On GPU acceleration of PQC algorithms

Daniel Römer*, Gero Knoblauch*†, Alexander Wiesmaier*†

*Hochschule Darmstadt – University of Applied Sciences
† European University of Technology, European Union

***Abstract*—The rise of quantum computers results in many cryptographic systems being no longer considered sufficiently secure. Algorithms from the field of post-quantum cryptography promise to provide security against the new systems. However, PQC algorithms are generally more computationally intensive than classical cryptography. In order to increase the suitability of PQC for everyday use, this paper investigates their acceleration using GPUs. For this purpose, we analyzed research in the field and closed some gaps with our own implementations. We implemented Dilithium, FrodoKEM, and SPHINCS+ on GPUs using CUDA and benchmarked them together with an existing GPU implementation of Kyber on a Tesla A100 and on a RTX 2070 Super. Dilithium performed convincingly on both GPUs, achieving speed-ups in key generation, signing, and verify by factors of around 820, 2,724 and 2,609 on the A100 and 198, 714 and 802 on the RTX 2070 using the optimal batch sizes. SPHINCS+ achieved speed-ups by factors of around 715, 4,114 and 5,915 on the A100 and 193, 193 and 134 on the RTX 2070. FrodoKEM's key generation, encapsulation, and decapsulation on the A100 were accelerated by factors of 9,989, 4,726, and 3,566. It performed speed-up factors of 107, 108, and 206 on the RTX 2070, respectively. We compared to Kybers acceleration factors of 476, 513 and 1782 on the A100 and 18.5, 17.4 and 184.5 on the RTX 2070. In addition, we investigated the effect of using a variable set of CUDA streams for FrodoKEM. Here, using 8 streams, a speedup of another 2% could be achieved.**

***Index Terms*—Post-quantum cryptography, Key encapsulation mechanism, Digital signature, GPU acceleration, Dilithium, Kyber, SPHINCS+, FrodoKEM.**

I. Introduction

The progressive development of quantum computers, which endanger many cryptographic systems whose security is based on the complexity of the underlying mathematical or algorithmic problems, is a fact that has been known for quite some time now. This applies in particular to integer factorization and the discrete-logarithm problem. To protect against attacks by post-quantum computers, new cryptographic algorithms must be developed, based on other mathematical problems. This emerging area of cryptography is called Post-Quantum Cryptography (PQC). The National Institute of Standards and Technology (NIST) has been reviewing numerous PQC submissions to determine their suitability in terms of security, key length, resource consumption, and resistance to quantum computing [1]. The International Organization for Standardization (ISO) also nominates cryptographic algorithms for standardization, reviews them, and updates them every five years [2].

This research work was supported by the National Research Center for Applied Cybersecurity ATHENE.

However, PQC algorithms tend to have an increased computational overhead and larger key sizes. These create challenges for devices with limited resources such as lifecycle, performance and memory consumption [3] [4] [5]. Our paper analyzes existing research on PQC algorithms on graphical processing unit (GPU)s in terms of performance and resource consumption. Therefore, the most promising of these ISO and NIST-PQC candidates will be identified and examined in more detail in this paper. The focus is on the winners of the third round and fourth round of the NIST-PQC standardization process, as well as the algorithms that will be published by the ISO in future revisions of the encryption algorithms and digital signatures standards. The results of the work should reveal optimization possibilities for PQC algorithms using GPUs. The fact-based evaluation of these optimization options should help make PQC algorithms more attractive for realistic use cases. It should be emphasized which of the NIST and ISO PQC algorithms can be best accelerated by hybrid processing on CPU and GPU. The paper heavily focuses on *batch mode* implementations, while presenting a single comparison to a FrodoKEM implementation in *single mode*. We compare our results with the benchmarks of existing related work.

A. Contributions

The main contributions of the paper are as follows:

(a) Implementations of GPU accelerated variants of PQC signature algorithms (SPHINCS+, Dilithium) and PQC key encapsulation algorithms (FrodoKEM, Kyber) using CUDA [6]. Inspired by previous implementations by Gupta et al. [7].
(b) Comparison of different implementation variants of FrodoKEM in batch mode and multi-batch mode, compared to an existing variant of FrodoKEM single mode by Gupta et al. [7].
(c) Multiple benchmarks on A100 and RTX 2070 Super for all listed algorithms and compare our results on the accelerability of different algorithms. This is used to create a ranking under different aspects.

II. Background

A. Parallel Processing

Due to its large number of cores, a GPU offers the possibility of executing many computing operations at the same time. In order to use this feature to accelerate a program, it must be optimized regarding the specifications of the graphics card used. Two different approaches can be chosen: Single mode processing and batch mode processing. In both cases, it must

be noted that the copying process from and to the graphics card also takes time, depending on the amount of data transferred [7].

1) Single Mode: The CPU and GPU work together in the course of the program to optimize execution time. All computationally intensive tasks that can be accelerated on the GPU due to the number of threads and warps are executed there. Large computing operations are broken down into small subtasks and then processed in parallel. In consideration of the transmission time, the CPU takes over all processes whose outsourcing to the GPU would not bring any added value. The execution time for individual algorithms can be accelerated by fine-grained optimizations [7].

2) Batch Mode: Alternatively, batch mode processing can be considered, which does not break down each step of an algorithm piece by piece and divide it between cores and threads. Instead, any number of instances of an algorithm are executed in parallel. Depending on the number of instances, the full spectrum of the GPU can be used and a gigantic number of operations can be executed per second compared to single mode processing [7].

B. Supported Acceleration Methods

1) Pinned Memory: By default, the host (CPU) works with memory management by page addressing. This form of memory management divides the virtual address space into pages of equal size and the physical address space into page frames of the same size. Graphics cards do not offer the option of directly accessing data from the page address. Temporary data arrays, ”pinned host arrays”, would therefore have to be created for each data access in order to transfer the data to the GPU’s internal memory. This problem can be avoided by directly allocating the host(CPU) arrays in pinned memory [7].

2) Memory Reuse: The allocation and release of memory on the graphics card during execution is a time-consuming process. In addition, it results in fragmentation of the memory area, which can degrade the performance of the system. To avoid this problem as far as possible, all local variables, whose size is already known before execution, can be assigned in advance and transferred to the kernels. Thus, many unnecessary malloc() operations within the kernels are avoided and the execution time of the algorithm is significantly improved by reusing memory space [7].

3) Memory Coalescing: To achieve faster memory reading, the number of total accesses to the GPU global memory should be as low as possible. GPUs offer the possibility to use warps (32 threads) to read from a global memory block with just a single global memory access. The gathered information of all threads is coalesced and then stored in shared memory [8].

4) Threat Divergence: Thread divergence can occur when different threads of a warp take different paths through the code. The GPU has to wait until all threads are done before it can continue with the next instruction. Various techniques can be used to mitigate thread divergence and improve GPU performance. One approach is to optimize the code by restructuring the algorithm. Alternatively, loops can also be unfolded. This principle is known as loop unrolling [9]. In addition, dynamic warp subdivision techniques can be used to handle thread divergences more efficiently. Depending on the path of the thread, a warp is divided into smaller subgroups within the warp to reduce serialization of thread execution [10].

5) Loop Unrolling: Loop unrolling is an optimization technique for loops that aims to improve the performance of programs by reducing loop overhead and increasing parallelism at statement level. Loop iterations are replicated and merged to reduce the number of control instructions and improve the utilization of hardware resources [11].

6) Minimize Latency: If data is transferred between the host and GPU between function calls, this can lead to poorer execution times. To minimize this problem, dedicated streams of the graphics card can be used to transfer data in parallel to the execution of kernels [7]. In addition, static code transformations can be used to optimize memory access patterns, what can be challenging due to the complex memory system and massive parallelism at the thread level [12]. Another approach to optimize transfer times is the use of unified memory [13].

7) Unified Memory: Unified memory is a memory concept that enables the CPU and GPU to operate on a unified memory address space. The underlying principle has previously been used in shared memory, where multiple processors can operate on the same physical memory space, eliminating the need for explicit data transfer between processors [13]. Unified memory simplifies programming by abstracting memory management and reducing the need for manual data transfers between CPU and GPU. However, the reduced complexity of memory management should not be used blindly, as unified memory can also lead to performance losses depending on the application.

C. Acceleration Inside the CUDA Kernel

There are various mathematical functions that can be optimized within CUDA kernels of a single-mode implementation. In this section, only a few of the most concise possibilities are shown, that are important for acceleration of PQC algorithms. These include the NTT, which is used for the multiplication of polynomials in MLWE-based methods. This can be accelerated by a smarter arrangement of the threads of the butterfly pattern combined in each level, as unnecessary memory accesses on the lower levels are avoided. Figure 1 serves as an illustration.

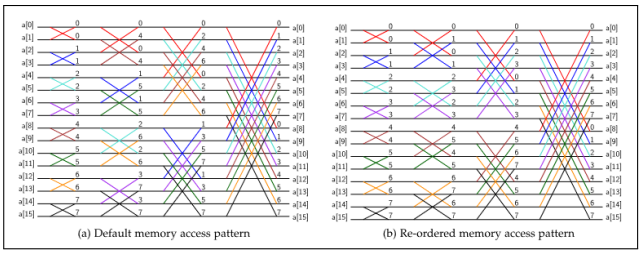


Fig. 1. Butterfly Pattern as depicted in [7]

A comparable optimization for NTT in Dilithium is presented in [14]. This is firmly anchored in the code depending on the warps per block, for the variants 1 warp and 4 warps, to solve the 8-level NTT or INTT more efficiently. The parameters 1 warp and 4 warps per block were selected as

they achieve the highest theoretical utilization of the threads, considering the Dilithium parameters.

Another optimization option would be matrix multiplication, which is common in all LWE methods. One possibility is the parallel partial calculation of the matrix in threads. However, this entails the problem that different optimizations must be performed for matrices of different sizes, since there is a direct dependency [7]. Better optimization is possible for MLWE, as the matrix is already saved as a polynomial NTT format. The multiplication vector is also transformed into the NTT space to use effective calculations on polynomials. Then it is transformed back into the normal number space for further calculations [14].

For the hash-based SPHINCS+ algorithm, binary trees of height h can be split into smaller binary trees to reduce the required memory and process the trees directly at thread level in parallel without generating thread divergences. The one-time signature scheme WOTS with its tree height of 7 can be parallelized according to the same principle [15].

Since pseudo-random values must be generated for all PQC algorithms to ensure the security of the generated keys, cryptographically secure and standardized algorithms such as Shake256 are used. In [16], these are accelerated by combining functional steps that actually run one after the other in a kernel, as they work on the same data and thus reduce memory accesses.

The final and most important point is that the CUDA kernel can be optimally configured for any operation using fine-grained benchmarking. This is presented in [16] and [17] and has a major impact on the execution times of single- and batch-mode implementations on GPUs.

III. Related work

Chanet et al. researched a high-throughput acceleration variant for NIST Authenticated encryption with associated data (AEAD) via GPU. They show experimental results that all NIST AEAD finalists can achieve high throughput (up to 111.53M AEAD per second), approximately 142.19% improvement compared to the unoptimized GPU version [18].
Shenet et al. researched a high-throughput variant for Dilithium. Across all three security levels, their GPU implementation achieves over 160× speedups for signing and over 80× speedups for verification on commercial and server-grade GPUs [19].
Kim et al. investigated parallel implementations of SPHINCS+ on different NVIDIA GPU architectures. Their implementation could achieve 44,391(resp. 24,997 and 11,401) signature generations, 725,118(resp. 354,309 and 100,168) key generations, and 285,680(resp. 155,800 and 106,280) verifications per second at security level 1(resp. 3 and 5) on RTX3090 [20].
Speedups for FrodoKEM could be achieved by Lee et al. with their dot product instructions approach. They tested on V100 GPU and achieved a 4.37× higher throughput for FrodoKEM [21].
There are also the mentioned implementations of Falcon and Mikita in VIII-B. Their Falcon-512 implementation achieves 58,595 signatures/second and 2,721,562 verifications/second on an A100 GPU. Mitaka implementation achieves 161,985 signatures/second and 1,421,046 verifications/second on the same GPU [22].
Furthermore, interesting news coming from NVIDIA. They introduced a new SDK for developers to ease the transition from current crypto systems to PQC protocols, which also includes an easy, flexible, and GPU-accelerated implementation of NIST-approved PQC operations. Using a single NVIDIA H100 SXM5 GPU they achieved throughputs, including up to 13.3 million key generations, 9.3 million encapsulations, and 8 million decapsulations per second for the batched NIST-approved PQC algorithm known as ML-KEM-768 [23]. Since the Kyber specification, ML-KEM is the name created for the algorithm by NIST [24].
Wu et al. [25] published a new variant of a CUDA-based Batch Parallel optimization of SPHINCS+ (CBPSPX), which was tested on RTX 3090 and RTX 4090 and compared to previous implementations of Sun et al. [15], Wang et al. [26] and Kim et al. [20]. They achieved speed-ups of 1874x, 1682x, and 1646x on RTX3090 compared to the reference implementation of SPHINCS+. This improves over previous implementations by 1.4x to 2.5x for signature generation and 4.6× to 11.3× for signature verification.
Chen et al. [27] published a new variant of an optimized GPU-based acceleration of CRYSTALS-Dilithium, which was tested on a RTX 4090 and an A100 and compared to previous implementations of Seo et al. [28] , Becker et al. [29] and Zhao et al. [30]. They achieved speed-ups of 114x, 158x, and 118x on the A100 compared to the reference implementation of Dilithium3. They achieved even better speed-ups on the RTX 4090, with 143×, 249× and 143×, respectively.

IV. GPU Implementations

We compared and tested four algorithms, two KEMs and two signature algorithms. The GPU implementation of Kyber is described in [7]. Inspired by Kyber, the algorithms Dilithium, FrodoKEM and SPHINCS+ got a batch mode variant as well. Since Dilithium and Kyber both belong to the category of lattice-based crypto schemes, based on MLWE, and are part of the CRYSTALS Cryptographic Suite, there are many similarities in the structure and implementation of these operations [31]. Although FrodoKEM is also a lattice method, it is based on simple LWE and stands out in comparison to Kyber because of its enormous ciphertext size. The last PQC algorithm considered, SPHINCS+, belongs to hash-based methods and is based on one-way functionalities with fixed or variable output size of SHA512 and SHAKE256 in combination with tree structures. All algorithms implemented in this work for execution on GPUs are based on the *batch-mode processing* presented in chapter II using a single CUDA stream.
A multistream variant was also implemented for FrodoKEM in order to be able to prove or disprove the positive effect of splitting into several CUDA streams.
To further improve execution times, all variables whose memory requirements were already determined before execution

were already initialized. The same applies to temporary variables that were allocated and released within functions in the CPU reference implementation.
The memory generated in this way fulfills the requirement of permanently allocated host memory (pinned memory) from chapter II and can therefore be used without further processing by the graphics card. The variables, which are stored in a value set, can also be used for signing and verification or encryption and decryption after key generation, which fulfills the requirement of memory reuse. Furthermore, apart from small arrays of up to 26 bytes, no further memory is allocated during execution. This means that there is less fragmentation of the memory area, which further improves access times.

A. Measurements

CUDA offers events to measure the time of GPU implementations. These offer the possibility of recording the time between two events or letting the system wait with the execution of code until a certain event is completed. Each measurement created as part of this work gives the average of $1,000$ measurements, each with the same parameters. The generated measurement data contains information such as batch size and all measured times between events. These are divided into the total time, time of the copy process and time consumption of the three sub-algorithms. In addition, the executions per second for the overall process and the three sub-algorithms are calculated and output using the formula from 1. When looking at the measurement data, it must therefore always be noted that the specified times do not apply to the execution of one iteration of the algorithm, but to the complete batch size.

$$\text{Operations/s} = \frac{\text{batchsize} * 1000}{t_total} \tag{1}$$

The single-mode implementation from [7] uses the C framework from Stebila [32] for time measurements. In addition, the internal measurement of the speed tests of the OpenQuantum-Safe library [33] was relied on for the measurements of the CPU reference implementations.

V. Evaluation on platforms

In this section, the generated measurements are analyzed and evaluated in more detail. For that, the test systems used are presented at the beginning and the software used is listed. The measured values for the four PQC algorithms, Kyber, Dilithium, FrodoKEM and SPHINCS$^+$, are presented graphically and explained in more detail. The focus is initially on the different test cases within the algorithms.
Finally, algorithms are evaluated within the categories of KEM and digital signature to make statements about their capabilities and capabilities. The various use cases in which the algorithms would offer advantages are also discussed.

A. Test systems

This section briefly describes the hardware used for the two test systems. The software used is also explained and its selection justified.

	Test system 1	Test system 2
CPU	(16x) AMD EPYC - Milan 2795 Mhz 1 Core 1 processor	Intel Core i7 – 10700K 3792 MHz 8 Cores 16 processors
GPU	NVIDIA TESLA $A100$ 6912 Cores 80 GB internal memory 5120 bit bus width 1,935 GB/s bandwidth Ampere architectur	NVIDIA GeForce RTX 2070 SUPER 2560 Cores 8 GB internal memory 256 bit bus width 448.0 GB/s bandwidth Turing architectur
RAM	128 GB	32 GB

TABLE I
Technical specifications of the test systems 1 & 2

The first test system is a virtual machine (VM) on an Openstack cluster at Darmstadt University of Applied Sciences. The VM is equipped with 128 GB RAM and 16 AMD EPYC - Milan processors. An NVIDIA A100 Tensor Core GPU is used as the GPU. This was specially developed for high-performance data centers that deal with the areas of artificial intelligence, data analysis or high-performance computing. Windows Server 2022 (21H2) is used as the operating system. In order for the VM to correctly recognize and use the assigned GPU after initialization, the correct NVIDIA driver had to be manually added. In this case, it is the *NVIDIA Data Center Driver for Windows* - Version 537.13.
The second test system is used primarily to generate comparative data to validate the measurements in test system 1. This is a regular Windows 10 (22H2) workstation. This is equipped with an Intel Core i7 – 10700K CPU, 32 GB RAM and an NVIDIA GeForce RTX 2070 SUPER GPU. The GPU uses the *GeForce Game Ready Driver* - Version 537.43. In the remainder of this chapter, the two systems are only referenced as test system 1 and test system 2, as shown in table I.
In addition to standard tools such as *CMake* and *GIT*, *Visual Studio* 2022 was used as the development environment and C/C++ as the programming language. The great advantage of Visual Studio is the seamless integration of the *CUDA Toolkit* from NVIDIA. The version 12.2 of CUDA was used.

B. Analysis of the Measurement Data

Four measurement series are shown in all diagrams. Regardless of the type of algorithm, the number of complete runs of all three sub-steps of the algorithm is always shown. This also includes the time that the system needs to transfer the data to and from the GPU. This is shown in blue. Another series of measurements that applies to all algorithms is key pair generation, which is shown in green. The red measurement series indicates the encapsulation for KEMs and the signing process for DSAs. The last series of measurements applies analogously to decapsulation and signature verification and is shown in orange.

C. Key Encapsulation Mechanisms

This section takes a closer look at the measurement data from the two KEM methods, Kyber and FrodoKEM.
The measurements for FrodoKEM are particularly relevant to the topic, as not only the CPU reference implementation

Test system 1					
Algorithm		Total/s	KeyGen/s	Encap/s	Decap/s
Kyber-768	ref	1,024	3,104	2,876	2,845
	gpu batch	461,621	1,479,420	1,477,498	5,071,705
	speedup	(x451)	(x476)	(x513)	(x1,782)
	batchsize	(35,000)	(35,000)	(35,000)	(35,000)
FrodoKem-976	ref	10.1	26.3	35.3	34.8
	gpu batch	43,643	262,727	166,833	124,097
	speedup	(x4,321)	(x9,989)	(x4,726)	(x3,566)
	batchsize	(25,000)	(20,000)	(25,000)	(25,000)
	gpu single	5.8	18.3	17.6	16.1
	speedup	(x0.57)	(x0.7)	(x0.5)	(x0.46)

Test system 2					
Algorithm		Total/s	KeyGen/s	Encap/s	Decap/s
Kyber-768	ref	1,254	3,717	3,743	4,179
	gpu batch	31,272	68,890	65,267	771,226
	speedup	(x24.9)	(x18.5)	(x17.4)	(x184.5)
	batchsize	(60,000)	(60,000)	(60,000)	(40,000)
FrodoKem-976	ref	13.6	34.7	45.3	45.7
	gpu batch	1,691	3,701	4,899	9,426
	speedup	(x124)	(x107)	(x108)	(x206)
	batchsize	(5,000)	(5,000)	(5,000)	(5,000)
	gpu single	302.4	827	986	922.5
	speedup	(x22.2)	(x23.8)	(x21.8)	(x20.2)

TABLE II
COMPARISON OF KEMS WITH CPU REFERENCE IMPLEMENTATION

and batch mode were tested here. In addition, a single-mode FrodoKEM implementation by Gupta et al. [7] and an extension of the batch mode to an arbitrary number of CUDA streams to investigate the influence of split processing of data on the execution speed. In addition, the method presented in [7] for optimizing the block size as a function of GPU and batch size is tested.

1) FrodoKEM:

a) Batch-mode: Figures 2 and 3 show the measurement results of the batch-mode implementation of FrodoKEM with a block size of 64. Unfortunately, it was only possible to measure up to a batch size of 35,000, as the high memory requirements of the structures used in FrodoKEM meant that the maximum memory was already utilized within one execution. This amounts to $0x7fffffff$ bytes, which corresponds to $2^{31} - 1$ bytes or 2,148 megabytes. The measured values shown in the figures 2 and 3 can be taken from the tables XIII and XIV, which can be found in the appendix. Measurements up to 200,000 were listed there.

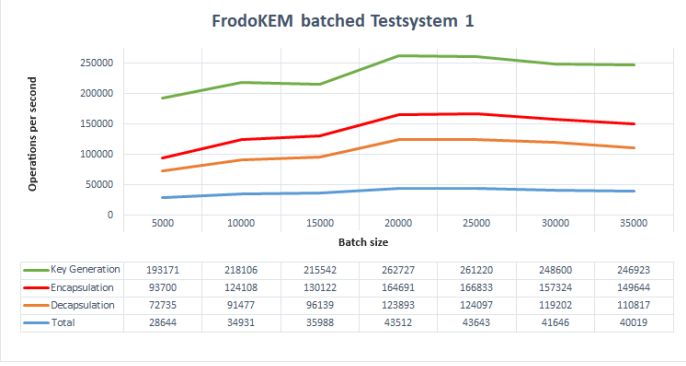


Fig. 2. Measurements FrodoKEM-batched on test system 1 with blocksize 64

In 2 you can see that all four measurement series on the test system 1 follow the same course with varying batch sizes. The highest throughput of operations per second is achieved in the range of batch size $20 - 25,000$, which can be seen at the zenith of the measurement series and is confirmed by the data in table II. From table II you can also see the speedups that could be achieved by the batch-mode implementation compared to the CPU reference implementation. Key generation was accelerated by a factor of $\sim 10,000$, while $\sim 4,700$ was achieved for encapsulation and $\sim 3,600$ for decapsulation. If the data transfer is included in the calculation, it can be seen that the entire FrodoKEM process, with a batch size of 25,000, is ∼4,300 times faster than the reference implementation. It is important to note that at this batch size, almost 22% of the total time is spent on copying the data between the host and GPU.

For batch sizes above 25,000, it can be seen that the GPU is already busy and can no longer enable faster processing. As the amount of data to be copied increases continuously, the overall process is additionally slowed down. In addition, the block size of FrodoKEM for test system 1 was optimized for small batch sizes according to the principle presented in [7]. As this optimization is carried out using thousands of independent measurements and took several hours even for small batch sizes, it was not carried out for all the batch sizes shown. Instead, it was optimized for the batch sizes 128, 256, 512 and 1024 and the average result was applied. With the help of these optimized block sizes, execution times could be achieved for the specified batch sizes that are $82 - 103\%$ faster than the previous measurement with block size 64. However, this positive effect of the adjusted block size ends as soon as a batch size of 15,000 is exceeded.

The measurement results with the optimized variant are shown in XV in the appendix and are compared with the previous measurements in XVII.

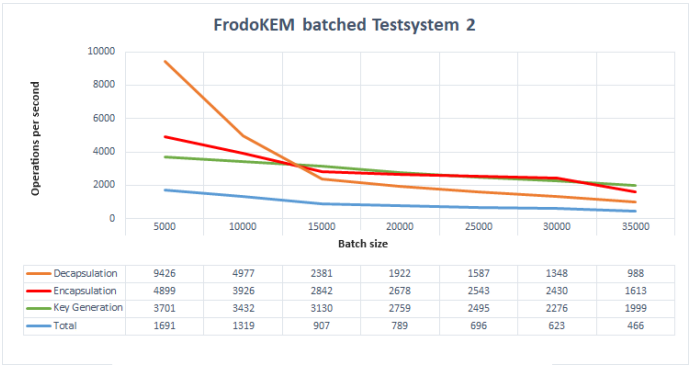


Fig. 3. Measurements FrodoKEM-batched on test system 2 with blocksize 64

If you compare the results from Figure 2 with those from Figure 3, you see a different picture. The optimal batch size is already reached at around 5,000. Even though this figure only starts at batch size 5,000, you can see the previous measured values in XIV.

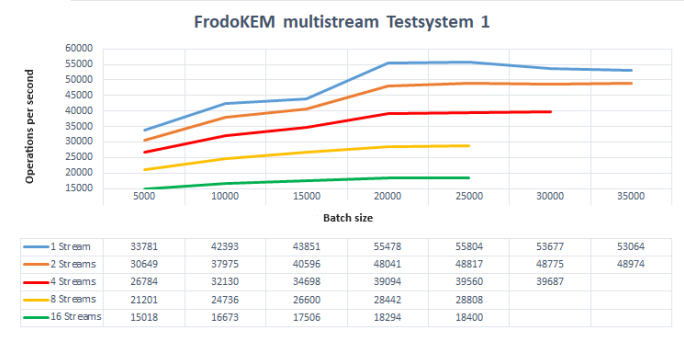
This table also shows that FrodoKEM only uses 2% of the time required to copy the data from and to the GPU

when executing all three sub-algorithms. As the batch size increases, the throughput for decapsulation in particular deteriorates rapidly and already falls significantly below that of key generation and encapsulation at $15,000$.
Especially in comparison with the CPU reference implementation, it can be seen that test system 2 is a normal workstation that cannot achieve the same performance as test system 1. The entire execution could only be accelerated by a factor of 124, as shown in table II. For key generation, encapsulation and decapsulation, this means speed-ups by a factor of 107, 108 and 206 respectively.

b) Multistream Batch-mode: These measurements are intended to illustrate the effect that the use of more than one CUDA stream has on the execution time. More detailed measurement data of the figures 4, 5, 6 and 7 can be taken from the measured value tables XIX, XX, XXI and XXII.
The graphs have different lengths, as the maximum batch size decreases as the number of CUDA streams increases. This can be attributed to the overhead caused by the segmentation of the data transmission. As already shown in the batch-mode measurement with a stream in 2, the best execution times are again achieved with a batch size of $20,000 - 25,000$.

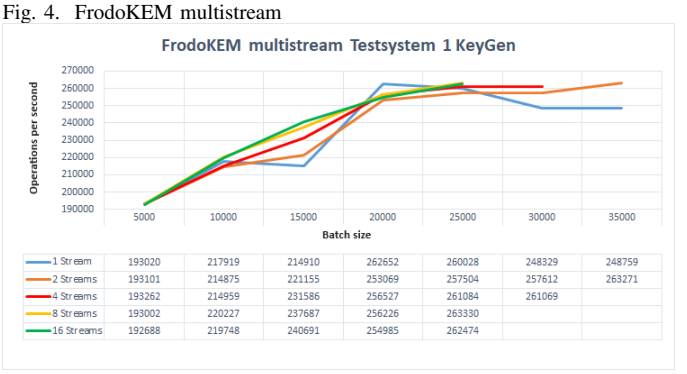
When looking at the diagrams for key generation, encapsulation and decapsulation, it can be seen that the most operations per second for all batch sizes are achieved when using 8 parallel CUDA streams. However, these three measurements do not consider the time taken to copy the data to and from the GPU. The data is transferred before key generation and after decapsulation and is therefore neglected for the measurements of the partial algorithms. The overall picture is shown in Figure 4, taking the copying process into account. There, the variant with one stream performs best. The more streams are used, the worse the execution time of the entire program sequence becomes. If the associated data from XVIII is considered, it can be seen that the data transfer for 16 streams takes up to 67% of the total program runtime. The multistream implementation of FrodoKEM used in this work does not yet fully utilize the capabilities of CUDA and GPUs. The execution of the kernels themselves could be accelerated by using more streams. However, the overhead caused by the data transfer to multiple streams in FrodoKEM cannot be negated by the faster execution times. Based on the results so far, a FrodoKEM variant could be implemented that uses 8 CUDA streams and combines and parallelizes the data transfer with the remaining kernel calls.

c) Single mode: The measurement results of the single-mode implementation from [7] are shown in table II. When looking at the measurements taken on test system 1, it is noticeable that the single-mode implementation there is slower than the CPU reference implementation. This contradicts the results obtained in [7] and cannot be explained without a detailed analysis of the execution on test system 1. As the GPU used is significantly more powerful, it is reasonable to assume that the 16 CPUs used on test system 1 are not working together optimally.
The same measurement on test system 2 resulted in



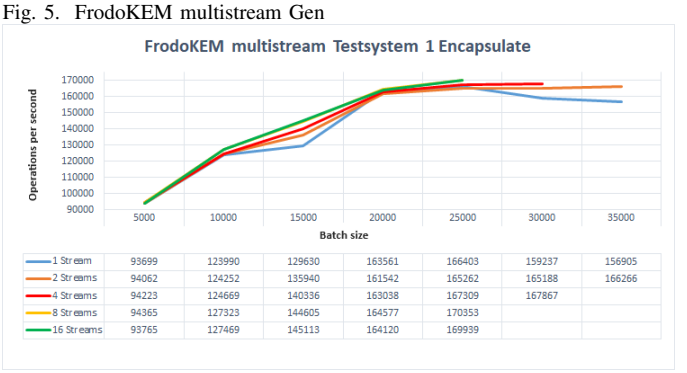
1 Stream	33781	42393	43851	55478	55804	53677	53064
2 Streams	30649	37975	40596	48041	48817	48775	48974
4 Streams	26784	32130	34698	39094	39560	39687	
8 Streams	21201	24736	26600	28442	28808		
16 Streams	15018	16673	17506	18294	18400		

Fig. 4. FrodoKEM multistream



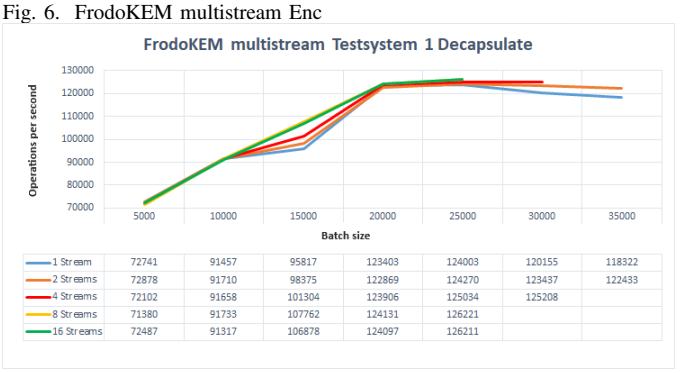
1 Stream	193020	217919	214910	262652	260028	248329	248759
2 Streams	193101	214875	221155	253069	257504	257612	263271
4 Streams	193262	214959	231586	256527	261084	261069	
8 Streams	193002	220227	237687	256226	263330		
16 Streams	192688	219748	240691	254985	262474		

Fig. 5. FrodoKEM multistream Gen



1 Stream	93699	123990	129630	163561	166403	159237	156905
2 Streams	94062	124252	135940	161542	165262	165188	166266
4 Streams	94223	124669	140336	163038	167309	167867	
8 Streams	94365	127323	144605	164577	170353		
16 Streams	93765	127469	145113	164120	169939		

Fig. 6. FrodoKEM multistream Enc



1 Stream	72741	91457	95817	123403	124003	120155	118322
2 Streams	72878	91710	98375	122869	124270	123437	122433
4 Streams	72102	91658	101304	123906	125034	125208	
8 Streams	71380	91733	107762	124131	126221		
16 Streams	72487	91317	106878	124097	126211		

Fig. 7. FrodoKEM multistream Dec

FrodoKEM running 22 times faster on average, although the reference values are also higher here thanks to the more powerful CPU. This proves the functionality and correctness of the single-mode CPU implementation. In the case of test system 2, depending on the sub-algorithm, it can cover $10 - 20\%$ of the operations per second that are possible with

the batch-mode variant. However, it must be borne in mind that the batch-mode variant can be scaled much more easily.

2) Kyber: It has been shown that the data structures used in Kyber have the lowest memory requirements of the algorithms considered in this paper. Therefore, measurements up to a batch size of $\sim 700,000$ could be carried out here. For a better overview, however, the same scale was used in the diagrams 8 and 9 of the measurement series of both test systems as for Crystals-Dilithium, which is limited to $65,000$. This also applies to the data in table II.
The figures are based on Tables IV and V, which can be found in the appendix. Measurements up to $200,000$ were listed there. Figure 8 shows that the measurement series of key generation, encapsulation and decapsulation created on test system 1 are still on an upward trend even at batch size $65,000$. Key generation and encapsulation are so close to each other with their operations per second that they can hardly be distinguished. While these two measurement series reach their maxima of $\sim 2,200,000$ and $\sim 2,300,000$ at a batch size of $220,000$, the zenith for decapsulation is already reached at batch size $135,000$ with $\sim 7,500,000$ operations per second. The entire Kyber algorithm, taking into account the data transfers, moves from batch size $130,000$ at $\sim 605,000 \pm 10,000$ operations per second. The values no longer changed significantly in the random measurements for all other batch sizes in $100,000$ increments for data verification.

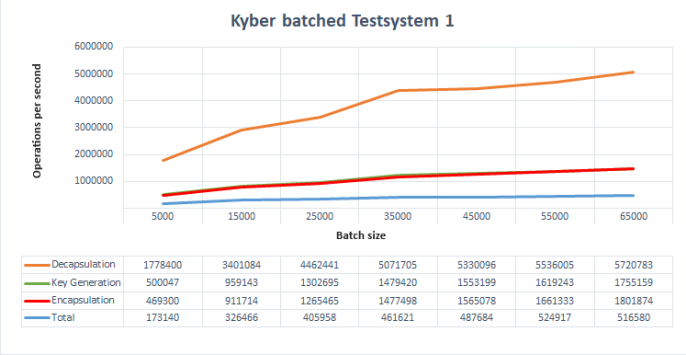


Fig. 8. Measurements Kyber-batched on test system 1 with blocksize 64

The measurements up to batch size $65,000$ already show an acceleration by a factor of 451 for the execution of Kyber compared to the CPU reference implementation. Similarly, key generation is 476 times faster, encapsulation 513 times faster and decapsulation $1,782$ times faster.
The block size was also optimized for Kyber using the framework from [7]. Thanks to the fast execution times, the batch sizes 1024, 4096, 16384 and 32768 could be used here. The measured values with optimized grid size are shown in table VI and are compared in VII and VIII with the measurement series shown here.
The evaluations show that an acceleration of 13% could be achieved for the batch size 16384. However, almost all other batch sizes were slowed down as a result. At second glance, it can be seen that the values optimized for smaller batch sizes have a positive effect on the execution times of the sub-algorithms for large batch sizes over $135,000$. However, the acceleration is only in the single-digit percentage range.

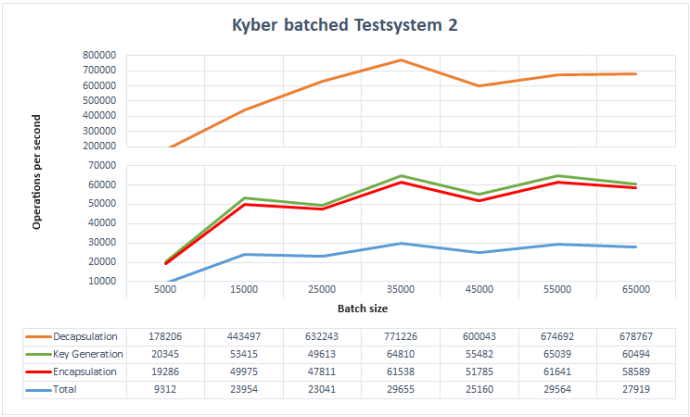


Fig. 9. Measurements Kyber-batched on test system 2 with blocksize 64

Figure 9 shows that the measurement series on the scale created from Dilithium ends with a downward trend. However, the fluctuation between different batch sizes already seen here continues. Meanwhile, the operations per second of all measurement series continuously approach their maximum. For all measurement series, this is $\sim 7-8\%$ above the values given in table II for the batch sizes up to $65,000$.
For key generation and encapsulation, the maximum throughput of $\sim 69,000$ and $\sim 65,000$ operations per second was achieved in Figure 9 with a batch size of $60,000$. This corresponds to an acceleration by a factor of 18.5 and 17.4 compared to the CPU reference implementation. For decapsulation, execution was accelerated by a factor of 184.5 with $770,000$ operations per second. This leads to an almost 25 times faster execution on test system 2 when considering the complete Kyber algorithm. Due to the low memory requirements and the fast execution times, Kyber does not require a strong GPU to cover realistic scenarios with a parallelized implementation.

D. Digital Signature Algorithms

In this section, the focus is on the measurements of the DSAs Dilithium and SPHINCS$^+$. These build on different ideas of lattice-based and hash-based cryptography.

1) Dilithium: Although Dilithium, like Kyber, is based on MLWE, the data structures used during execution time take up significantly more memory. This is shown by the maximum batch size of $65,536$ compared to $\sim 700,000$ for Kyber.

The diagrams in 10 and 11 are based on the measured value tables IX and X. As can be clearly seen in 12, all test series on test system 1 are still on an upward trend when the maximum batch size is reached. The GPU used is not yet fully utilized and could process more operations per second, although $\sim 37\%$ of the time required by the overall algorithm is already used for data transfer.

The values for Dilithium on test system 1 shown in Table III support this thesis. The key and signature generation are both in the range of $\sim 1,160,000$ operations per second at batch size $65,000$. Table III shows, however, that this only

Test system 1					
Algorithm		Total/s	KeyGen/s	Sign/s	Verify/s
Dilithium 3	ref	267	1,422	425	1,522
	gpu	321,748	1,165,697	1,157,606	3,970,451
	speedup	(x1,205)	(x820)	(x2,724)	(x2,609)
	batchsize	(65,000)	(65,000)	(65,000)	(65,000)
SPHINCS 192f	ref	2.16	60.74	2.39	44.72
	gpu	7,610	43,435	9,834	264,500
	speedup	(x3,523)	(x715)	(x4,114)	(x5,915)
	batchsize	(40,000)	(40,000)	(40,000)	(40,000)

Test system 2					
Algorithm		Total/s	KeyGen/s	Sign/s	Verify/s
Dilithium 3	ref	320	1,659	508	1,814
	gpu	118,301	328,601	362,612	1,454,710
	speedup	(x370)	(x198)	(x714)	(x802)
	batchsize	(15,000)	(10,000)	(25,000)	(25,000)
SPHINCS 192f	ref	2.8	78.54	3.04	56.81
	gpu	497	15,155	588	7,629
	speedup	(x178)	(x193)	(x193)	(x134)
	batchsize	(40,000)	(40,000)	(40,000)	(10,000)

TABLE III
COMPARISON WITH CPU-REFERENCE IMPLEMENTATION OF DIGITAL SIGNATURE-ALGORITHMS

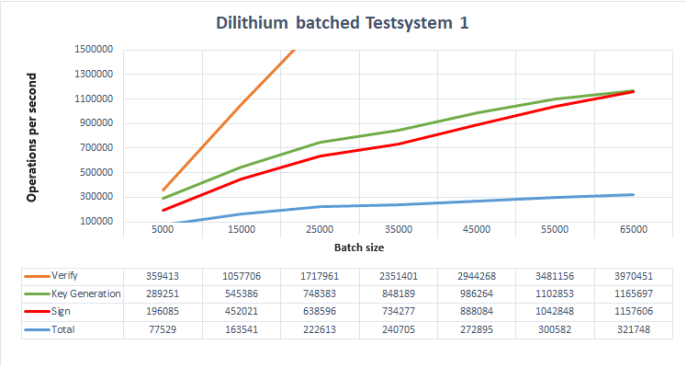


Fig. 10. Measurements Dilithium-batched on test system 1 with blocksize 64

represents an acceleration of 820 times compared to the CPU reference implementation for the key generation, while the signature generation is accelerated by a factor of $\sim 2,700$. Signature verification is also accelerated by a similar factor. At batch size $65,000$, almost $4,000,000$ signatures are verified per second.

Dilithium was also optimized in terms of block size and grid size using the framework from [7]. The measurements with the new parameters are shown in table XI and are compared with the previous results of test system 1 in XII. As with Kyber, the batch sizes 1024, 4096, 16384 and 32768 were used for the optimization.
The new parameters lead to accelerations of up to 6% for batch sizes smaller than $32,728$. However, the influence on the signature verification measurement series is negligible. As the batch size increases, the positive influence of the new block size diminishes. Despite the negative effect, however, the change is not significant at well under 1%.

On test system 2, the maximum batch size is again reached relatively early, as shown in 11. At a batch size of $15,000$, key generation performs around $323,000$ operations per second and then becomes steadily slower. Signature generation and verification reach their zenith at a batch size of $25,000$. While $\sim 360,000$ messages per second can be signed, up

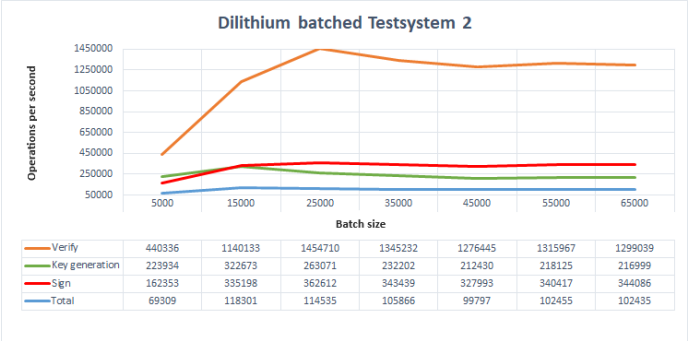


Fig. 11. Measurements Dilithium-batched on test system 2 with blocksize 64

to $\sim 1,454,000$ signatures per second are verified, which corresponds to an acceleration by the factors 714 and 802 compared to the CPU reference implementation. The table III also shows that 198 times as many key pairs can be generated. For the entire course of the Dilithium program, a speed-up of 370 times is achieved, taking into account the data transfer. If you look at the complete spectrum of measurements, you can see that the throughput remains virtually constant from a batch size of 25.00 due to the trade-off between quantity and data transfer.

2) SPHINCS$^+$: SPHINCS$^+$, as a hash-based PQC algorithm, is the first method in this paper that works in a fundamentally different way than the usual algorithms considered. The measurements discussed in this section are based on the SPHINCS$^+$-192f parameter set from [34]. The *fast* variation of SPHINCS$^+$ promises faster execution times at the cost of increased memory requirements.
The figures 12 and 13 are based on the tables XXIII and XXIV. To better visualize the progression of the measurement series, the measurement spaces were divided in the display and scaled individually for the sake of readability. For SPHINCS$^+$, measurements up to a batch size of $50,000$ could be realized before the maximum memory to be used was exceeded.

In 12 it can be seen that the optimum execution times on test system 1 are achieved for all measurement series with a batch size of $40,000$. Up to $\sim 44,000$ key pairs are generated per second. As can be seen in Table III, this accelerates the originally fastest sub-algorithm of SPHINCS$^+$ by a factor of 715. For the slowest operation of the CPU reference implementation, the signature generation, up to $\sim 10,000$ operations per second can be achieved in batch mode, which corresponds to an acceleration by a factor of $4,114$. This finding is particularly significant as the generation of signatures takes the most time when using SPHINCS$^+$. Signature verification scales best with the GPU and, with $\sim 265,000$ operations per second, can achieve a speed-up of $5,915$.

Using the optimization framework from [7] proved difficult for SPHINCS$^+$. The reason for this is the extensive kernels used, which combine up to 10 time-consuming functionalities in one call. Therefore, only optimizations for the batch sizes

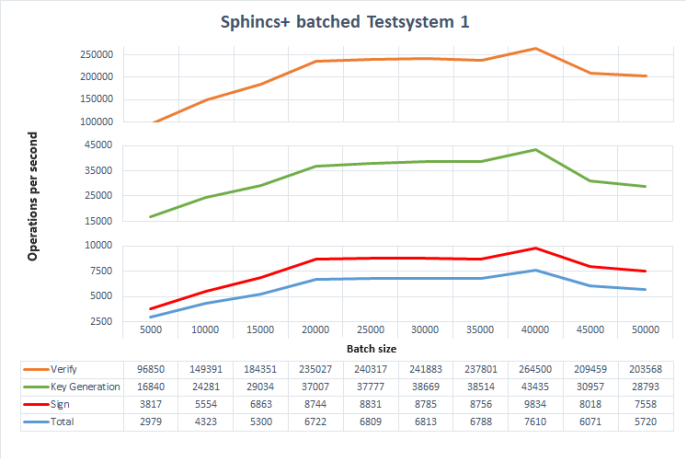


Fig. 12. Measurements SPHINCS$^+$-batched on test system 1 with blocksize 64

128 and 256 were carried out here, whereby the complexity of the framework was also reduced by a factor of 10. Nevertheless, test system 1 required around 48 hours for the optimization. For better optimizability, the largest kernels could be treated as host functionality, making the sub-steps more accessible in custom kernels.

The measurement results with optimized parameters are shown in table XXV and are compared with the previous results in XXVI. Although the optimization for the key generation achieves an added value of almost 80%, the overall throughput decreases by almost the same factor. This is due to the negative impact of optimization on signature generation and signature verification. Accordingly, an optimization of SPHINCS$^+$ by this framework, without a fundamental change of the code, is currently not meaningful.

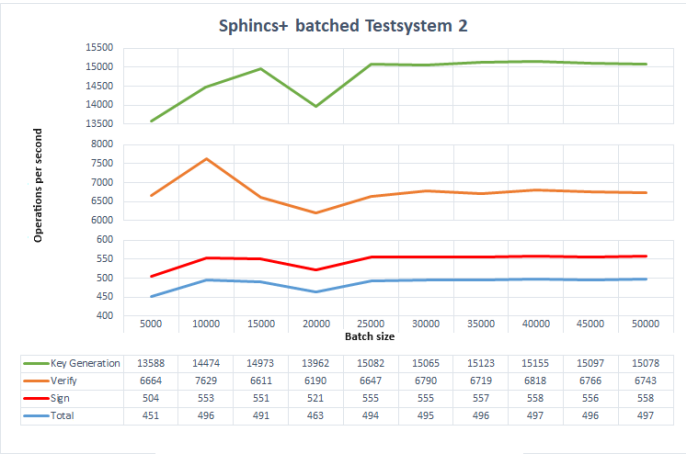


Fig. 13. Measurements SPHINCS$^+$-batched on test system 2 with blocksize 64

As on the first test system, the measurement series of test system 2 in Figure 13 show a uniform trend. As before, three different Y-scales were created here for clarification. The underlying data from table XXIV confirms this picture. Signature verification already reaches its maximum of $\sim 7,600$ operations per second at batch size $10,000$, which corresponds to an acceleration of 134 times compared to the reference implementation. The usual measurement series only reach their zenith at a batch size of $40,000$. $15,155$ generated keys per second correspond to an acceleration by a factor of 193. Signature generation is accelerated to the same extent to 588 operations per second. For the entire SPHINCS$^+$ algorithm, this results in an acceleration by a factor of 178 to almost 500 operations per second on test system 2, taking data transmission into account.

VI. Comparison and Evaluation of the Measurements

This section is used to map the results already presented from the previous chapter. For this purpose, data from the tables II and III were combined in the figures 14 and 16. They represent the acceleration factor of the operations executed per second compared to the respective CPU reference implementation. As in the previous figures in this chapter, four measurement series are shown in the figures. *Total* again contains the entire program sequence of the algorithm, taking into account the time elapsed for the data transfer. The key generation is self-explanatory from the name. Since KEM and DSAs are shown in the figures, the sub-algorithms of the different methods are shown here in combination. For Kyber and FrodoKEM, the two measurement series stand for encapsulation and decapsulation, while for Dilithium and SPHINCS$^+$ they describe signature generation and signature verification.

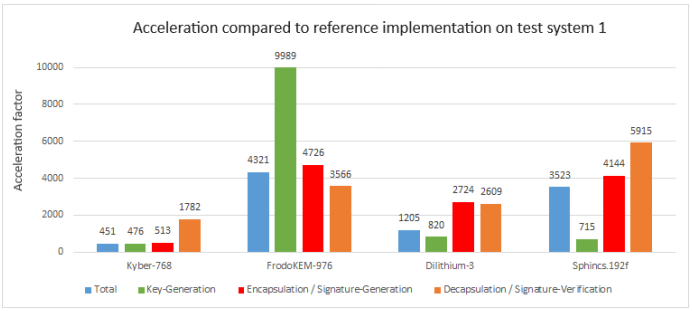


Fig. 14. Acceleration of Algorithms in comparison to CPU-reference implementation on test system 1

Figure 14 shows that the in general, slower algorithms in particular benefit massively from parallel execution on GPUs. For FrodoKEM, the key generation could be increased by a factor of 10,000. The accelerations of the other measurement series for FrodoKEM are similarly strong, as already explained in detail in section V-C1. For both KEMs, massive accelerations for the decapsulation could be achieved, while the other sub-algorithms at Kyber could achieve less than a third of the acceleration.

For the DSAs, especially SPHINCS$^+$ could benefit from the GPU implementation. Signature generation and verification could be massively accelerated, which is particularly significant for the slow signature generation of SPHINCS$^+$. In comparison, Dilithium was able to achieve about half as good results. However, it is clear to see that the time used by Dilithium for data transfer has a massive influence on the

overall execution time, which is why the overall acceleration is significantly lower.

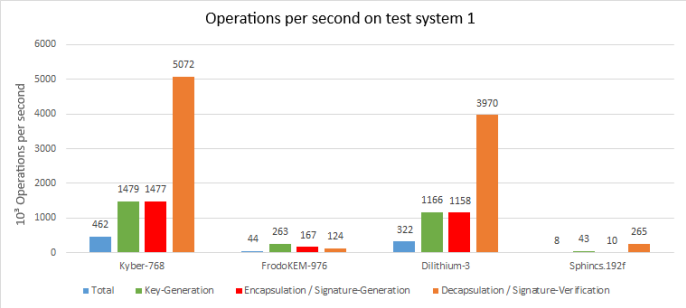


Fig. 15. Operations per second of all algorithms on test system 1

If we compare the acceleration shown with the operations performed per second in the CPU reference implementation, a different picture emerges. The figure 15 shows the operations performed per second depending on the algorithm. For better readability, the values have been reduced by 10^3.
FrodoKEM and SPHINCS+ could be significantly accelerated, but they still cannot keep up with the MLWE methods Kyber and Dilithium. These are based on a mathematical principle that favors the calculations within the algorithms and at the same time only requires a minimal amount of memory.

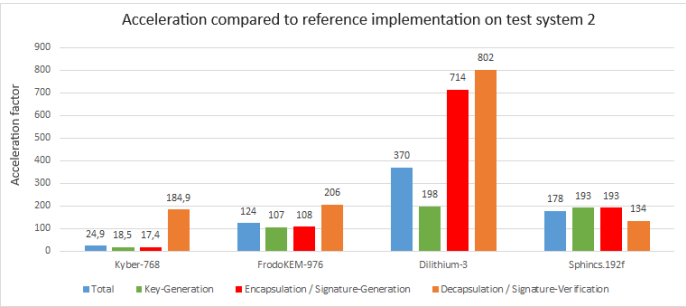


Fig. 16. Acceleration in comparison to reference implementation on test system 2

VII. Ranking of the Algorithms

In this section, the PQC algorithms Kyber, FrodoKEM, Dilithium and SPHINCS$^+$ are evaluated in different categories. Beforehand, the most important arguments for each algorithm are highlighted again and set in relation to its competitors. For the rankings, a distinction is made between the following categories: Key-Generation, KEM, DSA and GPU-PQC. GPU-PQC refers to the entirety of the algorithms and a comparison is made regardless of their underlying method.

a) Kyber: Thanks to the batch-mode implementation of Kyber, almost $1,500,000$ key pairs per second could be generated on test system 1, which represents an increase in over 25% compared to the next best algorithm. Compared to the CPU reference implementation, an acceleration factor of 476 was achieved. Unfortunately, the results on test system 2 were not as promising. There, $\sim 69,000$ key pairs could be generated per second. This only corresponds to an acceleration by a factor of 18.5. As a result, Kyber performs significantly worse than Dilithium on test system 2. Nevertheless, it must be emphasized for Kyber that the public key with $1,184$ bytes only requires 60% of the memory requirement of Dilithium. At $2,400$ bytes, the entire key pair only takes up 40% of the memory of the Dilithium variant.
Compared to the second KEM method, FrodoKEM, the memory requirement for the ciphertext of $1,216$ bytes is only 8%. Since the possible operations per second for encapsulation are almost $1,500,000$, which is 9 times higher, and for decapsulation over $5,000,000$, which is even 41 times higher, Kyber clearly wins out over FrodoKEM in these categories. It is important to note here that Kyber because of its fast execution times, has to spend more and more time on data transfer in relation to the actual runtime of the sub-algorithms as the batch size increases. We look at the measurements in Table IV. The time required settles between $37-38\%$ of the total runtime from a batch size of $80,000$.

b) FrodoKEM: As already shown, sensational speed-ups were achieved for FrodoKEM on test system 1. Only SPHINCS$^+$ was able to achieve the same increases to some extent. Unfortunately, this gives a too positive picture of the possible execution times of FrodoKEM. Because even on the powerful GPU, the maximum for all accelerations was reached early on. This maximum also corresponds to only 18%, 11% and 2% of the values achieved by Kyber for key generation, encapsulation and decapsulation. Despite the slow execution times, 20% of the total time is spent on data transfer. This is particularly evident when considering the size of the key pair and ciphertext. The key pair occupies $46,928$ bytes and is made up of the $31,296$ byte secret key and the $15,632$ byte public key. The ciphertext uses a further $15,768$ bytes. All in all, Kyber therefore only requires just under $\frac{1}{17}$ of memory.
Although FrodoKEM can therefore be accelerated excellently on GPUs, it cannot prevail in comparison to the MLWE methods, which require significantly less memory and time.

c) Dilithium: Thanks to the GPU implementation of Dilithium, accelerations by a factor of 820 could be achieved for the key generation on test system 1. Although this does not yet correspond to the capacities of Kyber with approximately $1,166,000$ key pairs per second, Dilithium shows clear strengths, especially when using weaker GPUs, such as test system 2. There, the output of the CPU reference implementation could be increased 200-fold. This leads to $329,000$ generated key pairs per second, which corresponds to an increase of $\sim 477\%$ compared to Kyber. Despite the higher memory requirements already mentioned, Dilithium can be accelerated significantly better than Kyber with the help of GPUs.
Looking at the signature generation and verification, it is clear to see that Dilithium is the significantly faster DSA compared to SPHINCS$^+$ with $1,160,000$ and $3,970,000$ operations per second. In addition to over 100 times faster performance, Dilithium also impresses with a signature

size of only $3,293$ bytes, which corresponds to 9% of the SPHINCS$^+$ signature. The degree of acceleration compared to the reference implementation, which can be achieved for all sub-algorithms, is also impressive. As already mentioned for Kyber, attention must also be paid to the time spent on data transmission for Dilithium. The measurements from IX follow a similar trend and also seem to approach their maximum at $37 - 38\%$.

d) SPHINCS$^+$: Similar to FrodoKEM, SPHINCS$^+$ shines in terms of the acceleration achieved compared to the CPU reference implementation. Signature generation and verification in particular were significantly accelerated on test system 1 with factors of 4.114 and 5.915. On the negative side, even with this massive acceleration, only $10,000$ signatures per second can be generated, while Dilithium can generate more than $1,150,000$ in comparison.
For SPHINCS$^+$ it is also worth mentioning that it has by far the smallest secret key at 96 bytes. This is extended in the algorithm by applying various hash operations to the required values. As the faster parameter set of SPHINCS$^+$ was checked in this work, a signature size of $35,664$ must be used. Unfortunately, the fact that only 2% of the time is spent on data transmission does not speak in favor of the algorithm, but against its slow execution.

A. Key-Generation

In addition to the speed at which the key pairs can be generated, the required memory and reusability also have an influence on the ranking. It is also important to consider whether an algorithm can only be accelerated well with a strong GPU or whether it also achieves decent results with a weaker GPU.

T-1. Kyber T-1. Dilithium 2. FrodoKEM 3. SPHINCS$^+$

As explained in the previous section, Kyber achieves better results than Dilithium. However, the acceleration on weaker GPUs worked better for Dilithium and the trend on strong GPUs suggests that Dilithium has the opportunity to catch up further with Kyber. Even though the key pairs for KEMs and DSAs have to fulfill different requirements, Kyber and Dilithium share first place in this category. The reasons for this are the aforementioned advantages for speed and acceleration that both bring with them and the dominant results they achieve compared to their competitors within their categories.

B. Key Encapsulation Mechanisms

To avoid redundancy in the chapters, encapsulation and decapsulation are not discussed individually, but the category of KEMs is considered directly.

1. Kyber 2. FrodoKEM

FrodoKEM could convince with its scalability on GPUs alone. However, as the reference values are on average almost 100 times slower than those of Kyber, there are no other points in FrodoKEM's favor. In addition to efficiency, the differences in size between keys and ciphertexts already mentioned confirm Kyber's clear victory in this category.

C. Digital Signature Algorithms

As previously for the KEMs, signature generation and verification are not considered separately, but the complete DSA is evaluated. In this category, both algorithms can be excellently accelerated by GPUs, as already shown in section V-D2. However, the large difference in the execution times of the reference implementations is once again the decisive factor for the better evaluation of the MLWE method.

1. Dilithium 2. SPHINCS$^+$

As a hash-based method, SPHINCS$^+$ shows weaknesses, particularly in the signature generation. These can be mitigated by more powerful GPUs compared to the other sub-algorithms, but cannot be completely ignored. The combination of a relatively large signature and the marginal number of operations performed per second compared to Dilithium results in the rankings given here.

a) GPU-PQC: Even though KEMs and DSAs were not developed for the same purpose and are not directly related to each other apart from a few basics, it makes sense in the context of this paper to compare them based on acceleration through a batch-mode implementation. As in the previous rankings, the actual performance and realistic usability are also included in the decision in addition to the acceleration.

T-1. Kyber T-1. Dilithium 2. SPHINCS$^+$ 3. FrodoKEM

Kyber is not only favored by all previous measurement results, but also by the low memory requirements for keys and ciphertext. However, as the focus of this work is on the acceleration of PQC algorithms by GPUs, Kyber lands in a split first place. Dilithium could be accelerated significantly better, despite the increased memory requirements for keys and signatures. This applies to all strengths of GPUs. The upward trend that can be seen indicates that Dilithium can achieve even better execution times than Kyber with even more powerful GPUs. Since both MLWE-based PQC algorithms have a justified claim to the best rating, this is not clearly awarded.

VIII. Discussion

A. Theory

In addition to the research already conducted on the implementation of PQC algorithms on GPUs, the literature research also focused on the theoretical ideas and principles behind the use of GPUs to accelerate algorithms. Varying solution approaches were found in the various papers. The common procedure for accelerating PQC algorithms using GPU is to identify and outsource the most time-consuming processes. The required data is transferred to the GPU, where it is divided into a predefined number of threads according to its parameters and processed in parallel before being sent to the CPU for

further use. This acceleration of a single iteration of a PQC algorithm was defined in [7] as a single-mode implementation. Another approach presented there for acceleration on GPUs is the batch-mode implementation. Instead of optimizing individual sub-algorithms on CUDA kernels, an attempt is made to utilize the entire bandwidth of the GPU used. To achieve this, any number of iterations of a PQC algorithm are started simultaneously and processed completely by the GPU, while the CPU is only responsible for data transfer from and to the GPU and forwards the commands to the relevant CUDA stream.

Further important theoretical principles for accelerating algorithms through the use of GPUs were demonstrated. Pinned memory defines memory on the host device that is not accessed through classic page addressing, but has already been defined as a data array for the use of the GPU [35]. This also provides the basis for memory reuse during the program, as the memory is previously allocated on the CPU and can be used by the GPU during execution time in several independent operations [22].

Optimized execution of memory accesses can be achieved through memory coalescing. Memory coalescing makes it possible to combine the read and write accesses of threads within a half-warp into a single memory access [36].

Another important point is the avoidance of thread divergence, as all threads involved must wait for the slowest thread to execute before the stream or kernel can process the next operation [9]. Here, it is often necessary to adapt the structure of the algorithm to avoid these scenarios [15]. Another method to achieve this is loop unrolling, which minimizes loop iterations by combining the functionalities into fewer iterations [11]. As the data transfer takes up a not insignificant part of the total runtime of a PQC algorithm on a GPU, it is important that this takes place in a separate copy stream, if possible independently of the executing stream [14].

The optimal configuration of the CUDA kernels used regarding block and grid size is achieved by benchmarking the finished GPU implementations and has a major influence on the performance of the algorithms [7], [17]. Even during development, optimal block and grid sizes for individual kernels can result if their parameters clearly specify them, as with a fixed-size matrix multiplication [37].

The listed properties of GPUs and acceleration on CPUs show that, in theory, any PQC algorithm can be accelerated using a GPU implementation. However, there are operations for which numerous optimized approaches already exist, such as NTT, matrix multiplication, SHAKE or SHA, which makes algorithms that include these more attractive for optimization using GPU.

B. Selected PQC algorithms

Since a global comparison was not readily possible and only some of the aforementioned research also provides the source code of the GPU implementations, this had to be examined in more detail before a decision could be made on how to proceed.

The code of several algorithms was provided by [7]. In addition to batch-mode implementations of Kyber and NewHope, single-mode implementations for FrodoKEM and NewHope were also developed. These were developed while all three algorithms were still part of the NIST-PQC competition. NewHope was no longer considered by NIST in the third round, as Kyber and other low-rank MLWE methods were preferred to RLWE methods.

FrodoKEM was listed by NIST as a replacement candidate for KEMs from the third round onwards, which can be used when the security provided by the unstructured LWE method is more important than performance. It could also become relevant again if new weaknesses are discovered in structured lattice methods in later rounds [16]. FrodoKEM was recommended by several institutions in the same period for ensuring *post-quantum confidentiality*. These include the Federal Office for Information Security (BSI) from Germany and the Netherlands National Communications Security Agency (NLNCSA). In addition, the ISO is in the active process of standardizing FrodoKEM [38].

Since a single-mode implementation was provided for **FrodoKEM**, it was decided to also implement a batch-mode variant as part of this paper. The **Kyber** batch-mode implementation was adopted to make comparisons for the acceleration of KEMs.

The framework for automatic parameter tuning in CUDA kernels presented in [39] was based on the GPU implementation of Kyber from [7]. It is interesting to note that the measurements were carried out as if two parties were communicating between independent systems. This produced results that are closer to reality than the strict execution on a system without data transfers between the sub-algorithms of Kyber.

The GPU implementation of Dilithium from [14] is an excellent example to illustrate the effectiveness of the single-mode approach. There, sequential operations were parallelized during rejection sampling. In addition, the access patterns for NTT operations were optimized and the resource requirements for hash functions were minimized. The I/O latency was also minimized, while the utilization of the threads was optimized by a dynamic task scheduler. By using multiple CUDA streams, the latency caused by data transfers between the CPU and GPU could be concealed.

The optimized single-mode implementation of Dilithium was then taken and executed in batch mode on the GPU. From this, an optimal number of parallel tasks could be generated for the GPU. This was then used for tests with a varying number of CUDA streams. Unfortunately, the code has not yet been made publicly available. Therefore, a custom batch-mode implementation of **Dilithium** was developed for this work. For this purpose, basic principles from [7] could be adopted, since both CRYSTALS algorithms are based on MLWE and use similar data structures.

The source code for a hybrid Falcon and Mikata GPU implementation has also been made available [22]. Mikata is a modification of the Falcon-Singature Scheme, which should be faster and better parallelizable. The lattice-based DSA Falcon is accelerated there as a hybrid variant of single-mode and batch-mode (called fine-grain and coarse-grain implementation there) on the GPU.

To create the same conditions for all algorithms in the batch-mode implementation, the Falcon implementation was not used. Instead, SPHINCS$^+$ was considered in more detail, although no code for its implementation was provided by [15]. The aim of this selection was to extend the spectrum with an algorithm that is not based on lattices. In addition, a second DSA was to be selected to have two candidates available for direct comparison for both categories. Therefore, the code-based KEMs BIKE, HQC and Classic McEliece were out of the question. SIKE, which is based on elliptic curve isogeny, was also not available as it is already considered unsafe. Accordingly, the hash-based DSA **SPHINCS$^+$** was chosen.

C. GPU-Implementations

After determining which PQC algorithms should be considered in more detail in this work, these were implemented on GPUs. An attempt was made to adapt existing ideas from the aforementioned work and to modify the CPU reference implementations of the algorithms so that they are suitable for batch-mode execution in CUDA kernels. To ensure the correctness of the algorithms, the basic structure of the reference implementation was retained in most cases. The methods explained in chapter II were used to accelerate the algorithms. All data that had to be communicated between the CPU and GPU was allocated directly as permanently assigned host memory (pinned memory). In addition, auxiliary variables that are actually initialized in the course of the program were outsourced to the value_sets explained in chapter **??**, thus minimizing the utilization of the threads' local memory. This not only speeds up communication, but also reuses memory that would otherwise be reallocated within operations. Thread divergences were also avoided, as the execution of a single iteration is still carried out on a single thread and thus no divergences can occur. The rejection loop within the signature generation of Dilithium is the exception here. Here, a variable number of attempts are required to generate a signature that does not reveal any information about the secret key. Therefore, in the batch-mode implementation, all instances of Dilithium must wait until the instance that requires the most runs has been completed, which takes an average of 4 iterations [40].
To guarantee the comparability of the measurements, an attempt was made to carry out the same optimizations in all four GPU implementations. Therefore, some theoretical speed-ups listed were disregarded. Additional insights were gained by comparing the single-mode and batch-mode implementation of FrodoKEM. Furthermore, by extending the batch-mode implementation to a variable number of streams, it was possible to prove the positive results achieved in [14] by using more streams. For time reasons, the optimization of the data transmission had to be omitted, which should be considered in future work in any case.
The optimization of a PQC algorithm on GPUs requires a deep understanding of the process as well as a practiced handling of CUDA. To develop hybrid implementations of single-mode and batch-mode, it is therefore advisable to limit the scope of the work to a single algorithm. Adequate optimizations of all sub-algorithms from different sources can then be carried out for this algorithm or developed in-house.
The results obtained in this work make it possible to make accurate statements about the acceleration of PQC algorithms on GPUs. The focus was on a uniform implementation of the algorithms to ensure comparability. By using two differently powerful test systems, further strengths and weaknesses of the algorithms could be identified.

D. Results

In chapter VI, the measurement results of the four PQC algorithms were systematically listed and analyzed. The best results of the individual measurement series were summarized in the tables II and III for KEMs and DSAs respectively and compared with the reference implementations on the same test system.
For FrodoKEM, in addition to the comparison with the other algorithms, a comparison was made between the acceleration achieved by single-mode and batch-mode. However, problems occurred on test system 1 during the measurements carried out for this purpose. The results achieved there are slower than the CPU reference implementation and cannot be adequately explained without closer examination. One possible explanation would be that the 16 single-core processors on test system 1 do not work together properly during the course of the program, and therefore additional time has to be spent on data transfer. As the NVIDIA Tesla A100 used on test system 1 is significantly more powerful than the NVIDIA GeForce RTX 2070 Super on test system 2, the poor performance of the single-mode implementation cannot be attributed to the GPU. The results achieved on test system 2 also achieved an average of 22 times as many operations per second as the reference implementation.
In addition, the FrodoKEM implementation was extended so that the number of CUDA streams can be varied. For the series of measurements with $1, 2, 4, 8$ and 16 streams, it was shown that the best execution times could be achieved with 8 parallel CUDA streams, as can be seen in the figures 7, 6 and 5. However, the increasing number of CUDA streams appears to be accompanied by a significant overhead in the allocation and transmission of data. This is shown in the diagrams by the increasingly smaller measurement series, as the maximum capacity of the application, which is $\sim 2,148$ MB, is reached earlier and earlier.
This drastic reduction in the possible batch size can be neglected in at least two applications. It becomes uninteresting if the algorithm already reaches its maximum throughput of operations per second before the limiting batch size is reached. Alternatively, it can be neglected for algorithms with a lower memory requirement, such as Kyber, as their maximum batch size is already 20 times higher when using a stream.
It is also important to mention that the number of streams has a negative impact on data transmission. As can be seen in the measurement series for figure 4, almost 50% of the time is spent on data transmission with 8 streams and almost 68% with 16 streams. However, as mentioned in [14], there are ways to optimize this with additional streams that are only

responsible for data transmission and an optimized sequence of transmissions, so a variable number of CUDA streams offers a good optimization option for future GPU implementations. The optimal number of streams depends on the use case and must be determined individually through measurements.

If you look at the reference implementations of the four PQC algorithms Kyber, Dilithium, FrodoKEM and SPHINCS$^+$, you can see massive differences in their execution speeds. The MLWE methods Kyber and Dilithium are between 20 and 150 faster than FrodoKEM and SPHINCS$^+$ in the categories KEM and DSA, depending on the sub-algorithm. Since the CRYSTALS algorithms have significantly lower memory requirements and faster execution times, it is reasonable to assume that they can be accelerated better on GPUs.

The figure 14 disproves this assumption for strong GPUs. It can be seen that the slow algorithms benefit to a greater extent from the increasing computing power. Kyber performs the worst, while the acceleration of Dilithium is almost the same as for SPHINCS$^+$. On the weaker GPU of test system 2, good results were still achieved for FrodoKEM and SPHINCS, as can be seen in 16. However, Dilithium again shows its strengths here and benefits the most from the GPU implementation, while Kyber performs the worst.

If we compare the speedups achieved from 14 with the operations per second achieved from 15, we can see that Kyber and Dilithium clearly dominate their categories. FrodoKEM was able to achieve an acceleration of $\sim 10,000$ for key generation, but with just under $263,000$ operations per second it only achieves $\sim 18\%$ of Kyber's throughput. The situation is even worse for encapsulation and decapsulation. For decapsulation, only $\sim 2.5\%$ of Kyber's operations per second are achieved.

A similar picture emerges for the comparison of Dilithium and SPHINCS$^+$. Signature verification performs best there with $265,000$ operations per second, which corresponds to around 7% of Dilithium.

If the ranking of the PQC algorithms is only based on the pure acceleration compared to the reference implementation, it would look as follows:

1. FrodoKEM 2. SPHINCS$^+$ 3. Dilithium 4. Kyber

Despite their high memory requirements and the associated data transfer overhead, FrodoKEM and SPHINCS$^+$ benefited the most from the batch-mode implementation on strong GPUs. Dilithium could be accelerated well regardless of the strength of the GPU and thus showed the most consistent results. Although Kyber could be accelerated well on the strong GPU, it could not achieve anywhere near the same results as the other algorithms.

However, if you look at the PQC algorithms in the overall context of their possible uses and fields of application, the ranking is reversed:

T-1. Kyber T-1. Dilithium 2. SPHINCS$^+$ 3. FrodoKEM

Although Dilithium accelerated significantly better on GPUs, it shares first place with Kyber, as the faster execution times were still achieved there and the memory requirement is less than half. Furthermore, without further tests on more powerful GPUs, it cannot be generally assumed that the acceleration of Dilithium can still be improved to the same extent. Since Kyber and Dilithium were also selected by NIST as the primary algorithms for KEMs and DSAs, it is not surprising that they perform so well. The structured polynomials of the MLWE methods can not only be processed well in parallel, but also save an enormous amount of memory.

The decision to put SPHINCS$^+$ in second place is based on the relatively small key pairs and the many parameter sets provided, which allow an application-specific configuration without jeopardizing the security of the algorithm. The speedups in signature generation achieved by the batch-mode GPU implementation also help to combat the biggest weakness of SPHINCS$^+$.

Although FrodoKEM achieved the best results for acceleration with a strong GPU, it requires over 60 Mb of memory for a combination of key pair and ciphertext. In combination with the dominance of Kyber in terms of execution times, no other placement is therefore possible. However, since the ISO and other institutions rely on the security of the unstructured Lattice method, further research is also useful here.

Several topics for future research can be derived from the results obtained in this work. However, it must be determined from the outset whether the implementations must be comparable with each other or whether the best possible result should be achieved for each PQC algorithm. The next logical step for the implementations created here is the extension to a hybrid implementation that combines the positive properties of single-mode and batch-mode variants. As this already exists for Dilithium and, according to the authors, the source code is to be published in the future, the analogous Kyber implementation would be the next interesting step. The situation is similar for SPHINCS$^+$ and FrodoKEM. Here, too, the hybrid implementations could bring additional added value.

However, the batch-mode implementation of the accelerated PQC algorithms is only suitable for use in certain applications. A server that has to process a large number of key exchanges every second during peak usage times can use the GPU implementation to increase the throughput according to the current requirements, which prevents backlogs and thus keeps latencies low for users. In contrast, the GPU implementation is not suitable for improving the latencies for individual requests, as the CPU variant can act faster thanks to the lack of data transfer. The DSAs can be used according to the same principle.

The GPU implementations developed in this work can be considered a guide and basis for future improvements to the methods. In their current form, they are significantly faster than the reference implementations in C, but can still be significantly improved by many optimizations on individual operations of the algorithms. By systematically optimizing the results on different batch sizes and with different grid and block sizes, several GPU parameter sets can be generated for each PQC algorithm, which can be used variably depending on the security, latency and throughput requirements of applications.

IX. Conclusion

The aim of the paper was to investigate the accelerability of PQC algorithms using GPUs. To this end, the current state of research on the topic was analyzed, and the most important achievements already made were identified. On this basis, important methods for accelerating GPU implementations were listed. The most important methods include execution in batch mode, i.e., running several instances of the algorithm in parallel. In addition, the correct allocation of host memory, optimal data transfer between CPU and GPU and fine-tuning the execution parameters of the CUDA kernel have a major influence on execution speeds.

The literature identified in the first step was then examined in more detail to determine which PQC algorithms already have a GPU implementation. This showed that variants on GPUs already exist for Kyber, Dilithium, Falcon, FrodoKEM, SPHINCS, Classic McEliece and SIKE. To ensure the comparability and thus the validity of the results, an attempt was made to accelerate all GPU implementations in the same way. It was therefore decided to disregard the hybrid GPU implementation of Falcon and to develop our own implementations. Based on the existing single-mode implementation, a batch-mode implementation was created for FrodoKEM. This also allowed a comparison to be made between these two variants. As a batch-mode implementation was already available for Kyber, Dilithium was also developed in-house, as these two methods are recommended by the National Institute of Standards and Technology (NIST) for the categories Key Encapsulation Mechanism (KEM) and Digital Signature Algorithm (DSA). SPHINCS+ was implemented to investigate a second DSA. Frodo-KEM was also extended so that any number of CUDA streams can be used to demonstrate the positive influence on execution times mentioned in other work. This variant is referred to below as a multistream implementation.
Two different test systems were used for the measurements. Test system 1 uses an Nvidia Tesla A100 with 80 GB of internal memory, and test system 2 uses an Nvidia GeForce RTX 2070 Super with 8 GB of internal memory. Each measurement was divided into four parts. The total time includes the execution times of the sub-algorithms and the time required for data transfer. The key generation for all algorithms is also measured. The KEMs also have the measurement series for encapsulation and decapsulation, while the signature generation and signature verification are measured for the DSAs.

The decapsulation of Kyber could be accelerated by a factor of 1,782 on test system 1, resulting in 5,072,000 operations per second with a batch size of 35,000. For Frodo-KEM, the key generation on the test system was increased by a factor of 10,000 to 262,727 operations per second with a batch size of 20,000, which is the highest acceleration achieved. The slowest process of SPHINCS+, the Signature generation, was accelerated by a factor of 4,114 to almost 10,000 operations per second on test system 1. Looking at the overall picture of the measurements on both test systems, Dilithium performed best. Accelerations by a factor of 700 - 800 were also achieved on test system 2 for the signature generation and verification.

The single-mode implementation of FrodoKEM achieved worse results on test system 1 than the CPU reference implementation, which suggests that the communication between the 16 CPUs and the GPU is not functioning optimally. On test system 2, however, it was able to achieve a 22-fold acceleration, which corresponds to around a tenth of the batch-mode implementation. The multistream variant of FrodoKEM has shown that the partial operations are executed fastest with 8 parallel CUDA streams. However, the data transfer must be optimized here before it can be used sensibly, as 50% of the time is currently spent on this with a batch size of 25,000 and 8 parallel CUDA streams.

The results indicate that the slow algorithms in particular, which have high memory requirements, can be accelerated well on powerful GPUs. If the ranking is therefore based solely on the acceleration achieved on GPUs compared to the reference implementation, FrodoKEM performs best. As it can be assumed that a powerful GPU is used for use cases on server environments, SPHINCS+ takes second place ahead of Dilithium and Kyber.

A further ranking was drawn up which, in addition to the acceleration on GPUs, also considers data such as the size of the key and ciphertext/signature, as well as the throughput achieved per second. Although Dilithium was significantly better at accelerating on GPUs, it shares first place with Kyber, as it still achieved the faster execution times and required less than half the memory. Furthermore, without further tests on more powerful GPUs, it cannot be assumed that the acceleration of Dilithium will continue to improve to the same extent. Since Kyber and Dilithium were also selected by NIST as the primary algorithms for KEMs and DSAs, it is not surprising that they perform so well. The structured polynomials of the MLWE methods can not only be processed well in parallel, but also save an enormous amount of memory. The decision to place SPHINCS+ in second place is based on the relatively small key pairs and the many parameter sets provided, which allow application-specific configuration without jeopardizing the security of the algorithm. The speed-ups achieved by the batch-mode GPU implementation in signature generation also help to combat the biggest weakness of SPHINCS+.

Overall, this paper shows that the acceleration of post-quantum cryptography (PQC) algorithms on GPUs can offer significant added value for applications with a high workload. The speedups vary depending on the algorithm and application field. However, it should be noted that the choice between CPU and GPU implementations depends on various factors, including the specific requirements of the application and the available hardware resources.

References

[1] D. D. M. Dr. Lily Chen and D. Y.-K. Liu, "Post-quantum cryptography standardization." Webnews, 2 2024. https://csrc.nist.gov/pqc-standardization.

[2] "Information security — Digital signatures with appendix," 6 2024. https://www.iso.org/standard/80492.html.

[3] L. Reichert, N. Coppik, and S. Finster, "Performance evaluation of quantum-resistant algorithms on industrial embedded systems," in *Availability, Reliability and Security* (B. Coppens, B. Volckaert, V. Naessens, and B. De Sutter, eds.), (Cham), pp. 127–148, Springer Nature Switzerland, 2025.

[4] T. Liu, G. Ramachandran, and R. Jurdak, "Post-quantum cryptography for internet of things: A survey on performance and optimization," 2024.

[5] T. Reddy.K, D. Wing, B. S, and K. Kwiatkowski, "Adapting Constrained Devices for Post-Quantum Cryptography," Internet-Draft draft-ietf-pquip-pqc-hsm-constrained-01, Internet Engineering Task Force, July 2025. Work in Progress.

[6] "Source code [online]." https://github.com/Reiiiatsu/PQC-GPU-Acceleration, 2025.

[7] N. Gupta, A. Jati, A. K. Chauhan, and A. Chattopadhyay, "Pqc acceleration using gpus: Frodokem, newhope, and kyber," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 575–586, 2021.

[8] G. Huy, "Global memory coalescing." Medium Article, 2024. https://giahuy04.medium.com/global-memory-coalescing-37a6f9d7e314.

[9] T. D. Han and T. S. Abdelrahman, "Reducing branch divergence in gpu programs," *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, 2011.

[10] J. Meng, D. Tarjan, and K. Skadron, "Dynamic warp subdivision for integrated branch and memory divergence tolerance," in *Proceedings of the 37th annual international symposium on Computer architecture*, pp. 235–246, 2010.

[11] R. C. O. Rocha, V. Porpodas, P. Petoumenos, L. F. W. Góes, Z. Wang, M. Cole, and H. Leather, "Vectorization-aware loop unrolling with seed forwarding," *Proceedings of the 29th International Conference on Compiler Construction*, 2020.

[12] H. Kim, S. U. Hong, J. H. Park, and H. Han, "Static code transformations for thread-dense memory accesses in gpu computing," *Concurrency and Computation Practice and Experience*, 2019.

[13] *An Evaluation of Unified Memory Technology on NVIDIA GPUs*, 2015.

[14] S. Shen, H. Yang, W. Dai, Z. Liu, and Y. Zhao, "High-throughput gpu implementation of dilithium post-quantum digital signature," 2022.

[15] S. Sun, R. Zhang, and H. Ma, "Efficient parallelism of post-quantum signature scheme sphincs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2542–2555, 2020.

[16] C. contributors, "Computer security resurce center - post-quantum cryptography," 2023. [Online; accessed 17-May-2023].

[17] V. Volkov and J. W. Demmel, "Benchmarking gpus to tune dense linear algebra," in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pp. 1–11, 2008.

[18] D. C. K. W. Jia-Lin Chan, Wai-Kong Lee, W.-S. Yap, B.-Y. Ooi, and B.-M. Goi, "High throughput acceleration of nist lightweight authenticated encryption schemes on gpu platform," *Cluster Computing*, vol. 27, pp. 11213–11235, 2024.

[19] S. Shen, H. Yang, W. Dai, H. Zhang, Z. Liu, and Y. Zhao, "High-throughput gpu implementation of dilithium post-quantum digital signature," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 11, pp. 1964–1976, 2024.

[20] D. Kim, H. Choi, and S. C. Seo, "Parallel implementation of sphincs+ with gpus," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 6, pp. 2810–2823, 2024.

[21] S. O. H. Wai-Kong Lee, Hwajeong Seo, R. Achar, A. Karmakar, and J. M. B. Mera, "Dpcrypto: Acceleration of post-quantum cryptography using dot-product instructions on gpus," 9 2022. https://ieeexplore.ieee.org/abstract/document/9788519.

[22] W.-K. Lee, R. K. Zhao, R. Steinfeld, A. Sakzad, and S. O. Hwang, "High throughput lattice-based signatures on gpus: Comparing falcon and mitaka." Cryptology ePrint Archive, Paper 2023/399, 2023. https://eprint.iacr.org/2023/399.

[23] S. R. Yarkin Doroz and M. Wolf, "Introducing nvidia cupqc for gpu-accelerated post-quantum cryptography." Webnews, 12 2024. https://developer.nvidia.com/blog/introducing-nvidia-cupqc-for-gpu-accelerated-post-quantum-cryptography/.

[24] N. contributors, "Module-lattice-based key-encapsulation mechanism standard," 2024. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf.

[25] J. Wu, Y. Yu, Z. Chen, H. Yang, C. Li, and Z. Liu, "Cbpspx: A cuda-based batch parallel optimization of post-quantum signature sphincs+," *IEEE Internet of Things Journal*, pp. 1–1, 2025.

[26] Z. Wang, X. Dong, H. Chen, Y. Kang, and Q. Wang, " CUSPX: Efficient GPU Implementations of Post-Quantum Signature SPHINCS+ ," *IEEE Transactions on Computers*, vol. 74, pp. 15–28, Jan. 2025.

[27] Z. Chen, W. He, J. Wu, J. Liu, B. Xu, X. Bian, and Z. Liu, "An optimized gpu-based acceleration of crystals-dilithium," in *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, 2025.

[28] S. C. Seo and S. An, "Parallel implementation of crystals-dilithium for effective signing and verification in autonomous driving environment," *ICT Express*, vol. 9, no. 1, pp. 100–105, 2023.

[29] H. Becker, V. Hwang, M. J. Kannwischer, B.-Y. Yang, and S.-Y. Yang, "Neon NTT: Faster dilithium, kyber, and saber on cortex-a72 and apple m1." Cryptology ePrint Archive, Paper 2021/986, 2021.

[30] C. Zhao, N. Zhang, H. Wang, B. Yang, W. Zhu, Z. Li, M. Zhu, S. Yin, S. Wei, and L. Liu, "A compact and high-performance hardware architecture for crystals-dilithium," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, p. 270–295, Nov. 2021.

[31] C. Team, "Crystals - cryptographic suite for algebraic lattices," 2022. [Online; accessed 24-May-2023].

[32] D. Stebila, "ds_benchmark.h: Macros for simple benchmarking of c code," 2021. [Online; accessed 21-September-2023].

[33] D. Stebila and M. Mosca, "Post-quantum key exchange for the internet and the open quantum safe project," *Selected Areas in Cryptography (SAC) 2016*, vol. 10532, pp. 1–24, Oktober 2017.

[34] J.-P. Aumasson, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, *et al.*, "Sphincs+ algorithm specifications and supporting documentation," 2021.

[35] S. C. Seo and S. An, "Parallel implementation of crystals-dilithium for effective signing and verification in autonomous driving environment," *ICT Express*, vol. 9, no. 1, pp. 100–105, 2023.

[36] V. Saxena, Y. Sabharwal, and P. Bhatotia, "Performance evaluation and optimization of random memory access on multicores with high productivity," in *2010 International Conference on High Performance Computing*, pp. 1–10, IEEE, 2010.

[37] S. An and S. C. Seo, "Efficient parallel implementations of lwe-based post-quantum cryptosystems on graphics processing units," *Mathematics*, vol. 8, no. 10, 2020.

[38] F. Team, "Frodokem - practical quantum-secure key encapsulation from generic lattices," 2023. [Online; accessed 21-September-2023].

[39] T. Ono, S. Bian, and T. Sato, "Automatic parallelism tuning for module learning with errors based post-quantum key exchanges on gpus," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2021.

[40] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: Algorithm specifications and supporting documentation," *NIST Post-Quantum Cryptography Standardization Round*, vol. 3, 2021.

X. Appendix

All time measurements in the appendix are prefixed with *t_* and are measured in milliseconds (*ms*). *t_total* contains the entire execution time including the data transfer time. The other measurement series are labeled with *kgen* for key generation, *enc* for encapsulation, *dec* for decapsulation, *sign* for signature generation and *verify* for signature verification. In addition, the time spent on data transfer between the GPU and CPU (*t_copy*) and the percentage of the total time taken up by the copying process (*copy_%*) are indicated. For all listed labels, there are also calculated executions per second (*EPS*) values that indicate the throughput of the different functions depending on the batchsize. Decimal digits were cut to represent only fully completed runs per second.

In the tables VII, VIII, XII, XVII and XXVI, the results achieved on test system 1 are compared with the optimized versions. The difference is calculated using the formula 2 to show the percentage difference between the optimized version and the standard version.

$$\text{Difference } \% = (1 - \frac{\text{Standard Version Measurement}}{\text{Optimized Version Measurement}}) * 100 \tag{2}$$

The measurement results of Kyber , Dilithium , FrodoKEM , SPHINCS$^+$ and the CPU reference implementation are shown one after the other.

XI. Measurements Kyber

TABLE IV
Kyber Batch-mode Test system 1

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_enc	enc/s	t_dec	dec/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	15.2379	65	0.0498	0.3269	6.2622	159	6.9888	143	1.9371	516
2	14.5639	137	0.0547	0.3756	5.9760	334	6.6626	300	1.8707	1069
4	14.6461	273	0.0564	0.3852	6.0122	665	6.6987	597	1.8787	2129
8	14.8583	538	0.0736	0.4950	6.1029	1310	6.7784	1180	1.9035	4202
16	15.3061	1045	0.0963	0.6292	6.3139	2534	6.9584	2299	1.9375	8258
32	16.3460	1957	0.1482	0.9068	6.7527	4738	7.4059	4320	2.0392	15692
64	17.0791	3747	0.2647	1.5496	7.0575	9068	7.6627	8352	2.0943	30558
128	18.7146	6839	0.4455	2.3802	7.6791	16668	8.3282	15369	2.2619	56590
256	19.1733	13351	0.8352	4.3562	7.7130	33190	8.3552	30639	2.2699	112781
512	20.0743	25505	1.5374	7.6584	7.7970	65666	8.4508	60585	2.2890	223674
1024	21.6016	47403	2.3995	11.1079	8.0591	127061	8.7977	116394	2.3454	436605
2048	23.7504	86230	2.6773	11.2727	8.8245	232080	9.8836	207212	2.3650	865962
4096	27.2479	150323	4.6754	17.1587	9.5172	430380	10.3952	394029	2.6602	1539758
8192	33.2301	246523	5.3579	16.1237	11.9073	687978	12.6150	649386	3.3498	2445502
16384	46.6073	351532	9.8609	21.1574	15.6750	1045229	16.6200	985800	4.4514	3680630
32768	71.8189	456258	20.2501	28.1960	22.4150	1461876	22.5674	1452006	6.5864	4975116
5000	28.8784	173140	5.4136	18.7464	9.9990	500047	10.6542	469300	2.8115	1778400
10000	34.9998	285715	6.2570	17.8773	12.2557	815944	13.0612	765629	3.4259	2918937
15000	45.9466	326466	9.4447	20.5559	15.6390	959143	16.4525	911714	4.4104	3401084
20000	48.9499	408581	10.8608	22.1877	16.4264	1217548	17.1159	1168504	4.5467	4398781
25000	61.5827	405958	17.0338	27.6601	19.1910	1302695	19.7556	1265465	5.6023	4462441
30000	69.3700	432463	18.9060	27.2538	21.9344	1367715	22.1260	1355871	6.4036	4684833
35000	75.8198	461621	21.5721	28.4518	23.6579	1479420	23.6887	1477498	6.9010	5071705
40000	82.1509	486909	23.7463	28.9057	25.4746	1570188	25.4011	1574734	7.5288	5312927
45000	92.2727	487684	26.1050	28.2912	28.9724	1553199	28.7525	1565078	8.4426	5330096
50000	98.0796	509789	27.3114	27.8461	30.9354	1616269	30.4966	1639526	9.3362	5355499
55000	104.7784	524917	27.7710	26.5045	33.9665	1619243	33.1059	1661333	9.9350	5536005
60000	112.7795	532011	32.1142	28.4752	35.4343	1693276	34.5670	1735761	10.6641	5626366
65000	125.8274	516580	41.3582	32.8690	37.0337	1755159	36.0736	1801874	11.3621	5720783
70000	141.9501	493130	51.2084	36.0749	39.8992	1754420	38.7803	1805041	12.0622	5803239
75000	147.1437	509705	53.9729	36.6804	41.0385	1827553	39.7278	1887845	12.4045	6046178
80000	153.7639	520278	57.0170	37.0809	42.5507	1880110	41.1828	1942560	13.0134	6147502
85000	161.5907	526020	58.6071	36.2688	45.4076	1871935	43.9899	1932262	13.5861	6256372
90000	166.0113	542131	61.2352	36.8861	46.2040	1947883	44.7107	2012942	13.8615	6492810
95000	173.1394	548690	65.1193	37.6109	47.6776	1992549	46.0441	2063240	14.2984	6644104
100000	179.5950	556808	66.3572	36.9482	50.0684	1997268	48.3628	2067703	14.8067	6753717
105000	184.8055	568164	69.0625	37.3704	51.1502	2052776	49.4079	2125167	15.1849	6914760
110000	191.3555	574846	72.8914	38.0921	52.4376	2097732	50.5278	2177020	15.4987	7097354
115000	198.8866	578218	75.1575	37.7891	54.7883	2098989	52.9342	2172509	16.0066	7184525
120000	204.1624	587767	77.4586	37.9397	56.0896	2139434	54.1697	2215261	16.4445	7297265
125000	213.3168	585982	80.0214	37.5129	58.6207	2132351	57.5057	2173697	17.1690	7280568
130000	218.7560	594269	81.9090	37.4431	59.7901	2174273	59.5421	2183330	17.5149	7422237
135000	225.7999	597874	85.8542	38.0223	61.1361	2208189	60.8152	2219838	17.9944	7502340
140000	242.2385	577942	90.4726	37.3486	66.9175	2092129	64.7012	2163792	20.1472	6948857
145000	245.2122	591324	89.7717	36.6098	68.6190	2113117	66.1017	2193590	20.7197	6998156
150000	253.9664	590629	95.2808	37.5171	69.9402	2144689	67.5407	2220884	21.2047	7073913
155000	261.4915	592753	97.1394	37.1482	72.4575	2139184	70.1323	2210109	21.7623	7122416
160000	268.2621	596431	100.3918	37.4230	73.8565	2166363	71.7091	2231236	22.3048	7173359
165000	276.3348	597101	104.6664	37.8767	75.5277	2184629	73.2846	2251496	22.8561	7219081
170000	288.1209	590030	109.4534	37.9887	78.7316	2159235	76.0679	2234844	23.8680	7122509
175000	291.8827	599555	108.6169	37.2125	80.7704	2166634	77.8388	2248237	24.6565	7097533
180000	302.0825	595863	113.2901	37.5030	83.1525	2164697	80.2051	2244245	25.4348	7076925
185000	307.3546	601910	113.7574	37.0118	85.3958	2166382	82.1784	2251198	26.0230	7109106
190000	315.1561	602875	117.7171	37.3520	87.0248	2183285	83.8262	2266594	26.5879	7146103
195000	322.9673	603776	120.2516	37.2334	89.4653	2179616	86.0131	2267096	27.2374	7159282
200000	329.4390	607092	122.2341	37.1037	91.4719	2186463	87.8984	2275353	27.8345	7185330

TABLE V
KYBER BATCH-MODE TEST SYSTEM 2

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_enc	enc/s	t_dec	dec/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	519.4615	1	0.0771	0.0148	240.8591	4	252.4527	3	26.0726	38
2	522.2928	3	0.0723	0.0138	242.5637	8	253.2627	7	26.3941	75
4	527.6936	7	0.0701	0.0133	250.2085	15	250.7767	15	26.6383	150
8	513.6362	15	0.0689	0.0134	237.8638	33	249.2650	32	26.4385	302
16	510.9288	31	0.0757	0.0148	235.9601	67	248.9103	64	25.9826	615
32	526.4181	60	0.0959	0.0182	242.5550	131	256.9082	124	26.8590	1191
64	520.1255	123	0.1354	0.0260	240.6782	265	252.5783	253	26.7338	2393
128	518.4518	246	0.2079	0.0401	240.4480	532	251.2779	509	26.5180	4826
256	519.0423	493	0.2783	0.0536	239.6369	1068	252.6325	1013	26.4945	9662
512	520.2675	984	0.4931	0.0948	240.1773	2131	252.8578	2024	26.7393	19147
1024	522.9127	1958	0.8840	0.1691	240.7675	4253	254.2483	4027	27.0127	37908
2048	524.3322	3905	1.6255	0.3100	241.3126	8486	254.2895	8053	27.1045	75559
4096	530.4574	7721	3.1953	0.6024	243.5017	16821	256.3109	15980	27.4495	149219
8192	554.7111	14768	6.2892	1.1338	252.2527	32475	266.3024	30762	29.8670	274282
16384	647.0950	25319	12.3483	1.9083	290.4243	56414	309.0939	53006	35.2284	465078
32768	1154.3962	28385	24.5626	2.1277	529.8142	61848	555.3849	59000	44.6346	734138
5000	536.9375	9312	3.8732	0.7214	245.7525	20345	259.2545	19286	28.0573	178206
10000	568.1096	17602	7.6903	1.3537	257.7091	38803	272.2410	36732	30.4691	328201
15000	626.1934	23954	11.4035	1.8211	280.8190	53415	300.1487	49975	33.8221	443497
20000	692.3849	28885	15.0171	2.1689	311.3255	64241	330.1125	60585	35.9297	556642
25000	1084.9778	23041	18.6590	1.7198	503.8907	49613	522.8864	47811	39.5418	632243
30000	1123.7329	26696	22.4330	1.9963	518.9153	57812	541.0620	55446	41.3226	725995
35000	1180.2336	29655	26.0669	2.2086	540.0385	64810	568.7460	61538	45.3822	771226
40000	1320.8914	30282	29.7551	2.2527	590.1248	67782	636.9448	62799	64.0666	624350
45000	1788.5154	25160	33.4830	1.8721	811.0712	55482	868.9662	51785	74.9946	600043
50000	1828.3264	27347	37.2560	2.0377	830.2487	60222	882.1659	56678	78.6558	635680
55000	1860.3635	29564	40.9389	2.2006	845.6457	65039	892.2601	61641	81.5186	674692
60000	1918.6331	31272	44.5578	2.3224	870.9412	68890	919.2902	65267	83.8437	715616
65000	2328.1177	27919	48.4649	2.0817	1074.4789	60494	1109.4120	58589	95.7617	678767
70000	2362.6899	29627	51.8242	2.1934	1090.2384	64206	1117.2020	62656	103.4253	676817
75000	2430.7852	30854	55.5702	2.2861	1121.1239	66897	1144.0204	65558	110.0708	681379
80000	2556.8406	31288	59.4450	2.3249	1166.8970	68557	1203.6029	66467	126.8957	630439
85000	3019.2915	28152	63.0580	2.0885	1387.5581	61258	1431.5139	59377	137.1617	619706
90000	3135.4226	28704	66.7859	2.1300	1432.7722	62815	1479.9792	60811	155.8855	577346
95000	3206.6072	29626	70.6361	2.2028	1469.8154	64633	1503.5555	63183	162.6001	584255
100000	3141.8789	31828	74.3680	2.3670	1460.8597	68452	1471.7828	67944	134.8682	741464
105000	3533.0645	29719	83.7078	2.3693	1654.9043	63447	1656.4807	63387	137.9719	761024
110000	3585.1008	30682	81.6420	2.2773	1678.6774	65527	1684.0266	65319	140.7550	781499
115000	3616.0649	31802	85.2836	2.3585	1691.7096	67978	1695.7590	67816	143.3127	802441
120000	3750.3145	31997	89.1390	2.3768	1742.6165	68861	1756.5906	68314	161.9687	740883
125000	4187.9658	29847	92.6952	2.2134	1953.5472	63986	1969.5193	63467	172.2044	725881
130000	4257.4956	30534	96.3477	2.2630	1986.0422	65456	1998.9174	65035	176.1884	737846
135000	4300.0112	31395	100.1096	2.3281	2004.5176	67347	2015.2574	66988	180.1265	749473
140000	4342.3252	32240	103.9144	2.3931	2023.3844	69191	2033.8542	68834	181.1717	772747
145000	4706.1816	30810	107.5651	2.2856	2203.2722	65811	2211.0571	65579	184.2874	786814
150000	4764.6577	31481	111.3028	2.3360	2230.4995	67249	2236.1123	67080	186.7428	803243
155000	4894.4521	31668	114.9244	2.3481	2279.4812	67997	2293.0479	67595	206.9990	748795
160000	5013.8750	31911	118.3446	2.3603	2341.0188	68346	2343.9282	68261	210.5835	759793
165000	5435.2979	30357	122.2200	2.2486	2569.3921	64217	2528.4812	65256	215.2043	766713
170000	5497.8779	30921	125.9695	2.2912	2598.6304	65419	2556.0769	66508	217.2019	782682
175000	5544.6943	31561	129.5780	2.3370	2619.4153	66808	2575.0142	67960	220.6874	792976
180000	5613.7451	32064	133.5088	2.3782	2650.7329	67905	2605.7576	69077	223.7460	804483
185000	5951.9004	31082	137.0078	2.3019	2820.6272	65588	2767.2715	66852	226.9941	814999
190000	6044.5703	31433	140.8559	2.3303	2864.5977	66326	2809.4419	67629	229.6756	827253
195000	6162.3271	31643	144.5398	2.3455	2908.0688	67054	2861.0425	68156	248.6751	784155
200000	6243.6807	32032	148.1011	2.3720	2946.0073	67888	2897.3521	69028	252.2203	792957

TABLE VI
KYBER BATCH-MODE TEST SYSTEM 1 (OPTIMIZED)

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_enc	enc/s	t_dec	dec/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	516.5579	1	0.0412	0.0080	239.1625	4	251.6888	3	25.6653	38
2	515.5491	3	0.0814	0.0158	238.4407	8	251.0398	7	25.9871	76
4	515.6136	7	0.0822	0.0159	239.2382	16	250.7198	15	25.5734	156
8	515.5211	15	0.0867	0.0168	237.7723	33	252.0577	31	25.6045	312
16	514.2278	31	0.0977	0.0190	237.4241	67	251.1082	63	25.5980	625
32	514.4623	62	0.1201	0.0234	237.6573	134	250.9178	127	25.7671	1241
64	515.0242	124	0.1412	0.0274	237.7874	269	251.1024	254	25.9932	2462
128	515.8260	248	0.2009	0.0390	237.8355	538	250.5588	510	27.2308	4700
256	517.9780	494	0.3192	0.0616	239.3214	1069	250.7527	1020	27.5847	9280
512	528.3492	969	0.5280	0.0999	247.1598	2071	253.0317	2023	27.6296	18530
1024	540.3812	1894	0.9954	0.1842	250.8739	4081	261.3885	3917	27.1235	37753
2048	571.7054	3582	1.8122	0.3170	270.1259	7581	273.1681	7497	26.5992	76994
4096	622.5002	6579	3.6304	0.5832	298.6167	13716	292.5922	13999	27.6610	148078
8192	774.7033	10574	7.1069	0.9174	381.8664	21452	352.5654	23235	33.1645	247010
16384	1233.5508	13281	14.0541	1.1393	571.2457	28681	514.5490	31841	133.7021	122541
32768	2659.5457	12320	28.0011	1.0529	1019.3779	32145	1069.6729	30633	542.4940	60402
5000	667.0731	7495	4.4069	0.6606	325.3217	15369	309.4613	16157	27.8832	179319
10000	825.7031	12110	8.6539	1.0481	408.7527	24464	370.6639	26978	37.6327	265726
15000	1122.5806	13362	12.9808	1.1563	549.7963	27282	474.8612	31588	84.9422	176590
20000	1639.0275	12202	17.6867	1.0791	667.6051	29957	677.7328	29510	276.0028	72463
25000	2017.3240	12392	21.4132	1.0615	756.5815	33043	823.7393	30349	415.5899	60155
30000	2439.9629	12295	25.6534	1.0514	920.7206	32583	944.4623	31764	549.1264	54632
35000	2712.6848	12902	29.8172	1.0992	1019.8337	34319	1085.5719	32241	577.4621	60610
40000	3212.0554	12453	34.0681	1.0606	1246.4967	32089	1268.2426	31539	663.2481	60309
45000	3882.4854	11590	38.4813	0.9912	1474.4066	30520	1596.8812	28179	772.7162	58236
50000	4142.0000	12071	42.6215	1.0290	1588.8568	31469	1700.6177	29401	809.9041	61735
55000	4551.6738	12083	46.9017	1.0304	1763.5079	31187	1820.4570	30212	920.8069	59730
60000	4850.1289	12370	51.8676	1.0694	1899.4812	31587	1895.8611	31647	1002.9191	59825
65000	5166.3325	12581	55.9397	1.0828	1994.8563	32583	2026.9092	32068	1088.6274	59708
70000	5509.1060	12706	59.8340	1.0861	2137.9033	32742	2158.2810	32433	1153.0878	60706
75000	5941.0371	12624	64.1971	1.0806	2228.1565	33660	2381.5732	31491	1267.1104	59189
80000	6092.4902	13130	69.1647	1.1352	2263.5217	35343	2401.6077	33311	1358.1964	58901
85000	6843.0752	12421	72.4547	1.0588	2591.9583	32793	2717.9646	31273	1460.6975	58191
90000	7790.6494	11552	77.0484	0.9890	2971.4360	30288	3147.9863	28589	1594.1790	56455
95000	7960.1816	11934	81.3665	1.0222	3185.1819	29825	3057.0425	31075	1636.5911	58047
100000	8172.5576	12236	84.9118	1.0390	3025.9670	33047	3284.4495	30446	1777.2292	56267
105000	8256.7666	12716	89.4575	1.0834	3114.5698	33712	3255.2014	32256	1797.5376	58413
110000	8604.3262	12784	93.6119	1.0880	3248.9712	33856	3372.5037	32616	1889.2390	58224
115000	9307.7627	12355	98.0638	1.0536	3421.7407	33608	3678.2400	31264	2109.7175	54509
120000	9707.1914	12361	102.8371	1.0594	3626.1108	33093	3805.8342	31530	2172.4087	55238
125000	10170.6338	12290	106.8273	1.0504	3770.1509	33155	4068.9075	30720	2224.7480	56186
130000	10586.6836	12279	110.5345	1.0441	3870.0188	33591	4201.7065	30939	2404.4233	54067
135000	10588.8213	12749	114.6674	1.0829	3861.5488	34960	4136.0449	32639	2476.5605	54511
140000	11316.3975	12371	119.0656	1.0522	4177.4009	33513	4413.8516	31718	2606.0793	53720
145000	11707.5137	12385	123.6471	1.0561	4558.4116	31809	4474.2495	32407	2551.2056	56835
150000	12112.6631	12383	127.4555	1.0522	4389.9683	34168	4708.4478	31857	2886.7913	51960
155000	11745.9600	13196	132.3758	1.1270	4339.3096	35719	4616.6655	33574	2657.6099	58323
160000	12386.4971	12917	136.3263	1.1006	4430.4214	36113	4958.4360	32268	2861.3132	55918
165000	13110.3018	12585	140.1806	1.0692	5020.2900	32866	5080.1543	32479	2869.6772	57497
170000	13698.6367	12409	144.7160	1.0564	4867.2007	34927	5399.3677	31485	3287.3516	51713
175000	14136.2256	12379	154.7563	1.0947	5265.6050	33234	5573.1035	31400	3142.7610	55683
180000	13876.2627	12971	153.4724	1.1060	5083.0972	35411	5452.1670	33014	3187.5256	56470
185000	14585.6660	12683	157.0597	1.0768	5367.1895	34468	5746.5630	32193	3314.8535	55809
190000	14984.4277	12679	161.6202	1.0786	5559.3862	34176	5929.5781	32042	3333.8433	56991
195000	15520.7393	12563	166.1237	1.0703	5592.6201	34867	6069.7363	32126	3692.2590	52813
200000	15979.9170	12515	170.8571	1.0692	5994.7134	33362	6006.8408	33295	3807.5059	52527

TABLE VII
KYBER COMPARISON BATCH / OPTIMIZED - TOTAL & KEY-GENERATION

batchsize	total/s_opt	total/s	dif_total %	kgen/s_opt	kgen/s	dif_kgen %
unit	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*
1	56	65	13.85	122	159	23.27
2	136	137	0.73	330	334	1.20
4	272	273	0.37	665	665	0.00
8	538	538	0.00	1307	1310	0.23
16	1042	1045	0.29	2527	2534	0.28
32	1913	1957	2.25	4617	4738	2.55
64	3441	3747	8.17	8066	9068	11.05
128	6529	6839	4.53	15582	16668	6.52
256	12332	13351	7.63	30321	33190	8.64
512	23824	25505	6.59	58322	65666	11.18
1024	46650	47403	1.59	115300	127061	9.26
2048	84221	86230	2.33	215192	232080	7.28
4096	148181	150323	1.42	398843	430380	7.33
8192	226931	246523	7.95	689655	687978	-0.24
16384	400721	351532	-13.99	1123201	1045229	-7.46
32768	419446	456258	8.07	1488441	1461876	-1.82
5000	168698	173140	2.57	473140	500047	5.38
10000	261371	285715	8.52	812447	815944	0.43
15000	359470	326466	-10.11	1037351	959143	-8.15
20000	453319	408581	-10.95	1291493	1217548	-6.07
25000	379047	405958	6.63	1268197	1302695	2.65
30000	477926	432463	-10.51	1406812	1367715	-2.86
35000	477970	461621	-3.54	1523973	1479420	-3.01
40000	540605	486909	-11.03	1612353	1570188	-2.69
45000	442426	487684	9.28	1560447	1553199	-0.47
50000	532244	509789	-4.40	1632556	1616269	-1.01
55000	482608	524917	8.06	1645606	1619243	-1.63
60000	545855	532011	-2.60	1712866	1693276	-1.16
65000	496632	516580	3.86	1777706	1755159	-1.28
70000	487995	493130	1.04	1788810	1754420	-1.96
75000	507473	509705	0.44	1850952	1827553	-1.28
80000	578318	520278	-11.16	1903120	1880110	-1.22
85000	520660	526020	1.02	1898102	1871935	-1.40
90000	540090	542131	0.38	1965617	1947883	-0.91
95000	540338	548690	1.52	2018349	1992549	-1.29
100000	550374	556808	1.16	2034081	1997268	-1.84
105000	555403	568164	2.25	2074681	2052776	-1.07
110000	565908	574846	1.55	2122039	2097732	-1.16
115000	570025	578218	1.42	2094181	2098989	0.23
120000	574648	587767	2.23	2114954	2139434	1.14
125000	564993	585982	3.58	2047747	2132351	3.97
130000	628960	594269	-5.84	2027551	2174273	6.75
135000	558229	597874	6.63	2049019	2208189	7.21
140000	561385	577942	2.86	2061500	2092129	1.46
145000	601224	591324	-1.67	2110432	2113117	0.13
150000	628623	590629	-6.43	2140457	2144689	0.20
155000	555475	592753	6.29	2123200	2139184	0.75
160000	557820	596431	6.47	2139590	2166363	1.24
165000	612770	597101	-2.62	2153260	2184629	1.44
170000	575739	590030	2.42	2145542	2159235	0.63
175000	592942	599555	1.10	2156937	2166634	0.45
180000	575839	595863	3.36	2163621	2164697	0.05
185000	660623	601910	-9.75	2163822	2166382	0.12
190000	638355	602875	-5.89	2182698	2183285	0.03
195000	654064	603776	-8.33	2185882	2179616	-0.29
200000	581962	607092	4.14	2199465	2186463	-0.59

TABLE VIII
KYBER COMPARISON BATCH / OPTIMIZED - ENCAPSULATION & DECAPSULATION

batchsize	enc/s_opt	enc/s	dif_enc %	dec/s_opt	dec/s	dif_dec %
unit	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*
1	133	143	6.99	509	516	1.36
2	298	300	0.67	1069	1069	0.00
4	596	597	0.17	2128	2129	0.05
8	1179	1180	0.08	4204	4202	-0.05
16	2298	2299	0.04	8133	8258	1.51
32	4265	4320	1.27	15016	15692	4.31
64	7741	8352	7.32	29990	30558	1.86
128	14763	15369	3.94	56433	56590	0.28
256	28613	30639	6.61	102124	112781	9.45
512	55778	60585	7.93	200964	223674	10.15
1024	107411	116394	7.72	390015	436605	10.67
2048	194666	207212	6.05	766871	865962	11.44
4096	369378	394029	6.26	1376936	1539758	10.57
8192	645161	649386	0.65	2429395	2445502	0.66
16384	1052147	985800	-6.73	4287245	3680630	-16.48
32768	1476287	1452006	-1.67	5059288	4975116	-1.69
5000	440091	469300	6.22	1594127	1778400	10.36
10000	761096	765629	0.59	2887529	2918937	1.08
15000	975911	911714	-7.04	3959037	3401084	-16.41
20000	1229695	1168504	-5.24	4983733	4398781	-13.30
25000	1244345	1265465	1.67	4271926	4462441	4.27
30000	1386965	1355871	-2.29	4806706	4684833	-2.60
35000	1520786	1477498	-2.93	5298354	5071705	-4.47
40000	1614219	1574734	-2.51	5490161	5312927	-3.34
45000	1575835	1565078	-0.69	5339000	5330096	-0.17
50000	1627604	1639526	0.73	5537324	5355499	-3.40
55000	1674333	1661333	-0.78	5652592	5536005	-2.11
60000	1749537	1735761	-0.79	5710891	5626366	-1.50
65000	1823410	1801874	-1.20	5857933	5720783	-2.40
70000	1848249	1805041	-2.39	6049502	5803239	-4.24
75000	1914528	1887845	-1.41	6242941	6046178	-3.25
80000	1972903	1942560	-1.56	6411571	6147502	-4.30
85000	1956716	1932262	-1.27	6536048	6256372	-4.47
90000	2030885	2012942	-0.89	6722036	6492810	-3.53
95000	2079888	2063240	-0.81	6955052	6644104	-4.68
100000	2105023	2067703	-1.80	7129233	6753717	-5.56
105000	2152870	2125167	-1.30	7263002	6914760	-5.04
110000	2201944	2177020	-1.14	7445375	7097354	-4.90
115000	2191695	2172509	-0.88	7548372	7184525	-5.06
120000	2234909	2215261	-0.89	7733617	7297265	-5.98
125000	2157481	2173697	0.75	7010700	7280568	3.71
130000	2129085	2183330	2.48	6947579	7422237	6.40
135000	2167534	2219838	2.36	7094819	7502340	5.43
140000	2168211	2163792	-0.20	7221569	6948857	-3.92
145000	2212005	2193590	-0.84	7395882	6998156	-5.68
150000	2238282	2220884	-0.78	7427460	7073913	-5.00
155000	2213553	2210109	-0.16	7403628	7122416	-3.95
160000	2232238	2231236	-0.04	7429862	7173359	-3.58
165000	2253353	2251496	-0.08	7451572	7219081	-3.22
170000	2242999	2234844	-0.36	7419360	7122509	-4.17
175000	2255876	2248237	-0.34	7479471	7097533	-5.38
180000	2256614	2244245	-0.55	7500479	7076925	-5.99
185000	2271503	2251198	-0.90	7500479	7109106	-5.51
190000	2287059	2266594	-0.90	7529700	7146103	-5.37
195000	2287498	2267096	-0.90	7556733	7159282	-5.55
200000	2295093	2275353	-0.87	7593503	7185330	-5.68

XII. Measurements Dilithium

TABLE IX
DILITHIUM BATCH-MODE TEST SYSTEM 1

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_sign	sign/s	t_verify	verify/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	40.6584	24	0.0605	0.1489	9.4882	105	20.2358	49	10.8739	91
2	40.6191	49	0.0606	0.1492	9.4515	211	20.2285	98	10.8786	183
4	40.6739	98	0.0663	0.1630	9.4635	422	20.2349	197	10.9091	366
8	40.7782	196	0.0918	0.2251	9.5009	842	20.2537	394	10.9317	731
16	41.0344	389	0.1277	0.3111	9.6384	1660	20.3031	788	10.9652	1459
32	41.4824	771	0.2162	0.5212	9.8620	3244	20.3894	1569	11.0148	2905
64	42.4062	1509	0.3984	0.9396	10.5090	6090	20.4781	3125	11.0207	5807
128	42.7317	2995	0.7022	1.6432	10.5252	12161	20.4828	6249	11.0216	1163
256	43.5067	5884	1.2987	2.9851	10.5829	24189	20.5718	12444	11.0532	2316
512	47.7711	10717	2.5091	5.2524	11.7096	43724	22.3550	22903	11.1974	45724
1024	53.7288	19058	3.3244	6.1873	13.9214	73555	24.2654	42199	12.2176	83813
2048	57.7429	35467	4.3888	7.6005	14.7963	138412	25.0385	81794	13.5193	15148
4096	63.4852	64518	7.5230	11.8499	16.7250	244902	25.4106	161192	13.8266	296239
8192	72.6266	112796	10.5810	14.5691	21.1136	387996	26.9723	303718	13.9597	586831
16384	92.6675	176804	16.7411	18.0658	28.0630	583829	33.5628	488159	14.3005	1145690
32768	141.7858	231109	40.9909	28.9104	39.5591	828331	46.4446	705528	14.7913	2215357
5000	64.4912	77529	7.7946	12.0863	17.2860	289251	25.4990	196085	13.9116	359413
10000	75.3525	132709	12.2230	16.2211	21.8596	457465	27.2097	367515	14.0602	711227
15000	91.7199	163541	16.8506	18.3718	27.5034	545386	33.1842	452021	14.1816	1057706
20000	97.8291	204438	19.5153	19.9484	29.4184	679846	34.5038	579646	14.3916	1389699
25000	112.3024	222613	25.1966	22.4364	33.4053	748383	39.1484	638596	14.5521	1717961
30000	130.6891	229552	32.6095	24.9520	38.3214	782851	45.0520	665896	14.7062	2039961
35000	145.4059	240705	41.5909	28.6033	41.2643	848189	47.6659	734277	14.8847	2351401
40000	155.2897	257583	48.2181	31.0504	43.0070	930081	49.0262	815889	15.0384	2659851
45000	164.8985	272895	53.3170	32.3332	45.6267	986264	50.6709	888084	15.2839	2944268
50000	174.6841	286230	59.0357	33.7957	48.0678	1040197	51.9446	962563	15.6359	3197768
55000	182.9780	300582	64.5679	35.2872	49.8706	1102853	52.7402	1042848	15.7993	3481156
60000	192.9653	310936	69.2552	35.8899	53.0633	1130725	54.6168	1098562	16.0301	3742961
65000	202.0213	321748	73.7394	36.5008	55.7606	1165697	56.1504	1157606	16.3709	3970451

TABLE X
DILITHIUM BATCH-MODE TEST SYSTEM 2

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_sign	sign/s	t_verify	verify/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	34.1396	29	0.0833	0.2439	8.1882	122	16.5769	60	9.2913	107
2	33.3136	60	0.0752	0.2258	7.8145	255	16.3680	122	9.0558	220
4	35.9598	111	0.0795	0.2211	8.3565	478	17.6141	227	9.9097	403
8	35.9018	222	0.0826	0.2302	8.1418	982	17.6902	452	9.9871	801
16	35.4515	451	0.0967	0.2727	8.2059	1949	17.3491	922	9.7998	1632
32	34.9869	914	0.1205	0.3444	8.2191	3893	17.1703	1863	9.4770	3376
64	38.6139	1657	0.1668	0.4319	9.7376	6572	18.8382	3397	9.8714	6483
128	41.6940	3069	0.2610	0.6260	10.9703	11667	20.2108	6333	10.2518	12485
256	42.8193	5978	0.4809	1.1230	11.2969	22661	20.6007	12426	10.4409	24519
512	44.3323	11549	0.8898	2.0071	11.9402	42880	20.9327	24459	10.5697	48440
1024	46.3656	22085	1.6717	3.6055	12.9786	78899	21.0587	48626	10.6566	96090
2048	50.6969	40396	3.1881	6.2886	14.7873	138497	21.8693	93647	10.8522	188718
4096	64.1890	63811	6.2454	9.7298	19.6105	208867	27.0409	151474	11.2922	362729
8192	86.4045	94809	12.3571	14.3015	27.3501	299523	34.5994	236767	12.0979	677141
16384	139.0216	117852	24.4626	17.5962	53.0422	308885	47.9267	341855	13.5900	1205588
32768	304.2335	107706	48.6255	15.9829	137.2565	238735	94.4990	346754	23.8525	1373778
5000	72.1398	69309	7.6598	10.6180	22.3280	223934	30.7970	162353	11.3550	440336
10000	93.7583	106657	14.9604	15.9564	30.4320	328601	36.1329	276755	12.2329	817464
15000	126.7944	118301	22.4018	17.6678	46.4866	322673	44.7497	335198	13.1564	1140133
20000	169.7237	117838	29.7192	17.5104	69.5774	287449	55.7942	358460	14.6329	1366781
25000	218.2737	114535	37.1129	17.0029	95.0311	263071	68.9442	362612	17.1855	1454710
30000	272.9353	109916	44.5320	16.3159	121.9921	245917	85.0557	352710	21.3555	1404789
35000	330.6040	105866	51.9453	15.7122	150.7306	232202	101.9103	343439	26.0178	1345232
40000	391.6747	102125	59.1902	15.1121	181.7639	220065	119.2558	335413	31.4647	1271267
45000	450.9153	99797	66.6290	14.7764	211.8344	212430	137.1977	327993	35.2541	1276445
50000	494.4860	101115	73.9352	14.9519	231.6891	215806	150.4987	332228	38.3630	1303339
55000	536.8186	102455	81.3091	15.1465	252.1489	218125	161.5662	340417	41.7943	1315967
60000	587.4916	102129	88.7300	15.1032	277.2401	216418	175.4154	342045	46.1059	1301351
65000	634.5479	102435	96.0654	15.1392	299.5397	216999	188.9058	344086	50.0370	1299039

TABLE XI
DILITHIUM BATCH-MODE TEST SYSTEM 1 (OPTIMIZED)

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_sign	sign/s	t_verify	verify/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	40.2796	24	0.0629	0.1561	9.3514	106	19.9909	50	10.8744	91
2	40.2410	49	0.0629	0.1563	9.3145	214	19.9855	100	10.8781	183
4	40.2901	99	0.0683	0.1695	9.3241	428	19.9905	200	10.9072	366
8	40.3870	198	0.0799	0.1978	9.3619	854	20.0132	399	10.9320	731
16	40.6533	393	0.1328	0.3267	9.4984	1684	20.0561	797	10.9659	1459
32	41.2862	775	0.2235	0.5414	9.7207	3291	20.3448	1572	10.9972	2909
64	41.7132	1534	0.3200	0.7672	10.0213	6386	20.3557	3144	11.0162	5809
128	42.0677	3042	0.6416	1.5252	10.0422	12746	20.3662	6284	11.0178	11617
256	43.2653	5916	1.3891	3.2107	10.1483	25225	20.5500	12457	11.1778	22902
512	46.7074	10961	2.1965	4.7027	11.0449	46356	22.2463	23015	11.2197	45633
1024	53.1202	19277	3.7865	7.1281	13.0504	78465	24.0646	42552	12.2188	83805
2048	57.3881	35686	5.2121	9.0822	13.8204	148186	24.8422	82440	13.5134	151553
4096	62.7656	65258	6.7586	10.7680	16.8498	243089	25.3579	161527	13.7994	296825
8192	70.4774	116235	10.3219	14.6456	19.9610	410399	26.1579	313174	14.0366	583616
16384	89.4897	183082	16.8309	18.8076	26.7423	612663	31.5683	519001	14.3483	1141879
32768	141.6276	231367	40.3271	28.4741	39.6010	827453	46.8375	699610	14.8620	2204813
5000	64.3588	77689	7.6319	11.8583	17.3446	288274	25.4688	196318	13.9135	359362
10000	73.2397	136538	12.0616	16.4687	20.6533	484183	26.3788	379092	14.1460	706915
15000	88.1637	170138	16.3978	18.5993	26.2255	571961	31.2212	480442	14.3191	1047551
20000	98.0645	203947	19.6351	20.0226	29.4410	679325	34.5469	578923	14.4415	1384897
25000	117.8842	212072	29.6386	25.1421	33.7741	740212	39.8879	626756	14.5836	1714258
30000	135.9017	220747	37.9239	27.9054	38.1607	786149	45.0636	665725	14.7535	2033419
35000	145.7201	240186	42.1133	28.9001	40.8171	857482	47.8438	731546	14.9458	2341794
40000	155.0304	258013	47.3831	30.5637	42.9631	931030	49.5225	807714	15.1617	2638224
45000	164.7757	273098	52.5558	31.8954	45.5649	987603	51.1888	879098	15.4662	2909565
50000	174.9643	285772	58.7850	33.5983	47.8466	1045006	52.5496	951481	15.7831	3167943
55000	183.1439	300310	63.3363	34.5828	49.9788	1100465	53.7247	1023737	16.1041	3415283
60000	193.3573	310306	68.3306	35.3390	53.1528	1128821	55.5266	1080563	16.3474	3670318
65000	202.6366	320771	73.6754	36.3584	55.5542	1170028	56.8183	1143997	16.5886	3918346

TABLE XII
DILITHIUM COMPARISON BATCH / OPTIMIZED

batchsize	total/s_opt	total/s	total_dif %	kgen/s_opt	kgen/s	kgen_dif %	sign/s_opt	sign/s	sign_dif %	verify/s_opt	verify/s	ver_dif %
unit	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*
1	24	24	0.00	106	105	-0.95	50	49	-2.04	91	91	0.00
2	49	49	0.00	214	211	-1.42	100	98	-2.04	183	183	0.00
4	99	98	-1.02	428	422	-1.42	200	197	-1.52	366	366	0.00
8	198	196	-1.02	854	842	-1.43	399	394	-1.27	731	731	0.00
16	393	389	-1.03	1684	1660	-1.45	797	788	-1.14	1459	1459	0.00
32	775	771	-0.52	3291	3244	-1.45	1572	1569	-0.19	2909	2905	-0.14
64	1534	1509	-1.66	6386	6090	-4.86	3144	3125	-0.61	5809	5807	-0.03
128	3042	2995	-1.57	12746	12161	-4.81	6284	6249	-0.56	11617	11613	-0.03
256	5916	5884	-0.54	25225	24189	-4.28	12457	12444	-0.10	22902	23160	1.11
512	10961	10717	-2.28	46356	43724	-6.02	23015	22903	-0.49	45633	45724	0.20
1024	19277	19058	-1.15	78465	73555	-6.68	42552	42199	-0.84	83805	83813	0.01
2048	35686	35467	-0.62	148186	138412	-7.06	82440	81794	-0.79	151553	151487	-0.04
4096	65258	64518	-1.15	243089	244902	0.74	161527	161192	-0.21	296825	296239	-0.20
8192	116235	112796	-3.05	410399	387996	-5.77	313174	303718	-3.11	583616	586831	0.55
16384	183082	176804	-3.55	612663	583829	-4.94	519001	488159	-6.32	1141879	1145690	0.33
32768	231367	231109	-0.11	827453	828331	0.11	699610	705528	0.84	2204813	2215357	0.48
5000	77689	77529	-0.21	288274	289251	0.34	196318	196085	-0.12	359362	359413	0.01
10000	136538	132709	-2.89	484183	457465	-5.84	379092	367515	-3.15	706915	711227	0.61
15000	170138	163541	-4.03	571961	545386	-4.87	480442	452021	-6.29	1047551	1057706	0.96
20000	203947	204438	0.24	679325	679846	0.08	578923	579646	0.12	1384897	1389699	0.35
25000	212072	222613	4.74	740212	748383	1.09	626756	638596	1.85	1714258	1717961	0.22
30000	220747	229552	3.84	786149	782851	-0.42	665725	665896	0.03	2033419	2039961	0.32
35000	240186	240705	0.22	857482	848189	-1.10	731546	734277	0.37	2341794	2351401	0.41
40000	258013	257583	-0.17	931030	930081	-0.10	807714	815889	1.00	2638224	2659851	0.81
45000	273098	272895	-0.07	987603	986264	-0.14	879098	888084	1.01	2909565	2944268	1.18
50000	285772	286230	0.16	1045006	1040197	-0.46	951481	962563	1.15	3167943	3197768	0.93
55000	300310	300582	0.09	1100465	1102853	0.22	1023737	1042848	1.83	3415283	3481156	1.89
60000	310306	310936	0.20	1128821	1130725	0.17	1080563	1098562	1.64	3670318	3742961	1.94
65000	320771	321748	0.30	1170028	1165697	-0.37	1143997	1157606	1.18	3918346	3970451	1.31

XIII. MEASUREMENTS FRODOKEM

TABLE XIII
FRODOKEM BATCH-MODE TEST SYSTEM 1

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_enc	enc/s	t_dec	dec/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	38.5282	25	0.0629	0.1633	9.0643	110	13.8797	72	15.5213	64
2	39.6403	50	0.0905	0.2284	9.3452	214	14.2947	139	15.9099	125
4	41.6263	96	0.1347	0.3235	9.3018	430	14.9785	267	17.2113	232
8	51.8512	154	0.2095	0.4041	10.0415	796	18.8969	423	22.7032	352
16	67.9152	235	0.3396	0.5001	12.9083	1239	24.5362	652	30.1310	531
32	95.1516	336	0.8045	0.8455	16.8729	1896	32.7906	975	44.6836	716
64	122.9067	520	1.5909	1.2944	22.7113	2817	41.3737	1546	57.2308	1118
128	125.2955	1021	3.3060	2.6386	22.7503	5626	41.6972	3069	57.5420	2224
256	128.5368	1991	5.2963	4.1204	23.0074	11126	42.3371	6046	57.8959	4421
512	132.7313	3857	7.3812	5.5610	23.3851	21894	42.9842	11911	58.9809	8680
1024	138.0603	7417	9.6035	6.9560	24.2171	42284	44.0014	23272	60.2384	16999
2048	159.3620	12851	13.0628	8.1969	24.5353	83471	52.8328	38763	68.9311	29710
4096	170.5682	24013	23.6208	13.8483	25.2458	162244	52.9734	77321	68.7282	59597
8192	273.7561	29924	41.5659	15.1836	44.4937	184115	79.4337	103130	108.2628	75667
16384	425.5444	38501	80.8605	19.0016	70.7905	231443	116.2929	140885	157.6005	103959
32768	738.9783	44342	160.9921	21.7858	122.6576	267150	197.0260	166313	258.3028	126858
5000	174.5532	28644	26.5659	15.2194	25.8837	193171	53.3617	93700	68.7419	72735
10000	286.2711	34931	50.5317	17.6517	45.8491	218106	80.5744	124108	109.3160	91477
15000	416.7977	35988	75.9074	18.2120	69.5920	215542	115.2757	130122	156.0227	96139
20000	459.6371	43512	100.6450	21.8966	76.1244	262727	121.4389	164691	161.4288	123893
25000	572.8196	43643	125.8106	21.9634	95.7048	261220	149.8498	166833	201.4545	124097
30000	720.3427	41646	157.3047	21.8375	120.6755	248600	190.6890	157324	251.6734	119202
35000	874.5765	40019	183.1097	20.9370	141.7441	246923	233.8881	149644	315.8344	110817

TABLE XIV
FRODOKEM BATCH-MODE TEST SYSTEM 2

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_enc	enc/s	t_dec	dec/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	959.3391	1	0.0737	0.0077	238.5698	4	350.7481	2	369.9475	2
2	957.8659	2	0.0915	0.0096	240.0732	8	348.0392	5	369.6620	5
4	966.7555	4	0.1093	0.0113	240.1852	16	352.1130	11	374.3479	10
8	969.9008	8	0.1715	0.0177	240.9932	33	351.7396	22	376.9964	21
16	981.7966	16	0.2741	0.0279	243.7244	65	355.8902	44	381.9079	41
32	986.7050	32	0.4397	0.0446	242.6390	131	357.9178	89	385.7085	82
64	1019.2513	62	0.8301	0.0814	246.8099	259	368.1877	173	403.4237	158
128	1031.5792	124	1.5365	0.1489	252.0013	507	372.6540	343	405.3874	315
256	1069.7479	239	3.0221	0.2825	270.8667	945	384.5278	665	411.3312	622
512	1141.0912	448	5.7630	0.5050	313.6710	1632	406.3066	1260	415.3506	1232
1024	1307.5991	783	11.4354	0.8745	421.9665	2426	456.3768	2243	417.8204	2450
2048	1699.5989	1204	22.9021	1.3475	671.4738	3050	581.6914	3520	423.5313	4835
4096	2641.9048	1550	45.4255	1.7194	1193.4158	3432	900.6451	4547	502.4180	8152
8192	5601.9502	1462	90.7848	1.6206	2544.5298	3219	1785.9147	4587	1180.7211	6938
16384	19463.3398	841	181.6830	0.9335	5811.8364	2819	5892.5356	2780	7577.2852	2162
32768	58638.9375	558	362.2607	0.6178	15281.4170	2144	15646.9629	2094	27348.2910	1198
5000	2956.6255	1691	54.9221	1.8576	1350.8523	3701	1020.4274	4899	530.4238	9426
10000	7579.1318	1319	110.1893	1.4539	2913.3118	3432	2546.5735	3926	2009.0574	4977
15000	16531.4531	907	164.7094	0.9963	4791.4229	3130	5277.1055	2842	6298.2183	2381
20000	25336.9004	789	220.2447	0.8693	7246.4775	2759	7468.0664	2678	10402.1104	1922
25000	35871.6641	696	275.1840	0.7671	10018.7686	2495	9830.1338	2543	15747.5801	1587
30000	48107.4219	623	330.7904	0.6876	13176.8330	2276	12344.7012	2430	22255.0996	1348
35000	74998.1172	466	386.9994	0.5160	17507.7266	1999	21692.1699	1613	35411.2227	988

TABLE XV
FRODOKEM BATCH-MODE TEST SYSTEM 1 (OPTIMIZED)

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_sign	enc/s	t_dec	dec/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	38.5450	25	0.0734	0.1903	9.0689	110	13.8828	72	15.5199	64
2	39.6694	50	0.0928	0.2339	9.3551	213	14.3027	139	15.9189	125
4	41.6538	96	0.1346	0.3232	9.3164	429	14.9887	266	17.2141	232
8	50.8357	157	0.2181	0.4290	9.7547	820	18.6251	429	22.2378	359
16	58.4116	273	0.4831	0.8270	9.8112	1630	22.2189	720	25.8984	617
32	59.0496	541	0.8935	1.5131	9.8827	3237	22.2820	1436	25.9914	1231
64	59.9922	1066	1.3868	2.3116	9.9755	6415	22.4542	2850	26.1757	2445
128	61.6793	2075	2.0859	3.3818	10.0994	12674	22.8135	5610	26.6804	4797
256	64.6459	3960	3.1380	4.8542	10.2816	24898	23.6617	10819	27.5645	9287
512	69.0627	7413	5.1683	7.4836	10.9039	46955	24.4461	20944	28.5444	17936
1024	75.9730	13478	6.9750	9.1809	12.6053	81235	25.9203	39505	30.4723	33604
2048	103.4252	19801	12.0478	11.6488	14.2588	143630	36.4072	56252	40.7114	50305
4096	134.8182	30381	23.2643	17.2560	20.3607	201171	43.0748	95090	48.1183	85123
8192	224.4496	36498	42.2745	18.8348	38.8840	210677	62.9335	130169	80.3576	101944
16384	438.9164	37328	80.4187	18.3221	80.0185	204752	119.3935	137226	159.0857	102988
32768	856.1691	38272	159.8638	18.6720	157.7876	207671	231.3676	141627	307.1503	106683
5000	145.8665	34277	27.1892	18.6398	22.2899	224316	44.8814	111404	51.5059	97076
10000	266.3033	37551	49.8935	18.7356	47.7122	209589	73.3942	136250	95.3033	104928
15000	411.1121	36486	73.9248	17.9817	76.1718	196923	112.6231	133187	148.3925	101083
20000	518.3678	38582	96.7384	18.6621	95.3749	209698	140.1164	142738	186.1382	107447
25000	645.8168	38710	121.9352	18.8808	119.3721	209429	173.2515	144298	231.2579	108104
30000	786.4292	38147	147.2185	18.7199	144.9455	206974	212.9184	140899	281.3468	106629
35000	912.8651	38340	171.7738	18.8170	167.5026	208951	248.4864	140852	325.1023	107658

TABLE XVI
FRODOKEM SINGLE-MODE

Test system 1	Time(μs): mean	Operations/s
Key generation	54736.842	18.2692308
Encapsulate	56888.889	17.57812497
Decapsulate	62058.824	16.11374395
Total	173684.555	5.757564338

Test system 2	Time(μs): mean	Operations/s
Key generation	1209.190	826.9999
Encapsulate	1014.199	985.9998
Decapsulate	1084.011	922.4999
Total	3307.400	302.3523

TABLE XVII
FRODOKEM COMPARISON BATCH / OPTIMIZED

batchsize	total/s_opt	total/s	total_dif %	kgen/s_opt	kgen/s	kgen_dif %	sign/s_opt	sign/s	sign_dif %	verify/s_opt	verify/s	ver_dif %
unit	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*
1	25	25	0.00	110	110	0.00	72	72	0.00	64	64	0.00
2	50	50	0.00	213	214	0.47	139	139	0.00	125	125	0.00
4	96	96	0.00	429	430	0.23	266	267	0.37	232	232	0.00
8	157	154	-1.95	820	796	-3.02	429	423	-1.42	359	352	-1.99
16	273	235	-16.17	1630	1239	-31.56	720	652	-10.43	617	531	-16.20
32	541	336	-61.01	3237	1896	-70.73	1436	975	-47.28	1231	716	-71.93
64	1066	520	-105.00	6415	2817	-127.72	2850	1546	-84.35	2445	1118	-118.69
128	2075	1021	-103.23	12674	5626	-125.28	5610	3069	-82.80	4797	2224	-115.69
256	3960	1991	-98.90	24898	11126	-123.78	10819	6046	-78.94	9287	4421	-110.07
512	7413	3857	-92.20	46955	21894	-114.47	20944	11911	-75.84	17936	8680	-106.64
1024	13478	7417	-81.72	81235	42284	-92.12	39505	23272	-69.75	33604	16999	-97.68
2048	19801	12851	-54.08	143630	83471	-72.07	56252	38763	-45.12	50305	29710	-69.32
4096	30381	24013	-26.52	201171	162244	-23.99	95090	77321	-22.98	85123	59597	-42.83
8192	36498	29924	-21.97	210677	184115	-14.43	130169	103130	-26.22	101944	75667	-34.73
16384	37328	38501	3.05	204752	231443	11.53	137226	140885	2.60	102988	103959	0.93
32768	38272	44342	13.69	207671	267150	22.26	141627	166313	14.84	106683	126858	15.90
5000	34277	28644	-19.67	224316	193171	-16.12	111404	93700	-18.89	97076	72735	-33.47
10000	37551	34931	-7.50	209589	218106	3.90	136250	124108	-9.78	104928	91477	-14.70
15000	36486	35988	-1.38	196923	215542	8.64	133187	130122	-2.36	101083	96139	-5.14
20000	38582	43512	11.33	209698	262727	20.18	142738	164691	13.33	107447	123893	13.27
25000	38710	43643	11.30	209429	261220	19.83	144298	166833	13.51	108104	124097	12.89
30000	38147	41646	8.40	206974	248600	16.74	140899	157324	10.44	106629	119202	10.55
35000	38340	40019	4.20	208951	246923	15.38	140852	149644	5.88	107658	110817	2.85

TABLE XVIII
FRODOKEM BATCH-MODE MULTISTREAM TEST SYSTEM 1 - TRANSFER TIME

	1 Stream			2 Streams			4 Streams			8 Streams			16 Streams		
batchsize	t_total	t_copy	copy_%	t_total	t_copy	copy_%	t_total	t_copy	copy_%	t_total	t_copy	copy_%	t_total	t_copy	copy_%
unit	*ms*	*ms*	*% of t_total*	*ms*	*ms*	*% of t_total*	*ms*	*ms*	*% of t_total*	*ms*	*ms*	*% of t_total*	*ms*	*ms*	*% of t_total*
16	67.3577	0.0060	0.0090	55.0485	0.2065	0.3751	42.6835	0.3400	0.7967	41.0821	0.5687	1.3844	41.5686	1.6119	3.8776
32	94.0931	0.0087	0.0092	68.3377	0.4028	0.5895	56.4310	0.5412	0.9591	43.8860	0.9813	2.2359	44.1238	2.7930	6.3300
64	121.2606	0.0061	0.0050	95.3724	0.7419	0.7779	69.7733	1.2165	1.7435	58.0857	1.8026	3.1033	47.7821	4.0122	8.3969
128	122.0393	0.0071	0.0058	123.3279	1.4853	1.2043	97.7691	1.6525	1.6902	72.8568	3.1288	4.2944	65.0813	7.1651	11.0095
256	123.2916	0.0072	0.0058	126.3744	3.1453	2.4888	126.3958	2.9164	2.3073	105.3588	7.3631	6.9886	84.1042	11.5959	13.7875
512	125.3635	0.0061	0.0049	129.9188	4.6332	3.5662	132.1737	6.7775	5.1277	135.3321	10.2620	7.5829	123.4124	22.7159	18.4065
1024	128.4775	0.0061	0.0048	134.2861	5.9512	4.4317	138.9899	10.6044	7.6297	147.3664	18.9435	12.8547	169.4021	40.8892	24.1374
2048	146.3372	0.0061	0.0042	154.8639	8.7151	5.6276	163.4435	17.4913	10.7018	181.8733	36.1320	19.8666	222.0963	76.5236	34.4551
4096	146.9695	0.0072	0.0049	159.8981	13.3216	8.3313	178.4606	31.9063	17.8787	217.0536	70.6866	32.5664	297.5826	151.0133	50.7467
8192	232.4652	0.0061	0.0026	245.9282	22.6699	9.2181	279.5170	61.7091	22.0770	356.8408	141.1806	39.5640	517.1345	300.4184	58.0929
16384	345.4478	0.0063	0.0018	380.2985	41.7659	10.9824	453.9977	122.2765	26.9333	601.6370	279.9089	46.5245	922.5081	598.8419	64.9145
32768	573.4236	0.0065	0.0011	647.0082	80.1683	12.3906	802.9085	243.0216	30.2677						
5000	148.0085	0.0062	0.0042	163.1351	15.4781	9.4879	186.6767	38.3937	20.5669	235.8290	86.8900	36.8445	332.9279	184.6778	55.4708
10000	235.8861	0.0060	0.0026	263.3298	27.2716	10.3564	311.2266	75.3931	24.2245	404.2658	171.3063	42.3747	599.7405	366.2757	61.0724
15000	342.0634	0.0063	0.0019	369.4891	38.8451	10.5132	432.2961	112.5709	26.0402	563.8914	257.8578	45.7283	856.8235	550.7893	64.2827
20000	360.5011	0.0064	0.0018	416.3097	50.6996	12.1783	511.5859	149.5392	29.2305	703.1785	342.4804	48.7046	1093.2487	731.7869	66.9369
25000	447.9945	0.0068	0.0015	512.1068	62.5725	12.2186	631.9361	186.8124	29.5619	867.7928	428.0362	49.3247	1358.6456	918.2068	67.5825
30000	558.8882	0.0070	0.0013	615.0608	73.9594	12.0247	755.8972	222.6738	29.4582						
35000	659.5712	0.0065	0.0010	714.6548	85.3372	11.9410									

TABLE XIX
FRODOKEM BATCH-MODE MULTISTREAM TEST SYSTEM 1 - TOTALS/S

	1 Stream		2 Streams		4 Streams		8 Streams		16 Streams	
batchsize	t_total	total/s	t_total	total/s	t_total	total/s	t_total	total/s	t_total	total/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
16	67.3577	237	55.0485	290	42.6835	374	41.0821	389	41.5686	384
32	94.0931	340	68.3377	468	56.4310	567	43.8860	729	44.1238	725
64	121.2606	527	95.3724	671	69.7733	917	58.0857	1101	47.7821	1339
128	122.0393	1048	123.3279	1037	97.7691	1309	72.8568	1756	65.0813	1966
256	123.2916	2076	126.3744	2025	126.3958	2025	105.3588	2429	84.1042	3043
512	125.3635	4084	129.9188	3940	132.1737	3873	135.3321	3783	123.4124	4148
1024	128.4775	7970	134.2861	7625	138.9899	7367	147.3664	6948	169.4021	6044
2048	146.3372	13995	154.8639	13224	163.4435	12530	181.8733	11260	222.0963	9221
4096	146.9695	27869	159.8981	25616	178.4606	22951	217.0536	18870	297.5826	13764
8192	232.4652	35239	245.9282	33310	279.5170	29307	356.8408	22957	517.1345	15841
16384	345.4478	47428	380.2985	43081	453.9977	36088	601.6370	27232	922.5081	17760
32768	573.4236	57144	647.0082	50645	802.9085	40811	235.8290	21201	332.9279	15018
5000	148.0085	33781	163.1351	30649	186.6767	26784	404.2658	24736	599.7405	16673
10000	235.8861	42393	263.3298	37975	311.2266	32130	563.8914	26600	856.8235	17506
15000	342.0634	43851	369.4891	40596	432.2961	34698	703.1785	28442	1093.2487	18294
20000	360.5011	55478	416.3097	48041	511.5859	39094	867.7928	28808	1358.6456	18400
25000	447.9945	55804	512.1068	48817	631.9361	39560				
30000	558.8882	53677	615.0608	48775	755.8972	39687				
35000	659.5712	53064	714.6548	48974						

TABLE XX
FRODOKEM BATCH-MODE MULTISTREAM TEST SYSTEM 1 - KEY-GENERATION/S

	1 Stream		2 Streams		4 Streams		8 Streams		16 Streams	
batchsize	t_gen	gen/s	t_gen	gen/s	t_gen	gen/s	t_gen	gen/s	t_gen	gen/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
16	12.8265	1247	10.1465	1576	9.4460	1693	9.5493	1675	9.3797	1705
32	16.7723	1907	13.0127	2459	10.2531	3121	9.5142	3363	9.7189	3292
64	22.6869	2821	16.9865	3767	13.1123	4880	10.3185	6202	9.7249	6581
128	22.7912	5616	22.8363	5605	17.2450	7422	13.2920	9629	10.6245	12047
256	23.0178	11121	23.0018	11129	23.1168	11074	17.4884	14638	13.7649	18598
512	23.3899	21889	23.4010	21879	23.4617	21822	23.4596	21824	18.1072	28276
1024	24.2206	42278	24.2636	42203	24.2328	42256	24.2227	42274	24.2183	42282
2048	24.5445	83440	24.5882	83292	24.5512	83417	24.5663	83366	24.5776	83327
4096	25.2583	162164	25.2050	162507	25.1856	162632	25.1964	162563	25.1791	162674
8192	44.6471	183483	41.2062	198805	39.1517	209237	39.1325	209339	39.1109	209455
16384	70.8370	231291	69.7778	234802	67.9320	241182	67.1360	244041	67.1709	243915
32768	123.1842	266008	124.5763	263035	122.8356	266762				
5000	25.9040	193020	25.8931	193101	25.8716	193262	25.9065	193002	25.9486	192688
10000	45.8885	217919	46.5385	214875	46.5205	214959	45.4076	220227	45.5065	219748
15000	69.7965	214910	67.8256	221155	64.7707	231586	63.1081	237687	62.3205	240691
20000	76.1461	262652	79.0298	253069	77.9644	256527	78.0561	256226	78.4359	254985
25000	96.1434	260028	97.0858	257504	95.7543	261084	94.9378	263330	95.2474	262474
30000	120.8072	248329	116.4541	257612	114.9119	261069				
35000	140.6984	248759	132.9426	263271						

TABLE XXI
FRODOKEM BATCH-MODE MULTISTREAM TEST SYSTEM 1 - ENCAPSULATION/S

	1 Stream		2 Streams		4 Streams		8 Streams		16 Streams	
batchsize	t_enc	enc/s	t_enc	enc/s	t_enc	enc/s	t_enc	enc/s	t_enc	enc/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
16	24.4496	654	20.3898	784	15.3065	1045	14.6270	1093	14.4011	1111
32	32.6841	979	24.6248	1299	20.8081	1537	15.5680	2055	14.9227	2144
64	41.3425	1548	32.8527	1948	24.8894	2571	20.9241	3058	15.8564	4036
128	41.7198	3068	41.5917	3077	33.4269	3829	25.3000	5059	21.5199	5947
256	42.3371	6046	42.3396	6046	42.3565	6043	34.1821	7489	26.4327	9684
512	42.9826	11911	42.9782	11913	43.0146	11902	42.8457	11949	35.1843	14551
1024	44.0051	23270	43.8828	23334	43.9806	23282	43.9989	23273	44.0093	23267
2048	52.8406	38758	52.7235	38844	52.6826	38874	52.6743	38880	52.5153	38998
4096	52.9687	77328	52.7793	77606	52.8387	77518	52.7263	77684	52.7323	77675
8192	79.4256	103140	77.4360	105790	75.5184	108476	75.3542	108713	74.5727	109852
16384	117.0178	140012	113.4190	144455	111.1380	147420	108.8857	150469	108.6619	150779
32768	194.5649	168416	191.1494	171426	188.4349	173895				
5000	53.3620	93699	53.1560	94062	53.0653	94223	52.9854	94365	53.3244	93765
10000	80.6510	123990	80.4814	124252	80.2122	124669	78.5403	127323	78.4504	127469
15000	115.7137	129630	110.3422	135940	106.8859	140336	103.7307	144605	103.3671	145113
20000	122.2783	163561	123.8064	161542	122.6707	163038	121.5229	164577	121.8615	164120
25000	150.2370	166403	151.2749	165262	149.4237	167309	146.7536	170353	147.1111	169939
30000	188.3978	159237	181.6103	165188	178.7120	167867				
35000	223.0643	156905	210.5056	166266						

TABLE XXII
FRODOKEM BATCH-MODE MULTISTREAM TEST SYSTEM 1 - DECAPSULATION/S

	1 Stream		2 Streams		4 Streams		8 Streams		16 Streams	
batchsize	t_dec	dec/s	t_dec	dec/s	t_dec	dec/s	t_dec	dec/s	t_dec	dec/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
16	30.0756	531	24.3057	658	17.5910	909	16.3371	979	16.1758	989
32	44.6280	717	30.2973	1056	24.8285	1288	17.8225	1795	16.6892	1917
64	57.2251	1118	44.7913	1428	30.5551	2094	25.0405	2555	18.1886	3518
128	57.5211	2225	57.4146	2229	45.4447	2816	31.1361	4110	25.7717	4966
256	57.9295	4419	57.8878	4422	58.0060	4413	46.3252	5526	32.3108	7923
512	58.9848	8680	58.9065	8691	58.9200	8689	58.7648	8712	47.4049	10800
1024	60.2457	16997	60.1885	17013	60.1720	17017	60.2014	17009	60.2853	16985
2048	68.9461	29704	68.8372	29751	68.7183	29802	68.5007	29897	68.4799	29906
4096	68.7353	59590	68.5921	59715	68.5299	59769	68.4444	59844	68.6579	59658
8192	108.3864	75581	104.6160	78305	103.1380	79427	101.1735	80969	103.0327	79508
16384	157.5868	103968	155.3357	105474	152.6512	107329	145.7067	112445	147.8333	110827
32768	255.6683	128166	251.1142	130490	248.6163	131801				
5000	68.7362	72741	68.6078	72878	69.3462	72102	70.0472	71380	68.9772	72487
10000	109.3404	91457	109.0382	91710	109.1010	91658	109.0115	91733	109.5079	91317
15000	156.5469	95817	152.4763	98375	148.0687	101304	139.1948	107762	140.3465	106878
20000	162.0702	123403	162.7738	122869	161.4118	123906	161.1191	124131	161.1641	124097
25000	201.6073	124003	201.1735	124270	199.9455	125034	198.0649	126221	198.0803	12621
30000	249.6763	120155	243.0371	123437	239.5996	125208				
35000	295.8023	118322	285.8695	122433						

XIV. MEASUREMENTS SPHINCS$^+$

TABLE XXIII
SPHINCS$^+$ BATCH-MODE TEST SYSTEM 1

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_sign	sign/s	t_verify	verify/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	890.2712	1	0.0739	0.0083	157.3506	6	714.5047	1	18.3425	54
2	912.1598	2	0.0857	0.0094	159.4837	12	727.0858	2	25.5046	78
4	949.6228	4	0.1092	0.0115	165.0946	24	752.9933	5	31.4259	127
8	1040.2534	7	0.1634	0.0157	181.4025	44	823.2211	9	35.4663	225
16	1173.4286	13	0.2935	0.0250	205.8752	77	927.3553	17	39.9048	400
32	1441.5236	22	0.5242	0.0364	255.9111	125	1137.8076	28	47.2806	676
64	1658.7515	38	0.9571	0.0577	297.0190	215	1309.8391	48	50.9360	1256
128	1659.5322	77	1.7575	0.1059	296.9622	431	1309.8226	97	50.9902	2510
256	1661.5609	154	3.5455	0.2134	297.0228	861	1309.9855	195	51.0066	5018
512	1667.0671	307	4.8951	0.2936	297.2766	1722	1313.7930	389	51.1023	10019
1024	1672.0125	612	7.5455	0.4513	298.0065	3436	1315.1763	778	51.2843	19967
2048	1685.7427	1214	11.2378	0.6666	299.3398	6841	1323.6038	1547	51.5613	39719
4096	1690.9581	2422	16.8484	0.9964	299.2817	13686	1323.1229	3095	51.7049	79218
8192	2301.5425	3559	29.1309	1.2657	410.0783	19976	1795.6598	4562	66.6733	122867
16384	2847.4424	5753	51.7460	1.8173	520.8193	31458	2193.1230	7470	81.7543	200405
32768	4391.8389	7461	96.2945	2.1926	770.9704	42502	3398.7534	9641	125.8197	260436
5000	1677.9727	2979	19.5585	1.1656	296.8977	16840	1309.8900	3817	51.6259	96850
10000	2313.1230	4323	34.0571	1.4723	411.8306	24281	1800.2968	5554	66.9381	149391
15000	2829.9841	5300	46.5099	1.6435	516.6281	29034	2185.4797	6863	81.3663	184351
20000	2974.9248	6722	62.1983	2.0908	540.4371	37007	2287.1921	8744	85.0963	235027
25000	3671.5554	6809	75.0189	2.0432	661.7695	37777	2830.7368	8831	104.0289	240317
30000	4402.7505	6813	88.3330	2.0063	775.8043	38669	3414.5872	8785	124.0264	241883
35000	5155.6655	6788	102.6900	1.9918	908.7551	38514	3997.0388	8756	147.1816	237801
40000	5256.0752	7610	116.8385	2.2229	920.8984	43435	4067.1086	9834	151.2287	264500
45000	7411.3779	6071	130.8363	1.7653	1453.6028	30957	5612.1035	8018	214.8385	209459
50000	8740.0781	5720	143.0082	1.6362	1736.5186	28793	6614.9316	7558	245.6179	203568

TABLE XXIV
SPHINCS$^+$ BATCH-MODE TEST SYSTEM 2

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_sign	sign/s	t_verify	verify/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	3519.3801	0	0.1214	0.0034	121.9408	8	3223.3940	0	173.9237	5
2	3658.5364	0	0.1121	0.0031	122.4735	16	3294.5566	0	241.3945	8
4	3790.2708	1	0.0912	0.0024	121.0065	33	3376.5366	1	292.6360	13
8	4100.5039	1	0.1099	0.0027	130.1040	61	3639.6492	2	330.6409	24
16	4495.0249	3	0.1981	0.0044	142.8497	112	3983.8235	4	368.1539	43
32	5254.9326	6	0.2674	0.0051	168.7656	189	4669.8193	6	416.0802	76
64	6141.1196	10	0.4535	0.0074	198.1645	322	5474.4932	11	468.0081	136
128	6191.1533	20	0.8245	0.0133	200.1975	639	5517.9014	23	472.2292	271
256	6200.0688	41	1.6095	0.0260	199.9376	1280	5526.7969	46	471.7242	542
512	6210.1265	82	3.0528	0.0492	201.0034	2547	5532.9292	92	473.1414	1082
1024	6287.8872	162	5.7267	0.0911	203.2657	5037	5602.1260	182	476.7691	2147
2048	7248.4658	282	11.4755	0.1583	237.1894	8634	6491.4878	315	508.3134	4029
4096	10732.6025	381	22.5422	0.2100	358.1412	11436	9619.6221	425	732.2968	5593
8192	18499.7227	442	45.0606	0.2436	614.1050	13339	16634.3438	492	1206.2122	6791
16384	37842.9297	432	89.8665	0.2375	1235.0972	13265	33653.1133	486	2864.8525	5718
32768	68835.7422	476	179.4356	0.2607	2264.5474	14469	61321.9180	534	5069.8486	6463
5000	11064.5361	451	27.7505	0.2508	367.9688	13588	9918.5215	504	750.2958	6664
10000	20144.5020	496	63.1385	0.3134	690.8925	14474	18079.8105	553	1310.6602	7629
15000	30547.1367	491	82.1990	0.2691	1001.7503	14973	27194.4375	551	2268.7480	6611
20000	43156.1797	463	110.1716	0.2553	1432.4063	13962	38382.6133	521	3230.9927	6190
25000	50543.9688	494	137.2317	0.2715	1657.5199	15082	44988.2109	555	3761.0078	6647
30000	60598.4063	495	164.4498	0.2714	1991.2639	15065	54024.8672	555	4417.8218	6790
35000	70540.6406	496	191.4415	0.2714	2314.2415	15123	62826.6055	557	5208.3491	6719
40000	80342.0156	497	219.5303	0.2732	2639.3579	15155	71616.8516	558	5866.2798	6818
45000	90670.7891	496	246.5184	0.2719	2980.6294	15097	80793.1875	556	6650.4399	6766
50000	100552.9844	497	273.7172	0.2722	3315.9500	15078	89548.4766	558	7414.8438	6743

TABLE XXV
SPHINCS$^+$ BATCH-MODE TEST SYSTEM 1 (OPTIMIZED)

batchsize	t_total	total/s	t_copy	copy_%	t_kgen	kgen/s	t_sign	sign/s	t_verify	verify/s
unit	*ms*	$\lfloor EPS \rfloor$	*ms*	*% of t_total*	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$	*ms*	$\lfloor EPS \rfloor$
1	4610.9722	0	0.0845	0.0018	157.4623	6	4225.7231	0	227.7025	4
2	4843.6670	0	0.1197	0.0025	159.4732	12	4366.8379	0	317.2370	6
4	5135.7222	0	0.1473	0.0029	165.0961	24	4577.2358	0	393.2433	10
8	5144.9209	1	0.2315	0.0045	165.0997	48	4580.6499	1	398.9391	20
16	5154.3628	3	0.4266	0.0083	165.0934	96	4586.5518	3	402.2913	39
32	5159.3135	6	0.8330	0.0161	165.0942	193	4586.5518	6	406.8346	78
64	5209.6055	12	1.7444	0.0335	166.6646	384	4630.9453	13	410.2515	156
128	5217.6416	24	3.7224	0.0713	166.6449	768	4634.1235	27	413.1506	309
256	5210.8301	49	5.2672	0.1011	166.2423	1539	4624.2026	55	415.1183	616
512	5380.3970	95	8.0810	0.1502	173.1098	2957	4781.5796	107	417.6262	1225
1024	5326.6807	192	10.1794	0.1911	170.0244	6022	4726.0093	216	420.4669	2435
2048	8135.3657	251	15.9124	0.1956	260.3910	7865	7216.9893	283	642.0737	3189
4096	11252.5039	364	25.7931	0.2292	359.0355	11408	9972.3135	410	895.3627	4574
8192	18086.2813	452	43.8559	0.2425	578.5584	14159	16043.1152	510	1420.7507	5765
16384	34568.6914	473	81.2091	0.2349	1106.7827	14803	30687.1836	533	2693.5151	6082
32768	64352.3672	509	145.5595	0.2262	2059.0156	15914	57121.8828	573	5025.9082	6519
5000	11309.7881	442	30.5798	0.2704	360.7349	13860	10017.5088	499	900.9640	5549
10000	21371.7910	467	53.4309	0.2500	683.3720	14633	18957.9316	527	1677.0576	5962
15000	31287.7461	479	78.0021	0.2493	1000.5560	14991	27764.9492	540	2444.2385	6136
20000	41174.8086	485	91.9018	0.2232	1318.0950	15173	36556.7852	547	3208.0308	6234
25000	51065.9180	489	118.1429	0.2314	1635.6102	15284	45341.6797	551	3970.4878	6296
30000	60939.4805	492	136.6588	0.2243	1952.1390	15367	54121.7578	554	4728.9287	6343
35000	70806.3594	494	154.8548	0.2187	2269.0881	15424	62895.6758	556	5486.7446	6379
40000	80666.2344	495	187.1805	0.2320	2585.8503	15468	71651.6094	558	6241.5957	6408
45000	90410.8594	497	190.6154	0.2108	2900.3601	15515	80347.3906	560	6972.4985	6453
50000	97344.7188	513	193.5379	0.1988	3116.8838	16041	86445.6172	578	7588.6729	6588

TABLE XXVI
SPHINCS$^+$ COMPARISON BATCH / OPTIMIZED

batchsize	total/s_opt	total/s	total_dif %	kgen/s_opt	kgen/s	kgen_dif %	sign/s_opt	sign/s	sign_dif %	verify/s_opt	verify/s	ver_dif %
unit	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*	$\lfloor EPS \rfloor$	$\lfloor EPS \rfloor$	*dif in %*
1	0	1	100.00	6	6	0.00	0	1	100.00	4	54	92.59
2	0	2	100.00	12	12	0.00	0	2	100.00	6	78	92.31
4	0	4	100.00	24	24	0.00	0	5	100.00	10	127	92.13
8	1	7	85.71	48	44	-9.09	1	9	88.89	20	225	91.11
16	3	13	76.92	96	77	-24.68	3	17	82.35	39	400	90.25
32	6	22	72.73	193	125	-54.40	6	28	78.57	78	676	88.46
64	12	38	68.42	384	215	-78.60	13	48	72.92	156	1256	87.58
128	24	77	68.83	768	431	-78.19	27	97	72.16	309	2510	87.69
256	49	154	68.18	1539	861	-78.75	55	195	71.79	616	5018	87.72
512	95	307	69.06	2957	1722	-71.72	107	389	72.49	1225	10019	87.77
1024	192	612	68.63	6022	3436	-75.26	216	778	72.24	2435	19967	87.80
2048	251	1214	79.32	7865	6841	-14.97	283	1547	81.71	3189	39719	91.97
4096	364	2422	84.97	11408	13686	16.64	410	3095	86.75	4574	79218	94.23
8192	452	3559	87.30	14159	19976	29.12	510	4562	88.82	5765	122867	95.31
16384	473	5753	91.78	14803	31458	52.94	533	7470	92.86	6082	200405	96.97
32768	509	7461	93.18	15914	42502	62.56	573	9641	94.06	6519	260436	97.50
5000	442	2979	85.16	13860	16840	17.70	499	3817	86.93	5549	96850	94.27
10000	467	4323	89.20	14633	24281	39.73	527	5554	90.51	5962	149391	96.01
15000	479	5300	90.96	14991	29034	48.37	540	6863	92.13	6136	184351	96.67
20000	485	6722	92.78	15173	37007	59.00	547	8744	93.74	6234	235027	97.35
25000	489	6809	92.82	15284	37777	59.54	551	8831	93.76	6296	240317	97.38
30000	492	6813	92.78	15367	38669	60.26	554	8785	93.69	6343	241883	97.38
35000	494	6788	92.72	15424	38514	59.95	556	8756	93.65	6379	237801	97.32
40000	495	7610	93.50	15468	43435	64.39	558	9834	94.33	6408	264500	97.58
45000	497	6071	91.81	15515	30957	49.88	560	8018	93.02	6453	209459	96.92
50000	513	5720	91.03	16041	28793	44.29	578	7558	92.35	6588	203568	96.76

XV. Measurements CPU-Reference Implementation

TABLE XXVII
Measurements CPU-Reference Implementation KEMs

Algorithm	kgen/s	encap/s	decap/s	total/s
Test system 1				
Kyber 512	4626.01	4422.00	4411.00	1509.67
Kyber 768	3103.66	2876.33	2845.33	1024.00
Kyber 1024	2218.33	2123.67	2102.00	728.00
FrodoKEM 640	58.78	73.85	78.07	23.83
FrodoKEM 976	26.25	35.32	34.84	10.08
FrodoKEM 1344	14.29	19.50	19.70	5.54
Test system 2				
Kyber 512	4958.99	5313.67	6411.98	1781.33
Kyber 768	3717.33	3743.33	4179.33	1253.67
Kyber 1024	2754.00	2687.33	2873.33	897.00
FrodoKEM 640	78.33	100.70	100.87	30.22
FrodoKEM 976	34.70	45.36	45.74	13.57
FrodoKEM 1344	18.91	25.20	25.18	7.44

TABLE XXVIII
Measurements CPU-Reference Implementation DSAs

Algorithm	kgen/s	sign/s	verify/s	total/s
Test system 1				
Dilithium 2	2458.33	726.42	2516.67	440.52
Dilithium 3	1422.00	424.67	1522.33	267.49
Diltihium 5	857.00	338.67	906.67	198.40
SPHINCS$^+$ 128f	86.64	3.92	66.16	3.38
SPHINCS$^+$ 128s	1.42	0.18	204.53	0.16
SPHINCS$^+$ 192f	60.74	2.39	44.72	2.16
SPHINCS$^+$ 192s	0.95	0.11	121.33	0.10
SPHINCS$^+$ 256f	23.24	1.15	44.24	1.06
SPHINCS$^+$ 256s	1.46	0.12	89.21	0.11
Test system 2				
Dilithium 2	2557.00	804.33	2954.33	524.83
Dilithium 3	1659.33	507.67	1813.67	319.56
Diltihium 5	1086.67	411.86	1111.00	236.86
SPHINCS$^+$ 128f	116.10	5.01	84.08	4.52
SPHINCS$^+$ 128s	1.82	0.24	234.18	0.21
SPHINCS$^+$ 192f	78.54	3.04	56.81	2.80
SPHINCS$^+$ 192s	1.22	0.14	162.17	0.12
SPHINCS$^+$ 256f	29.43	1.47	53.87	1.37
SPHINCS$^+$ 256s	1.85	0.16	113.11	0.14