Masked Vector Sampling for HQC

Maxime Spyropoulos[1,2], David Vigilant[1], Fabrice Perion[1], Renaud Pacalet[2] and Laurent Sauvage[2]
[1]*Thales, Meudon, France*
[2]*LTCI, Télécom Paris, Institut Polytechnique de Paris, France*

Keywords: HQC, Masking, Side-Channel Attack, Post-Quantum Cryptography.

Abstract: Anticipating the advent of large quantum computers, NIST started a worldwide competition in 2016 aiming to define the next cryptographic standards. HQC is one of these post-quantum schemes selected for standardization. In 2022, Guo *et al.* introduced a timing attack that exploited a weakness in HQC rejection sampling function to recover its secret key in 866,000 calls to an oracle. The authors of HQC updated its specification by applying an algorithm to sample vectors in constant time. A masked implementation of this function was later proposed for BIKE but it is not directly applicable to HQC. In this paper we propose a specification-compliant masked version of the HQC vector sampling which relies, to our knowledge, on the first masked implementation of the Barrett reduction.

1 INTRODUCTION

Post-Quantum Cryptography (PQC) gained interest when NIST, anticipating that the current public-key cryptosystems might be broken by quantum computers by 2030, launched its competition. In 2024 the competition reached an important milestone with the publication of three standards (nis, 2024) derived from three selected algorithms, out of 69 initially submitted. An extra fourth round took place and ended in March 2025 with the selection of HQC (nis, 2025). NIST also posted a call for proposals for extra short signature schemes (nis, 2022) that would ideally not rely on the problems of structured lattices.

In the meantime regarding the quantum threat, progresses are still ongoing on both technology and algorithms. Oded Regev published in 2023 an algorithm to factor an n-bits integer using $O(n^{3/2})$ quantum gates instead of Shor's original $O(n^2)$ (Regev, 2023). The same year, IBM presented its 1000+ qbits quantum computer and aims to reach 4000+ qbits by year 2025 (ibm, 2023).

HQC (Melchor et al., 2018) is a code-based Key-Encapsulation Mechanism (KEM), used to securely share a symmetric key between two parties. Its security relies on the hardness of random linear code decoding and is proved to be Indistinguishable under Adaptive Chosen Cipher Attack (IND-CCA2). It has reduced public key and ciphertext sizes, thanks to its quasi-cyclic structure. Its fast key generation and decapsulation made it "one of the two best candidates of the fourth round" (Alagic et al., 2022).

The capacity to develop side-channel resistant and efficient implementations of the algorithm is an explicit and important criterion for selection (nis, 2016). Side-Channel Attacks (SCA) have been first introduced by Kocher in 1996 (Kocher, 1996). They take advantage of data-dependent physical leakage (power consumption, execution time, ...) emanating from an implementation of a cryptographic algorithm. If a leak is correlated to operations involving a secret, an attacker can exploit it to recover information about the secret without breaking any underlying mathematical hard problem. The security evaluation of the scheme in the "real world" setup also raises questions about performance. Indeed, hardening an implementation against SCA may hinder its performance and competitiveness. HQC has already been thoroughly vetted against SCA, either against Timing Attack (Paiva and Terada, 2020; Wafo-Tapa et al., 2019; Guo et al., 2022) or Template Attack (Goy et al., 2022; Goy et al., 2024; Baïsse et al., 2024).

In (Guo et al., 2022) the execution time of the rejection sampling function of HQC was exploited to recover the secret key. The HQC team updated the specification to patch this vulnerability with a countermeasure designed by Sendrier (Sendrier, 2021) to sample vectors in constant time. In 2024 De-

Spyropoulos, M., Vigilant, D., Perion, F., Pacalet, R. and Sauvage, L.
Masked Vector Sampling for HQC.
DOI: 10.5220/0013637400003979
In *Proceedings of the 22nd International Conference on Security and Cryptography (SECRYPT 2025)*, pages 750-758
ISBN: 978-989-758-760-3; ISSN: 2184-7711

mange and Rossi (Demange and Rossi, 2024) provided the first complete high-order masked implementation of BIKE, another code-based algorithm, including a masked version of Sendrier's countermeasure. Although the new masked vector sampling is well suited for BIKE, it is not directly applicable to HQC, which specifications differ.

Contributions. This paper presents a masked specification-compliant vector generation function for HQC that relies on the first masked Barrett reduction to our knowledge. We also give a theoretical security proof backed by a practical evaluation of our implementation on a STM32 board. This practical evaluation raises a warning about our secure implementation and the one proposed by Demange and Rossi, showing that compilation optimizations should be handled carefully.

Organization. The remaining of the paper is organized as follows. Section 2 describes HQC, the masking and what motivated our approach. In Section 3 we adapt the BIKE secure implementation and propose our masked version of the HQC vector sampling, with security proof. Section 4 summarizes the practical evaluations of our implementation on an STM32 device. Finally, Section 5 concludes the paper.

2 BACKGROUND

2.1 HQC

HQC is a post-quantum scheme based on the theory of quasi-cyclic codes, where the secret key is generated independently from the code, such that the security reduction is independent from the decoding algorithm used for decryption. Our work refers to the specification of February 2025.

The authors use the Hofheinz-Hövelmanns-Kiltz transformation (a variant of the Fujisaki-Okamoto transformation that handles decryption failures) to turn the IND-CPA secure Public Key Encryption scheme (PKE) into an IND-CCA2 KEM.

HQC.PKE. The PKE version of HQC consists of three algorithms. k is the length of the shared key (128, 256 or 512 bits), it is a parameter set at initialization. Algorithm 8 generates public key pk and secret key sk given seed $\mathcal{R}$. The weights ω are parameters that depend on k. Algorithm 9 encrypts message $\mathbf{m}$ using pk and seed θ. $\mathcal{C}$ is the HQC public concatenated code. Finally, Algorithm 10 shows the decryption of ciphertext $\mathbf{c}$ knowing sk. The secret key is $(\mathbf{x}, \mathbf{y}, \sigma)$ although only $\mathbf{y}$ is needed for decryption.

HQC.KEM. The KEM version of HQC relies on three hash functions $(\mathcal{G}, \mathcal{H}, \mathcal{K})$, all based on SHAKE256, and consists of 2 algorithms. The sender generate a random symmetric key K with Algorithm 11 and exchange it as a ciphertext. The receiver retrieves the symmetric key with Algorithm 12.

2.2 Guo *et al.* Timing Attack

HQC June 2021 specification suffered from a timing flaw in its encryption function, which relies on the sampling of three fixed weights random vectors. The support of each vector was computed from random bytes generated by an eXtendable Output Function (XOF) named `seedexpander`. In most cases there was no collision between the indexes drawn and `seedexpander` was called only 3 times, once per vector. However, if at least one collision occurred, a second call to `seedexpander` was made.

This difference in the number of calls was noticeable in timing and the authors used this distinguisher to perform a Timing Attack and recover the secret key in $\approx \mathbf{8.7} \times \mathbf{10^5}$ decapsulations (Guo et al., 2022).

2.3 Sendrier's Countermeasure

In 2021, Sendrier proposed an algorithm to sample vectors in constant time in order to patch this vulnerability in both BIKE and HQC rejection sampling functions (Sendrier, 2021). The idea is to always ask for the same amount of pseudo randomness. In the event of a collision between two indexes, the index that was drawn is replaced by the current index of the loop iteration (see Algorithm 1). This is made possible by the introduction of a small bias that has no significant impact on the distribution (Sendrier, 2021).

In Algorithm 1, `compare` is a constant-time function returning `0xF...F` if the two input values are equal, else `0x0...0`. Following recommendations of Guo *et al.*, the HQC team added this countermeasure in their 2023 update of the scheme.

2.4 Masking the Vector Sampling

Masking. The principle of masking is to add randomness to any sensitive variable in the algorithm. A common masking scheme is the *boolean masking* (Chari et al., 1999; Goubin and Patarin, 1999) where any sensitive variable x is split into $d+1$ boolean shares, d of which are random, such that $x = x_0 \oplus \ldots \oplus x_d$. d is the masking order. During

Algorithm 1: Sendrier's constant time vector sampling.

Data: $s, seed, n$
Result: $sup[0], \ldots, sup[s-1]$
1 $prng \leftarrow$ prng_init($seed$);
2 **for** $i = 0$ ***to*** $s-1$ **do**
3 $\quad sup[i] \leftarrow i + \text{rand}(prng, n-i)$;
$\quad$ /* $sup[i] \in [i, n[$ */
4 **end**
5 **for** $i = s-1$ ***downto*** 0 **do**
6 $\quad found \leftarrow 0$; /* collision flag */
7 $\quad$ **for** $j = i+1$ ***to*** $s-1$ **do**
8 $\quad\quad found \leftarrow found \vee \text{compare}(sup[i], sup[j])$;
9 $\quad$ **end**
10 $\quad sup[i] \leftarrow (i \wedge found) \oplus (sup[i] \wedge \neg found)$;
11 **end**

computations the shares are processed such that any combination of d intermediate variables is independent of any sensitive variable. The observations of an attacker with access to up to d intermediate variables is thus independent of any secret. The algorithm is said d-probing secure.

Security Proof. To prove the security of an algorithm, the easiest way is to prove the security property of elementary functions, called *gadgets*. To achieve overall security one can follow the composition property outlined in (Barthe et al., 2016, Proposition 4):

An algorithm P is t-NI provided all its gadgets are t-NI, and all masked variables are used at most once as argument of a gadget call other than `refresh`.

t-NI is also introduced in (Barthe et al., 2016):

Definition 1. A gadget is t-non-interfering (t-NI) iff any set of at most t observations can be perfectly simulated from at most t shares of each input.

Remark 1. t-NI implies t-probing secure.

Remark 2. Any linear operation in binary finite field $\mathbb{F}_2$ is a t-NI gadget, as long as it is applied share-wise.

The `refresh` function was introduced in (Coron, 2013) to increase the overall randomness of an algorithm by re-randomizing the shares encoding a secret variable. Informally, Proposition 4 states that a masked input value has to be refreshed before being used as input in another gadget.

Masked Implementation. In 2024, Demange and Rossi proposed a fully masked and provably secure implementation of BIKE (Demange and Rossi, 2024) featuring a masked version of Sendrier's algorithm, named `SecFisherYates`. It is proved t-NI and the authors provided a C implementation publicly available on GitHub (dem, 2024).

Algorithm 2: Demange-Rossi masked SecFisherYates.

Data: $s \in \mathbb{N}$, $n \in \mathbb{N}$
Result: $[\![\mathbf{r}]\!] \in \mathbb{Z}_n^s$ a randomly generated vector without repeated values
1 **for** $i = s-1$ ***downto*** 0 **do**
2 $\quad [\![\mathbf{r}_i]\!] \leftarrow \texttt{sec}_{rand}(n-i)$;
3 $\quad [\![i]\!] \leftarrow \texttt{refresh}([\![i]\!])$;
4 $\quad [\![\mathbf{r}_i]\!] \leftarrow \texttt{sec}_{+}([\![\mathbf{r}_i]\!], [\![i]\!])$;
5 $\quad$ **for** $j = i+1$ ***to*** $s-1$ **do**
6 $\quad\quad [\![\mathbf{r}_j]\!] \leftarrow \texttt{refresh}([\![\mathbf{r}_j]\!])$;
7 $\quad\quad [\![b]\!] \leftarrow \texttt{sec}_{=}([\![\mathbf{r}_i]\!], [\![\mathbf{r}_j]\!])$;
8 $\quad\quad [\![\mathbf{r}_i]\!] \leftarrow \texttt{refresh}([\![\mathbf{r}_i]\!])$;
9 $\quad\quad [\![i]\!] \leftarrow \texttt{refresh}([\![i]\!])$;
10 $\quad\quad [\![\mathbf{r}_i]\!] \leftarrow \texttt{sec}_{if}([\![i]\!], [\![\mathbf{r}_i]\!], [\![b]\!])$;
11 $\quad$ **end**
12 **end**

3 PROPOSED COUNTERMEASURE

BIKE and HQC differ in the way they draw a random number in interval $\mathbb{Z}_n$. Given a p bits random number a, BIKE multiplies it by n and shifts p bits to the right (Algorithm 3, Line 4), whereas HQC returns the remainder of a modulo n (Algorithm 4, Line 3).

Algorithm 3: BIKE vector sampling.

Data: $seed, len, wt$
Result: $wlist$, a list of wt distinct elements of $\{0, \ldots, len-1\}$.
1 $wlist \leftarrow ()$; /* empty list */
2 $s_0, \ldots, s_{wt-1} \leftarrow$ SHAKE256-Stream($seed, 32 \cdot wt$) ;
$\quad$ /* parse as a sequence of wt non negative 32-bits integers */
3 **for** $i = wt-1$ ***downto*** 0 **do**
4 $\quad pos \leftarrow i + \lfloor (wt-i)s_i / 2^{32} \rfloor$;
5 $\quad wlist \leftarrow wlist, (pos \in wlist)\ ?\ i : pos$;
6 **end**

Algorithm 4: HQC vector sampling.

Data: $n, w, seed$
Result: w distinct elements of $\{0, \ldots, n\}$
1 $prng \leftarrow$ prng_init($seed$);
2 **for** $i = w-1$ ***downto*** 0 **do**
3 $\quad l \leftarrow i + (\text{rand}(prng) \bmod (n-i))$;
4 $\quad pos[i] \leftarrow (l \in \{pos[j], i < j < t)\ ?\ i : l$;
5 **end**
6 **return** $pos[0], \ldots, pos[w-1]$

This difference means that the `SecFisherYates` algorithm cannot be directly transposed to an HQC implementation, as it would not follow the specifications. We must thus design a side-channel resistant function that computes the remainder of a boolean masked value a modulo a public value n. Our design does not use division or modulo instructions because on some architectures their execution time depends on the numerator which, in our case, is a secret variable; a variation in execution time could leak information about the secret. Moreover, they are not directly applicable to boolean masking. We solve this problem with a masked Barrett reduction. Non-masked Barrett reduction was actually implemented in September 2023 in the PQClean version of HQC before being added to the HQC specification in February 2024.

3.1 Barrett Reduction

The Barrett reduction (Barrett, 1987) efficiently computes the remainder of an integer division in constant-time. To compute $x \bmod n$ one can use $r = x - \lfloor x/n \rfloor \times n$. Instead of using a division, Barrett approximates the quotient with an integer m such that $\frac{m}{2^p} \approx \frac{1}{n}$. We usually take $m = \lfloor \frac{2^p}{n} \rfloor$.

Remark 3. In our case, $p = 32$, because the function rand of HQC (Algorithm 4, Line 3) outputs pseudo-random 32-bits unsigned integers.

Since the quotient $\frac{m}{2^p}$ is only guaranteed to be less or equal $\frac{1}{n}$, a final subtraction is sometimes required. The main advantages are that variable m can be pre-computed, and dividing by 2^p comes down to a shift p bits to the right, which is virtually free.

Algorithm 5: Barrett reduction.

Data: a, n, p and m s.t. $m = \lfloor \frac{2^p}{n} \rfloor$
Result: $r = a \bmod n$
1 $q \leftarrow (a \times m) \gg p$;
2 $r \leftarrow a - q \times n$;
3 **if** $r \geq n$ **then**
4 $\quad r \leftarrow r - n$;
5 **end**

3.2 Masked Implementation

Our fully masked implementation of the Barrett reduction, uses gadgets introduced in (Demange and Rossi, 2024). It is constant-time by design, provably secure and works with any masking degree.

In the following, $[\![\mathbf{x}]\!]$ denotes the array containing the boolean shares of masked variable $\mathbf{x}$. All binary secure operations for which it makes sense come in two flavours: $\mathtt{sec}_{op}([\![A]\!], [\![B]\!])$ (both operands masked), $\mathtt{sec}_{op}([\![A]\!], B)$ (left operand masked, right operand public and unmasked). On top of the available gadgets we developed a new one, $\mathtt{sec\_}([\![a]\!], [\![b]\!])$, to compute the subtraction $[\![a]\!] - [\![b]\!]$ of 2 masked variables (see Algorithm 13 in appendix).

Algorithm. To preserve the constant-time property, we always compute the conditional subtraction. We first subtract n from the computed value $a\_qn = a - q \times n$), and store the result in A (Line 7). There are 2 possibilities: either A is negative and a_qn is the correct result or A is positive and the correct result. To evaluate the sign of A we shift it by 31 bits to the right and store the result in z; if A is negative then z is equal to 1, else A is positive and z is equal to 0.

Algorithm 6: Masked Barrett reduction.

Data: $[\![a]\!], m, n$
Result: $[\![r]\!] = [\![a]\!] \bmod n$
1 $[\![q]\!] \leftarrow \mathtt{sec}_{\times}([\![a]\!], m)$;
2 $[\![q]\!] \leftarrow [\![q]\!] \gg 32$; `/* Sharewise shift */`
3 $[\![qn]\!] \leftarrow \mathtt{sec}_{\times}([\![q]\!], n)$;
4 $[\![a]\!] \leftarrow \mathtt{refresh}([\![a]\!])$;
5 $[\![a\_qn]\!] \leftarrow \mathtt{sec\_}([\![a]\!], [\![qn]\!])$;
6 $minus\_n \leftarrow 2^{32} - n$;
7 $[\![A]\!] \leftarrow \mathtt{sec}_{+}([\![a\_qn]\!], minus\_n)$;
8 $[\![z]\!] \leftarrow [\![A]\!] \gg 31$;
9 $[\![A]\!] \leftarrow \mathtt{refresh}([\![A]\!])$;
10 $[\![a\_qn]\!] \leftarrow \mathtt{refresh}([\![a\_qn]\!])$;
11 $[\![r]\!] \leftarrow \mathtt{sec}_{if}([\![a\_qn]\!], [\![A]\!], [\![z]\!])$;
`/*` $[\![z]\!]$ `?` $[\![a\_qn]\!]$ `:` $[\![A]\!]$ `*/`

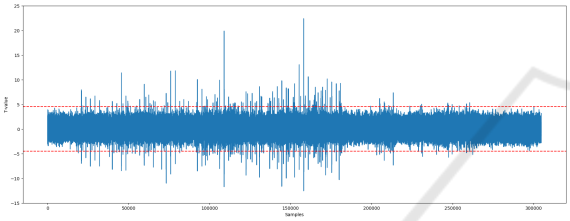
Theorem 1. *Masked Barrett reduction is t-NI secure*

Proof. m and n are public values, we do not need to mask them. as it operates independently on each share, the shift operation (Lines 2, 8) is t-NI. All used gadgets are t-NI. The only masked variables that are used twice without modification are a (Lines 1, 5), A (Lines 8, 11) and a_qn (Lines 7, 11). They are all refreshed (Lines 4, 9, 10) before their re-use. □

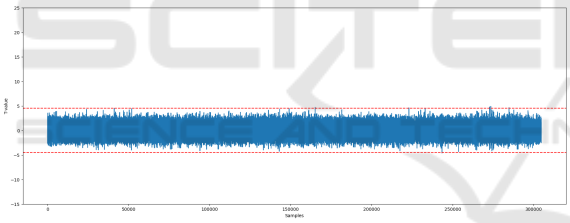
4 EXPERIMENTAL RESULTS

We conducted our experiments on the STM32 Nucleo-F439ZI development board equipped with an ARM Cortex-M4 core at 180 MHz. We selected the *pqm4* (Kannwischer et al.,) implementation of HQC. We retrieved the code of the gadgets from Github (dem, 2024) and implemented our solution. After setting the masking order to one, we ran 10,000 executions of our masked Barrett reduction, once with fixed inputs and once with random ones; and recorded

the resulting electro-magnetic traces. For comparison purposes, we ran this first experiment while fixing the masks to zero (virtually unmasking the secret value). Using a Test Vector Leakage Assessment (TVLA) (Becker et al., 2013) we confront these two sets of traces and obtain Figure 1a. The numerous peaks well above and below the ±4.5 threshold (Schneider and Moradi, 2016) (dotted red lines) indicates the presence of leaks. We then ran a second experiment, this time relying on the integrated True Random Number Generator (TRNG) of the board to produce the random masks required for the computation. This time, there are no visible peaks (Figure 1b) which is expected from a first order leakage analysis of a first order masking. Hence, the absence of leaks in the second experiment gives us better confidence concerning the soundness of our solution.



(a) with masks set to zero.



(b) with random masks.

Figure 1: TVLA of the masked Barrett reduction.

Remark 4. A formal security proof on the pseudo or source code is not a complete guarantee. Due to all optimizations performed by the compilation tool chain and even by the hardware, executing the corresponding compiled software on a real hardware CPU can leak secret data, even when the abstract algorithm was proved t-NI secure. Further analyzes on the assembly code, on the linked and loaded binary or even on the actual execution by the processor are needed before one can conclude that the masking provides the expected security level, as demonstrated for instance in (Coron et al., 2012).

Our results were obtained with the `-Og` compilation flag and all caches ON. While testing another configuration, we found out that compiling with the `-O3` flag introduces leaks (see Figure 3 in Appendix). As we explain in Appendix these leaks are due to a different mapping of the various shares on the physical CPU registers.

4.1 Accelerating the Multiplication

Boolean masking is one of several masking techniques. Arithmetic masking, for instance, splits a secret variable x into $d+1$ shares such that:

$$x = x_0 + \cdots + x_d \mod 2^k \quad (1)$$

In (Demange and Rossi, 2024) the authors decided to avoid conversions between boolean and arithmetic masking (BtoA and AtoB) because they are considered as expensive and because most BIKE operations are binary. As HQC also uses binary vectors and operations, taking advantage of this structure to do efficient computations (e.g., the addition of two vectors is a bitwise exclusive OR (XOR) of their components), the choice of boolean masking makes sense.

However, if we could find an operation which is faster under arithmetic masking, even taking into account the cost of the conversions, we could expect a performance improvement. The multiplication of our Barrett reduction function is an interesting candidate because in boolean masking it represents nearly 80% of the function, while multiplying by a public value n in arithmetic masking simply consists in multiplying each share of x by n; the cost is in $O(d)$. We thus compared the cost of a multiplication in boolean masking and the cost of a sequence BtoA, arithmetic multiplication, AtoB:

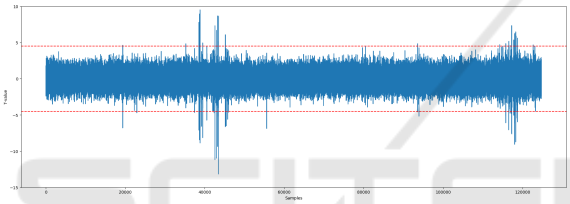
1. Boolean multiplication, no conversion: $O(k \times \log k \times d^2)$
2. Arithmetic multiplication, with conversions: $O(\log k \times d^2)$

with $k = 32$ bits and d the order of masking.

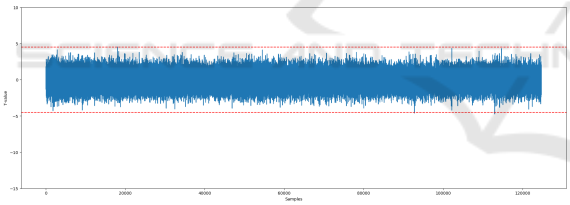
It appears that, theoretically, the multiplication could be significantly accelerated. We designed a new version of our masked Barrett reduction, replacing the boolean multiplications with arithmetic ones and the appropriate conversions (see Algorithm 7). For simplicity sake, we will refer to this version as *Arithmetic Barrett*, to contrast our first solution which relied exclusively on boolean masking. We based the code for the conversions on the pseudo-code described in (Coron et al., 2014, Algorithm 4 & 6). We ran the same experiments as described at the beginning of Section 4 to check if this new solution was still secure. The results are presented in Figure 2. As expected, using mask conversions and arithmetic masking does not affect the security of our solution.

Algorithm 7: Arithmetic Barrett.

Data: $[\![a]\!], m, n$
Result: $[\![r]\!] = [\![a]\!] \bmod n$
1 $[\![ari\_a]\!] \leftarrow \texttt{BtoA}([\![a]\!])$;
2 $[\![ari\_q]\!] \leftarrow [\![ari\_a]\!] \times m$; `/* Multiplication on each share */`
3 $[\![q]\!] \leftarrow \texttt{AtoB}([\![ari\_q]\!])$;
4 $[\![q]\!] \leftarrow [\![q]\!] \gg 32$
5 $[\![ari\_q]\!] \leftarrow \texttt{BtoA}([\![q]\!])$;
6 $[\![ari\_qn]\!] \leftarrow [\![ari\_q]\!] \times n$;
7 $[\![ari\_a\_qn]\!] \leftarrow [\![ari\_a]\!] - [\![ari\_qn]\!]$; `/* Sharewise subtraction */`
8 $[\![a\_qn]\!] \leftarrow \texttt{AtoB}([\![ari\_a\_qn]\!])$;
9 $minus\_n \leftarrow 2^{32} - n$;
10 $[\![A]\!] \leftarrow \texttt{sec}_{+}([\![a\_qn]\!], minus\_n)$;
11 $[\![z]\!] \leftarrow [\![A]\!] \gg 31$;
12 $[\![A]\!] \leftarrow \texttt{refresh}([\![A]\!])$;
13 $[\![a\_qn]\!] \leftarrow \texttt{refresh}([\![a\_qn]\!])$;
14 $[\![r]\!] \leftarrow \texttt{sec}_{if}([\![a\_qn]\!], [\![A]\!], [\![z]\!])$;



(a) with masks set to zero.



(b) with random masks.

Figure 2: TVLA of the masked Barrett reduction with mask conversions.

4.2 Performance Comparison

In Table 1, we compare the performance of the sampling of random vectors with weight 75 in five different settings: the HQC and BIKE reference implementations, HQC with our Barrett reduction masked at order one in boolean and arithmetic masking, and the Demange and Rossi masking scheme for BIKE. The following results were all obtained with the `-Og` compilation flag.

Our fully boolean masked solution is 51% more computationally intensive compared to the one proposed for BIKE. This is due to the fact that BIKE specification only requires a masked multiplication and a shift. In comparison, following HQC specification requires to execute the entire masked Barrett reduction (Algorithm 6). However, by using the arithmetic masking to accelerate the multiplications, our second solution achieves a performance only 5% slower than BIKE, despite the conversions. The impact of our second solution on the performance is thus limited while providing the key advantage of preserving the HQC specification.

Table 1: Comparison of the number of cycles needed to sample a random vector of weight $w = 75$ (ARM Cortex-M4 @180 MHz, caches ON).

Vector sampling	Cycles	Overhead
HQC	747,500	–
HQC Boolean Barrett	17,496,000	2340%
HQC Arithmetic Barrett	12,203,000	1632%
BIKE	746,500	–
BIKE Boolean	11,606,000	1554%

5 CONCLUSION

This paper proposes a first masked version of HQC vector sampling, based on Demange and Rossi's `SecFisherYates`, and new gadgets that we introduce, mainly the masked Barrett reduction, which was not publicly available so far. Adapting `SecFisherYates` to HQC is now as simple as replacing Line 2 in Algorithm 2 with our masked Barrett reduction.

Moreover, we developed a faster version with mask conversions and multiplications under arithmetic masking. We underline that BIKE's vector sampling could as well benefit from this acceleration.

The security assessment of our modular Barrett reduction is two-fold: we provide a security proof for our new gadgets, and we perform a practical side-channel evaluation on an STM32 device, that shows the absence of remaining leakages in this specific setting. This practical evaluation raises some warnings about compiler options, since some leakage appears with aggressive compiler optimization settings. It reminds the importance of practical validations.

We provide a publicly available C implementation. Vector sampling is only a part of the HQC algorithms. Even if side-channel resistant implementations of some other parts of HQC have been studied in (Goy et al., 2022), building a full HQC masked implementation is still let as future work.

ACKNOWLEDGMENTS

This work was partially funded by the grant 2022154 from the Appel à projets 2022 thèses AID Cifre-Défense by the Agence de l'Innovation de Défense (AID), Ministère des Armées (French Ministry of Defense). We wish to thank Benoît Cogliati for helpful discussions.

REFERENCES

(2016). Announcing request for nominations for public-key post-quantum cryptographic algorithms. https://csrc.nist.gov/News/2016/Public-Key-Post-Quantum-Cryptographic-Algorithms.

(2022). Pqc standardization process: Announcing four candidates to be standardized, plus fourth round candidates. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4.

(2023). The ibm quantum development roadmap. https://www.ibm.com/quantum/roadmap.

(2024). Announcing approval of three federal information processing standards (fips) for post-quantum cryptography. https://csrc.nist.gov/News/2024/postquantum-cryptography-fips-approved.

(2024). masked bike code. https://github.com/loicdemange/masked_BIKE_code.

(2025). Nist selects hqc as fifth algorithm for post-quantum encryption. https://www.nist.gov/news-events/news/2025/03/nist-selects-hqc-fifth-algorithm-post-quantum-encryption.

Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., et al. (2022). Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*.

Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., and Standaert, F.-X. (2015). On the cost of lazy engineering for masked software implementations. In Joye, M. and Moradi, A., editors, *Smart Card Research and Advanced Applications*, pages 64–81, Cham. Springer International Publishing.

Barrett, P. (1987). Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In Odlyzko, A. M., editor, *Advances in Cryptology — CRYPTO' 86*, pages 311–323, Berlin, Heidelberg. Springer Berlin Heidelberg.

Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y., and Zucchini, R. (2016). Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 116–129, New York, NY, USA. Association for Computing Machinery.

Baïsse, C., Moran, A., Goy, G., Maillard, J., Aragon, N., Gaborit, P., Lecomte, M., and Loiseau, A. (2024). Secret and shared keys recovery on hamming quasi-cyclic with sasca. Cryptology ePrint Archive, Paper 2024/440. https://eprint.iacr.org/2024/440.

Becker, G. T., Cooper, J., DeMulder, E. K., Goodwill, G., Jaffe, J., Kenworthy, G., Kouzminov, T., Leiserson, A. J., Marson, M. E., Rohatgi, P., and Saab, S. (2013). Test vector leakage assessment (tvla) methodology in practice.

Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. (1999). Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 398–412. Springer.

Coron, J.-S. (2013). Higher order masking of look-up tables. Cryptology ePrint Archive, Paper 2013/700. https://eprint.iacr.org/2013/700.

Coron, J.-S., Giraud, C., Prouff, E., Renner, S., Rivain, M., and Vadnala, P. K. (2012). Conversion of security proofs from one leakage model to another: A new issue. In Schindler, W. and Huss, S. A., editors, *Constructive Side-Channel Analysis and Secure Design*, pages 69–81, Berlin, Heidelberg. Springer Berlin Heidelberg.

Coron, J.-S., Großschädl, J., and Vadnala, P. K. (2014). Secure conversion between boolean and arithmetic masking of any order. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 188–205. Springer.

Demange, L. and Rossi, M. (2024). A provably masked implementation of bike key encapsulation mechanism. *Cryptology ePrint Archive*.

Gigerl, B., Hadzic, V., Primas, R., Mangard, S., and Bloem, R. (2021). Coco: Co-Design and Co-Verification of masked software implementations on CPUs. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1469–1468. USENIX Association.

Goubin, L. and Patarin, J. (1999). Des and differential power analysis the "duplication" method. In *Cryptographic Hardware and Embedded Systems: First InternationalWorkshop, CHES'99 Worcester, MA, USA, August 12–13, 1999 Proceedings 1*, pages 158–172. Springer.

Goy, G., Loiseau, A., and Gaborit, P. (2022). A new key recovery side-channel attack on hqc with chosen ciphertext. In *International Conference on Post-Quantum Cryptography*, pages 353–371. Springer.

Goy, G., Maillard, J., Gaborit, P., and Loiseau, A. (2024). Single trace hqc shared key recovery with sasca. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):64–87.

Guo, Q., Hlauschek, C., Johansson, T., Lahr, N., Nilsson, A., and Schröder, R. L. (2022). Don't reject this: Key-recovery timing attacks due to rejection-sampling in hqc and bike. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 223–263.

Kannwischer, M. J., Petri, R., Rijneveld, J., Schwabe, P., and Stoffelen, K. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

Kocher, P. C. (1996). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer.

Melchor, C. A., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.-C., Gaborit, P., Persichetti, E., Zémor, G., and Bourges, I. (2018). Hamming quasi-cyclic (hqc). *NIST PQC Round*, 2(4):13.

Paiva, T. B. and Terada, R. (2020). A timing attack on the hqc encryption scheme. In *Selected Areas in Cryptography–SAC 2019: 26th International Conference, Waterloo, ON, Canada, August 12–16, 2019, Revised Selected Papers 26*, pages 551–573. Springer.

Regev, O. (2023). An Efficient Quantum Factoring Algorithm.

Schneider, T. and Moradi, A. (2016). Leakage assessment methodology: Extended version. *Journal of Cryptographic Engineering*, 6:85–99.

Sendrier, N. (2021). Secure sampling of constant-weight words–application to bike. *Cryptology ePrint Archive*.

Wafo-Tapa, G., Bettaieb, S., Bidoux, L., Gaborit, P., and Marcatel, E. (2019). A practicable timing attack against hqc and its countermeasure. *Cryptology ePrint Archive*.

A APPENDICES

A.1 HQC

Algorithm 8: HQC.KeyGen.

Data: $\mathcal{R}$
Result: sk, pk
$\mathbf{h} = \text{Sample}(\mathcal{R})$;
$\mathbf{x} = \text{Sample}(\mathcal{R}, \omega)$;
$\mathbf{y} = \text{Sample}(\mathcal{R}, \omega)$;
$\sigma = \text{rand}(\mathbb{F}_2^k)$;
$sk = (\mathbf{x}, \mathbf{y}, \sigma)$;
$pk = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$;

Algorithm 9: HQC.Encrypt.

Data: $pk, \mathbf{m}, \theta$
Result: $\mathbf{c} = (\mathbf{u}, \mathbf{v})$
$\mathbf{r}_1 = \text{Sample}(\theta, \omega_r)$;
$\mathbf{r}_2 = \text{Sample}(\theta, \omega_r)$;
$\mathbf{e} = \text{Sample}(\theta, \omega_e)$;
$\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$;
$\mathbf{v} = \mathcal{C}.\text{Encode}(\mathbf{m}) + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$;

Algorithm 10: HQC.Decrypt.

Data: $sk, \mathbf{c}$
Result: $\mathbf{m}$
$\mathbf{m} = \mathcal{C}.\text{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$;

Algorithm 11: HQC.Encaps.

Data: pk
Result: $K, \mathbf{c}, salt$
$\mathbf{m} = \text{rand}(\mathbb{F}_2^k)$;
$salt = \text{rand}(\mathbb{F}_2^{128})$;
$\theta = \mathcal{G}(\mathbf{m} \| \texttt{firstBytes}(pk) \| salt)$;
$\mathbf{c} = \text{HQC.Encrypt}(pk, \mathbf{m}, \theta)$;
$K = \mathcal{K}(\mathbf{m}, \mathbf{c})$;

Algorithm 12: HQC.Decaps.

Data: $sk, \mathbf{c}, salt$
Result: K
$\mathbf{m}' = \text{HQC.Decrypt}(sk, \mathbf{c})$;
$\theta' = \mathcal{G}(\mathbf{m}' \| \texttt{firstBytes}(pk) \| salt)$;
$\mathbf{c}' = \text{HQC.Encrypt}(pk, \mathbf{m}', \theta')$;
if $\mathbf{m}' = \perp \vee \mathbf{c} \neq \mathbf{c}'$ **then**
  $K = \mathcal{K}(\sigma, \mathbf{c})$;
end
else
  $K = \mathcal{K}(\mathbf{m}, \mathbf{c})$;
end

Algorithm 13: $\texttt{sec\_}([\![a]\!], [\![b]\!])$ (secure minus).

Data: $[\![a]\!], [\![b]\!]$
Result: $[\![r]\!] = [\![a]\!] - [\![b]\!]$
$[\![nb]\!] \leftarrow [\![b]\!]$;
$[\![nb]\!]_0 \leftarrow \neg [\![nb]\!]_0$; /* NOT of the first share */
$[\![mb]\!] \leftarrow \texttt{sec}_+([\![nb]\!], 1)$; /* $-b = (\neg b) + 1$ */
$[\![r]\!] \leftarrow \texttt{sec}_+([\![a]\!], [\![mb]\!])$;

Theorem 2. *The masked subtraction Algorithm 13 is t-NI.*

Proof. The only gadget manipulating the data is $\texttt{sec}_+$, which is t-NI, and the negation is linear and only manipulates the first share. □

A.2 Example of Leak Introduced by the Compiler Optimizations

When testing our solution with the more aggressive `-O3` compilation option we discovered that the compiler decided to reuse the same register for two shares as can be seen on Listing 1. Since we use a first order masking, the transition between the two states of the register induces a XOR between the two values, which unmasks the secret as illustrated by Figure 3.

On Line 6, the value of `y[0]` is loaded in `r2` then on Line 8 the value of `y[1]` is loaded in `r2`. Physically, for the register to switch from the value `y[0]` to

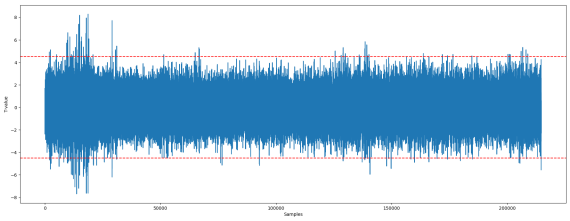


Figure 3: TVLA of the masked Barrett reduction compiled with `-O3`.

`y[1]` there is an implicit XOR between the two values. As $y = y[0] \oplus y[1]$, for all 0-bits of `y` there is no transition (that is, no consumed energy) between the corresponding bits of `y[0]` and `y[1]`, while for all 1-bits of `y` there is a transition from 0 to 1 or from 1 to 0, that consumes energy.

This involuntarily exposes the secret `y` and advocates for further analyses on the assembly code, on the linked and loaded binary or even on the actual execution by the processor as proposed for instance by (Gigerl et al., 2021). Increasing the masking order, as suggested by (Balasch et al., 2015), is another option but it is costly.

```
1 <boolean_sec_and>:
2 push {r3,r4,r5,r6,r7,lr}
3 mov  r5,r1       ;store y address in
      r5
4 ldrd r1,r3,[r0] ;r1 = x[0], r3 = x
      [1]
5 mov  r4,r2
6 ldr  r2,[r5,#0] ;r2 = y[0]
7 ands r1,r2       ;r1 = x[0] & y[0]
8 ldr  r2,[r5,#4] ;r2 = y[1]
9 ...
```

Listing 1: Abstract of the assembly code of $\texttt{sec}_{\&}$ for `-O3`.