A Side-Channel Assisted Attack on NTRU

Amund Askeland and Sondre Rønjom

Dept. of Informatics, University of Bergen, Norway
{amund.askeland,sondre.ronjom}@uib.no

Abstract. We take a look at the current implementation of NTRU submitted to the NIST post-quantum standardization project, and identify two strong sources of leakage in the unpacking of the secret key. The strength of the leakages is due to the target processor handling data with very different Hamming weight depending on parts of the secret key. We focus on using only these strong leakages, present a single-trace side-channel attack that reliably recovers a large portion of the secret key, and use lattice reduction techniques to find the remaining parts. Further, we show how small changes to the implementation greatly reduces the leakage without any overhead.

Keywords: Post-Quantum Cryptography · NTRU · Side-channel Attack · Power Analysis

1 Introduction

Most of today's public-key cryptosystems base their security on the assumed difficulty of computing factors or discrete logarithms. It is well known that if a large enough quantum computer is built, it can be used to solve these problems in polynomial time, and the assumption no longer holds [Sho94]. Post-Quantum cryptography (PQC) is a research area in cryptography based on problems that are believed to be hard, regardless of the existence of quantum computers.

The ongoing NIST Post-Quantum Cryptography standardization process is a project which intends to specify one or more quantum-resistant public-key cryptographic algorithms. The project started out with 69 submitted candidates and has gone through several rounds during which candidates have been evaluated based on their security, performance and other relevant characteristics. Currently, the project is in its third round, in which four main finalists are left in the Key Encapsulation Mechanism (KEM) category. These finalists are Classic McEliece, CRYSTALS-KYBER, NTRU and SABER.

Over the last decades, the amount of small embedded systems such as Internet of Things (IoT) devices has increased by a staggering amount. The fact that these systems are relevant for PQC is reflected in that NIST has asked for a focus on ARM Cortex-M4 when evaluating the performance of the PQC candidates [Moo]. With the increased amount of such devices, side-channel attacks have become one of the main threats against the security of these systems. A setting where an adversary can gain physical access to such a device in order to measure a side-channel is far from unrealistic, and for some side-channels like electromagnetic radiation (EM) getting close to the device might be enough. In the 3rd round of the process, NIST has requested an increased focus on resistance to side-channel attacks [MAA+20].

Side-channel attacks are attacks where an adversary exploits information that leaks through some measurable side-channel, the most common such channels being power consumption and timing. If the execution time of an implementation is dependent on some secret information, an adversary could measure the execution time and use these measurements to launch an attack. Fortunately, a simple and effective countermeasure to timing attacks is to make implementations such that they run in constant time. For the finalists, there has been a focus on implementing all computations that handle secret information in constant time, together with research on exploiting the timing side-channel for these finalists [GJN20].

Power analysis attacks are side-channel attacks where the adversary can measure the power consumption of a device running an implementation and use it to launch an attack. There is a wide range of types of power-analysis attacks, one of the simplest being Simple Power Analysis (SPA). SPA is a technique that involves directly interpreting power consumption measurements to recover secret information [KJJ99]. SPA typically requires strong leakages to be successful, like those from an implementation executing different instructions based on a key-bit. More advanced Power analysis attacks like Correlation Power Analysis (CPA) or Differential Power Analysis (DPA) can be used in a wider range of situations, for example where the measured power consumption leaks information about some intermediate value or when the leakage is small. Typically the effectiveness of DPA or CPA depends on having access to several measurements, while SPA gains less from having more measurements.

Protecting against power analysis attacks is less straightforward, and usually more expensive than countermeasures for timing attacks. A quite advanced countermeasure is masking, which effectively randomizes the computation of the cryptographic algorithm by splitting intermediate values into several shares [RRVV15]. Out of the four finalists, currently only SABER has a masked implementation intended to be resistant to first-order side-channel attacks [BDK+20]. While implementations that do not employ advanced countermeasures like masking are often referred to as "unprotected", there are naturally large differences in how easily attacked such implementations are. A focus by the designer of an implementation to make the expected secret-dependant leakage as small as possible is important. Such a focus can be enough to make attacks like SPA impractical, or drastically increase the number of measurements an adversary needs.

Several power analysis attacks using various techniques have been launched on the different NIST PQC candidates. In [RSRCB20] the authors present a side-channel assisted chosen chiphertext attack, and demonstrate it on KYBER. In [NDGJ21] a side-channel attack on the masked implementation of SABER is presented. In [SKL+20] the authors present single-trace side-channel attacks on the message encoding of several of the NIST PQC finalists, and discuss how it their attack can also be applied to NTRU. Their proposed attack on NTRU relies on dividing measurements into two groups where the Hamming weight is 0 or 1, for which they use machine learning based analysis. Side-channel attacks on the PQC finalists continues to be an important research topic especially considering the ongoing standardization process.

1.1 Contribution

In this paper, we point out two very strong sources of leakage in the submitted implementation of NTRU and present a side-channel assisted attack to exploit them. Both of the leakages stem from the target processor handling data with Hamming weight of either 0 or the bus width during unpacking of the secret key. Since this is the largest difference in Hamming weight we could have, the leakages are very easily measurable. Compared to other power-analysis attacks on NTRU [AKJ+18, SKL+20, HCY19] we rely only on these very strong sources of leakage, which makes for a very reliable and practical attack. The nature of the leakages we exploit is such that it does not leak the entire secret key, but we are able to recover the remaining parts using lattice reduction techniques. We also show how the strength of the leakage allows for single-trace EM attacks, and suggest an alternative implementation to limit the leakage.

```
KeyGen′(seed)
1. $(\mathbf{f}, \mathbf{g}) \leftarrow$ Sample_fg$(seed)$
2. $\mathbf{f}_q \leftarrow (1/\mathbf{f}) \mod (q, \mathbf{\Phi}_n)$
3. $\mathbf{h} \leftarrow (3 \cdot \mathbf{g} \cdot \mathbf{f}_q) \mod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
4. $\mathbf{h}_q \leftarrow (1/\mathbf{h}) \mod (q, \mathbf{\Phi}_n)$
5. $\mathbf{f}_p \leftarrow (1/\mathbf{f}) \mod (3, \mathbf{\Phi}_n)$
6. return $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{h})$
```

```
Encrypt($\mathbf{h}$, ($\mathbf{r}$,$\mathbf{m}$))
1. $\mathbf{m}' \leftarrow$ Lift$(\mathbf{m})$
2. $\mathbf{c} \leftarrow (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}') \mod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
3. return $\mathbf{c}$
```

```
Decrypt(($\mathbf{f}$, $\mathbf{f}_p$, $\mathbf{h}_q$), $\mathbf{c}$)
1. if $\mathbf{c} \not\equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$ return $(0, 0, 1)$
2. $\mathbf{a} \leftarrow (\mathbf{c} \cdot \mathbf{f}) \mod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
3. $\mathbf{m} \leftarrow (\mathbf{a} \cdot \mathbf{f}_p) \mod (3, \mathbf{\Phi}_n)$
4. $\mathbf{m}' \leftarrow$ Lift$(\mathbf{m})$
5. $\mathbf{r} \leftarrow ((\mathbf{c} - \mathbf{m}') \cdot \mathbf{h}_q) \mod (q, \mathbf{\Phi}_n)$
6. if $(\mathbf{r}, \mathbf{m}) \in \mathcal{L}_r \times \mathcal{L}_m$ return $(\mathbf{r}, \mathbf{m}, 0)$
7. else return $(0, 0, 1)$
```

Figure 1: Description of NTRU PKE from [CDH+]

1.2 Organization of the paper

In Section 2 we give the necessary background on NTRU and a standard lattice reduction attack against it. Section 3 describes some expected leakage points in the NTRU implementation, and describes the attack. In Section 4 we present some measurements and run the attack in practice. Section 5 presents a change to the implementation, along with measurements that show the effect of the change. Section 6 briefly discusses an alternative approach to the attack. Finally, Section 7 concludes the paper.

2 Preliminaries

2.1 NTRU

The NTRU submisstion to the NIST PQC standardization process is a KEM based on Hoffstein, Pipher and Silverman's public-key encryption scheme (PKE) [HPS98]. The PKE is parameterized by coprime positive integers (n, p, q), sample spaces $(\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m)$ and an injection Lift : $\mathcal{L}_m \to \mathbb{Z}[x]$ [CDH+]. We define $\mathbf{\Phi}_1$ as the polynomial $x - 1$ and $\mathbf{\Phi}_n$ as the polynomial $(x^n - 1)/(x - 1)$. Key-generation, encryption and decryption is performed as in Figure 1. Here Sample_fg samples two random polynomials, $\mathbf{f}$ and $\mathbf{g}$, with coefficients in $\mathbb{Z}_p$ and degree at most $n - 2$. Here $p = 3$, which makes $\mathbf{f}$ and $\mathbf{g}$ trinary polynomials. The polynomial $\mathbf{h}$ is the public key. Subscripts $_p$ and $_q$ are used to denote inverses with coefficients in $\mathbb{Z}_p$ and $\mathbb{Z}_q$, respectively.

From the PKE a Key Encapsulation Mechanism (KEM) with IND-CCA2 security can be constructed [CDH+]. The KEM uses the routines from the PKE and two hash functions (H_1, H_2) in its KeyGen, Encapsulate and Decapsulate routines as shown in Figure 2.

The submitted versions of NTRU are **ntruhps2048509**, **ntruhps2048677**, **ntruhps4096821**, and **ntruhrss701** where n is 509, 677, 821, and 701 respectively. There are some differences between NTRU-HRSS and NTRU-HPS, the most notable one from this paper's point of view being that the public key is generated as $\mathbf{h} = 3\mathbf{\Phi}_1 \cdot \mathbf{g} \cdot \mathbf{f}_q \mod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$ instead of as in Figure 1.

```
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. $(\mathbf{r}, \mathbf{m}) \leftarrow$ 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(($\mathbf{f}$, $\mathbf{f}_p$, $\mathbf{h}_q$, 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 $k_2$
```

Figure 2: Description of NTRU KEM from [CDH+]

2.2 Lattice-Based Attacks on NTRU

We now represent the polynomials as vectors, and note that we can find a matrix $\mathbf{H}$ such that $\mathbf{f}\,\mathbf{H} \equiv \mathbf{g} \pmod{q}$. For NTRU-HPS this matrix is simply the circulant matrix generated from $\mathbf{h}$ multiplied with the modular multiplicative inverse of 3 modulo q. For NTRU-HRSS we also need to account for the multiplication of $\mathbf{\Phi}_1$. We can construct a matrix $\mathbf{B}$ as in Equation 1, and we will then have that there is a vector $\mathbf{k}$ such that Equation 2 holds. The rows of $\mathbf{B}$ form the basis of a lattice referred to as the *NTRU public lattice*, which has dimension $2n$. From Equation 2 it is clear that this lattice contains the vector $(\mathbf{f}, \mathbf{g})$, as well as vectors corresponding to rotations of $\mathbf{f}$ and $\mathbf{g}$. Using the Gaussian heuristic, it can be shown that these vectors are probably the shortest in the lattice [HG07].

$$\mathbf{B} = \begin{pmatrix} \mathbf{I} & \mathbf{H} \\ \mathbf{0} & q \cdot \mathbf{I} \end{pmatrix} \tag{1}$$

$$(\mathbf{f}, \mathbf{k})\,\mathbf{B} = (\mathbf{f}, \mathbf{g}) \tag{2}$$

Finding the shortest vector in a lattice is referred to as the *shortest vector problem* (SVP), which is the basis for the security in NTRU. Techniques for solving SVP involve lattice reduction, i.e finding another basis for the same lattice spanned by shorter vectors. One of the most performant algorithms for lattice reduction is the BKZ algorithm [SE94, CN11]. BKZ is parameterized by a block size, β, which impacts both the running time and the probability of finding the shortest vector. In order to estimate the smallest block size required to solve SVP we can use an estimator like the one from [APS15], the estimated block size will also be closely related to the expected running time.

3 The Attack

The implementations we focus on come from the Round 3 submission package [Sch] to the NIST Post-Quantum standardization project. Our attack is applicable to all implementations within the submission package (all the versions, and their Reference/Optimized implementations). We assume that the implementations are executed on a processor that uses two's complement for representing negative numbers (an assumption we share with the implementation itself) and that the power consumption of the processor leak information about the Hamming weight of the data being processed.

3.1 Point of Attack

As we are interested in recovering the secret key, it is natural to focus on the decapsulation procedure of the KEM implementation. This procedure takes in a ciphertext and a secret key where the secret key consists of packed versions of $\mathbf{f}$, $\mathbf{f}_p$, $\mathbf{h}_q$ and s. Here s is a random 32-byte value that is used in case the decryption fails. In order to reduce the size of the secret key, groups of five coefficients from the trinary polynomials ($\mathbf{f}$ and $\mathbf{f}_p$) are packed into one byte of the secret key. We refer to groups of five consecutive coefficients as *quintuples* from now on. The packing of the trinary polynomials is performed as in Equation 3,

$$b = \sum_{i=0}^{4} c_i \cdot 3^i \tag{3}$$

where b is a byte of the secret key, and c_i are the coefficients of a quintuple. Unpacking of a quintuple then follows straight-forwardly from

$$c_i = \lfloor b/3^i \rfloor \mod 3. \tag{4}$$

In the implementation from the submission package, the unpacking is done in two steps where the last step is to reduce all coefficients modulo 3. This reduction is performed in a loop over all coefficients using the C-function `mod3()` shown in Listing 1.

At line 8 of the `mod3()` function, r will be in $\{0, 1, 2, 3, 4\}$. The last three lines is a constant-time version of `if (r < 3) return r; else return r-3;`. We note that at line 11, t will have a Hamming weight of W or $W - 1$ if $r < 3$ and 0 or 1 if $r \geq 3$, where W is the bus width of the target processor. Commonly $W = 32$ or $W = 64$. Further, we note that c will have a weight of W or 0, and $\sim c$ will have the opposite weight of c. Since the difference in weight between W and 0 is the maximum we can have, we can expect that it will be easy to distinguish between the cases $r < 3$ or $r \geq 3$ at line 8. For convenience we will refer to the case $r \geq 3$ at line 8 as a coefficient being *partially reduced*, which is what we expect to be able to recognise in measurements.

```
 1 static uint16_t mod3(uint16_t a) {
 2   uint16_t r;
 3   int16_t t, c;
 4 
 5   r = (a >> 8) + (a & 0xff);
 6   r = (r >> 4) + (r & 0xf);
 7   r = (r >> 2) + (r & 0x3);
 8   r = (r >> 2) + (r & 0x3);
 9 
10   t = r - 3;
11   c = t >> 15;
12   return (c&r) ^ (~c&t);
13 }
```

Listing 1: `mod3()` implementation

After $\mathbf{f}$ has been unpacked from the secret key, a transformation on its coefficients is performed to map them from Z_3 to Z_q. The function for doing this is shown in listing 2. Here we note that the intermediate result `-(r->coeffs[i] »1)` will have a Hamming weight of W if the coefficient was 2 or a weight of 0 if the coefficient was 0 or 1. From this we can expect to find all coefficients with a value of 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));
  }
}
```

Listing 2: `poly_Z3_to_Zq()` implementation

We note that the two points of leakage described above should be very strong as the cases we have to differentiate between have very different Hamming weight. We also expect that there will be smaller leakages present, for example we could try to differentiate between a Hamming weight of 0 and 1 in both functions, but these are much smaller and likely to be harder to exploit. The idea of this attack will be to only rely on the strongest points of leakage, in order to have a stronger attack that does not rely on high quality or a large number of measurements.

3.2 Partial Key-Recovery

As described, we have identified two interesting parts of the implementation where we expect to find strong leakage resulting in information about the coefficients of $\mathbf{f}$ and $\mathbf{f}_p$. The easiest leakage to interpret is the mapping from $\mathbb{Z}_3$ to $\mathbb{Z}_q$, where we expect to find all coefficients with a value of 2. This mapping not performed on $\mathbf{f}_p$, so we will only have this information available for $\mathbf{f}$. Since $\mathbf{f}$ is sampled randomly we expect that 1/3 of the coefficients can be recovered here.

The second piece of information we expect to find is whether a coefficient is partially reduced or not. To make use of this information, we divide the partially reduced coefficients into two cases depending on the value of r at line 8 in listing 1. If $r = 3$, the coefficient is a zero that has been unpacked as a nonzero value. From equations 3 and 4 we see that this corresponds to coefficients with value zero that have nonzero coefficients behind it within a quintuple. If $r = 4$, the coefficient has the value 1, but has been unpacked as a value, c, satisfying $\lfloor c/16 \rfloor + (c \mod 16) = 16 + 3k$ where k is an integer larger than zero. There are 26 values smaller than 3^5 that satisfies this, the smallest of them being 79. We note that since $79 > 3^4$, we can guarantee that any partially reduced coefficients found in the three last positions in a quintuple have the value zero.

A simple approach to decide all the coefficients we can with the available information, is to go through all candidates for a quintuple and see if what was observed in measurements is consistent with the candidate. All quintuple candidates that does not match the observations can be thrown away, and if all the remaining candidates agree on a coefficient this coefficient can be decided with certainty. This approach also allows us to find the expected number of fully recovered coefficients for both $\mathbf{f}$ and $\mathbf{f}_p$, which is 75.3 % and 13.5 % respectively. These percentages are approximations, since the calculations assume that all quintuples are equally probable, and does not take into account that the length of the key is not divisible by 5. A python-script for performing this calculation is available in the Appendix.

In addition to being able to decide a number of coefficients with certainty, we also find constraints for many of the undecided coefficients. If we for example find that the last three coefficients, c_2, c_3, c_4 of a quintuple are neither partially reduced nor have the value 2, we can say with certainty that they have to satisfy $1 \geq c_2 \geq c_3 \geq c_4 \geq 0$. Algebraically we can express this as $c_4 c_3 = c_4$ and $c_2 c_3 = c_3$. Other constraints we might encounter include coefficients of $\mathbf{f_p}$ which can only take values 0 or 2, which can be expressed as $c_i^2 + c_i \equiv 0 \pmod 3$.

3.3 Full Key-Recovery

In order to fully recover the secret key $\mathbf{f}$, we will rely on lattice reduction methods as described in Section 2.2. The recovered coefficients can be used in such a way that if we have recovered m coefficients, we can construct a lattice with dimension $2(n - m + 1)$, compared to $2n$ without any known coefficients. This is done by splitting $\mathbf{f}$ into a known and an unknown part, and precalculating the product of the known part and the public key. We also note that although the entire secret key used by the KEM consists of $\mathbf{f}$, $\mathbf{f}_p$,

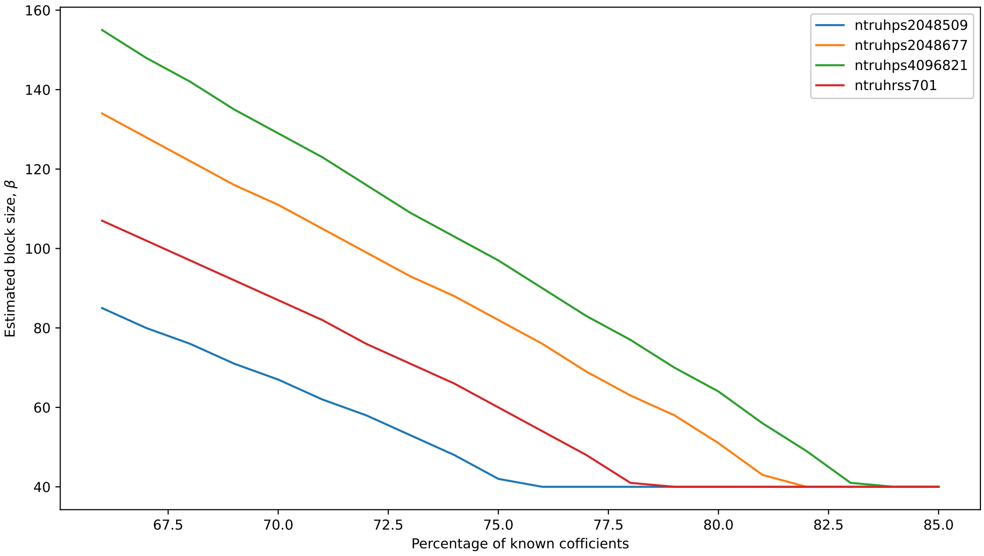

Figure 3: BKZ block size estimates

$\mathbf{h}_q$ and some random bytes s, recovering $\mathbf{f}$ will be enough to construct all of those values except for s which is not needed by an adversary.

Using the estimator from [APS15] we can estimate the block-size, β, required for BKZ-β to solve SVP for a given NTRU parameter set and m. For simplicity we will here limit estimates of the complexity of an attack to estimations of β. For reference in [ADH+19] Albrecht et al. ran BKZ-81 and BKZ-112 in 1h 23m and 60h, respectively (or 8h 36m and 2554h measured in CPU-time). Figure 3 shows our estimates of the required block size for different percentages of recovered coefficients, estimates are lower-capped at $\beta = 40$.

Ideally we would have liked to utilize the extra constraints on the coefficients of $\mathbf{f}$ that we find, but we have been unable to find a good method of incorporating these extra constraints into the lattice. In some cases guessing a few coefficients might be beneficial, but then there is no guarantee that the lattice is correct.

4 Experiments

The goal of our experiments will be to validate that the expected leakages outlined in Section 3.1 are present, and easily measurable as expected. We will also use the measurements to recover a large portion of $\mathbf{f}$ and run BKZ to recover all coefficients.

4.1 Setup

For our experiments we use an STMF32F303-based target board. STM32F303 is a microcontroller based around the 32-bit ARM cortex-m4 processor core, which is commonly used in embedded systems such as IoT devices. The specific target board comes with the ChipWhisperer Pro [Inc], which is the equipment we use for most of our measurements. The ChipWhisperer is quite ideal for experimentation, but does not necessarily represent the most realistic setup. One of the main reasons for this is that the ChipWhisperer supplies the clock signal for the target board, meaning the sampling frequency is perfectly synchronized with the target's clock frequency. To investigate the practicality of this

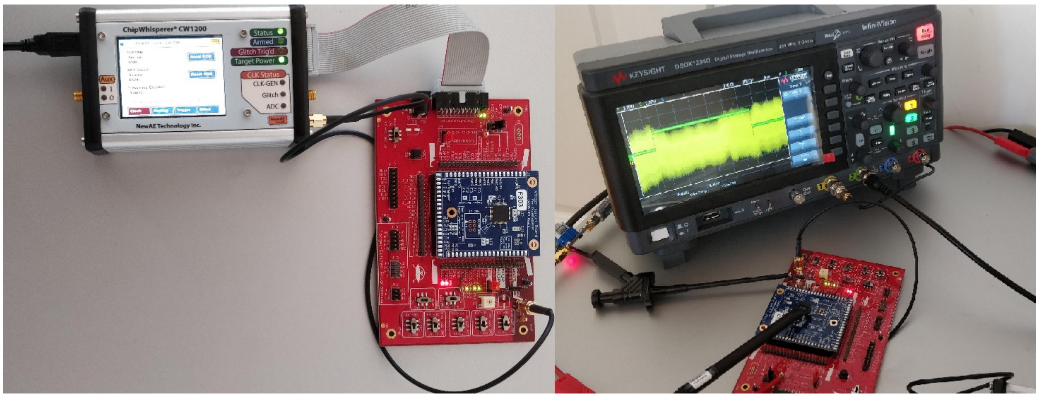

Figure 4: Left: ChipWhisperer setup. Right: Oscilloscope setup

attack, we will also do measurements with an oscilloscope. The measurements done with the ChipWhisperer will be voltage measurements over a shunt-resistor placed between the target processor and its supply. The oscilloscope measurements will done with an EM-probe, which measures the magnetic field radiating from the target device. A 20dB amplifier and a low-pass filter is connected between the EM-probe and the oscilloscope, and the EM-probe will be held at a distance of roughly 5 mm from the target microcontroller.

The target device is programmed with the implementation of **ntruhps2048509** from the submission package [Sch]. We compile the implementation using `arm-none-eabi-gcc`, with optimization flag `-Os`. The two different setups are shown in Figure 4.

4.1.1 A Note on Measurements

It is easy to get lost in what is actually being measured here. For the ChipWhisperer we will measure the voltage over a shunt resistor in series with the target processor. This voltage will in turn be proportional to the current draw of the target processor. Assuming we have a constant supply voltage, the measured voltage will be proportional to the power consumption.

The oscilloscope setup will measure a voltage over the EM-probe. This voltage will be proportional to the rate of change of the magnetic flux through the probe, which (ignoring any noise sources) will be proportional to the rate of change in the current draw of the target device. From this we can see that the measurements done with the oscilloscope setup will essentially be the derivative of the measurements from the ChipWhisperer setup. Although referring to the measurements as power consumption measurements might not be strictly correct, the origin of the measured values will be closely related to the power consumption of the target device.

4.2 Direct Measurements

We use the ChipWhisperer setup to take a single measurement of the power consumption of the target device during execution of the decapsulation procedure. Figure 5 shows large parts of this measurement, together with subtraces corresponding to a single execution of `mod3()` and a single round in `poly_z3_to_zq()`. There are n subtraces for each of the two, in the case of this particular measurement $n = 509$.

Figure 6 shows the 509 different subtraces for the different rounds in the loop of the `poly_z3_to_zq()` function. The subtraces where the coefficients had the value 2 has been coloured red, and we can clearly see that the subtraces fall into two groups. Note that the colouring naturally does not represent information that would be available to an adversary,

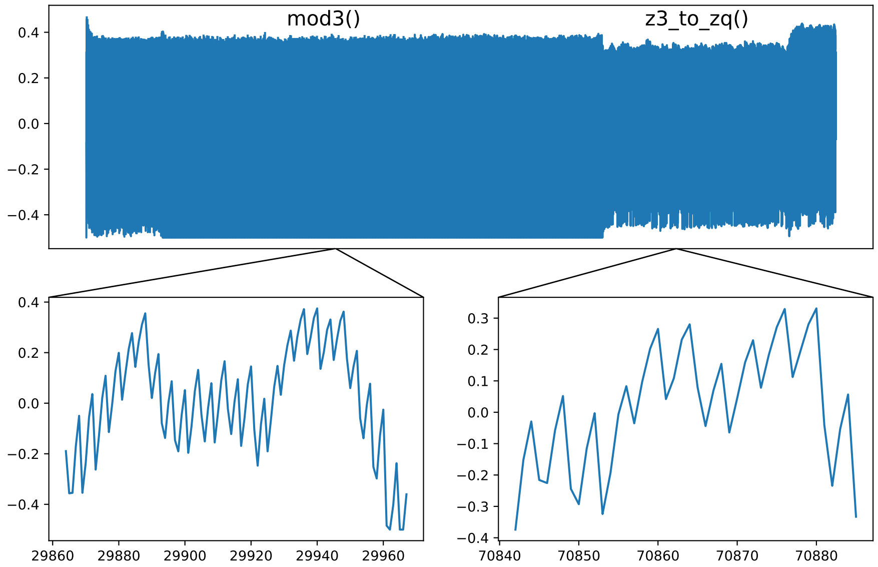

Figure 5: Full trace and subtraces

but has been used to clarify the plot. The two groups clearly formed by the subtraces is what an adversary would use to recover this information.

In the same manner, Figure 7 shows the 509 different subtraces for calls to the `mod3()` function. Here the traces corresponding to partially reduced coefficients have been coloured red. Again, we see that there is a large difference in the power consumption, and that the two cases are easily distinguishable.

4.3 EM Measurements

We use the oscilloscope setup to take a single measurement with the EM-probe. Figure 8 shows the 509 subtraces corresponding to the execution of `poly_z3_to_zq()`, and Figure 9 shows subtraces corresponding to executions of `mod3()`. The subtraces have been coloured in the same manner as for the ChipWhisperer setup.

Compared to the measurements done with the ChipWhisperer, it is far less obvious that the subtraces can be distinguished into two groups. However if we plot the Root-Mean-Square (RMS) values for parts of the subtraces, as shown in the lower parts of the figures, the difference is more clear.

4.4 Full Key-Recovery

We generate some sets of keys for **ntruhps2048509**, run our side-channel attack to recover the available information, and use the techniques outlined in Section 3.2 to recover a number of coefficients of $\mathbf{f}$. Using the recovered coefficients, we launch a lattice reduction attack as outlined in Section 3.3 to recover the full key. We use the BKZ implementation from fpylll [dt21]. In practice, we run BKZ with increasing block-sizes, β, starting at $\beta = 10$ until we recover the secret key. Table 4.4 sums the results from these experiments. Note that the running time has been purposefully left out, as little effort was put into speeding up the process and the experiments was run on a standard laptop. For reference experiment 1 took 1m 43 s and experiment 3 took 1h 18m, the main reason for the large time difference was that experiment 3 had to be run with a higher floating point precision, and could therefore not utilize the hardware it was run on.

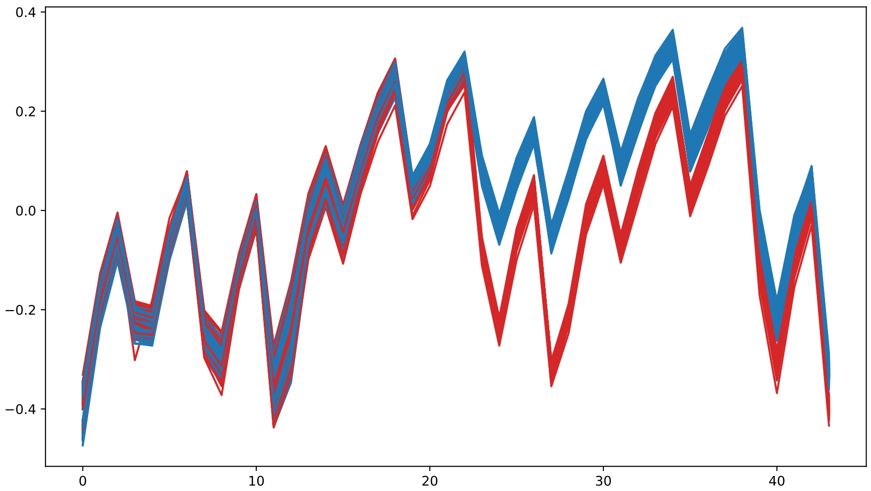

Figure 6: Subtraces showing difference in power consumption, `poly_z3_to_zq()`

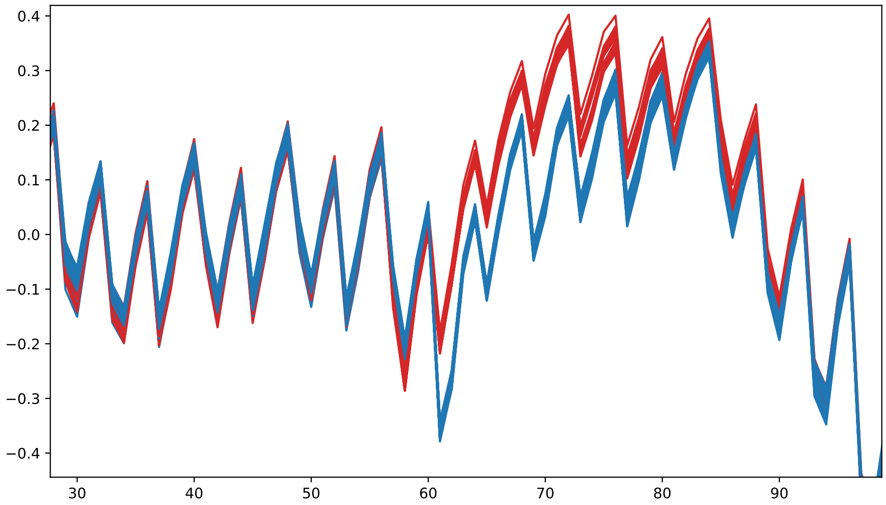

Figure 7: Subtraces showing difference in power consumption, `mod3()`

Table 1: Results from running BKZ-β on partially recovered keys

Experiment #	Recovered Coefficients	Percentage	Required β
1	393	77.4	42
2	373	73.4	51
3	377	74.2	52
4	377	74.2	52
5	386	76.0	41

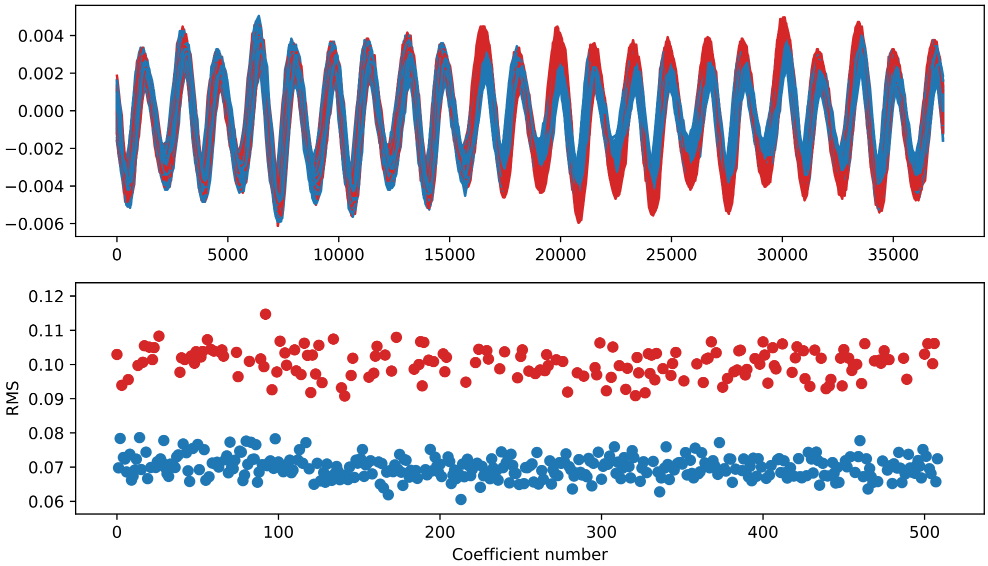

Figure 8: Subtraces and RMS values from EM measurements, `poly_z3_to_zq()`

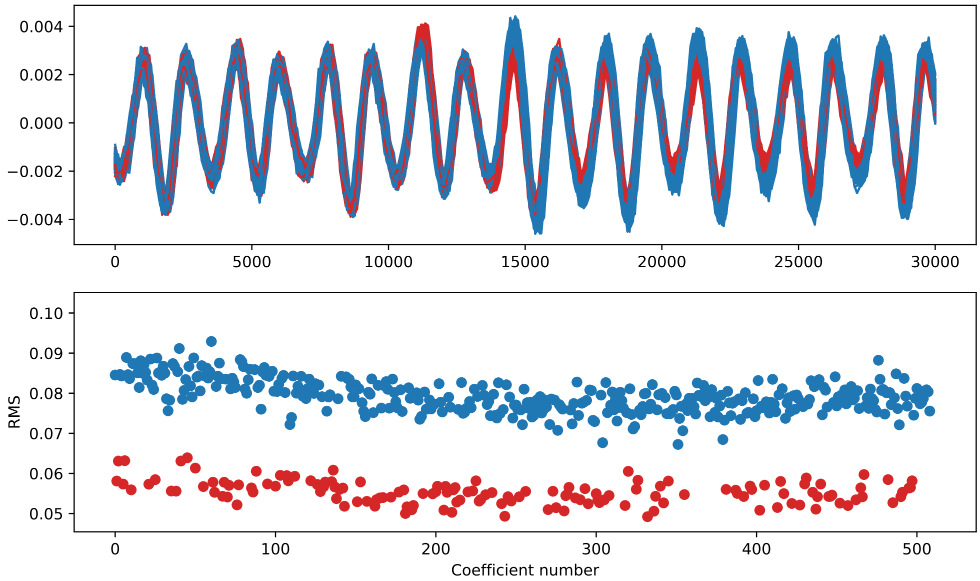

Figure 9: Subtraces and RMS values from EM measurements, `mod3()`

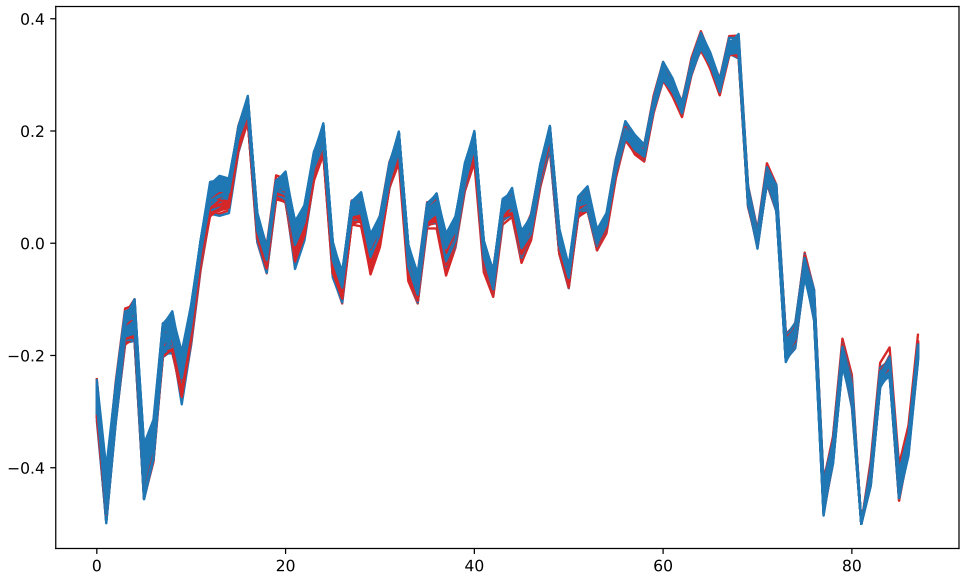

Figure 10: Subtraces, alternative `mod3()` implementation

5 Alternative Implementation

The two identified leakage points are expected to be very strong since the target processor will handle data with very different Hamming weight depending on the secret coefficients. In Listing 3 we present a functionally equivalent implementation of `mod3()`, that should result in a lower leakage. The instruction count between the two implementations is comparable, with the alternative implementation shown below using slightly fewer instructions when compiled with `arm-none-eabi-gcc`.

```c
static uint16_t mod3_alt(uint16_t a)
{
  uint16_t r;

  r = (a >> 8) + (a & 0xff);
  r = (r >> 4) + (r & 0xf);
  r = (r >> 2) + (r & 0x3);
  r = (r >> 2) + (r & 0x3);
  r = ((r >> 2) + (r)) & 0x3; // Map 4 -> 1
  return (r + ((r+1)>>2)) & 0x3; // Map 3 -> 0
}
```

Listing 3: Alternative `mod3()` implementation

We rerun parts of our earlier experiments with this alternative implementation of `mod3()`. The results are shown in Figure 10, as before the red subtraces correspond to coefficients that are partially reduced. As can be seen, there is a much lower difference in the power consumption between the red and blue subtraces compared to the original implementation.

6 Exploiting More of the Leakage

While the leakages related to $\mathbf{f}$ enables us to recover sufficiently many coefficients in order to attack NTRU directly using lattice reduction, additional information about $\mathbf{f}$ and $\mathbf{f}_p$ deduced from the leakage is not exploited in the above attack. Although we were unable to craft a dedicated algorithm exploiting this extra information and beat the direct attack, we briefly mention it here as a potential new and interesting research direction.

Since the two polynomials are inverses, $\mathbf{f}\,\mathbf{f}_p \equiv 1 \pmod{(3, \mathbf{\Phi}_n)}$, we can set up a system of $n-1$ bilinear equations with $2(n-1) - m - m_p$ unknowns where m and m_p are the number of known coefficients of $\mathbf{f}$ and $\mathbf{f}_p$, respectively. In addition to the leaked coefficients we have some extra constraints on the possible values of a quintuple. This can further be modelled as additional non-linear equations in 5 or less coefficient variables. Note that for this particular system, we know additionally that the remaining unknown coefficients of $\mathbf{f}$ are in $\{0, 1\}$.
Crafting a dedicated method for exploiting leakage from a polynomial and its inverse, such as a dedicated algebraic method able to exploit this structure, is left as an interesting open general problem that may have future applications in side-channel cryptanalysis of lattice based schemes.

7 Conclusion and Future Research

We have presented a single trace SPA attack on the NTRU implementation submitted to the NIST PQC standardization process. Although the implementation does not include any advanced countermeasures like masking, we point out that there are two very strong sources of leakage that makes an especially easy target for such attacks. In fact, the leakages that the attack relies on is so strong that subtraces can easily be divided into two sets using the naked eye. On average the SPA attack allows us to recover about 75 % of the secret key polynomial $\mathbf{f}$, and 13 % of its inverse $\mathbf{f}_p$. We consider the problem of recovering the entire secret key by utilizing the partially recovered $\mathbf{f}$ and $\mathbf{f}_p$ an interesting one for future research. In this work we use lattice-reduction techniques to recover the remaining parts of $\mathbf{f}$. Further we presented an alternative implementation that removes one of the strong sources of leakage, without any time-penalty.

8 Acknowledgments

We would like to thank Qian Guo for a helpful discussion and feedback on an early draft of this paper. We would also like to thank Martin Albrecht for his guidance in estimating the cost of lattice reduction attacks on NTRU instances with partially known keys.

References

[ADH$^+$19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. Cryptology ePrint Archive, Report 2019/089, 2019. https://eprint.iacr.org/2019/089.

[AKJ$^+$18] Soojung An, Suhri Kim, Sunghyun Jin, HanBit Kim, and HeeSeok Kim. Single trace side channel analysis on ntru implementation. *Applied Sciences*, 8:2014, 10 2018.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046, 2015. https://eprint.iacr.org/2015/046.

[BDK$^+$20] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel resistant implementation of saber. Cryptology ePrint Archive, Report 2020/733, 2020. https://eprint.iacr.org/2020/733.

[CDH$^+$] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Tsunekazu Saito, Peter Schwabe, William Whyte, Keita Xagawa, Takashi Yamakawa, and Zhenfei Zhang. NTRU Algorithm specifications and supporting documentation. https://ntru.org/f/ntru-20190330.pdf Accessed: 16.03.2021.

[CN11] Yuanmi Chen and Phong Q. Nguyen. Bkz 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 1–20, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[dt21] The FPyLLL development team. fpylll, lattice reduction for Python, Version: 0.5.5. Available at https://github.com/fplll/fplll, 2021.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the fujisaki-okamoto transformation and its application on frodokem. Cryptology ePrint Archive, Report 2020/743, 2020. https://eprint.iacr.org/2020/743.

[HCY19] Wei-Lun Huang, Jiun-Peng Chen, and Bo-Yin Yang. Power analysis on ntru prime. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):123–151, Nov. 2019.

[HG07] Nick Howgrave-Graham. A Hybrid Lattice-Reduction and Meet-in-the-Middle Attack Against NTRU. pages 150–169, 08 2007.

[HPS98] J Hoffstein, J Pipher, and J H Silverman. NTRU: A ring-based public key cryptosystem. *Algorithmic Number Theory. ANTS 1998. Lecture Notes in Computer Science*, 1423, 1998.

[Inc] NewAE Technology Inc. Chipwhisperer pro. https://www.newae.com/products/NAE-CW1200.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[MAA$^+$20] D Moody, G Alagic, D Apon, D Cooper, Q Dang, J Kelsey, Y Liu, C Miller, R Peralta, R Perlner, , A Robinson., D Smith-Tone., and J Alperin-Sheriff. Status report on the second round of the nist post-quantum cryptography standardization process, 2020. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, https://doi.org/10.6028/NIST.IR.8309 (Accessed 25.03.2021).

[Moo] Dustin Moody. The 2nd round of the nist pqc standardization process. https://csrc.nist.gov/CSRC/media/Presentations/the-2nd-round-of-the-nist-pqc-standardization-proc/images-media/moody-opening-remarks.pdf (Accessed 25.03.2021).

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked ind-cca secure saber kem. Cryptology ePrint Archive, Report 2021/079, 2021. https://eprint.iacr.org/2021/079.

[RRVV15] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-lwe implementation. Cryptology ePrint Archive, Report 2015/724, 2015. https://eprint.iacr.org/2015/724.

[RSRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based pke and kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):307–335, Jun. 2020.

[Sch] John M. Schanck. NTRU round 3 NIST submission package. https://ntru.org/release/NIST-PQ-Submission-NTRU-20201016.tar.gz Accessed: 16.03.2021.

[SE94] Claus Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–199, 08 1994.

[Sho94] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[SKL$^+$20] Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Taeho Lee, Jaeseung Han, Hyojin Yoon, Jihoon Cho, and Dong-Guk Han. Single-trace attacks on the message encoding of lattice-based kems. Cryptology ePrint Archive, Report 2020/992, 2020. https://eprint.iacr.org/2020/992.

Appendix

The following python script includes functions for recovering keys based on measurements, and a calculation of the expected number of recovered coefficients.

```python
# First few lines of mod3() function
def is_partially_reduced(a):
    r = (a >> 8) + (a & 0xff)
    r = (r >> 4) + (r & 0xf)
    r = (r >> 2) + (r & 0x3)
    r = (r >> 2) + (r & 0x3)
    return r >= 3

# These functions recover quintuples from measurements
def recover_quintuple_f(measured_part_red, measured_twos):
    valid_candidates = [] # list of valid quintuples
    for quint in range(3**5):
        candidate = [(quint//(3**i))%3 for i in range(5)]
        cand_twos = [candidate[i] == 2 for i in range(5)]
        cand_part_red = \
        [is_partially_reduced((quint//(3**i)))
        for i in range(5)]

        # Python is fine with comparing lists
        if measured_part_red == cand_part_red \
        and measured_twos == cand_twos:
            valid_candidates.append(candidate)

    # Now we have all valid candidates.
    # If they agree on a coefficient we can decide it
    n_cand = len(valid_candidates)
    sum_cand = [ sum([valid_candidates[i][j] for i \
    in range(n_cand)]) for j in range(5)]

    # We represent undecided coefficients as "4"
    recovered = [sum_cand[i]/n_cand if (sum_cand[i]\
        /n_cand).is_integer() else 4 for i in range(5)]
    return recovered

# Same as above, except we don’t know position of twos
def recover_quintuple_fp(measured_part_red):
    valid_candidates = []
    for quint in range(3**5):
        candidate = [(quint//(3**i))%3 for i in range(5)]
        cand_part_red = [is_partially_reduced(\
        (quint//(3**i))) for i in range(5)]

        if measured_part_red == cand_part_red:
            valid_candidates.append(candidate)

    n_cand = len(valid_candidates)
    sum_cand = [ sum([valid_candidates[i][j] for i in \
    range(n_cand)]) for j in range(5)]

    recovered = [sum_cand[i]/n_cand if (sum_cand[i]\
    /n_cand).is_integer() else 4 for i in range(5)]
    return recovered

```

```

# A loop to find expected number of recovered
# coefficients. Assuming quintuples are equally
# probable, and that key-length is divisible by 5
total_recovered_f = 0
total_recovered_fp = 0
for i in range(3**5):
    # What we would have observed in measurements
    measured_twos = [2 == (i//(3**j))%3 for j in range(5)]
    measured_part_red = [is_partially_reduced((i//(3**j)))\
    for j in range(5)]

    recovered_f = recover_quintuple_f(measured_part_red, \
    measured_twos)
    recovered_fp = recover_quintuple_fp(measured_part_red)
    for j in range(5):
        total_recovered_f += recovered_f[j] < 4
        total_recovered_fp += recovered_fp[j] < 4
print("Expected percentage of recovered coefficients:")
print("f:", 100*total_recovered_f/(3**5 * 5))
print("f_p:", 100*total_recovered_fp/(3**5 * 5))
```