In Mid-Stream: Removing the FO-Transform Helps against Leakage but is not Enough

Duyên Pay, Thomas Peters and François-Xavier Standaert

UCLouvain, ICTEAM Institute, Crypto Group, Louvain-la-Neuve, Belgium

Abstract. The Fujisaki-Okamoto transform is a popular solution to design post-quantum public key encryption schemes, or key encapsulation mechanisms. In order to ensure security against chosen-ciphertext attacks, it checks the validity of ciphertexts by re-encrypting decrypted messages. This operation in turn leads to severe side-channel weaknesses, because the re-encrypted messages can be made key-dependent. Hence, distinguishing them thanks to leakage is sufficient to extract (long-term) secret key information. As a result, recent works suggested to ensure the validity of ciphertexts by other means than re-encryption. For now, the main candidate for this purpose, integrated in the Polka encryption scheme (PKC 2023) and analyzed more generically by Hövelmanns et al. (EUROCRYPT 2025), is to use continuous norm checks through the decryption process. In this paper, we evaluate the extent to which replacing the FO-transform by such norm checks helps resistance against leakage. Negatively, we exhibit new attack vectors that were not anticipated in previous (heuristic) analyzes. Positively, we observe that the removal of the FO-transform nevertheless reduces the attack surface and we identify possible tracks to further minimize it. Overall, our results therefore shed light on the challenge of designing post-quantum public-key encryption schemes, or key encapsulation mechanisms, that can be efficiently protected against side-channel attacks. We hope they can inform theory about leakage sources that could be better taken over by design, to develop new schemes allowing a scarcer use of implementation-level countermeasures.

Keywords: Side-Channel Attacks · Post-Quantum Cryptography · FO-Transform

1 Introduction

The design of Post-Quantum (PQ) Key Encapsulation Mechanisms (KEMs) has been an important research goal over the last decade, culminating with the selection of CRYSTALS-Kyber as main standard by the NIST [SAB+22]. Yet, and despite continuous progresses, the efficient implementation of Kyber's decapsulation with security guarantees against physical (e.g., side-channel, fault) attacks remains a major challenge. As far as security against side-channel attacks is concerned, this is due to two attack vectors.

On the one hand, and quite unavoidably, the decryption of the lattice-based, CPA-secure, encryption scheme used by Kyber can be the target of standard DPA attacks [MOS11], directly threatening its long-term key [PPM17, PP19, HHP+21, HKG+21, KT23, WXT25]. On the other hand, a sequence of works has shown that the re-encryption step, included in the Fujisaki-Okamoto (FO) transform to check the validity of the ciphertexts in Kyber, can be the target of strong and hard to mitigate attacks [RRCB20, UXT+22, XPR+22, RRD+23]. These attacks exploit the fact that by carefully crafting (invalid) ciphertexts, the adversary can force the re-encryption of key-dependent messages. Therefore, side-channel adversaries only have to distinguish these messages thanks to leakage to gain information on Kyber's

E-mail: thi.pay@uclouvain.be (Duyên Pay)

long-term key: a considerably simpler goal than the aforementioned DPA attacks, which prompted a team from NXP to denote this situation as an "FO-Calypse" [ABF+].[1]

Distinguishing DPA attacks are (very) strong since they can easily exploit a lot of side-channel information, as reflected by the following taxonomy:

- *DPA vs. SPA:* a Differential Power Analysis (DPA) continuously accumulates information on a long-term secret s by observing leaking variables v that are expressed as a function $\mathsf{f}(s \star pl)$, with $\star$ a group operation between a secret input s and a public input pl, for an adversarially-chosen number of pl values; a SPA (Simple Power Analysis) accumulates a bounded amount of information on a such a long-term key, either because it targets leakage variables $v = \mathsf{f}(s \star pl)$ that can only be observed for a bounded number of pl values, or because these variables are independent of pl.

- *Guessing vs. distinguishing attacks:* a guessing side-channel attack extracts information about target intermediate variables of an implementation which must be exhaustively guessed, and therefore must be of limited size (e.g., 8-bit to 32-bit variables). This information can then be combined with heuristic strategies like soft analytical side-channel attacks [VGS14]; a distinguishing side-channel attack extracts information about all the target intermediate variables of an implementation, for only two values of these variables, determined by the inputs to be distinguished. Hence, this information can be directly combined (e.g., via maximum likelihood).

In other words, DPA provides side-channel adversaries with more information than SPA, and distinguishing attacks allow a direct combination of all the available leakage (vs. a heuristic combination of the leakage corresponding to target random random of tractable sizes for guessing attacks). As summarized in Table 1, this implies that the operations involved in Kyber's CPA decryption (i.e., polynomial multiplications mostly) must be masked and that the operations involved in Kyber's decapsulation for checking the validity of the ciphertexts (i.e., Keccak executions and polynomial multiplications mostly) must be shuffled and masked, leading to large overheads [BGR+21, BC22, RCDB24].

Table 1: Kyber's leakage-resistance requirements.

	CPA decryption	ciphertext validity checking	
		computation	control
targets (ops/var)	poly. multiplications	Keccak and poly. multiplications	$\bot, \top$
type of attack	guessing DPA	distinguishing DPA	$\emptyset$
countermeasure	masking	masking and shuffling	$\emptyset$

The prohibitive cost of these countermeasures in turn lead researchers to question the possibility of designing new PQ KEMs, or CCA-secure public-key encryption schemes, offering an easier path to side-channel security. Quite naturally in view of its strength, attention has first been paid to try avoiding the FO-Calypse attack vector, as witnessed by both specific and generic works. Specifically, the PKC 2023 work of Hoffmann et al. introduces Polka [HLM+23], a CCA-secure public-key encryption scheme which achieves rigidity (i.e., ensures the validity of the ciphertexts) via norm checks rather than re-encryption, on top of other interesting features for side-channel security (e.g., a randomized decryption using dummy ciphertexts, mostly key-homomorphic operations that are easy

[1] Some other pieces of work optimize generic attacks (e.g., against the number theoretic transform used in PQ cryptography) that can be exploited by the two attack vectors [AER23, HSST23].

to mask, aggressive optimizations relying on hard physical learning problems [DMMS21, HMM+23]). Its focus is on the security of the long-term key, a design goal usually denoted as leakage-resilience in symmetric cryptography [BBC+20]. Generically, the EUROCRYPT 2025 work of Hövelmanns et al. provides a comprehensive study of the properties needed to obtain such a rigidity via norm checks for PQ KEMs [HHMS25]. Its focus is more general and it does not come with a proposed instance.[2] Yet, from the viewpoint of our following investigations, the features it promotes are similar to Polka and raise as main question whether removing the FO-transform directly leads to cheaper side-channel security?

The physical security guarantees offered by schemes ensuring rigidity via norm checks are only heuristic so far: the informal conditions to obtain a leakage-resilient implementation of Polka are summarized in Table 2. As for Kyber, the CPA decryption requires security against guessing DPA. So the main advantage of removing the FO-transform presumably relates to operations needed to check the validity of the ciphertexts. In Polka, those are split between norm checks and hash function calls / Authenticated Encryption (AE) applied to the random coins output by the CPA decryption to produce a symmetric key / decrypt a message. It is expected that the first ones require only guessing SPA security while the latter ones can even leak in full if only leakage-resilience is needed.[3]

Table 2: Polka's former leakage-resilience requirements.

	CPA decryption	**ciphertext validity checking**		
		computation		**control**
targets (ops/var)	poly. multiplications	norm checks (H inputs)	(H outputs) AE	$\perp, \top$
type of attack	guessing DPA	guessing SPA	$\emptyset$	$\emptyset$
countermeasure	masking	parallelism or shuffling	$\emptyset$	$\emptyset$

In this paper, we refine the requirements of Table 2 by exhibiting two new DPA attack vectors against (the KEM part of) Polka, and discuss their general applicability to PQ KEMS and CCA-secure public-key encryption schemes. More precisely:

We first show that side-channel adversaries can craft ciphertexts so that the random coin extraction of the CPA decryption of Polka force intermediate values in the implementation to depend on its long-term key. A similar issue could pop up in any instantiations of the proposals from [HHMS25]. We further show that the dummy ciphertexts introduced in Polka during decryption in order to help with its (expected) SPA security requirements only make this attack mildly more difficult. So the guessing SPA requirements of Table 2 actually have to be changed to guessing DPA requirements, as updated in Table 3. Such attacks do not cancel the interest of the approach in [HLM+23, HHMS25], since a guessing DPA remains a significantly weaker attack vector than the distinguishing DPA against Kyber in this context. Yet, they imply that the norm checks of Polka require masking and the range checks of [HHMS25] cannot remain without protection in general.

Second, we show that the flag that is gradually updated during Polka's decryption (to indicate invalid ciphertexts), of which the leakage was expected to impact the scheme's leakage resistance only in [HLM+23] (i.e., the ephemeral symmetric key and the decrypted message), actually affects its leakage-resilience as well (i.e., the long-term key). A similar issue could pop up in any instantiations of the proposals from [HHMS25]. Interestingly, this second attack, denoted as a flag DPA in Table 3, is a distinguishing attack. Yet, it

[2] It is an interesting open question to find out whether adaptations of Polka can satisfy the requirements of [HLM+23], to obtain side-channel security for a KEM rather than a CCA-secure encryption scheme.

[3] For leakage-resistance, which aims to protect the confidentiality of the encrypted message, security against guessing DPA is again needed (see Appendix A, Table 4 for an update of Table 2).

Table 3: Polka's updated leakage-resilience requirements.

	CPA decryption	ciphertext validity checking		
		computation		control
targets (ops/var)	poly. multiplications	norm checks (H inputs)	(H outputs) AE	$\perp, \top$
type of attack	guessing DPA	guessing DPA	$\emptyset$	flag DPA
countermeasure	masking	masking	$\emptyset$	?

exploits a bit of information corresponding to a control signal that is scarcely manipulated by Polka's implementation. By contrast, the distinguishing DPA of Table 1 targets a bit of information that corresponds to two plaintexts affecting many intermediate variables in Kyber's implementation. So again, such an attack does not cancel the interest of the approach in [HLM+23, HHMS25], but it makes it more quantitative than qualitative.

Overall, these results further clarify the difficulty of designing PQ KEMs or CCA-secure encryption schemes enabling efficient and leakage-resilient implementations.[4] As reflected by our title, removing the FO-transform seems to remain a useful ingredient for this purpose, but it implies dealing with other (quantitatively smaller) sources of leakages in designs leveraging this tweak. Studying whether alternative schemes, possibly based on other hardness assumptions, can lead to better opportunities of efficient countermeasures, is an interesting scope for further research. Finding out whether more formal analyzes can be obtained for such schemes and lead to more confident security guarantees too.

2 Guessing DPA against a de-randomized Polka

In this section, we describe the blueprint of our new attacks against a de-randomized version of Polka (i.e., removing the dummy ciphertexts of its decryption). This simplification allows us to clarify the attack mechanism by exposing how key-dependent components can persist in Polka's decryption, despite the removal of the FO-transform. We start with a reminder about Polka's specifications, and follow with a baseline attack and a variant.[5]

2.1 De-randomized Polka decryption specifications

The structure of Polka follows the KEM-DEM paradigm followed by many PQ encryption schemes (e.g., Kyber). Yet, the way its decryption rejects ill-formated ciphertexts relies on norm checks applied to re-extracted noise values rather than on the FO-transform.

We first recall its Keygen and Encrypt procedures, which are left unchanged.

Keygen: Let $\{\sum_{i=1}^{2}(u_i - v_i) \bmod 3 \mid u_i, v_i \leftarrow \{0,1\}\}$ a noise distribution, to be used over integer vector coefficients, where mod 3 means the integer representative over $\{-1, 0, 1\}$. The ternary distribution over $\mathcal{R} = \mathbb{Z}_q[X]/(X^n + 1)$ is denoted as $\mathcal{D}^{\mathsf{coeff}}_{n,B}$, where $B = 1$. For any $r \leftarrow \mathcal{D}^{\mathsf{coeff}}_{n,B}$, we have $\|r\| \leq B$ for the infinity norm over the vector of coefficients. Keygen produces $b = p(as + e) \in \mathcal{R}^*$ from $a \leftarrow \mathcal{R}$, $s, e \leftarrow \mathcal{D}^{\mathsf{coeff}}_{n,B}$ and $p \geq 4B + 1 \in \mathbb{Z}$.

Encrypt: Given a public key (n, q, p, a, b) and a message $M \in \{0,1\}^{\ell_m}$:

[4] They directly apply to the leakage-resilient variant of NTRU [CDH+19] discussed in [HLM+23].

[5] We focus on Polka's variant with ternary noise distribution, defined as the difference of two binomials per coefficient. It also applies to variants with larger noise or other (e.g., Gaussian) distributions.

1. Sample $r, e_1, e_2 \leftarrow \mathcal{D}_{n,B}^{\mathsf{coeff}}$ and compute the KEM part:

$$c_1 = a \cdot r + e_1 \in R, \qquad c_2 = b \cdot r + e_2 \in R,$$

   together with $K = \mathsf{H}(r, e_1, e_2) \in \{0,1\}^\kappa$.

2. Compute $c_0 = \mathsf{E}_K(M)$ as the DEM part.

Output the ciphertext $C = (c_0, c_1, c_2)$.

We next specify the de-randomization process that is the target of our attacks, and we denote this modified algorithm as $\mathsf{Decrypt}^*$:

$\mathsf{Decrypt}^*$: Given the secret key $s \in R$ and $C = (c_0, c_1, c_2)$, do the following:

1. Initiate $\mathsf{flag} = 0$.
2. Compute $\mu = c_2 - p \cdot c_1 \cdot s$ over R_q.
3. Compute $e_2 = \mu \bmod p$. If $\|e_2\| > B$, set $\mathsf{flag} = 1$.
4. Compute $r = (c_2 - e_2) \cdot b^{-1} \in R_q$. If $\|r\| > B$, set $\mathsf{flag} = 1$.
5. Compute $e_1 = c_1 - a \cdot r \in R_q$. If $\|e_1\| > B$, set $\mathsf{flag} = 1$.
6. Sample $r^*, e_1^*, e_2^* \hookleftarrow D_{n,B}^{\mathsf{coeff}}$. If $\mathsf{flag} = 0$, compute $K = H(r, e_1, e_2) \in \{0,1\}^\kappa$, else compute $K = H^*(r^*, e_1^*, e_2^*)$. Return:

$$M = \mathsf{D}_K(c_0) \in \{0,1\}^{\ell_m} \cup \{\perp\}.$$

For honestly generated ciphertexts, the intermediate values after reduction modulo p are independent of the secret, because the key-dependent component $e_1 \cdot s$ is eliminated. That is, we have:

$$\begin{aligned} \mu = c_2 - p \cdot c_1 \cdot s &= b \cdot r + e_2 - p \cdot (a \cdot r + e_1) \cdot s, \\ &= p \cdot (a \cdot s + e) \cdot r + e_2 - p \cdot (a \cdot r + e_1) \cdot s, \\ &= p \cdot (e \cdot r + e_1 \cdot s) + e_2, \end{aligned}$$

so that $\mu \mod p$ gives back the random (key-independent) noise e_2. Therefore, after the computation of Step 2, which is an unavoidable guessing DPA target (but is easy-to-mask, since key-homomorphic), it was suggested in [HLM+23] that Steps 3, 4 and 5 could only be SPA targets, since manipulating ephemeral secrets.

We next show that this expectation is too optimistic.

2.2 Baseline attack description

The main idea of our attack is to choose noise polynomials (r, e_1, e_2) so that key-dependent terms survive the modular reduction and remain in the decryption process. From the chosen noise, the adversary constructs a ciphertext $C = (c_0, c_1, c_2)$ with $c_1 = a \cdot r + e_1$, $c_2 = b \cdot r + e_2$ (using public parameters) and an arbitrary c_0. By querying the $\mathsf{Decrypt}^*$ oracle on such ciphertexts, she obtains leakages corresponding to key-dependent decryption computations (and, in this simplified version, deterministic), enabling a guessing DPA.

Concretely, the effect of the modulo p reduction is neutralized by selecting e_1 to include a factor of $-p^{-1}$, so that the product $e_1 \cdot s$ remains visible after reduction. Given this class of admissible noise choices is still large, we further restrict our attention to ciphertexts that (i) appear plausibly valid (e.g., cannot be trivially filtered by sparsity tests on the

ciphertexts, like in side-channel assisted CCA attacks against Kyber [RRCB20]); and (ii) produce a simple, easily interpretable dependence on the secret, facilitating side-channel key-recovery attacks. A bit more precisely, the baseline adversary chooses noise as:

$$\begin{aligned} & r \hookleftarrow D_{n,B}^{\mathsf{coeff}}, \\ & e_1'' \hookleftarrow D_{n,B}^{\mathsf{coeff}},\ e_1 = e_1'' - p^{-1}, \\ & e_2 = 0, \end{aligned} \tag{1}$$

and sets $C = (c_0, c_1 = a \cdot r + e_1, c_2 = b \cdot r)$ with arbitrary c_0.

Evaluating $\mathsf{Decrypt}^*$ on this ciphertext therefore implies:

2. Computing $\mu = c_2 - p \cdot c_1 \cdot s$ over R. More specifically:

$$\begin{aligned} c_2 - p \cdot c_1 \cdot s &= b \cdot r - p \cdot (a \cdot r + e_1) \cdot s, \\ &= p \cdot e \cdot r - p \cdot (e_1'' - p^{-1}) \cdot s, \\ &= p \cdot (e \cdot r - e_1'' \cdot s) + s \quad (\|.\|_\infty < q/2), \end{aligned}$$

   where the right-hand side of the last equality holds over $\mathbb{Z}^n$.

3. Computing $\bar{e}_2 = \mu \pmod p = s$ and checking that $\|\bar{e}_2 = s\| \leq B$ ($\mathsf{flag} = 0$).

4. Computing $\bar{r} = (c_2 - \bar{e}_2) \cdot b^{-1} = (c_2 - s) \cdot b^{-1} = r - s \cdot b^{-1}$ and checking that $\left\|\bar{r} = r - s \cdot b^{-1}\right\| > B$ ($\mathsf{flag} = 1$).

5. Computing $\bar{e}_1 = c_1 - a \cdot \bar{r}$, which is worth:

$$\begin{aligned} c_1 - a \cdot \bar{r} &= a \cdot r + e_1 - a \cdot (r - s \cdot b^{-1}), \\ &= e_1 + a \cdot s \cdot b^{-1}, \end{aligned}$$

   so that $\|\bar{e}_1\| = \left\|e_1 + a \cdot s \cdot b^{-1}\right\| > B$ ($\mathsf{flag} = 1$).

6. Sample $r^*, e_1^*, e_2^* \hookleftarrow D_{n,B}^{\mathsf{coeff}}$, compute $K = H^*(r^*, e_1^*, e_2^*)$ and return:

$$M = \mathsf{D}_K(c_0) \in \{0,1\}^{\ell_m} \cup \{\perp\}.$$

Utilizing this special form of ciphertexts, the adversary can use the decryption leakage in order to recover the key s in several ways, namely:

1. *Guessing SPA attack path.* The `norm_check` on $\bar{e}_2$ in Step 3 directly exposes s and its leakage can therefore be used in order to mount a guessing SPA.

2. *Guessing DPA attack path.* The `poly_sub` related to $\bar{r}$ of Step 4 operates on $c_2 - s$, where the adversary controls c_2. The resulting leakage can be used for a guessing DPA, possibly amplified by leveraging the leakage of the `poly_mult` operation.

3. *Advanced DPA attack path.* The `norm_check` on $\bar{r}$ in Step 4 also takes $r - s \cdot b^{-1}$ as input, where the adversary controls r and b^{-1} is public. Hence its leakage can yield a linear equation in s, which can be continuously accumulated. We denote it as an advanced DPA because exploiting this leakage is less direct and requires recovering full coefficients of $s \cdot b^{-1}$ before solving for s. (By contrast, the previous Guessing DPA is a divide-and-conquer one where coefficients of s are recovered one by one).

A similar situation holds for Step 5, where the computations lead to a guessing DPA attack path and the norm checks additionally lead to an advanced DPA attack path.

2.3 Alternative attack description

Rather than fixing e_2 at zero, the adversary may choose e_2 as a probe to gain finer control over the leakage of Polka's decryption. That means choosing:

$$\begin{aligned} &r \hookleftarrow D_{n,B}^{\mathsf{coeff}}, \\ &e_1'' \hookleftarrow D_{n,B}^{\mathsf{coeff}},\ e_1 = e_1'' - p^{-1}, \\ &e_2 = \delta_j x^i \quad (i^{th} \text{ coefficient is } \delta_j \in \mathbb{Z}_p, \text{ and 0 elsewhere}). \end{aligned} \tag{2}$$

As a result, evaluating $\mathsf{Decrypt}^*$ on this ciphertext this times implies:

2. Computing $\mu = c_2 - p \cdot c_1 \cdot s = p \cdot (e \cdot r - e_1'' \cdot s) + e_2 + s \quad (\|.\|_\infty < q/2)$.
3. Computing $\bar{e}_2 = \mu \pmod p = e_2 + s$ and checking $\|\bar{e}_2 = e_2 + s\|$ such that if $|\delta_j + s_i| \leq B$ then $\mathsf{flag} = 0$, otherwise $\mathsf{flag} = 1$.
4. Computing $\bar{r} = (c_2 - \bar{e}_2) \cdot b^{-1} = (c_2 - e_2 - s) \cdot b^{-1} = r - s \cdot b^{-1}$ and checking that $\left\|\bar{r} = r - s \cdot b^{-1}\right\| > B$ ($\mathsf{flag} = 1$).
5. Computing $\bar{e}_1 = c_1 - a \cdot \bar{r}$, which is worth $e_1 + a \cdot s \cdot b^{-1}$ and checking that $\|\bar{e}_1\| > B$ ($\mathsf{flag} = 1$).
6. Sample $r^*, e_1^*, e_2^* \hookleftarrow D_{n,B}^{\mathsf{coeff}}$, compute $K = H^*(r^*, e_1^*, e_2^*)$ and return:

$$M = \mathsf{D}_K(c_0) \in \{0,1\}^{\ell_m} \cup \{\perp\}.$$

Thus, by carefully selecting small-norm e_2 polynomials, the adversary can elicit coefficient-wise information about s. Thanks to this information, she can turn the guessing SPA against the norm check of Step 3 into another guessing DPA. As mentioned in introduction, the adversary can also use the leakage on the flag (control) signal in order to deduce information about s, which will be specifically discussed in Section 5. Meanwhile, we show how the previous attacks against de-randomized Polka can be generalized to the actual (randomized) Polka, before demonstrating the concrete feasibility of both the baseline guessing DPA attack exploiting the computation leakage of Step 4, and the alternative guessing DPA attack exploiting the norm check leakage of Step 3 in Section 4.

3 Guessing DPA against (randomized) Polka

The decryption of Polka adds dummy ciphertexts (derived from fresh random coins) to the ciphertexts before any computation. They prevent the adversary to observe the manipulation of the same ephemeral secrets several times, and therefore the averaging of their leakage. Hence, they are expected to help for leakage-resilient implementations. We next show that these dummies only have a mild impact on the previous attacks.

We first recall the $\mathsf{Decrypt}$ computation of Polka that is the target of our attacks:

3.1 Polka specifications

The Keygen and $\mathsf{Encrypt}$ are identical to the de-randomized version. The $\mathsf{Decrypt}$ procedure, given the secret key s and ciphertext $C = (c_0, c_1, c_2)$ proceeds as follows:

1. Initiate $\mathsf{flag} = 0$.
2. Sample $r', e_1', e_2' \hookleftarrow D_{n,B}^{\mathsf{coeff}}$ and return $\perp$ if $\|r'\| > B$ or $\|e_1'\| > B$ or $\|e_2'\| > B$. Otherwise, compute $c_1' = a \cdot r' + e_1'$ and $c_2' = b \cdot r' + e_2'$.

3. Compute $\bar{c}_1 = c_1 + c'_1$ and $\bar{c}_2 = c_2 + c'_2$.

4. Compute $\bar{\mu} = \bar{c}_2 - p \cdot \bar{c}_1 \cdot s$ over R_q.

5. Compute $\bar{e}_2 = \mu \pmod p$, if $\|\bar{e}_2\| > 2B$, set $\mathsf{flag} = 1$.

6. Compute $\bar{r} = (\bar{c}_2 - \bar{e}_2) \cdot b^{-1}$, if $\|\bar{e}_2\| > 2B$, set $\mathsf{flag} = 1$.

7. Compute $\bar{e}_1 = \bar{c}_1 - a \cdot \bar{r}$, if $\|\bar{e}_2\| > 2B$, set $\mathsf{flag} = 1$.

8. Compute a. $e_2 = \bar{e}_2 - e'_2$, if $\|e_2\| > B$, set $\mathsf{flag} = 1$,
   b. $r = \bar{r} - r'$, if $\|r\| > B$, set $\mathsf{flag} = 1$,
   c. $e_1 = \bar{e}_1 - e'_1$, if $\|e_1\| > B$, set $\mathsf{flag} = 1$.

9. Sample $r^*, e_1^*, e_2^* \hookleftarrow D_{n,B}^{\mathsf{coeff}}$. If $\mathsf{flag} = 0$, compute $K = H(r, e_1, e_2) \in \{0,1\}^\kappa$, else compute $K = H^*(r^*, e_1^*, e_2^*)$. Return:

$$M = \mathsf{D}_K(c_0) \in \{0,1\}^{\ell_m} \cup \{\perp\}.$$

Compared to Section 2.1, the dummy ciphertexts replace the original, DPA-vulnerable product (Step 4) by the product between the long-term secret s and an ephemeral unknown factor $p \cdot \bar{c}_1$. In the honest case, the relation in Step 4 therefore becomes:

$$\begin{aligned}\bar{\mu} = \bar{c}_2 - p \cdot \bar{c}_1 \cdot s &= b \cdot (r + r') + (e_2 + e'_2) - p \cdot (a \cdot (r + r') + e_1 + e'_1) \cdot s, \\ &= p \cdot (e \cdot (r + r') - s \cdot (e_1 + e'_1)) + e_2 + e'_2 \quad (\|.\|_\infty < q/2).\end{aligned}$$

The adversary does not know the dummy values but their distribution is public. Hence, by controlling the chosen-ciphertext noise (r, e_1, e_2) and observing leakage mixing these values with the dummy noise (r', e'_1, e'_2), she can mount baseline and alternative attacks.

3.2 Baseline attack description

The adversary uses the same chosen ciphertexts as in the baseline attack of Section 2.2 and follows the rules of Equation 1. That means to compute $c_1 = a \cdot r + e_1$, $c_2 = b \cdot r + e_2$ with $e_2 = 0$ and $e_1 = e''_1 + p^{-1}$, and to pick arbitrary c_0. Decryption on the masked values $(\bar{c}_1, \bar{c}_2)$ proceeds on key-dependent values as follows since $\bar{\mu} = p \cdot (e \cdot (r + r') - (e''_1 + e'_1)) + e'_2 + s$:

5. Compute $\bar{e}_2 = \mu \pmod p = e'_2 + s$, check that $\|\bar{e}_2 = e'_2 + s\| \leq 2B$, $\mathsf{flag} = 0$.

6. Compute $\bar{r} = (\bar{c}_2 - \bar{e}_2) \cdot b^{-1} = (b \cdot (r + r') - s) \cdot b^{-1} = r + r' - s \cdot b^{-1}$,
   check that $\left\|\bar{r} = r + r' - s \cdot b^{-1}\right\| > 2B$, $\mathsf{flag} = 1$.

7. Compute $\bar{e}_1 = \bar{c}_1 - a \cdot \bar{r} = e_1 + e'_1 + a \cdot s \cdot b^{-1}$,
   check that $\left\|\bar{e}_1 = e_1 + e'_1 + a \cdot s \cdot b^{-1}\right\| > 2B$, $\mathsf{flag} = 1$.

8. Compute a. $\bar{e}_2 - e'_2 = s$, check that $\|s\| < B$,
   b. $\bar{r} - r' = r - s \cdot b^{-1}$, check that $\left\|r - s \cdot b^{-1}\right\| > B$,
   c. $\bar{e}_1 - e'_1 = e_1 + a \cdot s \cdot b^{-1}$,
   Check that $\left\|e_1 + a \cdot s \cdot b^{-1}\right\| = \left\|e_1 + a \cdot s \cdot b^{-1}\right\| > B$, flag remains 1.

9. Sample $r^*, e_1^*, e_2^* \hookleftarrow D_{n,B}^{\mathsf{coeff}}$, compute $K = H^*(r^*, e_1^*, e_2^*)$, and return:

$$M = \mathsf{D}_K(c_0) \in \{0,1\}^{\ell_m} \cup \{\perp\}.$$

As a result, the exploitable leakage (compared to Section 2.2) evolves as follows:

1. *From guessing DPA to unknown-input guessing DPA.* As mentioned above, the original DPA-vulnerable product of Step 4 is updated with an unknown factor. This transforms the guessing DPA of de-randomized Polka into an Unknown-Input (UI) guessing DPA. This operation is easy to mask (key-homomorphic).

2. *From guessing SPA to guessing UI-DPA* (`norm_check` on $\bar{e}_2$, now in Step 5). The dummy polynomial e'_2 replaces the deterministic input e_2 by $\bar{e}_2 = e'_2 + s$. Consequently, the guessing SPA that formerly targeted a fixed function of s becomes an UI-DPA with an unknown additive input e'_2. Note that if the operation were performed in a cyclic domain with no range extension (e.g., like a bitwise XOR), $\bar{e}_2$ would be statistically indistinguishable across different s. However, `norm_check` maps ring elements into centered integer ranges, so different coefficient values of s shift $\bar{e}_2$ into distinct integer intervals whose distributions are determined by the known sampling rule of e'_2. By repeated queries and leveraging the public distribution of e'_2, an adversary can distinguish these intervals and infer the value of s coefficient-wise.

3. *Obsolete guessing DPA paths* (on $\bar{r}$, now in Step 6). The large and random component c'_2 tends to mask key-dependent intermediates in full ring-wide transforms (like `poly_sub`, `poly_mult`), and effectively cover the distribution of s, making these operations less directly useful for DPA than when targeting de-randomized Polka.

4. *Possible Advanced UI-DPA paths* (`norm_check` on $\bar{r}$ and $\bar{e}_1$, now in Steps 6-7). Although r' and e'_1 are random, they follow constrained (non-uniform) distributions. Norm checks applied to $\bar{r} = r + r' - s \cdot b^{-1}$ and $\bar{e}_1 = e_1 + e'_1 + a \cdot s \cdot b^{-1}$ therefore remain statistically dependent on s. These operations leak information about $s \cdot b^{-1}$ and $a \cdot s \cdot b^{-1}$. Recovering s from them requires coefficient-wise reconstruction of the leaked polynomials and solving a system of linear equations, as in Section 2.2.

5. *SPA and DPA paths after de-randomization.* If the implementation does not abort early when a flag flips (i.e., continues computing and removes randomness at the end), key-dependent components propagate until the de-randomization step, restoring a guessing SPA from `norm_check` in Step 8.a and an advanced DPA in Step 8.b-c.

3.3 Alternative attack description

As in the case of de-randomized Polka, the adversary can also control e_2 in order to gain more information about s, by using the same chosen random noise (r, e_1, e_2) as in Equation 2 with $e_1 = e''_1 + p^{-1}$ and $e_2 = \delta_j x^i$. The intermediate result in Step 4 becomes:

$$\begin{aligned}\bar{\mu} = \bar{c}_2 - p \cdot \bar{c}_1 \cdot s &= b \cdot (r + r') + (e_2 + e'_2) - p \cdot (a \cdot (r + r') + e_1 + e'_1) \cdot s \\ &= p \cdot (e \cdot (r + r') - s \cdot (e'_1 + e''_1)) + e_2 + e'_2 + s \quad (\|.\|_\infty < q/2),\end{aligned}$$

and the rest of the decryption proceeds on key-dependent values as follows:

5. Compute $\bar{e}_2 = \mu \pmod p = e_2 + e'_2 + s$, check that $\|\bar{e}_2\|$,
if $|\delta_j + s_i + (e'_2)_i| \leq 2B$ then $\mathsf{flag} = 0$, $\mathsf{flag} = 1$ otherwise.

6. Compute $\bar{r} = (\bar{c}_2 - \bar{e}_2) \cdot b^{-1} = (b \cdot (r + r') - s) \cdot b^{-1} = r + r' - s \cdot b^{-1}$,
check that $\left\|\bar{r} = r + r' - s \cdot b^{-1}\right\| > 2B$, $\mathsf{flag} = 1$.

7. Compute $\bar{e}_1 = \bar{c}_1 - a \cdot \bar{r} = e_1 + e'_1 + a \cdot s \cdot b^{-1}$,
check that $\left\|\bar{e}_1 = e_1 + e'_1 + a \cdot s \cdot b^{-1}\right\| > 2B$, $\mathsf{flag} = 1$.

8. Compute a. $\bar{e}_2 - e'_2 = e_2 + s$, check that $\|e_2 + s\| < B$,
   b. $\bar{r} - r' = r - s \cdot b^{-1}$, check that $\left\|r - s \cdot b^{-1}\right\| > B$,
   c. $\bar{e_1} - e'_1 = e_1 + a \cdot s \cdot b^{-1}$,
   Check that $\left\|e_1 + a \cdot s \cdot b^{-1}\right\| = \left\|e_1 + a \cdot s \cdot b^{-1}\right\| > B$, flag remains 1.
9. Sample $r^*, e_1^*, e_2^* \hookleftarrow D_{n,B}^{\mathsf{coeff}}$, compute $K = H^*(r^*, e_1^*, e_2^*)$, and return:

$$M = \mathsf{D}_K(c_0) \in \{0,1\}^{\ell_m} \cup \{\perp\}.$$

As a result, the norm check on $\bar{e}_2$ (in Step 5) remains an UI-DPA target. Steps 6-7 continue to leak linear combinations involving s that allow advanced UI-DPA paths. Step 8.a (after removing e'_2) offers a (standard) guessing DPA on $e_2 + s$ and Steps 8.b-c remain vulnerable to advanced DPA. In Section 4, we will demonstrate the concrete feasibility of the UI-DPA against the norm check of Step 5 and the guessing DPA against Step 8.a.

4 Experimental validation

We now question the practicality of the findings in the previous section, and demonstrate that they are applicable both in simulations and using actual measurements. As already mentioned, we focus on three attacks and target operations: (1) the baseline DPA exploiting Step 4 of de-randomized Polka in Section 2.2; (2) the baseline DPA exploiting Step 5 of (randomized) Polka in Section 3.2; (3) the alternative DPA exploiting Step 3 of de-randomized Polka in Section 2.3 or Step 8 of (randomized) Polka in Section 3.3. Note that in the latter case the randomization does not affect the the alternative DPA, since we apply it after the de-randomization in Step 8. We start the section with preliminary observations, follow with simulated experiments and conclude with actual (software) measurements.

4.1 Preliminary observations

Besides the characteristics of the leakage function, there are a few additional properties that influence the complexity of side-channel attacks against PQ encryption schemes. We next detail three important ones that are relevant to our investigations:

Coefficient-wise independence. Target operations such as `norm_check` and `poly_sub`, when implemented serially, act independently on each coefficient. Thus, leakage from these operations can be exploited in full, to recover all the coefficients simultaneously, and a single query to the Decrypt oracle provides an equal amount of leakage for all coefficients. An adversary can therefore craft chosen ciphertexts with non-zero e_2 coefficients everywhere in their attacks. In our experiments targeting such operations, we nevertheless restrict our analysis to only one coefficient for simplicity.

Input range. Target operations operate on variables within different ranges. In Polka, the inputs come with two main ranges. For the implementation we focus on, we first have full-range polynomials with coefficients uniformly sampled from $\mathbb{Z}_q$, where $q = 59393$. Secondly, we have small polynomials with coefficients sampled from the Central Binomial Distribution (CBD), with range $\mathcal{S} = \{0, 1, -1\}$ and associated probabilities $P_{\mathcal{S}} = \{0.5, 0.25, 0.25\}$. According to the choice of crafted ciphertexts, `norm_check` can operate on different input ranges in different cases. Iin Step 3 of de-randomized Polka (and Step 5 in Polka), it checks small coefficients sampled from the CBD (or the sum of them in the alternative attack). In the next steps, it operates on uniformly distributed polynomials. This impacts the attacks' complexity, since larger (more "entropic") secrets are generally harder to recover.

Input representation. Finally, target operations can also act on different representations within the same range. For example, `poly_sub` operates on ring polynomials in standard representation (i.e., its coefficients are in $\mathbb{Z}_q = [0, \ldots, q-1]$), whereas `norm_check` needs a conversion to the centered representation ($[-\frac{q-1}{2}, \frac{q-1}{2}]$). Works such as [TMS24, PS24, NHPM25] showed that such changes of representation affect the informativeness of the leakages, hence the complexity of side-channel attacks.

4.2 Simulated evaluations

In order to obtain a first assessment of the attacks complexity, we simulate the leakage of intermediate computations of Polka according to the quite standard Hamming weight leakage model with additive Gaussian noise [MOP07]. Hence, the leakage for a variable X is modeled as:

$$\mathsf{L}(X) = \mathrm{HW}(\mathsf{repr}(X)) + N, \tag{3}$$

where repr denotes the representation used in the operation, and N is an independent Gaussian noise, with probability density function $\mathcal{N}(0, \sigma)$. The deterministic and data-dependent term is usually denoted as the signal. Besides, and as just mentioned, the range and representation of the operands in an implementation affect the variance of their signal and, therefore, the complexity of the side-channel attacks targeting them. For example, the signal corresponding to `norm_check` on the same small input range, represented in a 32-bit word, is worth $\{0 = \mathrm{HW}(\mathsf{repr}(0)), 1 = \mathrm{HW}(\mathsf{repr}(1)), 32 = \mathrm{HW}(\mathsf{repr}(q-1))\}$ with a centralized representation, leading to a signal variance of 324 3125; and $\{0 = \mathrm{HW}(\mathsf{repr}(0)), 1 = \mathrm{HW}(\mathsf{repr}(1)), 14 = \mathrm{HW}(\mathsf{repr}(q-1))\}$ with a standard representation, leading to a signal variance of 5.8125. For simplicity, we evaluate the leakage using only the centralized representation for the norm checks. The standard representation does not lead to significantly different conclusions on the practical feasibility of the attacks.

As usual with simulations, we use the Signal to Noise Ration (SNR) as a normalized indicator of the level of measurement noise [Man04], computed as:

$$\mathrm{SNR} = \frac{\mathrm{Var}(signal)}{\mathrm{Var}(noise)} = \frac{\mathrm{Var}\Big(\mathrm{HW}\big(\mathsf{repr}(\mathrm{X})\big)\Big)}{\mathrm{Var}(\mathrm{N})}.$$

The range and representation of the inputs determine the signal variance and the noise standard deviation σ is then adjusted to achieve a fixed SNR across scenarios.

Given the model in Equation 3, the adversary can learn the likelihood of the leakage given the value of intermediate variables $X = x$, namely $f(l|x) = f(l|\mathrm{HW}(\mathsf{repr}(x))$. She can then guess the key using a Maximum a posteriori (MAP) strategy for the three attack paths mentioned above. The baseline DPA exploiting Step 4 of de-randomized Polka in Section 2.2 and the alternative DPA exploiting Step 3 of de-randomized Polka in Section 2.3 are (known-input) guessing DPAs. The baseline DPA exploiting Step 5 of (randomized) Polka in Section 3.2 is an Unknown-Input (UI) DPA. The proceed as follows:

1. In a (known-input) guessing DPA, the observable variable is $X = V + S$ for some value V that would be C_2 in the baseline DPA against de-randomized Polka and E_2 in the alternative DPA targeting Polka's norm checks. The adversary uses the leakage to infer $f(l|s)$ from $f(l|\mathrm{HW}(\mathsf{repr}(s+v))$ with known v. She then applies MAP and computes a probability $p(s|l) = \frac{f(l|s)\cdot p(s)}{\sum_{s'} f(l|s')\cdot p(s')}$ for each key value.

2. In the UI guessing DPA, the observable variable is $X = E_2' + E_2 + S$, where E_2' is unknown to the adversary. Instead of directly observing the Gaussian distribution corresponding to the leakage of s, the adversary observes a mixture of distributions

induced by the unknown e'_2. Concretely, the posterior over s given observed leakage l therefore becomes:

$$p(s|l) = \sum_{e'_2} p(s, e'_2|l) = \sum_{e'_2, \bar{e}_2 = s + e'_2} p(\bar{e}_2|l),$$
$$= \sum_{e'_2, \bar{e}_2 = s + e'_2} \frac{f(l|\bar{e}_2)p(\bar{e}_2)}{f(l)}.$$

The adversary then applies MAP, but without knowing the value of e'_2.

Our simulations results are in Figure 1, which shows the number of attack traces (N_a) required to recover the secret with $> 99\%$ accuracy in function of the SNR. The dotted line is for the known-input baseline DPA against de-randomized Polka. The plain line is for the unknown-input baseline DPA against (randomized) Polka. The dashed line is for the known-input alternative DPA that applies to both versions of Polka.

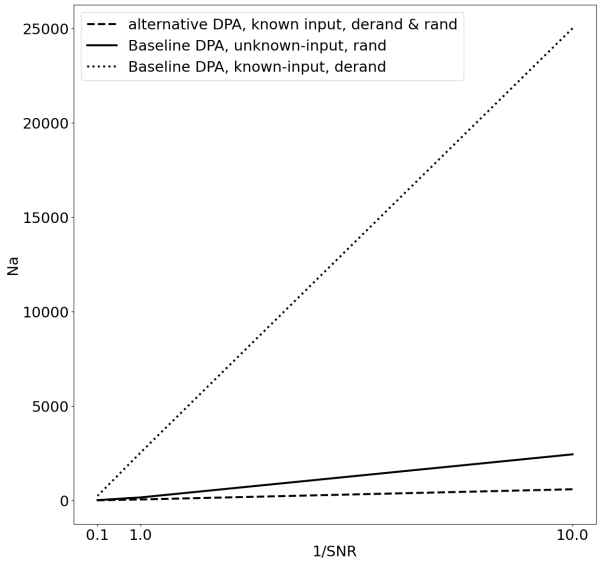

Figure 1: Number of traces to recover secret key with $> 99\%$ accuracy.

As expected, the complexity of a DPA grows inversely proportional to the SNR. The most significant gap is observed between the baseline DPA of Section 2.2 and the other attacks. This disparity arises from the difference in input ranges (and therefore in signal variance) that are caused by operations being performed on small-range values with centered representation or full-range uniformly distributed values. Interestingly, the cost of the unknown-input baseline DPA is only slightly higher than the cost of the known-input alternative attack targeting the norm checks of de-randomized and randomized Polka. As already explained, this is due to the fact that `norm_check` does not operate cyclically and the range of the unknown e'_2 is not large enough to effectively hide the secret.

We note that these simulated experiments are univariate and only exploit the leakage of the inputs of the target operations listed above. As clear from our results, and will be confirmed from actual measurements next, this already leads to damaging enough attacks to motivate the addition of countermeasures. It is however possible to further improve these attacks by better exploiting the multivariate leakage that computations on these values lead to. The Number Theoretic Tranforms (NTTs) that are involved in polynomial multiplications are a typical example [PPM17, PP19]. Given that such optimizations do

not impact our discussion on the leakage-resilience features of PQ encryption schemes getting rid of the FO-transform, we leave them as an interesting scope for further research, together with the concrete investigation of the advanced DPA attack paths.

4.3 Concrete experiments

The previous simulation results indicate that the baseline DPA exploiting Step 5 of (randomized) Polka in Section 3.2 and the alternative DPA exploiting the norm checks in Step 3 of de-randomized Polka in Section 2.3 or Step 8 of (randomized) Polka in Section 3.3 are interesting weak points. Accordingly, we confirm these attack paths based on actual measurements in this section. We use the reference Polka implementation for Cortex M4 devices from [SHM+]. The measurements are acquired with a CW308 UFO board, embedding an STM32F415-RGT6 microcontroller as target. The board clock is stabilized at 8 MHz using an external crystal oscillator. Leakage is captured with a CT1 current probe placed on a jumper between the on-board power regulator and the microcontroller. Traces are sampled with a PicoScope 5244B at 62.5 MSamples/s, with 12-bit resolution. For simplicity, we use a specific trigger to signify the start of the target operations on chip, which avoids the cost of trace alignment. In the profiling phase, we assume the side-channel adversary controls all sources of randomness (including the dummy ciphertexts). As a result, the SNR can be easily estimated for all the target operations (where, as usual, the signal is computed as the average of the mean leakages corresponding to the target intermediate variables). We then build multivariate Gaussian templates [CRR02], from which the same probabilities as in the previous section can be directly inferred.

Figure 2 reports the success rates in function of the number of attack traces, averaged over 1,000 experiments. Both attacks succeed with probability $> 99\%$ once the number of traces exceeds 40. The dummies only makes the UI-DPA slightly more expensive than its known-input counterpart, which is consistent with the simulation results. The security gain of adding such dummies in the design is therefore limited, even more as long as that last norm checks in (randomized) Polka operate after the dummies' removal.

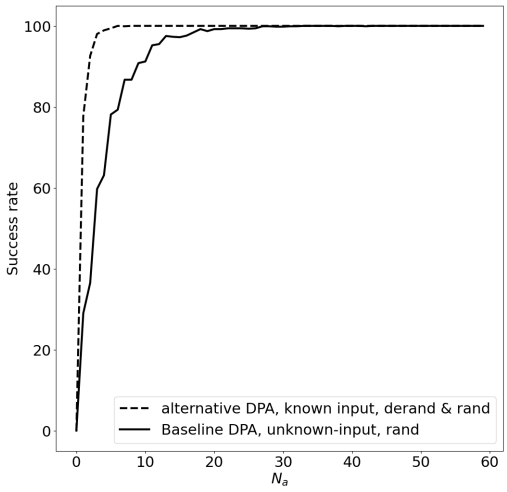

Figure 2: Success rate as a function of attack traces on `norm_check`.

5 Flag DPA against Polka

The attacks detailed in the previous sections exploit the leakage during the computations of intermediate values that involve the secret key during decryption. In this section, we describe new attacks that rather exploit the moment when a ciphertext is deemed invalid in the implementation. This invalidity information is stored in the flag (control) signal, as an updatable Boolean variable.[6] Admittedly, the fact that such a flag signal can leak critical information is already suggested in [BDH+19], where the authors formalize the "noise learning problem" which must be solved to exploit such leakages, and identify optimal bounds in terms of the number of oracle calls for this purpose. So the goal of this section is to use and slightly generalize this framework in order to analyze Polka (and related schemes), and to determine the extent to which the flag leakages on de-randomized and randomized noises that take place its (their) implementation is damaging.

5.1 Attack description against de-randomized Polka

The following attack shows the sensitivity of the value assigned to the flag variable during Polka's decryption. We show that, assuming the adversary can distinguish which step of the implementation recognizes a ciphertext as invalid, she can easily extract all the coefficients of the secret $s = \sum_{k=0}^{n-1} s_k x^k$. Again, we first analyze the de-randomized algorithm $\mathsf{Decrypt}^*$, given in Section 2.3. The adversary queries the decryption with the chosen ciphertext given in the alternative attack path which follow the rules of Equations 2. Unlike the attacks that exploit the leakage of the computation, the adversary in this case must be careful with the position of non-zero coefficient of e_2. That is, she proceeds on the ciphertext where only one coefficient is non-zero, one at a time.

Let (i, j) be the query where e_2 in Equations 2 equals to $e_{2,i,j} = jx^i$ (only i-th coefficient is non-zero and the value of the non-zero coefficient is j). The adversary queries decryption of multiple ciphertexts with KEM parts as:

$$c_{1,i,j} = a \cdot r_{i,j} + (e_{1,i,j} - p^{-1}),$$
$$c_{2,i,j} = b \cdot r_{i,j} + jx^i,$$

where $r_{i,j}$ and $e_{1,i,j}$ are sampled honestly. Similarly as in the alternative attack against de-randomized Polka, in Step 2 of Section 2.3, the result becomes:

$$\mu_{i,j} = c_{2,i,j} - p \cdot c_{1,i,j} \cdot s = p \cdot (e \cdot r_{i,j} - e_{1,i,j} \cdot s) + (jx^i + s);$$

which proceeds in later steps (with general value of B) as:

3. Computing $\bar{e}_{2,i,j} = [\mu_{i,j} \bmod p] = jx^i + s$ and checking that $\|jx^i + s\| \leq B$. If not, then set $\mathsf{flag} = 1$, which means $j + s_i \notin \{-B, \ldots, 0, \ldots, B\}$.

4. Computing $\bar{r}_{i,j} = (c_{2,i,j} - \bar{e}_{2,i,j}) \cdot b^{-1}$ and checking that $\|\bar{r}_{i,j}\| \leq B$. If not, then set $\mathsf{flag} = 1$.

5. Computing $\bar{e}_{1,i,j} = c_{1,i,j} - a \cdot \bar{r}_{i,j}$ and checking that $\|\bar{e}_{1,i,j}\| \leq B$. If not, then set $\mathsf{flag} = 1$.

Assuming that the leakage of flag allows the adversary to figure out when it flips to one at Step 3, she can make a successful key-recovery attack by querying $\mathsf{Decrypt}^*$ with a KEM part $(c_{1,i,j}, c_{2,i,j})$, for $i = 0$ to $n - 1$, and $j = 1$ to $2B + 1$.

[6] We next consider this flag leakage as an oracle. Natural options are to instantiate it are to use a direct leakage on the flag signal or to target the norm checks possibly leading the flag to flip.

More precisely, if we let $F^*_{i,j}$ be the value of flag at the end of Step 3 for a query (i,j), the attack proceeds as follows. Set $s'_k = B+1$, for $k=0$ to $n-1$. For $i=0$ up to $n-1$, set $j \leftarrow 0$, while $F^*_{i,j} = 0$, increment $j \leftarrow j+1$, query decryption with KEM part $(c_{1,i,j}, c_{2,i,j})$, and set $s'_i \leftarrow s'_i - 1$. Output $s = \sum_{k=1}^{n-1} s'_i$. Obviously, when the while loop terminates, say on index pair (i,j_i), j_i is the minimal $\delta > 0$ such that $\delta + s_i \notin \{-B, \ldots, 0, \ldots, B\}$ at Step 3, i.e., $s_i = B+1-\delta$ (and $j_i \leq 2B+1$), and $s'_i = B+1-j_i$. Hence, $s'_i = s_i$. Since the adversary knows $F^*_{i,j}$ after the (i,j) query, she can mount this attack. The total number of decryption queries is thus bounded by $n(2B+1)$. Eventually, since $F^*_{i,j} = 1$ implies $F^*_{i,j+1} = 1$, one can look for j as in a dictionary $[1, 2B+1]$ to reduce this bound down to around $n(1+\log 2B+1)$ queries. Practically, the adversary needs to guess the value of the flag from leakage. She can use multiple traces (from multiple queries of the same type) to reduce the error of a distinguishing attack until she possesses $F^*_{i,j}$ perfectly (e.g., MAP on m leakage traces) and then, deploys the above strategy. Or, she can accept knowing the flag with some probability of error from each decryption query, that is, obtaining $\tilde{F}^*_{i,j}$ where $\Pr[\tilde{F}^*_{i,j} == F^*_{i,j}] < 1$. The adversary can adapt the algorithm so that each query input (i,j) is made m times before j and s'_i are incremented to compensate this inherent noise. In either case, the presence of the noise requires some repetitions of the same query's input in order to correctly guess the value of the secret from the flag's information.

5.2 Removing the flag on de-randomized noise

The actual decryption of Polka in Section 3.1 still requires the computation of the norm checks after removing the additional noises introduced by the dummy ciphertexts. That is, even if these additional noises reduce the leakage of the first Decrypt steps (which we will investigate in the next section), the norm checks on what should be (r, e_1, e_2), as we just focused above against Decrypt*, are also made in Decrypt (Item 8). Hence, and in order to motivate this investigation of flag leakages corresponding to dummied computations, we first show that by slightly adapting the parameters of Polka, one can avoid the computation of these norm checks on (r, e_1, e_2) in Decrypt without affecting the black-box security.

Indeed, in Polka the parameter p must be so that $p \geq 4B+1$. This is due to the fact that the decryption must be correct for noises $\bar{r} = r + r'$, $\bar{e}_1 = e_1 + e'_1$ and $\bar{e}_2 = e_2 + e'_2$, where (r, e_1, e_2) comes from Encrypt and (r', e'_1, e'_2) from the dummy ciphertext in Decrypt. Since all these individual noises have their norm bounded by B, it means that $\bar{r}$, $\bar{e}_1$ and $\bar{e}_2$ that must be recovered have their norm bounded by $2B$. Still, Polka requires to also checks the norms of r, e_1 and e_2 afterwards (Item 8) for security reasons related to the rigidity property. Nevertheless, we observe that choosing $p \geq 6B+1$ would allow removing this last norm check on (r, e_1, e_2), essentially because a $2B$-bound on $(\bar{r}, \bar{e}_1, \bar{e}_2)$ implies a $3B$-bound on (r, e_1, e_2), since (r', e'_1, e'_2) are chosen faithfully during Decrypt. In other words, Decrypt could implicitly accept slightly larger noises (i.e., with norm up to $3B$) as long as Encrypt generates (r, e_1, e_2) with bounded norm B without affecting the rigidity property. Thanks to this minor modification (which impacts the size of the modulus q a bit), the attack of the previous subsection would not not apply anymore.

5.3 Attacks description against (randomized) Polka

Following Section 3.3 and the attack against de-randomized Polka in the previous section, on the query (i,j), the result in Step 4 of Polka becomes:

$$\bar{\mu}_{i,j} = \bar{c}_{2,i,j} - p \cdot \bar{c}_{1,i,j} \cdot s = p \cdot (e \cdot \bar{r}_{i,j} - \bar{e}_{1,i,j} \cdot s) + (jx^i + s + e'_{2,i,j}).$$

The later computations proceed as follows:

5. Compute $\bar{e}_{2,i,j} = \mu_{i,j} \pmod p = e'_{2,i,j} + jx^i + s$. Checking that $\|\bar{e}_{2,i,j}\| \leq 2B$. If not, then set flag $= 1$, which means $(e'_{2,i,j})_i + j + s_i \notin \{-2B, \ldots, 0, \ldots, 2B\}$;

6. Compute $\bar{r}_{i,j} = (\bar{c}_{2,i,j} - \bar{e}_{2,i,j}) \cdot b^{-1}$. Check that $\|\bar{r}_{i,j}\| \leq 2B$. If not, set $\mathsf{flag} = 1$;

7. Compute $\bar{e}_{1,i,j} = \bar{c}_{1,i,j} - a \cdot \bar{r}_{i,j}$. Check that $\|\bar{e}_{1,i,j}\| \leq 2B$. If not set $\mathsf{flag} = 1$;

where $r'_{i,j}, e'_{1,i,j}, e'_{2,i,j} \hookleftarrow D^{\mathsf{coeff}}_{n,B}$. Next, we assume again a leakage allows the adversary to learn flag at the end of Step 5. Following [BDH+19], we assume a Bounded-Offset (BO) oracle that asks "is this value in the range" and returns $\neg\,\mathsf{flag}$ in our case. That is, when the adversary makes a (i, j) decryption query, she also gets the output of $\mathsf{BO}_{2B}(e'_{2,i,j} + jx^i)$, for the unknown random noise $e'_{2,i,j}$ selected by the decryption oracle.

The difference with the simpler attack on $\mathsf{Decrypt}^*$ is that here, and unlike [BDH+19], the input of the BO oracle is not fully under the control of the adversary. Now, two (i, j) decryption queries that always have jx^i in common do not necessarily lead to the same values of flag at Step 5, due to the fresh noise $e'_{2,i,j} \hookleftarrow D^{\mathsf{coeff}}_{n,B}$ used in each call to $\mathsf{Decrypt}$. That is, the value taken by $F_{i,j}$ as the flag at the end of Step 5 for a (i, j) decryption query follows a binomial distribution (and is no more fixed like $F^*_{i,j}$, when s is fixed). Nonetheless, we observe that when $\mathsf{flag} = 1$, the decryption is always right to deem the input ciphertext invalid. It can just happen that $\mathsf{flag} = 0$ when $j + s_i \notin \{-B, \ldots, 0, \ldots, B\}$.

To compensate the additional distribution from $e'_{2,i,j}$, the adversary has to make multiple queries of the same input to ensure that $j + s_i \in \{-B, \ldots, 0, \ldots, B\}$ before incrementing j, and starting from $j = 1$. Her goal is still to identify the first time, i.e., the smallest j, when it does not occur. As before, the strategy is repeated for each index i representing the targeted i-th coefficient of the secret key. Since a single $\mathsf{flag} = 1$ is enough to detect invalidity, the number of queries of the same type to make depends on j and $D^{\mathsf{coeff}}_{n,B}$ (in addition to s_i, but which is unknown) as well as a chosen probability lower-bound with which the adversary would like to be confident about whether $j + s_i \in \{-B, \ldots, 0, \ldots, B\}$. We denote this number by ρ_j, and the algorithm then goes as follows:

For $i = 0$ up to $n - 1$:

- Set $s'_i = B$, $j \leftarrow 1$, $k \leftarrow 0$, $f \leftarrow 0$;
- While $f = 0$:
  - make a type-(i, j) decryption query;
  - $f \leftarrow F_{i,j}$;
  - $k \leftarrow k + 1$;
  - If $k = \rho_j$ and $F_{i,j} = 0$,
    $j \leftarrow j + 1$, $s'_i \leftarrow s'_i - 1$, $k \leftarrow 0$;

Output $s = \sum_{k=1}^{n-1} s'_i$.

The number of decryption queries is thus bounded by $n \sum_{j=1}^{2B+1} \rho_j$. If we let $\rho = \max_j \rho_j$, it is possible to reduce this bound down to $n\rho \log(2B{+}1)$ by looking for j_i so that $F_{i,j_i-1} = 0$ for the ρ_{j_1-1} repetitions and $F_{i,j_i} = 1$ for some type-(i, j_i) decryption queries, as in a dictionary. That is, by guessing j_i in the middle of the upper half of the remaining j's to test in the case $F_{i,j} = 0$, for ρ_j repetitions of the current index j, and in the middle of the lower half of the remaining j's to test if $F_{i,j} = 1$ for the current index j. This improvement is marginal for values of B used in the existing implementation of Polka. However, for the asymptotic case and for Polka's variant with Gaussian noise (i.e., $B = \sqrt{n}\sigma$ with standard deviation $\sigma \in \Omega(\sqrt{n})$), it is more substantial. Next, we finally evaluate the expected number of decryption queries of these attacks assuming that the leakage does not necessarily reveal flag with probability 1, and for Polka's concrete value of B.

5.4 Quantitative assessment

We now give concrete values for the number of queries expected to recover the key from the flag's information, specifically for implemented Polka's parameters. That is $B = 1$, the coefficients of the secret s and dummy noise e'_2 lie in the range $\mathcal{S} = \{-1, 0, 1\}$ with respective probabilities $P_{\mathcal{S}} = \{0.25, 0.5, 0.25\}$. We focus on recovering one coefficient of s, and multiply that cost by $n = 1024$ for the cost of the full polynomial s.

5.4.1 Attack against de-randomized Polka

The strategy to recover one secret coefficient s_i in de-randomized Polka is rather simple. We use a two-stage algorithm where the target of the query is to nudge edge-values ($s_i = 1$ or $s_i = -1$) to cross the bound instead of starting the *probe* $j = 0$ and increasing it one at a time (as in Section 5.1). We write $p_j = \Pr[s_i = j]$ for simplicity.

Perfect Flag. We assume that the adversary gets the exact value of the flag from the leakage (i.e., $F^*_{i,j}$). Each step of the attack requires a single decryption query:

Stage 1. Query $(i, 1)$. If $F^*_{i,1} = 1$ output $s_i = 1$ and stop otherwise
Stage 2. Query $(i, -1)$. If $F^*_{i,-1} = 1$ output $s_i = -1$ and output $s_i = 0$ otherwise.

As a result, the expected number of queries to recover s_i correctly is:

$$N_a^{\text{de-rand}} = 1 \cdot p_1 + 2 \cdot (p_0 + p_{-1}) = 1.75. \tag{4}$$

Imperfect flag. In order to deal with noisiness in the flag information, the attack repeats querying the same input several times to make it more reliable as in Section 5.1. We use a simply majority vote approach on the answers of the flag in each stage by repeating it m times. The two-stage algorithm is then adapted as follow:

Stage 1. Query $(i, 1)$ m (odd) times and obtain m binary flag values $\tilde{F}^{*1}_{i,1}, \ldots, \tilde{F}^{*m}_{i,1}$. If $\#\{k : \tilde{F}^{*k}_{i,1} = 1\} > m/2$ output $s_i = 1$ and otherwise ...
Stage 2. Query $(i, -1)$ m times and obtain m binary flag values $\tilde{F}^{*1}_{i,-1}, \ldots, \tilde{F}^{*m}_{i,-1}$. If $\#\{k : \tilde{F}^{*k}_{i,-1} = 1\} > m/2$ output $s_i = -1$ and otherwise output $s_i = 0$.

Let the probability of the flag being wrong in each query be $\Pr[\tilde{F}^{*k}_{i,1} \neq F^*_{i,1}] = \Pr[\tilde{F}^{*k}_{i,-1} \neq F^*_{i,-1}] = \eta$. The probability of a wrong majority vote thus equals:

$$\varepsilon(m, \eta) := \sum_{k=0}^{(m-1)/2} \binom{m}{k} (1-\eta)^k \eta^{m-k}.$$

The algorithm outputs an incorrect guess if (1) $s_i = 1$ and it stops at Stage 2, which occurs with probability $\varepsilon(m, \eta)$, or (2) $s_i \neq 1$ and it stops at Stage 1 with probability $\varepsilon(m, \eta)$ or it does not stop at Stage 1 with probability $1 - \varepsilon(m, \eta)$, but the second majority vote is wrong with probability $\varepsilon(m, \eta)$. Hence, the overall probability of error is:

$$\begin{aligned} P_{\text{err}} &= \varepsilon(m, \eta) \times p_1 + (2\,\varepsilon(m, \eta) - \varepsilon(m, \eta)^2) \times (p_0 + p_{-1}), \\ &= 1.75\,\varepsilon(m, \eta) + 0.75\,\varepsilon(m, \eta)^2. \end{aligned}$$

The expected number of queries is m times the probability that the attack stops at Stage 1 plus $2m$ times the probability that the attack stops at Stage 2:

$$\begin{aligned} N_a^{\text{de-rand}}(m, \eta) &= m \times \big((1 - \varepsilon(m, \eta)) \times p_1 + \varepsilon(m, \eta) \times (1 - p_1)\big) \\ &\quad + 2m \times \big((1 - \varepsilon(m, \eta)) \times (1 - p_1) + \varepsilon(m, \eta) \times p_1\big) \\ &= 1.75\,m - 0.5\,\varepsilon(m, \eta)\,m. \end{aligned}$$

5.4.2 Attack against Polka

As described in Section 5.3, the flag now gives information about the sum $j + s_i + e'_{2,i,j}$ with unknown value of $e'_{2,i,j}$. Conceptually, it is similar to give the adversary erroneous access to the flag value, where the error is induced by the added dummy noise. Therefore, a number of repetitions is always required, even when the perfect flag is given.

Perfect Flag. The adversary is oblivious of the exact value of $e'_{2,i,j}$ each time she makes a decryption query. Yet, she knows its distribution and the range of $j + s_i + e'_{2,i,j}$ for each value of s_i is deterministic, though overlap. The idea is again, to nudge edge-values in the range to cross the bound where this event depends on the distribution of the dummy (which is known). By repeatedly querying on the edge-input, the adversary can *wait* until the value of $e'_{2,i,j}$ that flips the flag appear. We recall that a single such flip is enough to make a perfect decision; there is no false positive. If there is no flip after some repetition, she can exclude a guessed value of s_i and keep proceeding the next step, with some probability of error. We adapt the two-stage algorithm on de-randomized Polka with repetition with a quantity m in order to eliminate the effect of the dummy as follows.[7]

Stage 1. Query $(i, 1)$ repeatedly:

- Stop as soon as a $F_{i,j} = 1$ is observed and output $s_i = 1$, otherwise ...
- After m queries with $\mathsf{flag} = 0$, proceed to Stage 2.

Stage 2. Query $(i, -1)$ repeatedly:

- Stop as soon as $F_{i,j} = 1$ is observed and output $s_i = -1$, otherwise ...
- After seeing m querieswith $F_{i,j} = 0$, output $s_i = 0$.[8]

Obviously, $\Pr[F_{i,1} = 1 | s_i = 1] = \Pr[e'_{2,i,1}[i] = 1]$ and $\Pr[F_{i,-1} = 1 | s_i = -1] = \Pr[e'_{2,i,-1}[i] = -1]$, where $e'_{2,i,j}[i]$ is the i-th coefficient of the additional noise $e'_{2,i,j}$ independently picked by the decryption in any (i, j)-query. If we let α be this probability, we have $\alpha = 0.25$. Since there is no false positive, the algorithm outputs an incorrect guess either if the flag never flips at Stage 1 when $s_i = 1$ or if the flag never flips at Stage 2 when $s_i = -1$. The total error probability is:

$$P_{\mathrm{err}} = (1 - \alpha)^m \times p_1 + (1 - \alpha)^m \times p_{-1} = \frac{1}{2}(1 - \alpha)^m.$$

Next, we compute the attacks' expected number of decryption queries:

$$N_a^{\mathrm{rand}}(m) = \sum_{j=-1}^{1} \mathbb{E}[Q(m) | s_i = j] \times p_j,$$

where $Q(m)$ is the random variable counting the number of decryption queries:

- If $s_i = 1$, the algorithm stops exactly after j decryption queries, for $j = 1$ to m, with probability $(1 - \alpha)^{j-1}\alpha$ if the output is correct, and it stops after $2m$ queries if the output is wrong (i.e., it outputs 0), which happens with probability $(1 - \alpha)^m$.
- If $s_i = -1$, the algorithm stops exactly after $m + j$ decryption queries, for $j = 1$ to m, with probability $(1 - \alpha)^{j-1}\alpha$ if the output is correct, and it stops after $2m$ queries if the output is wrong, which also happens with probability $(1 - \alpha)^m$.

[7] Section 5.3 uses the ρ notation. We use m here for consistency with the imperfect flag attacks.
[8] The number of repetitions is not necessary equal in Stages 1and 2, we fix them at m for simplicity.

- If $s_i = 0$, the algorithm always makes $2m$ queries to the decryption oracle.

We conclude that:

$$\begin{aligned} N_a^{\text{rand}}(m) &= \Big(2m \times (1-\alpha)^m + \textstyle\sum_{j=1}^{m} j \times (1-\alpha)^{j-1}\alpha\Big) \times p_1 \;+\; 2m \times p_0, \\ &\quad + \Big(2m \times (1-\alpha)^m + \textstyle\sum_{j=1}^{m} (m+j) \times (1-\alpha)^{j-1}\alpha\Big) \times p_{-1}, \\ &= m + 0.5 \times 0.75^m \times m + (2 + 0.25 \times m) \times (1 - 0.75^m), \\ &= 1.25\, m + 2 + (0.25\, m - 2)\, 0.75^m. \end{aligned}$$

Imperfect Flag. Let again η be the probability $\Pr[\tilde{F}_{i,1} \neq F_{i,1}] = \Pr[\tilde{F}_{i,-1} \neq F_{i,-1}]$ that the noise makes the flag erroneous. Similarly to the de-randomized case, (more) repetitions and a (modified) majority vote help removing the effect of such errors. Compared to the previous case where $\eta = 0$, the flag may wrongly flips, which introduces false positives. Therefore, the attack does not guess s_i after a first flip, but after a threshold of τ flips.

Stage 1. Query $(i, 1)$ m times, obtain flag values $\tilde{F}_{i,1}^1, \ldots, \tilde{F}_{i,1}^m$.
If $\#\{k : \tilde{F}_{i,1}^k = 1\} \geq \tau$ output $s_i = 1$, otherwise proceed to Stage 2.

Stage 2. Query $(i, -1)$ m times, obtain flag values $\tilde{F}_{i,-1}^1, \ldots, \tilde{F}_{i,-1}^m$.
If $\#\{k : \tilde{F}_{i,-1}^k = 1\} \geq \tau$ output $s_i = -1$, otherwise outputs $s_i = 0$.

We keep the notation $\alpha = \Pr[F_{i,1} = 1 | s_i = 1] = \Pr[F_{i,-1} = 1 | s_i = -1] = 0.25$, and additionally define $\tilde{\alpha} = \Pr[\tilde{F}_{i,1} = 1 | s_i = 1]$. Therefore, $\tilde{\alpha}$ is equal to $\Pr[F_{i,1} = 0 | s_i = 1] \cdot \eta + \Pr[F_{i,1} = 1 | s_i = 1] \cdot (1 - \eta) = \alpha + (1 - 2\alpha)\eta = 0.25 + 0.5\eta$. Similarily, we have $\Pr[\tilde{F}_{i,1} = 1 | s_i = 1] = \tilde{\alpha}$. Moreover, $\Pr[\tilde{F}_{i,1} = 1 | s_i \neq 1] = \Pr[\tilde{F}_{i,-1} = 1 | s_i \neq -1] = \eta$ since there is no matching $F_{i,j}$ in these two events. To compute the probability P_{err} of making a wrong guess for s_i, we finally set:

$$\varepsilon(m, \mathsf{p}, \tau) = \sum_{k=1}^{\tau-1} \binom{m}{k} \mathsf{p}^k (1 - \mathsf{p})^{m-k}.$$

At Stage 1, the algorithm incorrectly rejects $s_i = 1$ with probability $\varepsilon(m, \tilde{\alpha}, \tau)$, and incorrectly accepts $s_i = 1$ with probability $1 - \varepsilon(m, \eta, \tau)$. Symmetrically, at Stage 2, the algorithm incorrectly rejects $s_i = -1$ with probability $\varepsilon(m, \tilde{\alpha}, \tau)$, and incorrectly accepts $s_i = -1$ with probability $1 - \varepsilon(m, \eta, \tau)$. Hence:

$$\begin{aligned} P_{\text{err}} &= \varepsilon(m, \tilde{\alpha}, \tau) \times p_1, \\ &\quad + \big(1 - \varepsilon(m, \eta, \tau) + \varepsilon(m, \eta, \tau)\, \varepsilon(m, \tilde{\alpha}, \tau)\big) \times p_{-1}, \\ &\quad + \big(1 - \varepsilon(m, \eta, \tau) + \varepsilon(m, \eta, \tau)\, \varepsilon(m, \eta, \tau)\big) \times p_0. \end{aligned}$$

The algorithm always queries decryption the first m times in Stage 1, and it only enters Stage 2 unless Stage 1 outputs 1. Since the probability of entering Stage 2 is $\varepsilon(m, \tilde{\alpha}, \tau)$ if $s_i = 1$ and $\varepsilon(m, \eta, \tau)$ if $s_i \neq 1$, the expected number of queries equals:

$$N_a^{\text{rand}}(m, \eta) = m + m\big(0.25\, \varepsilon(m, \tilde{\alpha}, \tau) + 0.75\, \varepsilon(m, \eta, \tau)\big).$$

We pick the threshold τ using the cutoff constant in Neyman Pearson lemma for Bernoulli variables, that is $\tau \approx m \frac{\ln \frac{1-\eta}{1-\tilde{\alpha}}}{\ln \frac{\tilde{\alpha}(1-\eta)}{\eta(1-\tilde{\alpha})}}$. This allows evaluating the complexity N_a of attacks with accuracy 99.99% of guessing the correct secret in function of the introduced noise η (when $\eta = 0$, the results come from the perfect flag version) in Figure 3. The figure confirms that the attack against de-randomized Polka succeeds with low complexity, even when the probability to extract the flag decreases. As for the DPA attacks in Sections 2 and 3, the presence of dummies marginally improves side-channel security.

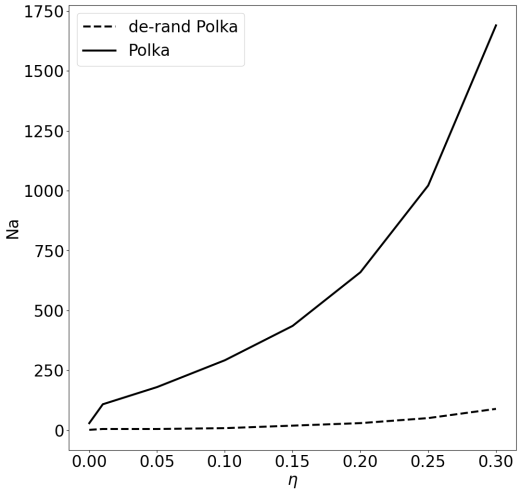

Figure 3: Number of queries to recover one coefficient of secret key from the flag with $> 99.99\%$ accuracy in function of the probability of error on the flag η.

Remark. The results of the attacks above are applicable for any $p > 4B + 1$. We notice that in the only implemented version of Polka, the choice of the modulo p is exactly $4B + 1$. This choice of p implies that the modular reduction over p always lies in the range $\{-2B, -2B + 1, \ldots, 2B - 1, 2B\}$ and there is no query (of e_2 in chosen ciphertexts) that could nudge the edge-values to cross the bound for the adversary to observe the flap flip. With this choice of p, the first norm check can be removed in Polka without any compensation in the correctness of the decryption process. This implies that attacks targeting the flag of the first check are not applicable. Removing this check also eliminates the leakage of the `norm_check` computation used by the attack shown in Section 4.

6 Discussion, impact and open problems

Our results show that removing the FO-transform has a more quantitative than qualitative impact for reducing the sources of leakage in PQ encryption schemes and key encapsulation mechanisms. More precisely, the instantiation of this idea for Polka allows decreasing the amount of leakage that can be exploited, while leaving a number of attack paths that require countermeasures. The addition of dummy ciphertexts is also shown to help to a moderate extent only. If Polka had to be implemented, it would therefore be necessary to protect its flag (and the norm checks driving it) anyway. In order to minimize the attack surface, one option could then be to move to a random execution as soon as a flag switches to one (e.g., as done for the hash function call in Step 6 of Section 2.1 or Step 9 of Section 3.1). But this should be done in a hard-to-notice manner: a likely non-trivial task. Alternatively, computations should be masked to prevent the guessing DPAs of Sections 2 and 3. Compared to Kyber, this remains an improvement, as it avoids the FO-Calypse where extracting a bit of information that corresponds to re-encrypted plaintexts (for which the leakage of many operations can be used) leads to key-recoveries. But it leaves wide room for improving the leakage-resilience or resistance of such schemes.

From the cryptographic design viewpoint, and given that removing the FO-Transform is one of the main candidate to improve the side-channel security of Kyber [HHMS25], this raises the question whether one could minimize the amount of norm checks, or enable non constant-time implementations that stop computing on sensitive data as early as possible. From the implementation viewpoint, this also raises the question of how to efficiently

protect the flag, mask computations, or move to random executions in a hard-to-notice manner. Given the difficulty to analyze systematically all the attack paths in PQ algorithms like Kyber and Polka, obtaining more formal guarantees relying on well circumvented falsifiable assumptions naturally remains a long-term challenge as well. And in case protecting such lattice-based schemes against leakage turns out to be difficult, it of course remains interesting to check whether other schemes, possibly based on other assumptions, allow a scarcer use of implementation-level countermeasures. Alternatively, the use of zero-knowledge proof techniques on top of the CPA encryption in order to avoid using the secret key before verifying the validity of the resulting ciphertext during decapsulation could provide a generic solution [ABH+22], though with possibly high overheads.

References

[ABF+] Melissa Azouaoui, Joppe W. Bos, Bjorn Fay, Marc Gourjon, Yulia Kuzovkova, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Surviving the FO-Calypse: Securing PQC Implementations in Practice. *Real Workd Cryto Symposium*, 2022.

[ABH+22] Melissa Azouaoui, Olivier Bronchain, Clément Hoffmann, Yulia Kuzovkova, Tobias Schneider, and François-Xavier Standaert. Systematic study of decryption and re-encryption leakage: The case of kyber. In *COSADE*, volume 13211 of *Lecture Notes in Computer Science*, pages 236–256. Springer, 2022.

[AER23] Guilhèm Assael, Philippe Elbaz-Vincent, and Guillaume Reymond. Improving Single-Trace Attacks on the Number-Theoretic Transform for Cortex-M4. In *HOST*, pages 111–121. IEEE, 2023.

[BBC+20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-Level vs. Implementation-Level Physical Security in Symmetric Cryptography - A Practical Guide Through the Leakage-Resistance Jungle. In *CRYPTO (1)*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020.

[BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing Arithmetic/Boolean Masking Conversions for Fun and Profit with Application to Lattice-Based KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):553–588, 2022.

[BDH+19] Ciprian Baetu, F. Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In *EUROCRYPT (2)*, volume 11477 of *Lecture Notes in Computer Science*, pages 747–776. Springer, 2019.

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and Higher-Order Implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):173–214, 2021.

[CDH+19] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John Schanck, Peter Schwabe, William Whyte, and Shenfei Zhang. NTRU Algorithm Specifications and Supporting Documentation. 2019.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[DMMS21] Sébastien Duval, Pierrick Méaux, Charles Momin, and François-Xavier Standaert. Exploring Crypto-Physical Dark Matter and Learning with Physical Rounding Towards Secure and Efficient Fresh Re-Keying. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):373–401, 2021.

[HHMS25] Kathrin Hövelmanns, Andreas Hülsing, Christian Majenz, and Fabrizio Sisinni. (Un)breakable Curses - Re-encryption in the Fujisaki-Okamoto Transform. In *EUROCRYPT (2)*, volume 15602 of *Lecture Notes in Computer Science*, pages 245–274. Springer, 2025.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):88–113, 2021.

[HKG+21] Meziane Hamoudi, Amina Bel Korchi, Sylvain Guilley, Sofiane Takarabt, Khaled Karray, and Youssef Souissi. Side-Channel Analysis of CRYSTALS-Kyber and A Novel Low-Cost Countermeasure. In *ICSP*, volume 1497 of *Communications in Computer and Information Science*, pages 30–46. Springer, 2021.

[HLM+23] Clément Hoffmann, Benoît Libert, Charles Momin, Thomas Peters, and François-Xavier Standaert. POLKA: Towards Leakage-Resistant Post-quantum CCA-Secure Public Key Encryption. In *Public Key Cryptography (1)*, volume 13940 of *Lecture Notes in Computer Science*, pages 114–144. Springer, 2023.

[HMM+23] Clément Hoffmann, Pierrick Méaux, Charles Momin, Yann Rotella, François-Xavier Standaert, and Balazs Udvarhelyi. Learning with Physical Rounding for Linear and Quadratic Leakage Functions. In *CRYPTO (3)*, volume 14083 of *Lecture Notes in Computer Science*, pages 410–439. Springer, 2023.

[HSST23] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting Belief Propagation to Counter Shuffling of NTTs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):60–88, 2023.

[KT23] Yen-Ting Kuo and Atsushi Takayasu. A Lattice Attack on CRYSTALS-Kyber with Correlation Power Analysis. In *ICISC (1)*, volume 14561 of *Lecture Notes in Computer Science*, pages 202–220. Springer, 2023.

[Man04] Stefan Mangard. Hardware Countermeasures against DPA ? A Statistical Analysis of Their Effectiveness. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.

[MOS11] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Inf. Secur.*, 5(2):100–110, 2011.

[NHPM25] Rishub Nagpal, Vedad Hadzic, Robert Primas, and Stefan Mangard. Efficient SPA Countermeasures using Redundant Number Representation with Application to ML-KEM. *IACR Cryptol. ePrint Arch.*, page 679, 2025. URL: https://eprint.iacr.org/2025/679.

[PP19] Peter Pessl and Robert Primas. More Practical Single-Trace Attacks on the Number Theoretic Transform. In *LATINCRYPT*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

[PS24] Duyên Pay and François-Xavier Standaert. Side-Channel Analysis of Arithmetic Encodings for Post-Quantum Cryptography: Cautionary Notes with Application to Kyber. In *AFRICACRYPT*, volume 14861 of *Lecture Notes in Computer Science*, pages 260–281. Springer, 2024.

[RCDB24] Prasanna Ravi, Anupam Chattopadhyay, Jan-Pieter D'Anvers, and Anubhab Baksi. Side-channel and Fault-injection attacks over Lattice-based Post-quantum Schemes (Kyber, Dilithium): Survey and New Results. *ACM Trans. Embed. Comput. Syst.*, 23(2):35:1–35:54, 2024.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):307–335, 2020.

[RRD+23] Gokulnath Rajendran, Prasanna Ravi, Jan-Pieter D'Anvers, Shivam Bhasin, and Anupam Chattopadhyay. Pushing the Limits of Generic Side-Channel Attacks on LWE-based KEMs - Parallel PC Oracle Attacks on Kyber KEM and Beyond. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(2):418–446, 2023.

[SAB+22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-Kyber Algorithm Specifications and Supporting Documentation. *NIST Post-Quantum Cryptography Standard*, 2022.

[SHM+] T. Schoenauen, C. Hoffmann, C. Momin, T. Peters, and F.-X. Standaert. Leveled Software Implementation of Polka and Comparison with Uniformly Masked Kyber. In *Proceedings of ACNS-SCI*. To appear in Lecture Notes in Computer Science. URL: https://perso.uclouvain.be/fstandae/PUBLIS/328.pdf.

[TMS24] Tolun Tosun, Amir Moradi, and Erkay Savas. Exploiting the Central Reduction in Lattice-Based Cryptography. *IEEE Access*, 12:166814–166833, 2024. doi:10.1109/ACCESS.2024.3494593.

[UXT+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of Re-encryption: A Generic Power/EM Analysis on Post-Quantum KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):296–322, 2022.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft Analytical Side-Channel Attacks. In *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.

[WXT25] Kai Wang, Dejun Xu, and Jing Tian. An Improved Two-Step Attack on Lattice-Based Cryptography: A Case Study of Kyber. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 44(9):3643–3647, 2025.

[XPR+22] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, David F. Oswald, Wang Yao, and Zhiming Zheng. Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems With Chosen Ciphertexts: The Case Study of Kyber. *IEEE Trans. Computers*, 71(9):2163–2176, 2022.

A Polka's leakage-resistance requirements

Table 4: Polka's former leakage-resistance requirements.

	CPA decryption	**ciphertext validity checking**		
		computation		**control**
targets (ops/var)	poly. multiplications	norm checks (H inputs)	(H outputs) AE	$\bot, \top$
type of attack	guessing DPA	guessing SPA	guessing DPA	flag DPA
countermeasure	masking	parallelism or shuffling	masking	$\emptyset$