From: zhenfei <zzhang@onboardsecurity.com>
Sent: Tuesday, January 23, 2018 10:32 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: NTRUEncrypt
Attachments: signature.asc

Dear all,

We would like to thank Jingnan He and Xianhui Lu for pointing out a bug in our code.

In the discrete Gaussian sampling algorithm ntru-pke-1024/DGS.c

```
void DGS (     int64_t   *v,      /* output   vector */
const uint16_t  dim,     /* input    dimension */
const uint8_t   stdev)   /* input    standard deviation */,
```

where the stdev is 724 and therefore requires more than 8 bits to store.

We have fixed this bug. The updated code will be available at

https://www.onboardsecurity.com/nist-post-quantum-crypto-submission

Best regards,

The NTRU team

From: hassan LAAJI <hmhingenieur@gmail.com>
Sent: Sunday, March 25, 2018 11:11 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: NTRUEncrypt

Hello, i'm very happy to contact you about your implementation in NIST.
I have a remark in your implementation:
perhaps with your compiler version it works goodly but in mine i find problem:
in file dgs.h :

```
void DGS (      int64_t   *v,       /* output   vector */
        const uint16_t  dim,      /* input    dimension */
        const uint64_t  stdev)   /* input    standard deviation */
{
```

and

```
 void DDGS (      int64_t  *v,
        const uint16_t  dim,
        const uint64_t  stdev,
        unsigned char   *seed,
              size_t  seed_len)
```

but in file poly.h :

```
void DGS (
        int64_t   *v,
    const uint16_t  N,
    const uint16_t  stdev);
```

```
/* deterministic DGS */
void DDGS (      int64_t  *v,
        const uint16_t  dim,
        const uint64_t  stdev,
        unsigned char   *seed,
            uint16_t  seed_len);
```

You must do the same types of parameters functions in both files : dgs.h ; poly.h
it work when i changed in both files:

```
oid DGS (      int64_t   *v,       /* output   vector */
        const uint16_t  dim,      /* input    dimension */
        const uint16_t  stdev)   /* input    standard deviation */
{
```

and

```
 void DDGS (      int64_t  *v,
        const uint16_t  dim,
        const uint64_t  stdev,
        unsigned char   *seed,
              uint16_t  seed_len)
```

Best regards

From: EL HASSANE LAAJI <e.laaji@ump.ac.ma>
Sent: Thursday, April 05, 2018 5:58 PM
To: pqc-comments; pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: NTRUEncrypt

Hi Researchers;

About document title: NTRUencrypt A Lattice Based Cryptography Algorithm: Page 5, algorithm 3 NTRU-pke Decrypt. I want to ask you if in line 2, it is **t=c-m ,** or **t=c-m' ?** Because **m** will be computed in line 6.

The same for algorithm 8.

Best regards.

From: EL HASSANE LAAJI <e.laaji@ump.ac.ma>
Sent: Monday, April 09, 2018 7:32 AM
To: pqc-comments; pqc-forum@list.nist.gov
Subject: Re: OFFICIAL COMMENT: NTRUEncrypt

Hi,
I have another remark:
- in doc :algorithm 1 you compute the public key as: h=g/(pf+1)
- but in EESS section 9.1.1 you compute the public key as: h=p* g/(pf+1) or like you wrire: h=f^(-1) *g*p where f=1+pF
Why you add p factor ?

Best regards

Le jeudi 5 avril 2018, EL HASSANE LAAJI <e.laaji@ump.ac.ma> a écrit :

> Hi Researchers;
>
> About document title: NTRUencrypt A Lattice Based Cryptography Algorithm: Page 5, algorithm 3 NTRU-pke Decrypt. I want to ask you if in line 2, it is **t=c-m ,** or **t=c-m' ?** Because **m** will be computed in line 6.
>
> The same for algorithm 8.
>
> Best regards.

From: Zhenfei Zhang <zzhang@onboardsecurity.com>
Sent: Monday, April 09, 2018 8:04 AM
To: EL HASSANE LAAJI
Cc: pqc-comments; pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: NTRUEncrypt

Hi El,

> About document title: NTRUencrypt A Lattice Based Cryptography Algorithm: Page 5, algorithm 3 NTRU-pke Decrypt. I want to ask you if in line 2, it is **t=c-m ,** or **t=c-m' ?** Because **m** will be computed in line 6.

> The same for algorithm 8.

Thanks for pointing this out. Yes it is a typo - should be t = c - m'.

- in doc :algorithm 1 you compute the public key as: h=g/(pf+1)
- but in EESS section 9.1.1 you compute the public key as: h=p* g/(pf+1) or like you wrire: h=f^(-1) *g*p where f=1+pF

It's an inconsistency of the description in the report and the EESS1 spec.
In the end we will be using pg/(pf+1) in the encryption scheme - it doesn't really matter if we encode g/(pf+1) or pg/(pf+1) in the public key.
In the report we slightly changed it so that both NTRUEncrypt and pqNTRUSign uses a same key gen function
- in pqNTRUSign we no longer have the p factor for the public key.

On that note, I just noticed an another typo: algorithm 2, line 5, it should be: t = p*r*h.

Regards,
Zhenfei

On Mon, Apr 9, 2018 at 7:31 AM, EL HASSANE LAAJI <e.laaji@ump.ac.ma> wrote:

> Hi,
> I have another remark:
> - in doc :algorithm 1 you compute the public key as: h=g/(pf+1)
> - but in EESS section 9.1.1 you compute the public key as: h=p* g/(pf+1) or like you wrire: h=f^(-1) *g*p where f=1+pF
> Why you add p factor ?
>
> Best regards

Le jeudi 5 avril 2018, EL HASSANE LAAJI <e.laaji@ump.ac.ma> a écrit :

> Hi Researchers;
>
> About document title: NTRUencrypt A Lattice Based Cryptography Algorithm: Page 5, algorithm 3 NTRU-pke Decrypt. I want to ask you if in line 2, it is **t=c-m ,** or **t=c-m' ?** Because **m** will be computed in line 6.
>
> The same for algorithm 8.
>
> Best regards.

From: Zhenfei Zhang <zzhang@onboardsecurity.com>
Sent: Tuesday, June 05, 2018 3:45 PM
To: pqc-forum
Subject: [pqc-forum] NTRUEncrypt

Hi all,

We would like to report that we have fixed the bugs reported in the email.
Last week, we were also informed by Ray and Dustin about a bug in key generations.
In our code, the fixed weight sparse polynomial generation function within key gen does not always return a fixed weight polynomial with balanced number of +/- 1s.
We have also fixed this bug in this revision.
For the latest version of our code please see:
https://github.com/NTRUOpenSourceProject/ntru-crypto/tree/master/NIST

Regards,
Zhenfei

On Sat, May 19, 2018 at 11:15 AM, Zhenfei Zhang <zzhang@onboardsecurity.com> wrote:

> Hi Markku,
>
> Thanks again for the reminder.
> We do have a patch which was supposed to be available at our website.
> I make sure they are available next week.
>
> Cheers,
> Zhenfei
>
> On Sat, May 19, 2018 at 10:46 AM, Markku-Juhani O. Saarinen <mjos.crypto@gmail.com> wrote:
>
> > Hi,
> >
> > The reference implementation of NTRUEncrypt KEM-1024 does not work -- the encryption and decryption parts do not generate the same shared secret.
> >
> > I notified the design team more about this more than a month ago, and they rapidly acknowledged the problem, but I haven't seen a bugfix yet.
> >
> > I don't know what precisely is causing this but there is at least one apparent bug in file NTRUEncrypt/Reference_Implementation/ntru-kem-1024/NTRUEncrypt.c, function mask_m():
> >
> > ```
> > 274:   /* extract the last bit of rh */
> > 275:   for (i=0;i<LENGTH_OF_HASH*2;i++)
> > 276:   {
> > 277:      seed[i] = (rh[i*8] & 1);
> > 278:      for (j=1;j<8;j++);
> > 279:      {
> > 280:         seed[i] <<= 1;
> > 281:         seed[i] += (rh[i*8+j] & 1);
> > 282:      }
> > 283:   }
> > ```

Note the semicolon at the end of line 278 -- this is not a loop, it just sets j=8 and executes the following bit on lines 280-281 once.

The KAT files are probably useless as the error appears to be on the encrypt side.

The smaller variants of NTRUEncrypt (KEM-443 and KEM-743) do successfully encrypt/decrypt. The corresponding function of these variants looks little different so it is not obvious to me how to correct the error.

Cheers,
- markku

--
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: Kaufman, Charlie <Charlie.Kaufman@dell.com>
Sent: Saturday, June 30, 2018 12:19 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: NTRUEncrypt

I have some questions about the pad function used by NTRU Encrypt as specified in the NTRUEncrypt.pdf document that accompanied the submission. The document says the following:

> The encryption algorithm in Algorithm 2 uses a padding method to deal with potential insufficient entropy in a message. Assuming the message length is valid and less than (N − 173) bits , the padding algorithm works as follows:
> 1. Convert msg into a bit string. Each bit forms a binary coefficient for the lower part of the polynomial m, starting from coefficient 0.
> 2. The last 167 coefficients of m(x) are randomly chosen from {−1, 0, 1} (with an input seed). This gives over 256 bits entropy.
> 3. The length of msg is converted into an 8 bit binary string, and forms the last 173 to 168 coefficients of m(x).

It says the length of the message is converted into an 8 bit binary string and fills coefficients 173 – 168. But that is only 6 coefficients, not 8.
And an 8 bit binary string can only encode a value up to 255, while the message can be as long as N-174 bits. So either the message has to be a multiple of 8 bits, and this is the length in bytes, or the intent is that the length is encoded as 6 ternary digits (allowing a value up to 728), or something else. What was the intended behavior?

It also does not say what is placed in the ternary digits that would have been occupied by the message had the message been longer. It might be they should all be zeros, or random values of 0 or 1, or random values of 0, 1, or -1. What was the intended behavior?

Finally, when the message is being extracted from its padded encoding, it does not say what an implementation is supposed to do if the length is larger than the encoding allows and whether the implementation is supposed to check that the padded digits have any particular values. What was the intended behavior?

Thank you!
Charlie Kaufman
(charlie.kaufman@dell.com)

From: Zhenfei Zhang <zzhang@onboardsecurity.com>
Sent: Tuesday, July 10, 2018 12:22 PM
To: Perlner, Ray (Fed); charlie.kaufman@dell.com
Cc: pqc-forum
Subject: Re: [pqc-forum] FW: OFFICIAL COMMENT: NTRUEncrypt

> There should be no overlapping. The message should not be longer than msg_len which is set to floor((N-175)/8)

I meant to say "max_msg_len" which is set to floor((N-175)/8)

Zhenfei

On Tue, Jul 10, 2018 at 12:19 PM, Zhenfei Zhang <zzhang@onboardsecurity.com> wrote:

Hi,

Please see my reply in line.

I have some questions about the pad function used by NTRU Encrypt as specified in the NTRUEncrypt.pdf document that accompanied the submission. The document says the following:

> The encryption algorithm in Algorithm 2 uses a padding method to deal with potential insufficient entropy in a message. Assuming the message length is valid and less than (N − 173) bits , the padding algorithm works as follows:
> 1. Convert msg into a bit string. Each bit forms a binary coefficient for the lower part of the polynomial m, starting from coefficient 0.
> 2. The last 167 coefficients of m(x) are randomly chosen from {−1, 0, 1} (with an input seed). This gives over 256 bits entropy.
> 3. The length of msg is converted into an 8 bit binary string, and forms the last 173 to 168 coefficients of m(x).

It says the length of the message is converted into an 8 bit binary string and fills coefficients 173 – 168. But that is only 6 coefficients, not 8.
And an 8 bit binary string can only encode a value up to 255, while the message can be as long as N-174 bits. So either the message has to be a multiple of 8 bits, and this is the length in bytes, or the intent is that the length is encoded as 6 ternary digits (allowing a value up to 728), or something else. What was the intended behavior?

Sorry. There is a typo. It should be 175 rather than 173.
The message is encoded as

msg | msg_len | pad

where
* pad is 167 bits of trinaries that gives 256 bits entropy.
* msg_len is 8 bits; and length is in bytes, i.e., (N-175)/8 which is less than 100 bytes for all parameter sets (so even if it were 6 bits it is still sufficient)

It also does not say what is placed in the ternary digits that would have been occupied by the message had the message been longer. It might be they should all be zeros, or random values of 0 or 1, or random values of 0, 1, or -1. What was the intended behavior?

There should be no overlapping. The message should not be longer than msg_len which is set to floor((N-175)/8)

Finally, when the message is being extracted from its padded encoding, it does not say what an implementation is supposed to do if the length is larger than the encoding allows and whether the implementation is supposed to check that the padded digits have any particular values. What was the intended behavior?

It ignores the padding (we should have checked the # of +/- 1s here); extract the msg_len; and then extract the first msg_len*8 coefficients from the padded message.

Cheers,
Zhenfei

On Tue, Jul 10, 2018 at 11:58 AM, Perlner, Ray (Fed) <ray.perlner@nist.gov> wrote:

> It seems this didn't make it to the forum but was meant to.
>
> -----Original Message-----
> From: Kaufman, Charlie [mailto:Charlie.Kaufman@dell.com]
> Sent: Saturday, June 30, 2018 12:19 AM
> To: pqc-comments <pqc-comments@nist.gov>
> Cc: pqc-forum@list.nist.gov
> Subject: OFFICIAL COMMENT: NTRUEncrypt
>
> I have some questions about the pad function used by NTRU Encrypt as specified in the NTRUEncrypt.pdf document that accompanied the submission. The document says the following:
>
> > The encryption algorithm in Algorithm 2 uses a padding method to deal
> > with potential insufficient entropy in a message. Assuming the message
> > length is valid and less than (N − 173) bits , the padding algorithm
> > works as follows:
> > 1. Convert msg into a bit string. Each bit forms a binary coefficient
> > for the lower part of the polynomial m, starting from coefficient 0.
> > 2. The last 167 coefficients of m(x) are randomly chosen from {−1, 0, 1}
> > (with an input seed). This gives over 256 bits entropy.
> > 3. The length of msg is converted into an 8 bit binary string, and forms
> > the last 173 to 168 coefficients of m(x).
>
> It says the length of the message is converted into an 8 bit binary string and fills coefficients 173 – 168. But that is only 6 coefficients, not 8.
> And an 8 bit binary string can only encode a value up to 255, while the message can be as long as N-174 bits. So either the message has to be a multiple of 8 bits, and this is the length in bytes, or the intent is that the length is encoded as 6 ternary digits (allowing a value up to 728), or something else. What was the intended behavior?
>
> It also does not say what is placed in the ternary digits that would have been occupied by the message had the message been longer. It might be they should all be zeros, or random values of 0 or 1, or random values of 0, 1, or -1. What was the intended behavior?
>
> Finally, when the message is being extracted from its padded encoding, it does not say what an implementation is supposed to do if the length is larger than the encoding allows and whether the implementation is supposed to check that the padded digits have any particular values. What was the intended behavior?

From: EL HASSANE LAAJI <e.laaji@ump.ac.ma>
Sent: Saturday, July 28, 2018 4:43 PM
To: pqc-comments; pqc-forum
Subject: [pqc-forum] Re: OFFICIAL COMMENT: NTRUEncrypt

Hi Researchers;
I give you two remarques:
1- In your implementation NTRencrypt_1024: it is not necesseary to recompute fntt in Decryption function because keygenration function and the decryption function both are beside the server. that increase the of decryption about 200%(in my PC 40 ms rather than 80ms).
2- in your implementation NTRUencrypt_743 : in trinay polynomial generation :the type of **f** is **(uint16_t *) that's means all coefficients are positive.**

```
 /* generate a trinary polynomial with fixed number of +/- 1s */
void  trinary_poly_gen(
      uint16_t  *f,
   const uint16_t  N,
   const uint16_t  d)

/.....
if (f[coeff[i]]==0)
          {
            f[coeff[i]]=-1;
            count++;
          }
/.....
```

the same for others functions.
best regards

2018-04-05 22:58 GMT+01:00 EL HASSANE LAAJI <e.laaji@ump.ac.ma>:

> Hi Researchers;
>
> About document title: NTRUencrypt A Lattice Based Cryptography Algorithm: Page 5, algorithm 3 NTRU-pke Decrypt. I want to ask you if in line 2, it is **t=c-m ,** or **t=c-m' ?** Because **m** will be computed in line 6.
>
> The same for algorithm 8.
>
> Best regards.

From: Zhenfei Zhang <zzhang@onboardsecurity.com>
Sent: Sunday, July 29, 2018 1:26 PM
To: EL HASSANE LAAJI
Cc: pqc-comments; pqc-forum
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: NTRUEncrypt

HI,

1- In your implementation NTRencrypt_1024: it is not necesseary to recompute fntt in Decryption function because keygenration function and the decryption function both are beside the server. that increase the of decryption about 200%(in my PC 40 ms rather than 80ms).

Yes you may store the ntt form of f, so that the decryption will be faster.
However, not that the f polynomial is significantly smaller than its ntt form.
So it will be the choice of the user to decide if they want to store f (for space) or fntt (for efficiency).

2- in your implementation NTRUencrypt_743 : in trinay polynomial generation :the type of **f** is **(uint16_t *) that's means all coefficients are positive.**

f is a trinary polynomial over a polynomial ring mod 2^12. It really doesn't matter if it is stored in an uint16_t or an int16_t since we will perform mod 2^12 eventually.
In our code we choose to use uint16_t for all polynomials so we don't need to care for the signs the whole time.

Zhenfei

On Sun, Jul 29, 2018 at 12:58 PM, EL HASSANE LAAJI <e.laaji@ump.ac.ma> wrote:

> 2018-07-28 21:42 GMT+01:00 EL HASSANE LAAJI <e.laaji@ump.ac.ma>:
> Hi Researchers;
> I give you two remarques:
> 1- In your implementation NTRencrypt_1024: it is not necesseary to recompute fntt in Decryption function because keygenration function and the decryption function both are beside the server. that increase the of decryption about 200%(in my PC 40 ms rather than 80ms).
> 2- in your implementation NTRUencrypt_743 : in trinay polynomial generation :the type of **f** is **(uint16_t *) that's means all coefficients are positive.**
> /* generate a trinary polynomial with fixed number of +/- 1s */
> void trinary_poly_gen(
> uint16_t *f,
> const uint16_t N,
> const uint16_t d)
>
> /.....
> if (f[coeff[i]]==0)
> {
> **f[coeff[i]]=-1;**
> count++;
> }
> /.....
> the same for others functions.
> best regards