A Side-Channel Assisted Attack on NTRU

Amund Askeland
Sondre Rønjom

UNIVERSITY OF BERGEN

NTRU

KeyGen($seed$)

1. $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{h}) \leftarrow \text{KeyGen}'(seed)$
2. $s \leftarrow_{\$} \{0,1\}^{256}$
3. return $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q, s), \mathbf{h})$

Encapsulate($\mathbf{h}$)

1. $coins \leftarrow_{\$} \{0,1\}^{256}$
2. $(\mathbf{r}, \mathbf{m}) \leftarrow \text{Sample\_rm}(coins)$
3. $\mathbf{c} \leftarrow \text{Encrypt}(\mathbf{h}, (\mathbf{r}, \mathbf{m}))$
4. $k \leftarrow H_1(\mathbf{r}, \mathbf{m})$
5. return $(\mathbf{c}, k)$

Decapsulate($(\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q, \mathrm{s}), \mathbf{c}$)

1. $(\mathbf{r}, \mathbf{m}, fail) \leftarrow \text{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{c})$
2. $k_1 \leftarrow H_1(\mathbf{r}, \mathbf{m})$
3. $k_2 \leftarrow H_2(s, \mathbf{c})$
4. if $fail = 0$ return k_1
5. else return k_2

Decrypt Implementation

```
int owcpa_dec(unsigned char *rm, const unsigned char *ciphertext,
const unsigned char *secretkey)  {
 int i;
 int fail;
 poly x1, x2, x3, x4;

 poly *c = &x1, *f = &x2, *cf = &x3;
 poly *mf = &x2, *finv3 = &x3, *m = &x4;
 poly *liftm = &x2, *invh = &x3, *r = &x4;
 poly *b = &x1;

 poly_Rq_sum_zero_frombytes(c, ciphertext);
 poly_S3_frombytes(f, secretkey);
 poly_Z3_to_Zq(f);

 poly_Rq_mul(cf, c, f);
 poly_Rq_to_S3(mf, cf);

 poly_S3_frombytes(finv3, secretkey+NTRU_PACK_TRINARY_BYTES);
 poly_S3_mul(m, mf, finv3);
```

Decrypt Implementation

```c
int owcpa_dec(unsigned char *rm, const unsigned char *ciphertext,
const unsigned char *secretkey)  {
 int i;
 int fail;
 poly x1, x2, x3, x4;

 poly *c = &x1, *f = &x2, *cf = &x3;
 poly *mf = &x2, *finv3 = &x3, *m = &x4;
 poly *liftm = &x2, *invh = &x3, *r = &x4;
 poly *b = &x1;

 poly_Rq_sum_zero_frombytes(c, ciphertext);
 poly_S3_frombytes(f, secretkey);
 poly_Z3_to_Zq(f);

 poly_Rq_mul(cf, c, f);
 poly_Rq_to_S3(mf, cf);

 poly_S3_frombytes(finv3, secretkey+NTRU_PACK_TRINARY_BYTES);
 poly_S3_mul(m, mf, finv3);
```

Mapping $\mathbb{Z}_3$ To $\mathbb{Z}_q$

- Maps {0, 1, 2} to {0, 1, q-1}
- Highlighted intermediate result:
  - "...0000" if coefficient was 1 or 0
  - "...1111" if coefficient was 2

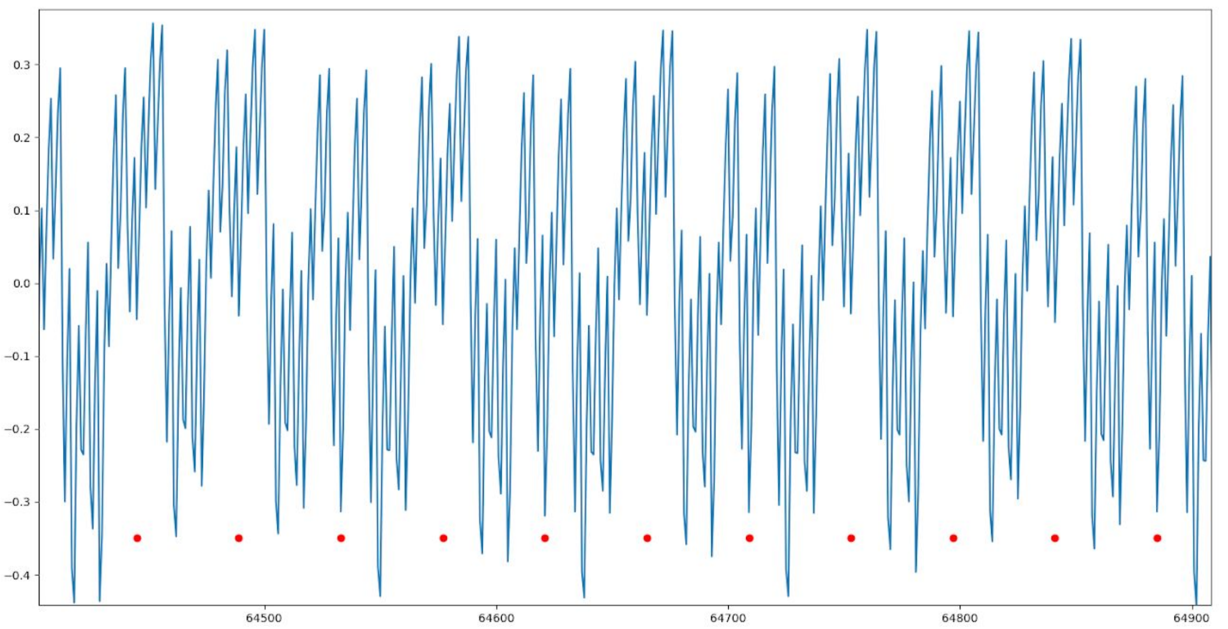
```
/* Map {0, 1, 2} -> {0,1,q-1} in place */
void poly_Z3_to_Zq(poly *r) {
  int i;
  for (i = 0; i < NTRU_N; i ++) {
    r->coeffs[i] = r->coeffs[i] | ((-(r->coeffs[i]>>1)) & (NTRU_Q-1));
  }
}
```

Mapping $\mathbb{Z}_3$ To $\mathbb{Z}_q$

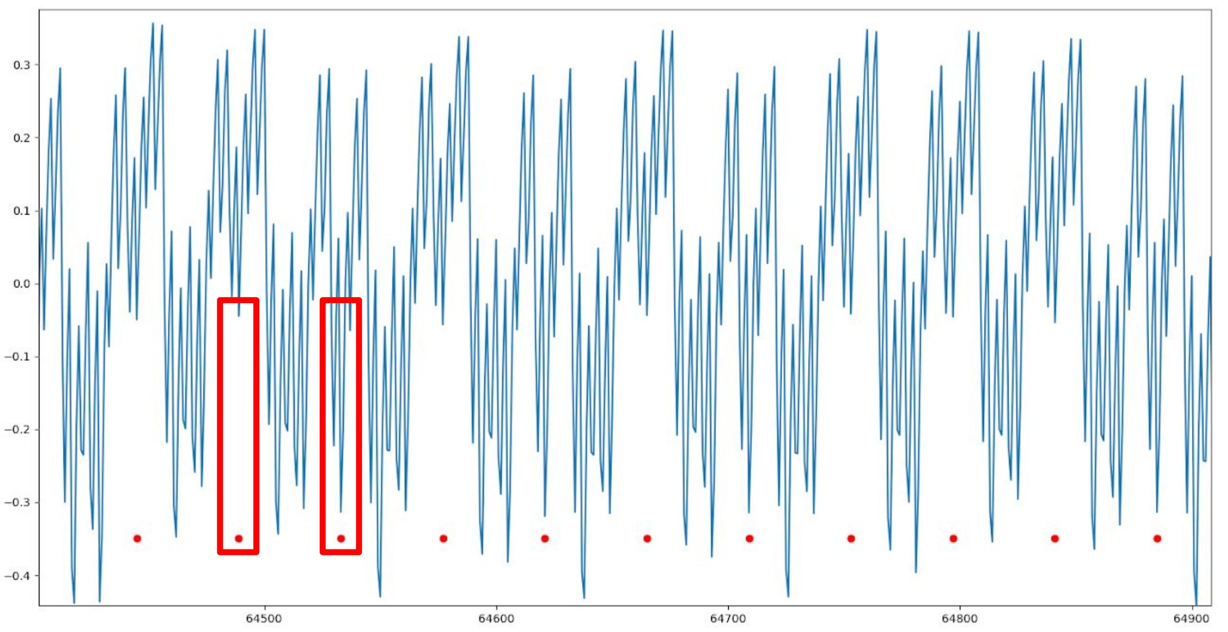
- Maps {0, 1, 2} to {0, 1, q-1}
- Highlighted intermediate result:
  - "...000" if coefficient was 1 or 0
  - "...111" if coefficient was 2

```
/* Map {0, 1, 2} -> {0,1,q-1} in place */
void poly_Z3_to_Zq(poly *r) {
  int i;
  for (i = 0; i < NTRU_N; i ++) {
    r->coeffs[i] = r->coeffs[i]  ((-(r->coeffs[i]>>1)) & (NTRU_Q-1));
  }
}
```

Power Measurement



Power Measurement



Decrypt Implementation

```
int owcpa_dec(unsigned char *rm, const unsigned char *ciphertext,
const unsigned char *secretkey)  {
 int i;
 int fail;
 poly x1, x2, x3, x4;

 poly *c = &x1, *f = &x2, *cf = &x3;
 poly *mf = &x2, *finv3 = &x3, *m = &x4;
 poly *liftm = &x2, *invh = &x3, *r = &x4;
 poly *b = &x1;

 poly_Rq_sum_zero_frombytes(c, ciphertext);
 poly_S3_frombytes(f, secretkey);
 poly_Z3_to_Zq(f);

 poly_Rq_mul(cf, c, f);
 poly_Rq_to_S3(mf, cf);

 poly_S3_frombytes(finv3, secretkey+NTRU_PACK_TRINARY_BYTES);
 poly_S3_mul(m, mf, finv3);
```

Decrypt Implementation

```
int owcpa_dec(unsigned char *rm, const unsigned char *ciphertext,
const unsigned char *secretkey)  {
 int i;
 int fail;
 poly x1, x2, x3, x4;

 poly *c = &x1, *f = &x2, *cf = &x3;
 poly *mf = &x2, *finv3 = &x3, *m = &x4;
 poly *liftm = &x2, *invh = &x3, *r = &x4;
 poly *b = &x1;

 poly_Rq_sum_zero_frombytes(c, ciphertext);
 poly_S3_frombytes(f, secretkey);
 poly_Z3_to_Zq(f);

 poly_Rq_mul(cf, c, f);
 poly_Rq_to_S3(mf, cf);

 poly_S3_frombytes(finv3, secretkey+NTRU_PACK_TRINARY_BYTES);
 poly_S3_mul(m, mf, finv3);
```

Packing Coefficients

- Five consecutive coefficients packed as $b = \sum_{i=0}^{4} f_i \cdot 3^i$
  - [2, 1, 0, 0, 1] -> 86
- Unpacking in two steps
  - 86 -> [86, 28, 9, 3, 1]
  - [86, 28, 9, 3, 1] -> [2, 1, 0, 0, 1]

Packing Coefficients

- Five consecutive coefficients packed as $b = \sum_{i=0}^{4} f_i \cdot 3^i$
  - [2, 1, 0, 0, 1] -> 86
- Unpacked in two steps
  - 86 -> [86, 28, 9, 3, 1]
  - [86, 28, 9, 3, 1] -> [2, 1, 0, 0, 1]

Modulo 3

- Highlighted intermediate result:
  - "...000" if $a \equiv 0 \pmod 3$ and $a \neq 0$
  - "...000" if
  - "...111" otherwise

```
static uint16_t mod3(uint16_t a) {
  uint16_t r;
  int16_t t, c;
  r = (a >> 8) + (a & 0xff); // r mod 255 == a mod 255
  r = (r >> 4) + (r & 0xf); // r' mod 15 == r mod 15
  r = (r >> 2) + (r & 0x3); // r' mod 3 == r mod 3
  r = (r >> 2) + (r & 0x3); // r' mod 3 == r mod 3
  t = r - 3;
  c = t >> 15;
  return (c&r) ^ (~c&t);
}
```

Modulo 3

- Highlighted intermediate result:
  - "...000" if $a \equiv 0 \pmod 3$ and $a \neq 0$
  - "...000" if $a \in S$
  - "...111" otherwise

```c
static uint16_t mod3(uint16_t a) {
  uint16_t r;
  int16_t t, c;
  r = (a >> 8) + (a & 0xff); // r mod 255 == a mod 255
  r = (r >> 4) + (r & 0xf); // r' mod 15 == r mod 15
  r = (r >> 2) + (r & 0x3); // r' mod 3 == r mod 3
  r = (r >> 2) + (r & 0x3); // r' mod 3 == r mod 3
  t = r - 3;
  c = t >> 15;
  return (c&r) ^ (~c&t);
}
```

$$S = \{79, 94, 109, 124, 127, 139, 142, 154, 157, 169, 172, 175, 184, 187, 190, 199, 202, 205, 214, 217, 220, 223, 229, 232, 235, 238\}$$

Modulo 3

- Highlighted intermediate result:
  - "...000" if $a \equiv 0 \pmod 3$ and $a \neq 0$
  - "...000" if $a \in S$
  - "...111" otherwise

```
static uint16_t mod3(uint16_t a) {
  uint16_t r;
  int16_t t, c;
  r = (a >> 8) + (a & 0xff); // r mod 255 == a mod 255
  r = (r >> 4) + (r & 0xf); // r' mod 15 == r mod 15
  r = (r >> 2) + (r & 0x3); // r' mod 3 == r mod 3
  r = (r >> 2) + (r & 0x3); // r' mod 3 == r mod 3
  t = r - 3;
  c = t >> 15;
  return (c&r) ^ (~c&t);
}
```

$$S = \{79, 94, 109, 124, 127, 139, 142, 154, 157, 169, 172, 175, 184, 187, 190, 199, 202, 205, 214, 217, 220, 223, 229, 232, 235, 238\}$$

Partial Key Recovery

- Iterate 3^5 candidates for quintuples
  - Discard those not matching measurements
- On average we recover 75% of **f**

Partial Key Recovery

- Iterate 3^5 candidates for quintuples
  - Discard those not matching measurements
- On average we recover 75% of **f**

Full Key Recovery

- Apply lattice reduction

$$\mathbf{B} = \begin{pmatrix} \mathbf{I} & \mathbf{H} \\ \mathbf{0} & q \cdot \mathbf{I} \end{pmatrix}$$

$$(\mathbf{f}, \mathbf{k})\,\mathbf{B} = (\mathbf{f}, \mathbf{g})$$

Some Remarks

- Relies only on very strong leakages
  - Robust and single trace
- Leakage can be reduced
- Implementation does not claim SCA protection

Some Remarks

- Relies only on very strong leakages
  - Robust and single trace
- Leakage can be reduced
- Implementation does not claim SCA protection

Some Remarks

- Relies only on very strong leakages
  - Robust and single trace
- Leakage can be reduced
- Implementation does not claim SCA protection

Questions?