A Survey of Software Implementations for the Number Theoretic Transform

Ahmet Can Mert[1], Ferhat Yaman[2], Emre Karabulut[2], Erdinç Öztürk[3], Erkay Savaş[3], and Aydin Aysu[2(✉)]

[1] Graz University of Technology, Graz, Austria
[2] North Carolina State University, Raleigh, NC, USA
aaysu@ncsu.edu
[3] Sabanci University, Istanbul, Turkey

Abstract. This survey summarizes the software implementation knowledge of the Number Theoretic Transform (NTT)—a major subroutine of lattice-based cryptosystems. The NTT is a special type of Fast Fourier Transform defined over finite fields, and as such, NTT enables faster polynomial multiplication. There have been over a decade of implementations of NTT following different design methods (e.g., CPU vs. GPU), aiming different optimization goals (e.g., memory-footprint vs. high-throughput), and proposing different styles of optimizations at different abstraction levels (e.g., arithmetic vs. assembly). At the same time, there are several techniques for evaluating and mitigating implementation attacks on NTT. Yet there is no quick guideline to help new developers/practitioners or future researchers given the continuing industry and academic efforts on NTT implementations. Our goal in this paper is to provide an overview of a decade of work. To that end, we survey NTT software implementations and categorize them based on their target platforms, optimization goals, and implementation security enhancements. We furthermore provide an executive summary of the key ideas proposed in related works. We hope this paper to be a designer pit stop into the NTT world and help them navigate to their destination.

Keywords: Number Theoretic Transform · Lattice-Based Cryptography · Software Implementations

1 Introduction

Lattice-based cryptography is a relatively new and versatile tool that allows formulating public-key encryption schemes [40,47], key-exchange protocols [8,19], key encapsulation mechanisms [20,36], homomorphic encryption systems [23,24,38], attribute-based encryption [22], digital signatures [5,43,46], and hash functions [15,60], among others. Efficient lattice-based encryption systems typically work with polynomials. For example, 5 out of 7 finalists that remained in the Round-3 of NIST's post-quantum cryptography competition use lattice-based cryptography, and all of those candidates work with polynomials. Also, three out

C. Silvano et al. (Eds.): SAMOS 2023, LNCS 14385, pp. 328–344, 2023.
https://doi.org/10.1007/978-3-031-46077-7_22

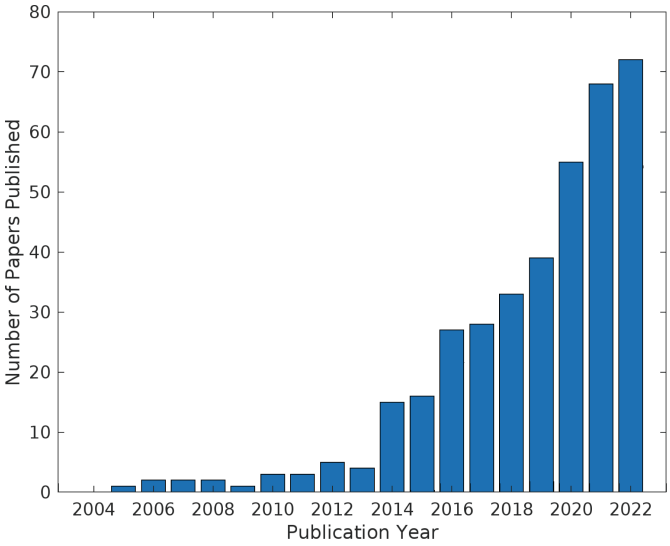

Fig. 1. The number of papers containing "Number Theoretic Transform" and "software implementation" in manuscript. The number grows steadily since 2004.

of four algorithms selected for standardization in 2022 are lattice-based. These cryptosystems need to multiply polynomials, and thus polynomial multiplication is a fundamental computing unit.

The Number Theoretic Transform (NTT) enables faster polynomial multiplication. NTT is essentially a discrete Fourier Transform used in lattice cryptography to operate on finite fields with polynomials. Therefore, just like how the Fast Fourier Transform (FFT) converts convolutions in the time domain to point-wise multiplications in the frequency domain, reducing the computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot \log n)$, the NTT can transform the schoolbook polynomial multiplication in one domain to coefficient-wise multiplications in another domain, achieving the same complexity reduction. Thus, the importance of NTT for lattice cryptography (and cryptography in general) is similar to the importance of FFT for signal-processing applications.

Along with the lattice cryptosystems using polynomials, the first paper on NTT implementations for lattice-based cryptography appeared about a decade ago [41]. Since then, NTT designs have covered hardware implementations [10,41,72], software implementations [4,6,7,11,14,16–18,21,28,29,31,33,41,42,44,56,57,59,61–64,67,71,73,75,80,81,84,90,95,96], hardware/software co-designs [30,51], and architecture extensions [51]. NTT solutions with each design method further cover an immense range—e.g., a software design has an extensive selection to target from 8-bit microcontrollers for resource-starved IoT nodes to Graphics Processing Units (GPUs) for server-side applications. A number of design goals and heuristics have, likewise, been adopted. In addition to the classical design goals such as area-cost, memory footprint, throughput, latency, power,

energy, flexibility, and the combinations thereof, cryptosystem implementations have the extra dimension of security [79]. Typical design extensions for implementation security include protections against side-channel and fault attacks [7,71,75,77,80].

Figure 1 quantifies our claims for software implementations of NTT. The figure shows the Google Scholar count of papers that contain the keywords "Number Theoretic Transform" and "software implementation" in the manuscript across the years from 2004 to 2022. The number has steadily grown from zero papers in 2004 to 72 papers in 2022. It is highly likely that this number will continue to grow further given the increasing number and use of lattice-based cryptography in a range of applications.

Given the richness of NTT implementations' corpus, our goal in this survey is to guide the newcomers and be the nexus between them and the prior works. We aim to achieve this for *software* implementers. An ideal reader of our paper is an engineer/researcher/educator who is new to this field and who wants to apply NTT for a target application with certain specifications or who wants to expand on the prior research. We argue that a good way to achieve this goal is to survey existing work and provide relevant pointers along with a useful summary. To that end, we identified 42 publications, classified based on the implementation aspects, provide an executive summary of key ideas, and organize this information, succinctly, in a table. Using such a table, interested readers can indeed analyze relevant works and quickly identify what has been done or what can be applied to meet their goals.

Our survey is unique in its focus and systematization, and it presents up-to-date information. Prior works, by contrast, either surveyed lattice theory rather than implementations [70] or have a broader scope covering different compute units or aspects in lattice-based cryptography [48,49,65,68]. The closest work to ours is by Valencia *et al.* [87], which describes NTT's design space exploration by surveying 10 relevant papers available at the time.

The rest of this article is organized as follows. Section 2 provides a formal background on the NTT. Section 3 describes our categorization method. Section 4 respectively present the prior work on software implementations, and Sect. 5 concludes the paper.

2 Preliminaries

This section describes the basics of the Number Theoretic Transform and why it is useful for lattice-based cryptographic systems.

2.1 Lattice-Based Cryptography

Hard lattice problems are studied for a very long time; however, their appearance in the field of cryptography was made by Ajtai's work, which is based on the short integer solution (SIS) problem [1]. Another lattice problem, learning with error (LWE), by Regev was proposed in 2005 [76]. These two problems and

their variants, ring-LWE (R-LWE) and ring-SIS (R-SIS), are the most important lattice problems in the field of cryptography. Lattice-based cryptography is favorable since it is conjectured to be secure against the attacks by quantum computers and the majority of the candidates in the Round-3 of NIST's post-quantum cryptography competition is lattice-based cryptographic schemes [3]. It also forms the mathematical basis for fully homomorphic encryption [38].

2.2 The Number Theoretic Transform

NTT-based polynomial multiplication is one of the most efficient and widely-used methods for polynomial multiplication in lattice-based cryptography. NTT and INTT operations can convert schoolbook polynomial multiplication operation into coefficient-wise multiplication operation [41]. The NTT is defined as DFT using integer-valued numbers over the ring $\mathbf{Z}_q$. Therefore, any DFT algorithm can be adapted to be used as an NTT algorithm. The NTT operation transforms a vector (or a polynomial) $\boldsymbol{a}$ with n elements into another vector (or polynomial) $\overline{\boldsymbol{a}}$ with n elements as defined in Eq. 1. The NTT operation working on an n-element vector is called n-point (pt) NTT.

$$\bar{\boldsymbol{a}}_i = \sum_{j=0}^{n-1} a_j \cdot \omega^{ij} \pmod{q} \text{ for } i = 0, 1, \ldots, n-1. \tag{1}$$

The NTT operation uses a constant called n^{th} root of unity, $\omega \in \mathbb{Z}_q$, which is also defined as *primitive root* or *twiddle factor*. For the existence of NTT operation, twiddle factor should satisfy the conditions $\omega^n \equiv 1 \pmod{q}$ and $\omega^i \neq 1 \pmod{q}$ $\forall i < n$, where $q \equiv 1 \pmod{n}$.

Inverse of NTT (INTT) operation can also be performed using almost the same formula with NTT operation as shown in Eq. 2. INTT operation uses the modular inverse of twiddle factor in $\mathbb{Z}_q$, $\omega^{-1} \pmod{q}$, and the resulting coefficients of INTT operation are multiplied with $n^{-1} \pmod{q}$ in $\mathbb{Z}_q$.

$$\boldsymbol{a}_i = \frac{1}{n} \sum_{j=0}^{n-1} \bar{a}_j \cdot \omega^{-ij} \pmod{q} \text{ for } i = 0, 1, \ldots, n-1. \tag{2}$$

Applying NTT and INTT operations as shown in Eq. 1 and Eq. 2 still yields high computational complexity. Therefore, efficient NTT algorithms use *divide-and-conquer* approach where each operation is divided into half-sized operations recursively. There are mainly two approaches based on the way division is performed: *Decimation-in-time* (DIT) and *Decimation-in-frequency* (DIF). The first approach uses Cooley-Tukey (CT) butterfly while the latter requires Gentleman-Sande (GS) butterfly [27]. The CT butterfly takes a, b and w as inputs and generates $a - b \cdot w \pmod{q}$ and $a + b \cdot w \pmod{q}$ as outputs. Similarly, the GS butterfly calculates $a + b \pmod{q}$ and $(a - b) \cdot w \pmod{q}$ as outputs. The DIF and DIT NTT algorithms can be adapted to work with different input and output polynomial ordering. For example, DIF NTT operation can take an input

with naturally ordered coefficients and outputs the resulting polynomial with bit-reversed ordered coefficients (i.e. coefficient at index $1 = 001_2$ actually corresponds to the coefficient at index $4 = 100_2$ for an 8-pt NTT). Similarly, it can be tweaked to take input polynomials with bit-reversed ordered coefficients and produce the output with naturally ordered coefficients [27]. These variations can eliminate redundant bit-reverse operations during the computations.

When the polynomial multiplication operation is performed over the polynomial ring $\mathbf{Z}[x]_q/\phi(x)$, a polynomial reduction operation by $\phi(x)$ is also required after the multiplication. When the polynomial $\phi(x)$ has the form of $x^n + 1$, *negative wrapped convolution* (NWC) technique can be utilized to eliminate the polynomial reduction operation and reduce computational complexity. However, this technique requires input and output polynomials to be multiplied with the powers of $2n^{th}$ root of unity, ψ, which are referred as pre-processing and post-processing operations, respectively. The constant ψ should satisfy the conditions $\psi^{2n} \equiv 1 \pmod q$ and $\psi^i \neq 1 \pmod q$ $\forall i < 2n$, where $q \equiv 1 \pmod{2n}$ [72]. The polynomial multiplication operation over the ring $\mathbf{Z}[x]_q/(x^n + 1)$ with NWC is shown in Eq. 3, where $\odot$ represents coefficient-wise multiplication. It should be noted that the multiplication with $n^{-1} \pmod q$ after INTT can be merged with post-processing operation [73].

$$\boldsymbol{c} = \texttt{INTT}(\texttt{NTT}(\boldsymbol{a} \odot [\psi^0, \ldots, \psi^{(n-1)}]) \odot \texttt{NTT}(\boldsymbol{b} \odot [\psi^0, \ldots, \psi^{(n-1)}])) \odot [\psi^0, \ldots, \psi^{-(n-1)}] \tag{3}$$

3 Categorization Method

This section describes the rationale behind our categorization strategy. We chose to represent the related works in four primary dimensions: target application, target platform, implementation security, and optimization target.

- Target application denotes if the NTT is used in a standalone fashion or if it is used as a component in a higher-level protocol. Such protocols include post-quantum cryptosystems (PQC), Digital Signatures (DS), Fully Homomorphic Encryption (FHE), and large integer multiplication, among others.
- Target platform indicates the desired execution environment. Indeed, software is used in various domains from edge devices to the cloud and everything in between. These devices include micro-controllers (μc) for the resource-constrained edge/IoT, central processing units (CPU) for general-purpose computing, and GPUs for efficient acceleration. Field Programmable Gate Arrays (FPGAs) are also becoming ever more popular due to their flexible acceleration nature at low energy — these devices can be used to configure a soft processor and execute software.
- Implementation security (shorthand, imp. sec.) refers to whether or not side-channel attacks or fault injection attacks were taken into account in the implementation and if there is some sort of defense deployed. This is done in a binary fashion, where ✓and ✗, respectively, marks if there is a defense or not. Obviously, such defenses will increase the area-delay overheads.

– Optimization target corresponds to the pursued metric of the software developers. Based on our review, the common options for optimization are speed (clock cycle count or number of operations per second), area (memory footprint), security (minimizing overheads of defense), energy (amount of effort required per operation), or their combination.

We also sort the papers in our categorization in the order they appear in the literature based on the year alone.

4 Software Implementation Summary

Table 1, 2, 3 show our overview based on the categorization described above. Each table shows the software for a different class of devices including CPU, embedded, and GPUs, respectively. We hope this table helps future adopters. For example, someone who is interested in the NTT implementations for microcontrollers with implementation security countermeasures and latency optimization can refer to our table and read the related papers listed. We next describe the key contribution of these papers in the rest of the section.

CPU Implementations. The first implementation of NTT in software is described by Gottert *et al.* [41]. The paper shows a baseline implementation on a conventional CPU without much emphasis on NTT optimizations. NTT is first optimized in software by utilizing SIMD operations, pre-calculating NTT constants, and memory access concatenations for Intel CPUs [44]—this approach was further advanced shortly after [33].

Another interesting strategy is to grow the size of the coefficients and reduce the number of reductions as well as multiplications within those reductions. This implementation has been carried out with a portable C implementation and a high-speed implementation using assembly with AVX2 [59].

A notable option is to use Preprocess-then-NTT [95], which improves a critical NTT limitation of setting $2n|q-1$ and allows using smaller modulus q. Note that this is useful for sketching new algorithms as standardized algorithms will come with pre-set q and n values. The approach was later extended by removing some redundant computations and reducing the value of q [96].

To our best knowledge, the first side-channel-aware implementation of NTT focused on a constant-time modular reduction approach [80]. This constant-time modular reduction was further accelerated with a modified version of the Montgomery algorithm and vectorized assembly optimizations [81].

Other optimizations include utilizing Radix-4 along with earlier vectorization techniques [63]. Likewise, Tan *et al.* combined earlier techniques for a low-latency CPU implementation, which include (i) nega-cyclic convolution with pre- and post-processing, (ii) CT-based butterfly structure to merge pre-processing and NTT operations, and extra bit-reversal removal [84].

Alkim *et al.* optimized the NTT for both memory footprint and latency [6]. The optimizations involve a hybrid (NTT + Karatsuba) polynomial multiplication, which takes a CRT map of NTT operation and then applies Karatsuba

Table 1. Literature Survey of CPU-based NTT Implementations

Work	Target Application	Implementation Security	Optimization Target
[41]	PQC	✗	Latency
[44]	PQC	✗	Latency
[33]	PQC	✗	Latency
[11]	PQC	✗	Latency
[59]	Standalone Impl.	✗	Latency
[80]	Standalone Impl.	✓	Security
[63]	Standalone Impl.	✗	Latency
[81]	PQC	✓	Latency
[95]	PQC	✗	Key size
[84]	PQC	✗	Latency
[61]	PQC	✗	Latency
[6]	PQC	✗	Latency-Area
[28]	PQC	✗	Latency
[96]	PQC	✗	Efficiency
[64]	Standalone Impl.	✗	Verification
[16]	HE	✗	Latency

algorithm for small polynomial multiplication. Chung *et al.* pursued an intriguing approach where NTT was retrofitted on NTT-unfriendly rings by applying the Good's trick [28]. The paper shows significant improvement over earlier work and implementation on CPU as well as on an ARM-Cortex-M4.

Verification is an important aspect yet not as thoroughly analyzed. A notable exception to this is achieved in the context of detecting overflows in NTT by using a static loop unrolling with a specialized abstract interpretation method [64].

Finally, an optimized NTT was used in the context of HE [16]. This work presents an Intel AVX-512 C++ library for polynomial arithmetic including NTT. The NTT/INTT implementations follow radix-2 CT and GS FFT implementations, respectively. One 512-bit vector executes 8 butterfly operations.

Microcontroller Implementations. There is significant research on realizing NTT on embedded microcontrollers given their use in edge/IoT applications.

The 32-bit Arm Cortex-M4 has been a popular target platform. NTT performance can be optimized by increasing memory usage and storing pre-computed twiddle factors, rather than generating them on-the-fly [67]. This work also offers the first two iterations of the DIT radix-2 NTT algorithm. The NTT is also implemented by using nega-cyclic convolution in addition to storing pre-computed ω values in memory [17]. Later on, negative-wrapped NTT along with computational optimizations from an earlier hardware implementation was ported to ARM Cortex-M4 [29]. Another optimization was performed by using signed Montgomery reductions to reduce required instructions (from four to

three cycles), merging two NTT stages to reduce load and store instructions, and unrolling NTT loops [21]. These optimizations were even pushed further by a 2-cycle modular reduction for Montgomery arithmetic, optimized small-degree polynomial multiplications with lazy reduction and early termination, more aggressive layer merging in the NTT to further save load/store operations, and pre-compute vs. on-the-fly trade-offs [4]. An adaptation of the Good's trick to cover NTT-unfriendly parameters and related comparison with other optimizations was later shown on Cortex-M4 [7].

NTT was also optimized for 8-bit Atmel microcontrollers [73]. The optimizations include removing the bit-reversal operations, using the subtract-and-shift algorithm for modular reduction with assembly optimizations, and optimizing the first and last stages of NTT.

Another work has implemented NTT software on ARM7TDMI and ATmega64 without much emphasis on optimization but to provide comparative analysis and execution suitability on Java cards including energy costs [18]. A similar approach was also taken for the use and energy costs of NTT in the context of Identity Based Encryption (IBE) on a RISC-V-based microcontroller without much emphasis again on NTT optimizations [14]. Other implementations include vectorization for ARM-NEON architecture [11,56], which was further assembly-optimized in subsequent works [57].

Table 2. Literature Survey of Microcontroller-based NTT Implementations

Work	Target Application	Implementation Security	Optimization Target
[17]	Authentication protocol	✗	Area
[67]	PQC	✗	Latency
[18]	PQC	✗	Latency-Area
[73]	PQC	✗	Latency
[57]	PQC	✗	Latency-Area
[29]	PQC	✗	Latency
[56]	PQC	✗	Latency
[71]	PQC	✓	Security
[21]	PQC	✗	Latency
[28]	PQC	✗	Latency
[4]	PQC	✗	Latency
[7]	PQC	✓	Latency-Area
[14]	PQC, IBE, TLS	✓	Energy
[90]	Standalone Impl.	✗	Area
[75]	Standalone Impl	✓	Security

Side-channel analysis of software has also received increasing attention, which includes demonstrating single-trace leakages of NTT [71,74], simple power analysis leakage of inverse NTT [91]. Subsequently, shuffling and masking defenses are

proposed to protect the NTT against power, EM, and timing side-channels [75], while other two prior works [7,14] aim to avoid only timing-side-channels.

GPU Implementations. GPUs are another venue for software implementations, which enable the efficient realization of more complex lattice-based protocols, as shown in Table 3. For example, the NVIDIA CUDA Fast Fourier Transform (cuFFT) library accelerates NTT by parallelizing iterative NTT algorithm [2]. Larger NTTs, which are especially useful in FHE, attribute-based encryption [31], lattice-based hash function [86], or large-integer multiplication [26], are also accelerated by GPUs, even with using a multi-GPU system [88]. The GPUs are also widely used accelerator platforms for PQC applications. Since polynomial multiplication is a well-known bottleneck in PQC applications, GPU is employed to accelerate NTT with different optimization techniques [2,12,37,45,53–55,82,94]. Prior works further gave performance evaluations to inform the algorithm developers on efficient parameters [2,12].

GPU parallelized the NTT for-loops with avoiding warp divergence that slows performance [53,55]. The subsequent work [45] accelerated multiple PQC algorithms (FrodoKEM, NewHope, and Kyber) by executing NTT operations with multi-threading. Interestingly, a NewHope GPU acceleration [37] differed from the prior work [45] by avoiding frequently using high-cost interthread memories and instead performing register shuffling to share intermediate data. Combining the first two levels of NTT is another proposed method to improve performance by reducing memory read/write operations [54]. Zhao *et al.* [94] eliminated warp divergence by efficiently executing butterfly operations in an NTT stage. Shen *et al.* [82] applied radix-8 NTT/INTT and performed an efficient 3-level processing which uses shared memory for fast data exchange between NTT stages.

The implementation of large integers and polynomials on GPUs can be challenging due to their size. The Chinese remainder theorem (CRT) and NTT techniques address this problem as proposed in cuHE [32]. cuHE works with 32-bit GPU kernel arithmetic operations and represents large integers with integers smaller than 32-bit. Additionally, cuHE parallelizes NTT execution while leveraging the GPUs' shared memory architecture. However, cuHE faces two practical issues in their NTT kernels: shared memory conflicts and thread divergence. To address shared memory conflicts, Chang *et al.* [25] and Goey *et al.* [39] proposed implementations that store pre-computed twiddle factors in registers. They also utilized warp shuffle instructions that enable storing twiddle factors within a limited register memory space. A subsequent study [13] proposed another solution to address shared memory conflicts while eliminating thread divergence. The proposed solution stores a 64-bit integer into 2×32-bit integer arrays to eliminate memory conflicts. Additionally, thread divergence is resolved by storing pre-computing the twiddle factors in constant memory.

In a later study, Wang *et al.* [88] built a parallel NTT architecture by employing small integers rather than larger ones. Additionally, the presented solution enhances the performance of Fully Homomorphic Encryption (FHE) by postponing the implementation of modular reduction to later stages in the encryption and re-encryption processes. This approach enabled the execution of each

Table 3. Literature Survey of GPU-based NTT Implementations

Target Application	Work
Post-quantum cryptography	[2,12,37,45,53–55,82,94]
Homomorphic encryption	[9,32,34,39,50,52,58,69,83,85,88,89,92,97]
Zero-knowledge proof	[66]
Hash function	[86]
Attribute-based encryption	[31]
Proxy Re-encryption	[78]
Integer multiplication	[25,26]
Standalone implementation	[13,35]

HE operation in the NTT domain. Kim *et al.* [52] provided a comprehensive evaluation by considering batch size, utilization of GPU threads with register-based high radix implementations, and the use of shared memory during NTT operations. The solution also included a novel on-the-fly twiddle computation method, which reduces memory size without introducing latency by avoiding the need to store all the pre-computed twiddle factors. Ozerk et al. [69] proposed a hybrid approach to optimize the different levels of NTTs. The authors aimed to reduce the dependencies between for-loops in the kernels. Specifically, the solution applied small radix NTTs on a single kernel with inline dedicated functions, while larger radix NTTs used multiple kernels and took advantage of shared memory.

The prior work [31] presented a negative-wrapped NTT implementation on GPU that is employed in the Key-Policy Attribute-Based Encryption (KP-ABE) scheme. Their method followed [32] and utilized a special 64-bit prime that enables efficient twiddle factor multiplication during NTT operation by converting multiplications into left shift operations. The offered method also parallelized NTT by using 4-step NTT which divides a large NTT operation into smaller NTTs [93]. Specifically, the authors divided large NTTs into 64-pt NTT operations to be performed on separate threads in GPU while reducing thread communication overhead. Similarly, [35] used iterative Cooley-Tukey FFT which decomposes a large FFT to multiple smaller FFTs and utilizes tensor cores that can perform small FFTs efficiently.

NTT also found application in Proxy Re-Encryption (PRE) with GPUs [78], where it is utilized to accelerate NTT operations by employing the CRT representation of large integers. The authors utilized shared memory for dynamic polynomial operations and avoided race conditions by utilizing block-level and stream-level synchronizations.

The NTT is also utilized in large integer multiplication with GPUs [25,26]. Chang *et al.* [26] divided large integers into multiple words and applied the NTT to each word to perform multiplication. They split the 4096-point NTT operation into sixty-four 64-point NTTs (64×64-point NTTs), which are later

divided into multiple 8×8-point NTTs. This approach enabled computing the divided NTTs independently by storing twiddle factors in registers. However, the presented solution needs to store twiddle factors of top-level NTT operations in the global memory.

Zero-knowledge proof (ZKP) constructions involve computationally expensive polynomial evaluations such as significantly large-degree NTTs. Recently, hardware and software-based accelerators for NTT are proposed to speed up these expensive computations [66,93]. [66] proposed a parallel NTT implementation on NVIDIA 1080/2080 Ti GPUs for a polynomial-size of 2^{20}.

5 Conclusion

NTT has been a key computational component of next-generation cryptosystems. As such, many techniques have been proposed to implement it in software, and even more, will appear in the near future. This paper provided a quick guide into the NTT world. Our paper intends to be the first stop for anyone who is interested in getting into the software implementation of NTT. A future extension of this work could be to cover other implementation styles including hardware implementations of NTT, in which there are plenty more papers to be analyzed.

Acknowledgments. This paper is supported in part by NSF award no CCF 2146881. Erkay Savaş is supported by the European Union's Horizon Europe research and innovation programme under grant agreement No: 101079319.

References

1. Ajtai, M.: Generating hard instances of lattice problems. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 99–108 (1996)
2. Akleylek, S., Dağdelen, Ö., Yüce Tok, Z.: On the efficiency of polynomial multiplication for lattice-based cryptography on GPUs using CUDA. In: Pasalic, E., Knudsen, L.R. (eds.) BalkanCryptSec 2015. LNCS, vol. 9540, pp. 155–168. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29172-7_10
3. Alagic, G., et al.: Status report on the second round of the NIST post-quantum cryptography standardization process. US Department of Commerce, NIST (2020)
4. Alkim, E., Alper Bilgin, Y., Cenk, M., Gérard, F.: Cortex-M4 optimizations for R, M LWE schemes. IACR Trans. Cryptographic Hardw. Embed. Syst. **2020**(3), 336–357 (2020)
5. Alkim, E., Barreto, P.S.L.M., Bindel, N., Kramer, J., Longa, P., Ricardini, J.E.: The lattice-based digital signature scheme qTESLA. Cryptology ePrint Archive, Report 2019/085 (2019)
6. Alkım, E., Bilgin, Y.A., Cenk, M.: Compact and simple RLWE based key encapsulation mechanism. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 237–256. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30530-7_12

7. Alkim, E., et al.: Polynomial multiplication in NTRU prime: comparison of optimization strategies on cortex-m4. IACR Trans. Cryptographic Hardw. Embed. Syst. **2021**(1), 217–238 (2020)
8. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: 25th USENIX, pp. 327–343 (2016)
9. Alves, P.G.M., Ortiz, J.N., Aranha, D.F.: Performance of hierarchical transforms in homomorphic encryption: a case study on logistic regression inference. Cryptology ePrint Archive (2022)
10. Aysu, A., Patterson, C., Schaumont, P.: Low-cost and area-efficient FPGA implementations of lattice-based cryptography. In: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 81–86 (2013). https://doi.org/10.1109/HST.2013.6581570
11. Azarderakhsh, R., Liu, Z., Seo, H., Kim, H.: Neon PQCryto: fast and parallel ring-LWE encryption on arm neon architecture. Cryptology ePrint Archive, Report 2015/1081 (2015). https://eprint.iacr.org/2015/1081
12. Badawi, A.A., Veeravalli, B., Aung, K.M.M., Hamadicharef, B.: Accelerating subset sum and lattice based public-key cryptosystems with multi-core CPUs and GPUs. J. Parallel Distrib. Comput. **119**, 179–190 (2018)
13. Badawi, A.A., Veeravalli, B., Mi Aung, K.M.: Faster number theoretic transform on graphics processors for ring learning with errors based cryptography. In: 2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), pp. 26–31 (2018). https://doi.org/10.1109/SOLI.2018.8476725
14. Banerjee, U., Chandrakasan, A.P.: Efficient post-quantum TLS handshakes using identity-based key exchange from lattices. In: 2020 IEEE International Conference on Communications (ICC), ICC 2020, pp. 1–6 (2020)
15. Bentahar, K., Silverman, J., Saarinen, M.J.O., Smart, N.: Lash (2006)
16. Boemer, F., Kim, S., Seifu, G., de Souza, F.D., Gopal, V.: Intel HEXL: accelerating homomorphic encryption with Intel AVX512-IFMA52. Cryptology ePrint Archive, Report 2021/420 (2021). https://eprint.iacr.org/2021/420
17. Boorghany, A., Jalili, R.: Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. Cryptology ePrint Archive, Report 2014/078 (2014). https://eprint.iacr.org/2014/078
18. Boorghany, A., Sarmadi, S.B., Jalili, R.: On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. ACM Trans. Embed. Comput. Syst. **14**(3) (2015)
19. Bos, J., et al.: Frodo: take off the ring! practical, quantum-secure key exchange from LWE. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1006–1018 (2016)
20. Bos, J., et al.: Crystals-Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE EuroS&P, pp. 353–367. IEEE (2018)
21. Botros, L., Kannwischer, M.J., Schwabe, P.: Memory-efficient high-speed implementation of Kyber on Cortex-M4. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2019. LNCS, vol. 11627, pp. 209–228. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_11
22. Boyen, X.: Attribute-based functional encryption on lattices. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 122–142. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_8
23. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. Cryptology ePrint Archive, Report 2011/344 (2011)
24. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. Cryptology ePrint Archive, Report 2013/541 (2013). https://eprint.iacr.org/2013/541

25. Chang, B.C., Goi, B.M., Phan, R.C.W., Lee, W.K.: Accelerating multiple precision multiplication in GPU with Kepler architecture. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 844–851 (2016)
26. Chang, B.C., Goi, B.M., Phan, R.C.W., Lee, W.K.: Multiplying very large integer in GPU with pascal architecture. In: 2018 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE), pp. 401–405 (2018)
27. Chu, E., George, A.: Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms. CRC Press (1999)
28. Chung, C.M.M., Hwang, V., Kannwischer, M.J., Seiler, G., Shih, C.J., Yang, B.Y.: NTT multiplication for NTT-unfriendly rings. Cryptology ePrint Archive, Report 2020/1397 (2020). https://eprint.iacr.org/2020/1397
29. de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Efficient software implementation of ring-LWE encryption. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 339–344 (2015)
30. Cousins, D.B., Rohloff, K., Sumorok, D.: Designing an FPGA-accelerated homomorphic encryption co-processor. IEEE Trans. Emerg. Top. Comput. **5**(2), 193–206 (2017). https://doi.org/10.1109/TETC.2016.2619669
31. Dai, W., et al.: Implementation and evaluation of a lattice-based key-policy ABE scheme. IEEE Trans. Inf. Forensics Secur. **13**(5), 1169–1184 (2018)
32. Dai, W., Sunar, B.: cuHE: a homomorphic encryption accelerator library. In: Pasalic, E., Knudsen, L.R. (eds.) BalkanCryptSec 2015. LNCS, vol. 9540, pp. 169–186. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29172-7_11
33. Du, C., Bai, G., Chen, H.: Towards efficient implementation of lattice-based public-key encryption on modern CPUs. In: 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 1230–1236 (2015). https://doi.org/10.1109/Trustcom.2015.510
34. Duong-Ngoc, P., Pham, T.X., Lee, H., Nguyen, T.T.: Flexible GPU-based implementation of number theoretic transform for homomorphic encryption. In: 2022 19th International SoC Design Conference (ISOCC), pp. 259–260 (2022)
35. Durrani, S., et al.: Accelerating Fourier and number theoretic transforms using tensor cores and warp shuffles. In: 2021 30th International Conference on Parallel Architectures and Compilation Techniques, pp. 345–355. IEEE (2021)
36. D'Anvers, J.-P., Karmakar, A., Sinha Roy, S., Vercauteren, F.: Saber: module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2018. LNCS, vol. 10831, pp. 282–305. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89339-6_16
37. Gao, Y., Xu, J., Wang, H.: CuNH: efficient GPU implementations of post-quantum KEM NewHope. IEEE Trans. Parallel Distrib. Syst. **33**(3), 551–568 (2021)
38. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford, CA, USA (2009). aAI3382729
39. Goey, J.Z., Lee, W.K., Goi, B.M., Yap, W.S.: Accelerating number theoretic transform in GPU platform for fully homomorphic encryption. J. Supercomput. **77**, 1455–1474 (2021)
40. Goldreich, O., Goldwasser, S., Halevi, S.: Public-key cryptosystems from lattice reduction problems. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 112–131. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052231
41. Göttert, N., Feller, T., Schneider, M., Buchmann, J., Huss, S.: On the design of hardware building blocks for modern lattice-based encryption schemes. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 512–529. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_30

42. Greconici, D.O.C., Kannwischer, M.J., Sprenkels, D.: Compact Dilithium implementations on Cortex-M3 and Cortex-M4. IACR Trans. Cryptographic Hardw. Embed. Syst. **2021**(1), 1–24 (2020)
43. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 530–547. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_31
44. Güneysu, T., Oder, T., Pöppelmann, T., Schwabe, P.: Software speed records for lattice-based signatures. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 67–82. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_5
45. Gupta, N., Jati, A., Chauhan, A.K., Chattopadhyay, A.: PQC acceleration using GPUs: FrodoKEM, NewHope, and Kyber. IEEE Trans. Parallel Distrib. Syst. **32**(3), 575–586 (2021)
46. Hoffstein, J., Howgrave-Graham, N., Pipher, J., Silverman, J.H., Whyte, W.: NTRUSign: digital signatures using the NTRU lattice. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 122–140. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36563-X_9
47. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ring-based public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054868
48. Howe, J., Prest, T., Apon, D.: SoK: How (not) to design and implement post-quantum cryptography. Cryptology ePrint Archive, Report 2021/462 (2021)
49. Imran, M., Pagliarini, S.: An experimental study of building blocks of lattice-based nist post-quantum cryptographic algorithms. Electronics **9**(11) (2020)
50. Jung, W.: Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs. IACR Trans. Cryptographic Hardw. Embed. Syst. 114–148 (2021)
51. Karabulut, E., Aysu, A.: RANTT: a RISC-V architecture extension for the number theoretic transform. In: 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), pp. 26–32 (2020). https://doi.org/10.1109/FPL50879.2020.00016
52. Kim, S., Jung, W., Park, J., Ahn, J.H.: Accelerating number theoretic transformations for bootstrappable homomorphic encryption on GPUs. In: 2020 IEEE International Symposium on Workload Characterization, pp. 264–275. IEEE (2020)
53. Lee, W.-K., Akleylek, S., Yap, W.-S., Goi, B.-M.: Accelerating number theoretic transform in GPU platform for qTESLA scheme. In: Heng, S.-H., Lopez, J. (eds.) ISPEC 2019. LNCS, vol. 11879, pp. 41–55. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34339-2_3
54. Lee, W.K., Hwang, S.O.: High throughput implementation of post-quantum key encapsulation and decapsulation on GPU for internet of things applications. IEEE Trans. Serv. Comput. **15**(6), 3275–3288 (2021)
55. Lee, W.-K., et al.: Parallel implementation of Nussbaumer algorithm and number theoretic transform on a GPU platform: application to qTESLA. J. Supercomput. **77**, 3289–3314 (2021)
56. Liu, Z., Azarderakhsh, R., Kim, H., Seo, H.: Efficient implementation of ring-LWE encryption on high-end IoT platform. In: Hancke, G.P., Markantonakis, K. (eds.) RFIDSec 2016. LNCS, vol. 10155, pp. 76–90. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62024-4_6

57. Liu, Z., Seo, H., Sinha Roy, S., Großschädl, J., Kim, H., Verbauwhede, I.: Efficient ring-LWE encryption on 8-bit AVR processors. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 663–682. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_33
58. Livesay, N., et al.: Accelerating finite field arithmetic for homomorphic encryption on GPUs. In: IEEE Micro, pp. 1–9 (2023)
59. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 124–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_8
60. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: a modest proposal for FFT hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_4
61. Lyubashevsky, V., Seiler, G.: NTTRU: truly fast NTRU using NTT. IACR Trans. Cryptographic Hardw. Embed. Syst. **2019**(3), 180–201 (2019)
62. Mert, A.C., Karabulut, E., Ozturk, E., Savas, E., Aysu, A.: An extensive study of flexible design methods for the number theoretic transform. IEEE Trans. Comput. **71**, 2829–2843 (2020). https://doi.org/10.1109/TC.2020.3017930
63. Mohsen, A.W., Sobh, M.A., Bahaa-Eldin, A.M.: Performance analysis of number theoretic transform for lattice-based cryptography. In: 2018 13th International Conference on Computer Engineering and Systems (ICCES), pp. 442–447 (2018)
64. Navas, J.A., Dutertre, B., Mason, I.A.: Verification of an optimized NTT algorithm. In: Christakis, M., Polikarpova, N., Duggirala, P.S., Schrammel, P. (eds.) NSV/VSTTE -2020. LNCS, vol. 12549, pp. 144–160. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63618-0_9
65. Nejatollahi, H., Dutt, N., Ray, S., Regazzoni, F., Banerjee, I., Cammarota, R.: Post-quantum lattice-based cryptography implementations: a survey. ACM Comput. Surv. **51**(6) (2019)
66. Ni, N., Zhu, Y.: Enabling zero knowledge proof by accelerating zk-SNARK kernels on GPU. J. Parallel Distrib. Comput. **173**, 20–31 (2023)
67. Oder, T., Pöppelmann, T., Güneysu, T.: Beyond ECDSA and RSA: lattice-based digital signatures on constrained devices. In: 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2014)
68. O'Sullivan, E., Regazzoni, F.: Special session paper: efficient arithmetic for lattice-based cryptography. In: 2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 1–3 (2017)
69. Özerk, Ö., Elgezen, C., Mert, A.C., Öztürk, E., Savaş, E.: Efficient number theoretic transform implementation on GPU for homomorphic encryption. J. Supercomput. **78**(2), 2840–2872 (2022)
70. Peikert, C.: A decade of lattice cryptography. Cryptology ePrint Archive, Report 2015/939 (2015). https://eprint.iacr.org/2015/939
71. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 130–149. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30530-7_7
72. Pöppelmann, T., Güneysu, T.: Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 139–158. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33481-8_8

73. Pöppelmann, T., Oder, T., Güneysu, T.: High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 346–365. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_19
74. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 513–533. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_25
75. Ravi, P., Poussier, R., Bhasin, S., Chattopadhyay, A.: On configurable SCA countermeasures against single trace attacks for the NTT - a performance evaluation study over Kyber and Dilithium on the arm Cortex-M4. Cryptology ePrint Archive, Report 2020/1038 (2020). https://eprint.iacr.org/2020/1038
76. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM (JACM) **56**(6), 1–40 (2009)
77. Reparaz, O., Roy, S.S., Vercauteren, F., Verbauwhede, I.: A masked ring-LWE implementation. Cryptology ePrint Archive, Report 2015/724 (2015)
78. Sahu, G., Rohloff, K.: Accelerating lattice based proxy re-encryption schemes on GPUs. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 2020. LNCS, vol. 12579, pp. 613–632. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65411-5_30
79. Schaumont, P., Aysu, A.: Three design dimensions of secure embedded systems. In: Gierlichs, B., Guilley, S., Mukhopadhyay, D. (eds.) SPACE 2013. LNCS, vol. 8204, pp. 1–20. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41224-0_1
80. Scott, M.: A note on the implementation of the number theoretic transform. Cryptology ePrint Archive, Report 2017/727 (2017)
81. Seiler, G.: Faster AVX2 optimized NTT multiplication for ring-LWE lattice cryptography. Cryptology ePrint Archive, Report 2018/039 (2018)
82. Shen, S., Yang, H., Dai, W., Liu, Z., Zhao, Y.: High-throughput GPU implementation of Dilithium post-quantum digital signature (2022)
83. Shivdikar, K., et al.: Accelerating polynomial multiplication for homomorphic encryption on GPUs (2022)
84. Tan, T.N., Lee, H.: High-secure fingerprint authentication system using ring-LWE cryptography. IEEE Access **7**, 23379–23387 (2019)
85. Türkoğlu, E.R., Özcan, A., Ayduman, C., Mert, A.C., Öztürk, E., Savaş, E.: An accelerated GPU library for homomorphic encryption operations of BFV scheme. In: 2022 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1155–1159 (2022). https://doi.org/10.1109/ISCAS48785.2022.9937503
86. Ulu, M.E., Cenk, M.: A parallel GPU implementation of SWIFFTX. In: Slamanig, D., Tsigaridas, E., Zafeirakopoulos, Z. (eds.) MACIS 2019. LNCS, vol. 11989, pp. 202–217. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43120-4_16
87. Valencia, F., Khalid, A., O'Sullivan, E., Regazzoni, F.: The design space of the number theoretic transform: a survey. In: 2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), pp. 273–277 (2017). https://doi.org/10.1109/SAMOS.2017.8344640
88. Wang, W., Hu, Y., Chen, L., Huang, X., Sunar, B.: Exploring the feasibility of fully homomorphic encryption. IEEE Trans. Comput. **64**(3), 698–706 (2015). https://doi.org/10.1109/TC.2013.154
89. Wang, Z., Li, P., Li, Z., Cao, J., Wang, X., Meng, D.: HE-Booster: an efficient polynomial arithmetic acceleration on GPUs for fully homomorphic encryption. IEEE Trans. Parallel Distrib. Syst. **34**(4), 1067–1081 (2023)

90. Xu, J., Wang, Y., Liu, J., Wang, X.: A general-purpose number theoretic transform algorithm for compact RLWE cryptoprocessors. In: 2020 IEEE 14th International Conference on Anti-counterfeiting, Security, and Identification (ASID), pp. 1–5 (2020). https://doi.org/10.1109/ASID50160.2020.9271722
91. Xu, Z., Pemberton, O., Roy, S.S., Oswald, D.: Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: the case study of Kyber. Cryptology ePrint Archive, Report 2020/912 (2020)
92. Zhai, Y., et al.: Accelerating encrypted computing on Intel GPUs. In: 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 705–716. IEEE (2022)
93. Zhang, Y., et al.: PipeZK: accelerating zero-knowledge proof with a pipelined architecture. In: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), pp. 416–428. IEEE (2021)
94. Zhao, X., Wang, B., Zhao, Z., Qu, Q., Wang, L.: Highly efficient parallel design of Dilithium on GPUs (2022)
95. Zhou, S., et al.: Preprocess-then-NTT technique and its applications to Kyber and NewHope. In: Guo, F., Huang, X., Yung, M. (eds.) Inscrypt 2018. LNCS, vol. 11449, pp. 117–137. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14234-6_7
96. Zhu, Y., Liu, Z., Pan, Y.: When NTT meets Karatsuba: preprocess-then-NTT technique revisited. Cryptology ePrint Archive, Report 2019/1079 (2019)
97. Özcan, A., Ayduman, C., Türkoğlu, E.R., Savaş, E.: Homomorphic encryption on GPU. IEEE Access 1 (2023). https://doi.org/10.1109/ACCESS.2023.3265583