IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–21. DOI:XXXXXXXX

Power Reveals, Timing Conceals

Side-Channel Attacks and Hiding Countermeasures for HQC's Fixed-Weight Vector Sampling

Dina Hesse[1], Markus Krausz[2], Raagavan Murugananthan[2], Tabea Wollinger[1] and Tim Güneysu[1,3]

[1] Horst Görtz Institute for IT Security, Ruhr University Bochum, Bochum, Germany, firstname.lastname@rub.de
[2] TÜVIT, Essen, Germany, m.krausz@tuvit.de,r.murugananthan@tuvit.de
[3] DFKI GmbH, Bremen, Germany

Abstract.
Fixed-weight sampling is a core primitive in many post-quantum schemes, including the HQC key encapsulation mechanism. An early implementation of fixed-weight vector sampling in HQC was shown by Guo et al. (CHES 2022) to suffer from a timing side-channel vulnerability, leading to complete key recovery. This timing side-channel was fixed in the current HQC version, however, power side-channel leakage is not addressed. In this work, we demonstrate that fixed-weight vector sampling in HQC is vulnerable to power side-channel attacks and present two practical attacks.
First, we construct a power-based distinguisher targeting the support-vector generation and employ the strategy developed by Guo et al. (CHES 2022) to recover the shared key. Our attack recovers the key with a 100% success rate using 900,000 distinguisher calls. On these grounds, we evaluate hiding countermeasures based on dummy operations and find that they linearly increase the trace requirement for a successful distinguishing attack by the number of dummy operations.
Second, we target a masked software implementation of the fixed-weight vector sampling in HQC and demonstrate a single-trace attack on the support conversion that recovers the secret key, again with a success rate of 100%. We then discuss leakage attribution, specifically how shares are unintentionally recombined. Finally, we investigate how hiding techniques such as bitslicing, shuffling, and dummy operations can enhance the security of the implementation. In particular, the use of shuffling can lead to a complete prevention of our attack.
Our results show that fixed-weight vector sampling of HQC is highly susceptible to power side-channel analysis. In particular, our results highlight that a combination of masking and hiding is required to effectively protect the implementations.

Keywords: Side-Channel Analysis · Hiding · Masking · HQC · FWVS

1 Introduction

In 2025, NIST selected Hamming Quasi-Cyclic (HQC) [AAB+22] as a code-based Key Encapsulation Mechanism (KEM) for standardization for Post-Quantum Cryptography (PQC). As part of the transition from standardization to real-world deployment, HQC implementations must ensure robust security even in adversarial environments. This is particularly crucial for target platforms such as embedded microcontrollers, smart cards, and hardware accelerators, where an attacker with physical access can observe side-channels like power consumption.

During key generation and encryption in HQC, secret bit vectors with a fixed Hamming weight are generated, which is considered as Fixed-Weight Vector Sampling (FWVS). While

mathematically elegant in theory, achieving a secure implementation in practice is highly challenging. An early implementation of HQC's FWVS was shown by Guo et al. [GHJ+22] to be vulnerable to a timing side-channel attack, which enabled key recovery and, thus, prompted a specification change. While the current HQC specification eliminates timing leakage by employing a constant-time Fisher–Yates shuffle algorithm, power side-channel leakage has not been addressed yet.

Power Side-Channel Analysis (SCA) exploits data-dependent variations in the power consumption by observing a number of power traces of a device. PQC schemes are highly interesting targets to SCA since they operate on large vectors, including more complex building blocks, such as random sampling and decoding, compared to traditional cryptographic schemes. In many schemes, vectors or polynomials are processed highly serially, element by element. This sequentiality enables horizontal single-trace and multivariate analyses that facilitate the aggregation of leakages. Large vector operations not only require particular care to achieve efficient implementations, but, at the same time, show a large attack surface for SCA due to those involved trade-offs.

This holds in particular when implementing masking, which splits secret intermediate values into independent shares that are processed separately, ideally resulting in secret-independent leakage. However, in particular for non-linear operations, implementing masking is non-trivial and unintended or careless recombination of shares quickly results in observable leakage. Since masking itself often comes at large overhead, we are in need of an efficient methodology to achieve a sufficiently high noise level to effectively mitigate SCAs.

As an additional solution, hiding techniques provide a relevant set of countermeasures for PQC in general and, in particular, HQC. Hiding aims to increase the noise level in the system either in amplitude (via noise injection) or in time (via random delays, dummy operations, or shuffling). All this leads to a reduction of the signal-to-noise ratio to require higher effort from the attacker in terms of higher measurement load or more sophisticated methods for analysis. In single-trace SCA settings, hiding can render an attack impractical.

Prior work has shown that side-channel secure fixed-weight sampling in HQC [GHJ+22] as well as in other PQC algorithms [HFG+26] is challenging to implement efficiently and securely. In this matter, [KLRBG23] evaluates masked implementations of FWPS in software and reports significant overheads. In response to this, we review FWVS in HQC in depth to explore the following questions: Can the power side-channel leakage of fixed-weight sampling be utilized in practice, and to what extent can hiding countermeasures lead to secure and efficient implementations?

1.1 Our Contributions

To address these questions, we examine the issue of side-channel protection of FWVS by presenting two power side-channel attacks on HQC and discuss potential countermeasures and their implications.

- We build a **power side-channel distinguisher** for a hardware implementation of the constant-time Fisher-Yates support-vector generation that enables complete key recovery following the strategy of Guo et al. [GHJ+22].

- We evaluate hiding via **dummy operations** as part of this implementation with different configurations and show empirically that the attacker's effort grows approximately linearly with the number of dummy operations.

- Finally, we target a **masked software implementation** of HQC's FWVS and demonstrate a novel **single-trace attack** on the support conversion that recovers the secret key. We attribute the observed leakage to unintentional recombinations of

shares and discuss hiding techniques for mitigation, including the impact of bitslicing, shuffling, and dummy operations.

1.2 Related Work

Most recent side-channel attacks on HQC target the decapsulation procedure, particularly the decoding steps. In this context, Schamberger et al. [SRSWZ20] demonstrate full key recovery by exploiting leakage that indicates successful error correction. Subsequent work developed increasingly efficient attacks on the Reed-Solomon decoder [SHR+22], the Reed-Muller decoder [PRJ+25], and the Hadamard transform [GLG22]. An attack on optimized software implementations requiring fewer than 100 traces [MNMD25] puts further demands for masking as a countermeasure.

For fixed-weight sampling in other schemes, [HFG+26] present a single-trace attack on a masked Fixed-Weight Polynomial Sampling (FWPS) implementation for NTRU, and [KLRBG23] present bitsliced implementations for different FWVS variants.

2 Preliminaries

2.1 Side-Channel Attacks and Countermeasures

One of the most prominent hardware attack scenarios is Side-Channel Analysis (SCA). It is a passive attack that observes a side-channel during computation and exploits the observation to retrieve information about the processed secrets. A side-channel is unintended information leakage from channels other than the intended data pathway of a cryptographic implementation. Examples for these channels are physical characteristics of the system, such as timing differences, photonic emissions, electric power consumption and Electromagnetic (EM) emissions. In this work, we focus on the observation of the power consumption, caused by the secret-related transistor switching in the chip. Since the introduction of timing and power SCA [Koc96, KJJ99], many attacks and countermeasures have been proposed and discussed. Some attacks only require one or a few observed computations (called traces) to recover the secret and are commonly named Simple Power Analysis (SPA) or single-trace attacks. Other attacks, like Differential Power Analysis (DPA), Correlation Power Analysis (CPA) and Deep-Learning based SCA, usually involve up to millions of traces.

Masking The main classes of side-channel countermeasures are *masking* and *hiding*. Masking splits secret intermediate values into shares, ensuring that each share is independent of the secret. This makes the leakage independent of the secret. By using theoretical models, e.g. the noisy leakage [PR13] or the t-probing model [ISW03], the security of masking scheme designs can be theoretically proven. These proofs rely on a certain noise level of the system. Masking amplifies this noise to protect the cryptographic implementations [DDF14]. The abstraction level of these models sometimes leads to a discrepancy between the theoretical security proofs and practical security [WBD24, HNPS24], which can be assessed by statistically evaluating measured side-channel leakage traces.

In this paper, $x^{B,b}$ denotes a Boolean shared integer x of bit length b, where $x_i^{B,b}$ is the i-th share of $x^{B,b}$. Bold letters denote vectors.

Hiding In contrast to masking, which aims to *eliminate* the dependency between secret and leakage, hiding *reduces* the statistical dependence between the observable side-channel signal and the intermediate (secret-dependent) values of the computation. A broad range of hiding techniques targeting power and EM side-channels have been proposed. Based on [MOP07], they can be grouped into strategies that increase noise and those that reduce

Table 1: Parameter sets for HQC.

Instance	n	ω	$\omega_r = \omega_e$
HQC-1	17,669	66	75
HQC-3	35,851	100	114
HQC-5	57,637	131	149

the signal component of the side-channel. Some techniques affect the amplitude of the power, whereas others operate in the time domain and thus influence the amplitude domain only indirectly. Software-based hiding methods mostly target the time domain. Hiding countermeasures do not guarantee perfect protection as the leakage is not eliminated, but the data-dependent power consumption is hidden, and therefore attacks require more traces to extract information. In the following, two types of hiding countermeasures are explained in more detail.

Shuffling randomizes the order of independent operations in a cryptographic implementation, so that sensitive computations occur at varying times across executions and, thus, hindering reliable trace alignment [VMKS12]. For k independent computation steps, the strongest shuffling strategy samples the order of the computations uniformly from all $k!$ permutations. Lighter variants, such as random-start shuffling, yield only k possible orderings. Effectiveness of the shuffling grows with the number of independent computation steps k, but algorithmic dependencies often limit how many operations can be shuffled.

Another hiding technique randomly inserts dummy operations before, during, and after the execution of the algorithm. The number of total dummy operations is fixed, but the positions are random in each iteration. As the number of dummy operations increases, the temporal position of the actual operation becomes more variable. However, this comes at the cost of increased execution time [MOP07]. Furthermore, it is important that the dummy operations are not distinguishable from real operations. If they are distinguishable, the time when a dummy operation is executed can be cut out of the trace, which results in perfectly aligned traces.

2.2 HQC

Hamming Quasi-Cyclic (HQC) is a code-based KEM built from a Public-Key Encryption (PKE) using the HHK transform [HHK17], a variant of the FO transform [FO99]. Algorithm 1 depicts the key generation, Algorithm 2 the encryption, and Algorithm 3 the decryption of HQC-PKE. The FWVS we target in this paper is highlighted in red. $\mathcal{R}$ denotes the polynomial ring $\mathbb{F}_2[X]/(X^n - 1)$ and $\mathcal{R}_\omega = \{x \in \mathcal{R}; \mathrm{HW}(x) = \omega\}$. $\xleftarrow{\$}{}^{\theta}$ denotes uniformly random sampling of an element seeded with θ. Table 1 lists the relevant parameters for FWVS in HQC.

Algorithm 1 KeyGen

Require:
Ensure: (sk, pk)
1: $\boldsymbol{h} \xleftarrow{\$} \mathcal{R}$
2: $(\boldsymbol{x}, \boldsymbol{y}) \xleftarrow{\$} \mathcal{R}_\omega^2$
3: sk $\leftarrow (\boldsymbol{x}, \boldsymbol{y})$
4: $\boldsymbol{s} \leftarrow \boldsymbol{x} + \boldsymbol{hy}$
5: pk $\leftarrow (\boldsymbol{h}, \boldsymbol{s})$

Algorithm 2 Encrypt

Require: pk, $\boldsymbol{m}, \theta$
Ensure: ct
1: $\boldsymbol{e} \xleftarrow{\$}{}^{\theta} \mathcal{R}_{\omega_e}$
2: $(\boldsymbol{r}_1, \boldsymbol{r}_2) \xleftarrow{\$}{}^{\theta} \mathcal{R}_{\omega_r}^2$
3: $\boldsymbol{u} \leftarrow \boldsymbol{r}_1 + \boldsymbol{hr}_2$
4: $v \leftarrow C.\mathrm{Encode}(\boldsymbol{m}) + \boldsymbol{sr}_2 + \boldsymbol{e}$
5: ct $\leftarrow (\boldsymbol{u}, \boldsymbol{v})$

Algorithm 3 Decrypt

Require: sk, ct
Ensure: $\boldsymbol{m}$
1: $\boldsymbol{m} \leftarrow C.\mathrm{Decode}(\boldsymbol{v} - \boldsymbol{uy})$

Fixed-Weight Vector Sampling Sampling a uniformly random vector with n elements of which $n - \omega$ are zero and ω are non-zero is commonly named FWVS and a primitive used in multiple relevant PQC schemes to generate a polynomial. An overview of various applications and (masked) algorithmic approaches for FWVS is provided by [KLRBG23]. In HQC, FWVS is used during key generation to sample the secret key consisting of $\mathbf{x}$ and $\mathbf{y}$, and also during encryption to generate $\mathbf{r_1}$, $\mathbf{r_2}$, and $\mathbf{e}$.

The HQC specification changed multiple times with respect to FWVS. Originally, rejection sampling was used to realize FWVS for key generation and encryption. For rejection sampling, a uniform random integer i is repeatedly drawn such that $0 \leq i < n$. If i has been drawn before, it gets rejected. This sampling is repeated until ω distinct integers have been selected which form the support set of the fixed-weight vector (sparse representation). Since following arithmetic operations require the vector in dense representation, the support set is then converted into a bit-string of length n with Hamming weight ω.

In 2022, the HQC specification was updated to address the timing attack on rejection sampling by Guo et al. [GHJ+22] and adopted a constant-time Fisher-Yates algorithm by Sendrier [Sen21]. Constant-time Fisher-Yates also samples a (close to) uniformly random support set.

In 2025, the specification was changed again: instead of using the same algorithm for FWVS in key generation and encryption, rejection sampling is now only used for key generation. Here, the attack by Guo et al. does not apply. For encryption, the constant-time Fisher-Yates algorithm is chosen, even though it is slower, but it does not inhibit timing leakage with respect to the input randomness. Independent of the support set sampling algorithm, the conversion to the dense representation is always the same.

3 Side-Channel Attack on Support-Vector Generation

In this section, we examine the power side-channel resistance of the constant-time Fisher-Yates support-vector generation algorithm by Sendrier [Sen21] used in the encryption of HQC. To this end, we analyze a hardware implementation following the design of [DXN+24] and demonstrate a successful side-channel attack leading to full key recovery. Eventually, we discuss hiding countermeasures that could prevent our attack.

3.1 Attack Strategy

Our attack follows the strategy of Guo et al. [GHJ+22]. They exploit the timing side-channel in rejection sampling to build a plaintext distinguisher, which is then used as a plaintext-checking oracle within the key recovery attack of Băetu et al. [BDH+19]. The replacement of the rejection sampling algorithm with the constant-time Fisher-Yates sampler in the latest revisions of the HQC specification renders this distinguisher infeasible.

However, while the timing side-channel is eliminated, the power side-channel remains a potential threat. In our attack, we construct a plaintext distinguisher based on the power side-channel and use it analogously to the attacks in [GHJ+22] and [BDH+19]. Further, we show that the power side-channel distinguisher is more precise than the timing-based distinguisher in [GHJ+22]. In the following, we first give a brief summary of the attack strategy. Then, we describe the construction of our distinguisher and discuss its strengths and limitations. Since masking of FWVS is not trivial and often leads to large designs [KLRBG23], we subsequently investigate how the constant-time Fisher-Yates sampler can be protected against power side-channels using hiding techniques.

Key Recovery from [GHJ+22] This attack uses the re-encryption property of HQC-KEM.Decaps to form the distinguisher from the power side-channel. For this strategy,

the attacker first generates two ciphertexts: c_0 and the differential ciphertext c_{diff}. c_0 is calculated with $\boldsymbol{e}, \boldsymbol{r_2}$ set to zero and $\boldsymbol{r_1}$ set to the one's vector as

$$c_0 = (\boldsymbol{u}, \boldsymbol{v}), \tag{1a}$$

$$\boldsymbol{u} = \boldsymbol{r}_\Gamma 30257 + \boldsymbol{h} \cdot \boldsymbol{r_2} = 1, \tag{1b}$$

$$\boldsymbol{v} = C.\text{Encode}(\boldsymbol{m}) + \text{Truncate}(\boldsymbol{s} \cdot \boldsymbol{r_2} + \boldsymbol{e}, \ell) = C.\text{Encode}(\boldsymbol{m}). \tag{1c}$$

Due to the fixed all-zero and all-one vectors, the v component of the ciphertext consists only the encoding of the error correcting code C without any error added.

Next, c_{diff} is constructed by flipping one bit of c_0 at a time until the distinguisher finds that the corresponding messages are not equal ($m_0 \neq m_{\text{diff}}$) for the first time. This yields a ciphertext with exactly one more bit error than the decoder can correct.

Using (c_0, c_{diff}) as inputs to the distinguisher, the attacker recovers the error vector $\boldsymbol{e}$ bit by bit. Flipping the i-th bit of c_{diff} and querying the distinguisher reveals whether position i was set to one in $\boldsymbol{e}$: if the plaintexts match after flipping, bit i was part of $\boldsymbol{e}$, otherwise it was not. The attacker can then use this method to recover the entries of $\boldsymbol{e}$, and all indices where $\boldsymbol{e}[i] = 1$ provide information about $\boldsymbol{y}$. Therefore, one c_{diff} only recovers some bits of $\boldsymbol{y}$. Thus, the attacker repeats the flip-based procedure (generating a new c_{diff}) and querying the distinguisher again, until all bits of $\boldsymbol{y}$ have been recovered.

Algorithm 4 Generate Support-Vector (Constant-Time Fisher-Yates)

Require: weight ω, XOF context ctx

for $i \leftarrow \omega - 1, i \geq 0, i \leftarrow i - 1$ **do**
  $\boldsymbol{pos}[i] \leftarrow i + \text{rand}(n - i, ctx)$
for $j \leftarrow \omega - 1, i \geq 0, i \leftarrow i - 1$ **do**
  duplicate_found $\leftarrow 0$
  for $k \leftarrow j + 1, i < \omega, i \leftarrow i + 1$ **do**
    if $\boldsymbol{pos}[j] = \boldsymbol{pos}[k]$ **then** ▷ implemented constant-time
      duplicate_found $\leftarrow 1$
  if duplicate_found $= 1$ **then** ▷ implemented constant-time
    $\boldsymbol{pos}[j] \leftarrow j$

3.2 Practical Power Side-Channel Distinguisher

Our distinguisher compares the power traces of the re-encryption of the input ciphertexts c_0 and c_{diff} and outputs whether the corresponding plaintexts are equal ($m_0 = m_{\text{diff}}$). The measurement setup is based on a ChipWhisperer CW305 target board equipped with a Xilinx Artix-7 (xc7a100tftg256-2) FPGA and a ChipWhisperer Husky capture device, which controls the target and performs the measurements at 12-bit resolution and a sampling rate of $72\,\text{MS}\,\text{s}^{-1}$. To ensure synchronous measurements, the Husky and the target device share the same clock with a frequency of 6 MHz.

In order to compare the power traces of the input ciphertexts c_0 and c_{diff}, we first build a template for the power trace of the re-encryption of the ciphertext c_0 to which future measurements are compared. During the encryption, the three vectors $\boldsymbol{r_1}, \boldsymbol{r_2}$, and $\boldsymbol{e}$ are sampled using the constant-time Fisher-Yates. For this attack, it is sufficient to analyze the FWVS of one vector, we chose $\boldsymbol{r_2}$. More specifically, we target the support-vector generation of $\boldsymbol{r_2}$ via constant-time Fisher-Yates to form the distinguisher.

We perform a specific t-test of the support set generation and control the randomness such that the output is identical for both groups, except the value in $\boldsymbol{pos}$ at one index. Furthermore, no collision occurs for both groups. As a result, the execution traces are

identical except for the memory access and comparisons involving the differing value. The part showing statistically significant leakage of the corresponding t-test is shown in Figure 1. We see leakage at the sample points where the differing value is loaded from the Block-RAM (BRAM). This indicates that reading from the BRAM is a source of leakage.

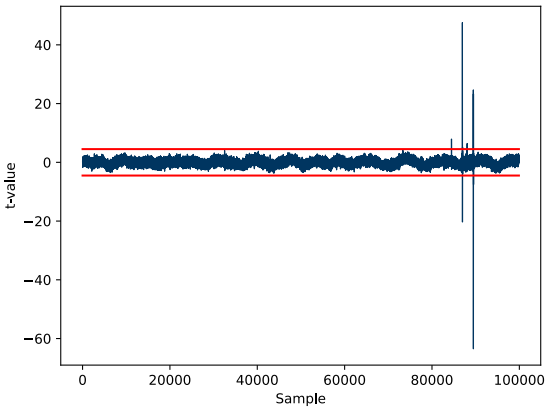
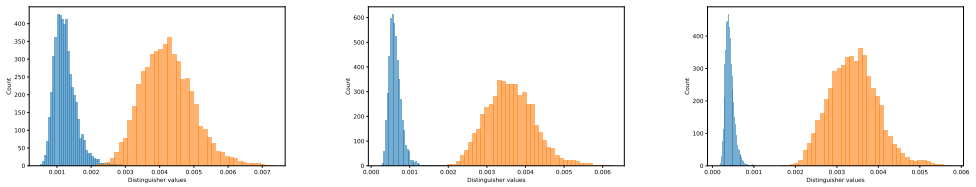


Figure 1: Specific t-test of the support-vector generation with 10,000 traces.

Applying this information, we use all ω samples at which a new value of the support vector is loaded from the BRAM as the Point of Interests (POIs) for the template. In Algorithm 4, line 6 implicitly shows this, as $\boldsymbol{pos}[j]$ must be loaded in order to compare it to the rest of the vector. For profiling, we measured $N_{\text{template}} = 500$ traces, averaged them and extracted the POIs to build the template of c_0.

A distinguisher call consists of three runs of the decryption with c_{diff} and the evaluation of the corresponding power traces. From the measured traces we extract the POIs and compare them against the c_0 template. Our experiments show that for two or more measured traces, the two possible outcomes ($m_0 = m_{\text{diff}}$ or $m_0 \neq m_{\text{diff}}$) form distinct distributions. Figure 2 illustrates the distributions of 10,000 distinguisher calls for $N_{temp} = 500$ and $N_{traces} \in \{1, 2, 3\}$. With $N_{traces} \geq 2$, the histograms do not overlap, and thus a threshold exists such that all distinguisher calls in our experiments are classified correctly. Our further experiments indicate that increasing the number of template traces N_{template} does not noticeably improve the distinguisher. Therefore, for a resilient distinguisher, we use the oracle with $N_{traces} = 3$, $N_{template} = 500$, and a threshold of 0.0013. This results in a 100% success rate of the distinguisher in $10,000$ oracle calls.



(a) $N_{traces} = 1$ (b) $N_{traces} = 2$ (c) $N_{traces} = 3$

Figure 2: Histograms for the template distinguisher using POIs with identical inputs (blue) and different inputs (orange).

Attack Performance On average, around 170 different ciphertexts c_{diff} are needed for a successful attack. Each requires around $5,000$ oracle queries to the distinguisher, which

results in an overall number of approximately $900,000$ distinguisher calls performed per attack. The execution time for one call is 0.084s on average. This leads to an execution time of 21.32 hours for a complete attack with $N_{traces} = 3$, $N_{template} = 500$, which is confirmed by our experiments.

3.3 Countermeasures (Dummy Operations)

Our attack reveals a vulnerability in the constant-time Fisher-Yates sampler to power side-channel distinguishing attacks, underscoring the necessity of countermeasures to mitigate the leakage. Masking the entire sampler is one option, however, as [KLRBG23] shows in the software context, this leads to a significant performance overhead. Even a masked implementation can still inhibit sufficient leakage for an attack as we demonstrate in Section 4. Thus, we discuss in the following how timing based hiding techniques can harden the implementation.

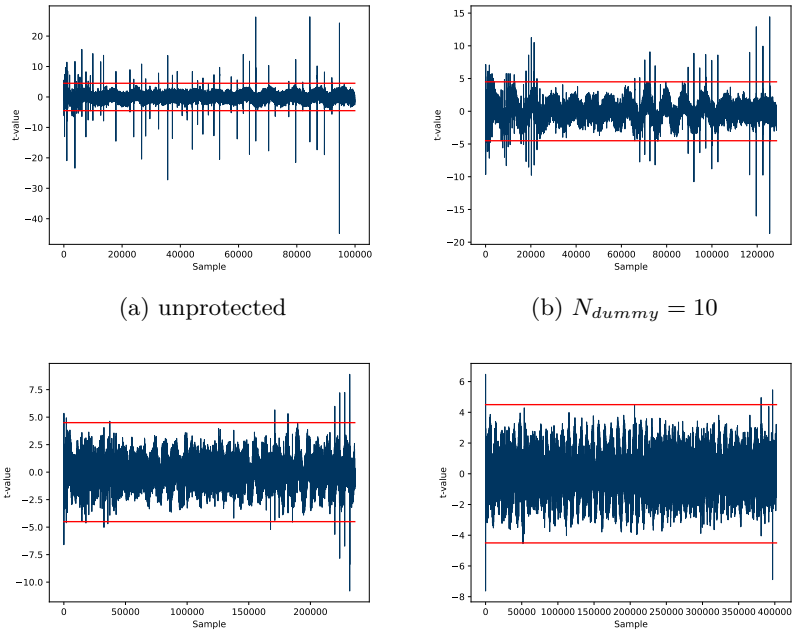
In the support-vector generation part of the Fisher-Yates sampler, it is not possible to shuffle the execution, because the comparisons must be executed in a fixed order. However, inserting additional operations that cannot be distinguished from real ones impedes our attack because the dummy operations can make two identical messages appear different. Next, we will describe our implementation of the dummy operations before analyzing the effectiveness of the protection using Test Vector Leakage Assessment (TVLA) and evaluating whether the dummy operations can be identified and how they should be generated. Then, we will evaluate our attack in the presence of dummy operations with different configurations and placement strategies.

Implementation In our distinguishing attack, we exploit the power consumption of the samples when a support-vector value is loaded from the BRAM. To hide this process, we insert dummy values in the BRAM, and load and process them similarly to the real values. These values are additionally inserted entries in the *pos* array that trigger identical control flow and memory accesses like real values, but do not influence the resulting support vector. The only difference to the normal values is that if a real value is compared to a dummy value, collisions are ignored. A non-dummy counter is required, which counts the number of processed real values. In the case of a rejection, the correct value replacing this rejected value can be calculated with this counter. This is necessary as the index does not correspond to the correct value anymore if dummy values are placed between the real values. These measures ensure deterministic execution of the algorithm and, thus, preserve correctness during the re-encryption step.

The dummy values introduce a memory and computational overhead. The memory overhead scales linearly with the number of dummy operations. Since both real and dummy values require three bytes of storage, the relative increase in BRAM usage is defined by the ratio of dummy values to the weight ω. The computational complexity also increases as the number of required iterations scales with the total number of values in the *pos* array. The number of comparisons grows quadratically with the number of dummy operations.

The dummy values are stored in the *pos* array at index i after initialization (Line 3 in Algorithm 4). Placement of the dummy values is either random within the *pos* array or using a more advanced placement strategy.

The first option is to place the dummy values randomly across the whole *pos* array. In this case, an additional FWVS is required to produce a vector with $\omega + N_{\text{dummy}}$ bits and a Hamming weight of N_{dummy}. All 1 bits in the vector represent positions of the N_{dummy} dummy values inserted in the *pos* array. Another placement strategy is to insert the dummy values at the beginning and end of the *pos* array. In this case, N_1 dummy values are loaded at the beginning and $N_2 = N_{\text{dummy}} - N_1$ values at the end. In this case, the probability for dummy operations varies across indices. A third strategy is to mix the two approaches above and put N_1 dummy operations at the beginning, N_2 at



(a) unprotected (b) $N_{dummy} = 10$

(c) $N_{dummy} = 40$ (d) $N_{dummy} = 75$

Figure 3: TVLA of the unprotected and protected implementation with dummy operations for different numbers N_{dummy}.

the end, and N_3 operations randomly distributed between the real operations. To sample N_1, N_2 and N_3 uniformly under the constraints $N_i \geq 0$ and $N_1 + N_2 + N_3 = N_{\text{dummy}}$, the stick-breaking construction described in [TJ10] can be used.

TVLA In order to evaluate the security gains of the dummy operations, we measure 1000 power traces and compute non-specific t-tests with fixed vs. random inputs. In Figure 3, the resulting t-values are displayed for the unprotected implementation and the protected implementation with different numbers of dummy operations. It shows that the dummy operations effectively reduce the side-channel leakage as the t-values are reduced in the protected cases. This reduction can be explained by the temporal misalignment introduced by the dummy operations. The real operations are shifted randomly in time. Consequently, the input-dependent leakage is distributed across multiple sample points and more traces are required to achieve the same amount of leakage. Additionally, the results show that increasing the number of dummy operations N_{dummy} leads to lower t-values, indicating stronger temporal misalignment. This increased resistance against side-channel attack could be undermined, if an attacker is able to distinguish dummy operations from real operations to fully reverse the temporal misalignment. In the following, we analyze the difference between the power traces of dummy and real operations.

Distinguish Dummy Operations We assume an attacker model in which the adversary has access to a similar device for profiling, where the positions of dummy values are known. During the attack, the dummy positions are unknown and random for each execution. Consequently, only single-trace classification is applicable to identify the dummy operations. The t-test shown in Figure 4, comparing traces with a dummy value at $pos[0]$ to those with a non-dummy value at the same position, reveals significant leakage. The value $pos[0]$

is processed during the last sample points, where the most leakage is seen, thus indicating dummy-dependent power consumption.

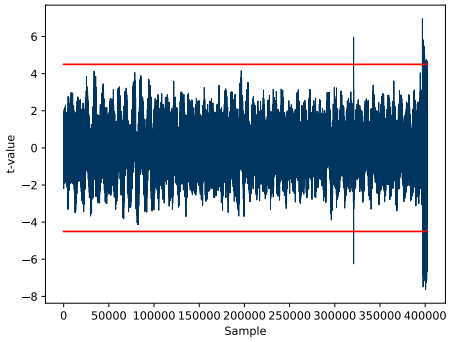


Figure 4: T-test with 2000 traces, comparing traces with dummy and non-dummy values at $pos[0]$.

However, since the dummy positions are random for each execution, the TVLA does not determine whether the dummy operations could be identified in a single-trace analysis. Therefore, we additionally evaluate whether a template attack can distinguish between dummy and real values in a single-trace setting. We build two templates for each index, one for dummy values and one for real values with 1000 measured traces per template for HQC-1. Then, we measure a single trace and try to classify the dummy operations. For the random and mixed placement strategies, this template attack does not perform significantly better than a classifier that only takes the a-priori probability for the different placement strategies into account. Only for the beginning-and-end strategy, it is possible to classify 90% of the values correctly, and therefore this placement strategy is not recommended. For the random and mixed strategies, our results suggest that, in a single-trace scenario, an attacker could not identify all dummy operations and, thus, could not retrieve the complete real array. Thus, the hiding technique cannot be bypassed, so it effectively increases the difficulty of the original distinguishing attack.

Evaluation Results of Dummy Operations in Hardware This increased difficulty can be measured by quantifying the number of traces required for a successful attack compared to an attack on an unprotected implementation. The same evaluation strategy is used, but extended to account for the increased trace length due to dummy operations. Additionally, an a priori classifier is used to determine at which positions a dummy operation is placed and where a real operation is processed, so that only the POIs where the probability for a real operation is larger than 50% are then analyzed. The success rate of the distinguisher is evaluated for different parameter sets using 2,000 calls, and the results are shown in Table 2.

Table 2: Distinguisher success rate for different dummy configurations.

		Random			Beginning-and-end			Mixed		
	N_{traces}	10	50	100	10	50	100	10	50	100
N_{dummy}	10	0.70	1.00	1.00	0.58	0.97	1.00	0.61	1.00	1.00
	40	0.57	0.98	1.00	0.53	0.75	0.86	0.52	0.81	0.89
	75	0.51	0.88	0.97	0.51	0.66	0.73	0.51	0.64	0.72

Recall that, for the unprotected implementation, a success rate of over 99% is achieved

for $N_{\text{traces}} = 1$ and a success rate of 100% for $N_{\text{traces}} = 3$. Activating the dummy operations significantly decreases the success rate and increases the number of traces required for a successful attack with N_{dummy}. For the random placement strategy, achieving a success rate above 99% requires approximately 20 traces for $N_{\text{dummy}} = 10$, 60 traces for $N_{\text{dummy}} = 40$, and more than 100 traces for $N_{\text{dummy}} = 75$, The beginning-and-end strategy performs best in this case, but, as discussed above, it can be easily bypassed. This is why the mixed placement strategy seems to perform best when considering the bypassing method and the averaging effect of several traces of the distinguisher.

These results demonstrate that dummy operations enhance security against the distinguishing attack by reducing the success rate of the distinguisher and thus leading to more iterations and increases runtime. A lower success rate increases the probability of incorrect decisions, which accumulate over multiple attack iterations and increase the number of required distinguisher calls. When the success rate approaches 50%, the distinguisher behaves similarly to random guessing, making the attack impractical. Setting the number of dummy operations and the placement strategy, controls the computational overhead and induced timing noise. Therefore, dummy operations are a good choice for this part of the FWVS in HQC. Complementing them with another side-channel countermeasure, such as masking, would further increase security.

4 Side-Channel Attack on FWVS Conversion

In this section, we present a novel, single-trace SCA on a first-order, masked software implementation, which also leads to full key recovery. We further demonstrate that this leakage would enable the previously discussed distinguishing attack. Our target is the publicly available masked HQC implementation by Goy et al. [GSA+25]. This implementation features a masked FWVS presented in [SVP+24] which builds on a masked Fisher-Yates implementation developed for BIKE [DR24].

4.1 Attack Strategy

Our attack aims at the conversion from support-vector to dense vector at the end of the FWVS. The masked implementation by Goy et al. [GSA+25] does not follow the most recent specification of HQC with respect to FWVS, but utilizes the constant-time Fisher-Yates for support vector sampling during key generation *and* encryption. Recall that the current specification of HQC features rejection sampling to draw the support set during key generation. However, the conversion from support to dense vector is the same for both sampling algorithms. Thus, our attack applies to key generation and encryption (the encryption is used in encapsulation and decapsulation), independent of the specification version.

The conversion from support vector to dense representation in [SVP+24] works in two steps. It transforms a vector of ω indices determining the non-zero bits into a vector of n bits, split into $\lceil n/W \rceil$ words, given a wordsize W. The first step generates, for each element in the support set, an entry in two intermediate vectors: In the $\boldsymbol{index}$ vector, each index from the support vector is translated into an index indicating the corresponding word. In the $\boldsymbol{bit}$ vector, each index from the support set is translated into a word-sized bitmask that only has the corresponding bit set. Algorithm 5 depicts a simplified version of the algorithmic approach used in [SVP+24] to construct the bitmask with a single bit set at a specified index with masking. The output of Algorithm 5 forms one element of $\boldsymbol{bit}$.

In the second step, a loop iterates over the words of the dense vector $\boldsymbol{v}$ and securely applies the bitmasks from $\boldsymbol{bit}$ to the zero-initialized word of $\boldsymbol{v}$ if the current index matches an element from $\boldsymbol{index}$. Algorithm 6 illustrates how [SVP+24] implemented the second step with masking.

Algorithm 5 Single bitmask from index [SVP+24]

Require: $j^{B,v}$, $v \leftarrow \lceil \log_2(W) \rceil$, wordsize W
Ensure: $r^{B,W}$ with only one bit set at position j
1: $r^{B,W} \leftarrow 0^{B,W}$
2: $b^{B,W} \leftarrow 1^{B,W}$
3: **for** $i \leftarrow 0$, $i < W$, $i \leftarrow i + 1$ **do**
4: $\quad c^{B,1} \leftarrow \text{SecCompEq}^{\text{B,v}}(j^{B,v}, i)$
5: $\quad m^{B,W} \leftarrow \text{SecOppos}^{\text{B,W}}(c^{B,1})$ ▷ maps 0 to 00...0 and 1 to 11...1
6: $\quad m^{B,W} \leftarrow \text{Refresh}^{\text{B,W}}(m^{B,W})$
7: $\quad m^{B,W} \leftarrow \text{SecAnd}^{\text{B,W}}(b^{B,W}, m^{B,W})$
8: $\quad r^{B,W} \leftarrow \text{Refresh}^{\text{B,W}}(r^{B,W})$
9: $\quad r^{B,W} \leftarrow \text{SecOr}^{\text{B,W}}(r^{B,W}, m^{B,W})$
10: $\quad b^{B,W} \leftarrow \text{SecShiftLeft}^{\text{B,W}}(b^{B,W}, 1)$

Our SCA recovers information from both, Step 1 and Step 2, to be able to reconstruct the output vector of FWVS. Algorithm 5 and Algorithm 6 both utilize intermediate bitmasks with all bits set to zero or one to perform a conditional select based on Boolean operations. These bitmasks are the value we target in our SCA and are marked in red in the algorithmic descriptions. Distinguishing between two values with extreme distance in Hamming weight (0 versus W, with $W = 32$ for common micro-controllers, including our target-device) is a prime application for SCA.

Recovering the bitmasks for each iteration and each execution of both algorithms directly leads to the sampled vectors. For key generation, two such vectors form the secret key ($\mathbf{x}$, $\mathbf{y}$). For encryption, the knowledge of two vectors $\mathbf{r_2}$ and $\mathbf{e}$ allows to compute the message and thus the shared key in the KEM. Identifying the bitmask set to ones in Algorithm 5 reveals the index within a word for one support set entry. Identifying the bitmasks set to ones in the inner loop in Algorithm 6 further reveals indices within the set of words for the support set.

To recover one output vector of FWVS, W bitmasks have to be recovered for each of the ω executions of Algorithm 5. Further, $\lceil n/W \rceil \times \omega$ bitmasks are required for Algorithm 6. In total, this adds up to $(\lceil n/W \rceil + W) \times \omega$ values. For example, to recover the secret key vector $\mathbf{y}$ in HQC-1 given a wordsize $W = 32$, $(\lceil 17,669/32 \rceil + 32) \times 66 = 38,610$ bitmasks need to be recovered.

In Algorithm 5, three gadgets operate on the targeted bitmasks and are potential points in time for the SCA; the first two gadgets are also used in Algorithm 6:

- `SecOppos`: computes $f(x) = -x \pmod{2^W}$ and is used in the sampler to create a bitmask from single bit: 1 is mapped to 111...1 and 0 to 000...0.
- `SecAnd`: Performs a Boolean `and`. In the context of the sampler, one argument is the mask of zeros or ones.
- `Refresh`: Re-randomizes the shares of the input and is used for the targeted bitmask.

As we show in Section 4.2, `SecOppos` has sufficient leakage such that our SPA can distinguish the bitmasks of a single trace with perfect accuracy.

With perfect classification precision, one can directly recover the sampled vector and thus, compute the secret key. To improve the attack for lower precisions, one can exploit the known frequency of the two distinct bitmask values for FWVS: In Algorithm 5, the comparison $i = j$ only evaluates to true once for the W loop iterations. Thus, m is set to a bitmask of ones for a single iteration, and to zeros for $W - 1$ iterations. Therefore, the attacker can select only the one trace that was assigned to ones with the most confidence.

Algorithm 6 Sparse to dense conversion, Step 2 [SVP+24]

Require: $\boldsymbol{index}^{B,v}$, $\boldsymbol{bit}^{B,W}$, $v \leftarrow \lceil \log_2(n/W) \rceil$, wordsize W
Ensure: $\boldsymbol{v}^{B,W}$ vector in dense representation

1: **for** $i \leftarrow 0$, $i < \lceil n/W \rceil$, $i \leftarrow i+1$ **do**
2: $\quad \boldsymbol{v}[i]^{B,W} \leftarrow 0^{B,W}$
3: $\quad t^{B,W} \leftarrow 0^{B,W}$
4: $\quad$ **for** $j \leftarrow 0$, $j < \omega$, $j \leftarrow j+1$ **do**
5: $\quad\quad c^{B,1} \leftarrow \text{SecCompEq}^{\text{B,v}}(\boldsymbol{index}[j]^{B,v}, i)$
6: $\quad\quad m^{B,W} \leftarrow \text{SecOppos}^{\text{B,W}}(c^{B,1})$ ▷ maps 0 to 00...0 and 1 to 11...1
7: $\quad\quad \boldsymbol{bit}[j]^{B,W} \leftarrow \text{Refresh}^{\text{B,W}}(\boldsymbol{bit}[j]^{B,W})$
8: $\quad\quad m^{B,W} \leftarrow \text{SecAnd}^{\text{B,W}}(\boldsymbol{bit}[j]^{B,W}, m^{B,W})$
9: $\quad\quad t^{B,W} \leftarrow \text{SecOr}^{\text{B,W}}(t^{B,W}, m^{B,W})$
10: $\quad \boldsymbol{v}[i]^{B,W} \leftarrow \text{SecOr}^{\text{B,W}}(\boldsymbol{v}[i]^{B,W}, t^{B,W})$

Furthermore, when targeting the session key of HQC-KEM, respectively the message of HQC-PKE, a multi-trace scenario emerges for the attack we present, because the encryption invoked during the decapsulation can arguably be triggered multiple times with the same ciphertext. A multi-trace SPA can lift good, but not perfect accuracy to practical levels by accumulating class probabilities of multiple traces of the same value.

Note that the leakage of the bitmasks of the FWVS conversion can also be used to build a power side-channel distinguisher for the plaintext-oracle attack by [UXT+22, GHJ+22] which we used in Section 3. Different messages lead to different input randomness for FWVS and thus different output vectors. The ability to recover the output vector implies the ability to distinguish output vectors and therefore to construct the required plaintext-distinguisher for the attack.

4.2 Practical Evaluation

Our practical evaluation is based on a ChipWhisperer CW308 board with a STM32F303 target featuring an ARM Cortex-M4. Traces were taken with a Teledyne Lecroy WavePro 404 HD oscilloscope at 100 MS s^{-1} and an Agilent 33250A waveform generator was used as an external clock set to 10 MHz. The code written by [SVP+24] was taken from the public repository[1] and compiled with `arm-none-eabi-gcc` version 10.3.1.

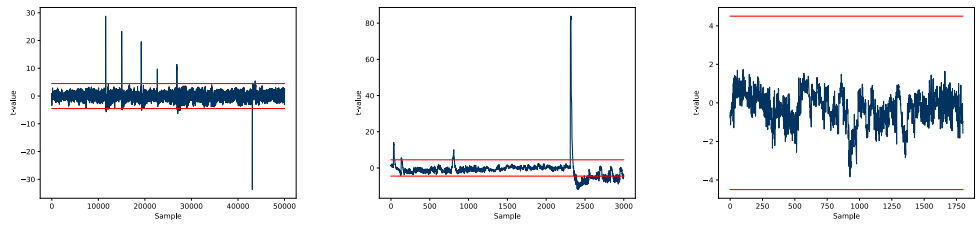
[SVP+24] performed TVLA for their implementation of the Barrett reduction. They observe no leakage for 10,000 traces with optimization flag `-Og`, but leakage with `-O3`. They do not report leakage assessment for other parts of the sampler or lower-level gadgets.

Based on the observations in Section 4, we evaluate three gadgets used in the FWVS conversion which operate on zeros or ones bitmasks with Boolean shared operands. Following the findings of [SVP+24], we performed TVLA with two different compiler settings: `-Og` and `-O3`.

Figure 5 depicts the results of our first-order, fixed-vs-random t-test with 10,000 traces on the three gadgets compiled with `-Og`. For this test, we sampled the inputs from the complete input set that applies to the gadgets: Boolean shared values in $[0, 2^W - 1]$. For `SecOppos` and `SecAnd`, we observe significant indications for leakage, but not for `Refresh`.

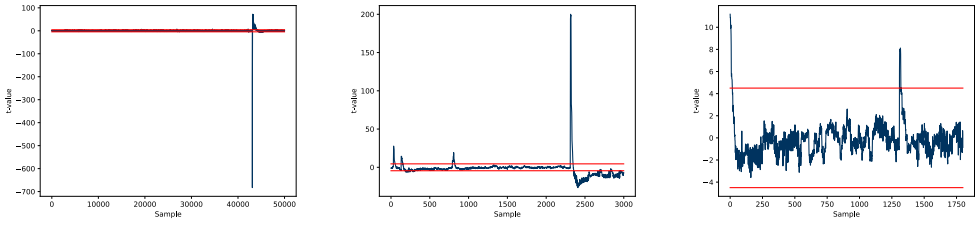
We repeated the TVLA on the three gadgets compiled with `-Og` with inputs limited to the targeted values in the FWVS components: 000...0 and 111...1 as single input for `SecOppos` and `Refresh` and as the second input for `SecAnd`. Interestingly, in this scenario also the t-test on the `Refresh` gadget indicates leakage as shown in Figure 6. The TVLA results for compilation with `-O3` can be found in the appendix Section A.

[1] https://github.com/Tchoupignouf/masked_HQC_implem, commit `31e2254`



(a) `SecOppos` (b) `SecAnd` (c) `Refresh`

Figure 5: First-order, fixed-vs-random t-test on 10,000 traces with inputs sampled from the complete input set applicable to the gadget (`-Og`).



(a) `SecOppos` (b) `SecAnd` (c) `Refresh`

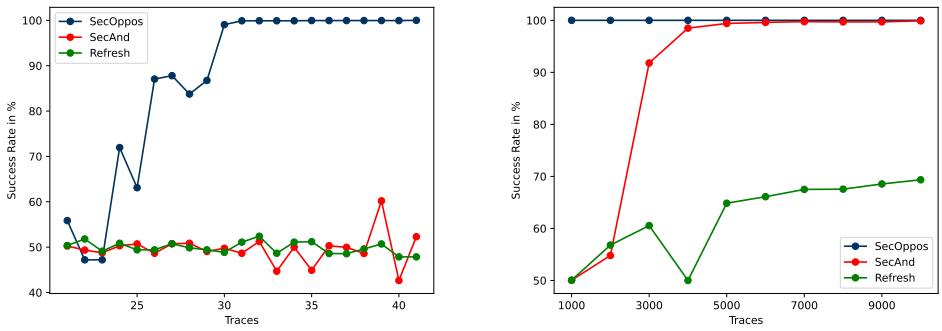
Figure 6: First-order, fixed-vs-random t-test on 10,000 traces with inputs sampled from the input set applicable to the targeted FWVS components (`-Og`).

We performed a template attack with POI selection based on the Sum of Squared pairwise T-differences (SOST) and characterized the traces based on Multivariate Normal Distribution (MVND). For our test setup, the class allocation achieves a precision of 100% targeting the output value of `SecOppos`, a precision of 99.7% targeting the second operand of `SecAnd`, and 76.0% targeting the input of `Refresh` for a single trace. Compiling with `-O3` instead of `-Og` improves the results, leading to a precision of 100% for `SecAnd` and 93.6% for `Refresh`.

Figure 7 plots the progression of the accuracy depending on the number of profiling traces. For `SecOppos`, perfect precision is already achieved using only 40 profiling traces. Targeting `SecAnd`, around 7,000 traces are needed to reach the highest accuracy, and for `Refresh`, we still observe a trend towards more accuracy after 10,000 traces. Note that one FWVS execution already involves enough calls of the respective gadget to record these number of profiling traces.

In Figure 8, we plot the voltage distributions for a selected POI of `SecOppos` for the two input values relevant for FWVS. For this histogram, we sample the same amount of the two different input values. In the context of the FWVS conversion, a zero input appears $\frac{W-1}{1}$ more often than a one. Since the distributions do not overlap, a non-profiled SPA will arguably also be able to distinguish the two values with perfect accuracy and thus allow to execute our attack without template building.

We also performed a multi-trace template attack targeting the `Refresh` gadget as an representative example of a scenario where a single-trace attack alone is not sufficient (76% success rate in this case). After 24 traces of the same input, our template attack always assigns the correct class with 100% accuracy.



(a) (b)

Figure 7: Success rate of the template attack based on the number of profiling traces for the targeted gadgets (`-Og`).

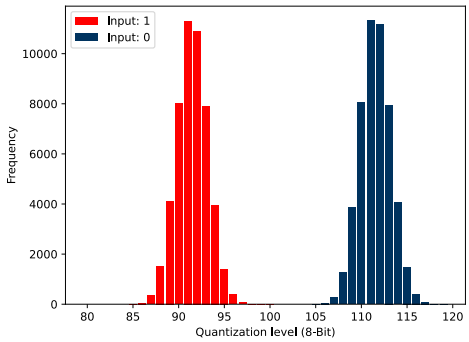


Figure 8: Voltage distributions for a selected POI for `SecOppos` (`-Og`).

4.3 Leakage Attribution

To identify the origin of the significant leakage despite masking, we inspect the object files generated by the compiler. Listing 1 exemplarily depicts a pattern we found, that contributes to the leakage: It constitutes one part of the `SecOppos` gadget and performs a loop over the shares of the input array and copies them to the output array. First, the two `add` instructions calculate the pointers for the input and output array. Then, a share of the input is loaded to register `fp` and immediately stored in the output array. Finally, the loop counter in `r3` is incremented and the branching for the loop is performed. Critical is register `fp`, where the first share loaded in the first loop iteration is overwritten with the second share in the second iteration. This leads to leakage of the Hamming distance of the two shares, which translates to Hamming weight leakage of the unshared secret value.

A common solution to avoid these kind of leakage sources is controlling the register and instruction usage by implementing low-level gadgets in handwritten assembly. After preventing leakage at the architectural level, microarchitectural effects must also be considered [ZMM23].

Listing 1: Leaking instruction pattern from `SecOppos` gadget compiled with `-Og`

```
8003f72:        add.w   r2, r0, r3, lsl #3
8003f76:        add.w   ip, r1, r3, lsl #3
8003f7a:        ldrd    fp, ip, [ip]
8003f7e:        strd    fp, ip, [r2]
8003f82:        adds    r3, #1
8003f84:        uxtb    r3, r3
8003f86:        cmp     r3, #1
8003f88:        bls.n   8003f72
```

4.4 Countermeasures

Alongside side-channel mitigating measures based on instruction and register selection (see Section 4.3), we advocate for countermeasures at the algorithmic level. For Algorithm 5 we propose a variant that avoids the ones or zeros bitmasks, following the practical guideline (iii) for side-channel secure implementations by Hesse et al. [HFG+26]. Furthermore, we reduce the number of costly non-linear operations by using a `xor` instead of a Boolean `or`. Our solution is depicted in Algorithm 7. Note that the secure `shift` and the secure `xor` used in Algorithm 7 are linear operations.

Algorithm 7 Single bitmask from index

Require: $j^{B,v}$, $v \leftarrow \lceil \log_2(W) \rceil$, wordsize W
Ensure: $r^{B,W}$ with only one bit set at position j
1: $r^{B,W} \leftarrow 0^{B,W}$
2: **for** $i \leftarrow 0$, $i < W$, $i \leftarrow i + 1$ **do**
3: $\quad c^{B,1} \leftarrow \text{SecCompEq}^{\text{B,v}}(j^{B,v}, i)$
4: $\quad c^{B,W} \leftarrow \text{SecShiftLeft}^{\text{B,W}}(c^{B,1}, i)$
5: $\quad r^{B,W} \leftarrow \text{Refresh}^{\text{B,W}}(r^{B,W})$
6: $\quad r^{B,W} \leftarrow \text{SecXor}^{\text{B,W}}(r^{B,W}, c^{B,W})$

Bitslicing To avoid the ones or zeros bitmasks in the sparse to dense conversion, one can apply bitslicing. Bitslicing is as effective as a lightsaber, as it can not only reduce leakage, but also significantly improve performance for FWVS [KLS+25]. By processing multiple values in parallel, bitslicing increases the switching noise and thus decreases the signal-to-noise ratio. Furthermore, a bitsliced implementation processes the targeted bitmasks bitwise, and thus does not inhibit the significant Hamming-weight differences. In Algorithm 8, we show a bitsliced conversion, where input and output are in the bitsliced domain. Bitsliced gadgets for `compare`, `conditional move`, and `or`, which are required for this algorithm have already been realized in [KLRBG23]. Bitslicing can also be applied to the support-vector generation: Krausz et al. [KLRBG23] present bitsliced implementations for constant-time Fisher-Yates and rejection sampling.

Shuffling Both Goy et al. [GSA+25] and Krausz et al. [KLS+25] mention shuffling as a promising way to harden FWVS against SCA. Unlike support-vector generation, where the computational steps must be performed in a specific order to ensure functional correctness, the conversion can be straightforwardly protected by shuffling. In the sparse-to-dense conversion in Algorithm 6, both the inner and outer loops can be shuffled with negligible performance overhead and also Algorithm 5 can be adapted for shuffling.

Algorithm 8 Bitsliced sparse to dense conversion, Step 2

Require: $\boldsymbol{index}^{B,v}$, $\boldsymbol{bit}^{B,W}$, $v \leftarrow \lceil \log_2(n/W) \rceil$, wordsize W
Ensure: $\boldsymbol{v}^{B,W}$ vector in dense representation
1: **for** $i \leftarrow 0$, $i < \lceil n/W \rceil$, $i \leftarrow i + W$ **do**
2: $\quad \boldsymbol{v}[i : i + W - 1]^{B,W} \leftarrow [0, 0, \ldots, 0]^{B,W}$
3: $\quad \boldsymbol{t}^{B,W} \leftarrow [0, 0, \ldots, 0]^{B,W}$
4: $\quad \boldsymbol{i} \leftarrow [i, i + 1, \ldots, i + W - 1]$
5: $\quad$ **for** $j \leftarrow 0$, $j < \omega$, $j \leftarrow j + 1$ **do**
6: $\quad\quad \boldsymbol{c}^{B,1} \leftarrow \mathrm{SecCompEq}_{\mathrm{bs}}^{\mathrm{B,v}}(\boldsymbol{index}[j]^{B,v}, \boldsymbol{i})$
7: $\quad\quad \boldsymbol{t}^{B,W} \leftarrow \mathrm{SecCmov}_{\mathrm{bs}}^{\mathrm{B,W}}(\boldsymbol{bit}[j]^{B,W}, \boldsymbol{c}^{B,1})$
8: $\quad\quad \boldsymbol{v}[i]^{B,W} \leftarrow \mathrm{SecOr}_{\mathrm{bs}}^{\mathrm{B,W}}(\boldsymbol{v}[i]^{B,W}, \boldsymbol{t}^{B,W})$

The attack presented in this section would be completely prevented by shuffling, as it requires knowledge of the currently processed index. If the index is unknown to the adversary, the recoverable information cannot be used. Circumventing shuffling by recovering the index value is unlikely with a single side-channel trace, especially when assuming Hamming-weight leakage.

Dummy Operations Hiding by adding dummy items into the arrays, as we discussed in detail in Section 3, can also be utilized in a masked implementation and in the support conversion. When dummy operations are already used in the support vector generation, keeping them for the conversion can be beneficial. However, as discussed above, the security improvement of dummy operations is proportional to their number, and their overhead scales proportionally. Therefore, one must carefully consider whether the additional noise bath inserted by the dummy operations is required or whether shuffling alone is sufficient.

5 Conclusion

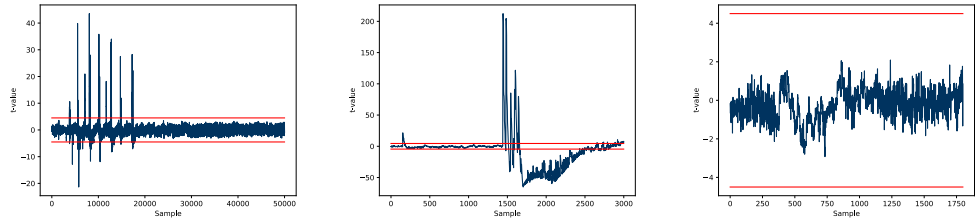
In this paper, we present two practical attacks on HQC's fixed-weight vector sampling. The first attack demonstrates that even a small amount of leakage can enable a distinguishing attack that leads to complete key recovery. Dummy operations can be an effective countermeasure in this case, as they make identical vectors appear different.

Our second attack exploits leakage of an implementation of the masked conversion process. It demonstrates that masking FWVS is not trivial, and that masking should be supplemented with noise generation and hiding countermeasures. Dummy operations introduce an overhead proportional to the number of added dummies. In contrast, the overhead introduced by shuffling is negligible since the computation steps are merely rearranged and thus the preferable hiding measure for the conversion. Where possible, bitslicing should be applied, as this increases switching noise and can speed up computations.

Acknowledgments

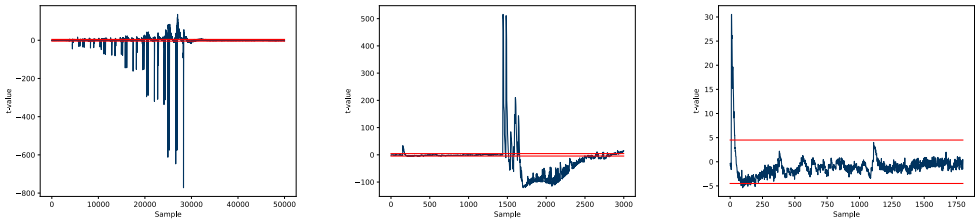
The work described in this paper has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and by the German Federal Ministry of Research, Technology and Space (BMFTR) through the projects EASEPROFIT (16KIS2126), POST (16ME2157) and DI-OWAS (16ME0967).

A TVLA Results for `-O3`



(a) `SecOppos` (b) `SecAnd` (c) `Refresh`

Figure 9: First-order, fixed-vs-random t-test on 10,000 traces with inputs sampled from the complete input set applicable to the gadget (`-O3`).



(a) `SecOppos` (b) `SecAnd` (c) `Refresh`

Figure 10: First-order, fixed-vs-random t-test on 10,000 traces with inputs sampled from the input set applicable to the targeted FWVS components (`-O3`).

References

[AAB+22] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[BDH+19] Ciprian Băetu, F. Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 747–776. Springer, Cham, May 2019.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Berlin, Heidelberg, May 2014.

[DR24] Loïc Demange and Mélissa Rossi. A provably masked implementation of BIKE key encapsulation mechanism. *CiC*, 1(1):23, 2024.

[DXN+24] Sanjay Deshpande, Chuanqi Xu, Mamuri Nawan, Kashif Nawaz, and Jakub Szefer. Fast and efficient hardware implementation of HQC. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *SAC 2023*, volume 14201 of *LNCS*, pages 297–321. Springer, Cham, August 2024.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999.

[GHJ+22] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR TCHES*, 2022(3):223–263, 2022.

[GLG22] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. A new key recovery side-channel attack on HQC with chosen ciphertext. In Jung Hee Cheon and Thomas Johansson, editors, *PQCrypto 2022*, pages 353–371. Springer, Cham, September 2022.

[GSA+25] Guillaume Goy, Maxime Spyropoulos, Nicolas Aragon, Philippe Gaborit, Renaud Pacalet, Fabrice Perion, Laurent Sauvage, and David Vigilant. Side-channel sensitivity analysis on HQC: Towards a fully masked implementation. Cryptology ePrint Archive, Report 2025/1344, 2025.

[HFG+26] Dina Hesse, Jakob Feldtkeller, Tim Güneysu, Julius Hermelink, Georg Land, Markus Krausz, and Jan Richter-Brockmann. t-probing (in-)security: Pitfalls on noise assumptions. *IACR TCHES*, 2026(2):1–27, 2026.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Cham, November 2017.

[HNPS24] Julius Hermelink, Kai-Chun Ning, Richard Petri, and Emanuele Strieder. The insecurity of masked comparisons: SCAs on ML-KEM's FO-transform. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2430–2444. ACM Press, October 2024.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Berlin, Heidelberg, August 1999.

[KLRBG23] Markus Krausz, Georg Land, Jan Richter-Brockmann, and Tim Güneysu. A holistic approach towards side-channel secure fixed-weight polynomial sampling. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 94–124. Springer, Cham, May 2023.

[KLS+25] Markus Krausz, Georg Land, Florian Stolz, Jan Richter-Brockmann, and Tim Güneysu. To extend or not to extend: Agile masking instructions for PQC. In Yongdae Kim, Atsuko Miyaji, and Mehdi Tibouchi, editors, *CANS 25*, volume 16351 of *LNCS*, pages 70–92. Springer, Singapore, November 2025.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Berlin, Heidelberg, August 1996.

[MNMD25] Nathan Maillet, Cyrius Nugier, Vincent Migliore, and Jean-Christophe Deneuville. Key recovery from side-channel power analysis attacks on non-SIMD HQC decryption. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 70–102. Springer, Cham, August 2025.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2007.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Berlin, Heidelberg, May 2013.

[PRJ+25] Thales B. Paiva, Prasanna Ravi, Dirmanto Jap, Shivam Bhasin, Sayan Das, and Anupam Chattopadhyay. Et tu, brute? Side-channel assisted chosen ciphertext attacks using valid ciphertexts on HQC KEM. In Ruben Niederhagen and Markku-Juhani O. Saarinen, editors, *Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Part II*, pages 294–321. Springer, Cham, April 2025.

[Sen21] Nicolas Sendrier. Secure sampling of constant-weight words – application to BIKE. Cryptology ePrint Archive, Report 2021/1631, 2021.

[SHR+22] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. A power side-channel attack on the reed-muller reed-solomon version of the HQC cryptosystem. In Jung Hee Cheon and Thomas Johansson, editors, *PQCrypto 2022*, pages 327–352. Springer, Cham, September 2022.

[SRSWZ20] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. A power side-channel attack on the cca2-secure hqc kem. In *International Conference on Smart Card Research and Advanced Applications*, pages 119–134. Springer, 2020.

[SVP+24] Maxime Spyropoulos, David Vigilant, Fabrice Perion, Renaud Pacalet, and Laurent Sauvage. Masked vector sampling for HQC. Cryptology ePrint Archive, Report 2024/1106, 2024.

[TJ10] Yee Whye Teh and Michael I Jordan. Hierarchical bayesian nonparametric models with applications. *Bayesian nonparametrics*, 1:158–207, 2010.

[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 TCHES*, 2022(1):296–322, 2022.

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 740–757. Springer, Berlin, Heidelberg, December 2012.

[WBD24] Ruize Wang, Martin Brisfors, and Elena Dubrova. A side-channel attack on a higher-order masked CRYSTALS-Kyber implementation. In Christina Pöpper and Lejla Batina, editors, *ACNS 2024, Part III*, volume 14585 of *LNCS*, pages 301–324. Springer, Cham, March 2024.

[ZMM23] Jannik Zeitschner, Nicolai Müller, and Amir Moradi. PROLEAD_SW - probing-based software leakage detection for ARM binaries. Cryptology ePrint Archive, Report 2023/034, 2023.