First-Order Masked Kyber on ARM Cortex-M4

Report

Daniel Heinz[1,2], Matthias J. Kannwischer[3], Georg Land[4,5], Thomas Pöppelmann[2], Peter Schwabe[6,7] and Daan Sprenkels[7]

[1] Research Institute CODE, Universität der Bundeswehr München, Germany
daniel.heinz@unibw.de
[2] Infineon Technologies AG, Germany thomas.poeppelmann@infineon.com
[3] Academia Sinica, Taipei, Taiwan matthias@kannwischer.eu
[4] Ruhr-Universität Bochum, Germany georg.land@ruhr-uni-bochum.de
[5] DFKI GmbH, Cyber-Physical Systems, Bremen, Germany
[6] Max Planck Institute for Security and Privacy, Bochum, Germany peter@cryptojedi.org
[7] Radboud University, Nijmegen, The Netherlands daan@dsprenkels.com

Abstract. In this work, we present a fast and first-order secure Kyber implementation optimized for ARM Cortex-M4. Most notably, to our knowledge this is the first liberally-licensed open-source Cortex-M4 implementation of masked Kyber. The ongoing NIST standardization process for post-quantum cryptography and newly proposed side-channel attacks have increased the demand for side-channel analysis and countermeasures for the finalists. On the foundation of the commonly used PQM4 project, we make use of the previously presented optimizations for Kyber on a Cortex-M4 and further combine different ideas from various recent works to achieve a better performance and improve the security in comparison to the original implementations.
We show our performance results for first-order secure implementations. Our masked Kyber768 decapsulation on the ARM Cortex-M4 requires only 2 978 441 cycles, including randomness generation from the internal RNG. We then practically verify our implementation by using the t-test methodology with 100 000 traces.

Keywords: Lattice-Based Cryptography · Kyber · Side-Channel Analysis · ARM Cortex-M4

1 Introduction

In recent years the importance of post-quantum cryptography has significantly grown. Due to Shor's polynomial-time algorithm [Sho99] for solving prime factorization and discrete-logarithm problem, the usage of classical asymmetric cryptography including RSA and elliptic-curve cryptography is considered insecure against attacks involving large-scale quantum computers. Consequently the National Institute of Standards and Technologies (NIST) has started a standardization process to select appropriate post-quantum-secure algorithms [Nat16]. The focus has shifted towards side-channel security of the proposed schemes as different attacks on unmasked constant time implementations have been proposed in the meantime [RRCB20, RBRC20, PP19].

Schemes based on RLWE and its variants are among the most promising remaining candidates in the NIST post-quantum project. Oder et al. [OSPG18] presented the first fully masked CCA2-secure RLWE-scheme akin to NewHope [ADPS16]. While NewHope is no longer under consideration by NIST, two closely related finalists are Kyber [ABD+20] and Saber [DKRV20]. Recently, the protection of Saber [BDK+21] against side-channel attacks

has received research attention. Van Beirendonck et al. provide detailed performance numbers of first-order masked Saber on the ARM Cortex-M4. Their results were recently improved by Abdulrahman et al. [ACC+22]. The goal of our work is to make a comparison between the two lattice-based finalists in a first-order masked setting possible, which is why we also provide first performance numbers on this platform [HKL+21].

Contribution. This is a short report describing a first reference implementation on the ARM Cortex-M4 for a first-order masked Kyber which was presented at the third NIST Standardization Conference in June 2021 [HKL+21]. We make use of the existing PQM4 project which already provides strongly optimized assembly code for post-quantum cryptography on the Cortex-M4 including an optimized Kyber implementation [ABCG20]. Moreover, we combine different existing approaches and optimize them for the case of Kyber. Finally, we independently verify performance numbers and confirm first-order resistance using the fixed-vs-random TVLA methodology.

Related Work & Code. In comparison to the concurrent works of Bos et al. [BGR+21] and Fritzmann et al. [FVBBR+21] our masked Kyber implementation is especially tailored to the Cortex-M4 processor and the well known PQM4 project. In contrast to the M4 implementation in [BGR+21], we additionally provide t-test results on the Cortex-M4 as practical side-channel protection is sometimes hard to achieve, i.e. due to compiler optimizations. Additionally, we hereby verify that the theoretical concepts work in practice. Furthermore, our code is available to the public at `https://github.com/masked-kyber-m4/mkm4`.

2 Preliminaries

In this section we briefly introduce our notation and the Kyber key-encapsulation mechanism [ABD+20]. Furthermore, we recall some important works on masking RLWE in general and Kyber in particular.

2.1 Notation

For $x \in \mathbb{R}$, we define $\lceil x \rfloor = \lfloor x + \frac{1}{2} \rfloor \in \mathbb{Z}$. Let $\mathbb{Z}_q$ denote the quotient ring $\mathbb{Z}/q\mathbb{Z}$ for an integer $q > 1$. Thus, $\mathbb{Z}_q$ is the ring of cosets $x + q\mathbb{Z}$ with addition and multiplication operations. For $a, b \in \mathbb{Z}$, we write $a \bmod^{(+)} b$ for the unique integer $\hat{a} \equiv a \bmod b$ such that $0 \leq \hat{a} < b$. Let $R = \mathbb{Z}[X]/(f)$, where f is usually $f = X^n + 1$ for n being a power of 2, and $R_q = R/(q) = \mathbb{Z}_q[X]/(f)$ for some positive integer q. Any element $\mathbf{a} \in R_q$ as well as vectors of these elements are denoted as bold lower case letter. We use the notation $\mathbf{a}[i]$ for $i = 0, \ldots, n-1$ to access the i-th coefficient of a polynomial $\mathbf{a} \in R_q$. Matrices of elements in R_q are denoted as bold upper-case letters. For a given set S and a probability distribution D over S, we use $s \xleftarrow{r} D$ to mean $s \in S$ sampled according to D using coins r. In addition, we use $s \xleftarrow{\$} S$ to mean $s \in S$ sampled uniformly at random from S. Hereby, $U(q)$ denotes the uniform distribution on R_q, whereas χ denotes an error distribution to be defined for the specific algorithm. When covering our implementation we define the $x \bmod q$ operation for integers x, q to always produce an output in the range $[0, q-1]$. Unless stated otherwise, when we access an element $\mathbf{a}[i]$ of a polynomial $\mathbf{a} \in R_q$, we always assume that $\mathbf{a}[i]$ is reduced modulo q and in the range $[0, q-1]$.

2.2 Kyber Key Encapsulation Mechanism

For later reference we provide a simplified version of the public-key encryption scheme Kyber.CPA = (Kyber.CPA.Gen, Kyber.CPA.Enc, Kyber.CPA.Dec) as in Algo-

rithms 1, 2, and 3. Define integers $n = 256, q = 3329, \eta_2 = 2$ and let k, η_1, d_t, d_u, d_v be positive integers. We denote $\mathcal{M} = \{0,1\}^n$ as the plaintext space, where each message $m \in \mathcal{M}$ can be seen as a polynomial in R with coefficients in $\{0,1\}$. Define the functions

$$\begin{aligned}\textsc{Compress}_q(x,d) &:= \lceil (2^d/q) \cdot x \rfloor \bmod^{(+)} 2^d, \\ \textsc{Decompress}_q(x,d) &:= \lceil (q/2^d) \cdot x \rfloor.\end{aligned}$$

componentwise on each coefficient of a polynomial. We set χ_η as the centered binomial distribution with support $\{-\eta, \ldots, \eta\}$, and let $\chi_{n,\eta}$ be the distribution of polynomials of degree n with entries independently sampled from χ_η. When we apply the NTT/INTT to a vector of polynomials, the NTT/INTT gets applied to each polynomial individually.

Algorithm 1: Kyber.CPA.Gen.

$(\rho, \sigma) \xleftarrow{\$} \{0,1\}^{256} \times \{0,1\}^{256}$;
$\mathbf{A} \xleftarrow{\rho} U(q)^{k \times k}$;
$(\mathbf{s}, \mathbf{e}) \xleftarrow{\sigma} \chi_{n,\eta_1}^k \times \chi_{n,\eta_1}^k$;
$\hat{\mathbf{s}} \leftarrow \mathrm{NTT}(\mathbf{s})$;
$\hat{\mathbf{e}} \leftarrow \mathrm{NTT}(\mathbf{e})$;
$\hat{\mathbf{t}} \leftarrow \mathbf{A} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$;
return $pk_{CPA} := (\hat{\mathbf{t}}, \rho)$, $sk_{CPA} := \hat{\mathbf{s}}$;

Algorithm 2: Kyber.CPA.Enc.

Input: $pk_{CPA} = (\hat{\mathbf{t}}, \rho)$
Input: $m \in \mathcal{M}$
Input: $r \xleftarrow{\$} \{0,1\}^{256}$
$\mathbf{A} \xleftarrow{\rho} U(q)^{k \times k}$;
$(\mathbf{r}, \mathbf{e_1}, \mathbf{e_2}) \xleftarrow{r} \chi_{n,\eta_1}^k \times \chi_{n,\eta_2}^k \times \chi_{n,\eta_2}$;
$\hat{\mathbf{r}} \leftarrow \mathrm{NTT}(\mathbf{r})$;
$\mathbf{u} \leftarrow \mathrm{INTT}(\mathbf{A} \circ \hat{\mathbf{r}}) + \mathbf{e_1}$;
$\mathbf{v} \leftarrow \mathrm{INTT}(\hat{\mathbf{t}} \circ \hat{\mathbf{r}}) + \mathbf{e_2} + \lceil \frac{q}{2} \rfloor \cdot m)$;
$c_1 \leftarrow \textsc{Compress}_q(\mathbf{u}, d_u)$;
$c_2 \leftarrow \textsc{Compress}_q(\mathbf{v}, d_v)$;
return $c := (c_1, c_2)$;

Algorithm 3: Kyber.CPA.Dec.

Input: $sk_{CPA} = \hat{\mathbf{s}}$
Input: $c = (\mathbf{u}, \mathbf{v})$
$\mathbf{u} \leftarrow \textsc{Decompress}_q(\mathbf{u}, d_u)$;
$\mathbf{v} \leftarrow \textsc{Decompress}_q(\mathbf{v}, d_v)$;
return $m = \textit{Compress}_q(\mathbf{v} - \textit{INTT}(\hat{\mathbf{s}} \circ \textit{NTT}(\mathbf{u})), 1)$;

In order to obtain a CCA2-secure encryption scheme from the CPA-secure building blocks, Kyber makes use of a tweaked version of the Fujisaki–Okamoto transform [FO99]. The decrypted message is re-encrypted and then compared to the received ciphertext. If the ciphertext was not honestly generated, this comparison for equality fails and some pseudo-random key is returned. The resulting key-encapsulation mechanism is shown in Algorithms 4, 5, and 6.

Algorithm 4:
KYBER.CCAKEM.GEN.

1 $z \xleftarrow{\$} \{0,1\}^{256}$;
2 $(pk, sk') = \text{KYBER.CPA.GEN}()$;
3 $sk := (sk'||pk||\text{H}(pk)||z)$;
4 **return** pk, sk ;

Algorithm 5:
KYBER.CCAKEM.ENCAPS.

Input: Public key of CCAKEM pk
1 $m \xleftarrow{\$} \{0,1\}^{256}$;
2 $m \leftarrow \text{H}(m)$;
3 $(\bar{K}, r) := \text{G}(m||\text{H}(pk))$;
4 $c := \text{KYBER.CPA.ENC}(pk, m, r)$;
5 $K := \text{KDF}(\bar{K}||\text{H}(c))$;
6 **return** c, k ;

Algorithm 6:
KYBER.CCAKEM.DECAPS.

Input: Ciphertext of CCAKEM c
Input: Secret key of CCAKEM sk
1 Extract $(sk'||pk||\text{H}(pk)||z)$ from sk ;
2 $m' := \text{KYBER.CPA.DEC}(sk', c)$;
3 $(\bar{K}', r') := \text{G}(m'||\text{H}(pk))$;
4 $c' := \text{KYBER.CPA.ENC}(pk, m', r')$;
5 **if** $c = c'$ **then**
6 $\quad K := \text{KDF}(\bar{K}'||\text{H}(c))$;
7 **else**
8 $\quad K := \text{KDF}(z||\text{H}(c))$;
9 **end**
10 **return** K ;

3 Concept of the Implementation

The Kyber decapsulation algorithm (Algorithm 6) consists of four major building blocks visualized in Figure 1:

- A CPA-secure decryption,
- A CPA-secure re-encryption,
- A masked hash G,
- A masked comparison of ciphertexts.

In Figure 1, the non-linear parts have been marked in red. These parts are not straightforward to mask as it is not possible to split them into multiple shares easily. The decryption and re-encryption will be analyzed in the following sections. For the masked hash G, which is defined as SHA3-512 for Kyber, we make use of the approach in [BDPA10].

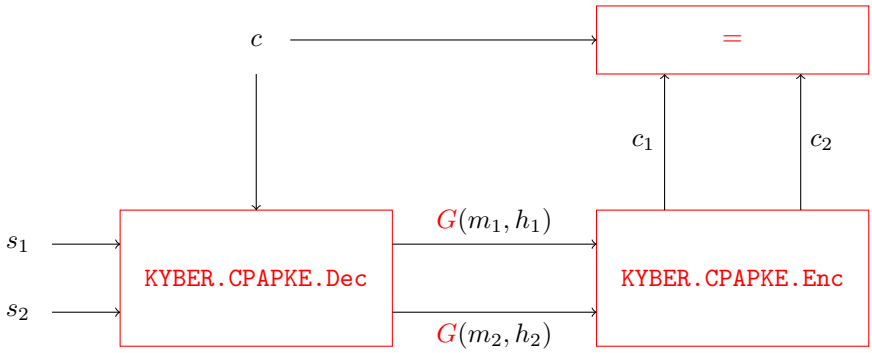

Figure 1: The masked CCA2-secure Kyber decapsulation.

3.1 Kyber Decryption

As most parts of the Kyber decryption depicted in Figure 2 are linear operations, they can be calculated on each share individually. However, the masked compression is non-linear and it is not possible to calculate it on each share separately. The input to the compression is shared arithmetically. The output on the other hand is a Boolean-shared message. This requires us to use an arithmetic-to-Boolean (A2B) conversion during compression. In [OSPG18, Alg. 2], a masked decoder has already been proposed. In the first step $q/4$ is subtracted from one share. Then, both arithmetic shares modulo q are transformed to a sharing (y_1, y_2) modulo 2^{16} - requiring no rejection sampling. Finally, $q/2$ is subtracted from one share y_i. We obtain the shared message by extracting the most significant bit from the Boolean sharing of (y_1, y_2).

It is important to mention that the A2B conversion from [OSPG18] has been shown to be insecure in [BDV21]. Therefore, we used the fixed single-lookup A2B algorithm from [BDV21] in our first-order masked decoder.

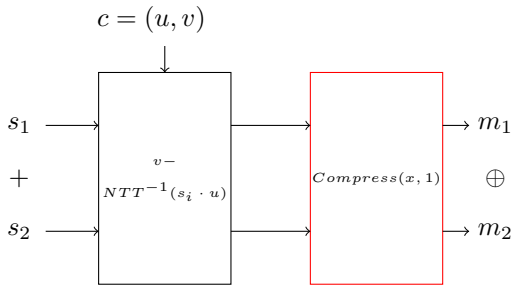

Figure 2: The masked CPA-secure Kyber decryption.

3.2 Kyber Re-Encryption

The Kyber re-encryption is the computationally most expensive part to mask. It includes the shared computation of a pseudorandom function (PRF), a centered binomial sampling, as well as compression and decompression of single coefficients. This is additionally made harder by the fact that the input seed is given in boolean sharing, whereas some intermediates are arithmetically shared as can be seen in detail in Figure 3 where $\oplus$ indicates boolean sharing and $+$ indicates arithmetic sharing.

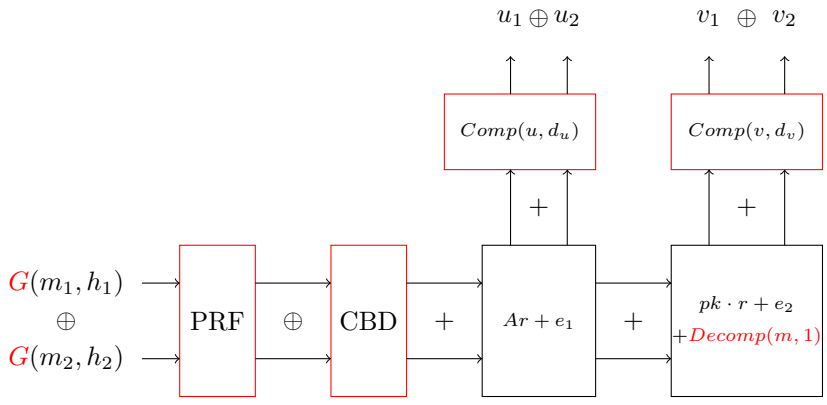

Figure 3: The masked CPA-secure Kyber re-encryption.

3.2.1 Masked Centered Binomial Sampling

To sample deterministically from a centered binomial distribution using a seed as input, Kyber specifies the usage of SHAKE as PRF to generate a sufficiently many random bits. To sample one coefficient (with $\eta = 2$), $a_0 + a_1 - b_0 - b_1$ is computed, where a_0, a_1, b_0, b_1 are four consecutive bits from the SHAKE output stream.

For the masked implementation, the seed is given in Boolean shares and our implementation expands it using a masked first-order secure implementation of Keccak [BDPA10]. Then our implementation follows the strategy of [SPOG19] and works on bitsliced Boolean shares to compute $a_0 + a_1 - b_0 - b_1 + \eta$. Here, η is added in order to avoid negative results. The result is then converted to an arithmetic sharing. For this, we follow the B2A_q strategy from [FVBBR+21, Alg. 12] by implementing a masked and bitsliced ripple-carry adder. Finally, the arithmetically shared samples are unpacked from the bitsliced representation and the first share is subtracted by η in order to return to the centered distribution.

3.2.2 Masked Decompression

The masked decompression with a decompression parameter $d = 1$ is exactly off by one if the decompression is performed on each share separately and both shares are equal to one. All other cases produce a correctly shared output. This is why we chose to adapt the strategy of [OSPG18] to subtract m_1 `AND` m_2 from the result of the sharewise operation in a masked way. For the arithmetic subshares m'_1, m''_1 (and m'_2, m''_2 respectively) we can calculate

$$\begin{aligned} c_1 &= pk \cdot r_1 + e_{2,1} - (m_1 \texttt{ AND } m_2) \\ &= pk \cdot r_1 + e_{2,1} - (m'_1 m'_2) - (m'_1 m''_2) - (m''_1 m'_2) - (m''_1 m''_2). \end{aligned}$$

It is important to execute the calculations exactly in order and to not store any unmasked values in registers as this would leak information. The method works because the register from which the values are subtracted is already randomized. It holds the result of $pk \cdot r_1 + e_{2,1}$, which is different in every execution of the masked decapsulation.

3.2.3 Masked Comparison

Recent work [BDH+21, DHP+21] has shown that little information, like the position of the difference between the submitted and re-encrypted ciphertext, can be used to significantly reduce the security level of the scheme. This is why we opted for the masked comparison as presented in a recent work on masking Kyber [BGR+21]. We do not directly compare compressed ciphertexts but check if the decompressed coefficients of the polynomial are in the correct interval. This choice allows us to omit the masked compression in the re-encryption step where the compression parameters d_u and d_v are not equal to one. Following the notation of [BGR+21], let a denote the masked sensitive variable and b denote the public variable. One can compute a lower bound $S(b)$ and a higher bound $E(b)$ such that $Compress(x) = b$ if and only if $S(b) \leq a \leq E(b) - 1$. Instead of checking if $Compress(a) = b$, one can now simply check if $S(b) \leq a \leq E(b) - 1$. This can be done by subtracting the value $S(b)$ and $E(b)$ from the masked a and then extracting the most significant bit from the two shares using an A2B conversion or an A2A conversion [BDK+21]. Extracting the MSB can be interpreted as a sign check because $a - S(b)$ is negative and $a - E(b)$ is positive if a is in the interval. The two bits are then combined into a final output bit using bitsliced calls to `SecAND`.

4 Evaluation

This section shows the performance and leakage evaluation of our implementation. Note that we use as baseline the existing non-masked implementation by Alkim et al. [ABCG20], which is also part of the PQM4 project [KRSS], and then modified the code to include the aforementioned countermeasures. Critical subroutines are implemented in assembly to avoid the recombination of shares in registers due to compiler optimizations.

4.1 Performance

In this section we present the performance of our first-order masked Kyber768 implementation. We measured performance on the STM32F407VG microcontroller with 32-bit ARM Cortex-M4 with FPU core that is mounted on the STM32F407VG Discovery board. The measurement setup is based on the PQM4 project [KRSS]. The CPU is clocked at 24 MHz to avoid wait states due to the slow speed of the memory controller. For the compilation, we used `arm-none-eabi-gcc` version `11.2.0` with compiler flags `-O3 -std=gnu99 -mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16`. The randomness required for masking is sampled using the internal hardware random number generator available on the STM32F407VG Discovery board. All in all, our result of 2 978 441 cycles for first-order masked Kyber decryption is similar to the performance numbers provided in [BGR+21] who reported 3 116 000 cycles on a STM32F407G.

Operation	Cycles
`crypto_kem_dec`	2 978 441
`indcpa_dec`	132 048
`hashg`	108 588
`indcpa_enc`	2 168 034
`comparison`	350 544
`hashh`	114 557
`kdf`	108 655
`indcpa_dec`	132 048
`unpackdecompress`	11 293
arithmetic operations	55 360
`masked_poly_tomsg`	65 286
`indcpa_enc`	2 168 034
`masked_poly_getnoise`	1 242 988
`poly_frommsg`	85 866
`masked_mattacc`	497 234
arithmetic operations	419 678

4.2 Leakage Evaluation

The leakage evaluation was performed on the ChipWhisperer Lite platform with an STM32F303 target [OC14] featuring an Arm Cortex-M4 core. This allows easy verification with a standardized platform. Furthermore, low noise level and well aligned traces are advantages of this setup. Instead of a random number generator which is missing on the ChipWhisperer Lite, we employ a deterministic pseudorandom sampler from a fixed input seed. We applied the commonly used t-test methodology [SM15] to each function separately. We use a fixed vs. random t-test (FvR) as it is proposed in the literature. In future work, a fixed plus noise vs. random t-test (FNvR) may also be helpful to differentiate between sensitive and non-sensitive output leakage [BDH+21]. If any t-value larger than 4.5 would appear in our diagram, this would be a strong indicator of remaining leakage. However, this is not the case for 100 000 traces measures for each experiment. The plot of the our t-test results is provided in Figures 4, 5, 6, and 7.

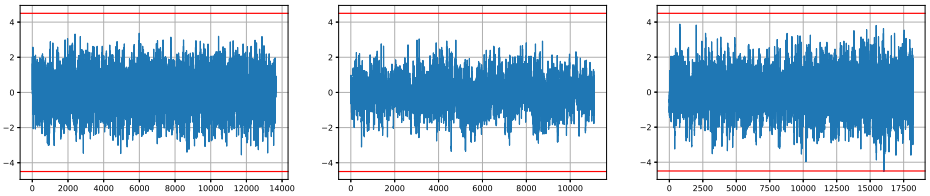

Figure 4: `polyinvntt_masked()`, `polysub_masked()`, `polybasemul_masked()`

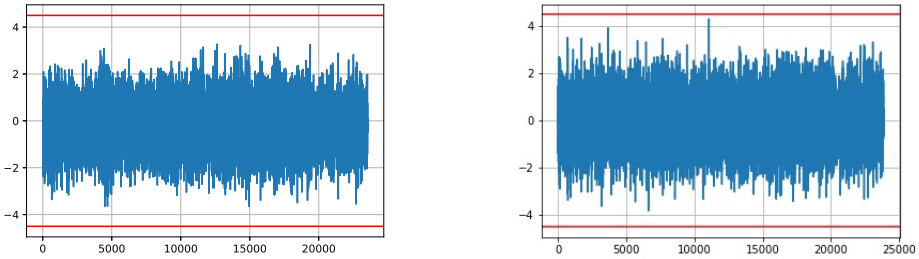

Figure 5: `polytomsg_masked()`, `polyfrommsg_masked()`

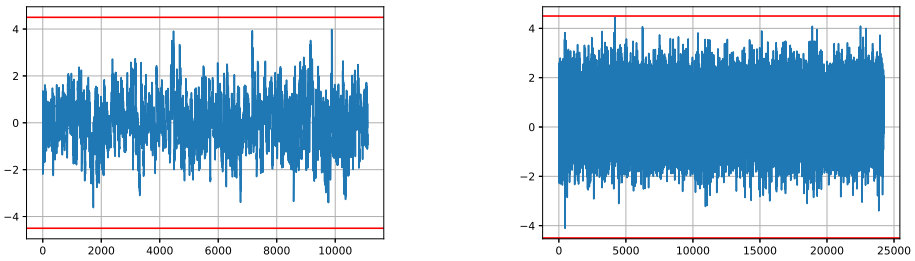

Figure 6: `polyreduce_masked()`, `polycompare_masked()`

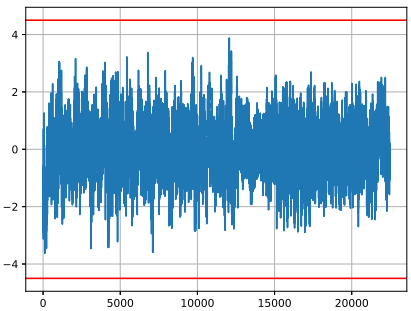

Figure 7: `masked_cbd()`

5 Conclusion

In this work, we presented a first-order masked Kyber specifically for the ARM Cortex-M4. We combined different approaches from previous works and practically verified the first-order resistance with the ChipWhisperer Lite. As previous attacks, i.e. [NDGJ21] have shown, first-order masking is not enough to achieve practical side-channel resistance. Higher-order implementations should be combined with different countermeasures as shuffling or hiding. Additionally, the computational overhead of some operations in higher-order solutions still seems quite high and it might be part of future work to reduce it.

Acknowledgments

This work has been supported by the European Commission through the ERC Starting Grant 805031 (EPOQUE), by the H2020 project PROMETHEUS (grant agreement ID 780701), by the Federal Ministry of Education and Research of Germany through the QuantumRISC (16KIS1038) and PQC4Med (16KIS1044) projects, and by the Sinica Investigator Award AS-IA-109-M01. This work was supported by the German Federal Ministry of Education and Research (BMBF) under the project “Aquorypt" (16KIS1017). Presented project results were partly supported by the project that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 830927.

We would like to thank the authors of [BDV21] for making their code available to us in advance. Also we would like to thank Peter Pessl for his time and his remarks on making our code more efficient.

References

[ABCG20] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R, M} LWE schemes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):336–357, 2020. 2, 7

[ABD+20] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS–Kyber (version 3.0) – submission to round 3 of the NIST post-quantum project. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf. 1, 2

[ACC+22] Amin Abdulrahman, Jiun-Peng Chen, Yu-Jia Chen, Vincent Hwang, Matthias J. Kannwischer, and Bo-Yin Yang. Multi-moduli NTTs for saber on Cortex-M3 and Cortex-M4. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):127–151, 2022. 2

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343. USENIX Association, 2016. 1

[BDH+21] Shivam Bhasin, Jan-Pieter D’Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck. Attacking and defending masked polynomial comparison for lattice-based cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):334–359, 2021. 6, 7

[BDK+21] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel-resistant implementation of SABER. *ACM J. Emerg. Technol. Comput. Syst.*, 17(2):10:1–10:26, 2021. 1, 6

[BDPA10] G. Bertoni, J. Daemen, Michaël Peeters, and G. V. Assche. Building power analysis resistant implementations of Keccak. 2010. 4, 6

[BDV21] Michiel Van Beirendonck, Jan-Pieter D'Anvers, and Ingrid Verbauwhede. Analysis and comparison of table-based arithmetic to boolean masking. *IACR Cryptol. ePrint Arch.*, 2021:67, 2021. 5, 9

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and higher-order implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):173–214, 2021. 2, 6, 7

[DHP+21] Jan-Pieter D'Anvers, Daniel Heinz, Peter Pessl, Michiel Van Beirendonck, and Ingrid Verbauwhede. Higher-order masked ciphertext comparison for lattice-based cryptography. *IACR Cryptol. ePrint Arch.*, page 1422, 2021. https://eprint.iacr.org/2021/1422. 6

[DKRV20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER – submission to round 3 of the NIST post-quantum project. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. https://www.esat.kuleuven.be/cosic/pqcrypto/saber/. 1

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999. 3

[FVBBR+21] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):414–460, 2021. 2, 6

[HKL+21] Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Daan Sprenkels. First-order masked kyber on ARM Cortex-M4, 2021. Third PQC Standardization Conference, https://csrc.nist.gov/Presentations/2021/first-order-masked-kyber-on-arm-cortex-m4. 2

[KRSS] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4. 7

[Nat16] National Institute of Standards and Technology (NIST). Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf, 2016. 1

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure saber KEM implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):676–707, 2021. 9

[OC14] Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 243–260. Springer, 2014. 7

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure and masked ring-LWE implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):142–174, 2018. 1, 5, 6

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019. 1

[RBRC20] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. Drop by drop you break the rock - exploiting generic vulnerabilities in lattice-based PKE/KEMs using EM-based physical attacks. *IACR Cryptol. ePrint Arch.*, page 549, 2020. 1

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):307–335, 2020. 1

[Sho99] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.*, 41(2):303–332, 1999. 1

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015. 7

[SPOG19] Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 534–564. Springer, 2019. 6