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