Revisiting Simple Power Analysis of Polynomial Multiplication in the HQC Implementation

Trevor Yap[0000−0001−8651−574X], Shivam Bhasin[0000−0002−6903−5127], and Léo Weissbart[0000−0003−0288−9686]

Nanyang Technological University, Singapore
{trevor.yap, sbhasin, leojoseph.weissbart}@ntu.edu.sg

Abstract. The Hamming Quasi-Cyclic (HQC) scheme is a post-quantum key encapsulation mechanism recently selected for standardization by NIST, making the security of its implementations a critical concern. In this work, we revisit the side-channel resistance of the latest HQC reference implementation, with a focus on its polynomial multiplication routine used in decryption. While prior work demonstrated a simple power analysis (SPA) attack against a lookup-table-based multiplication, the updated implementation replaces this with a bitwise schoolbook multiplication.

Despite these modifications, we show that the updated function `schoolbook_mul` still shows clear, visually distinguishable variations in power consumption that depend on secret data. Using these patterns, an attacker can recover individual bits of the secret polynomial, demonstrating that SPA remains effective against the updated design. To address this vulnerability, we analyze potential countermeasures and propose a zero-cost mitigation based on swapping operand roles so that the mask computation depends only on public data. This eliminates direct leakage of secret-dependent control flow while preserving performance.

Keywords: Simple Power Analysis · HQC · Polynomial Multiplication · Post-Quantum Cryptography.

1 Introduction

With quantum computers becoming a reality, the threat has grown more significant than ever before. Many previously secure cryptographic primitives, such as RSA, can be easily broken using algorithms designed for quantum computers, such as Shor's algorithm [16]. In response, the National Institute of Standards and Technology (NIST) initiated a standardization process for post-quantum cryptography (PQC) in 2016. This effort aims to evaluate and standardize quantum-resistant public-key cryptographic algorithms. On March 11, 2025, Hamming quasi-cyclic (HQC) was selected as a new alternative in the fourth round of the project. In addition to the algorithms themselves, their implementations are also required to be evaluated during the selection process.

In this work, we investigate the most recent implementation of HQC available on its website [3]. After extracting the archive, the implementation under

study is located in the file `hqc-submission_2025-08-22`, which corresponds to the latest version released following its announcement as the fifth algorithm selected for PQC standardization. We also investigate the polynomial multilication algorithm, namely the function `schoolbook_mul`.

Previously, in [17], a Simple Power Analysis (SPA) targeting the polynomial multiplication was proposed. The authors analyzed the implementation submitted in round 4, specifically targeting the function `base_mul`. This function relies on a lookup table to mitigate cache-timing side-channel leakage during the multiplication process. Their attack succeeds because the table access pattern depends on secret-dependent values, allowing information leakage through power consumption traces.

As a result, the paper subsequently proposed an alternative implementation that eliminates the use of lookup tables (see Listing 2 in [17]). Since no lookup table is used, the previously described SPA technique is no longer directly applicable. Following this suggestion, the HQC team revised their implementation based on the proposed approach, with several modifications in the final design. In the reference code, this revised approach can be found in the function `schoolbook_mul`. However, we show that SPA remains feasible even against this updated implementation. Next, we discuss which countermeasures are viable for protecting our attack. We present a no-cost countermeasure to protect against our attack.

Paper Organization First, we review prior work on side-channel analysis of HQC in Section 2. Section 3 introduces the necessary HQC preliminaries for this paper, and Section 4 describes the targeted HQC implementation. Subsequently, we present our SPA results in Section 5, followed by the proposed countermeasure in Section 6. Finally, we conclude the paper in Section 7.

2 Prior Works

Schamberger et al. [14] were the first to perform a side-channel attack on HQC. They targeted a version of HQC using a BCH decoder and later adapted their attack to the Reed-Muller–Reed-Solomon (RMRS) variant in [13]. Their approach constructs an oracle that detects whether the Reed-Muller (RM) decoding output is zero. Goy et al. [4] improved this attack by training a linear discriminant classifier. Subsequent works further reduced the number of required traces by introducing more advanced techniques, such as creating offline templates from the RM code [2] or applying Soft Analytic Side-Channel Analysis (SASCA) [5, 1].

However, these works operate in a chosen-ciphertext setting, where selecting a malicious ciphertext may cause decryption failures and could potentially be detected. Paiva et al. [10] instead consider the valid-ciphertext setting and construct a distinguisher based on the `find_peak` and `expand_and_sum` functions.

Most of these approaches exploit power consumption or electromagnetic (EM) emanations as their primary leakage sources to build distinguishers, while

other works focus on timing and cache-based side channels [11, 6, 7, 15]. However, these attacks generally target either the decoder or the sampler to recover the secret.

A DPA-style attack exploiting the Toeplitz structure of the multiplication was proposed in [8]. However, it requires around 20 traces and relies on a chosen-ciphertext setting for the attack to be feasible. In contrast, SPA-based approaches targeting polynomial multiplication, as proposed in [17], can potentially succeed with a single trace. Profiling attacks on polynomial multiplication have also been explored [12], though these focus on earlier submitted implementations. In this work, we investigate whether the updated submission remains susceptible to SPA attacks. Especially, since in the previous SPA [17], the attack succeeds because a look-up table is used based on 4-bits of the secret, enabling information leakage through power consumption traces. However, the updated implementation no longer provides direct table access, as suggested in [17], and therefore the previously described SPA attack no longer applies. This highlights the need to reassess SPA vulnerability in the updated design, which we address in the remainder of this work.

3 Preliminaries of HQC

In this section, we will present the essential parts of HQC for our discussion. HQC is an IND-CCA2 secure PQC scheme for Key Encapsulation Mechanism (KEM). HQC relies on the Quasi-Cyclic Syndome Decoding problem as its hardness problem to be secure. To built the KEM, HQC develops its own Public-Key Encryption (PKE). In this paper, we will focus on the decryption algorithm of HQC.

Firstly, we shall denote the notation requires here. We denote bold lowercases (e.g. $\boldsymbol{x}$) as vectors and bold uppercase (e.g. $\boldsymbol{X}$) as matrix. Any binary vector $\boldsymbol{a} = (a_0, \ldots, a_{n-1}) \in \mathbb{F}_2^n$ can also be considered as a polynomial $a_0 + a_1 X^1 + \cdots + a_{n-1} X^{n-1}$ in the ring $\mathcal{R} \in \mathbb{F}_2[X]/(X^n - 1)$.

For any two vectors $u, v \in \mathcal{R}$, the multiplication $\boldsymbol{w} = \boldsymbol{u} \cdot \boldsymbol{v}$ in the ring $\mathcal{R}$ is defined as:

$$w_\ell = \sum_{i+j=\ell} u_i v_j.$$

We can rewrite the multiplication using

$$\boldsymbol{rot}(\boldsymbol{x}) = \begin{pmatrix} x_0 & x_{n-1} & \ldots & x_1 \\ x_1 & x_0 & \ldots & x_2 \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1} & x_{n-2} & \ldots & x_0 \end{pmatrix}$$

and we have

$$\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u} \times rot(\boldsymbol{v})^T = \boldsymbol{v} \times rot(\boldsymbol{u})^T = \boldsymbol{v} \cdot \boldsymbol{u}$$

However, when implementing the multiplication, one uses the either the Karatsuba or Toom-Cook algorithm [17].

The HQC scheme make use of a public code $\mathcal{C} \subseteq \mathbb{F}_2^n$ with dimension k. Current versions of HQC uses the duplicated first-order Reed-Muller concatenated with a shorten Reed-Solomon code, we simply call it Reed-Solomon Reed-Muller Code (RSRM) [9]. We denote $\xleftarrow{\$} \mathcal{X}$ as sampling uniformly from the set $\mathcal{X}$, $\xleftarrow{\$,w} \mathcal{R}$ as sampling uniformly from the polynomial ring $\mathcal{R}$ with the constraint that the sampled element has a Hamming weight w. The sampling can be made deterministic by using a pseudo-random generator with a random seed θ.

We shall briefly describe HQC PKE. HQC PKE has three algorithms: key generation (Algorithm 1), encryption (Algorithm 2) and decryption (Algorithm 3).

For the key generation algorithm, the public vector $\boldsymbol{h} \in \mathcal{R}$ are uniformly sampled and the secret variables $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{R}$ is also uniformly sampled with both their Hamming weight restricted to w. Then, we set the secret key as $sk = (\boldsymbol{x}, \boldsymbol{y})$ and public key as $pk = (\boldsymbol{h}, \boldsymbol{s})$ with $\boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y}$. Therefore, this suggest that if either $\boldsymbol{x}$ or $\boldsymbol{y}$ is recovered, the full secret key is recovered.

Algorithm 1. Key Generation

Input: hqc parameters
Output: public key pk, secret key sk

$\boldsymbol{h} \xleftarrow{\$} \mathcal{R}$
$(\boldsymbol{x}, \boldsymbol{y}) \xleftarrow{\$,w} \mathcal{R}^2$
$\boldsymbol{s} \leftarrow \boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y}$
$pk = (\boldsymbol{h}, \boldsymbol{s})$
$sk = (\boldsymbol{x}, \boldsymbol{y})$
return pk, sk.

Next, during encryption $\boldsymbol{e}, \boldsymbol{r_1}, \boldsymbol{r_2}$ are uniformly sampled from $\mathcal{R}$ with some restriction on their Hamming weights. These are initialized by a seed θ. The ciphertext is then create via $\boldsymbol{c} = (\boldsymbol{u}, \boldsymbol{v})$ where

$$\boldsymbol{u} = \boldsymbol{r_1} + \boldsymbol{h} \cdot \boldsymbol{r_1}$$
$$\boldsymbol{v} = \mathit{Encode}(\boldsymbol{m}) + \boldsymbol{s} \cdot \boldsymbol{r_2} + \boldsymbol{e}$$

with $\mathit{Encode}(\boldsymbol{m}) = \boldsymbol{mG}$ encode the message $\boldsymbol{m}$ using the RSRM Code with $\boldsymbol{G}$ being its the generator matrix.

Algorithm 2. Encrypt

Input: message $\boldsymbol{m}$, public key pk, randomness seed θ
Output: ciphertext c

$\boldsymbol{e} \xleftarrow{\$,w_e,\theta} \mathcal{R}$
$(\boldsymbol{r_1}, \boldsymbol{r_2}) \xleftarrow{\$,w_r,\theta} \mathcal{R}^2$
$\boldsymbol{u} \leftarrow \boldsymbol{r_1} + \boldsymbol{h} \cdot \boldsymbol{r_1}$
$\boldsymbol{v} \leftarrow \mathit{Encode}(\boldsymbol{m}) + \boldsymbol{s} \cdot \boldsymbol{r_2} + \boldsymbol{e}$
return $\boldsymbol{c} = (\boldsymbol{u}, \boldsymbol{v})$.

Now, during the decryption algorithm, it decode the $\boldsymbol{v} - \boldsymbol{u} \cdot \boldsymbol{v}$. Since

$$\boldsymbol{v} - \boldsymbol{u} \cdot \boldsymbol{v} = Encode(\boldsymbol{m}) + \boldsymbol{x} \cdot \boldsymbol{r_2} - \boldsymbol{r_1} \cdot \boldsymbol{y} + \boldsymbol{e}$$

Now, suppose the error term $\boldsymbol{e'} = \boldsymbol{x} \cdot \boldsymbol{r_2} - \boldsymbol{r_1} \cdot \boldsymbol{y} + \boldsymbol{e}$ is sufficiently small, then the RSRM code $\mathcal{C}$ can correct it and recover the message $\boldsymbol{m}$. In this paper,

Algorithm 3. Decrypt

Input: secret key $sk = (\boldsymbol{x}, \boldsymbol{y})$, ciphertext $\boldsymbol{c} = (\boldsymbol{u}, \boldsymbol{v})$
Output: message $\boldsymbol{m}$
1: $\boldsymbol{v'} \leftarrow \boldsymbol{v} - \boldsymbol{u} \cdot \boldsymbol{y}$
2: $\boldsymbol{m} \leftarrow Decode(\boldsymbol{v'})$
3: **return** $\boldsymbol{m}$.

we shall mainly focus on the multiplication of $\boldsymbol{u} \cdot \boldsymbol{y}$ in the decryption (green in Algorithm 3).

4 HQC Implementation

Following the announcement that HQC has been selected as the fifth algorithm for PQC standardization, the HQC team updated their submission on their website. The updated files can be found on their GitLab repository[1] or in their archived release (after downloading and unzipping) under the filename `hqc-submission_2025-08-22` [3]. We observe that the multiplication $\boldsymbol{u} \cdot \boldsymbol{y}$ is implemented using a Karatsuba algorithm followed by a modular reduction. The Karatsuba algorithm recursively calls itself until the input size reaches 16 words (each 64 bits), at which point it invokes the function `schoolbook_mul`. By following the inputs of the recursive calls within the Karatsuba algorithm, we observe that `schoolbook_mul` takes a portion of the secret y as input `a`, and a portion of the ciphertext u as input `b`. Therefore, we focus on extracting the bits of `a`.

The `schoolbook_mul` implements a schoolbook-style multiplication of two large integers represented as arrays of 64-bit words, with $n \leq 16$ corresponding to maximum of 1024-bit inputs. It iterates over each word of the first operand `a` and, within each word, over all 64 bits. For every bit of `a[i]`, a mask is generated to determine whether that bit is set, allowing the computation to proceed without conditional branches. If the bit is 1, a shifted version of the second operand `b` is XORed into the result array at the appropriate position; if the bit is 0, no change occurs. The shifting is handled across word boundaries to correctly align contributions in the 2048-bit result. Overall, this corresponds to a bitwise schoolbook multiplication where partial products are accumulated using XOR.

Since the mask is derived from the bits of `a`, which correspond directly to the secret $\boldsymbol{y}$, one would expect observable differences in the power consumption

[1] https://gitlab.com/pqc-hqc/hqc/

trace depending on whether the mask bit is 0 or 1. Consequently, differences within the trace can be exploited to recover the secret $\boldsymbol{y}$. We will show this in Section 5 that an observable differences can be seen.

```
static void schoolbook_mul(uint64_t *r, const uint64_t *a, const uint64_t
    *b, size_t n) {
    memset(r, 0, 2 * n * sizeof(uint64_t));
    for (size_t i = 0; i < n; i++) {
        uint64_t ai = a[i];
        for (int bit = 0; bit < 64; bit++) {
            uint64_t mask = -((ai >> bit) & 1ULL);
            size_t base = i;
            int sh = bit;
            int inv = 64 - sh;
            if (sh == 0) {
                for (size_t j = 0; j < n; j++) {
                    r[base+j]^=b[j] & mask;
                }
            } else {
                for (size_t j = 0; j < n; j++) {
                    r[base + j] ^= (b[j] << sh) & mask;
                    r[base + j + 1] ^= (b[j] >> inv) & mask;
                }
            }
        }
    }
}
```

Fig. 1: C code snippet of the `schoolbook_mul` operation used by the multiplication in HQC. The target operations and variables for SCA are highlighted in red and in blue.

5 Simple Power Analysis

5.1 Experimental Set-up

We measure the power consumption of the target device while executing the `schoolbook_mul` function on a 32-bit STM32F303 ARM Cortex-M4 using the ChipWhisperer-Lite board. Figure 2 shows the ChipWhisperer-Lite board used in this paper. The power traces are captured using the board's built-in oscilloscope, which is limited to approximately 25,000 sample points per trace.

Due to this restriction on trace length, each captured trace reveals only about 5 bits of the secret variable. This limitation can be mitigated by using an external oscilloscope or alternative platforms capable of recording longer traces, such as the ChipWhisperer-Husky or ChipWhisperer-Pro (CW1200).

5.2 Attack Methodology

Now we describe our SPA attack. The attack is purely visual in nature. Figure 3 presents the power consumption traces corresponding to the computation of the

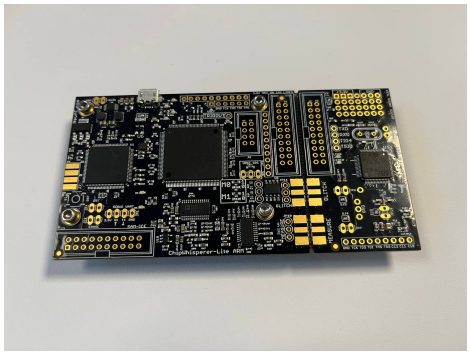
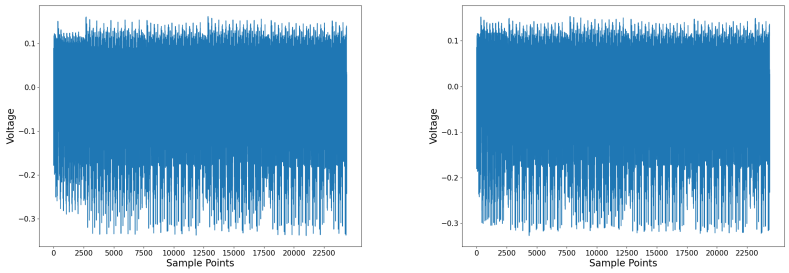


Fig. 2: ChipwWhisperer-Lite Board with 32-bit STM32F303.

first five bits of `a[0]`. These traces are extracted from the execution of lines $5-20$ in Figure 1, restricted to `bit` $= 0, \ldots, 4$. Suppose `a[i]` $= a_{63}a_{62}\ldots a_0$., then Figure 3a depicts the trace obtained when $a_4a_3a_2a_1a_0 = 0b01010$ while Figure 3b shows the trace for $a_4a_3a_2a_1a_0 = 0b01010$.

From Figure 3, we identify clearly distinguishable temporal segments within the traces. Each segment corresponds to the processing of a single bit of `a[0]`, as determined by the iteration over `bit` in the implementation. Specifically, the interval from sample points 0 to 2400 corresponds to the processing of a_0, 2400 to 7500 corresponds to a_1, 7500 to 12500 corresponds to a_2, 12500 to 17500 corresponds to a_3, and 17500 to 23000 corresponds to a_4. These segment boundaries are approximate and serve as empirical estimates derived from visual inspection of the traces.

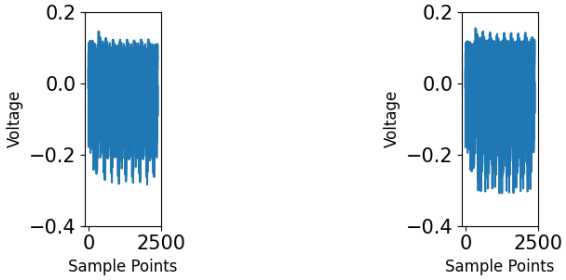


(a) $0b01010$ (b) $0b00101$

Fig. 3: Power consumption traces corresponding to the computation of the first 5 bits of `a[0]`.

In particular, lines $11-13$ (highlighted in red in Figure 1) correspond to the computation of a_0. For this case, we observe two distinct patterns in the

power consumption trace, depending on the value of the processed bit. This is the leftmost segment of the trace from sample point 0 to 2500. When $a_0 = 0$, the mask evaluates to `0x0000000000000000`, which results in a trace segment with lower power consumption compared to the case where $a_0 = 1$, for which the mask evaluates to `0xFFFFFFFFFFFFFFFF`. Figure 4 shows a side-by-side comparison of the trace segments corresponding to the two possible values of a_0. This is line



(a) $a_0 = 0$ (b) $a_0 = 1$

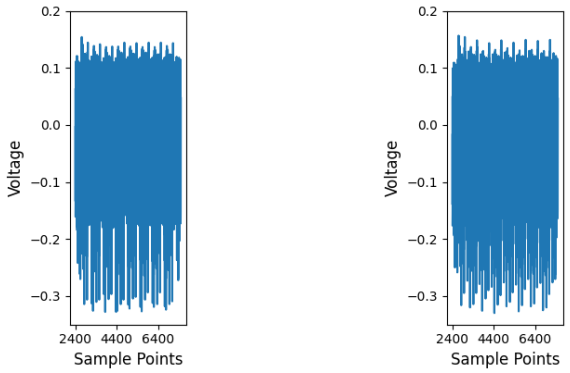
Fig. 4: Trace Segment for different values of a_0 (sample point 0 to 2500)

$16 - 19$. A similar pattern can also be observed for `bit` > 0. lines $15 - 18$ (highlighted in blue in Figure 1) corresponds to the computation of a_0.

For simplicity, we restrict our discussion to the case `bit` $= 1$, as the remaining cases exhibit analogous behaviour. In this setting, we again observe two distinct patterns in the power consumption trace, depending on the value of the processed bit. Similarly to a_0, when $a_1 = 0$, the mask evaluates to `0x0000000000000000`, resulting in a less dense power consumption segment compared to when $a_1 = 1$, which the mask evaluates to `0xFFFFFFFFFFFFFFFF`, producing a visibly different trace pattern. Figure 5 presents a side-by-side comparison of the trace segments for both possible values of a_1. We evaluate all $2^4 = 16$ possible key-dependent patterns, corresponding to the complete set of 4-bit binary values from $0b0000$ to $0b1111$. Each pattern induces a distinct power consumption trace, reflecting how different key bit combinations influence the underlying computation. As shown in Figure 6, the traces are organized according to their binary key values, allowing a systematic comparison across all possible cases. This exhaustive enumeration enables us to capture the full range of observable leakage behaviors. We observe distinguishable side-channel patterns between bit values equal to 0 and 1, consistent with the discussion above. These results demonstrate the effectiveness of the proposed attack.

5.3 Attack Evaluation

For HQC, the private key $\boldsymbol{y}$ has 17669, 35851, or 57637 bits, depending on the security level, and is stored as 277, 561, or 901 words of 64 bits. The



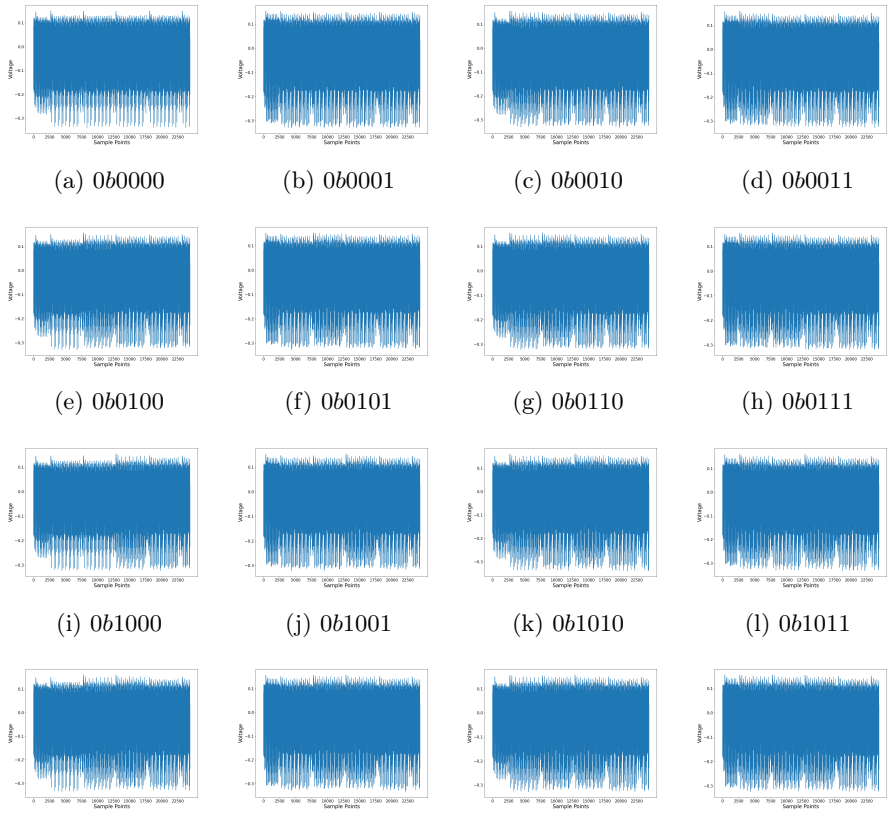
(a) $a_1 = 0$ (b) $a_1 = 1$

Fig. 5: Trace Segment for different values of a_1 (sample point 2400 to 7000)

function `karatsuba_mul` implements the Karatsuba multiplication algorithm with a threshold of 16 on the block size n; below this threshold, it invoke `schoolbook_mul`. Algorithm 7 presents the fucntion `karatsuba_mul` used in the reference implementation.

Taking HQC-128 as an example, `karatsuba_mul` is initially invoked with $n = 277$ words. Let a and b be the input to the `karatsuba_mul`. We further split the inputs into two half $\mathrm{a} = [\mathrm{a}_l, \mathrm{a}_h]$ and $\mathrm{b} = [\mathrm{b}_l, \mathrm{b}_h]$. For $n > 16$, the function applies the Karatsuba recursion by setting $m = \lfloor n/2 \rfloor$ and performing three recursive calls: one on the lower halves ($z_0 = \mathrm{a}_l * \mathrm{b}_l$, with $n_0 = m$ words), one on the upper halves ($z_2 = \mathrm{a}_h * \mathrm{b}_h$, with $n_1 = n - m$ words), and one on the XOR of the halves ($z_{\mathrm{mid}} = (\mathrm{a}_h + \mathrm{a}_l) * (\mathrm{b}_h + \mathrm{b}_l)$, also with n_1 words). Figure 8 illustrates the resulting recursion tree. The recursion terminates after five levels, producing $3^5 = 243$ invocations of `schoolbook_mul`. Note that the leaves are reached with 8 or 9 words as the threshold value of in the implementation is fixed to be 16. As such, sizes of 17 and 18 still exceed the threshold and are therefore split once more, into 8/9 and 9/9 respectively. For HQC-128 the leaf sizes are 37×8 and 206×9 words; for HQC-192 the tree has depth 6 with leaves of 8 or 9 words, and for HQC-256 it has depth 6 with leaves of 14 or 15 words.

To recover the key, not all of these calls need to be observed. The z_{mid} branch multiplies the XOR of two halves of $\boldsymbol{y}$, whereas the z_0 and z_2 branches pass a contiguous slice of $\boldsymbol{y}$ through unchanged. Following only z_0 and z_2 edges from the root therefore yields $2^5 = 32$ leaves whose `a` operands are part of $\boldsymbol{y}$ and which together cover all 277 words. Recovering the full secret thus requires observing only 32 of the 243 invocations of `schoolbook_mul` (resp. 64 of the 729 invocations for HQC-192 and HQC-256), each leaking $8 \times 64 = 512$ or $9 \times 64 = 576$ bits of $\boldsymbol{y}$ by the analysis of Section 5.2. As a result, for HQC-128, the attacker will have to apply the aforementioned SPA methodology of Section 5.2 to the purple leaf node while ignoring the computations corresponding to the grey leaf node. A similar tree structure can be constructed for HQC-192 and HQC-256 as well.



(a) $0b0000$ (b) $0b0001$ (c) $0b0010$ (d) $0b0011$

(e) $0b0100$ (f) $0b0101$ (g) $0b0110$ (h) $0b0111$

(i) $0b1000$ (j) $0b1001$ (k) $0b1010$ (l) $0b1011$

(m) $0b1100$ (n) $0b1101$ (o) $0b1110$ (p) $0b1111$

Fig. 6: Power consumption traces for all 2^4 key-dependent executions, covering all key values from $0b0000$ to $0b1111$. Each subplot corresponds to one key value.

```c
static void karatsuba_mul(uint64_t *r, const uint64_t *a, const
 uint64_t *b, size_t n, uint64_t *tmp_buffer) {
if (n <= KARATSUBA_THRESHOLD) {
    schoolbook_mul(r, a, b, n);
    return;
}

size_t m = n >> 1;
size_t n0 = m;
size_t n1 = n - m;

/* take successive chunks of tmp_buffer for each intermediate result
 */
uint64_t *z0 = tmp_buffer;   /* a_l*b_l, size 2*n words */
uint64_t *z2 = z0 + 2 * n;   /* a_h*b_h, size 2*n words */
uint64_t *zmid = z2 + 2 * n; /* (a_h+a_l)*(b_h+b_l), size 2*n words */

/* ta and tb hold the sums of low and high halves: */
/* ta[i] = a0[i] XOR a1[i], tb[i] = b0[i] XOR b1[i] for i < n1 */
uint64_t *ta = zmid + 2 * n;
uint64_t *tb = ta + n;

/* buffer for child recursions */
uint64_t *child_buffer = tmp_buffer + 8 * n;

/* 1) a_l*b_l */
karatsuba_mul(z0, a, b, n0, child_buffer);

/* 2) a_h*b_h */
karatsuba_mul(z2, a + m, b + m, n1, child_buffer);

/* 3) (a_h+a_l)*(b_h+b_l) */
for (size_t i = 0; i < n1; i++) {
    uint64_t loa = (i < n0 ? a[i] : 0);
    uint64_t lob = (i < n0 ? b[i] : 0);
    ta[i] = loa ^ a[m + i];
    tb[i] = lob ^ b[m + i];
}
karatsuba_mul(zmid, ta, tb, n1, child_buffer);

/* 4) assemble into r */
memset(r, 0, 2 * n * sizeof(uint64_t));
for (size_t i = 0; i < 2 * n0; i++) r[i] ^= z0[i];
for (size_t i = 0; i < 2 * n1; i++) r[2 * m + i] ^= z2[i];
for (size_t i = 0; i < 2 * n1; i++) {
    uint64_t z0i = (i < 2 * n0 ? z0[i] : 0);
    uint64_t z2i = (i < 2 * n1 ? z2[i] : 0);
    uint64_t mid = zmid[i] ^ z0i ^ z2i;
    r[m + i] ^= mid;
}
}
```

Fig. 7: C code snippet of the `karatsuba_mul` operation used by the multiplication in HQC.

We measure the power consumption at the start of the polynomial multiplication phase when decrypting the ciphertext using the private key. With the available trace length, we can observe sample points related to the 5 bits of the private key. This can be easily overcome by using either an external osciloscope or capturing devices with streaming mode like CW1200 ChipWhisperer-Pro or ChipWhisperer-Husky that could capture long traces.

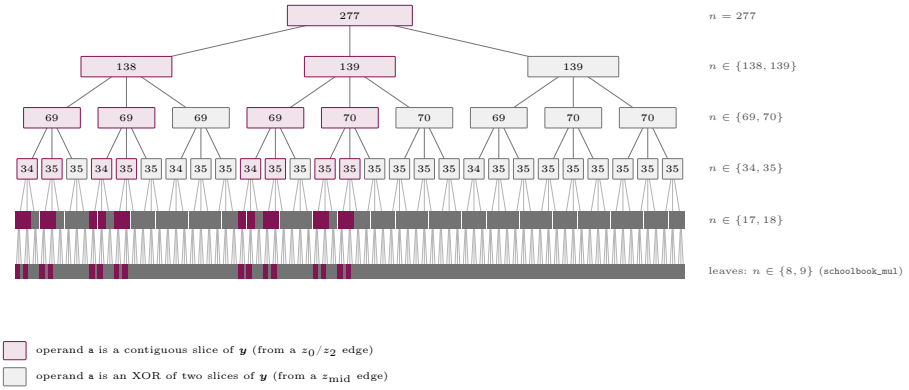


Fig. 8: Full recursion tree of `karatsuba_mul` for HQC-128, where the secret $\boldsymbol{y}$ occupies 277 words of 64 bits. Every one of the 364 nodes across the 5 recursion levels is shown; from level 4 onward nodes are drawn as small unlabelled marks since their sizes only alternate between the two values given at the right of each row. Pink nodes carry a contiguous slice of $\boldsymbol{y}$ (reached only via z_0/z_2 edges); gray nodes carry an XOR of two slices (reached via at least one z_{mid} edge). The bottom row is the 243 actual calls to `schoolbook_mul`, each of which is a point where the attacker can capture a power trace as in Section 5.

In terms of the time required for full key recovery. For our setup, processing 5 bits takes approximately 0.73 seconds. Extrapolating this, a single 64-bit word requires about 9.34 seconds. For HQC-128, considering only the purple block and assuming the grey block does not introduce additional waiting, the total runtime is approximately $32 \times 9 \times 9.34$ seconds, which is about 44.83 minutes. However, if the attacker must also wait for each grey block to complete, the attack only need to consider the first 126 blocks (each with 8 or 9 words of 64 bits). Hence, the total time increases to roughly $126 \times 9 \times 9.34$ seconds, corresponding to approximately 3 hours for full key recovery.

6 Proposed Countermeasures

In this section, we propose countermeasures against our SPA attack. First, we consider the masking scheme proposed by [17]. The corresponding pseudocode is given below:

$$
\begin{aligned}
a_0 &\xleftarrow{\$} (\mathbb{F}_2^{64})^{16} \\
a_1 &= a \oplus a_0 \\
r_0 &= \texttt{schoolbook_mul}(a_1, b) \\
r_1 &= \texttt{schoolbook_mul}(a_0, b) \\
r &= r_0 \oplus r_1
\end{aligned}
$$

However, as noted by the authors, an attacker can still successfully recover the secret key when both multiplications are targeted. The same observation applies in our SPA setting. The masking does not provide effective protection against this type of analysis.

This naturally raises the question of how such an attack could be mitigated. Rather than relying on more complex countermeasures, a simple and effective solution is to change the operand ordering in `schoolbook_mul`. In particular, input `a` should not contain any secret-dependent data. By swapping the inputs so that the ciphertext $\boldsymbol{u}$ is used as input `a` and the secret $\boldsymbol{y}$ is used as input `b`, the `mask` computation in `schoolbook_mul` no longer depends on the secret $\boldsymbol{y}$, but instead on the public variable $\boldsymbol{u}$. As a result, even if the same leakage patterns described in Section 5 are observed, they correspond only to bits of $\boldsymbol{u}$, which is publicly known. Consequently, no sensitive information is revealed, and the attack yields no exploitable knowledge about the secret key. We further emphasize that no additional experiments are required to validate this, since the same leakage patterns described in Section 5 will still be observable, but the SPA is no longer able to extract any meaningful information from them.

However, this implementation may still be subsceptible to profiling attack as describes in [12]. As this is not the focus of our work, we shall leave this for future work.

7 Conclusion

In this work, we presented a simple power analysis (SPA) on the latest HQC implementation, focusing on the `schoolbook_mul` function used within the Karatsuba multiplication. Although the update removed earlier lookup-table-based constructions and replaced them with a more schoolbook bitwise multiplication strategy, we show that the implementation still exhibits visually distinguishing power consumption patterns based on the secret, enabling the recovery of key information from direct inspection of power traces.

To mitigate this vulnerability, we proposed a simple, zero-cost countermeasure based on swapping operand roles so that the masking operation depends solely on public data. This modification preserves performance while eliminating the direct dependence of power variations on secret values. However, while effective against the SPA technique presented in this work, it does not fully address more advanced side-channel methodologies, such as profiling-based attacks, which may still exploit residual leakage behavior. As a result, our findings highlight that further care is required when designing and validating constant-time implementations for post-quantum cryptographic schemes like HQC, and suggest that side-channel resistance must be evaluated.

References

1. Baïsse, C., Moran, A., Goy, G., Maillard, J., Aragon, N., Gaborit, P., Lecomte, M., Loiseau, A.: Secret and shared keys recovery on hamming quasi-cyclic with SASCA.

Des. Codes Cryptogr. **93**(6), 2137–2157 (2025). https://doi.org/10.1007/S10623-025-01575-2, https://doi.org/10.1007/s10623-025-01575-2
2. Dong, H., Guo, Q.: OT-PCA: New Key-Recovery Plaintext-Checking Oracle Based Side-Channel Attacks on HQC with Offline Templates. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2025**(1), 251–274 (2025). https://doi.org/10.46586/TCHES.V2025.I1.251-274, https://doi.org/10.46586/tches.v2025.i1.251-274
3. Gaborit, P., Aguilar Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Persichetti, E., Zémor, G., Bos, J., Dion, A., Lacan, J., Robert, J.M., Véron, P., Barreto, P.L., Ghosh, S., Gueron, S., Güneysu, T., Misoczki, R., Richter-Brokmann, J., Sendrier, N., Tillich, J.P., Vasseur, V.: Hqc website. https://pqc-hqc.org/resources.html, accessed: 2026-04-22
4. Goy, G., Loiseau, A., Gaborit, P.: A new key recovery side-channel attack on HQC with chosen ciphertext. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13512, pp. 353–371. Springer (2022). https://doi.org/10.1007/978-3-031-17234-2_17, https://doi.org/10.1007/978-3-031-17234-2_17
5. Goy, G., Maillard, J., Gaborit, P., Loiseau, A.: Single trace HQC shared key recovery with SASCA. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2024**(2), 64–87 (2024). https://doi.org/10.46586/TCHES.V2024.I2.64-87, https://doi.org/10.46586/tches.v2024.i2.64-87
6. Guo, Q., Hlauschek, C., Johansson, T., Lahr, N., Nilsson, A., Schröder, R.L.: Don't Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(3), 223–263 (2022). https://doi.org/10.46586/TCHES.V2022.I3.223-263, https://doi.org/10.46586/tches.v2022.i3.223-263
7. Guo, Q., Nabokov, D., Nilsson, A., Johansson, T.: SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-quantum Encryption Schemes. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14441, pp. 203–236. Springer (2023). https://doi.org/10.1007/978-981-99-8730-6_7, https://doi.org/10.1007/978-981-99-8730-6_7
8. Huang, Z., Wang, W., Zhou, X., Yu, Y.: DPA-Style Attacks on HQC. IACR Cryptol. ePrint Arch. **2025**, 1966 (2025), https://eprint.iacr.org/2025/1966
9. Melchor, C.A., Aragon, N., Deneuville, J., Gaborit, P., Lacan, J., Zémor, G.: Efficient error-correcting codes for the HQC post-quantum cryptosystem. Des. Codes Cryptogr. **92**(12), 4511–4530 (2024). https://doi.org/10.1007/S10623-024-01507-6, https://doi.org/10.1007/s10623-024-01507-6
10. Paiva, T.B., Ravi, P., Jap, D., Bhasin, S., Das, S., Chattopadhyay, A.: Et tu, brute? side-channel assisted chosen ciphertext attacks using valid ciphertexts on HQC KEM. In: Niederhagen, R., Saarinen, M.O. (eds.) Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Taipei, Taiwan, April 8-10, 2025, Proceedings, Part II. Lecture Notes in Computer Science, vol. 15578, pp. 294–321. Springer (2025). https://doi.org/10.1007/978-3-031-86602-9_11, https://doi.org/10.1007/978-3-031-86602-9_11
11. Paiva, T.B., Terada, R.: A timing attack on the HQC encryption scheme. In: Selected Areas in Cryptography–SAC 2019: 26th International Conference, Water-

loo, ON, Canada, August 12–16, 2019, Revised Selected Papers 26. pp. 551–573. Springer (2020)
12. Rabas, T., Bucek, J., Grosso, V., Zenknerová, K., Lórencz, R.: Profiling Side-Channel Attack on HQC Polynomial Multiplication Using Machine Learning Methods. In: Rivain, M., Sasdrich, P. (eds.) Constructive Approaches for Security Analysis and Design of Embedded Systems - First International Conference, CASCADE 2025, Saint-Etienne, France, April 2-4, 2025, Proceedings. pp. 580–602. Lecture Notes in Computer Science, Springer (2025). https://doi.org/10.1007/978-3-032-01405-4_24, https://doi.org/10.1007/978-3-032-01405-4_24
13. Schamberger, T., Holzbaur, L., Renner, J., Wachter-Zeh, A., Sigl, G.: A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13512, pp. 327–352. Springer (2022). https://doi.org/10.1007/978-3-031-17234-2_16, https://doi.org/10.1007/978-3-031-17234-2_16
14. Schamberger, T., Renner, J., Sigl, G., Wachter-Zeh, A.: A Power Side-Channel Attack on the CCA2-Secure HQC KEM. In: Liardet, P., Mentens, N. (eds.) Smart Card Research and Advanced Applications - 19th International Conference, CARDIS 2020, Virtual Event, November 18-19, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12609, pp. 119–134. Springer (2020). https://doi.org/10.1007/978-3-030-68487-7_8, https://doi.org/10.1007/978-3-030-68487-7_8
15. Schröder, R.L., Gast, S., Guo, Q.: Divide and Surrender: Exploiting Variable Division Instruction Timing in HQC Key Recovery Attacks. In: Balzarotti, D., Xu, W. (eds.) 33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024. USENIX Association (2024), https://www.usenix.org/conference/usenixsecurity24/presentation/schr%C3%B6der
16. Shor, P.W.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, November 20-22, 1994. pp. 124–134. IEEE Computer Society (1994). https://doi.org/10.1109/SFCS.1994.365700, https://doi.org/10.1109/SFCS.1994.365700
17. Velek, P., Rabas, T., Bucek, J.: Simple Power Analysis of Polynomial Multiplication in HQC. CoRR **abs/2601.07634** (2026). https://doi.org/10.48550/ARXIV.2601.07634, https://doi.org/10.48550/arXiv.2601.07634