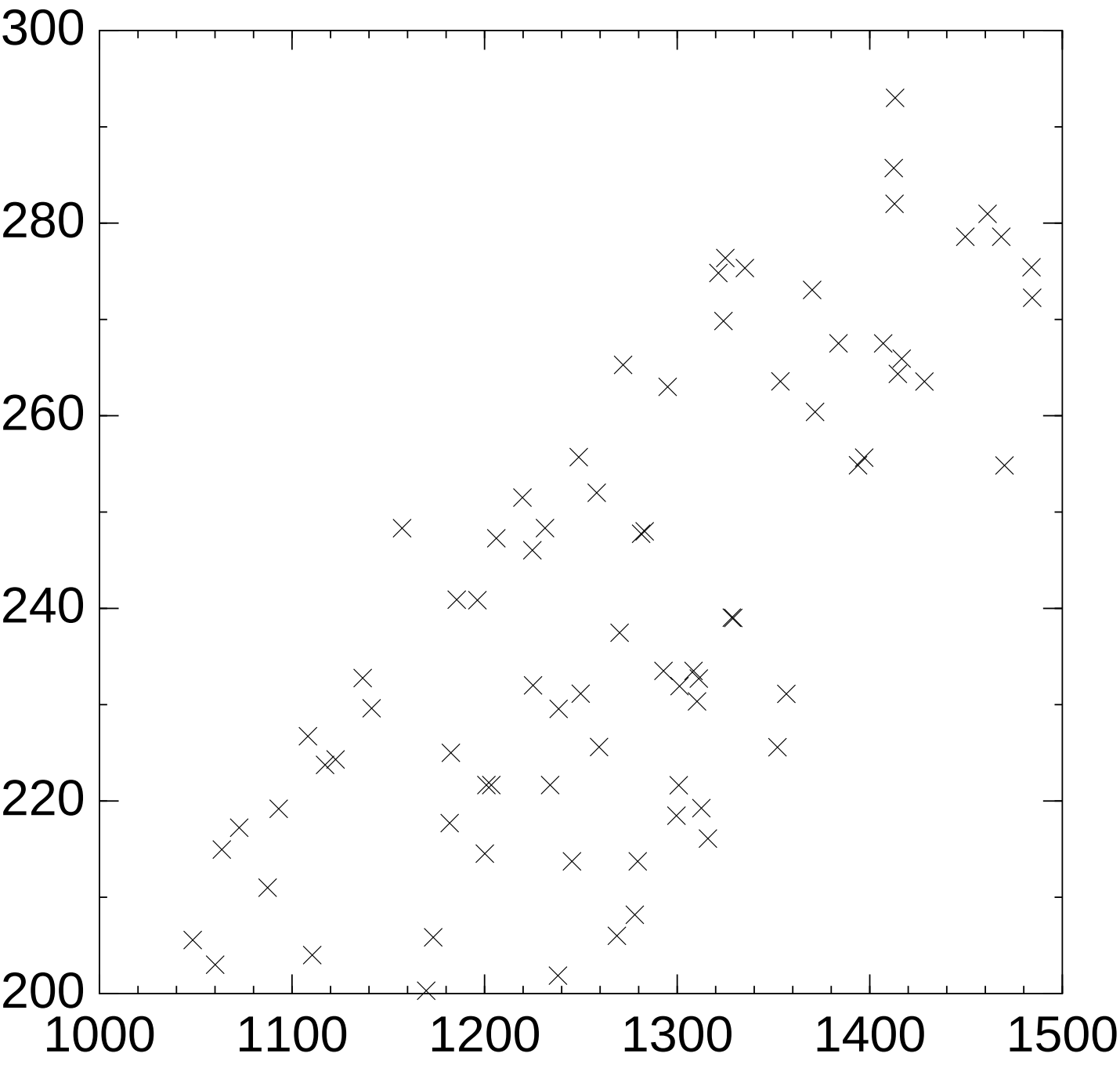
From this perspective, eliminating 1% of ciphertext size is as important as eliminating 10% of CPU time. Of course the exact ratio changes if one varies the CPU speed and the network speed.

When a cryptosystem modification (e.g., switching from variable weight to fixed weight) affects both cost _and_ security, it's standard to see the impact by

* considering the cost change at specific security levels, or
* considering the security change at specific cost levels, or
* considering the change in the cost-vs-security tradeoff graph.

The first two mechanisms seem to be problematic for NTRU-HRSS-KEM, since NTRU-HRSS-KEM has a relatively sparse set of parameters. Concretely, my impression is that the proposed dimension 701 is at the edge of what would typically be called a "performance cliff" in NTRU-HRSS-KEM, and this also creates relatively large discontinuities in the effect of modifications. (Obviously the same comparison difficulties are even larger for systems that are limited to dimensions {512,768,1024}.) A two-dimensional cost-vs-security graph for NTRU-HRSS-KEM would help clarify the picture.

---Dan


300
280
260
240
220
200
1000
1100
1200
1300
1400
1500

From: EL HASSANE LAAJI <e.laaji@ump.ac.ma>
Sent: Friday, May 18, 2018 9:56 AM
To: pqc-comments; pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: NTRU Prime

Hi, NTRUprime Authors;
I installed your implementation, NTRUprime_streamlined_4591-761, in Window7-DevC++ environnement .I just changed :(for replacing crypto_int16.h...)

```
typedef char small;
typedef int modq;
typedef int crypto_int16;
typedef unsigned int crypto_uint16;
typedef long crypto_int32;
typedef unsigned long crypto_uint32;
```

the keygeneration and encryption work goodlly but the decryption failed, it return(-1).
what is the problem, is there others thing to change.
thanks
Best regards

Kerman, Sara J. (Fed)

From: Olga Akolzina <4akolzinaolga@gmail.com>
Sent: Wednesday, June 20, 2018 9:10 AM
To: pqc-forum
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: NTRU Prime

Dear William Whyte!

Thank you very much for your positive evaluation of our work!

In accordance with your recommendations, we continued our work in the field of polynomial multiplication function for NTRU-similar algorithms that use the properties of a small polynomial and computation time for them doesn't depend on this polynomial.

The following results were obtained.

Processor Intel (R), Core (TM) i5-4440 CPU @ 3.1 Ghz.

Programming languages: C, Assembler, 64 bits

Small polynomial representation in the form of nonzero coefficients set is:

The polynomial for NTRUPrime is (N = 761, q = 4591, d = 125)

Precalculations is 1540 tacts, depend only on the second (open) factor. The more often can be done in advance.

Multiplication is 12856 cycles.

The deviation from the mean doesn't exceed 2.12% and no more than the measurement error.

Small polynomial representation is in the form A1 * A2 + A3

The polynomial: N = 743, D1 = 11, D2 = 11, D3 = 15, q = 2048

Multiplication is 10809 cycles. There is no precomputation.

The execution time deviation from the mean is ± 1.8559 and no more than the measurement error

Best regards,
I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina

2018-03-28 13:25 GMT+03:00 <4akolzinaolga@gmail.com>:

> More precise research results
>
> 1. For 10,000 keys' maximum dispersion of all measurements is