From: Yongge Wang <yongge.wang@gmail.com>
Sent: Sunday, December 24, 2017 12:59 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: HK17

Dear HK17 Designer and all,

I have two comments on this submission.

1. This submission is a Key Agreement Protocol (DH style) and seems not fall into any of the NIST PQC CFP categories (pk signature, pk encryption, kem).

2. I think the protocol may not be correct (or am I missing something?) The protocol arithmetic operations are over quaternion/octonion. So we need to be aware of the fact that the quaternions are not commutative and the octonions are neither commutative nor associative. In the protocol specification, Section 3.1 (Computing Session Keys step i), it claims that k_A=k_B. The proof for this correctness fact is as follows:

k_A = f'(q_A)^m . h'(q_A)^r . q_B . h'(q_A)^s . f'(q_A)^n = h'(q_A)^r . f'(q_A)^m . q_B . f'(q_A)^n . h'(q_A)^s = k_B

In the above proof, obviously the "commutative" property is used. As we have mentioned, the commutative property does not hold for quaternion (neither for octonion). So the correctness proof is invalid.

The designer has given some numerical examples.. I did not check it.. but I am not sure why the numerical example actually is correct without the commutative property.

thanks!
Yongge

From: Yongge Wang <yongge.wang@gmail.com>
Sent: Sunday, December 24, 2017 1:25 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: HK17

Furthermore, the implementation may not be correct since the proposal claims
that in the implementation, "no big number libraries needed."
(both implementation contains no big number libraries)

It should be noted that in the computation of
f'(q_A)^m . h'(q_A)^r . q_B . h'(q_A)^s . f'(q_A)^n we have f'(q_A)=(a,b,c,d) where
a/b/c/d are in the format of 0.xxxx and the polynomial f has degree d=16 or 32.
Since m is any integer (e.g., 245 as in the numerical example). Thus f'(q_A)^m
is in the format of XXX.xxxxxx...xxxx, where there may be 245x16x4=15680
fractional digits... this will obviously not work on any 32-bit CPUs if one does not use any special numeric package. In the proposal, the examples round each number to 4-fractional digits during computation... but this will make the equation non-valid (the round takes places at different steps at Alice or Bob side).

thanks!
Yongge

On Sun, Dec 24, 2017 at 8:59 PM, Yongge Wang <yongge.wang@gmail.com> wrote:

> Dear HK17 Designer and all,
>
> I have two comments on this submission.
>
> 1. This submission is a Key Agreement Protocol (DH style) and seems not fall into any of the NIST PQC CFP categories (pk signature, pk encryption, kem).
>
> 2. I think the protocol may not be correct (or am I missing something?) The protocol arithmetic operations are over quaternion/octonion. So we need to be aware of the fact that the quaternions are not commutative and the octonions are neither commutative nor associative. In the protocol specification, Section 3.1 (Computing Session Keys step i), it claims that k_A=k_B. The proof for this correctness fact is as follows:
>
> k_A = f'(q_A)^m . h'(q_A)^r . q_B . h'(q_A)^s . f'(q_A)^n = h'(q_A)^r . f'(q_A)^m . q_B . f'(q_A)^n . h'(q_A)^s = k_B
>
> In the above proof, obviously the "commutative" property is used. As we have mentioned, the commutative property does not hold for quaternion (neither for octonion). So the correctness proof is invalid.
>
> The designer has given some numerical examples.. I did not check it.. but I am not sure why the numerical example actually is correct without the commutative property.
>
> thanks!
> Yongge

From: Yongge Wang <yongge.wang@gmail.com>
Sent: Monday, December 25, 2017 9:59 AM
To: pqc-forum@list.nist.gov; pqc-comments
Subject: Re: [pqc-forum] OFFICIAL COMMENT: HK17

On Mon, Dec 25, 2017 at 3:00 PM, D. J. Bernstein <djb@cr.yp.to> wrote:

> Tanja Lange and I have been discussing this, and our impression is that
> the stated identity follows from the Moufang identities. It's important
> to note that f and h are evaluated at the same quaternion or octonion.
> The submitters should spell out the details.

thanks for pointing out this.. that is right.. if both f and h evaluate on the same quaternion or octonion, then the commutative law works there..

> > f'(q_A)^m is in the format of XXX.xxxxxx...xxxx, where
> > there may be 245x16x4=15680 fractional digits...
> The implementation reduces modulo (e.g.) 251 at each step, and this is
> compatible with the definition of the shared secret.

OK, this may be one potential interpretation of the missing part in the proposal. But it may not be something that the submitter has in mind? From the numerical example, it shows that each time, when secret is derived using mod operation, all the fractional part is simply dropped (not rounded)... But the computation of r_A/r_B in the numerical example has four digit fractional part. So I think during the intermediate steps for computing r_A/r_B, even if mod operation is done, it is different from the mod operation in the secret derivation step.

E.g., in the K_B computation process, 0.3184 mod 215 = 79. This is obtained by using the fact 0.3184x251=79.9184. Note that here 0.9184 is dropped completely..

Even if the mod operation is defined correctly
in the intermediate steps, there are issues... e.g.,
(2*0.1021) mod 251 != 2*(0.1021 mod 251) mod 251

the reason is
(2*0.1021) mod 251 =0.2042*251 mod 251 = 51.2291 mod 251 = 51
2*(0.1021 mod 251) mod 251 = 2*(0.1021*251 mod 251) mod 251=50

thank!
Yongge

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Monday, December 25, 2017 12:16 PM
To: pqc-forum@list.nist.gov
Cc: pqc-comments
Subject: OFFICIAL COMMENT: HK17

Dear designers, dear all,

The following attack script breaks HK17 for all proposed parameters.
Specifically, this script breaks the Python key-exchange implementation included in the HK17 submission. This key-exchange implementation appears to match the intent of the HK17 documentation, except that the documentation includes a normalization step; this step does not affect the attack.

This attack takes p+1 simple computations (and is a search so Grover's algorithm is applicable, but all proposed parameters are small enough to be broken by a non-quantum attack). For comparison, the submission says

* 2^64 pre-quantum security for p=251,
* 2^128 pre-quantum security for p=65521,
* 2^256 pre-quantum security for p=4294967291, and
* 2^512 pre-quantum security for p=18446744073709551557.

Our attack takes about 2^8, 2^16, 2^32, and 2^64 simple computations for these parameter sets.

For simplicity the attack script focuses on the case that the public key rA is invertible, which occurs almost all of the time. Slightly more work should be able to handle the occasional exceptions.

To use this script, save the following Python code as break.py; copy octonions.py from the HK17 submission to octonions.py in this directory; copy HK17-O.py from the HK17 submission to ref.py in this directory; and run "python break.py".

---Daniel J. Bernstein and Tanja Lange

```
import octonions
import ref
import sys

p = ref.modulo
print ref.message
print ref.times

print "eve observes public parameters and alice's public key:"
print 'oa =',ref.oa
print 'ob =',ref.ob
print 'rA =',ref.rA

def modprecip(x):
  x %= p
```

```
  if x == 0: raise Exception('dividing by 0')
  return pow(x,p-2,p)

def octonionrecip(x):
  xnormsq = sum(xi**2 for xi in x) % p
  xconj = (x[0],-x[1],-x[2],-x[3],-x[4],-x[5],-x[6],-x[7])
  return octonions.scale(xconj,modprecip(xnormsq),p)

try:
  rArecip = octonionrecip(ref.rA)
except:
  raise Exception('public key is not invertible, skipping this case for simplicity')

try:
  obrecip = octonionrecip(ref.ob)
except:
  raise Exception('ob is not invertible, should have caught from rA test') assert octonions.multiply(obrecip,ref.ob,p) ==
(1,0,0,0,0,0,0,0)

# goal: write rA as x*ob*y for some x,y in \F_p[oa] # this forces x to be in \F_p + oa\F_p # wlog take x to be 1 or in \F_p +
oa for i in range(p):
  for j in [0,1]:
    if j == 0 and i != 1: continue
    x0 = (i,0,0,0,0,0,0,0)
    x1 = octonions.scale(ref.oa,j,p)
    x = octonions.summ(x0,x1,p)
    try:
      xrecip = octonionrecip(x)
    except:
      continue
    t = octonions.multiply(xrecip,ref.rA,p)
    # goal: write t as ob*y for some y in \Z[oa]
    y = octonions.multiply(obrecip,t,p)
    if octonions.multiply(y,ref.oa,p) == octonions.multiply(ref.oa,y,p):
      print "eve's secret key:",x,y
      print "now eve looks at bob's ciphertext (DH public key):"
      print 'rB =',ref.rB
      k = octonions.multiply(x,octonions.multiply(ref.rB,y,p),p)
      print "eve's session key =",k
      print 'now peek at secrets to verify attack worked:'
      print "alice's session key =",ref.k1
      print "bob's session key =",ref.k2
      sys.exit(0)
```

From: Yongge Wang <yongge.wang@gmail.com>
Sent: Tuesday, December 26, 2017 3:04 AM
To: pqc-comments; pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: HK17

After reading Bernstein-Lange attack, one may wonder whether the scheme could be made secure by choosing a large enough prime p (e.g., 1000 bit p). Unfortunately, no matter how big the p it is, one can break the protocol by solving a homogeneous quadratic equation system of eight equations in four unknowns. I believe this could be done in constant steps(?) using Kipnis and Shamir's relinearization techniques or the Gröbner basis algorithm or F4/F5 algorithms). For details, see https://webpages.uncc.edu/yonwang/octonionDH.pdf

In case I am wrong, please let me know. thanks!
Yongge

On Mon, Dec 25, 2017 at 8:15 PM, D. J. Bernstein <djb@cr.yp.to> wrote:

> Dear designers, dear all,
>
> The following attack script breaks HK17 for all proposed parameters.
> Specifically, this script breaks the Python key-exchange implementation
> included in the HK17 submission. This key-exchange implementation
> appears to match the intent of the HK17 documentation, except that the
> documentation includes a normalization step; this step does not affect
> the attack.
>
> This attack takes p+1 simple computations (and is a search so Grover's
> algorithm is applicable, but all proposed parameters are small enough to
> be broken by a non-quantum attack). For comparison, the submission says
>
> * 2^64 pre-quantum security for p=251,
> * 2^128 pre-quantum security for p=65521,
> * 2^256 pre-quantum security for p=4294967291, and
> * 2^512 pre-quantum security for p=18446744073709551557.
>
> Our attack takes about 2^8, 2^16, 2^32, and 2^64 simple computations for
> these parameter sets.
>
> For simplicity the attack script focuses on the case that the public key
> rA is invertible, which occurs almost all of the time. Slightly more
> work should be able to handle the occasional exceptions.
>
> To use this script, save the following Python code as break.py; copy
> octonions.py from the HK17 submission to octonions.py in this directory;
> copy HK17-O.py from the HK17 submission to ref.py in this directory; and
> run "python break.py".
>
> ---Daniel J. Bernstein and Tanja Lange

From: perret <ludovic.perret@lip6.fr>
Sent: Tuesday, December 26, 2017 6:23 AM
To: Yongge Wang
Cc: pqc-comments; pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: HK17

Dear Yongge,

> Le 26 déc. 2017 à 09:03, Yongge Wang <yongge.wang@gmail.com> a écrit :
>
> After reading Bernstein-Lange attack, one may wonder whether the
> scheme could be made secure by choosing a large enough prime p (e.g.,
> 1000 bit p). Unfortunately, no matter how big the p it is, one can
> break the protocol by solving a homogeneous quadratic equation system
> of eight equations in four unknowns. I believe this could be done in
> constant steps(?) using Kipnis and Shamir's relinearization techniques
> or the Gröbner basis algorithm or F4/F5 algorithms). For details, see
> https://na01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fwebpa
> ges.uncc.edu%2Fyonwang%2FoctonionDH.pdf&data=02%7C01%7Csara.kerman%40n
> ist.gov%7Caa6d77871bfe4ad42f7408d54c530907%7C2ab5d82fd8fa4797a93e05465
> 5c61dec%7C1%7C1%7C636498841885235239&sdata=LvZLqdEojH%2Bcn6lzPecoKUuu4
> gylRA4uN%2Fi7zvd3lAM%3D&reserved=0
>
> In case I am wrong, please let me know. thanks!

In general, such a small system can be indeed solved very easily.
Do you have a code for generating these equations ?

Best Regards,

Ludovic Perret
Université Pierre et Marie Curie
Tel : 01 44 27 87 59
web : https://na01.safelinks.protection.outlook.com/?url=http:%2F%2Fwww-polsys.lip6.fr%2F~perret%2F&data=02%7C01%7Csara.kerman%40nist.gov%7Caa6d77871bfe4ad42f7408d54c530907%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C0%7C636498841885235239&sdata=rjApscIt4TT7ZbClzo7IINP9ryD7snnAFE7Xpha%2BvCY%3D&reserved=0

> Yongge
>
> On Mon, Dec 25, 2017 at 8:15 PM, D. J. Bernstein <djb@cr.yp.to> wrote:
> Dear designers, dear all,
>
> The following attack script breaks HK17 for all proposed parameters.
> Specifically, this script breaks the Python key-exchange implementation
> included in the HK17 submission. This key-exchange implementation
> appears to match the intent of the HK17 documentation, except that the

From: Yongge Wang <yongge.wang@gmail.com>
Sent: Tuesday, December 26, 2017 1:49 PM
To: perret
Cc: pqc-comments; pqc-forum
Subject: Re: [pqc-forum] OFFICIAL COMMENT: HK17

Dear Ludovic,
thanks for the message.

> In general, such a small system can be indeed solved very easily.

thanks for this confirmation.

> Do you have a code for generating these equations ?

I do not have a python or C script.. (I am generally lazy in writing code). But I just revised the document at https://webpages.uncc.edu/yonwang/octonionDH.pdf . It now has the exact formula to build the homogeneous quadratic equation system using the publicly observed values r_A, r_B, o_A, o_B. In the current draft, these are equation (11) and equation (12).. So if one is good at python, one can quickly convert them to python (I have limited python knowledge). Then one may use the Gröbner basis algorithm or F4/F5 algorithms to solve the equation (again, I do not have experience with these packages.. so did not try).

Thanks!
Yongge

On Tue, Dec 26, 2017 at 2:23 PM, perret <ludovic.perret@lip6.fr> wrote:

> Dear Yongge,
>
> > Le 26 déc. 2017 à 09:03, Yongge Wang <yongge.wang@gmail.com> a écrit :
> >
> > After reading Bernstein-Lange attack, one may wonder whether the scheme could be made
> > secure by choosing a large enough prime p (e.g., 1000 bit p). Unfortunately, no matter how big the p it is,
> > one can break the protocol by solving a homogeneous quadratic equation system of eight equations
> > in four unknowns. I believe this could be done in constant steps(?) using Kipnis and Shamir's relinearization
> > techniques or the Gröbner basis algorithm or F4/F5 algorithms). For details, see
> > https://webpages.uncc.edu/yonwang/octonionDH.pdf
> >
> > In case I am wrong, please let me know. thanks!
>
> In general, such a small system can be indeed solved very easily.
> Do you have a code for generating these equations ?
>
> Best Regards,
>
> Ludovic Perret
> Université Pierre et Marie Curie
> Tel : 01 44 27 87 59
> web : http://www-polsys.lip6.fr/~perret/