From: Ron Steinfeld <ron.steinfeld@monash.edu>
Sent: Tuesday, January 02, 2018 7:29 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: CFPKM

Dear All,

The following C function "crypto_kem_atk_dec" breaks the IND-CPA security of the CFPKM KEM for both CFPKM128 and CFPKM182 parameter sets.

The attack function quickly decrypts the shared secret given only the ciphertext and the public key, with high probability close to 1. It uses a rounded product of ciphertext and public keys to compute the shared secret instead of a rounded product of ciphertext and secret keys as in the legitimate decryption algorithm.

The attack run-time is about m (=81 and 116 for the CFPKM128 and CFPKM182 parameter sets respectively) multiplications and MS-bit roundings mod q (q~2^50 and 2^55 respectively for CFPKM128 and CFPKM182 parameter sets). This attack run-time is faster than the run-time of the legitimate "crypto_kem_dec" reference implementation of the decryption algorithm that uses the secret key. In comparison, the CFPKM128 and CFPKM182 are claimed to achieve 2^128 and 2^192 IND-CPA security, respectively.

The attack decryption function "crypto_kem_atk_dec" can be found as an additional function in the modified version of the CFPKM reference implementation file KEM.c available at the following link:
https://drive.google.com/file/d/1Jrysn5nM0J3UItQAfUF_A6r_W29gXX4x/view?usp=sharing

The attack function successfully decrypted the session key in all 100 KATs for CFPKM128 and CFPKM182.

To test the attack function with KATs, replace "KEM.c" in the design reference implementation with the above modified version, and replace the the modified KAT generation program "PQCgenKAT_kem.c" in the design reference implementation with the modified version available at the following link:
https://drive.google.com/file/d/1c5IT_pWTGrC2CMf5fK7AaLB3_jFtycDJ/view?usp=sharing
The response file contains two additional entries for each KAT, sd (the shared key decrypted by the attack) and mt (=0 if sd matches the encrypted shared secret ss, and 1 else). Computed response files for 100 KATs for both CFPKM128 and CFPKM182 are available at the following links:
https://drive.google.com/file/d/1na0j8X3cpIUuPoUMx1oX9BV2LZgQHbwi/view?usp=sharing
and
https://drive.google.com/file/d/1ibXVWI10KkIkRDT7TTC4xU0VIzoh1SlH/view?usp=sharing
respectively.

Best Regards,

Ron Steinfeld

--
Dr. Ron Steinfeld

Senior Lecturer,
Cybersecurity Lab,

Faculty of Information Technology,
Monash University,
Clayton VIC 3800
Australia

Email: ron.steinfeld@monash.edu
Phone: +61 3 99055225
Fax: +61 3 9905 5159
Web:
* Personal: http://users.monash.edu.au/~rste/
* Monash Cybersecurity Lab: http://www.monash.edu/cybersecurity-lab

==========

The attack decryption function (calling the functions defined in the design reference implementation source file KEM.c):

```
int crypto_kem_atk_dec(unsigned char *ss, const unsigned char *ct, const unsigned char *pk){
        int i;
        unsigned long long *b1=malloc(M*sizeof(unsigned long long));
        unsigned char *seed=malloc(SEEDSIZE*sizeof(unsigned char));
        unpack_pk(b1, seed, pk);

        unsigned long long *b2=malloc(M*sizeof(unsigned long long));
        unsigned char *c=malloc(M*sizeof(unsigned char));
        unpack_ct(b2,c,ct);

        unsigned long long *w = malloc(M*sizeof(unsigned long long));
        for (i=0;i < M;i++)
                {
                        w[i]=(b1[i]*b2[i]) ;}


        kem_rounding(ss, w);

        return 0;
}
```

From: Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov>
Sent: Tuesday, January 02, 2018 8:00 AM
To: pqc-forum@list.nist.gov
Cc: Ron Steinfeld; Tanja Lange
Subject: [pqc-forum] FW: OFFICIAL COMMENT: CFPKM

Hi all,

Fernando's post isn't showing up on the forum for some reason (I also didn't receive it via my non-work email forum subscription), so I'm posting it here; in case it was the Python script attachment that was the problem, posting it as text at the bottom

On 1/2/18, 7:12 AM, "Fernando Virdia" <fernando.virdia.2016@live.rhul.ac.uk> wrote:

Dear CFPKM authors,

We think there is a practical attack leading to the recovery of the higher order bits of Key_b, and hence the shared secret, circumventing the polynomials with errors problem.

Correctness of the scheme depends on Alice and Bob agreeing on the most significant bits (MSB) of Key_a and Key_b. In particular,

```
Key_b = MSB of { f(s_b) \odot b_1 + e_3 }
      = MSB of { f(s_b) \odot f(s_a)
               + f(s_b) \odot e_1
               + e_3 }
```

where all the terms involving the e_i have small coefficients. Therefore, the shared secret should consist of the MSB of f(s_b) \odot f(s_a). These can be recovered from the public values

```
b_1 = f(s_a) + e_1
b_2 = f(s_b) + e_2
```

by computing the component-wise product

```
b_1 \odot b_2 = f(s_a) \odot f(s_b)
              + f(s_a) \odot e_2
              + f(s_b) \odot e_1
              + e_1 \odot e_2
```

We attached a Sage script executing this attack on the 128 bits KATs.

Best regards

```
# -*- coding: utf-8 -*-
"""
Shared secret recovery attack against CFPKM 128 KATs.
The script assumes the KATs to be in "./CFPKM/KAT/KEM/CFPKM128/PQCkemKAT_128.rsp".

The (un)pack_{pk,ct} functions are translated and adapted from the reference implementation.

AUTHOR:

   Martin R. Albrecht - 2017
   Fernando Virdia - 2017

"""

from sage.all import vector, IntegerModRing, ceil, log, floor, parent, ZZ, set_random_seed, randint


def openKAT(path):
   # utility function
   def ReadHex(buf):
      if len(buf) == 0:
         return ['\x00']
      else:
         res = []
         for x in range(len(buf)/2):
            res += [int("0x" + buf[2*x:2*x+2], 0)]
         return res

   l = []
   with open(path) as f:
      el = {}
      for line in f:
         if line in ["# CFPKM\n", "\n"]:
            continue
         if "count" in line:
            l += [el]
            el = { "count": line.split("=")[1].strip() }
         else:
            pre, fix = line.split("=")
            el[pre.strip()] = ReadHex(fix.strip())

      l += [el]
   return  l[1:]


def balance(e, q=None):
   try:
      p = parent(e).change_ring(ZZ)
```

```
        return p([balance(e_, q=q) for e_ in e])
    except (TypeError, AttributeError):
        if q is None:
            try:
                q = parent(e).order()
            except AttributeError:
                q = parent(e).base_ring().order()
        e = ZZ(e) % q
        return e-q if e>q//2 else e


def size_estimate(e):
    # check x != 0 to avoid ceil(-Infinity) that fails
    return vector(ZZ, len(e), [ceil(log(abs(x), 2)) if x != 0 else 0 for x in e])


def odot(a, b, q):
    return vector(IntegerModRing(q), len(a), [a[i] * b[i] for i in range(len(a))])


LAMBDA = 256
SEEDSIZE = 48
LOG2_Q = 50
N = 80
B = 6
M = 81
Q = 1125899906842624
COFSIZE = 4096
SECRETVAL_LENGTH = 1
SHAREDKEYSIZE = M * B / 8
ERROR_LENGTH = 1
PK_LENGTH = M * 8
RANGE = 7
B_BAR = LOG2_Q - B
CRYPTO_SECRETKEYBYTES = N + SEEDSIZE
CRYPTO_PUBLICKEYBYTES = PK_LENGTH + SEEDSIZE CRYPTO_BYTES = M CRYPTO_CIPHERTEXTBYTES = PK_LENGTH + M


def pack_pk (b1, seed):
    """
    :params:  b1, list(int)
    :params:  seed, list(int)

    :returns: pk, list(int)
    """
    b1 = b1[::]
    pk = [0] * CRYPTO_PUBLICKEYBYTES
    for i in range(SEEDSIZE):
        pk[i] = seed[i]
    mask = 255
    for i in range(M):
        for j in range(8)[::-1]:
```

```
            temp = b1[i] & mask
            b1[i] = b1[i] >> 8
            pk[SEEDSIZE+i*8+j] = temp
    return pk


def unpack_pk(pk):
    """
    :params:  pk, list(int)

    :returns: seed, list(int)
    :returns: b1, list(int)
    """
    seed = pk[:SEEDSIZE]
    b1 = [0] * M
    for i in range(M):
        # unpacks PK to give out seed and the public vector b1*/
        for j in range(7):
            temp = pk[i*8+j+SEEDSIZE]
            b1[i]=b1[i] + temp
            b1[i]=b1[i] << 8
        b1[i] = b1[i] + pk[i*8+7+SEEDSIZE]
    return seed, b1


def pack_ct(b2, c):
    """
    :params: b2, list(int)
    :params: c, list(int)

    :returns: ct, list(int)
    """
    b2 = b2[::]
    ct = [0] * CRYPTO_CIPHERTEXTBYTES
    for i in range(M):
        ct[i] = c[i]
    mask = 255

    for i in range(M):
        for j in range(8)[::-1]:
            temp = b2[i] & mask # this is casted to (unsigned char) in the ref implementation
            b2[i] = b2[i] >> 8
            ct[M+i*8+j] = temp
    return ct


def unpack_ct(ct):
    """
    :params:  ct, list(int)

    :returns: b2, list(int)
    :returns: c, list(int)
```

```
    """
    c = [0] * M
    b2 = [0] * M
    for i in range(M):
        c[i] = ct[i]

    for i in range(M):
        for j in range(7):
            temp = ct[i*8+j+M]
            b2[i] = b2[i] + temp
            b2[i] = b2[i] << 8
        b2[i] = b2[i] + ct[i*8+7+M]
    return (b2, c)


def test_pack_unpack():
    kat = openKAT("CFPKM/KAT/KEM/CFPKM128/PQCkemKAT_128.rsp")

    ix = randint(0, len(kat)-1)
    pk = kat[ix]["pk"]
    ct = kat[ix]["ct"]

    # test pack/unpack pk
    print "Saved pk"
    print pk
    print
    seed1, b11 = unpack_pk(pk)
    pk2 = pack_pk(b11, seed1)
    print "Packed o Unpacked (pk) = pk"
    print pk2 == pk
    print

    seed2, b12 = unpack_pk(pk2)
    print "seeds match", seed1 == seed2
    print "b1 match", b11 == b12
    print

    # test pack/unpack ct
    print "Saved ct"
    print ct
    print
    b21, c1 = unpack_ct(ct)
    ct2 = pack_ct(b21, c1)
    print "Packed o Unpacked (ct) = ct",
    print ct2 == ct
    print

    b22, c2 = unpack_ct(ct2)
    print "b2 match", b21 == b22
    print "c match", c1 == c2
```

```
def attack():
    kat = openKAT("CFPKM/KAT/KEM/CFPKM128/PQCkemKAT_128.rsp")

    est = []
    for ix in range(len(kat)):
        pk = kat[ix]["pk"]
        ct = kat[ix]["ct"]
        ss = kat[ix]["ss"]

        seed, b1 = unpack_pk(pk)
        b2, c = unpack_ct(ct)

        b1 = vector(IntegerModRing(Q), b1)
        b2 = vector(IntegerModRing(Q), b2)
        ss = vector(IntegerModRing(Q), ss)

        # Print the bitlength of the difference between b1 odot b2 and the shared secret.
        est += [size_estimate(balance(odot(b1, b2, Q) - 2**B_BAR * ss, Q))]
        print est[ix]
```