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?

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.
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 indices 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

--

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/.
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 $(f_1*f_2) + f_3$, as described in

J. Hoffstein, J.H. Silverman, *Random Small Hamming Weight Products with applications to cryptography*  

... 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
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 \((f_1*f_2)+f_3\).

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 \((f_1*f_2) +f_3\), 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,
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%2Fwww.cr.yp.to%2Fntruprime&data=02%7C01%7Csara.kerman%40nist.gov%7C980b212c6f2e4882159608d589be1681%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C1%7C636566371878471392&sdata=%2FKSgNVObQ4YrLrgJhXqEt8T7P46fnkWelk35mkz4%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.

--

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/.
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.
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
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%.

>> 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

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
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

```c
#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>

--

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/.