Secure hardware NTT implementation against SASCA and CPA attacks

Rafael Carrera Rodriguez, Florent Bruguier, Emanuele Valea, Pascal Benoit

Secure hardware NTT implementation against SASCA and CPA attacks

Rafael Carrera Rodriguez*†, Florent Bruguier*, Emanuele Valea†, Pascal Benoit*
*LIRMM, University of Montpellier, CNRS, Montpellier, France {rafael.carrera-rodriguez, florent.bruguier, pascal.benoit}@lirmm.fr
†Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France {rafael.carrerarodriguez, emanuele.valea}@cea.fr

***Index Terms*—PQC, NTT, side-channel attacks, FPGA**

I. Introduction

The Number Theoretic Transform (NTT) is an operation that allows to reduce the complexity of polynomial multiplications. It works like a Fast Fourier Transform over the ring of the integers. Then, the multiplication of two polynomials in the NTT can be solved by a point-wise multiplication (PWM), whose complexity is linear with the number of coefficients. This operation is key in several Post-Quantum Cryptography (PQC) algorithms like the key-encapsulation mechanism CRYSTALS-Kyber [1]. Therefore its efficient and secure implementation is essential for the correct deployment of quantum-resistant algorithms in physical systems.

The NTT has been subject of research efforts to assess its vulnerability to side-channel analyses. In 2017, Primas et. al. [2] presented a Soft-Analytical Side Channel Analysis (SASCA) against the inverse NTT. It works by representing the algorithm as a graph and executing the Belief Propagation algorithm with side-channel information. Additionally, the PWM operation is also vulnerable to correlation power analyses (CPA), as shown by [3].

To defend an NTT implementation against such attacks, several countermeasures are proposed. For the SASCA attack, countermeasures are focused on breaking the regularity of the NTT algorithm. One countermeasure is masking the twiddle factors used in each butterfly operation of NTT [4], by randomizing the power of the root of unity. To the knowledge of the authors, no NTT hardware implementation has been presented with this countermeasure.

In this work, we present a hardware implementation of the NTT for CRYSTALS-Kyber using the masking countermeasure from [4]. Moreover, we show how this countermeasure can be easily integrated with the blinding countermeasure from [5] to protect against CPA attacks on the PWM. The level of security of this implementation is configurable at runtime. This means that the user can decide the number of masks used at each round of NTT, making a tradeoff between performance and security.

II. NTT, SASCA attack and countermeasures

In CRYSTALS-Kyber [1], in order to reduce complexity for the polynomial multiplications, the authors choose to use the NTT. However, because of the modulus q chosen in CRYSTALS-Kyber, the field $\mathbb{Z}_q$ contains n-th primitive roots of unity, but not $2n$-th primitive ones. Therefore, the NTT of a polynomial $f \in R_q$ is a vector of 128 polynomials of degree 1.

Multiplication of two elements $f, g \in R_q$ is a polynomial multiplication. This operation, transformed in the NTT domain, becomes a point-wise multiplication (PWM), which, for this specific way of executing NTT, is defined as follows for the even and odd coefficients $2i$ and $2i+1$:

$$\hat{h}_{2i} = \hat{f}_{2i}\hat{g}_{2i} + \hat{f}_{2i+1}\hat{g}_{2i+1} \cdot \zeta^{2br_7(i)+1}$$
$$\hat{h}_{2i+1} = \hat{f}_{2i}\hat{g}_{2i+1} + \hat{f}_{2i+1}\hat{g}_{2i}$$

In [2], the authors presented an attack against the NTT by performing a Soft-Analytical Side Channel Attack. It is a profiling attack where both algorithm and side-channel information is used, by representing the NTT as a factor graph. Then, the Belief Propagation algorithm is used to obtain a probability for a guessed coefficient. In [4], the authors presented a countermeasure against such attack. This countermeasure works by randomizing the power of ζ, by adding a random mask in the basic butterfly operation of an NTT. The operation requirements for each masked butterfly depend on the masks used on input and output. I.e., if the inputs and outputs have the same mask or not. This fact itself depend on the number of masks of each butterfly. Equations (1) and (2) present two versions of the masked Cooley-Tukey butterfly, with the same mask at input and same mask at output (SISO), and different mask at input and different mask at output (DIDO), respectively. x represents the actual twiddle factor power, i, j represent previous masks and y, k, l represent new masks. It can be noticed that in the SISO case, two multiplications are needed, whereas in the DIDO case, four multiplications are needed. If an NTT hardware accelerator has a processing element with four multipliers, in the case of DIDO, a single operation is performed, while in the case of SISO, two operations can be performed in parallel. Moreover, if the NTT is performed in an unprotected fashion, 4 operations can be executed at the same time. The number of SISO-like or DIDO-like operations in an NTT with the countermeasure from [4], is directly related with the number of masks used at each round of the NTT. The authors from [4], argue that the number of masks increase the security of such an implementation. Therefore, an user could choose the

number of masks, by making a tradeoff between security and performance.

$$c' = a' \cdot \zeta^{y} + b' \cdot \zeta^{x+y} \tag{1}$$
$$d' = a' \cdot \zeta^{y} - b' \cdot \zeta^{x+y}$$
$$c' = a' \cdot \zeta^{2n-i+k} + b' \cdot \zeta^{2n-j+k+x} \tag{2}$$
$$d' = a' \cdot \zeta^{2n-i+l} - b' \cdot \zeta^{2n-j+l+x}$$

Such countermeasure can be integrated with another countermeasure to prevent CPA attacks in the PWM operation. Let the output of even and odd coefficients of an NTT, be masked by powers j and k, $\hat{f}'_{2i} = \hat{f}_{2i} \cdot \zeta^{j}$, $\hat{f}'_{2i+1} = \hat{f}_{2i+1} \cdot \zeta^{k}$. Then, executing PWM between a masked polynomial $\hat{f}'$ and a polynomial $\hat{g}$, equals to $\hat{h}'_{2i} = \hat{f}'_{2i} \cdot \hat{g}_{2i} + \hat{f}'_{2i+1} \cdot \hat{g}_{2i+1} \cdot \zeta^{x} = \hat{f}_{2i} \cdot \hat{g}_{2i} \cdot \zeta^{j} + \hat{f}_{2i+1} \cdot \hat{g}_{2i+1} \cdot \zeta^{k+x}$ and $\hat{h}'_{2i+1} = \hat{f}'_{2i} \cdot \hat{g}_{2i+1} + \hat{f}'_{2i+1} \cdot \hat{g}_{2i} = \hat{f}_{2i} \cdot \hat{g}_{2i+1} \cdot \zeta^{j} + \hat{f}_{2i+1} \cdot \hat{g}_{2i} \cdot \zeta^{k}$. If $k = j$, then $\hat{h}'_{2i} = \hat{h}_{2i} \cdot \zeta^{j}, \hat{h}'_{2i+1} = \hat{h}_{2i+1} \cdot \zeta^{j}$, and the real polynomial can be easily retrieved from the masked polynomial. This equality can be assured if the number of masks is limited to 128. This countermeasure is similar to the blinding proposed by [5].

III. Description of Architecture

In this work, we propose an NTT module with [4]'s countermeasure that can execute unprotected NTT and INTT, PWM and also their masked versions. The user can not only choose if the operation is protected or not, but can also decide the number of masks 2^i, where $0 < i < 7$, to ensure that the maximum number of masks per round is 128.

To perform all the possible combination of operations a processing element is designed with two so-called Double Butterfly Units (DBU), shown in Figure 1. Such DBUs can perform 2 butterfly operations each in the unprotected mode, 1 when two multiplications are needed and half of a butterfly operation, when 4 multiplications are needed. For the PWM case, the DBUs are cascaded to perform the operation with 4 multiplications, by reducing the number of multiplications from 5 to 4 of with a Karatsuba reduction.

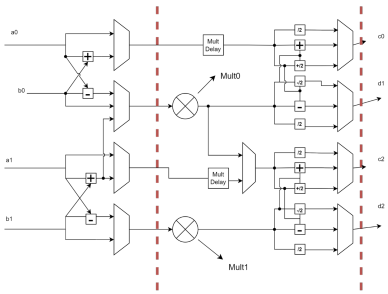

Figure 1: Double Butterfly Unit (DBU) architecture

The rest of the architecture is shown in Figure 2. It is composed by 6 basic parts: control unit, decoders, memories, processing element, masking unit and routing units. The control unit generates the coefficient indexes, the twiddle factor powers and the configuration signals for all the other modules. The decoder takes the coefficient indexes and decodes them for each of the memory banks. The memories are composed firstly by a primary memory, where the input polynomial is put for all of the operations and after an in-place strategy of execution, the output of the operations is stored. Then, there is the secondary memory, that stores the second polynomial for a PWM operation. Finally, the twiddle factor ROM, that contains all the possible 256 factors. The masking unit is charged of taking the actual power of the twiddle factor and mask it with random values, according to instructions from the control unit. Finally, the routing units direct the data and addresses to the correct memory banks and the correct inputs of the processing element.

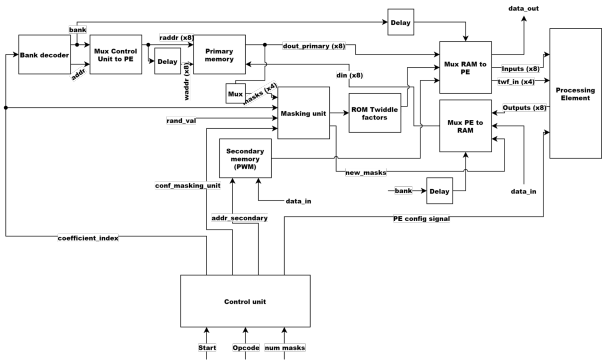

Figure 2: Full architecture

In order to allow for varying levels of parallelization depending on the butterfly type executed, a special scheduling of coefficients is used. 8 banks of primary memory are used, with the scheduling scheme from [6]. This allows to change the level of parallelism, while still avoiding memory collisions and read-after-write dependency issues.

A preliminary synthesis of the architecture was done for a Xilinx Artix-7 FPGA. It uses 4122 LUTs, 1445 FFs, 6 BRAM and 4 DSPs, with a frequency of 133 MHz. If area-time product ($ATP = (LUT + FF) * Time$) is taken to compare it with NTT implementations that also employ 4 multipliers for parallelization, our implementation has an overhead from around 2.3 to 5.4 times. Such an overhead is expected as none of the compared implementations employ countermeasures against SASCA or CPA attacks. However, this implementation is still a work in progress, and work is being done to reduce the overhead in ATP.

IV. Conclusion and Perspective

We present for the first time a hardware implementation of an NTT polynomial multiplier for CRYSTALS-Kyber equipped with countermeasures against SASCA and CPA attacks. It must be stressed that this is a work in progress. Improvements in area utilization and frequency are expected in future versions. Moreover, a leakage assessment will be performed on an FPGA programmed with our implementation, to verify if security properties are met in a real scenario.

REFERENCES

[1] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation," 2020. [Online]. Available: https://pq-crystals.org/kyber/.

[2] R. Primas, P. Pessl, and S. Mangard, "Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption," in *Lecture Notes in Computer Science*, pp. 513–533, Springer International Publishing, 2017.

[3] C. Mujdei, A. Beckers, J. Bermudo Mera, A. Karmakar, L. Wouters, and I. Verbauwhede, "Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication." Cryptology ePrint Archive, Paper 2022/474, 2022.

[4] P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay, "On configurable sca countermeasures against single trace attacks for the ntt," in *Security, Privacy, and Applied Cryptography Engineering* (L. Batina, S. Picek, and M. Mondal, eds.), (Cham), pp. 123–146, Springer International Publishing, 2020.

[5] M.-J. O. Saarinen, "Arithmetic coding and blinding countermeasures for lattice signatures," *Journal of Cryptographic Engineering*, vol. 8, pp. 71–84, jan 2017.

[6] J. Mu, Y. Ren, W. Wang, Y. Hu, S. Chen, C.-H. Chang, J. Fan, J. Ye, Y. Cao, H. Li, and X. Li, "Scalable and conflict-free NTT hardware accelerator design: Methodology, proof and implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.