From: Bo-Yin Yang <moscito@gmail.com>
Sent: Thursday, January 04, 2018 5:39 PM
To: pqc-forum
Subject: [pqc-forum] Official Comment: SRTPI

Dear designers, dear all,

We have broken SRTPI under CPA and TPSig under KMA.

The public key of SRTPI is equivalent to a set of MQ public keys, but as the designers themselves say in Remark 1.3, the operations involved in decryption are completely affine. Therefore

Plaintext = [constant matrix] Ciphertext + [constant vector]

Where the constant matrix and vector only depend on the key pair in question. Note that some of the vectors here are described as matrices in the specification; we "flatten" such matrices into vectors over GF(2).

The attack generates ciphertext-plaintext pairs from the public key, and solves for the constant matrix and vector using simple linear algebra. This suffices to decrypt arbitrary ciphertext. The fact that the plaintext is random-padded prior to using the public map adds only a small complication: the attack generates many encryptions of the zero plaintext to filter out the effects of the random padding.

Specifically, we compute 401 (RANDOMCOVER+1) encryptions of zero to eliminate the random padding and the constant term. We then use these intermediate results and the encryption of the 192 unit vectors to determine the linear equations in the message bits.

An implementation of the attack appears below. Here is sample output from a 3.5GHz Haswell core, showing that Eve ends up decrypting faster than Alice, after Eve's one-time processing of the public key (which takes a fraction of a second, less time than key generation):

```
240773752 ns for alice creating key pair (crypto_encrypt_keypair)
184193788 ns for eve analyzing public key (one time, independent of ciphertext)
319343 ns for bob creating ciphertext (crypto_encrypt)
770812 ns for alice decrypting (crypto_encrypt_open)
10521 ns for eve analyzing ciphertext
eve's plaintext   654458bded89cd87311ae0ea81f3d1d3af86bddef3fe46f8
bob's plaintext   654458bded89cd87311ae0ea81f3d1d3af86bddef3fe46f8
alice's plaintext 654458bded89cd87311ae0ea81f3d1d3af86bddef3fe46f8
```

In several experiments, the attack always successfully decrypted a legitimate ciphertext. We would expect a failure probability on the

scale of 2^(-80), which can be reduced through a larger choice of RANDOMCOVER, or eliminated with more effort.

Also for the TPSig signature scheme the secret-key part of the computation is entirely linear. The authors themselves mention this in Remark 1.7 but do not seem to grasp the implications.

We first describe how the signing process works and then how Eve can sign any message of her choice to be valid with Alice's public key, after seeing O(n) signatures on random messages. The SRTPI submission package does not include code for TPSig, so we did not implement the signature attack.

To sign m, Alice computes h(m), an n-bit hash of m, and constructs a matrix M from it and some secret constant values. She then computes the signature X=X0+U0+C{^+}*M, where X0, U0, and C^{+} are secret and fixed.

So, also here the secret-key operation is linear of the form
    Signature = [constant matrix] h(m) + [constant vector]
where we again "flatten" matrices from the specification into vectors over GF(2). Eve can recover the unknown constant parts from O(n) signatures. She removes the constant term by subtracting one signature from all the remaining ones and then recovers the matrix in the linear term by Gaussian elimination. After that she recovers the constant term from one of the signatures.

Bo-Yin Yang, Daniel J. Bernstein, Tanja Lange

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include "api.h"
#include "rng.h"

long long nanoseconds(void)
{
  struct timespec t;
  clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&t);
  return (long long) t.tv_nsec + 1000000000 * (long long) t.tv_sec;
}

int bit(unsigned char *c,int pos)
{
  return 1 & (c[pos / 8] >> (pos & 7));
}

void xor(unsigned char *c1,const unsigned char *c2)
```

```
{
  int i;
  for (i = 0;i < 512;++i) c1[i] ^= c2[i];
}

void swap(unsigned char *c1,unsigned char *c2)
{
  int i;
  for (i = 0;i < 512;++i) {
    unsigned char u = c1[i];
    c1[i] = c2[i];
    c2[i] = u;
  }
}

unsigned char entropy_input[48];

#define MLEN 24

unsigned char pk[CRYPTO_PUBLICKEYBYTES];
unsigned char sk[CRYPTO_SECRETKEYBYTES];

unsigned char mzero[MLEN];
unsigned char czero[512];

#define RANDOMCOVER 400
unsigned char cdiff[RANDOMCOVER][512];
int cdiffpivot[RANDOMCOVER];
int cdiffpivotlen;

unsigned char mbits[MLEN * 8][512];
unsigned char cbits[MLEN * 8][512];
int cbitspivot[RANDOMCOVER];
int cbitspivotlen;

unsigned char m[MLEN];
unsigned char c[512]; unsigned long long clen;
unsigned char t[512]; unsigned long long tlen;
unsigned char e[512];

int main()
{
  long long t0, t1;
  int i, j, k, l;


  srandom(time(0));
  for (i = 0;i < 48;++i) entropy_input[i] = random();
  randombytes_init(entropy_input,0,256);


  t0 = nanoseconds();
```

```
if (crypto_encrypt_keypair(pk,sk) != 0) abort();

t1 = nanoseconds();
printf("%lld ns for alice creating key pair (crypto_encrypt_keypair)\n",t1 - t0);
t0 = nanoseconds();


if (crypto_encrypt(czero,&clen,mzero,MLEN,pk) != 0) abort();

for (i = 0;i < RANDOMCOVER;++i) {
  if (crypto_encrypt(cdiff[i],&clen,mzero,MLEN,pk) != 0) abort();
  xor(cdiff[i],czero);
}

i = 0;
for (j = 0;j < 4096;++j)
  /* have reduced positions 0...j-1 using cdiff[0...i-1] */
  for (k = i;k < RANDOMCOVER;++k)
    if (bit(cdiff[k],j)) {
      swap(cdiff[i],cdiff[k]);
      for (l = 0;l < RANDOMCOVER;++l)
        if (l != i)
          if (bit(cdiff[l],j))
            xor(cdiff[l],cdiff[i]);
      cdiffpivot[i++] = j;
      break;
    }
cdiffpivotlen = i;

for (i = 0;i < MLEN * 8;++i) {
  for (j = 0;j < 512;++j) mbits[i][j] = 0;
  mbits[i][i / 8] = 1 << (i & 7);
  if (crypto_encrypt(cbits[i],&clen,mbits[i],MLEN,pk) != 0) abort();
  xor(cbits[i],czero);

  /* now project away from the subspace of randomness */
  /* alternative: force randombits to return 0 at this point */
  for (k = 0;k < cdiffpivotlen;++k)
    if (bit(cbits[i],cdiffpivot[k]))
      xor(cbits[i],cdiff[k]);
}

i = 0;
for (j = 0;j < 4096;++j)
  for (k = i;k < MLEN * 8;++k)
    if (bit(cbits[k],j)) {
      swap(mbits[i],mbits[k]);
      swap(cbits[i],cbits[k]);
      for (l = 0;l < MLEN * 8;++l)
        if (l != i)
```

```
        if (bit(cbits[l],j)) {
          xor(mbits[l],mbits[i]);
          xor(cbits[l],cbits[i]);
        }
      cbitspivot[i++] = j;
      break;
    }
cbitspivotlen = i;

t1 = nanoseconds();
printf("%lld ns for eve analyzing public key (one time, independent of ciphertext)\n",t1 - t0);
t0 = nanoseconds();


randombytes(m,MLEN);
if (crypto_encrypt(c,&clen,m,MLEN,pk) != 0) abort();

t1 = nanoseconds();
printf("%lld ns for bob creating ciphertext (crypto_encrypt)\n",t1 - t0);
t0 = nanoseconds();


if (crypto_encrypt_open(t,&tlen,c,clen,sk) != 0) abort();
if (tlen != MLEN) abort();

t1 = nanoseconds();
printf("%lld ns for alice decrypting (crypto_encrypt_open)\n",t1 - t0);
t0 = nanoseconds();


xor(c,czero);
for (k = 0;k < cdiffpivotlen;++k)
  if (bit(c,cdiffpivot[k]))
    xor(c,cdiff[k]);

for (j = 0;j < 512;++j) e[j] = 0;
for (k = 0;k < cbitspivotlen;++k)
  if (bit(c,cbitspivot[k])) {
    xor(c,cbits[k]);
    xor(e,mbits[k]);
  }

/* can speed up attack by composing these two matrix steps */
/* but eve is already more than ten times faster than alice */

t1 = nanoseconds();
printf("%lld ns for eve analyzing ciphertext\n",t1 - t0);
t0 = nanoseconds();


printf("eve's plaintext   ");
for (j = 0;j < MLEN;++j) printf("%02x",e[j]); printf("\n");
```

```
  printf("bob's plaintext   ");
  for (j = 0;j < MLEN;++j) printf("%02x",m[j]); printf("\n");

  printf("alice's plaintext ");
  for (j = 0;j < MLEN;++j) printf("%02x",t[j]); printf("\n");


  return 0;
}
```

From: נריה גרנות <neryagr@gmail.com>
Sent: Monday, January 08, 2018 1:07 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: SRTPI

We Dr yossi Peretz and nerya granot wish to withdraw our submission SRTPI because an attack was discovered