In-depth Analysis of Side-Channel Countermeasures for CRYSTALS-Kyber Message Encoding on ARM Cortex-M4

Hauke Malte Steffen, Lucie Johanna Kogelheide, and Timo Bartkewitz

Division for Hardware Evaluation,
TÜV Informationstechnik GmbH, TÜV NORD Group,
Essen, Germany.
{h.steffen,l.kogelheide,t.bartkewitz}@tuvit.de

Abstract. A variety of post-quantum cryptographic schemes are currently undergoing standardization in the National Institute of Standards and Technology's post-quantum cryptography standardization process. It is well known from classical cryptography that actual implementations of cryptographic schemes can be attacked by exploiting side-channels, e.g. timing behavior, power consumption or emanation in the electromagnetic field. Although several of the reference implementations currently in the third and final standardization round are – to some extent – implemented in a timing-constant fashion, resistance against other side-channels is not taken into account yet.
Implementing sufficient countermeasures, however, is challenging. We therefore exemplarily examine CRYSTALS-Kyber, which is a lattice-based key encapsulation mechanism currently considered as a candidate for standardization. By analyzing the power consumption side-channel during message encoding we develop four more and compare six different implementations with an increasing degree of countermeasures.
We show that introducing randomization countermeasures is crucial as all examined implementations aiming at reducing the leakage by minimizing the Hamming distance of the processed intermediate values only are vulnerable against single-trace attacks when implemented on an ARM Cortex-M4.

Keywords: Post-Quantum Cryptography · NIST Competition · Message Encoding · CRYSTALS-Kyber · Side-Channel Analysis.

1 Introduction

Quantum computers have been a merely theoretical construction for many decades. However, during the last years significant progress has been made and increasingly large quantum computers have been built [14, 15]. A cryptographically relevant quantum computer threatens today's most wide-spread asymmetric cryptographic schemes, namely Rivest-Shamir-Adleman (RSA) and Elliptic

Curve Cryptography (ECC). These schemes rely on either the integer factorization problem or the discrete logarithm problem which a quantum computer can efficiently solve using Shor's algorithm [30]. For industries with products in the field for a long time (e.g. automotive) or data that might be valuable even decades from now (e.g. health data) the transition to quantum resistant cryptographic schemes therefore has to be initiated as soon as possible [19, 20, 29].

The field of Post-Quantum Cryptography (PQC) is based on mathematical problems that are hard to solve for both classical and quantum computers, thereby offering suitable replacement candidates for RSA and ECC. The most prominent standardization effort for PQC is conducted by the National Institute of Standards and Technology (NIST) in their PQC standardization process [23]. The Key Encapsulation Mechanism (KEM) CRYSTALS-Kyber is a third round candidate of the NIST PQC standardization process [22].

Side-channel attacks exploit channels which unintentionally carry data dependent information, e.g. power consumption or timing behavior [8, 12, 16, 17]. By monitoring these channels during execution of a security critical function an attacker might extract secret data. Side-channel attacks thereby do not focus on attacking the algorithm itself but on a potentially insecure implementation. Side-channel attacks also apply to PQC schemes and resistance against side-channel attacks is an evaluation criteria in the third and final round of the NIST PQC standardization process [18].

Simple Power Analysis (SPA) aims at extracting a secret by measuring only one execution of the security relevant function while Differential Power Analysis (DPA) requires an attacker to record a certain number of traces in order to perform an attack. In general, DPA is considered the more powerful attack technique. However, if an SPA does succeed the results are devastating as only a single trace is enough to attack the implementation. Side-channel resistance of the remaining candidates in the NIST PQC standardization process has for example been investigated in [26, 31, 32], with CRYSTALS-Kyber being one of the examined – and vulnerable – candidates. The second round candidate NewHope proved vulnerable against SPA, with the authors suggesting that a nearly identical attack path could also be applied to CRYSTALS-Kyber [1].

To counter SPA and DPA, masked implementations of CRYSTALS-Kyber have been proposed [4, 7, 10]. Masking reduces side-channel leakage by processing data in shares. An attacker can only recompute the original value if she can correctly recover all involved shares. However, in case of a high SPA success rate, conducting an SPA on the involved shares becomes a feasible attack path. Therefore, on top of examining sufficiency of masking schemes themselves [5] it might be necessary to implement additional countermeasures.

This work aims at comparing countermeasures which are applicable on top of a masking approach. CRYSTALS-Kyber hereby is merely chosen as an exemplary PQC scheme, as both publications on side-channel vulnerabilities as well as first suggestions on how to mitigate the threat do exist.

The following chapters are organized as follows: Section 2 briefly introduces CRYSTALS-Kyber. Section 3 outlines several attack paths motivating the se-

lection of the message encoding step for the attacks conducted in this work. Section 4 presents the six different implementations which have been examined introducing the subsequently added countermeasures for each of the implementations.

Following the attack path lined out for NewHope in [1], we first evaluate the reference implementation submitted to the third round of the NIST PQC standardization process [2]. The second implementation is based on an approach to reduce the Hamming distance of the leaking values as suggested by Amiet et al. [1]. For the third implementation, we introduce the use of a dummy polynomial aiming at hiding the processing of the involved coefficients. The fourth implementation on top of that balances the look-ups of the involved polynomials. For the fifth implementation, we use randomness to invert the order in which the polynomials are processed. The sixth implementation then fully randomizes the order in which the involved data is processed.

Section 5 contains the experimental results for each of the implementations with all but the last implementation failing to withstand the conducted attacks. Summing up the experimental results, Section 6 comes to the conclusion that relatively simple countermeasures are not sufficient to prevent an SPA. Therefore, more sophisticated countermeasures have to be developed to secure PQC not only against SPA but also against the more powerful DPA. We show that randomization countermeasures can reduce the SPA success rate to random guessing, making these countermeasures a potentially beneficial extension even for masked implementations.

2 Background on CRYSTALS-Kyber

Kyber is an IND-CCA2-secure KEM originally published in [6]. To obtain CCA-security, Kyber applies a variant of the Fujisaki–Okamoto (FO) transform [11] to the CPA-secure Public Key Encryption (PKE) scheme Kyber.CPAPKE. In general, KEMs are used by the communicating parties to generate shared keys for symmetric encryption allowing them to establish a secure communication channel. PKE is used to transmit encrypted data between the participants while processing the KEM.

Kyber is parametrized by a set of chosen integers. The security strength of the exchanged symmetric keys is basically determined by n which also defines the ring together with prime number q within this lattice-based scheme.

Algorithm 1 describes the encapsulation of the Kyber KEM scheme. For each execution of the encapsulation, the message m is randomly chosen and hashed by the initiator. Afterwards, m and the hash of the public key pk are hashed into the preliminary key $\bar{K}$ and into the random coins r. Thereafter, pk, m, and r are given to the encryption function of the PKE scheme (line 4, Algorithm 1), which is described in Algorithm 2. The shared key K is derived from the preliminary key $\bar{K}$, and the ciphertext c is sent to the responder. The symmetric primitives $\mathrm{H}(\cdot)$, $\mathrm{G}(\cdot)$, and $\mathrm{KDF}(\cdot)$ are preferably instantiated by SHA3-256, SHA3-512, and SHAKE-256, respectively [3, 21].

Algorithm 1 KYBER.CCAKEM.Enc(pk): encapsulation [3]

Input: Public key pk
Output: Ciphertext c
Output: Shared key K
1: $m \leftarrow \{0,1\}^{256}$
2: $m \leftarrow \mathrm{H}(m)$
3: $(\bar{K},\, r) := \mathrm{G}(m||\mathrm{H}(pk))$
4: $c := \mathrm{KYBER.CPAPKE.Enc}(pk,\, m,\, r)$
5: $K := \mathrm{KDF}(\bar{K}||\mathrm{H}(c))$
return (c, K)

The random message m is the only unknown session related variable and has to remain secret while it is incorporated as fresh entropy during the encapsulation. With knowledge about m, it is possible to reconstruct the encapsulation, and hence compute the shared key K.

During the encryption (line 5, Algorithm 2), m is given to the Decode($\cdot$) function. This processing, also denoted as *message encoding*, only bases on the message itself, potentially leaking information about the message m. Please note that the Decode($\cdot$) function is further discussed in Section 4.1 which describes the implementation of the message encoding step for the reference implementation [2].

Algorithm 2 KYBER.CPAPKE.Enc(pk, m, r): encryption [3]

Input: Public key pk
Input: Message m
Input: Random coins r
Output: Ciphertext c
1: $\hat{\mathrm{t}} := \mathrm{Decode}(pk)$
2: $\hat{\mathrm{A}} := \mathrm{Sample}(pk)$
3: $(\hat{\mathrm{r}},\, \mathrm{e}_1,\, \mathrm{e}_2) := \mathrm{Sample}(r)$
4: $\mathrm{u} := \mathrm{NTT}^{-1}(\hat{\mathrm{A}}^T \circ \hat{\mathrm{r}}) + \mathrm{e}_1$
5: $\mathrm{v} := \mathrm{NTT}^{-1}(\hat{\mathrm{t}}^T \circ \hat{\mathrm{r}}) + \mathrm{e}_2 + \mathrm{Decompress}(\mathrm{Decode}(\mathrm{m}))$
6: $c_1 := \mathrm{Encode}(\mathrm{Compress}(\mathrm{u}))$
7: $c_2 := \mathrm{Encode}(\mathrm{Compress}(\mathrm{v}))$
return $c = (c_1||c_2)$

As a result of the applied FO transform, the decrypted message m' is re-encrypted and compared with the received ciphertext c during the decapsulation, which is described in line 6 of Algorithm 3. Therefore, the message encoding can be targeted at both participating sides of the KEM.

During the CPAPKE decryption, the Decode($\cdot$) function, which is not explicitly described in this section, processes not only the message m but also the secret key sk (see [3]).

Algorithm 3 KYBER.CCAKEM.Dec(sk, c): decapsulation [3]

Input: Secret key sk
Input: Ciphertext c
Output: Shared key K

1: pk := $sk + 12 \cdot k \cdot \frac{n}{8}$
2: h := $sk + 24 \cdot k \cdot \frac{n}{8} + 32$
3: z := $sk + 24 \cdot k \cdot \frac{n}{8} + 64$
4: m' := KYBER.CPAPKE.Dec(sk, c)
5: $(\bar{K}', r')$:= G($m'||h$)
6: c' := KYBER.CPAPKE.Enc(pk, m', r')
7: **if** $c = c'$ **then**
8: K := KDF($\bar{K}'||$H(c))
9: **else**
10: K := KDF($z||$H(c))
11: **end if**

return K

3 Side-channel Attack Paths Against CRYSTALS-Kyber

This section aims at categorizing several potential side-channel attack paths allowing for a reconstruction of the shared key. To generate a shared key basically only two non-public variables are involved: the session-bound random message m and a participant's secret key. The random message m is generated during the key encapsulation (Algorithm 1) and transformed into the ciphertext c utilizing the public key. Consequently, if an attacker manages to obtain m, she can easily compute the shared key.

The secret key, by contrast, is utilized within the key decapsulation to recover m or rather m' from c (Algorithm 3). Hence, if an attacker manages to obtain the secret key, she can compute the shared key as well.

Potentially, side-channels might leak sufficient information on these two variables and also, trivially, side-channels might leak sufficient information on the shared key itself during its generation.

Different attack categories can be considered: attack category 1 aims at recovering the message m by attacking the key encapsulation. Attack categories 2 and 3 concentrate on the secret key, and the shared key, respectively. Naturally, attacks on the message (cat. 1) and the shared key (cat. 3) only allow for intercepting a single session and need to be continuously repeated, whereas attacks on the secret key (cat. 2) would allow for intercepting all sessions established with the same secret key. This work focuses on attacks on the message with the attack paths A to E explained in the following paragraph. Category 2 and 3 are not discussed any further throughout this work but, nevertheless, require as much attention as category 1 in order to create a properly secured implementation.

- **Attack Path 1.A** The random message m is freshly generated for each encapsulation call (Algorithm 1, line 1), and thus can be observed only once. Therefore, an SPA with horizontal attacks is feasible. Especially the fetch

from a random number source as well as the move from such a source to a dedicated Random-Access Memory (RAM) variable is in the focus of this attack path.

- **Attack Path 1.B** The random message m is fed into the hash function $H(\cdot)$ (Algorithm 1, line 2). Here, m is most likely moved from its dedicated RAM variable to an interface RAM variable of the hash function. The same holds true for the output $H(m)$ that simply replaces m in the remainder of the encapsulation. Additionally, the message treatment, e.g. the message scheduling for SHA3-256, could be exploited. Again, an SPA with horizontal attacks is the method of choice.

- **Attack Path 1.C** This attack path (Algorithm 1, line 3) is similar to path B. With the input of $G(\cdot)$, an attacker might obtain m, whereas the 256 most significant bits of the output $\bar{K}$ of $G(m||H(pk))$ could directly be used to compute the shared key K since the ciphertext c is public. Note that r is not of interest since it can be computed if m is recovered or not required if $\bar{K}$ is recovered. Though, r alone cannot be used as an attack vector.

- **Attack Path 1.D** This attack path is related to the message encoding (Algorithm 1, line 4) utilized within the CPAPKE encryption (Algorithm 2, line 5). We refer to the following section where we elaborate on that process.

- **Attack Path 1.E** This attack path (Algorithm 1, line 5) is likewise similar to path B and C. With the input of $KDF(\cdot)$, an attacker might obtain $\bar{K}$, whereas the output is the shared key K.

Summarizing, each step of the key encapsulation is suitable to recover the shared key if side-channels leak sufficient information.

We decided to concentrate on attack path 1.D for several reasons: on the one hand there are already published works dealing with securing the message encoding [1] and on the other hand the message m is processed in bitwise manner during the message encoding. In contrast, the other attacks paths only allow for observing the target variables while they are moved. It is widely believed in side-channel attacks that the smaller the portion of the target variable the better the exploitability in case side-channels leak sufficient information on that portion. Therefore, attacks on the message encoding are presumably more hazardous.

In order to secure an implementation against side-channel attacks, a variety of countermeasures can be considered. Masked implementations have been proposed, e.g. by Reparaz et al. [28], and Oder et al. [25], with masked encoding functions processing two shares individually. However, in case of a high SPA success rate both shares might be determined by an attacker who can thereby reconstruct the original message. Reparaz et al. [27] also presented an additively homomorphic Ring-Learning-With-Errors masking that does not require a masked encoder and uses an unprotected encoding function. Consequently, in case single-trace SPA attacks apply flawlessly (which we demonstrate in Section 5.2), masking in shape of sharing is not effective at all.

4 Message Encoding With Countermeasures

Focusing on the message encoding step (see Section 3, attack path 1.D), we implement and attack different countermeasures. The third up to the sixth implementation candidates have been designed and developed through the course of this work. Please note that we do not claim full effectiveness. Our selection of countermeasures shall rather demonstrate how to proceed to minimize side-channel leakage only by modifying the message encoding algorithm without introducing complex masking or hiding schemes. The following approaches are explained in detail in Section 4.1 to Section 4.6:

1. The message encoding step as implemented in the reference implementation [2] without additional countermeasures against SPA.
2. An implementation of the message encoding according to [1], which aims at reducing the Hamming distance of the leaking values based on a multiplicative approach.
3. A dummy polynomial is included aiming at hiding the processing of the involved coefficients.
4. The preceding approach is improved by balancing the look-ups of the polynomials, leading to a Hamming distance independent of the processed bits.
5. The order of the processed polynomials is randomly inverted for each execution of the encoding, changing the signature of the processed bit in the power side-channel.
6. Additionally, the processed bytes and bits are randomly shuffled for each execution of the encoding.

For all presented implementations, an SPA is carried out aiming at recovering the value of the processed bits by examining a single trace. Within the implementations, the message m, and the prime q are denoted as `msg`, and `KYBER_Q`, respectively.

4.1 Message Encoding According to Reference Implementation

Listing 1 presents the reference implementation [2] of the message encoding function as submitted to the third round of the NIST PQC standardization process. It takes a 32-byte message `msg` as an input and converts it to a polynomial `r` of degree 256. To this end, the function iterates in a bitwise manner over `msg` and sets a coefficient of `r` either to 0 or to the constant `(KYBER_Q+1)/2`, depending on whether the bit of `msg` is 0 or 1.

To set the coefficients of the polynomial to the correct value in line 6 of Listing 1, a mask is calculated which is either 0x0000 or 0xFFFF, depending directly on a single bit of `msg`.

```
1  void poly_frommsg(poly *r, const uint8_t msg[
   KYBER_INDCPA_MSGBYTES]) {
2     unsigned int i,j;
3     int16_t mask;
4     for(i=0;i<KYBER_N/8;i++) {
5        for(j=0;j<8;j++) {
6           mask = -(int16_t)((msg[i] >> j) & 1);
7           r->coeffs[8*i+j] = mask & ((KYBER_Q+1)/2);
8        }
9     }
10 }
```

Listing 1: CRYSTALS-Kyber – Message Encoding [2]

The two values of the mask have the maximum possible Hamming distance of 16, i.e. all bit positions differ. We assume that this leads to a distinguishable difference in the amount of power consumption when processing the mask, and thus allows for extracting information on the actual secret message as for each bit of `msg` the value of the mask is evaluated again.

4.2 Message Encoding With Multiplication

Amiet et al. [1] presented an approach to make the attack more difficult by reducing the Hamming distance between the two possible values of the mask. To encode a message, the coefficients of the polynomial are calculated by multiplying the message bit and `(KYBER_Q+1)/2`. Hence, line 6 and 7 in Listing 1 are replaced by Listing 2. The two possible values for the mask are 0 and 1 reducing the maximum possible Hamming distance from 16 to one.

```
6     mask = ((msg[i] >> j) & 1);
7     r->coeffs[8*i+j] = mask*((KYBER_Q+1)/2);
```

Listing 2: CRYSTALS-Kyber – Message Encoding with multiplication [1]

Decreasing the Hamming distance should result in reduction of the observed leakage, however, an SPA, as described in [1], might still be applicable.

4.3 Message Encoding Using Data Independent Polynomial Generation

To counteract vulnerabilities still present in the previous implementation, we first remove the mask evaluation as it may leak information about the message bit during its storing and loading instructions. Furthermore, information leakage is reduced by generating polynomials in a data independent fashion: additionally to the already provided `r`, we define a second polynomial `r_d` which is discarded after the message encoding. We first initialize all coefficients of `r` and `r_d` to the constant `(KYBER_Q+1)/2`. Afterwards, each time a single bit of `msg` is processed, one coefficient of one of the polynomials is set to zero. If the extracted message

bit is zero, the coefficient of the real polynomial `r` is altered, otherwise the coefficient of the dummy polynomial `r_d` is altered. The reference implementation is modified by replacing all lines from line 3 onwards in Listing 1 by Listing 3.

```
3   poly r_d;
4   poly *p_r[2] = {r, &r_d};
5   for(i=0;i<KYBER_N;i++) {
6      r->coeffs[i] = (KYBER_Q+1)/2;
7      r_d.coeffs[i] = (KYBER_Q+1)/2;
8   }
9   for(i=0;i<KYBER_N/8;i++) {
10     for(j=0;j<8;j++) {
11        p_r[(msg[i] << (7-j)) >> 7]->coeffs[8*i+j] = 0;
12     }
13  }
```

Listing 3: CRYSTALS-Kyber – Message Encoding using data independent polynomial generation

In contrast to the previous implementations, information leakage should be reduced significantly as the very same operation of setting a polynomial to zero is performed each time independently of the processed value. Remaining leakage could still be caused by determining which polynomial should be used, based on the currently evaluated bit of `msg`.

4.4 Message Encoding Using Data Independent Polynomial Generation With Balanced Byte Look-Up

We extend the previously introduced approach by balancing the look-ups of the polynomials by covering the extracted message bits with alternating masks. To do so, we initialize a pointer array `p_r` of size 256 alternately containing both polynomials. Furthermore, we define two mask values with identical Hamming weight for later balancing of the look-ups. Line 4 of Listing 3 is replaced by Listing 4. We remark that the code presented in Listing 4 can be placed outside the message encoding function.

```
4   poly *p_r[256];
5   uint32_t xorMasks[2] = {0xaaaaaaaa, 0x55555555};
6   for(i=0;i<256;i+=2) {
7      p_r[i] = r;
8      p_r[i+1] = &r_d;
9   }
```

Listing 4: CRYSTALS-Kyber – Message Encoding using data independent polynomial generation with balanced byte look-up – Initialization

While processing the bits of `msg` the index of `p_r` is calculated as an 8 bit value with Hamming distance independent of the processed bits for each message byte. This corresponds to replacing line 11 in Listing 3 with Listing 5.

```
11    p_r[((xorMasks[j & 1] ^ msg[i]) >> j) & 0xff]->
       coeffs[8*i+j] = 0;
```

Listing 5: CRYSTALS-Kyber – Message Encoding using data independent polynomial generation with balanced byte look-up – Balanced look-up

As a result, potential information leakage depending on the polynomial look-ups should be reduced. Remaining leakage might be caused by the the data dependency of the addressed polynomial as well as the evaluated message bit.

4.5 Message Encoding Using Polynomial Randomization

In this section, we present an additional measure to decrease leakage of the polynomial processing. The strategy is to shift the pointer array `p_r` and the balancing array `xorMask` by 0 or 1, depending on the most significant bit of the first message byte (MSB). As the message `msg` is randomly chosen, evaluating the MSB serves as a source of randomness without introducing an additional fetch from a random number generator. To first extend the used arrays, we replace line 4 of Listing 4 by Listing 6.

```
4    poly *p_r[256+1];
5    uint32_t xorMasks[3] = {0xaaaaaaaa,0x55555555,0xaaaaaaaa};
```

Listing 6: CRYSTALS-Kyber – Message Encoding using polynomial randomization – Initialization

In order to randomly invert the polynomial look-ups, the arrays are shifted to the left by adding Listing 7 after line 8 of Listing 3.

```
9    uint32_t b_inv = ((0xaaaa00aa ^ msg[0]) >> 7) & 0xff;
10   for(i=0; i<255;i++) {
11      *(p_r+i) = *(p_r+i+b_inv);
12   }
13   for(i=0; i<2;i++) {
14      *(xorMasks+i) = *(xorMasks+i+b_inv);
15   }
```

Listing 7: CRYSTALS-Kyber – Message Encoding using polynomial randomization – Inversion by shifting

Compared to the preceding implementation, information leakage should further decrease. But again, processing the polynomials can still cause small differences in the amount of power consumption. We furthermore remark that this implementation introduces a new data dependency based on the most significant bit of the MSB.

4.6 Message Encoding Using Byte and Bit Level Random Ordering

We extend the previous approach by shuffling the order in which the bytes and their bits are processed. Again, the MSB is used as a source of randomness. We define two masking variables `i_m`, and `j_m` to shuffle the bytes and their bits. Therefore, line 9 of Listing 7 is replaced by Listing 8.

```
9    uint32_t rand = (0xaaaa00aa ^ msg[0]);
10   uint8_t i_m = rand & 0x1f;
11   uint8_t j_m = (rand >> 5) & 0xff;
12   uint32_t b_inv = (rand >> 7) & 0xff;
```

Listing 8: CRYSTALS-Kyber – Byte and bit level random ordering – Initialization

The shuffling variables `i_m`, and `j_m` are added to the loop counters `i`, and `j`, respectively while iterating through the bytes and bits of `msg` by an exclusive-or. Therefore, the order of the bytes is shuffled and the bits of each byte are processed in the same but randomized order. For this, we replace line 9 onwards in Listing 3 by Listing 9.

```
9    uint8_t i_r, j_r;
10   for(i=0;i<KYBER_N/8;i++){
11      i_r = i ^ i_m;
12      for(j=0;j<8;j++){
13         j_r = j ^ j_m;
14         p_r[((xorMasks[(j_r & 1)] ^ msg[i_r]) >> j_r) & 0xff
     ]->coeffs[8*i_r+j_r] = 0;
15      }
16   }
```

Listing 9: CRYSTALS-Kyber – Byte and bit level random ordering – Shuffled look-ups

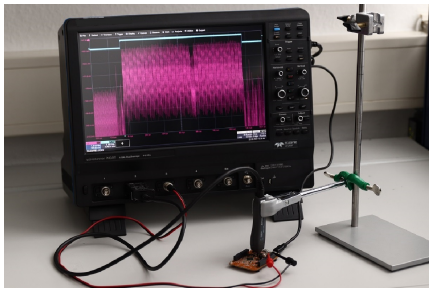
Introducing this level of randomization should significantly reduce the observed leakage. However, the masking variables in Listing 8 themselves become a target of side-channel analysis, potentially requiring additional protection.

5 Experimental Results

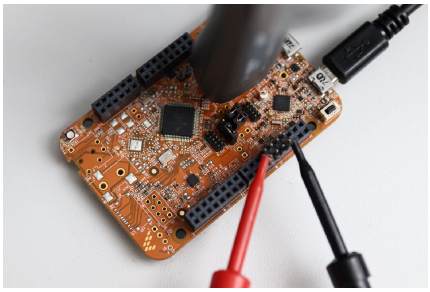
This section presents our practical results of the side-channel analysis of Kyber's message encoding, targeting all implementations listed in Section 4.1 to Section 4.6.

5.1 Measurement Setup

Figure 1 shows the setup targeting the Cortex-M4 processor (@120 MHz) on the FRDM-K22F development board (rev. D) programmed with the MCUXpresso software development kit (11.2.1) to prepare the board for our measurements [24]. Hardware modifications are necessary – all capacitors between the measuring point and the power pins of the Cortex-M4 have been removed – in order not to degrade the power consumption signal. Power traces are recorded utilizing a populated resistor with a Teledyne LeCroy AP033 active differential probe attached to a Teledyne LeCroy HDO9404M. The horizontal resolution of the oscilloscope is set to 0.1 ns, i.e. 10 GS/s sampling rate. A dedicated trigger



(a) Teledyne LeCroy oscilloscope



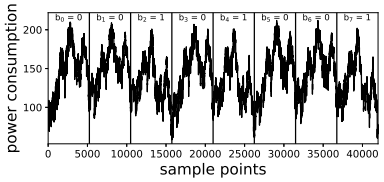
(b) FRDM-K22F development board

Fig. 1: Measurement setup

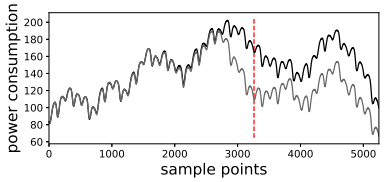
signal – a signal pulse framing the encoding – is utilized via a general purpose pin. Thus, only minimal alignment (via cross-correlation) is needed.

5.2 Message Encoding According to Reference Implementation

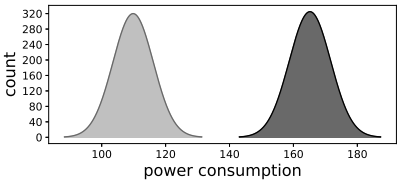
First, the reference implementation according to [2] is targeted, involving processing of a 16 bit mask for the encoding of each bit of the message. An exemplary power trace as well as the analysis results are depicted in Figure 2.



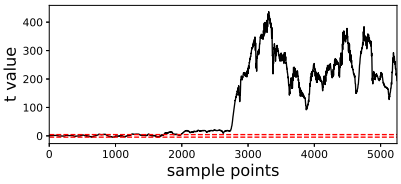
(a) Power consumption while processing one byte



(b) Mean traces for processing 0 (black) and 1 (gray) with the maximum distance marked by the red line



(c) Distribution of power consumption



(d) t-test result

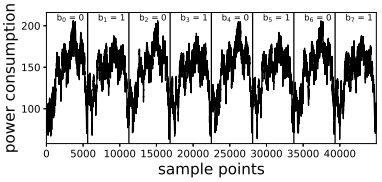
Fig. 2: Side-channel analysis of message encoding according to the reference implementation [2]

Figure 2a depicts the power consumption while processing one byte. Whether a 0 or a 1 is processed results in a clearly distinguishable pattern in the power

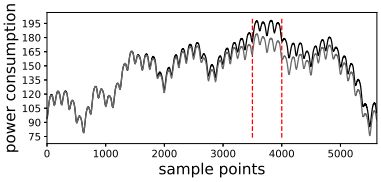
trace allowing for extraction of the message `msg` by observation of a single trace with the bare eye. Focusing on processing one bit only, Figure 2b shows the mean traces for the two classes 0 and 1 for a total of 8,000 involved traces. The red vertical line marks the sample for which the difference between the mean traces reaches its maximum. For this sample, all traces are analyzed, resulting in Figure 2c which depicts the means' distributions which we assume to be Gaussian for the two classes 0 and 1. Whereas the means are significantly distinguishable, the variances are very close. Performing a t-test over the whole sample range, with considered noise thresholds according to [9], Figure 2d is obtained. On top of the t-test a single-trace SPA is performed which results in a success rate[1] of 100.0 %. The available traces were halved for profiling as well as matching and Points Of Interest (POI) were selected by Sum Of Squared pairwise T-differences (SOST) [13].

5.3 Message Encoding With Multiplication

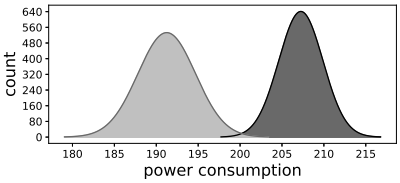
In order to reduce the Hamming distance of the processed internal values, the calculation of the 16 bit mask is replaced by a multiplication operation according to the approach suggested in [1] and outlined in Section 4.2. Each bit of the message `msg` is multiplied with the constant `(KYBER_Q+1)/2`. Thus, the evaluated mask is either equal to 0 or 1.



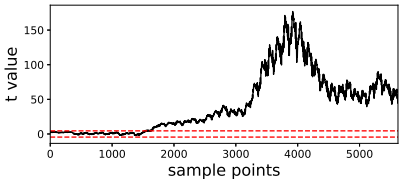
(a) Power consumption while processing one byte



(b) Mean traces for processing 0 (black) and 1 (gray) with the maximum distance located within the red lines



(c) Distribution of power consumption



(d) t-test result

Fig. 3: Side-channel analysis of message encoding with multiplicative mask

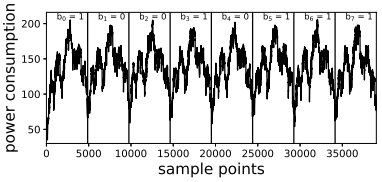
Figure 3a illustrates a single trace of the power consumption. Though the shape of the trace changed in comparison to Figure 2a, significant differences can still

[1] Proportion of correctly classified traces, i.e. $r_S = \frac{\text{Correctly classified traces}}{\text{Number of traces}}$.

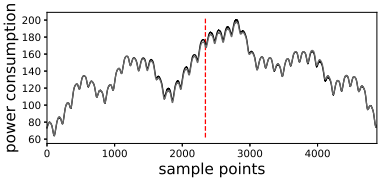
be observed for the two classes. Thus, a single-trace attack by observation with the bare eye is still possible. The mean traces for processing 0 and 1 are shown in Figure 2b. As for the reference implementation, a total of 8,000 traces is analyzed. Figure 2c depicts the power distribution for the two classes 0 and 1 as extracted from the sample with the highest power consumption within the timeframe marked by the red vertical lines in Figure 2b. Whilst the two distributions move closer together and the variances are more distinguishable compared to the reference implementation, the overlap is still negligible. This is reflected in the t-test result, depicted in Figure 2d. The SPA success rate is still 100.0 %, despite reduction of Hamming distance by the multiplication approach. Presumably due to the high horizontal oscilloscope resolution, the leakage is fully exploitable leading to a flawless single-trace SPA.

5.4 Message Encoding Using Data Independent Polynomial Generation

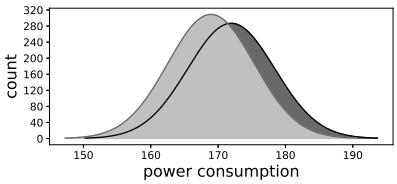
As leakage of the mask could be exploited even for a Hamming distance of only one, an alternative approach is examined which does not require a mask (compare Section 4.3). Instead, a dummy polynomial is used, and for each bitwise encoding step, a coefficient of either the polynomial `r` or its dummy counterpart `r_d` is set to zero. Pointers to the real and the dummy polynomial are stored in a pointer array. Therefore, the same operation is performed for each encoding step. When all bits have been processed, the dummy polynomial is discarded.



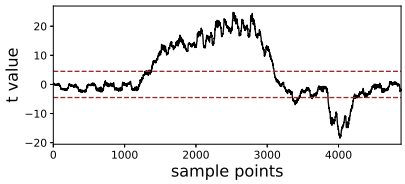
(a) Power consumption while processing one byte



(b) Mean traces for processing 0 (black) and 1 (gray) with the maximum distance marked by the red line



(c) Distribution of power consumption



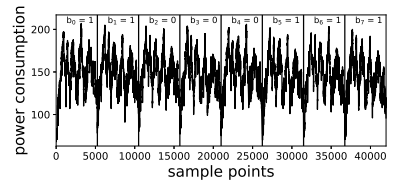
(d) t-test result

Fig. 4: Side-channel analysis of message encoding with data independent polynomial generation

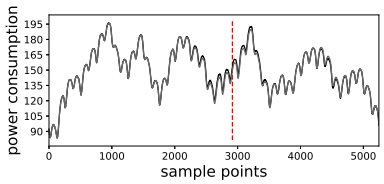
In contrast to the previous measurements, it is indistinguishable to the bare eye whether a 0 or 1 is processed (compare Figure 4a). Figure 4b depicts the mean traces for processing 0 and 1 with small differences in the mean power consumption still identifiable for the two classes. Figure 4c depicts the distributions of the measured power values for one selected sample for the two classes. The distributions lie closer together and the overlap is strongly increased, however, the variance of the two classes significantly differs. Figure 4d shows the corresponding t-test result which still indicates information leakage. When an SPA is conducted, the success rate noticeably drops to 68.6 % compared to the attacks on the previous implementations.

5.5 Message Encoding Using Data Independent Polynomial Generation with Balanced Byte Look-Up

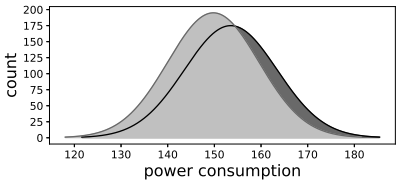
Aiming at further reducing the remaining leakage, the look-ups of the polynomials are balanced, meaning that the selection of whether an operation shall be performed on the real or the dummy polynomial is done with the help of two masking values with identical Hamming weight (compare Section 4.4).



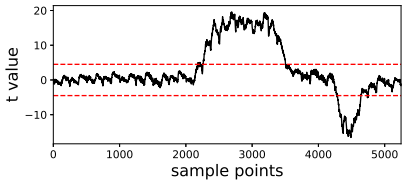
(a) Power consumption while processing one byte



(b) Mean traces for processing 0 (black) and 1 (gray) with the maximum distance marked by the red line



(c) Distribution of power consumption



(d) t-test result

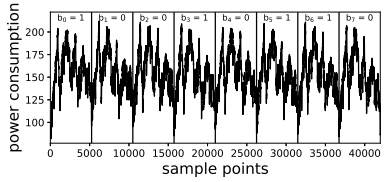
Fig. 5: Side-channel analysis of message encoding with data independent polynomial generation with balanced byte look-up

The power trace depicted in Figure 5a cannot be interpreted by the bare eye only. Comparing the average traces for both classes, subtle differences are still visible (compare Figure 5b). This results in the two distributions depicted in Figure 5c not fully overlapping and also differing in their variance. The t-test as presented in Figure 5d accordingly yields results above the noise threshold.

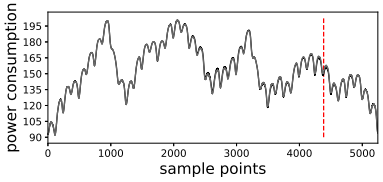
The conducted SPA results in a success rate of 67.9 %, nearly as high as the unbalanced implementation shown in the previous section.

5.6 Message Encoding Using Polynomial Randomization

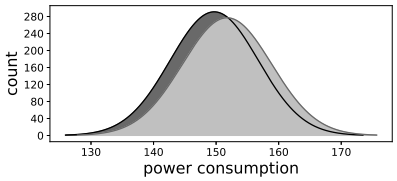
In order to introduce greater variance, the ordering of real and dummy polynomials within the pointer array is randomized for each function call (compare Section 4.5). Thereby, leakage caused by accessing the same index values over and over again shall be reduced. However, the distributions in Figure 6c as well as the t-test results in Figure 6d indicate that the leakage is only slightly reduced.



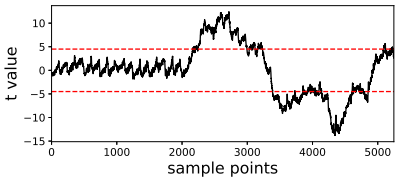
(a) Power consumption while processing one byte



(b) Mean traces for processing 0 (black) and 1 (gray) with the maximum distance marked by the red line



(c) Distribution of power consumption



(d) t-test result

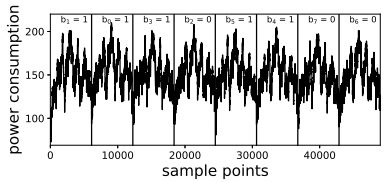
Fig. 6: Side-channel analysis of message encoding with polynomial randomization

The conducted SPA still yields a success rate of 64.0 %.

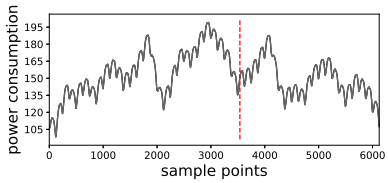
5.7 Shuffled Message Encoding Using Byte and Bit Level Random Ordering

The last investigated implementation shuffles the processed message bytes as well as the order in which the bits of each byte are processed (compare Section 4.6).

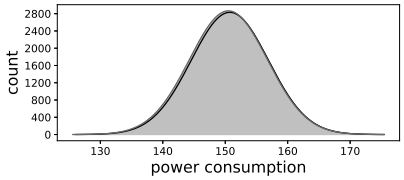
In contrast to the previous tests the number of analyzed traces is increased from 8,000 traces to 80,000 traces. To this end, Figure 7d shows the t-test result for ten times more traces compared to the previous implementations. Analyzing this larger trace set, the t-test yields results slightly above the noise threshold for a very limited range of samples. When only 8,000 traces are included, the t-test values remain below the noise barrier. The distributions for the two classes as shown in Figure 7c are indistinguishable from each other.



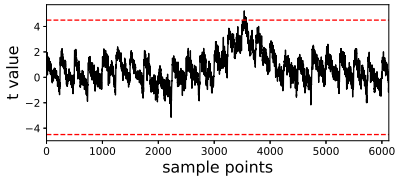
(a) Power consumption while processing one byte



(b) Mean traces for processing 0 (black) and 1 (gray) with the maximum distance marked by the red line



(c) Distribution of power consumption



(d) t-test result

Fig. 7: Side-channel analysis of message encoding with byte and bit level random ordering

Performing an SPA, the success rate reduces to 50.1 % which corresponds to random guessing.

5.8 Comparison of Countermeasures

Table 1 summarizes our results of Section 5.2 to Section 5.7 for the applied t-tests and SPA as well as the required number of clock cycles and the overhead with respect to the reference implementation. Please recall that the implementation with random byte and bit level ordering has been analyzed with a higher number of involved traces compared to all other implementations (80,000 compared to 8,000 traces).

Table 1: Comparison of Implementations

	t-test	SPA		clock cycles
Implementation	$t$max	# POI	success rate	(overhead)
Reference implementation [2]	437	1,535	100.0 %	11,732
Multiplication [1]	177	796	100.0 %	12,500 (1.09×)
Data independent polynomial gen.	24.8	525	68.6 %	16,066 (1.42×)
Balanced data independent polynomial gen.	19.6	700	67.9 %	19,425 (1.66×)
Polynomial randomization	13.8	1,231	64.0 %	26,893 (2.29×)
Byte and bit level random ordering	5.2	1,755	50.1 %	29,211 (2.49×)

POIs are selected by means of SOST. The selection of POIs for the SPA is conducted in such a way that only samples are included for which the SOST value reaches at least 20 % of the maximum SOST value. A high number of selected POIs therefore corresponds to either an implementation which can be easily

attacked (please refer to the row regarding the reference implementation) or an implementation for which nearly all measured sample points are independent of the processed data (please refer to the row regarding the implementation with random ordering). In the latter case many sample points lie above the threshold of 20 % due to the fact that the maximum SOST value itself is low.

Comparing the maximum absolute t-test values $t_{\max}$ with the SPA success rates, it can be observed that both indicators are reduced for successively added countermeasures. For the last implementation, the t-test values only slightly exceed the noise barrier and the SPA success rate reaches 50.1 % which corresponds to random guessing.

6 Conclusion

Achieving resistance against side-channel analysis is crucial for PQC implementations to make PQC schemes suitable replacement candidates for currently used asymmetric cryptographic schemes. In this work, we examined various countermeasures for the CRYSTALS-Kyber message encoding step on an ARM Cortex-M4. For a total of six different implementations, we performed a side-channel analysis targeting the power domain. The amount of leakage is classified using a t-test, then, an SPA is conducted targeting the processing of individual bits.

Masking only is a suitable countermeasure if the success rate for an SPA is lower than 100 %. However, for the first two examined implementations, the value of the processed bit can be read from the power trace with the bare eye. Processing this value in a number of shares would not improve side-channel resistance as the shares could be attacked with the same success rate leading to full recovery of the message.

The Cortex-M4 shows significant leakage even for already protected implementations, e.g. accessing a dummy in comparison to the real polynomial still results in exploitable leakage, which leads us to the conclusion that it is a challenging task to implement an algorithm in a side-channel secured fashion on this hardware. The most promising countermeasure which we could identify is full randomization of the order of the processed bits. It is therefore considered beneficial to introduce randomization countermeasures even on top of masked implementations.

The message decoding function is the inverse operation of the examined message encoding. In order to apply the presented randomization approach to the decoding step, however, one byte has to be decoded first to serve as the source of randomness. To minimize leakage at this point in time, the independent polynomial generation countermeasure could be applied.

Furthermore, the same message is encoded twice, first by the initiator and then by the responder during the re-encryption. In combination with the decoding step, the very same message could be attacked up to three times. However, choosing different but static randomization bytes prevents such an attack.

When a randomization approach is selected, fetching and processing random numbers becomes a suitable target for side-channel analysis and has to be implemented in a side-channel secure fashion as well.

References

1. Amiet, D., Curiger, A., Leuenberger, L., Zbinden, P.: Defeating NewHope with a Single Trace. In: Ding, J., Tillich, J. (eds.) Post-Quantum Cryptography, PQCrypto 2020. LNCS, vol. 12100, pp. 189–205. Springer (2020)
2. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehle, D.: NIST Submission Package for round 3 (2020), `https://pq-crystals.org/kyber/resources.shtml`
3. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehle, D.: CRYSTALS - Kyber: Algorithm Specifications And Supporting Documentation (version 3.01) (2021), `https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf`
4. Bache, F., Paglialonga, C., Oder, T., Schneider, T., Güneysu, T.: High-Speed Masking for Polynomial Comparison in Lattice-based KEMs. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 483–507 (2020)
5. Bhasin, S., D'Anvers, J.P., Heinz, D., Pöppelmann, T., Beirendonck, M.V.: Attacking and Defending Masked Polynomial Comparison for Lattice-Based Cryptography. Cryptology ePrint Archive, Report 2021/104 (2021)
6. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018. pp. 353–367. IEEE (2018)
7. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C.: Masking Kyber: First- and Higher-Order Implementations. Cryptology ePrint Archive, Report 2021/483 (2021)
8. Diffie, W., Hellman, M.: New Directions in Cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (2006)
9. Ding, A.A., Zhang, L., Durvaux, F., Standaert, F., Fei, Y.: Towards Sound and Optimal Leakage Detection Procedure. In: Eisenbarth, T., Teglia, Y. (eds.) Smart Card Research and Advanced Applications – CARDIS 2017. LNCS, vol. 10728, pp. 105–122. Springer (2017)
10. Fritzmann, T., Beirendonck, M.V., Roy, D.B., Karl, P., Schamberger, T., Verbauwhede, I., Sigl, G.: Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography. Cryptology ePrint Archive, Report 2021/479 (2021)
11. Fujisaki, E., Okamoto, T.: Secure Integration of Asymmetric and Symmetric Encryption Schemes. In: Wiener, M.J. (ed.) Advances in Cryptology – CRYPTO '99. LNCS, vol. 1666, pp. 537–554. Springer (1999)
12. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2001. pp. 251–261. Springer (2001)
13. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. Stochastic Methods. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2006. pp. 15–29. Springer (2006)
14. Google: A Preview of Bristlecone, Google's New Quantum Processor (2018), `https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html`
15. IBM: IBM's Roadmap For Scaling Quantum Technology (2020), `https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap`
16. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) Advances in Cryptology – CRYPTO' 99. pp. 388–397. Springer (1999)

17. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO '96. pp. 104–113. Springer (1996)
18. Moody, D., Alagic, G., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Liu, Y.K., Miller, C., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D., Alperin-Sheriff, J.: Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process (2020)
19. Mosca, M.: Towards quantum-safe cryptography. In: Mosca, M., Lenhart, G., Pecen, M. (eds.) 1st Quantum-Safe-Crypto Workshop. pp. 39–49. ETSI (2013), https://docbox.etsi.org/Workshop/2013/201309_CRYPTO/e-proceedings_Crypto_2013.pdf
20. Mosca, M.: Cybersecurity in an Era with Quantum Computers: Will We Be Ready? IEEE Secur. Priv. **16**(5), 38–41 (2018)
21. National Institute of Standards and Technology: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 202, U.S. Department of Commerce, Washington, D.C. (2015)
22. National Institute of Standards and Technology: PQC Standardization Process: Third Round Candidate Announcement (2020), https://www.nist.gov/news-events/news/2020/07/pqc-standardization-process-third-round-candidate-announcement
23. NIST: Post Quantum Cryptography – Workshops and Timeline (2021), https://csrc.nist.gov/Projects/post-quantum-cryptography/workshops-and-timeline
24. NXP: FRDM-K22F: NXP Freedom Developement Platform for Kinetis K22 MCUs (2021), https://www.nxp.com/design/development-boards/freedom-development-boards/mcu-boards/nxp-freedom-development-platform-for-kinetis-k22-mcus:FRDM-K22F
25. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical CCA2-Secure and Masked Ring-LWE Implementation. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 142–174 (2018)
26. Ravi, P., Roy, S.S., Chattopadhyay, A., Bhasin, S.: Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 307–335 (2020)
27. Reparaz, O., de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Additively Homomorphic Ring-LWE Masking. In: Takagi, T. (ed.) Post-Quantum Cryptography – PQCrypto 2016. LNCS, vol. 9606, pp. 233–244. Springer (2016)
28. Reparaz, O., Roy, S.S., Vercauteren, F., Verbauwhede, I.: A Masked Ring-LWE Implementation. In: Güneysu, T., Handschuh, H. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2015. LNCS, vol. 9293, pp. 683–702. Springer (2015)
29. Rodriguez-Henriquez, F., Jaques, S., Lochter, M., Mosca, M.: How long can we safely use pre-quantum ECC? (2020), https://eccworkshop.org/2020
30. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
31. Sim, B., Kwon, J., Lee, J., Kim, I., Lee, T., Han, J., Yoon, H.J., Cho, J., Han, D.: Single-Trace Attacks on Message Encoding in Lattice-Based KEMs. IEEE Access **8**, 183175–183191 (2020)
32. Xu, Z., Pemberton, O., Roy, S.S., Oswald, D.: Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems with Chosen Ciphertexts: The Case Study of Kyber. Cryptology ePrint Archive, Report 2020/912 (2020)