

# Implementation and Benchmarking of Round 2 Candidates in the NIST Post-Quantum Cryptography Standardization Process Using Hardware and Software/Hardware Co-design Approaches

Viet Ba Dang<sup>1</sup>, Farnoud Farahmand<sup>1</sup>, Michal Andrzejczak<sup>2</sup>,  
Kamyar Mohajerani<sup>1</sup>, Duc Tri Nguyen<sup>1</sup> and Kris Gaj<sup>1</sup>

<sup>1</sup> Cryptographic Engineering Research Group,  
George Mason University  
Fairfax, VA, U.S.A.

<sup>2</sup> Military University of Technology  
Warsaw, Poland

**Abstract.** Performance in hardware has typically played a major role in differentiating among leading candidates in cryptographic standardization efforts. Winners of two past NIST cryptographic contests (Rijndael in case of AES and Keccak in case of SHA-3) were ranked consistently among the two fastest candidates when implemented using FPGAs and ASICs. Hardware implementations of cryptographic operations may quite easily outperform software implementations for at least a subset of major performance metrics, such as speed, power consumption, and energy usage, as well as in terms of security against physical attacks, including side-channel analysis. Using hardware also permits much higher flexibility in trading one subset of these properties for another. A large number of candidates at the early stages of the standardization process makes the accurate and fair comparison very challenging. Nevertheless, in all major past cryptographic standardization efforts, future winners were identified quite early in the evaluation process and held their lead until the standard was selected. Additionally, identifying some candidates as either inherently slow or costly in hardware helped to eliminate a subset of candidates, saving countless hours of cryptanalysis. Finally, early implementations provided a baseline for future design space explorations, paving a way to more comprehensive and fairer benchmarking at the later stages of a given cryptographic competition. In this paper, we first summarize, compare, and analyze results reported by other groups until mid-May 2020, i.e., until the end of Round 2 of the NIST PQC process. We then outline our own methodology for implementing and benchmarking PQC candidates using both hardware and software/hardware co-design approaches. We apply our hardware approach to 6 lattice-based CCA-secure Key Encapsulation Mechanisms (KEMs), representing 4 NIST PQC submissions. We then apply a software-hardware co-design approach to 12 lattice-based CCA-secure KEMs, representing 8 Round 2 submissions. We hope that, combined with results reported by other groups, our study will provide NIST with helpful information regarding the relative performance of a significant subset of Round 2 PQC candidates, assuming that at least their major operations, and possibly the entire algorithms, are off-loaded to hardware.

**Keywords:** Post-Quantum Cryptography · hardware · software/hardware co-design · FPGA · System on Chip · ASIC · Key Encapsulation Mechanism · digital signature · public-key · ARM · NEON

## 1 Introduction

Hardware benchmarking has played a major role in all recent cryptographic standardization efforts, such as the AES, eSTREAM, SHA-3 [11, 31, 44, 45], and CAESAR contests [16, 17]. With the emergence of commonly-accepted hardware application programming interfaces (APIs) [37], development packages [33, 36], specialized optimization tools [30, 22], new design methodologies based on High-Level Synthesis (HLS) [34, 35], and mandatory hardware implementations in the final round of the CAESAR contest [16], the percentage of initial submissions implemented in hardware grew from 27.5% in the SHA-3 contest [29] to 49.1% in the CAESAR competition [17, 28]. In Round 2, all AES, all SHA-3, and all but one CAESAR candidates had at least one hardware implementation reported by the end of the evaluation process.

In almost all cases, candidates performing particularly well in hardware were identified quite early during the evaluation process. For example, Keccak led in terms of speed in hardware already in Round 2 of the SHA-3 contest. It outperformed 13 remaining Round 2 candidates and the old standard SHA-2. AEGIS-128 was identified as one of the three fastest authenticated ciphers in Round 2 of the CAESAR contest when implemented using high-performance FPGAs, Virtex-6, Virtex-7, Stratix IV, and Stratix V. It outperformed at least 25 other candidates and the current standard AES-GCM. At the same time, during each contest, several candidates were identified as particularly costly, slow, or cumbersome to implement in hardware. Examples included Mars during the AES contest; BMW, ECHO, and SIMD during the SHA-3 competition; HS1-SIV, POET, and OMD in the CAESAR contest. The early identification of hardware inefficiency helped to focus the effort of the cryptographic community on more promising candidates, potentially saving countless hours of cryptanalysis.

**Hardware vs. software.** Cryptographic algorithms are routinely implemented using both software and hardware. By software, we mean implementations that can be executed using processors. These processors may vary from low-cost low-power embedded processors, such as ARM Cortex-M4, to high-performance general-purpose microprocessors, such as Intel Core i7, with Haswell microarchitecture, supporting Advanced Vector Extensions 2 (AVX2) and the AES New Instructions (AES-NI). The common feature is that all of these processors are typically programmed using high-level programming languages, such as C. Code written in these languages is portable among different processor types. Software implementations can be further optimized by using assembly language programming, involving instructions specific to a given processor (or more accurately to its Instruction Set Architecture (ISA)). Assembly language programs are not easily portable among processors based on different ISAs.

By hardware, we mean implementations that can be executed using Field Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), Programmable Logic (PL) of System on Chip FPGAs (SoC FPGAs), Application-Specific Standard Products (ASSPs), etc. The common feature is that most of these implementations are developed using hardware description languages (HDLs), such as VHDL and Verilog. These languages differ substantially from high-level programming languages by introducing the concepts of an entity, connectivity, concurrency, and timing. HDL source code is transformed by a synthesis tool to a netlist composed of basic logic components and connections among these components. Because of its generic nature, HDL code can be easily ported among different technologies, such as FPGAs and ASICs. ASIC implementations are faster, use less power, and require less physical area. FPGA implementations have the advantage of less expensive development tools, much shorter design cycle, and reconfigurability, understood as an ability to change the function of all internal building blocks and connections among them, even after a given integrated circuit has been deployed in actual products.

Low cost and short development cycle are decisive factors making FPGAs more suitable

97 for benchmarking and ranking candidates during the evaluation period. Reconfigurability  
98 supports the algorithm and parameter agility, making FPGAs more frequently used than  
99 ASICs during the early stages of deployment of PQC in real products. The relative  
100 performance of cryptographic algorithms in FPGAs has been shown correlated to their  
101 relative performance in ASICs [31]. At the same time, this correlation is not guaranteed  
102 to hold across multiple classes of cryptographic transformations, e.g., it is not guaranteed  
103 to work equally well for hash functions and PQC algorithms. Therefore, both FPGA and  
104 ASIC benchmarking studies are essential.

105 Although software implementations are likely to be dominant during the first phase of  
106 deploying PQC standards in real applications, hardware implementations will inevitably  
107 follow. They are likely to start from hardware accelerators for constrained environments,  
108 such as smart cards and Internet of Things devices. Low-cost low-power processors  
109 used in such applications may not be able to keep up with the increased demands for  
110 computational power and energy usage. Thus, these processors may need to be extended  
111 with hardware accelerators. In the medium term, high-performance security processors  
112 enhanced with new PQC standards will emerge. These processors will be optimized to  
113 process in hardware all the algorithms associated with secure communication (such as  
114 those used in the post-quantum versions of TLS, IPsec, IKE, and WTLS/WAP protocols)  
115 and secure storage. Finally, in the longer-term, support for new instructions, enabling  
116 the efficient and side-channel resistant implementations of PQC standards, is likely to  
117 be added to the most popular processor ISAs. Co-processors for such instructions are,  
118 effectively, hardware implementations of PQC. Taking into account that the new PQC  
119 standards are likely to remain in use for decades, all of the mentioned above use cases  
120 should be given considerable weight. In particular, the performance of a given algorithm  
121 in hardware may affect its long-term performance in software, on processors equipped  
122 with new specialized instructions. Even if Round 2 hardware implementations are not a  
123 final word in terms of the algorithm performance, they provide the first glimpse into each  
124 candidate's suitability for hardware acceleration. They also establish an open source-code  
125 base on which more optimized implementation and implementations protected against  
126 side-channel and fault attacks can be built in Round 3 and beyond.

127 **High-speed vs. lightweight.** Assuming the use of the same technology, hardware  
128 implementations outperform software implementations using at least one, and typically  
129 multiple metrics, such as speed, power consumption, energy usage, and security against  
130 physical attacks. They also allow much higher flexibility in trading one subset of these  
131 metrics for another. From the point of view of benchmarking and ranking of candidates,  
132 such flexibility may become a curse, especially taking into account that no two metrics  
133 are likely to have a simple linear dependence on each other. A practical solution to this  
134 problem is to focus during the evaluation process on two major types of implementations:  
135 high-speed and lightweight.

136 In high-speed implementations, the primary target is speed. For PQC schemes, this target  
137 amounts to minimizing the execution times of major operations involving the public and  
138 private key, respectively. For Key Encapsulation Mechanisms (KEMs), these operations are  
139 encapsulation and decapsulation; for digital signature schemes, signature verification and  
140 generation; for public-key encryption (PKE), encryption and decryption. The time of key  
141 generation may also play a major role in the case when a public-private key pair cannot be  
142 reused for security reasons. The resource utilization is secondary. Still, hardware designers  
143 typically aim at achieving the Pareto optimality, in which any further improvement in  
144 speed comes at the disproportionate cost in terms of resource utilization. The primary  
145 advantage of high-speed implementations is that they reveal the inherent potential of a  
146 given algorithm for parallelization. As long as the resource-utilization limit is sufficiently  
147 high, this limit does not affect the ranking of algorithms. As a result, the ranking is  
148 strongly correlated with the features of algorithms themselves and is not substantially

149 influenced by any additional assumptions and technology choices. Additionally, only  
150 high-speed hardware implementations may effectively compete with optimized software  
151 implementations targeting high-performance processors with vector instructions (e.g.,  
152 AVX2).

153 In lightweight implementations, the primary targets are typically minimum resource  
154 utilization and minimum power consumption, under the assumption that the execution  
155 time does not exceed a predefined maximum. Another way of formulating the goal is  
156 to achieve minimum execution time, assuming a given maximum budget in terms of  
157 resource utilization, power consumption, or energy usage. The maximum budget on  
158 resource utilization is related to the cost of implementation; the budget on power assures  
159 correct operation without overheating or devoting additional resources to cooling. The  
160 maximum energy usage affects how long a battery-operated device can function before  
161 the next battery recharge. In the context of the standardization process for cryptographic  
162 algorithms, the mentioned above maximum budgets are very hard to select. Any change in  
163 these thresholds may favor a different subset of candidates. With new standards remaining  
164 in use for decades, timing, cost, and power requirements of new and emerging applications  
165 are very challenging to predict.

166 Additionally, changes in technology significantly affect which hardware architectures  
167 meet particular constraints. For example, an architecture capable of accomplishing the  
168 execution time of 0.1 seconds (or below), under a certain power or energy budget, may  
169 substantially change with the improvements in technology. As a result, the majority of  
170 current limits are selected somewhat arbitrarily by different designers, or left undefined in  
171 their reports. Consequently, the ranking of PQC candidates based on their lightweight  
172 implementations, especially those developed by different groups, is extremely challenging  
173 and assumption-dependent. These rankings have little to do with the parallelization  
174 allowed by each algorithm, as most of the operations must be executed sequentially due  
175 to the small resource budget. The primary feature of algorithms these implementations  
176 reveal is the number and complexity of its distinct elementary operations. Each major  
177 operation infers an additional functional unit, increasing resource utilization and power  
178 consumption. Additionally, lightweight hardware implementations can outperform only  
179 software implementations targeting specific low-cost low-power embedded processors, such  
180 as Cortex-M4.

181 In the case of FPGA implementations, resource utilization is a vector, such as (#LUTs,  
182 #flip-flops, #DSP units, #BRAMs). No single element of this vector can be expressed  
183 in terms of other elements. As a result, imposing a resource limit implies specifying the  
184 values of all components of this resource vector. One possible approach may be to choose  
185 the resources of the smallest FPGA of a given low-cost FPGA family. However, FPGA  
186 families and their resources change over time, so this limit has only a physical meaning  
187 during the limited time, covering the evaluation period, and may lose its significance just  
188 a few years after the standard is published and deployed. Finally, the same FPGA device  
189 may also need to accommodate any overhead associated with countermeasures against  
190 side-channel attacks. At the same time, this overhead or even effective countermeasures  
191 may remain unknown at the time of the candidates' evaluation.

192 As a result, in this paper, we focus on the development, benchmarking, and ranking of  
193 high-speed implementations. At the same time, we do our best to summarize lessons  
194 learned from the development of lightweight implementations by other groups.

195 **Speeding-up the development process.** Traditionally software and hardware bench-  
196 marking were conducted separately by different groups of experts, equipped with different  
197 knowledge and tools. Even the units for expressing speed were different – cycles per byte  
198 for software and megabits per second for hardware. For PQC algorithms, this approach is  
199 hard to maintain. These algorithms are simply too complex and too different from the  
200 current state-of-the-art in public-key cryptography to permit the development of optimized

201 pure-hardware implementations for a significant fraction of Round 2 candidates by any  
202 single group within the time frame imposed by the NIST evaluation process.

203 Two approaches to overcome the long development time have emerged. The first is  
204 software/hardware co-design [21, 77]; the second is the use of HLS [14, 21, 57].

205 Software/hardware co-design has been used for years in the industry and studied extensively  
206 in academia, with the goal of reaching performance targets using a shorter development  
207 cycle than that typical for hardware-only implementations. To the best of our knowledge,  
208 no benchmarking of software/hardware co-designs was reported during any previous  
209 cryptographic competitions. As a result, multiple problems specific to cryptographic  
210 contests, such as the choice of the most representative platform(s) and the fairness of  
211 software/hardware partitioning schemes, have never been addressed. It should be clearly  
212 stated that software/hardware benchmarking is not intended as a replacement for pure-  
213 hardware benchmarking. On the contrary, applying this approach to selected Round  
214 2 candidates and developing a library of hardware accelerators for major operations of  
215 these candidates will make it much easier to develop hardware-only implementations in  
216 subsequent rounds. Although the software/hardware co-design approach can be used to  
217 realize both high-speed and lightweight implementations, in this paper, we focus on its  
218 application to high-speed designs.

219 Within the proposed framework, one of the first issues to address is the choice of the appropriate  
220 platform. In particular, we need a computing platform allowing fast communication  
221 across the software/hardware boundary. We also need the suitable prototyping board, as  
222 the timing measurements had to be performed experimentally, and the computing platform  
223 had to be well-suited for attempting various software/hardware partitioning schemes. The  
224 choice of a suitable device and prototyping board is addressed in Section 4. With the  
225 preferred platform identified, our second major concern is the fairness of software/hardware  
226 benchmarking, especially in terms of deciding which operations within each evaluated  
227 scheme should be offloaded to hardware. In this paper, we propose a comprehensive  
228 approach to address this issue, aimed at achieving the best possible trade-off between the  
229 final performance and the required development time. This approach is described in detail  
230 in Section 4.

231 The second approach to substantially accelerating the development time is the use of  
232 High-Level Synthesis. This approach amounts to refactoring a software implementation in  
233 C or C++ in such a way that this implementation can be used as an input to a High-Level  
234 Synthesis tool, such as Vivado HLS or LegUp, capable of automatically transforming such  
235 an implementation to HDL code. The result is a pure hardware implementation obtained  
236 based on the code written in a traditional programming language (typically C and C++).  
237 This language is turned into a high-level hardware description language using synthesis  
238 directives encoded using pragmas and specific coding techniques aimed at exposing the  
239 potential for parallelization and resource utilization reduction.

240 This approach has been demonstrated to substantially reduce the development time of  
241 Round 2 and Round 3 CAESAR candidates. At the same time, it provided an almost  
242 identical ranking of candidates in terms of throughput and throughput to area ratio.  
243 However, taking into account significant differences between the complexity and underlying  
244 operations of secret-key authenticated ciphers and public-key PQC schemes, the use of HLS  
245 for benchmarking of PQC candidates remains controversial. The common perception is  
246 that obtained results are significantly worse in terms of both speed and resource utilization  
247 compared to manual HDL coding. However, our preliminary research indicates that, with  
248 a proper approach, the penalty in terms of the execution time in clock cycles can be made  
249 negligibly small. Only the penalty in terms of resource utilization and clock frequency  
250 remains. The former overhead affects only the secondary metric in high-speed designs;  
251 the latter can be kept in a similar range for multiple candidates. As a result, the use of  
252 high-level synthesis when applied to high-speed designs should remain an active area of

research, and should not be dismissed upfront before more case studies are performed.

**Choice of FPGA family.** One of the major concerns is the NIST recommendation to focus on hardware benchmarking using the Xilinx Artix-7 FPGA family. This recommendation appeared in several NIST presentations related to Round 2 of the NIST standardization process, e.g., during PQCrypto 2019 in May 2019 and the Second PQC Standardization Conference in August 2019. We believe that, in its current form, this recommendation is counterproductive, and it impedes rather than supports fair and comprehensive hardware and software/hardware benchmarking.

Let us start by explaining what an FPGA family is and what influence does it have on an evaluation process. FPGA family is a set of FPGA devices sharing the same internal structure and the same process technology (also known as technology node or process node), described by a number related to the size and density of transistors that can be fabricated using a given manufacturing process. With the steady improvements in process technology, described by Moore's Law, the maximum capacity and speed of FPGA devices have been steadily increasing while their prices have remained approximately the same. Every new generation of FPGA devices of a particular vendor receives a unique name, referred to as a family name. Every family consists of multiple devices with various distinct sizes to match the needs of different applications. All devices of a particular family share the same internal architecture and process technology but differ in terms of the number of resources of a particular type, such as Look-Up Tables (LUTs), flip-flops (FFs), block memories, and digital signal processing units (DSP units) or multipliers. Most vendors release both low-cost families (such as Xilinx Artix-7) and high-performance families (such as Xilinx Virtex-7). Most of them also release mid-range families, such as Xilinx Kintex-7. The maximum amount of resources available in the largest device of a low-cost family is naturally significantly smaller than the equivalent amount in the largest device of a high-performance family (e.g., over 5 times smaller for Artix-7 vs. Virtex-7).

Additionally, in recent years, FPGA vendors started releasing new types of programmable devices that enhance Programmable Logic of traditional FPGAs with the Processing System based on a hardwired embedded processor, such as ARM. Since this processor is custom designed, it takes full advantage of a given technological process and operates at a clock frequency significantly higher than Programmable Logic. With a fast processor and an efficient interface between this processor and Programmable Logic, these devices are ideal for software/hardware co-designs targeting high-speed. Although these types of devices appear under multiple commercial names, they are often collectively referred to as System on Chip FPGAs (SoC FPGAs). The first family of this type was Xilinx Zynq-7000, released in 2011, based on ARM Cortex-A9 embedded processors.

Hardware designs are described in hardware description languages. HDL code is typically identical for all FPGA families. As opposed to software, where each processor may require different optimized assembly language code, no such concepts exist for hardware. As a result, it is straightforward to synthesize the same HDL code targeting various FPGA families from various vendors, as long as the maximum capacity of the largest device of a given family is not exceeded.

Giving preference to the Xilinx Artix-7 family has several undesired consequences summarized below:

1. Artix-7 is a low-cost FPGA family. As such, it is not very suitable for high-speed implementations. Hardware resources of even the largest device of this family are often insufficient to demonstrate the full potential for parallelizing operations a given PQC algorithm. Thus, the use of Artix-7 makes perfect sense for benchmarking lightweight implementations but may lead to suboptimal results for high-speed implementations.
2. Artix-7 is a traditional FPGA, and not an SoC FPGA. As a result, the only way to develop a single-chip software/hardware implementation using Artix-7 is the use

305 of so-called "soft" processor cores, i.e., processors implemented using programmable  
306 logic. Soft processors compatible with Artix-7 include MicroBlaze and lightweight  
307 versions of RISC-V. All of them operate at much lower clock frequency than hardwired  
308 embedded processors of SoC FPGAs.

- 309 3. Artix-7 is unsuitable for HLS designs. Such designs typically take significantly more  
310 resources than designs based on writing code manually in HDL. As a result, assuming  
311 the Pareto optimization for high-speed, they are unlikely to fit in the largest Artix-7  
312 FPGA.
- 313 4. Artix-7 is a relatively old FPGA family, released by Xilinx in 2010. By the time of  
314 the release of the PQC standard, this family will be at least 12 years old. While  
315 still relatively popular for low-cost applications, this family does not represent the  
316 state-of-the-art in FPGA technology.
- 317 5. It is not customary to base ranking of candidates in cryptographic contests on results  
318 obtained for a single family of a single vendor. Although Xilinx is the largest developer  
319 of FPGAs and SoC FPGAs, Intel comes a strong second, and other vendors, such as  
320 Microchip and Lattice Semiconductor, also develop FPGAs suitable for implementing  
321 cryptographic algorithms. During the SHA-3 competition, the results were reported  
322 for seven FPGA families from two major vendors, Xilinx and Altera. During the  
323 CAESAR contest, four Xilinx families and four Altera families were employed. For  
324 all of these families, results were generated based on the same HDL code. There was  
325 no need to purchase multiple tools or boards. Free or trial versions of tools were  
326 sufficient. The designs ended with the generation of post-place-and-route reports,  
327 which correctly described the worst-case performance of any particular instance of  
328 the given FPGA device.
- 329 6. Based on the authors' experiences, multiple reviewers of papers devoted to imple-  
330 mentations of Round 2 PQC candidates treated the NIST's choice of Artix-7 as an  
331 absolute requirement. Submissions not complying with this requirement were subject  
332 to rejection or requests for major revisions. As a result, a noble goal of making the  
333 results more comparable with one another was turned into a reason for suppressing  
334 or delaying the publication of relevant results.

335 Taking these concerns into account, our recommendation for Round 3 is to encourage  
336 reporting results for at least the following FPGA families:

- 337 1. For lightweight hardware implementations and lightweight software/hardware im-  
338 plementations based on soft processor cores: Xilinx Artix-7 (for compatibility with  
339 Round 2 results) and Intel Cyclone 10 LP.
- 340 2. For lightweight software/hardware implementations based on the use of hard processor  
341 cores: Xilinx Zynq 7000-series and Intel Cyclone V SoC FPGAs.
- 342 3. For high-speed hardware and high-speed software/hardware implementations: Zynq  
343 Xilinx UltraScale+ and Intel Stratix 10 SoC.

344 One of the reasons for selecting Zynq Xilinx UltraScale+, even for pure hardware im-  
345 plementations that do not require SoC capabilities, is the support for these devices by  
346 the free version of the Xilinx toolset, called Vivado HL WebPACK, which is sufficient to  
347 generate all required benchmarking results. Xilinx Virtex-7 UltraScale+ FPGAs, which  
348 could be considered as a natural candidate, are not supported by the same free version  
349 of tools. The Zynq Xilinx UltraScale+ family is also recommended for high-speed soft-  
350 ware/hardware implementations based on the use of hard processor cores because of  
351 moderate cost of suitable prototyping boards and the availability of a free Benchmarking

352 Setup for Software/Hardware Implementations of PQC Schemes, developed at George  
 353 Mason University [20].

## 354 2 Previous Work

**Table 1:** Reported Hardware Implementations

Algorithms	High-Speed	Lightweight
<b>Lattice-based : Encryption/Key Exchange</b>		
CRYSTALS-KYBER	[80], [14] <sup>H</sup> , CERG	[12], [13]*, [1], [26]
FrodoKEM	[38], [14] <sup>H</sup> , [18]	[12], [13]*
LAC	[80], CERG	–
NewHope	[14] <sup>H</sup> , [27], [81], [80], [41], CERG	[12], [13]*, [1], [26]
NTRU	[14] <sup>H</sup> , CERG	–
NTRU Prime	CERG	–
Round5	[18], [4], CERG	[3]
SABER	[14] <sup>H</sup> , [18], [54], [63]	[26]
Three Bears	–	–
<b>Isogeny-based : Encryption/Key Exchange</b>		
SIKE	[49], [53], [19]	[53]
<b>Code-based : Encryption/Key Exchange</b>		
BIKE	[40], [60], [61]	–
Classic McEliece	[75], [14] <sup>H</sup>	–
HQC	–	–
LEDAcrypt	[14] <sup>H</sup>	[39]
NTS-KEM	–	–
ROLLO	–	–
RQC	–	–
<b>Lattice-based : Digital Signature</b>		
CRYSTALS-DILITHIUM	[14] <sup>H</sup>	[12], [13]*
FALCON	–	–
qTESLA	[14] <sup>H</sup>	[12], [13]*, [76]
<b>Symmetric-based : Digital Signature</b>		
Picnic	[42]	–
SPHINCS+	[14] <sup>H</sup>	–
<b>Multivariate : Digital Signature</b>		
GeMSS	–	–
LUOV	–	–
MQDSS	[14] <sup>H</sup>	–
Rainbow	[24]	–

<sup>H</sup> design developed using the High-Level Synthesis (HLS) approach  
 \* extended version of [12]

355 Hardware and software/hardware implementations of Round 2 PQC candidates reported  
 356 to date are summarized in Table 1. The PQC candidates are grouped by family and a type  
 357 of scheme. All Encryption and Key Exchange schemes are listed first, followed by Digital  
 358 Signature schemes. The Encryption and Key Exchange schemes have candidates from  
 359 three major families: lattice-based, isogeny-based, and code-based. The Digital Signature  
 360 schemes have candidates representing lattice-based, symmetric-based, and multivariate  
 361 families. All implementations are classified as either High-Speed or Lightweight. However,  
 362 the dividing line is not always very clear, and, in multiple cases, the authors have not used  
 363 these terms explicitly by themselves.



364 HLS-based implementations are distinguished with the superscript <sup>H</sup>. These implementa-  
 365 tions were reported in only one paper [14]. They have been shown to give substantially  
 366 different results than implementations developed using traditional Register-Transfer Level  
 367 (RTL) methodology, in which HDL code is developed manually. Therefore, in this section,  
 368 we focus on implementations in which a hardware part of the design was developed using  
 369 traditional RTL methodology. The HLS designs are discussed separately in Section 5.4.  
 370 Eight out of 26 candidates (31%) do not have any high-speed implementation to date, and  
 371 13 (50%) do not have any RTL high-speed implementation. 17 out of 26 (65%) do not have  
 372 any lightweight implementation. The coverage of the code-based family is the weakest, with  
 373 only 3 out of 7 candidates (BIKE, Classic McEliece, and LEDAcrypt) implemented targeting  
 374 high-speed (including only two using RTL-based approach), and only 1 out of 6 (LEDAcrypt)  
 375 realized using a lightweight approach. Similarly, the multivariate family remains mostly  
 376 unexplored. Only two out of four candidates have their implementations reported, including  
 377 only one following the RTL methodology. The symmetric-based digital signatures have no  
 378 lightweight implementations, and even among high-speed implementations, only one is the  
 379 RTL-based implementation.

380 The coverage of the lattice-based and isogeny-based encryption/key exchange schemes is  
 381 the most complete. Eight out of nine lattice-based KEMs have high-speed implementations  
 382 reported. The only exception is Three Bears. Five out of these eight have, on top of that, at  
 383 least one lightweight implementation. In terms of the number of various implementations,  
 384 NewHope leads the way with 10 related publications, followed by CRYSTALS-KYBER,  
 385 with 7, and FrodoKEM and Saber with 5. The only isogeny-based scheme, SIKE, has  
 386 been thoroughly explored in hardware as well, especially taking into account the earlier  
 387 implementations of the underlying key agreement scheme SIDH [47], [9], [48], [46], [50].  
 388 The coverage of lattice-based signatures is not as good as lattice-based KEMs. In par-  
 389 ticular, FALCON appears to be very difficult to implement, using either the high-speed  
 390 or lightweight approach. Additionally, even in the case of CRYSTALS-DILITHIUM,  
 391 somewhat surprisingly, its only high-speed implementation to date is an HLS-based design.  
 392 In Tables 2–8, we summarize major results for hardware and software/hardware imple-  
 393 mentations of KEMs. Most of the schemes are KEMs with indistinguishability under the  
 394 chosen-ciphertext attack (IND-CCA). Some are PKEs with indistinguishability under the  
 395 chosen-plaintext attack (IND-CPA). If an IND-CPA-secure PKE is reported, this fact is  
 396 marked with a superscript <sup>cpa</sup>. All mentioned above tables have the same fields. The  
 397 first two columns contain a reference to the publication and the name of the algorithm  
 398 variant, respectively. The superscript <sup>z</sup> next to the publication reference indicates the  
 399 implementation using Zynq-7000 SoC FPGA. The implementations targeting Artix-7 and  
 400 Zynq-7000 are grouped together because the programmable logic of both families is realized  
 401 using the same technological process and composed of the same basic building blocks.  
 402 In the third column, the type of implementation is indicated, with HW standing for  
 403 hardware, and SW/HW standing for software/hardware. Among the software/hardware  
 404 implementations, we specify the embedded processors used with the following notation:  
 405 <sup>RV</sup> represents a RISC-V processor with the RV32IM ISA, i.e., RISC-V with the base 32-bit  
 406 integer ISA and the standard Integer Multiplication and Division extension. <sup>c</sup> represents a  
 407 custom processor, and <sup>A9</sup> a hard processor of the Zynq-7000 SoC FPGA family, namely  
 408 ARM Cortex-A9. Unlike the first two options, this processor operates with the frequency  
 409 significantly higher than the maximum clock frequency of programmable logic. At the same  
 410 time, the transfer of control and data between the processor and the hardware accelerator  
 411 contributes a non-negligible transfer overhead to all reported execution times.

**Table 2:** Level I KEMs and PKEs on Artix-7 (default) and Zynq-7000 (indicated with the superscript <sup>z</sup>)

Design	Algorithm	Type	Target	Max. Freq.	LUT	FF	Slice	DSP	BR AM	Key Generation		Encaps./Enc. <sup>cpa</sup>		Decaps./ (Dec.+Enc.) <sup>cpa</sup>	
										cycles	$\mu s$	cycles	$\mu s$	cycles	$\mu s$
[81] <sup>z</sup>	NewHope-512 <sup>cpa</sup>	HW	HS	200	6,780	4,026	-	2	7.0	4,200	21.0	6,600	33.0	9,100	45.5
[75]	mciece348864 <sup>cpa</sup>	HW	HS	106	81,339	132,190	-	0	236.0	202,787	1,920.3	2,720	25.8	12,743	120.7
[75]	mciece348864 <sup>cpa</sup>	HW	LW	108	25,327	49,383	-	0	168.0	1,599,882	14,800.0	2,720	25.2	18,358	169.8
[26] <sup>z</sup>	Kyber-512	SW/HW <sup>RV</sup>	LW	-	23,925	10,844	-	21	32.0	150,106	-	193,076	-	204,843	-
[38]	FrodoKEM-640 16x	HW	HS	172	2,587	2,994	855	16	0	204,766	1,190.5	-	-	-	-
[13]	Kyber-512	SW/HW <sup>RV</sup>	LW	171	5,796	4,694	1,692	16	0	-	-	207,269	1,212.1	-	209,867
[26] <sup>z</sup>	NewHope-512	SW/HW <sup>RV</sup>	LW	25*	14,975	2,539	4,173	11	14.0	54,861	2,194.4	-	-	209,867	1,408.5
[49]	SIKEp434	HW	HS	132	21,946	24,328	8,006	240	26.5	123,860	-	207,299	-	226,742	5,842.7
[49]	SIKEp503	HW	HS	130	24,610	27,759	9,186	264	33.5	530,000	4,009.1	930,000	7,034.8	980,000	7,413.0
[13]	NewHope-512	SW/HW <sup>RV</sup>	LW	25*	14,975	2,539	4,173	11	14.0	640,000	4,926.9	1,140,000	8,776.0	1,200,000	9,237.9
[26] <sup>z</sup>	LightSaber	SW/HW <sup>RV</sup>	LW	-	23,925	10,844	-	21	32.0	97,969	3,918.8	236,812	9,472.5	258,872	10,354.9
[1]	Kyber-512	SW/HW <sup>RV</sup>	LW	59	1,842	1,634	-	5	34.0	710,000	11,993.2	971,000	16,402.0	870,000	14,695.9
[1]	NewHope-512	SW/HW <sup>RV</sup>	LW	59	1,842	1,634	-	5	34.0	904,000	15,270.3	1,424,000	24,054.1	1,302,000	21,993.2
[53]	SIKEp434	SW/HW <sup>c</sup>	HS	162	22,595	11,558	7,491	162	37.0	1,474,200	9100	2,494,800	15,400.0	2,656,800	16,400.0
[53]	SIKEp503	SW/HW <sup>c</sup>	HS	162	22,595	11,558	7,491	162	37.0	1,733,400	10,700.0	2,932,200	18,100.0	3,126,600	19,300.0
[38]	FrodoKEM-640 1x	HW	LW	191	971	433	290	1	0	3,237,288	16,949.2	-	-	-	-
[53]	SIKEp434	SW/HW <sup>c</sup>	LW	190	4,246	2,131	1,180	1	0	-	-	3,275,862	17,241.4	-	-
[53]	SIKEp503	SW/HW <sup>c</sup>	LW	162	4,446	2,152	1,254	1	12.5	-	-	-	-	3,306,122	20,408.2
[61]	BIKE Level 1	HW	LW	143	10,976	7,115	3,512	57	21.0	2,187,902	15,300.0	3,718,004	26,000.0	3,946,804	27,600.0
[61]	BIKE Level 1	HW	LW	121	10,702	4,940	3,334	7	15.0	2,602,603	18,200.0	4,390,104	30,700.0	4,676,105	32,700.0
[13]	FrodoKEM-640	SW/HW <sup>RV</sup>	LW	96	29,448	5,498	8,419	7	28.0	2,671,000	21,903.0	153,000	1,252.0	13,120,000	107,580.0
[40]	BIKE-1 Level 1 <sup>cs</sup>	HW	HS	165	1,907	1,049	608	0	7.0	259,000	2,691.0	12,000	127.0	13,120,000	136,443.0
[40]	BIKE-3 Level 1 <sup>cs</sup>	HW	HS	170	1,397	925	453	0	4.0	95,500	578.0	-	-	-	-
[40]	BIKE-2 Level 1 <sup>cs</sup>	HW	HS	160	3,874	2,141	1,312	0	10.0	2,150,000	13,437.0	-	-	-	-

<sup>z</sup> Design implemented on Zynq-7000<sup>cpa</sup> Design of a PKE variant resistant against Chosen-Plaintext Attack (CPA)<sup>cs</sup> Designs for the variants BIKE-1, BIKE-2, and BIKE-3 consolidated by submitters to BIKE on May 3, 2020<sup>RV</sup> co-design using RISC-V RV32IM<sup>c</sup> co-design using a custom processor<sup>A9</sup> co-design using ARM Cortex-A9

\* Preliminary result

**Table 3:** Level III & V KEMs and PKEs on Artix-7 (default) and Zynq-7000 (indicated with the superscript  $z$ )

Design	Algorithm	Type	Target	Max. Freq.	LUT	FF	Slice	DSP	BR AM	Key Generation		Encaps./Enc. $^{cpa}$		Decaps./Dec. + Enc. $^{cpa}$		
										cycles	$\mu s$	cycles	$\mu s$	cycles	$\mu s$	
<b>Security Level III</b>																
[75]	mceliece460896 $^{cpa}$	HW	LW	107	38,669	74,858	0	303.0	0	303.0	5,002,044	46,704.4	3,360	31.4	31,005	289.5
[38]	FrodoKEM-976 16x	HW	HS	169	2,869	3,000	908	16	0	16	476,05	2,816.9	—	—	—	—
[54] <sup>z</sup>	Saber	SW/HW <sup>A9</sup>	HS	157	7,213	5,087	2042	16	19.0	16	—	—	479,993	2,857.1	483,073	3,076.9
[13]	Kyber-768	SW/HW <sup>RV</sup>	LW	125	7,400	7,331	28	2.0	2.0	2.0	—	3,273.0	—	4,147.0	—	3,844.0
[49]	SIKEp610	HW	HS	125	14,975	2,539	4,173	11	14.0	11	84,110	3,364.4	184,080	7,363.2	198,011	7,920.4
[53]	SIKEp610	SW/HW <sup>c</sup>	HS	125	29,447	33,198	10,843	312	39.5	39.5	900,000	7,182.8	1,810,000	14,445.3	1,780,000	14,205.9
[38]	FrodoKEM-976 1x	HW	LW	162	22,595	11,558	7,491	162	37.0	37.0	2,916,000	18,000.0	5,443,200	33,600.0	5,508,000	34,000.0
[53]	SIKEp610	SW/HW <sup>c</sup>	LW	189	1,243	441	362	1	0	0	7,560,000	40,000.0	—	—	—	—
[61]	BIKE Level 3	HW	LW	187	4,650	2,118	1,272	1	19.0	1	—	—	7,480,000	40,000.0	—	—
[13]	FrodoKEM-976	SW/HW <sup>RV</sup>	LW	162	4,888	2,153	1,390	1	19.0	1	4,347,204	30,400.0	8,108,108	56,700.0	8,208,208	47,619.0
[53]	SIKEp610	SW/HW <sup>c</sup>	LW	143	10,976	7,115	3,512	57	21.0	57	11,600,207	95,122.6	601,099	4,929.1	37,596,111	57,400.0
[61]	BIKE Level 3	HW	HS	122	9,808	5,075	2,996	7	23.0	23.0	930,179	9,674.2	42,162	438.5	37,596,111	308,291.2
[13]	FrodoKEM-976	SW/HW <sup>RV</sup>	LW	96	28,784	5,553	8,184	7	33.0	33.0	26,005,326	1,040,213.0	29,749,417	1,189,976.7	30,421,175	391,015.2
[13]	FrodoKEM-976	SW/HW <sup>RV</sup>	LW	25*	14,975	2,539	4,173	11	14.0	14.0	8,000	40.0	12,500	62.5	17,300	1,216,847.0
<b>Security Level V</b>																
[81] <sup>z</sup>	NewHope-1024 $^{cpa}$	HW	HS	200	6,781	4,127	—	2	8.0	2	8,000	40.0	12,500	62.5	17,300	86.5
[41] <sup>z</sup>	NewHope-1024 $^{cpa}$	HW	HS	190	13,244	8,272	—	24	18.0	24	—	—	34,000	178.0	30,600	160.0
[13]	Kyber-1024	SW/HW <sup>RV</sup>	LW	25*	14,975	2,539	4,173	11	14.0	11	116,841	4,673.6	236,886	9,475.4	256,828	10,273.1
[13]	NewHope-1024	SW/HW <sup>RV</sup>	LW	25*	14,975	2,539	4,173	11	14.0	11	97,969	3,918.8	236,812	9,472.5	258,872	10,354.9
[26] <sup>z</sup>	Kyber-1024	SW/HW	LW	—	23,925	10,844	—	21	32.0	21	349,673	—	405,477	—	424,682	—
[26] <sup>z</sup>	NewHope-1024	SW/HW	LW	—	23,925	10,844	—	21	32.0	21	235,420	—	392,734	—	450,541	—
[49]	SIKEp751	HW	HS	127	40,792	49,982	15,794	512	43.5	43.5	1,250,000	9,842.5	2,210,000	17,401.6	2,340,000	18,425.2
[27] <sup>z</sup>	NewHope-1024 $^{cpa}$	SW/HW	HS	25	26,606	26,303	—	32	1.0	32	357,052	14,282.1	589,285	23,571.4	756,932	30,277.3
[26] <sup>z</sup>	FireSaber	SW/HW	LW	—	23,925	10,844	—	21	32.0	21	1,300,272	—	1,622,818	—	1,898,051	—
[1]	Kyber-1024	SW/HW <sup>RV</sup>	LW	59	1,842	1,634	—	5	34.0	5	2,203,000	37,212.8	2,619,000	44,239.9	2,429,000	41,030.4
[53]	SIKEp751	SW/HW <sup>c</sup>	HS	162	22,595	11,558	7,491	162	37.0	37.0	3,742,200	23,100.0	6,188,400	38,200.0	6,658,200	41,100.0
[1]	NewHope-1024	SW/HW <sup>RV</sup>	LW	59	1,842	1,634	—	5	34.0	5	1,776,000	30,000.0	2,742,000	46,317.6	2,528,000	42,702.7
[53]	SIKEp751	SW/HW <sup>c</sup>	LW	143	10,976	7,115	3,512	57	21.0	57	7,965,108	55,700.0	13,156,013	92,000.0	14,185,614	99,200.0
[13]	FrodoKEM-1344	SW/HW <sup>RV</sup>	LW	25*	14,975	2,539	4,173	11	14.0	14.0	67,994,170	2,719,766.8	71,501,358	2,860,054.3	72,526,695	2,901,067.8

<sup>z</sup> Design implemented on Zynq-7000<sup>cpa</sup> Design of a PKE variant resistant against Chosen-Plaintext Attack (CPA)<sup>RV</sup> co-design using RISC-V RV32IM<sup>c</sup> co-design using a custom processor<sup>A9</sup> co-design using ARM Cortex-A9

\* Preliminary result

<sup>KD</sup> total execution time of Key Generation and Decapsulation

[41] only reports latency of Encapsulation and total latency of Key Generation and Decapsulation

412 The next column, Max. Freq., corresponds to the maximum clock frequency in MHz. The  
413 next five columns are used to report FPGA resource utilization, described as a vector (LUT,  
414 FF, Slice, DSP, BRAM), where the subsequent fields represent the number of look-up  
415 tables, flip-flops, slices, DSP units, and 36 kbit Block RAMs. For the last of these values,  
416 BRAM, 0.5 represents the use of an 18-kbit block RAM.

417 In the case of KEMs, the remaining 6 columns are used to show the execution time of  
418 Key Generation, Encapsulation, and Decapsulation, expressed in clock cycles and  $\mu s$ ,  
419 respectively. In the cases when only results for the IND-CPA PKE are reported, the last  
420 two columns represent the sum of the execution times of Encryption and Decryption. This  
421 convention is used because the most popular transformations between an IND-CPA-secure  
422 PKE and the corresponding IND-CCA-secure KEM involve both the Decryption and  
423 Encryption operations on the receiver's side. Additionally, these two operations dominate  
424 the total Decapsulation time. For all execution times, the value in  $\mu s$  can be obtained by  
425 dividing the corresponding number of clock cycles by the maximum clock frequency in  
426 MHz.

427 In Tables 2 and 3, we summarize implementations targeting Xilinx Artix-7 FPGAs  
428 and related Xilinx Zynq-7000 SoC FPGAs. For the security level 1, six candidates -  
429 Classic McEliece, CRYSTALS-Kyber, FrodoKEM, NewHope, SIKE, and Saber - have  
430 implementations of all three operations reported. The preliminary implementations of  
431 BIKE focused on key generation only [40, 5]. For security level 3, NewHope does not have  
432 a variant. For security level 5, the results are missing for Classic McEliece.

433 For most KEMs, the time of decapsulation is longer than the time of encapsulation. Table  
434 entries are ordered according to the time of decapsulation in  $\mu s$  (and, if needed, according  
435 to the decapsulation time in clock cycles).

436 The ranking of candidates listed in Tables 2 and 3 is very challenging to determine based  
437 on available results. First, it may be unfair to compare pure hardware implementations  
438 with software/hardware implementations. Secondly, it is hard to compare lightweight  
439 implementations with high-speed implementations, as they are optimized with different  
440 primary metrics in mind. Third, software/hardware implementations based on different  
441 processors are very challenging to compare with one another. Finally, even for imple-  
442 mentations using exactly the same type of implementation (software/hardware) and the  
443 same type of processor (RISC-V), such as those reported in [26], the comparison may  
444 be unintentionally biased. In the specific case of [26], significantly different hardware  
445 support was provided for algorithms that can take advantage of the Number Theoretic  
446 Transform - Kyber and NewHope - vs. the algorithm that cannot - Saber. An additional,  
447 relatively minor factor is that several results for Classic McEliece and NewHope concern  
448 their IND-CPA-secure PKEs rather than IND-CCA-secure KEMs.

449 Taking all these factors into account, almost the only ranking that is quite clear from  
450 Tables 2 and 3 is the ranking of candidates that have results available for pure hardware  
451 implementations targeting high-speed. In this specific category, the ranking for the security  
452 level 1 is: 1. NewHope, 2. Classic McEliece, 3. FrodoKEM. If we assume that a  
453 software/hardware implementation of SIKE with a custom processor is almost as efficient  
454 as a pure hardware implementation, then we can also add SIKE at position 4. At level 3,  
455 NewHope does not have a variant, and at level 5, Classic McEliece and FrodoKEM, do  
456 not have high-speed pure hardware implementations reported.

457 In Tables 4 and 5, we summarize implementations targeting Xilinx Virtex-7 FPGAs.  
458 Unfortunately, the only conclusion that can be drawn from these tables is an advantage of  
459 Classic McEliece over SIKE in terms of all performance metrics other than the number of  
460 LUTs and flip-flops.

**Table 4:** Level 1 KEMs on Virtex-7 (default) and Virtex-6 (indicated with the superscript  $V_6$ )

Design	Algorithm	Type	Target	Max. Freq.	LUT	FF	Slice	DSP	BR AM	Key Generation		Encap./Enc. <sup><i>cpa</i></sup>		Decap./Dec. <sup><i>cpa</i></sup>		
										cycles	$\mu s$	cycles	$\mu s$	cycles	$\mu s$	
<b>Security Level I</b>																
[19]	SIKEp434	HW	HS	250	12,818	18,271	5,527	195	32.0	–	–	1,095,000	4,400.0	1,095,000	4,400.0	4,400.0
[19]	SIKEp503	HW	HS	244	13,963	19,935	6,163	225	34.0	–	–	1,440,000	5,900.0	1,440,000	5,900.0	5,900.0
[49]	SIKEp434	HW	HS	168	21,059	23,819	8,121	240	26.5	530,000	3,147.3	930,000	5,522.6	980,000	5,819.5	5,819.5
[49]	SIKEp503	HW	HS	166	23,746	27,609	8,907	264	33.5	640,000	3,857.7	1,140,000	6,871.6	1,200,000	7,233.3	7,233.3
[53]	SIKEp434	SW/HW	HS	142	21,210	13,657	7,408	162	38.0	981,180	6,900.0	1,677,960	11,800.0	1,777,500	12,500.0	12,500.0
[53]	SIKEp503	SW/HW	HS	142	21,210	13,657	7,408	162	38.0	1,166,040	8,200.0	1,976,580	13,900.0	2,104,560	14,800.0	14,800.0
[39] <sup><i>v6</i></sup>	LEDAkem-128 <sup><i>cpa</i></sup>	HW	LW	235	104	53	33	0	1.0	–	–	712,000	3,029.8	–	–	–
[53]	SIKEp434	SW/HW	LW	152	10,937	7,132	3,415	57	21.0	2,191,781	14,400.0	3,713,851	24,400.0	3,957,382	26,000.0	26,000.0
[53]	SIKEp503	SW/HW	LW	152	10,937	7,132	3,415	57	21.0	2,602,740	17,100.0	4,383,562	28,800.0	4,672,755	30,700.0	30,700.0

<sup>*cpa*</sup> Design of a KEM variant resistant against Chosen-Plaintext Attack (CPA)

<sup>*v6*</sup> Design implemented on Virtex-6

<sup>*o*</sup> Design for an old parameter set changed by the submitters on March 19th, 2020

**Table 5:** Level 3 & 5 KEMs and PKEs on Virtex-7

Design	Algorithm	Type	Target	Max. Freq.	LUT	FF	Slice	DSP	BR AM	Key Generation		Encap./Enc. $c_{pa}$		Decap./Dec. $c_{pa}$	
										cycles	$\mu s$	cycles	$\mu s$	cycles	$\mu s$
<b>Security Level III</b>															
[75]	mceliece460896 $c_{pa}$	HW	HS	131	109,484	168,939	0	446.0	515,806	3,943.5	3,360	25.7	17,931	137.1	
[19]	SIKEp610	HW	HS	239	16,226	26,757	270	38.5	–	–	2,280,000	9,550.0	2,280,000	9,550.0	
[49]	SIKEp610	HW	HS	166	28,217	33,297	312	39.5	900,000	5,428.2	1,810,000	10,916.8	1,780,000	10,735.8	
[53]	SIKEp610	SW/HW	HS	142	21,210	13,657	7,408	38.0	1,962,360	13,800.0	3,654,540	25,700.0	3,711,420	26,100.0	
[53]	SIKEp610	SW/HW	LW	152	10,937	7,132	3,415	21.0	4,353,120	28,600.0	8,097,412	53,200.0	8,219,178	54,000.0	
<b>Security Level V</b>															
[75]	mceliece6960119 $c_{pa}$	HW	HS	130	116,928	188,324	0	607.0	974,306	7,500.4	5,413	41.7	25,135	193.5	
[75]	mceliece6688128 $c_{pa}$	HW	HS	137	122,624	186,194	0	589.0	1,046,139	7,658.4	5,024	36.8	29,754	217.8	
[75]	mceliece8192128 $c_{pa}$	HW	HS	130	123,361	190,707	0	589.0	1,286,179	9,901.3	6,528	50.3	32,765	252.2	
[75]	mceliece6960119 $c_{pa}$	HW	LW	141	44,154	88,963	0	563.0	11,179,636	79,570.4	5,413	38.5	46,141	328.4	
[75]	mceliece6688128 $c_{pa}$	HW	LW	136	44,345	83,637	0	446.0	12,389,742	91,034.1	5,024	36.9	52,333	384.5	
[75]	mceliece8192128 $c_{pa}$	HW	LW	134	45,150	88,154	0	525.0	15,185,314	113,154.4	6,528	48.6	55,330	412.3	
[19]	SIKEp751	HW	HS	233	20,207	39,339	452	41.5	–	–	2,965,000	12,750.0	2,965,000	12,750.0	
[49]	SIKEp751	HW	HS	163	39,953	50,079	512	43.5	1,250,000	7,664.0	2,210,000	13,550.0	2,340,000	14,347.0	
[53]	SIKEp751	SW/HW	HS	142	21,210	13,657	7,408	38.0	2,516,940	17,700.0	4,166,460	29,300.0	4,479,300	31,500.0	
[53]	SIKEp751	SW/HW	LW	152	10,937	7,132	3,415	21.0	7,960,426	52,300.0	13,150,685	86,400.0	14,185,693	93,200.0	

 $c_{pa}$  Design of a PKE variant resistant against Chosen-Plaintext Attack (CPA)

461 In Tables 6 and 7, we summarize the results for ASICs. ASIC performance studies have  
462 been reported in [80], [12]-[13], and [26]. All three studies were conducted using different  
463 ASIC processes and standard-cell libraries. Therefore, the obtained results cannot be  
464 compared across any two, not to mention three, publications from this list. In all three  
465 cases, the approach was the development of a lattice-based co-processor supporting at least  
466 three different IND-CCA secure KEMs. The design presented in [80] is a domain-specific  
467 vector co-processor, leveraging the extensible RISC-V architecture. This co-processor has  
468 been integrated with an open-source RISC-V microprocessor, supporting the RV32IMC  
469 ISA. The RISC-V core has been modified to recognize the custom instructions and forward  
470 them to the vector co-processor. Similarly, in [12]-[13], the domain-specific Sapphire crypto-  
471 processor is coupled with an efficient RISC-V microprocessor, supporting the RV32IM  
472 ISA. In [26], the authors proposed an enhanced RISC-V architecture that embeds a set of  
473 powerful tightly coupled accelerators to speed up lattice-based PQC. These accelerators  
474 are deeply integrated into the RISC-V pipeline. RISC-V was also extended with 28 new  
475 instructions for performing packed modular arithmetic, butterfly operation, update of  
476 Twiddle factors, update/multiplication with scaling factors, bit-reversal, hash computations,  
477 and binomial sampling.

478 In all three co-processors, all supported lattice-based KEMs share the same resources.  
479 Therefore, only the comparison in terms of the execution time, power consumption, and  
480 energy was possible. All implementations adopted the SW/HW co-design approach. In [80],  
481 the target was high-speed. In [12]-[13] and [26] the minimum power and energy. In Table 6,  
482 we report the execution times, and in Table 7 both energy and power consumption at a  
483 specific operating frequency, common for all schemes compared within each study.

484 Performance estimates for ThreeBears have been provided in [32]. The implementations  
485 are assumed to be realized in the ASIC technology, with the standard-cell library TSMC  
486 40 LP. The target clock frequency is 100 MHz. Two accelerators, 32-bit and 64-bit, are  
487 considered. All estimates are made by using the arithmetic core from a Rambus public-key  
488 crypto accelerator and data reported in [12]. No actual implementation is attempted.

489 In the study reported in [12]-[13], for the TSMC 40nm library, the ranking of candidates in  
490 terms of the decapsulation time is 1. NewHope, 2. Kyber, 3. Frodo-KEM for the security  
491 level 1, and 1. Kyber, 2. NewHope, 3. Frodo-KEM for the security level 5. FrodoKEM  
492 lags behind the NewHope by a factor larger than 46 for the security level 1, and 280 for the  
493 security level 5. In terms of the power usage, the differences among all three candidates  
494 are very small. In terms of energy usage, NewHope and Kyber are very close to each other,  
495 and FrodoKEM lags behind by more than two orders of magnitude.

496 In terms of comparison with ThreeBears, the estimated performance of BabyBear is better  
497 than that reported for Kyber-512 and NewHope-512, for both 64-bit and 32-bit accelerators.  
498 Similarly, the performance of MamaBear is estimated to be better than that of Kyber-768  
499 for both types of accelerators. For PapaBear, only the 64-bit accelerator is estimated to  
500 be faster than Kyber-1024 and NewHope-1024. The 32-bit accelerator is slightly slower.  
501 However, one should keep in mind that the processor from [12] has been fully implemented,  
502 while the results in [32] are based on estimates only.

503 One of the advantages of ThreeBears is that its hardware implementations can share  
504 resources with implementations of classical public-key schemes, such as Elliptic Curve  
505 Cryptography and RSA. This feature may be particularly important during the transition  
506 period, when hybrid schemes, based on both classical public-key cryptography (mostly  
507 ECC) and PQC, are likely to be used concurrently. However, the importance of this  
508 resource sharing will likely diminish over time, when classical schemes are gradually  
509 phased out. On top of that, no PQC digital signature scheme based on similar big-integer  
510 arithmetic is currently under consideration for a new NIST PQC standard.

Table 6: All KEMs on ASIC

Design	Algorithm	Type	Target	Max. Freq.	Area (kGE)	SRAM (kB)	Key Gen.		Encapsulation		Decapsulation		Technology
							cycles	us	cycles	us	cycles	us	
<b>Security Level 1 &amp; 2</b>													
[80]	Kyber-512	SW/HW	HS	300	979	12.00	18,556	61.9	45,886	153.0	79,989	266.6	
[80]	NewHope-512	SW/HW	HS	300	979	12.00	18,563	61.9	44,513	148.4	84,501	281.7	TSMC 28 nm
[80]	LAC-128-v3a	SW/HW	HS	300	979	12.00	107,511	358.4	189,550	631.8	281,953	939.8	
[32] <sup>E</sup>	BabyBear	SW/HW	HS	100	145	-	24,300	243.0	36,900	369.0	60,800	608.0	
[32] <sup>E</sup>	BabyBear	SW/HW	LW	100	120	-	57,500	575.0	93,200	932.0	135,800	1,358.0	
[12]	NewHope-512	SW/HW	LW	72	106	40.25	52,063	723.1	136,077	1,890.0	142,295	1,976.3	TSMC 40 nm
[13]	Kyber-512	SW/HW	LW	72	106	40.25	54,861	762.0	134,965	1,874.5	146,068	2,028.7	
[12]	FrodoKEM-640	SW/HW	LW	72	106	40.25	11,453,942	159,082.5	11,609,668	161,245.4	12,035,513	167,159.9	
[26]	Kyber-512	SW/HW	LW	45	170	465*	150,106	3,316.5	193,076	4,265.9	204,843	4,525.9	
[26]	NewHope-512	SW/HW	LW	45	170	465*	123,860	2,736.6	207,299	4,580.2	226,742	5,009.8	UMC 65 nm
[26]	LightSaber	SW/HW	LW	45	170	465*	366,837	8,105.1	526,496	11,632.7	657,583	14,529.0	
<b>Security Level 3 &amp; 4</b>													
[32] <sup>E</sup>	MamaBear	SW/HW	HS	100	145	-	46,000	460.0	65,200	652.0	98,300	983.0	
[32] <sup>E</sup>	MamaBear	SW/HW	LW	100	120	-	121,000	1,210.0	177,800	1,778.0	238,900	2,389.0	
[13]	Kyber-768	SW/HW	LW	72	106	40.25	84,110	1,168.2	184,080	2,556.7	198,011	2,750.2	TSMC 40 nm
[12]	FrodoKEM-976	SW/HW	LW	72	106	40.25	26,005,326	361,185.1	29,749,417	413,186.3	30,421,175	422,516.3	
<b>Security Level 5</b>													
[80]	Kyber-1024	SW/HW	HS	300	979	12.00	39,689	132.3	81,569	271.9	136,475	454.9	TSMC 28 nm
[80]	NewHope-1024	SW/HW	HS	300	979	12.00	36,584	121.9	85,871	286.2	161,623	538.7	
[32] <sup>E</sup>	PapaBear	SW/HW	HS	100	145	-	74,200	742.0	100,900	1,009.0	143,000	1,430.0	
[13]	Kyber-1024	SW/HW	LW	72	106	40.25	116,841	1,622.8	236,886	3,290.1	256,828	3,567.1	
[12]	NewHope-1024	SW/HW	LW	72	106	40.25	97,969	1,360.7	236,812	3,289.1	258,872	3,595.4	TSMC 40 nm
[32] <sup>E</sup>	PapaBear	SW/HW	LW	100	120	-	206,400	2,064.0	288,400	2,884.0	368,000	3,680.0	
[12]	FrodoKEM-1344	SW/HW	LW	72	106	40.25	67,994,170	944,363.5	71,501,358	993,074.4	72,526,695	1,007,315.2	
[26]	Kyber-1024	SW/HW	LW	45	170	465*	349,673	7,725.9	405,477	8,958.8	424,682	9,383.2	
[26]	NewHope-1024	SW/HW	LW	45	170	465*	235,420	5,201.5	392,734	8,677.3	450,541	9,954.5	UMC 65 nm
[26]	FireSaber	SW/HW	LW	45	170	465*	1,300,272	28,728.9	1,622,818	35,855.5	1,898,051	41,936.6	

<sup>E</sup> The results reported in [32] were estimated using the arithmetic core from a Rambus public-key crypto accelerator and data reported in [12]. No actual implementation was attempted.

All SW/HW co-designs use RISC-V.

\* Numbers reported in kGE



**Table 7:** Power and Energy Comparison for all KEMs on ASIC

Design	Algorithm	Type	Target	Freq.	Area (kGE)	Memory (kB)	Key Generation		Encapsulation		Decapsulation		Technology
							Power (mW)	Energy ( $\mu$ J)	Power (mW)	Energy ( $\mu$ J)	Power (mW)	Energy ( $\mu$ J)	
<b>Security Level 1</b>													
[80]	Kyber-512	SW/HW	HS	300	979	12,00	29.26	1.81	23.67	3.62	24.94	6.65	
[80]	NewHope-512	SW/HW	HS	300	979	12,00	31.84	1.97	27.77	4.12	23.82	6.71	TSMC 28 nm
[80]	LAC-128-v3a	SW/HW	HS	300	979	12,00	25.90	9.28	24.33	15.37	23.74	22.31	
[12]	NewHope-512	SW/HW	LW	72	106	40.25	5.30	4.37	5.30	10.02	5.80	11.46	
[13]	Kyber-512	SW/HW	LW	72	106	40.25	6.02	4.59	5.43	10.18	5.95	12.07	TSMC 40 nm
[12]	FrodoKEM-640	SW/HW	LW	72	106	40.25	6.65	1,057.65	7.01	1,129.95	6.88	1,150.83	
[26]	NewHope-512	SW/HW	LW	10	170	465*	-	-	-	-	2.42	135.03	
[26]	Kyber-512	SW/HW	LW	10	170	465*	-	-	-	-	2.58	141.41	UMC 65 nm
[26]	LightSaber	SW/HW	LW	10	170	465*	-	-	-	-	2.78	431.18	
<b>Security Level 3</b>													
[13]	Kyber-768	SW/HW	LW	72	106	40.25	6.10	7.13	5.52	14.11	6.05	16.64	
[12]	FrodoKEM-976	SW/HW	LW	72	106	40.25	6.70	2,420.97	7.05	2,912.95	6.94	2,932.13	TSMC 40 nm
<b>Security Level 5</b>													
[80]	Kyber-1024	SW/HW	HS	300	979	12,00	35.45	4.69	29.20	7.94	25.57	11.63	
[80]	NewHope-1024	SW/HW	HS	300	979	12,00	29.36	3.58	24.53	7.02	23.57	12.70	TSMC 28 nm
[12]	NewHope-1024	SW/HW	LW	72	106	40.25	6.13	8.35	5.05	16.59	5.89	21.17	
[13]	Kyber-1024	SW/HW	LW	72	106	40.25	6.21	10.08	5.65	18.59	6.12	21.83	TSMC 40 nm
[12]	FrodoKEM-1344	SW/HW	LW	72	106	40.25	6.75	6,374.45	7.10	7,050.83	7.00	7,051.21	
[26]	NewHope-1024	SW/HW	LW	10	170	465*	-	-	-	-	2.41	259.98	
[26]	Kyber-1024	SW/HW	LW	10	170	465*	-	-	-	-	2.60	307.68	UMC 65 nm
[26]	FireSaber	SW/HW	LW	10	170	465*	-	-	-	-	2.77	1335.48	

All SW/HW co-designs using RISC-V RV32IM

\* Numbers reported in kGE

**Table 8:** All KEMs and PKEs on Zynq Ultrascale+

Design	Algorithm	Type	Target	Max. Freq.	LUT	FF	Slice	DSP	BRAM	Key Gen.		Encapsulation		Decapsulation	
										cycles	us	cycles	us	cycles	us
<b>Security Level 1</b>															
[18]	R5ND_IKEM_0d	SW/HW	HS	260	55,442	82,341	10,627	0	2	-	-	-	19.0	-	24.0
[18]	LightSaber	SW/HW	HS	322	12,343	11,288	1,989	256	3.5	-	-	-	53.0	-	56.0
[18]	FrodoKEM-640	SW/HW	HS	402	7,213	6,647	1,186	32	13.5	-	-	-	1,223.0	-	1,319.0
<b>Security Level 3</b>															
[63]	Saber	HW	HS	250	45,895	18,705	-	0	2	4,320	17.3	5,231	20.9	6,461	25.8
[63]	Saber	HW	HS	250	25,079	10,750	-	0	2	5,435	21.8	6,618	26.5	8,034	32.1
[18]	R5ND_3KEM_0d	SW/HW	HS	249	73,881	109,211	14,307	0	2	-	-	-	24.0	-	33.0
[18]	Saber	SW/HW	HS	322	12,566	11,619	1,993	256	3.5	-	-	-	60.0	-	65.0
[18]	FrodoKEM-976	SW/HW	HS	402	7087	6693	1190	32	17	-	-	-	1,642.0	-	1,866.0
<b>Security Level 5</b>															
[18]	R5ND_5KEM_0d	SW/HW	HS	212	91,166	151,019	18,733	0	2	-	-	-	32.0	-	42.0
[41]	NewHope-1024 <sup>cpa</sup>	HW	HS	406	13,961	8,149	-	25	18	-	-	34,000	83.0	30,600 <sup>KD</sup>	75,0 <sup>KD</sup>
[18]	FireSaber	SW/HW	HS	322	12,555	11,881	2,341	256	3.5	-	-	-	74.0	-	80.0
[18]	FrodoKEM-1344	SW/HW	HS	417	7,015	6,610	1,215	32	17.5	-	-	-	2,186.0	-	3,120.0

All SW/HW co-designs using ARM Cortex-A53

<sup>cpa</sup> Design of a PKE variant resistant against Chosen-Plaintext Attack (CPA)

<sup>KD</sup> total execution time of Key Generation and Decryption

**Table 9:** Digital Signature Schemes on Artix-7, Kintex-7 and Virtex-7

Design	Algorithm	Type	Target	Max. Freq.	LUT	FF	Slice	DSP	BR AM	Key Gen. cycles	Signature Verification us	Signature Generation us	Family		
<b>Security Level 1 &amp; 2</b>															
[42]	Picnic-L1-FS	HW	HS	91	90,535	23,516	25,160	0	52.5	–	29,600	325.6	31,300	344.3	
[baner19a]	qTESLA-I <sup>o2</sup>	SW/HW	LW	25*	14,975	2,539	4,173	11	14.0	4,846,949	193,878.0	1,556.9	168,273	6,730.9	
[baner19a]	Dilithium-I	SW/HW	LW	25*	14,975	2,539	4,173	11	14.0	95,202	3,808.1	5,703.0	376,392	15,055.7	
[baner19a]	Dilithium-II	SW/HW	LW	25*	14,975	2,539	4,173	11	14.0	130,022	5,200.9	7,397.3	514,246	20,569.8	
[76]	qTESLA-p-I	SW/HW	LW	121	7,212	4,378	2,438	15	139.0	925,431	7,648.2	946,520	4,165,160	34,422.8	
[24]	Rainbow-Ic <sup>o1</sup>	HW	HS	90	52,895	32,476	15,112	0	67.0	–	–	–	979	10.9	
[24]	Rainbow-Ia	HW	HS	111	27,712	27,679	8,939	0	59.0	–	–	–	1,980	17.8	
[42]	Picnic-L1-FS	HW	HS	125	90,037	23,105	–	0	52.5	–	29,600	237.0	31,300	250.0	
[24]	Rainbow-Ic <sup>o1</sup>	HW	HS	167	52,721	32,475	15,976	0	67.0	–	–	–	979	5.9	
[24]	Rainbow-Ia	HW	HS	181	27,556	27,675	7,065	0	59.0	–	–	–	1,980	10.9	
<b>Security Level 3</b>															
[baner19a]	qTesla-III-speed <sup>o2</sup>	SW/HW	LW	25*	14,975	2,539	4,173	11	14.0	11,898,241	475,929.6	67,712	2,708.5	12,683.3	
[baner19a]	qTesla-III-size <sup>o2</sup>	SW/HW	LW	25*	14,975	2,539	4,173	11	14.0	11,479,190	459,167.6	69,154	2,766.2	13,937.2	
[baner19a]	Dilithium-III	SW/HW	LW	25*	14,975	2,539	4,173	11	14.0	167,433	6,697.3	229,481	634,763	25,390.5	
[76]	qTESLA-p-III	SW/HW	LW	121	7,475	4,518	2,473	15	147.0	2,305,220	19,051.4	2,315,950	19,140.1	64,009.0	
<b>Security Level 4 &amp; 5</b>															
[42]	Picnic-L5-FS	HW	HS	125	167,530	33,164	–	0	98.5	–	–	146,600	1,173.0	154,500	1,236.0
[baner19a]	Dilithium-IV	SW/HW	LW	25*	14,975	2,539	4,173	11	14.0	223,272	8,930.9	276,221	11,048.8	32,625.4	

<sup>o1</sup> Design for a parameter set withdrawn at the beginning of Round 2<sup>o2</sup> Design for a heuristic parameter set withdrawn by the submitters on Aug. 20, 2019

All SW/HW co-designs using RISC-V RV32IM

\* Preliminary result

**Table 10:** Digital Signature Schemes on ASIC

Design	Algorithm	Type	Target	Max. Freq.	Area (kGE)	SRAM (kB)	Key Gen.		Signature Verification		Signature Generation		Technology
							cycles	us	cycles	us	cycles	us	
<b>Security Level 1 &amp; 2</b>													
[12]	qTESLA-I <sup>o2</sup>	SW/HW	LW	72	106	40.25	4,846,949	67,318.7	38,922	540.6	168,273	2,337.1	TSMC 40 nm
[12]	Dilithium-I	SW/HW	LW	72	106	40.25	95,202	1,322.3	142,576	1,980.2	376,392	5,227.7	
[12]	Dilithium-II	SW/HW	LW	72	106	40.25	130,022	1,805.9	184,933	2,568.5	514,246	7,142.3	
<b>Security Level 3</b>													
[12]	qTesla-III-speed <sup>o2</sup>	SW/HW	LW	72	106	40.25	11,898,241	165,253.3	67,712	940.4	317,083	4,403.9	TSMC 40 nm
[12]	qTesla-III-size <sup>o2</sup>	SW/HW	LW	72	106	40.25	11,479,190	159,433.2	69,154	960.5	348,429	4,839.3	
[12]	Dilithium-III	SW/HW	LW	72	106	40.25	167,433	2,325.5	229,481	3,187.2	634,763	8,816.2	
<b>Security Level 4</b>													
[12]	Dilithium-IV	SW/HW	LW	72	106	40.25	223,272	3,101.0	276,221	3,836.4	815,636	11,328.3	TSMC 40 nm

<sup>o2</sup> Design for a heuristic parameter set withdrawn by the submitters on Aug. 20, 2019

**Table 11:** Power and Energy Comparison for Digital Signature Schemes on ASIC

Design	Algorithm	Type	Target	Max. Freq.	Area (kGE)	SRAM (kB)	Key Generation Power (mW)	Key Generation Energy ( $\mu$ J)	Signature Power (mW)	Signature Verification Energy ( $\mu$ J)	Signature Power (mW)	Signature Generation Energy ( $\mu$ J)	Technology
<b>Security Level 1 &amp; 2</b>													
[12]	qTESLA-I <sup>o2</sup>	SW/HW	LW	72	106	40.25	7.89	531.55	7.99	4.32	9.99	23.34	TSMC 40 nm
[12]	Dilithium-I	SW/HW	LW	72	106	40.25	6.82	9.00	7.73	15.31	6.77	35.41	
[12]	Dilithium-II	SW/HW	LW	72	106	40.25	7.24	13.08	7.49	19.23	7.68	54.82	
<b>Security Level 3</b>													
[12]	qTesla-III-speed <sup>o2</sup>	SW/HW	LW	72	106	40.25	7.64	1,262.39	7.30	6.86	9.97	43.91	TSMC 40 nm
[12]	qTesla-III-size <sup>o2</sup>	SW/HW	LW	72	106	40.25	7.71	1,229.18	7.59	7.27	9.97	48.23	
[12]	Dilithium-III	SW/HW	LW	72	106	40.25	7.36	17.11	7.41	23.63	7.40	65.26	
<b>Security Level 4</b>													
[12]	Dilithium-IV	SW/HW	LW	72	106	40.25	6.89	21.38	7.44	28.55	6.93	78.53	TSMC 40 nm

<sup>o2</sup> Design for a heuristic parameter set withdrawn by the submitters on Aug. 20, 2019

All SW/HW co-designs using RISC-V RV32IM

511 In the study reported in [80], for the most advanced TSMC 28nm library, the ranking of  
512 candidates in terms of the most time-critical decapsulation time is 1. Kyber, 2. NewHope,  
513 3. LAC. The difference between positions 1 and 2 is about 6% for the security level 1,  
514 and 18% for the security level 5. LAC lags behind NewHope by a factor larger than 3.  
515 Kyber also uses less energy than NewHope for each major operation. The energy usage for  
516 decapsulation is almost identical for Kyber and NewHope. At the security level 1, Kyber  
517 uses about 1% less energy, and at the security level 5 about 7% less energy. At the same  
518 time, at the security level 1, LAC requires over 3 times more energy.

519 Finally, in the study reported in [26], for the UMC 65nm library, the ranking of candidates  
520 in terms of the decapsulation time is: 1. Kyber, 2. NewHope, 3. Saber. With very  
521 similar power consumption reported for all three candidates, the differences in terms of the  
522 execution time translate to the similar differences in energy usage. However, as explained  
523 before, this study may be unintentionally biased toward candidates that are able to take  
524 advantage of NTT. In particular, for decapsulation, the reported speed-up vs. the baseline  
525 pure software implementation running on RISC-V is 9.2, 7.4, and 2.3 for NewHope-512,  
526 Kyber-512, and LightSaber, respectively. Similarly, at level 5, the corresponding speed-ups  
527 are 9.4, 9.3, and 2.3, respectively. The speed-up for NewHope and Kyber came primarily  
528 from a dedicated NTT and Modular Arithmetic Unit. It resulted in a decrease in the  
529 decapsulation time for NewHope and Kyber by factors ranging between 2.7 and 3.6 as  
530 compared to the best Cortex-M4 software implementation reported to date. The speed-up  
531 for Saber came primarily from a half-word integer multiplication and accumulation block.

532 It resulted in the optimized software/hardware implementations of Saber performing  
533 decapsulation in approximately the same or slightly longer execution time than the best  
534 Cortex-M4 implementations. As a result, it is reasonable to conclude that additional  
535 optimizations might be possible for Saber, bringing its execution time and energy usage  
536 to the same range as those of NewHope and Kyber. However, more work is required to  
537 demonstrate these improvements in practice. The differences between Kyber and NewHope  
538 in terms of the execution times for encapsulation and decapsulation remained at or below  
539 10% for both security levels.

540 In Table 8, we compare results reported by our own group at the end of 2019 in [18],  
541 with results reported by other groups for Saber and NewHope, respectively. All results  
542 were obtained using the same SoC FPGA, Zynq UltraScale+. The software/hardware  
543 implementation of Round5 was very close to the pure hardware implementation. The same  
544 was not the case for the software/hardware implementation of Saber, where a significant  
545 percentage of the execution time was devoted to functions remaining in software and to  
546 the transfer of data and control between software and hardware. As a result, the most  
547 accurate comparison between Round5 and Saber is possible at the security level 3, for  
548 which the pure hardware implementation of Saber was reported in [63]. Based on this  
549 implementation Saber outperforms Round5 by a small margin in terms of the execution  
550 times for encapsulation and decapsulation. At the same time, even the fastest reported  
551 implementation of Saber uses 1.6x fewer LUTs than Round5, with the same number of  
552 BRAMs and DSP units. FrodoKEM is demonstrated to be by far slower than Saber and  
553 Round5 for all security levels.

554 Somewhat differently, for the security level 5, the pure hardware implementation of  
555 NewHope, reported in [41], is not fast enough to outperform the software/hardware  
556 implementation of Round5 from [18]. However, the comparison is somewhat complicated  
557 by the fact that, in [41], the results are reported the IND-CPA-secure PKE (rather than the  
558 IND-CCA-secure KEM), and only the sum of the key generation and decryption (rather  
559 than the decryption itself) is reported in the paper.

560 In Tables 9, 10, and 11, we summarize results available for the implementations of  
561 digital signatures. The implementations targeting FPGAs are considered first in Table 9.  
562 Unfortunately, multiple results available for qTESLA concern heuristic parameter sets

563 that have been withdrawn by submitters on Aug. 20, 2019. Among the remaining designs,  
 564 for Artix-7, the ranking of candidates for the security level 1 is 1. Picnic, 2. Dilithium,  
 565 and 3. qTESLA. The differences among these candidates in terms of the execution time  
 566 for the signature generation (more critical) and signature verification are very significant.  
 567 At the same time, only the implementation of Picnic is a high-speed and pure hardware  
 568 implementation. The remaining implementations are software/hardware implementations  
 569 based on RISC-V. Additionally, the number of LUTs for Picnic is approximately 6 times  
 570 larger than for Dilithium, and the number of BRAMs, 3.75 times larger. At the same  
 571 time, compared to Picnic, the execution time for signature generation is 12 times longer  
 572 for Dilithium-I and 16 times longer for Dilithium-II.

573 For security level 3, no implementation of Picnic is available. The implementations of  
 574 Dilithium-III and qTESLA-p-III are comparable in terms of type, target, and resource  
 575 utilization. At the same time, the implementation of Dilithium is an order of magnitude  
 576 more efficient. The implementations of digital signature schemes targeting Kintex-7 and  
 577 Virtex-7 are summarized in the same table. For the Kintex-7 implementations, Rainbow  
 578 substantially outperforms Picnic at the security level 1. For all remaining families and  
 579 security levels, only one candidate with the up-to-date parameter set is reported.

580 In Tables 10 and 11, Dilithium and qTESLA are compared from the point of view of  
 581 their execution time, energy usage, and power consumption in ASICs. Unfortunately, the  
 582 practical importance of the underlying study, reported in [12] and performed in the first  
 583 half of 2019, was diminished by the use of heuristic parameter sets of qTESLA, withdrawn  
 584 by the submitters on Aug. 20, 2019.

### 585 3 Choice of Algorithms to Implement

586 In this paper, we focus on KEMs with indistinguishability under chosen-ciphertext attack  
 587 (IND-CCA). Our primary goal was to implement all lattice-based IND-CCA secure KEMs  
 588 described in the specifications of Round 2 PQC candidates. Eventually, we fell short of  
 589 this goal by not implementing a KEM of a single lattice-based candidate, Three Bears.  
 590 Additionally, we focused on Ring Learning with Rounding (RLWR) variants of Round5,  
 591 and thus, we did not attempt to implement any LWR variants of this submission.

592 The submission packages of four candidates – LAC, NTRU, NTRU Prime, and Round5  
 593 – describe two substantially different KEMs each. As a result, we have implemented 12  
 594 KEMs representing 8 Round 2 candidates. For each implemented KEM, we generated  
 595 results for all supported security levels.

596 With a few exceptions, we did not generate results for the underlying public-key encryption  
 597 schemes (PKE) or concurrently proposed IND-CPA secure KEMs. The reason for that  
 598 was a focus on the highest-level schemes, which could be securely used to agree on shared  
 599 session keys, based on the long-term public-private key pairs valid for an extended period of  
 600 time. In this scenario, the time of the public-private key-pair generation is non-critical, and  
 601 the design can focus entirely on minimizing the time of encapsulation and decapsulation.  
 602 All implemented PQC candidates can be divided into the following major sub-families,  
 603 listed below together with their Round 2 representatives:

- 604 • LWE : Learning With Errors - FrodoKEM
- 605 • RLWE : Ring Learning with Errors - LAC (including LAC-v3a and LAC-v3b) and  
606 NewHope
- 607 • Module-LWE: Module Learning with Errors - CRYSTALS-KYBER
- 608 • RLWR : Ring Learning With Rounding - Round5 (with and without an error  
609 correcting code)

Table 12: Features of selected NIST Round 2 PQC KEMs

Feature	LAC-(v3a/v3b)	NewHope	Round5	Kyber	Saber	FrodoKEM
Underlying problem	Ring-LWE: Ring Learning With Errors	Ring-LWE: Ring Learning With Errors	RLWR: Ring Learning With Rounding	Module-LWE: Module Learning with Errors	Mod-LWR: Module Learning with Rounding	LWE: Learning With Errors
Degree $n$	Power of 2 Byte-level	Power of 2 Prime	$2^8 < n < 2^{11}$ Power of 2	Power of 2 Prime	Power of 2 Power of 2	$n \equiv 0 \pmod{8}$ Power of 2
Modulus $q$	Prime / Power of 2	Prime	Power of 2	Prime	Power of 2	Power of 2
Other major parameters	$\psi_n^h$ : Binomial distribution, $[e, l_m, l_d]$ : BCH code	$k$ : noise parameter, $\gamma$ : NTT parameter	$p, t$ : other moduli	$k$ : the lattice dimension as a multiple of $n$ , $\eta$ : noise parameter	$l$ : number of polynomials per vector, $p, T$ : other moduli, $\mu$ : parameter of CBD	$B$ : number of bits, encoded in each matrix entry, $\sigma$ : standard deviation
Hash-based functions	SHA3-512 SHAKE256	SHAKE128, SHAKE256	1: SHAKE128 3, 5: SHAKE256	SHA3-256, SHA3-512, SHAKE128, SHAKE256	SHA3-256, SHA3-512, SHAKE128	1: SHAKE128 3, 5: SHAKE256
Sampling	Integers are sampled from a fixed-weight centered binomial distribution (CBD)	Integers are sampled from a centered binomial distribution (CBD)	Integers from a uniform distribution are produced by a DRBG taking a random seed	Integers are sampled from a centered binomial distribution (CBD)	Integers are sampled from a centered binomial distribution (CBD)	Integers are sampled from an approximation of a rounded continuous Gaussian distribution
Decryption failures	Yes	Yes	Yes	Yes	Yes	Yes
Polynomial Rings	$\mathbb{Z}_q[x]/(x^n + 1)$	$\mathbb{Z}_q[x]/(x^n + 1)$	$\mathbb{Z}_q[x]/\Phi_{n+1}^{**}$	$\mathbb{Z}_q[x]/(x^n + 1)$	$\mathbb{Z}_q[x]/(x^n + 1)$	None
#Polynomial Multiplications in Encapsulation	2	2	2	$k^2 + k$	$l^2 + l$	None 2 matrix-by-matrix*
#Polynomial Multiplications in Decapsulation	3	3	3	$k^2 + 2k$	$l^2 + 2l$	None 3 matrix-by-matrix*

\* Elements of matrices in  $\mathbb{Z}_q$ \*\*  $\Phi_{n+1} = (x^{n+1} - 1)/(x - 1)$



**Table 13:** Features of NIST Round 2 NTRU-based PQC KEMs

Feature	NTRU-HPS	NTRU-HRSS	Streamlined NTRU Prime	NTRU LPRime
Underlying problem	Shortest Vector Problem	Shortest Vector Problem	Shortest Vector Problem	Shortest Vector Problem
Polynomial $P$	$x^n - 1$	$\Phi_n = (x^n - 1)/(x - 1)^{**}$	$x^n - x - 1$ irreducible in $\mathbb{Z}_q[x]$	$x^n - x - 1$ irreducible in $\mathbb{Z}_q[x]$
Degree $n^*$	Prime	Prime	Prime	Prime
Modulus $q$	power of 2 with $q/8 - 2 \leq 2n/3$	power of 2 with $q > 8\sqrt{2}(n + 1)$	Prime	Prime
Other major parameters	$w$ : Fixed weight for f and r	N/A	$w$ : Fixed weight for f and r. $3w \leq 2n$ $16w + 1 \leq q$	$w$ : Fixed weight for b and a. $3w \leq 2n$ $16w + 2\delta + 3 \leq q$
Hash-based functions	SHA3-256	SHA3-256	SHA3-512	SHA3-512
Sampling	Fixed-weight and variable-weight polynomials are sampled from a uniform distribution	Variable-weight polynomials are sampled from a uniform distribution	Fixed-weight polynomials are sampled from a uniform distribution	Fixed-weight polynomials are sampled from a uniform distribution
Decryption failures	No	No	No	No
Polynomial Rings	$\mathbb{R}/q$ : $\mathbb{Z}_q[x]/(x^n - 1)$ $\mathbb{S}/q$ : $\mathbb{Z}_q[x]/(\Phi_n)^{**}$ $\mathbb{S}/3$ : $\mathbb{Z}_3[x]/(\Phi_n)^{**}$	$\mathbb{R}/q$ : $\mathbb{Z}_q[x]/(x^n - 1)$ $\mathbb{S}/3$ : $\mathbb{Z}_3[x](x - 1)/(x^n - 1)$	$\mathbb{R}/q$ : $\mathbb{Z}_q[x]/(x^n - x - 1)$ $\mathbb{R}/3$ : $\mathbb{Z}_3[x]/(x^n - x - 1)$	$\mathbb{R}/q$ : $\mathbb{Z}_q[x]/(x^n - x - 1)$ $\mathbb{R}/3$ : $\mathbb{Z}_3[x]/(x^n - x - 1)$
#Polynomial Multiplications in Encapsulation	1 in $\mathbb{R}/q$	1 in $\mathbb{R}/q$	1 in $\mathbb{R}/q$	2 in $\mathbb{R}/q$
#Polynomial Multiplications in Decapsulation	1 in $\mathbb{R}/q$ 1 in $\mathbb{S}/q$ 1 in $\mathbb{S}/3$	1 in $\mathbb{R}/q$ 1 in $\mathbb{S}/q$ 1 in $\mathbb{S}/3$	2 in $\mathbb{R}/q$ 1 in $\mathbb{R}/3$	3 in $\mathbb{R}/q$

\* Denoted by  $p$  in the specification of Streamlined NTRU Prime and NTRU LPRime

\*\*  $\Phi_n = (x^n - 1)/(x - 1)$  irreducible in  $\mathbb{Z}_q[x]$

- 610 • Mod-LWR : Module Learning with Rounding - Saber,
- 611 • NTRU-based : NTRU (including NTRU-HPS and NTRU-HRSS) and NTRU Prime
- 612 (including Streamlined NTRU Prime and NTRU LPRime).

613 Both implemented variants of LAC were announced in the middle of Round 2, on Dec. 19,  
614 2019. The implemented variants of the remaining algorithms have remained unchanged  
615 since the beginning of Round 2.

616 The following two submissions did not limit the generation of pseudorandom bits to any  
617 particular algorithm (e.g., SHAKE): LAC and NTRU. As a result, for each of them, we  
618 selected a variant of a pseudorandom number generator most efficient on our benchmarking  
619 platform. In the case of CRYSTALS-Kyber, we selected one of the variants described in  
620 the specification - a variant based on the SHA-3 functions.

621 Selected features of all implemented KEMs are summarized in Tables 12 and 13.

622 In all of these KEMs, the elementary operation is multiplication mod  $q$ . In FrodoKEM,  
623 LAC-v3b, Round5, Saber, NTRU-HPS, and NTRU-HRSS,  $q$  is a power of two, which  
624 significantly simplifies the reduction mod  $q$ . In NewHope and Kyber,  $q$  is a special prime,  
625 selected in such a way to support speeding up polynomial multiplication in  $\mathbb{Z}_q[x]/(x^n + 1)$   
626 using the Number Theoretic Transform (NTT). In LAC-v3a,  $q$  is a one-byte prime (251).  
627 In Streamlined NTRU Prime and NTRU LPRime, it is a prime smaller than  $2^{13}$ . The  
628 moduli chosen for NTRU Prime algorithms may potentially lead to a higher resistance  
629 against future attacks.

630 In FrodoKEM, the most time-consuming operation is a matrix-by-matrix multiplication,  
631 where each component of a matrix is an element of  $\mathbb{Z}_q$ . In Kyber and Saber, the most

**Table 14:** Parameter sets of investigated algorithms. Notation: Sk - Secret Key, Pk - Public key, Ct - Ciphertext.

Algorithm	Parameter Set	Security Level	Degree $n$	Modulus $q$	Sk Size [bytes]	Pk Size [bytes]	Ct Size [bytes]
FrodoKEM	Frodo-640	1	640	$2^{15}$	19,888	9,616	9,720
Kyber	KYBER512	1	256	3329	1,632	800	736
LAC-v3a	LAC-128	1	512	251	1,056	544	704
LAC-v3b	LAC-128	1	512	256	1,056	544	704
NewHope	NEWHOPE512-CCA-KEM	1	512	12289	1,888	928	1,120
NTRU-HPS	ntruhs2048677	1*	77	$2^{11}$	1,235	931	931
NTRU-HRSS	ntruhss701	1*	701	$2^{13}$	1,452	1,138	1,138
Str NTRU Prime	kem/sntrup653	2	653	$4621 < 2^{13}$	1,518	994	897
NTRU LPRime	kem/ntrulpr653	2	653	$4621 < 2^{13}$	1,125	897	1,025
Round5	R5ND_CCA_1KEM_0d	1	586	$2^{13}$	708	676	740
Round5	R5ND_CCA_1KEM_5d	1	508	$2^{10}$	493	461	620
Saber	LightSaber-KEM	1	256	$2^{13}$	1,568	672	736
FrodoKEM	Frodo-976	3	976	$2^{16}$	31,296	15,632	15,744
Kyber	KYBER768	3	256	3329	2,400	1,184	1,088
LAC-v3a	LAC-192	3	1024	251	2,080	1,056	1,352
LAC-v3b	LAC-192	3	1024	256	2,080	1,056	1,352
NTRU-HPS	ntruhs4096821	3*	821	$2^{12}$	1,592	1,230	1,230
Str NTRU Prime	kem/sntrup761	3	761	$4591 < 2^{13}$	1,763	1,158	1,039
NTRU LPRime	kem/ntrulpr761	3	761	$4591 < 2^{13}$	1,294	1,039	1,167
Round5	R5ND_CCA_3KEM_0d	3	852	$2^{12}$	1,031	983	1,103
Round5	R5ND_CCA_3KEM_5d	3	756	$2^{12}$	828	780	934
Saber	Saber-KEM	3	256	$2^{13}$	2,304	992	1,088
Str NTRU Prime	kem/sntrup857	4	857	$5167 < 2^{13}$	1,463	1,184	1,312
NTRU LPRime	kem/ntrulpr857	4	857	$5167 < 2^{13}$	1,999	1,322	1,184
FrodoKEM	Frodo-1344	5	1344	$2^{16}$	43,088	21,520	21,632
Kyber	KYBER1024	5	256	3329	3,168	1,568	1,568
LAC-v3a	LAC-256	5	1024	251	2,080	1,056	1,464
LAC-v3b	LAC-256	5	1024	256	2,080	1,056	1,464
NewHope	NEWHOPE1024-CCA-KEM	5	1024	12289	3,680	1,824	2,208
Round5	R5ND_CCA_5KEM_0d	5	1170	$2^{13}$	1,413	1,349	1,509
Round5	R5ND_CCA_5KEM_5d	5	946	$2^{11}$	1,042	978	1,285
Saber	FireSaber-KEM	5	256	$2^{13}$	3,040	1,312	1,472

\* assuming non-local computational models

632 time-consuming operations are matrix-by-vector and vector-by-vector multiplications,  
633 where each element of a matrix or a vector is a polynomial with  $n$  coefficients in  $Z_q$ , and  
634 the multiplication of such polynomials is performed modulo the reduction polynomial  
635  $x^n + 1$ . In New Hope, LAC, Round5, and all NTRU-based KEMs, the most time-consuming  
636 operation is a polynomial multiplication.

637 The only KEMs with no Decryption Failure in the underlying PKE are NTRU-based  
638 KEMs (NTRU-HPS, NTRU-HRSS, Streamlined NTRU Prime, and NTRU LPRime).

639 Round5 and NTRU-based KEMs use sampling from the uniform distribution. In LAC,  
640 NewHope, Kyber, and Saber, a Centered Binomial Distribution (CBD) is used. In  
641 FrodoKEM, an approximation of a rounded continuous Gaussian distribution is required.  
642 Parameter sets of 12 investigated algorithms are summarized in Table 14. The specification  
643 of NTRU associates two different security categories with each parameter set for NTRU-

644 HPS and NTRU-HRSS. In this paper, we conservatively assumed the lower security level  
 645 based on the so-called non-local computational models (see [72], Section 5.3 Security  
 646 Categories). The same computation model is implicitly assumed by the submitters of the  
 647 other investigated algorithms.

648 In Table 14, we have divided parameter sets into three groups with security levels 1 and 2, 3  
 649 only, and 4 and 5, respectively. Only the first group contains variants of all 12 investigated  
 650 algorithms (with ten at level 1 and two at level 2). The second group includes 10 variants  
 651 at the security level 3. Finally, the last group includes 10 variants total (with two at level  
 652 4 and eight at level 5).

## 653 4 Methodology

### 654 4.1 Assumptions

655 All implemented schemes are Key Encapsulation Mechanisms (KEMs). For each of them,  
 656 we support two major operations: Encapsulation and Decapsulation. Whenever possible,  
 657 hardware resources and software functions are shared between these two operations. All  
 658 parameter sets of the given PQC scheme share the same HDL code. At the same time,  
 659 the choice among parameter sets is made at the time of synthesis, so the exact amount of  
 660 FPGA resources required to implement each particular parameter set can be determined  
 661 and reported. The key generation is assumed to be performed in software or using a  
 662 separate hardware unit.

663 Based on the considerations discussed in Section 1, our optimization target is high-speed  
 664 for both hardware and software/hardware implementation approaches. In both cases, the  
 665 primary goal is the minimum execution time for Encapsulation and Decapsulation. No  
 666 explicit limits are imposed on any resources of the FPGA platform, such as Configurable  
 667 Logic Block Slices, LUTs, flip-flops, BRAMs, or DSP units. The goal is to demonstrate  
 668 each algorithm’s inherent ability to execute multiple operations in parallel.

669 All implementations are required to be constant-time to make them resistant against  
 670 any known timing attacks. No physical access to the device or its proximity is assumed,  
 671 which means that countermeasures against power-based and electromagnetic analysis-based  
 672 attacks are considered non-essential. Developing and implementing such countermeasures  
 673 is beyond the scope of this study.

674 HDL code is required to be portable among multiple state-of-the-art FPGA families of  
 675 Xilinx and Intel, assuming that a given design fits in the largest device of a given family.  
 676 The code does not use any vendor or family-specific primitives or megafunctions. Each  
 677 hardware unit uses only a single clock. This clock can operate at an arbitrary clock  
 678 frequency lower than or equal to the maximum clock frequency determined by the critical  
 679 path of a given hardware unit. All reported execution times correspond to this maximum  
 680 clock frequency.

### 681 4.2 Choice of Benchmarking Platforms for Round 2

682 **Hardware.** The submissions selected for the hardware-only implementations (CRYSTALS-  
 683 KYBER, LAC, New Hope, and Round5) have moderate resource requirements, even when  
 684 optimized for high-speed. As a result, we have decided to generate results for two FPGA  
 685 families: Artix-7 and Virtex-7. Based on Section 2, these families were selected for  
 686 benchmarking by the largest number of other groups to date.

687 **Software/hardware co-design.** In recent years, several hardware/software co-design  
 688 platforms have emerged. The most popular in the industry are those based on integrating  
 689 an ARM-based processor and FPGA fabric on a single chip. Examples include Xilinx  
 690 Zynq 7000 System on Chip (SoC), Xilinx Zynq UltraScale+ MPSoC, Intel Cyclone V SoC

691 FPGAs, Intel Arria 10 SoC FPGAs, and Intel Agilex F-Series SoC FPGAs. These devices  
692 support software/hardware co-designs based on a traditional high-level language program  
693 running on an ARM processor, with the most time-critical computations performed on a  
694 dedicated hardware accelerator. The advantages of these platforms include the use of the  
695 most popular embedded processor family (ARM) operating at high speed (1 GHz or above),  
696 state-of-the-art commercial tools (available for free, or at a reduced price for academic  
697 use), availability of relatively inexpensive prototyping boards, and practical deployment in  
698 multiple environments.

699 The primary alternatives are FPGA-based systems with so-called "soft" processor cores  
700 implemented in reconfigurable logic. Examples include Xilinx MicroBlaze, Intel Nios II, and  
701 the open-source RISC-V, originally developed at the University of California, Berkeley [59,  
702 78, 79]. The main advantage of these systems over "hard" processor cores is flexibility in  
703 the allocation of resources to processor cores, including the possibility of extending them  
704 with special instructions specific to PQC. Additionally, they are easy to port between  
705 different FPGA families, and even between FPGAs and ASICs. A disadvantage compared  
706 to the "hard" option is that the "soft" processors operate at much lower clock frequencies  
707 (typically 200-450 MHz).

708 During Round 2, NIST asked designers to focus on the ARM Cortex-M4 for embedded  
709 software implementations and the Artix-7 for FPGA implementations. However, we are not  
710 aware of any SoC FPGA that contains a Cortex-M processor and the Artix-7 FPGA fabric  
711 on a single chip. Even if such a chip existed, it would be more suitable for benchmarking  
712 of lightweight implementations (optimized for minimum cost and power consumption),  
713 rather than benchmarking of the high-speed implementations targeted by our study.

714 As a result, we have based our choice of a platform primarily on the projected practical  
715 importance of various platforms during the initial period of deploying new PQC standards,  
716 and the expected speed-up over pure-software implementations. These priorities led us to  
717 choose devices from the "hard" processor class, with a hard-wired ARM processor, and  
718 among them, the Zynq UltraScale+ family from Xilinx Inc., the vendor with the biggest  
719 market share in this device category. Zynq UltraScale+ and similar SoC FPGAs are likely  
720 to be used for practical deployments of PQC in the near future, wherever device speed  
721 and time-to-market are of primary concern. Implementations using these devices are even  
722 more likely than implementations using only hardware.

723 However, the use of soft-core processors, and in particular the free and open-source RISC-  
724 V, should be considered as a natural next step, especially in light of DARPA's recent  
725 selection of the RISC-V Instruction Set Architecture (ISA) for investigation within its  
726 cybersecurity-related programs [55]. Since these soft-core processors can be implemented  
727 practically on any modern FPGA family, the choice of the family should be dependent  
728 primarily on the selected type of implementation: lightweight vs. high-speed.

729 Based on the above discussion, we chose the Xilinx Zynq UltraScale+ MPSoC XCZU9EG-  
730 2FFVB1156E as our target device and the Xilinx ZCU102 Evaluation Kit as a prototyping  
731 board.

732 Our target device, Xilinx Zynq UltraScale+ MPSoC XCZU9EG-2FFVB1156E, is composed  
733 of two major parts sharing the same chip. The primary component of the Processing  
734 System (PS) is a quad-core ARM Cortex-A53 Application Processing Unit, running at  
735 1.2 GHz. As in the software benchmarking experiments conducted by other groups, we  
736 utilize only one core in all our experiments. The Programmable Logic (PL) includes  
737 a programmable FPGA fabric similar to that of Virtex UltraScale+ FPGAs, including  
738 Configurable Logic Block (CLB) slices, Block RAMs, DSP units, etc. The frequency of  
739 operation depends on the particular logic instantiated in the reconfigurable fabric but  
740 typically does not exceed 400 MHz.

741 **Computer-Aided Design Tools.** The software used is Xilinx Vivado Design Suite HLx  
742 Edition, and Xilinx Software Development Kit (XSDK), all with version number 2018.2.

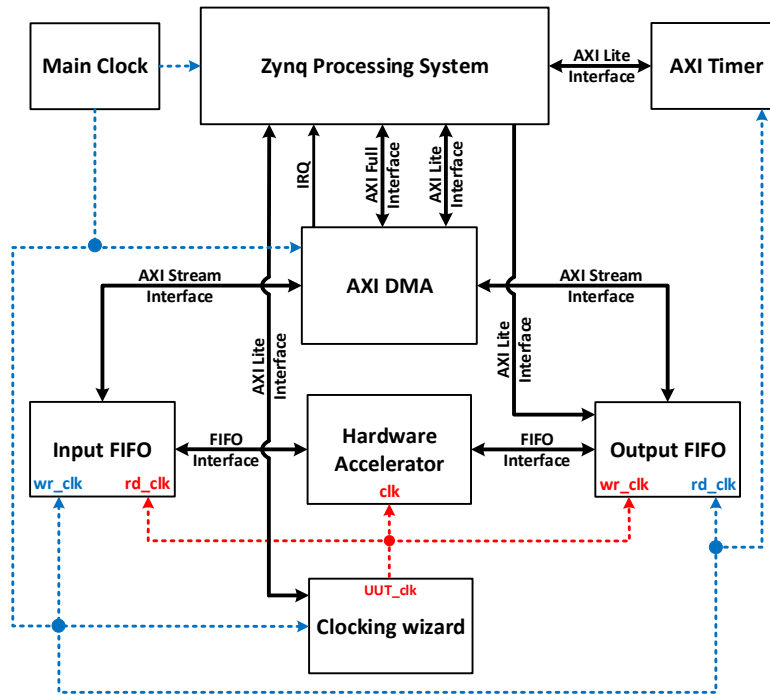


Figure 1: Block diagram of software/hardware co-design.

### 743 4.3 Benchmarking Setup for Software/Hardware Co-design

744 A high-level block diagram of the experimental software/hardware co-design platform is  
 745 shown in Fig. 1. The Hardware Accelerator is connected, through the dual-clock Input  
 746 and Output FIFOs, to the AXI DMA, supporting the high-speed communication with  
 747 the Processing System. Timing measurements are performed using the popular Xilinx  
 748 IP unit called AXI Timer, which is capable of measuring time in clock cycles of the 200  
 749 MHz system clock. The Hardware Accelerator can operate at a variable clock frequency,  
 750 controlled from software using the Clocking wizard unit.

### 751 4.4 Interface and Communication Protocol

752 The interface of the hardware accelerator is shown in Fig. 2. This interface is assumed to  
 753 be identical for both hardware and software/hardware implementations and matches the  
 754 interface of the Input and Output FIFOs, shown in Fig. 3. The default width of the data  
 755 bus is 64 bits. Each particular operation, such as load public key, start encapsulation, etc.,  
 756 is initiated by sending an appropriate header (in the form of a single 64-bit word) from  
 757 a program running on the ARM processor to the data input of a hardware accelerator.  
 758 When an operation requires additional data, this data is transmitted using the subsequent  
 759 Input FIFO words.

760 After the hardware accelerator produces results or detects an error, a header word is sent  
 761 in the opposite direction. If an additional output is required, this output follows the header  
 762 and is arranged in 64-bit words. The detailed format of the exchanged inputs and outputs  
 763 is left up to the designer of a hardware accelerator.

764 Compared to an earlier proposed PQC Hardware API [25], the adopted interface is  
 765 significantly simpler and more flexible. Only one input port, *infifo*, is used in place of  
 766 three separate ports, Public Data Input (PDI), Secret Data Input (SDI), and Random  
 767 Data Input (RDI). Only one output port, *outfifo*, is used in place of two separate ports,

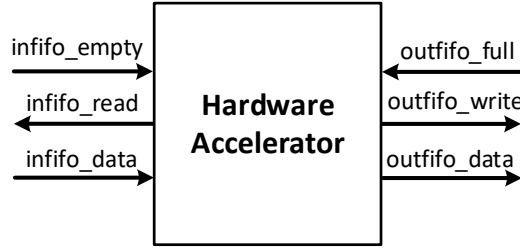
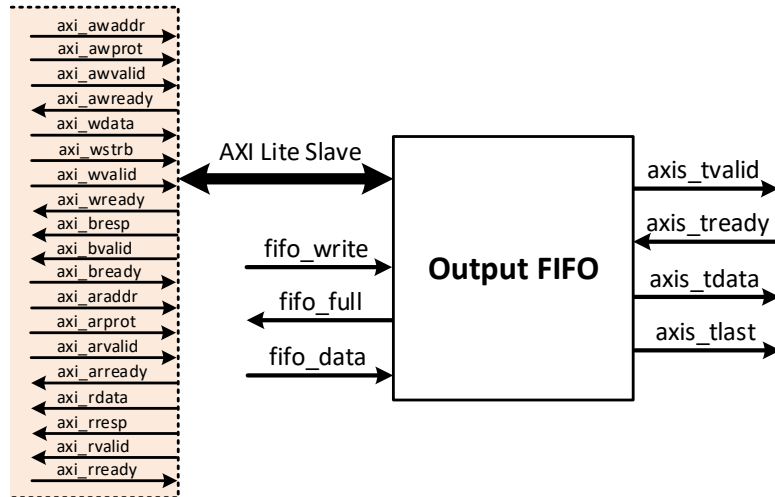


Figure 2: Hardware accelerator interface.



(a) Input FIFO



(b) Output FIFO

Figure 3: The Input and Output FIFO Interface.

768 Public Data Output (PDO) and Secret Data Output (SDO).  
 769 The proposed interface does not provide physical separation among the public, secret, and  
 770 random data. Still, it appears to be sufficient at the current stage of the evaluation process  
 771 for both pure hardware and software/hardware implementations. It also significantly  
 772 simplifies the software/hardware partitioning and transfer of data between the processor  
 773 and the hardware accelerator.

#### 774 4.5 Porting Software Implementations to ARM Cortex-A53

775 To minimize overhead, we have run software in the Bare Metal mode, without any operating  
 776 system. We have started from the best high-level language implementations of selected  
 777 candidates available to date. In order to be run on ARM Cortex-A53 in the Bare Metal  
 778 mode, these implementations had to be modified as described below.  
 779 Since no functions of Open-SSL are available in the Bare Metal mode, we have adopted  
 780 for AES the Optimized ANSI C code of the Rijndael cipher based on the use of T-boxes,  
 781 developed by Vincent Rijmen, Antoon Bosselaers, and Paulo Barreto [62]. Compared

782 to the OpenSSL implementation, the selected implementation is written entirely in C,  
 783 rather than in an assembly language of a specific processor. It does not contain any  
 784 countermeasures against cache-timing attacks.

785 For SHA-3, for all candidates other than Round5, we adopted fips202.c from SUPERCOP  
 786 by Ronny Van Keer, Gilles Van Assche, Daniel J. Bernstein, and Peter Schwabe. For  
 787 Round5, we used r5\_xof\_shake.c by Markku-Juhani O.Saarinen and keccak1600.c from  
 788 SUPERCOP, by the same authors as fips202.c

789 For all investigated KEMs, the encapsulation operation uses multiple calls to the function  
 790 `randombytes()`, which produces a sequence of random bytes with uniform distribution.  
 791 Other PQC benchmarking projects use a version of this function based on operating system  
 792 functions and/or functions from OpenSSL [**berns19**, 64, 43, 68]. None of these options  
 793 is available in the Bare Metal mode. Therefore, in our code, we use the implementation  
 794 of `randombytes()` proposed by Saarinen in April 2018 [64], which is an improved version  
 795 of the implementation developed by NIST for the generation of known-answer tests [58].  
 796 Since both of these implementations are based on AES in the ECB mode, from the  
 797 OpenSSL library, we have replaced the code of AES by the mentioned above standalone,  
 798 optimized implementation of AES in C [62]. As a result, the selected implementation of  
 799 `randombytes()` is likely to have different timing characteristics than the implementations  
 800 used in other benchmarking studies, such as SUPERCOP [**berns19**], pqcbench [64],  
 801 pqm4 [43], and liboqs [68].

802 Taking into account that the C implementations of NTRU-HPS, NTRU-HRSS, and  
 803 Streamlined NTRU Prime use `randombytes()` to generate 3211, 1400, and 2611 bytes,  
 804 respectively, we have sped up these function calls by a) limiting the number of bytes  
 805 returned by `randombytes()` to 32, and b) generating the remaining random bytes using  
 806 SHAKE128. For the three KEMs mentioned above, this change resulted in the speed-up  
 807 of the relevant functions by a factor greater than 3.

808 No attempt at the optimization of the software implementations of KEMs by employing  
 809 assembly language coding has been made.

## 810 **4.6 Software Profiling, C Source Code Analysis, and Software/Hardware** 811 **Partitioning**

812 Our first step in evaluating the suitability of cryptographic algorithms for software/hardware  
 813 co-design was profiling of their software implementations using one core of the ARM  
 814 Cortex-A53. Profiling produced a list of the most time-consuming functions, including  
 815 their absolute execution time, percentage execution time, and the number of times they  
 816 are called.

817 We decided which functions to offload to hardware based on the highest potential for the  
 818 total speed-up, as well as the fairness of comparison among investigated algorithms. The  
 819 total speed-up obtained by offloading an operation to hardware depends on two major  
 820 factors: the percentage of the execution time taken in software by an operation offloaded  
 821 to hardware, and the speed-up for the offloaded operation itself. In order to maximize  
 822 the first factor, we gave priority to operations that take the largest percentage of the  
 823 execution time, preferably more than 90%. These operations may involve a single function  
 824 call, several adjacent function calls, or a sequence of consecutive instructions in C. It  
 825 is preferred that a given operation is executed only once, or only a few times, as each  
 826 transfer of control and data between software and hardware involves a certain fixed timing  
 827 overhead, independent of the size of input and output to the accelerator. In order to  
 828 maximize the second factor, we gave priority to operations that have a high potential for  
 829 parallelization in hardware, and a small total size of inputs and outputs (which will need  
 830 to be transferred to and from the hardware accelerator, respectively)

831 Most of the data required to make informed decisions regarding software/hardware par-  
 832 titioning can be obtained by profiling software implementations, possibly extended with

833 some small modifications required to gather all relevant data. However, determining the  
834 potential for parallelization requires some knowledge of hardware or at least basic concepts  
835 of concurrent computing.

836 To assure fairness in our comparison, we offloaded to hardware all operations common to  
837 or similar across the implemented algorithms (e.g., all polynomial multiplications), and  
838 all operations that contributed significantly to the total execution time. Nevertheless, it  
839 should be understood that this heuristic procedure may need to be repeated several times  
840 because, after each round of offloading to hardware, different software operations may  
841 emerge as taking the majority of the total execution time. This process can stop when the  
842 development effort required for offloading the next most-critical operation to hardware is  
843 disproportionately high compared to the projected speed-up.

844 All encapsulations involve a single call to the function `randombytes()`, returning a seed to  
845 a pseudorandom number generator. This function could be possibly offloaded to hardware  
846 by implementing a True Random Number Generator (TRNG) in Programmable Logic.  
847 However, the correct implementation of a TRNG in FPGA fabric is a substantial project  
848 by itself. Additionally, in some cases, the seed would need to be transferred back to  
849 software, while in others it could be used directly by a hardware accelerator. To avoid  
850 these additional complications, in all the current software/hardware partitioning schemes,  
851 the seed is assumed to be generated in software.

## 852 4.7 RTL Design Methodology

853 The design of a hardware accelerator follows a traditional Register-Transfer Level (RTL)  
854 methodology. The entire system is divided into the Datapath and Controller. The Datapath  
855 is described using a hierarchical block diagram, and the Controller using hierarchical  
856 algorithmic state machine (ASM) charts. Multiple local controllers may be advantageous  
857 compared to a single global Controller. The RTL approach, although not novel by itself, is  
858 an important part of our methodology as it facilitates very efficient hardware accelerator  
859 designs. The block diagrams and ASM charts are very easy to translate to efficient and  
860 fully synthesizable VHDL code.

## 861 4.8 Potential Software Optimizations

862 Our software/hardware implementations could be potentially sped up by accelerating the  
863 remaining software part using assembly language programming.

864 The ARM Cortex-A53 is a microarchitecture implementing the ARMv8-A 64-bit instruction  
865 set. This instruction set consists of traditional RISC instructions operating on 31 general-  
866 purpose 64-bit registers, as well as Single-Instruction Multiple-Data (SIMD) instructions  
867 operating on 32 128-bit registers, treated as vectors composed of smaller data words. The  
868 SIMD architecture extension for the ARM Cortex-A processors, including Cortex-A53,  
869 is referred to as NEON [7]. The NEON 128-bit registers are considered as vectors of  
870 elements of the same data type, with NEON instructions operating on multiple elements  
871 simultaneously. Multiple data types are supported by this technology, including floating-  
872 point and integer operations.

873 A programmer can take advantage of NEON instructions using any of the following  
874 methods: a) Auto-vectorization by a compiler, b) Use of NEON-enabled libraries, c) using  
875 NEON intrinsics, and d) Using hand-coded NEON assembly language code [7].

876 Auto-vectorization is the process by which a compiler automatically analyzes the code and  
877 identifies opportunities to optimize performance using NEON instructions and register files.  
878 NEON-enabled libraries focus on signal and image processing, computer vision, physics,  
879 and machine learning. Examples include Arm Compute Library, Ne10, Libyuv, and Skia.  
880 NEON intrinsics are function calls that the compiler replaces with an appropriate NEON  
881 instruction or sequence of NEON instructions. Intrinsics provide almost as much control



882 as writing assembly language but leave the allocation of registers to the compiler [6].  
883 Finally, using hand-coded NEON assembly language code provides the programmer with  
884 the highest level of control, which may be used for the most aggressive and advanced  
885 optimizations.

886 Our implementations take advantage of only Option a) Auto-vectorization by a compiler.  
887 Regarding Option b), we are not aware of any NEON-enabled library that explicitly  
888 benefits lattice-based cryptosystems. No attempt was made to optimize software parts of  
889 our implementations using either intrinsics or hand-coded NEON assembly.

890 Our justification for these choices is as follows. Operations that are most suitable for a  
891 speed-up using NEON instructions are typically also excellent candidates for offloading  
892 to hardware. As a result, at the time when further offloading to hardware is judged to  
893 be either counterproductive or too labor intensive, the remaining operations executed  
894 in software are most likely sequential in nature and cannot take advantage of NEON  
895 instructions and registers.

896 Still, some speed up could be potentially accomplished by hand-coding these operations  
897 using scalar instructions of the ARMv8-A 64-bit instruction set. However, doing that would  
898 make the implementation much less portable. A similar effort could possibly be better  
899 spent on offloading all remaining operations to hardware. Even if the further offloaded  
900 operations cannot by themselves benefit from any substantial speed-up, moving them to  
901 hardware will eventually eliminate the entire transfer time, which remains substantial in  
902 all our software/hardware co-designs.

903 As a result, optimizing software implementations using NEON instructions and registers  
904 should be treated as an alternative optimization path, starting from the same starting point  
905 as our software/hardware co-designs. This starting point is a portable C implementation,  
906 that can be easily profiled and analyzed for inherent parallelism.

## 907 4.9 Verification and Generation of Results

908 Functional verification of the hardware description language (HDL) code is performed by  
909 comparing simulation results with precomputed outputs generated by a reference software  
910 implementation.

911 Fully verified and independently optimized VHDL code is then combined with the *optimized*  
912 software implementation of a given PQC candidate. Functional verification of the integrated  
913 software/hardware design is performed by running the code on the prototyping board  
914 and comparing the obtained outputs with outputs generated by a functionally equivalent  
915 reference implementation, run on the same ARM Cortex-A53 processor.

916 Experimental timing measurements follow, with the hardware accelerator's clock set (using  
917 the Clocking wizard) to the optimal target frequency identified during the synthesis and  
918 implementation runs. The execution time is measured by using the AXI Timer module,  
919 shown in Fig. 1, in clock cycles of the AXI Timer, which operates at the default clock  
920 frequency of 200 MHz.

921 The encapsulation time does not include the time necessary to transfer public key to the  
922 hardware accelerator. Similarly, the decapsulation time does not include the time necessary  
923 to transfer private key to the hardware accelerator. In case a public key of the receiver  
924 is required during decapsulation, this key is assumed to be a part of the corresponding  
925 private key.

926 The time required for a key upload is calculated as a difference between the time necessary  
927 for transferring a concatenation of the key and the first input (e.g., the seed for encapsu-  
928 lation, or the ciphertext for decapsulation) minus the time required to transfer the first  
929 input itself. This convention is consistent with the fact that the transmission of the key  
930 does not need to be repeated if the same key is reused multiple times. At the same time,  
931 the key upload overhead is typically so small that it is not efficient to send the key well

932 before its first use. As a result, the key upload is assumed to be always combined with the  
 933 transmission of the first input.

## 934 5 Results

### 935 5.1 Results for Hardware Implementations

**Table 15:** Maximum frequency and resource utilization of hardware implementations on Artix-7.

Algorithm	Security Category: Parameter Set	Max. Freq.	LUT	FF	Slice	DSP	BR AM
Kyber	1: KYBER512	210	11,864	10,348	3,989	8	15.0
Kyber	2: KYBER768	210	11,884	10,380	3,984	8	15.0
Kyber	5: KYBER1024	210	12,183	12,441	4,511	8	15.0
LAC-v3a	1: LAC-128	185	23,314	15,950	7,099	0	8.5
LAC-v3a	3: LAC-192	172	38,898	26,174	11,700	0	11.5
LAC-v3a	5: LAC-256	167	42,721	26,872	12,903	0	11.5
LAC-v3b	1: LAC-128	192	18,955	15,958	5,421	0	8.5
LAC-v3b	3: LAC-192	190	28,362	26,182	7,949	0	11.5
LAC-v3b	5: LAC-256	167	32,184	26,882	8,995	0	11.5
NewHope	1: NEWHOPE512-CCA-KEM	225	9,000	8,732	3,194	4	12.0
NewHope	5: NEWHOPE1024-CCA-KEM	225	9,000	8,732	3,194	4	12.0
Round5	1: R5ND_CCA_1KEM_0d	185	57,137	80,676	21,291	0	3.0
Round5	3: R5ND_CCA_3KEM_0d	165	78,825	107,564	29,441	0	3.0
Round5	5: R5ND_CCA_5KEM_0d			Doesn't fit			
Round5	1: R5ND_CCA_1KEM_5d	204	36,578	56,355	14,042	0	3.0
Round5	3: R5ND_CCA_3KEM_5d	174	59,852	95,170	24,869	0	3.0
Round5	5: R5ND_CCA_5KEM_5d	169	69,548	113,913	28,286	0	3.0

**Table 16:** Maximum frequency and resource utilization of hardware implementations on Virtex-7.

Algorithm	Security Category: Parameter Set	Max. Freq.	LUT	FF	Slice	DSP	BR AM
Kyber	1: KYBER512	245	13,745	11,107	4,590	8	14.0
Kyber	3: KYBER768	245	13,889	11,113	4,500	8	14.0
Kyber	5: KYBER1024	245	14,163	13,179	5,172	8	14.0
LAC-v3a	1: LAC-128	286	24,452	16,097	7,320	0	8.5
LAC-v3a	3: LAC-192	250	39,220	26,325	12,021	0	11.5
LAC-v3a	5: LAC-256	208	44,722	27,033	13,659	0	11.5
LAC-v3b	1: LAC-128	294	18,972	16,061	5,300	0	8.5
LAC-v3b	3: LAC-192	286	28,344	26,206	7,654	0	11.5
LAC-v3b	5: LAC-256	213	32,177	26,846	9,111	0	11.5
NewHope	1: NEWHOPE512-CCA-KEM	295	11,345	8,838	3,572	4	12.0
NewHope	5: NEWHOPE1024-CCA-KEM	295	11,345	8,838	3,572	4	12.0
Round5	1: R5ND_CCA_1KEM_0d	238	62,407	80,726	23,918	0	3.0
Round5	3: R5ND_CCA_3KEM_0d	208	78,727	107,631	28,034	0	3.0
Round5	5: R5ND_CCA_5KEM_0d	215	108,472	156,532	39,008	0	3.0
Round5	1: R5ND_CCA_1KEM_5d	256	38,350	56,413	14,731	0	3.0
Round5	3: R5ND_CCA_3KEM_5d	222	59,824	95,270	23,505	0	3.0
Round5	5: R5ND_CCA_5KEM_5d	202	69,561	113,933	28,643	0	3.0

936 Six CCA-secure KEMs representing four candidate - CRYSTALS-Kyber, LAC, NewHope,  
 937 and Round5 - have been implemented in pure hardware. LAC is represented by two variants,  
 938 v3a with  $q=251$  and v3b with  $q=256$ . Round5 is represented by R5ND\_CCA\_KEM\_0d

939 - a ring variant without any error correcting code, and R5ND\_CCA\_KEM\_5d - a ring  
 940 variant with the XE5 forward error correcting code used to decrease decryption failure  
 941 rates during decapsulation (and thus improve bandwidth and security).

942 The maximum clock frequency and resource utilization of our hardware implementations  
 943 of all six KEMs are summarized in Table 15 for Xilinx Artix-7 FPGAs and in Table 16  
 944 for Xilinx Virtex-7 FPGAs. All but one KEM fit within the largest device of the Artix-7  
 945 family. The only one that does not is the security level 5 variant of Round5 without error  
 946 correction. Taking into account that we target high-speed implementations, more suitable  
 947 for high-performance FPGAs, such as Virtex-7, the inability to fit the high-speed version of  
 948 the highest-security variant in low-cost FPGA family should not be used against Round5.  
 949 LAC-v3b, with  $q=256$ , clearly outperforms LAC-v3a, with  $q=251$ , in terms of both the  
 950 maximum clock frequency and resource utilization. For example, for the security level 1 on  
 951 Artix-7, the implementation of LAC-v3b has about 4% higher frequency and requires about  
 952 19% fewer LUTs. In the case of Round5, R5ND\_CCA\_KEM\_5d (with error correction)  
 953 significantly outperforms R5ND\_CCA\_KEM\_0d (without error correction). For example,  
 954 at the security level 1 on Artix-7, the difference is at the level of 10% in terms of clock  
 955 frequency, and 36% in terms of the number of LUTs.

956 Taking into account the best variants of all four submissions at the security level 1, all  
 957 clock frequencies are in a very small range between 192 and 210 MHz for Artix-7 and  
 958 between 235 and 294 for Virtex-7. Thus, no significant advantage in terms of the maximum  
 959 clock frequency is demonstrated by any candidate.

960 Ranking of candidates in terms of resource utilization is also very difficult because of  
 961 no clear equivalence between various elements of the resource utilization vectors. For  
 962 example, on Artix-7, NEWHOPE512-CCA-KEM uses about 4 times fewer LUTs than  
 963 R5ND\_CCA\_1KEM\_5d, but requires 4 vs. 0 DSP units, and 4 times more BRAMs.  
 964 Thus, none of these implementations can be claimed to be clearly superior vs. the other.

965 However, an important differentiating factor is the use of either similar or significantly  
 966 different amount of resources for implementing different security levels. It is generally  
 967 more desirable to have an algorithm that can be implemented using the same amount of  
 968 resources, independently of the security level. This feature allows an easier upgrade of a  
 969 security level. It also indirectly implies that the 3-in-1 or 2-in-1 designs will have a similar  
 970 resource utilization as the lowest-security variant rather than the resource utilization higher  
 971 than that of the highest-security variant. Out of six investigated KEMs, this desirable  
 972 property is exhibited only by Kyber and NewHope. On top of that, Kyber is slightly  
 973 more flexible, due to the existence of a variant at the security level 3. On the other hand,  
 974 NewHope has a small advantage in terms of all elements of the resource utilization vector  
 975 (e.g., for level 1 at Artix-7, it uses 9000 vs. 11,864 LUTs, 8,732 vs. 10,348 FFs, 3,194 vs.  
 976 3,989 slices, 4 vs. 8 DSP units, and 12 vs. 15 BRAMs).

977 The times necessary to load a public key (required for encapsulation) and a secret (private)  
 978 key (required for decapsulation) are proportional to the size of the respective key and  
 979 inversely proportional to the maximum clock frequency of a given PQC unit. All transfers  
 980 are assumed to be conducted using a 64-bit infifo\_data bus. The sizes of keys for all  
 981 variants of all investigated algorithms are summarized in Table 14. The maximum clock  
 982 frequencies are listed in Table 15 for Artix-7 and Table 16 for Virtex-7. In Fig. 4, we  
 983 compare these key loading times for Artix-7, and in Fig. 5 for Virtex-7. For both Artix-7  
 984 and Virtex-7, R5ND\_CCA\_KEM\_5d has the shortest key-loading times, and NewHope  
 985 the longest. However, the differences among these times are relatively minor. They do not  
 986 exceed a factor of 2 for loading a public key, and 3 for loading a private key.

987 The ranking of all 6 implemented KEMs in terms of the two primary performance metrics,  
 988 for high-speed implementations, is shown in Fig. 6 for Artix-7 and in Fig. 7 for Virtex-7.  
 989 The exact results and relative differences among the candidates are also summarized in  
 990 Tables 17 and 18. The primary metrics used for ranking are the execution times for

**Table 17:** Ranking of hardware implementations in terms of the execution time for encapsulation. For each algorithm, the first number represent the execution time in  $\mu\text{s}$ ; the second number is the ratio of the execution time for a given algorithm and the best execution time in the given ranking.

Artix-7								
Level 1			Level 3			Level 5		
<b>Round5_5d</b>	12.2	1.00	<b>Kyber</b>	19.9	1.00	<b>Round5_5d</b>	27.6	1.00
<b>Kyber</b>	14.8	1.21	<b>LAC-v3b</b>	21.2	1.07	<b>LAC-v3b</b>	28.1	1.02
<b>LAC-v3b</b>	14.8	1.21	<b>Round5_5d</b>	21.6	1.09	<b>Kyber</b>	28.4	1.03
Round5_0d	16.0	1.31	Round5_0d	25.6	1.29	<b>NewHope</b>	30.3	1.10
<b>NewHope</b>	16.3	1.34	LAC-v3a	29.1	1.46	LAC-v3a	33.9	1.23
LAC-v3a	17.9	1.47						
Virtex-7								
Level 1			Level 3			Level 5		
<b>LAC-v3b</b>	9.6	1.00	<b>LAC-v3b</b>	14.1	1.00	<b>LAC-v3b</b>	22.1	1.00
<b>Round5_5d</b>	9.7	1.01	<b>Round5_5d</b>	16.9	1.20	<b>Round5_5d</b>	23.0	1.04
LAC-v3a	11.5	1.20	<b>Kyber</b>	17.1	1.21	<b>NewHope</b>	23.1	1.05
<b>NewHope</b>	12.4	1.29	LAC-v3a	20.1	1.43	<b>Kyber</b>	24.3	1.10
Round5_0d	12.5	1.30	Round5_0d	20.3	1.44	Round5_0d	26.9	1.22
<b>Kyber</b>	12.6	1.31				LAC-v3a	27.2	1.23

**Table 18:** Ranking of hardware implementations in terms of the execution time for decapsulation. For each algorithm, the first number represent the execution time in  $\mu\text{s}$ ; the second number is the ratio of the execution time for a given algorithm and the best execution time in the given ranking.

Artix-7								
Level 1			Level 3			Level 5		
<b>Round5_5d</b>	16.3	1.00	<b>Kyber</b>	27.2	1.00	<b>Kyber</b>	36.2	1.00
<b>LAC-v3b</b>	18.9	1.16	<b>Round5_5d</b>	28.4	1.04	<b>Round5_5d</b>	36.4	1.01
Round5_0d	20.6	1.26	<b>LAC-v3b</b>	28.7	1.06	<b>LAC-v3b</b>	37.9	1.05
<b>Kyber</b>	21.4	1.31	Round5_0d	33.2	1.22	<b>NewHope</b>	41.5	1.15
<b>NewHope</b>	22.0	1.35	LAC-v3a	37.4	1.38	LAC-v3a	43.8	1.21
LAC-v3a	22.2	1.36						
Virtex-7								
Level 1			Level 3			Level 5		
<b>LAC-v3b</b>	12.4	1.00	<b>LAC-v3b</b>	19.1	1.00	<b>Kyber</b>	31.0	1.00
<b>Round5_5d</b>	13.3	1.07	<b>Round5_5d</b>	22.7	1.19	<b>Round5_5d</b>	31.2	1.01
LAC-v3a	14.4	1.16	<b>Kyber</b>	23.3	1.22	<b>NewHope</b>	31.7	1.02
Round5_0d	16.0	1.29	LAC-v3a	25.8	1.35	Round5_0d	35.8	1.15
<b>NewHope</b>	16.8	1.35	Round5_0d	27.0	1.41	<b>LAC-v3b</b>	37.7	1.22
<b>Kyber</b>	18.3	1.48				LAC-v3a	43.5	1.40

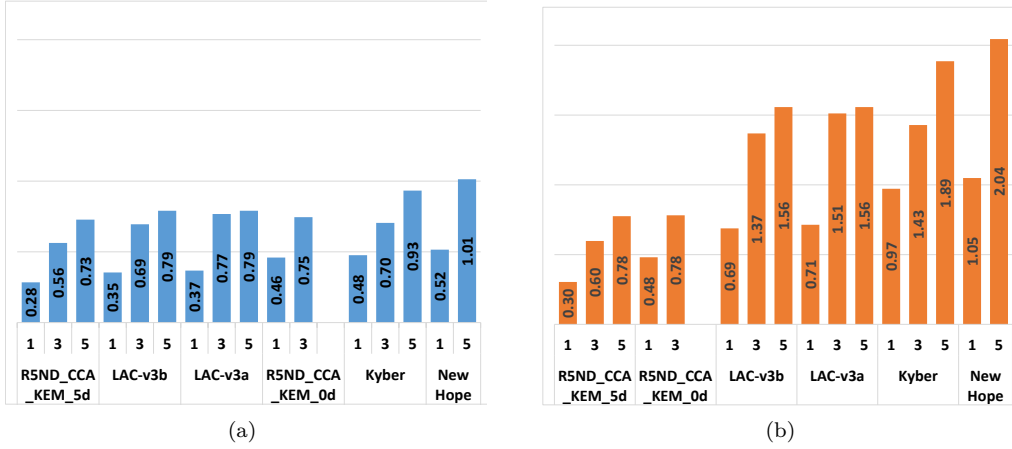


Figure 4: (a) Public Key and (b) Private Key transfer latency ( $\mu s$ ) on Artix-7

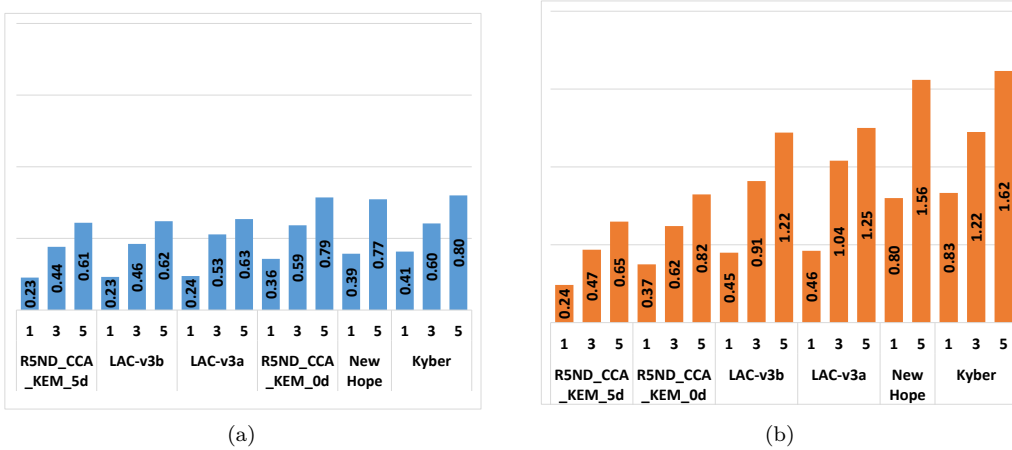
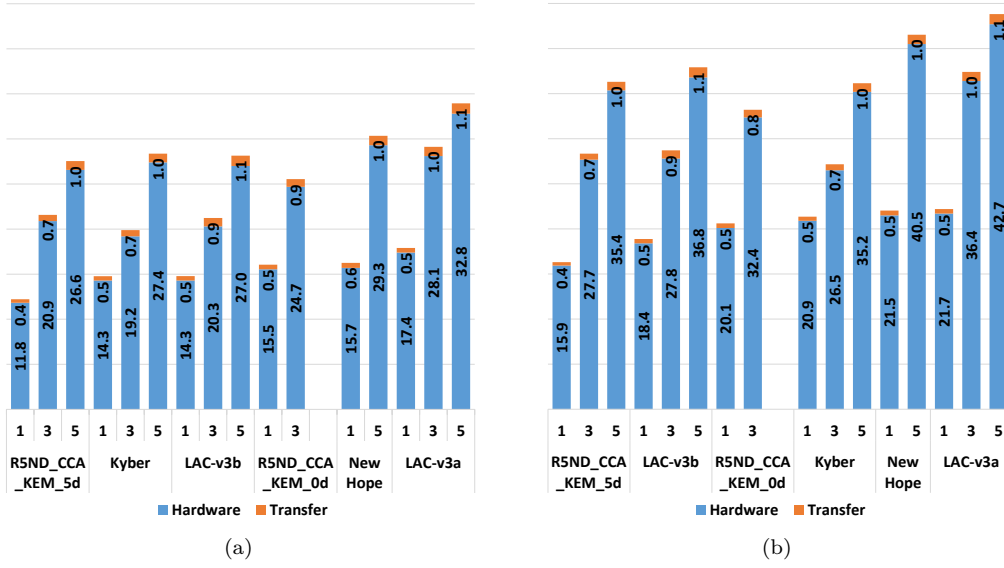


Figure 5: (a) Public Key and (b) Private Key transfer latency ( $\mu s$ ) on Virtex-7

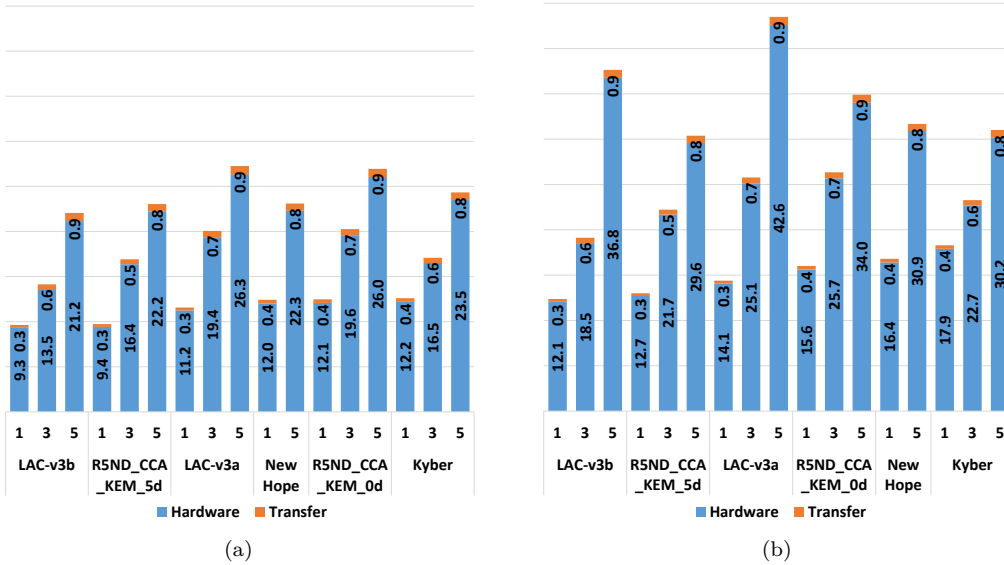
991 encapsulation and decapsulation, respectively.

992 LAC-v3b, with  $q=256$ , clearly outperforms LAC-v3a, with  $q=251$ , in terms of both  
 993 encapsulation and decapsulation time. Similarly, R5ND\_CCA\_KEM\_5d (with error  
 994 correction) outperforms R5ND\_CCA\_KEM\_0d (without error correction). The relative  
 995 ranking of Kyber, LAC-v3b, NewHope, and R5ND\_CCA\_KEM\_5d, changes depending  
 996 on the operation, security level, and FPGA family, but overall differences are minuscule.  
 997 Thus, none of these four algorithms has a clear edge over the other in terms of hardware  
 998 efficiency. Overall, the most efficient variants of all four candidates are in a virtual tie with  
 999 one another.

1000 In Table 19, we compare our hardware implementation of NewHope with the best high-speed  
 1001 implementation of this algorithm available to date. This implementation was described  
 1002 in [81], but it covered only a subset of the functionality of the IND-CCA KEM, namely  
 1003 the IND-CPA secure public-key encryption (PKE). Since for our own implementation,  
 1004 we could generate results for any subset of the complete CCA KEM design and using an  
 1005 arbitrary platform, the presented comparison is as fair as possible. Both sets of results  
 1006 concern exactly the same functionality, implemented using the same optimization target,  
 1007 with results generated using exactly the same platform.



**Figure 6:** Execution Time for (a) Encapsulation and (b) Decapsulation ( $\mu s$ ) on Artix-7



**Figure 7:** Execution Time for (a) Encapsulation and (b) Decapsulation ( $\mu s$ ) on Virtex7

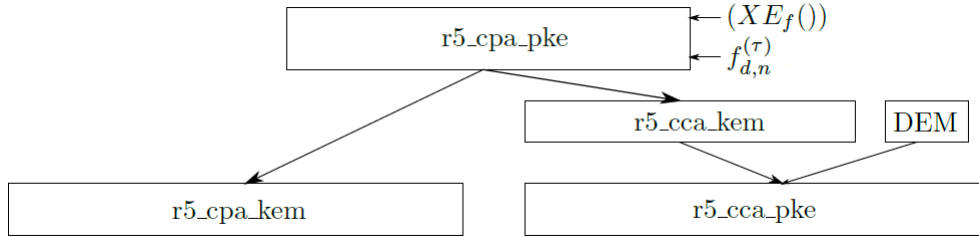
1008 Our implementation outperforms the design by Zhang et al. [81] in terms of all execution  
 1009 times. At security level 1, the speed-up varies between 2.2 for decryption, through 2.4 for  
 1010 key generation, to 2.6 for encryption. Similarly, at the security level 5, the speed-up varies  
 1011 between 2.0 for decryption, through 2.4 for key generation, to 2.6 for encryption. The  
 1012 penalty paid for this increase in speed is the increase in the number of LUTs by 33%, in  
 1013 flip-flops by a factor of 2.2, doubling the number of DSP units from 2 to 4, and increasing  
 1014 the number of BRAMs from 7-8 to 12. Overall, taking into the optimization for high-speed,  
 1015 our design is superior. However, the design by [81] provides an interesting example of  
 1016 trading the speed for a reduction in resource utilization.

**Table 19:** Comparison between this work and the best hardware design of NewHope reported in the literature to date. All results for Zynq-7000 SoC FPGA.

Design	Max. Freq. 200 225	Freq. Ratio	LUT 6,780 9,009	LUT Ratio	FF 4,026 8,768	FF Ratio	DSP 2 4	DSP Ratio	BR 7 12	BR AM	Key Generation			CPA Encryption			CPA Decryption		
											cycles	cycl. ratio	$\mu s$ ratio	cycles	cycl. ratio	$\mu s$ ratio	cycles	cycl. ratio	$\mu s$ ratio
<b>NewHope-512 CPA-PKE</b>																			
[81]	200	0.89	6,780	0.75	4,026	0.46	2	0.50	7	12	4,200	2.16	21.0	6,600	2.16	33.0	2,500	1.68	12.5
<b>TW</b>	225		9,009		8,768		4				1,946	8.6	8.6	3,061	2.44	13.6	1,487	1.68	6.6
<b>NewHope-1024 CPA-PKE</b>																			
[81]	200	0.89	6,781	0.75	4,127	0.47	2	0.50	8	12	8,000	2.14	40.0	12,500	2.18	62.5	4,800	1.66	24.0
<b>TW</b>	225		9,009		8,768		4				3,738	16.6	16.6	5,746	2.45	25.5	2,892	1.66	12.9

**Table 20:** Comparison of the execution times of major operations of the related CPA PKE and CCA KEM schemes when implemented in hardware using Artix-7. The ratio columns contain ratios of the execution times of Encapsulation/Encryption and Decapsulation/(Encryption+Decryption). All execution times are calculated without taking into account the time necessary to read inputs and offload outputs.

Algorithms	CPA-PKE				CCA-KEM					
	Encryption		Decryption		Encapsulation			Decapsulation		
	cycles	us	cycles	us	cycles	us	ratio	cycles	us	ratio
Kyber-512	2,602	12.4	1,608	7.7	2,995	14.3	1.15	4,395	20.9	1.04
Kyber-768	3,498	16.7	1,800	8.6	4,035	19.2	1.15	5,555	26.5	1.05
Kyber-1024	5,074	24.2	1,992	9.5	5,755	27.4	1.13	7,395	35.2	1.05
LAC-128-v3a	3,021	16.3	864	4.7	3,215	17.4	1.07	4,023	21.7	1.03
LAC-192-v3a	4,516	26.3	1,461	8.5	4,840	28.1	1.07	6,272	36.4	1.05
LAC-256-v3a	5,156	30.9	1,607	9.6	5,480	32.8	1.06	8,499	42.7	1.05
LAC-128-v3b	2,542	13.2	864	4.5	2,736	14.3	1.08	3,544	18.4	1.04
LAC-192-v3b	3,542	18.6	1,461	7.7	3,866	20.3	1.09	5,297	27.8	1.06
LAC-256-v3b	4,182	25.0	1,607	9.6	4,506	27.0	1.08	7,525	36.8	1.06
NewHope-512	3,061	13.6	1,487	6.6	3,538	15.7	1.16	4,829	21.5	1.06
NewHope-1024	5,746	25.5	2,892	12.9	6,583	29.3	1.15	9,111	40.5	1.05



**Figure 8:** Dependencies between CPA and CCA versions of Round5 proposals [73]

**Table 21:** Comparison between CPA-KEM, CCA-KEM and CCA-PKE variants of R5ND\_5d on Artix-7

Algorithm	Max. Freq.	Encaps./Encrypt.				Decaps./Decrypt.			
		cycles	ratio	us	ratio	cycles	ratio	us	ratio
CPA_1KEM	204	2,308	1.00	11.3	1.00	1,137	1.00	5.6	1.00
CCA_1KEM	204	2,492	1.08	12.2	1.08	3,328	2.93	16.3	2.93
CCA_1PKE	183	2,518	1.09	13.8	1.22	3,352	2.95	18.3	3.29
CPA_3KEM	169	3,582	1.00	21.2	1.00	1,726	1.00	10.2	1.00
CCA_3KEM	174	3,755	1.05	21.6	1.02	4,932	2.86	28.3	2.78
CCA_3PKE	163	3,782	1.06	23.2	1.09	4,956	2.87	30.4	2.98
CPA_5KEM	154	4,435	1.00	28.8	1.00	2,123	1.00	13.8	1.00
CCA_5KEM	169	4,655	1.05	27.5	0.96	6,137	2.89	36.3	2.63
CCA_5PKE	156	4,683	1.06	30.0	1.04	6,161	2.90	39.5	2.86

<sup>1017</sup> In Table 20, we compare the execution times of the CCA-KEM schemes with the execution  
<sup>1018</sup> times of the underlying CPA-PKE schemes, for Kyber, LAC, and NewHope. The ratios



1019 of the encapsulation and encryption times vary between to 1.07 and 1.17. That means  
 1020 that the PKE encryption is a dominant operation, and an overhead of other operations  
 1021 does not exceed 17%. For all four KEMs listed in this table, decapsulation includes one  
 1022 call to decryption and one call to encryption. Thus, the ratio listed under Decapsulation  
 1023 is a ratio of the execution time of decapsulation over the sum of the execution times of  
 1024 encryption and decryption. This ratio varies between 1.03 and 1.06, which means that the  
 1025 overhead of remaining operations does not exceed 6%.  
 1026 In Table 21, we compare our hardware implementations of different schemes of the Round5  
 1027 proposal. The dependencies among these schemes are graphically illustrated in Fig. 8. Our  
 1028 comparison contains results for maximum clock frequency and the execution times of major  
 1029 operations in the CPA-KEM, CCA-KEM, and CCA-PKE schemes. These metrics illustrate  
 1030 the cost of additional security of CCA-KEM as compared to CPA-KEM. The biggest  
 1031 difference is in the execution time of the CCA-KEM decapsulation and the CCA-PKE  
 1032 decryption, as compared to the CPA-KEM decapsulation. This difference comes from  
 1033 the Fujisaki-Okamoto transformation, used for providing CCA security. In CCA-KEM,  
 1034 during decapsulation, one additional CPA-PKE encryption is performed. CCA-KEM  
 1035 and CCA-PKE use the same parameter set, but CCA-PKE includes CCA-KEM and  
 1036 performs additional symmetric-key encryption after encapsulation of the key used for it.  
 1037 The differences in the execution times of the CPA-KEM and CCA-KEM encapsulations  
 1038 and the CCA-PKE encryption are negligible.

## 1039 5.2 Profiling the best available software implementations in C

**Table 22:** Source of software implementations

Algorithm	Software Source	Ref	Opt
FrodoKEM	<a href="https://github.com/Microsoft/PQCrypto-LWEKE">https://github.com/Microsoft/PQCrypto-LWEKE</a>	✓	✓
Kyber	<a href="https://github.com/pq-crystals/kyber">https://github.com/pq-crystals/kyber</a>	✓	
LAC	<a href="https://github.com/pqc-lac/lac-intel64">https://github.com/pqc-lac/lac-intel64</a>	✓	✓
NewHope	<a href="https://github.com/newhopecrypto/newhope">https://github.com/newhopecrypto/newhope</a>	✓	
NTRU	<a href="https://github.com/jschanck/ntru">https://github.com/jschanck/ntru</a>	✓	
NTRU Prime	<a href="https://bench.cr.yp.to/supercop.html">https://bench.cr.yp.to/supercop.html</a>	✓	✓
Round5	<a href="https://github.com/r5embed/r5embed">https://github.com/r5embed/r5embed</a>	✓	✓
Saber	<a href="https://github.com/KULeuven-COSIC/SABER">https://github.com/KULeuven-COSIC/SABER</a>	✓	

1040 We implemented 12 CCA-secure KEMs representing eight Round 2 lattice-based candi-  
 1041 dates using the software/hardware co-design approach described in detail in Section 4. In  
 1042 Table 22, we list the repositories containing C source code used as a starting point for our  
 1043 software/hardware implementations. In the case of four candidates - FrodoKEM, LAC,  
 1044 NTRU Prime, and Round5 - optimized implementations in C, different than reference  
 1045 implementations exist. For the remaining four candidates, their best portable implemen-  
 1046 tations are the same (or almost the same) as their reference implementations submitted  
 1047 at the beginning of Round 2. We used the mentioned above implementations in C as a  
 1048 starting point for our first software implementation of each of the 12 implemented KEMs,  
 1049 ported to ARM Cortex-A53 using the procedure described in Section 4.5.

1050 The results of profiling for the obtained pure-software implementations, running on a single  
 1051 core of ARM Cortex-A53, at the frequency of 1.2 GHz, are presented in the left portions  
 1052 of Tables 29, 30, 31, 32, 33, 34, 35, 36, 37, and 38, in Appendix A.

1053 For each of the 12 investigated algorithms and each major operation (Encapsulation and  
 1054 Decapsulation), two to five most time-consuming functions are identified. For each of these  
 1055 functions, we provide their execution time (in microseconds) and the percentage of the  
 1056 total execution time. In the right portions of the same tables, we list in bold functions  
 1057 offloaded to hardware. For the functions combined together, they are listed in the same

1058 field of the table, with sub-indices, such as 1.1, 1.2, 1.3, etc. A single execution time and a  
 1059 single percentage of the software/hardware execution time is given for such a combined  
 1060 function.

1061 It is important to note that the execution time of all functions offloaded to hardware, listed  
 1062 in Tables 29–38 include both the execution time in hardware as well as the time necessary  
 1063 to transfer control, inputs, and outputs between the processor and a hardware accelerator.  
 1064 It should also be mentioned that the number of functions offloaded to hardware may be  
 1065 misleading, as these functions may appear at different levels of hierarchy. For example, for  
 1066 the encapsulation in Kyber, only two functions are offloaded. However, these are function  
 1067 involving the majority of operations of Kyber, amounting to 99.55-99.81% of the total  
 1068 execution time in the software-only implementation. For all algorithms, at least the first  
 1069 and the second most time-consuming functions are offloaded to hardware.

1070 The total percentage of the execution time taken by a portable software implementation  
 1071 to execute operations offloaded to hardware is shown in Figs. 9 and 10.

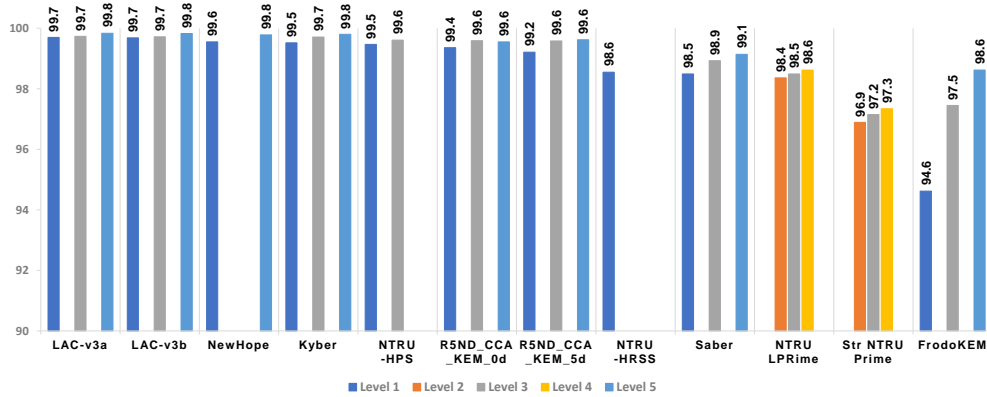


Figure 9: Encapsulation: The Software Part Sped Up by Hardware [%]

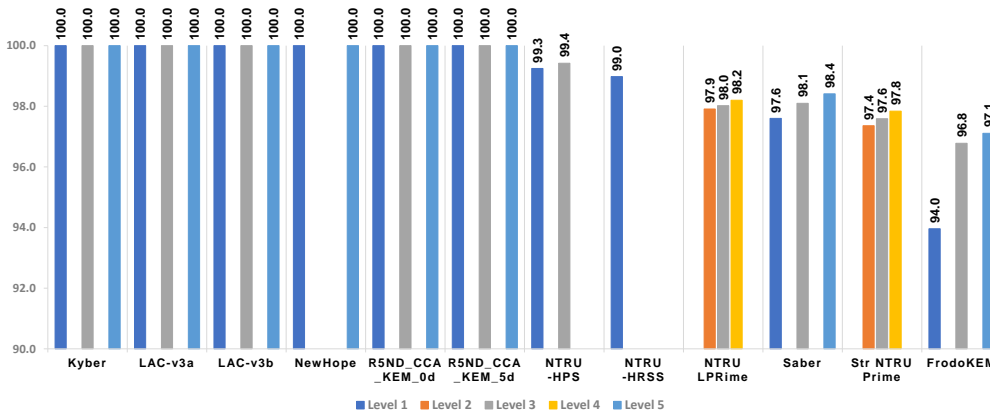


Figure 10: Decapsulation: The Software Part Sped Up by Hardware [%]

### 1072 5.3 Results for Software/Hardware Implementations

1073 Twelve hardware accelerators developed using the methodology described in Section 4 are  
 1074 characterized in Table 23 using their maximum clock frequency and resource utilization

**Table 23:** Maximum frequency and resource utilization of hardware accelerators developed as a part of software/hardware co-designs targeting Zynq Ultrascale+

Algorithm	Security Category: Parameter Set	Max. Freq.	LUT	FF	Slice	DSP	BR AM
FrodoKEM	Frodo-640	402	7,213	6,647	1,186	32	13.5
FrodoKEM	Frodo-976	402	7,087	6,693	1,190	32	17.0
FrodoKEM	Frodo-1344	417	7,015	6,610	1,215	32	17.5
Kyber	KYBER512	410	12,034	10,532	2,327	8	14.0
Kyber	KYBER768	405	12,195	10,461	2,253	8	14.0
Kyber	KYBER1024	405	12,589	12,574	2,635	8	14.0
LAC-v3a	LAC-128	385	25,123	16,005	3,720	0	8.5
LAC-v3a	LAC-192	370	41,898	26,233	6,134	0	11.5
LAC-v3a	LAC-256	357	46,756	26,989	6,774	0	11.5
LAC-v3b	LAC-128	400	18,311	15,966	2,672	0	8.5
LAC-v3b	LAC-192	385	27,209	26,193	4,024	0	11.5
LAC-v3b	LAC-256	357	33,234	26,567	4,889	0	11.5
NewHope	NEWHOPE512-CCA-KEM	490	9,307	8,928	1,721	4	11.0
NewHope	NEWHOPE1024-CCA-KEM	490	9,307	8,928	1,721	4	11.0
NTRU-HPS	ntruhs2048677	200	42,578	22,717	8,235	677	8.5
NTRU-HPS	ntruhs4096821	200	49,735	30,599	9,924	821	8.5
NTRU-HRSS	ntruhss701	200	48,773	25,178	8,110	701	2.5
NTRU LPRime	kem/ntrulpr653	278	45,901	39,426	8,938	0	8.0
NTRU LPRime	kem/ntrulpr761	263	55,054	45,133	9,769	0	8.0
NTRU LPRime	kem/ntrulpr857	250	64,022	50,120	10,554	0	8.0
Str NTRU Prime	kem/sntrup653	278	62,797	33,531	9,110	0	9.0
Str NTRU Prime	kem/sntrup761	263	70,066	38,144	10,319	0	9.0
Str NTRU Prime	kem/sntrup857	250	78,379	42,274	11,509	0	9.0
Round5	R5ND_CCA_1KEM_0d	294	52,589	80,875	10,154	0	3.0
Round5	R5ND_CCA_3KEM_0d	267	72,870	107,748	13,360	0	3.0
Round5	R5ND_CCA_5KEM_0d	250	99,310	156,732	18,095	0	3.0
Round5	R5ND_CCA_1KEM_5d	357	38,116	56,189	7,538	0	3.0
Round5	R5ND_CCA_3KEM_5d	294	54,532	95,436	12,395	0	3.0
Round5	R5ND_CCA_5KEM_5d	238	69,254	114,007	12,774	0	3.0
Saber	LightSaber-KEM	322	12,343	11,288	1,989	256	3.5
Saber	Saber-KEM	322	12,566	11,619	1,993	256	3.5
Saber	FireSaber-KEM	322	12,555	11,881	2,341	256	3.5

1075 when implemented on Xilinx Zynq UltraScale+ SoC FPGA. All results have been obtained  
1076 after placing and routing.

1077 NewHope, Kyber, and FrodoKEM are able to achieve the highest clock frequencies, above  
1078 400 MHz for all parameter sets. LAC has frequencies between 350 and 400 MHz, depending  
1079 on a variant and security level. The maximum frequency of Round5 decreases significantly  
1080 with the increase in the security level, especially for a version with the error-correcting  
1081 code, where the frequency drops from 357 MHz for security level 1 to 238 MHz for security  
1082 level 5. On the other hand, Saber has the same clock frequency, 322 MHz, for all of  
1083 its parameter sets. The operating frequencies for the two variants of NTRU Prime are  
1084 in the range 250-280. They are limited mainly by the reduction modulo  $q$ . To reduce  
1085 numbers with the prime modulus  $q$ , we selected the conditional subtraction method, which  
1086 is relatively simple but comes with a long critical path. NTRU-HPS and NTRU-HRSS  
1087 have the lowest clock frequency of 200 MHz. These frequencies are affected by the logic  
1088 for converting polynomials from  $R/q$  to  $S/q$  and from  $R/q$  to  $S/3$ .

1089 The accelerators for NTRU-HPS and NTRU-HRSS involve the highest number of integer  
1090 multiplications performed in parallel. These multiplications in the FPGA fabric are  
1091 delegated to dedicated DSP units. The DSP units are also taken advantage of in Saber and  
1092 to a lower extent in FrodoKEM, Kyber, and NewHope. LAC, Round5, NTRU LPRime,

1093 and Streamlined NTRU Prime do not involve any integer multiplications in hardware.  
1094 This is because the coefficients of one of the multiplied polynomials always belong to the  
1095 set  $\{-1, 0, 1\}$ .

1096 FrodoKEM is the algorithm with the highest utilization of BRAMs, which reaches 17.5  
1097 blocks. The algorithms with the lowest utilization of BRAMs (between 2.5 and 3.5) include  
1098 NTRU-HRSS, Round5, and Saber. The remaining KEMs require 8–14 BRAMs.

1099 Round5, Streamlined NTRU Prime, and NTRU LPrime use the largest number of LUT,  
1100 flip-flops (FFs), and Slices. FrodoKEM, NewHope, Kyber, and Saber use the smallest  
1101 number. The amount of resources used increases noticeably with the increase in the  
1102 security level for 8 out of 12 KEMs. The following algorithms have a desirable property  
1103 that the security levels do not substantially affect resource utilization (except for the small  
1104 increase in the number of BRAMs in FrodoKEM): FrodoKEM, Kyber, NewHope, and  
1105 Saber.

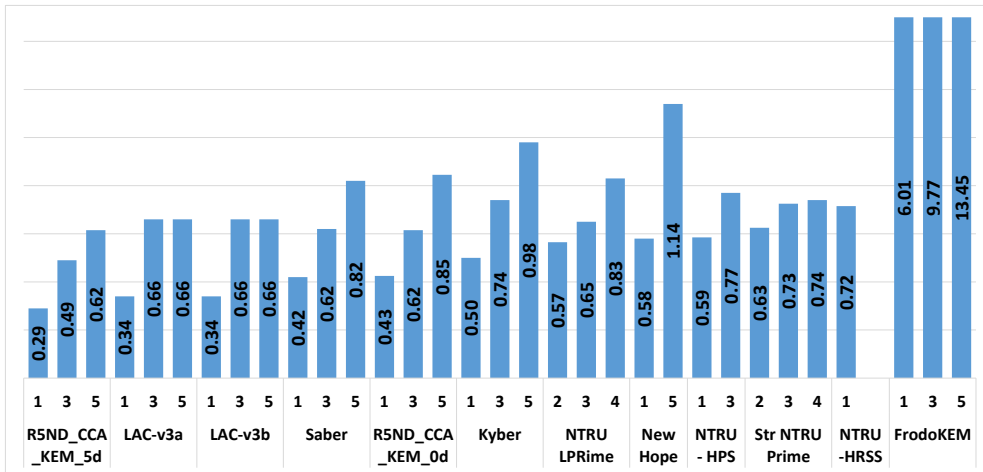
1106 Because of the timing dependencies, and in particular, the bottleneck caused by SHAKE,  
1107 our implementation of FrodoKEM cannot be easily sped up by trading additional resources  
1108 for speed. This example clearly illustrates the potential algorithmic limits on the amount  
1109 of parallelization (and thus the maximum speed-up), which is independent of the amount  
1110 of hardware resources available to the designer.

1111 The times necessary to load a public key (required for encapsulation) and a secret (private)  
1112 key (required for decapsulation) are proportional to the size of the respective key and  
1113 inversely proportional to the maximum clock frequency of a given PQC unit. All transfers  
1114 are assumed to be conducted using a 64-bit `infifo_data` bus. The sizes of keys for all  
1115 variants of all investigated algorithms are summarized in Table 14. The maximum clock  
1116 frequencies are listed in Table 23. In Fig. 11, we compare these key loading times for  
1117 all 12 implemented KEMs. R5ND\_CCA\_KEM\_5d has the shortest key-loading times,  
1118 and FrodoKEM the longest. However, the differences among these times are relatively  
1119 minor for all KEMs other than FrodoKEM. They do not exceed a factor of 2 for loading a  
1120 public key, and 4 for loading a private key. Except for NewHope at the security level 5  
1121 and FrodoKEM at all security levels, the public-key loading times stay below 1  $\mu$ s, and  
1122 private-key loading times below 2  $\mu$ s.

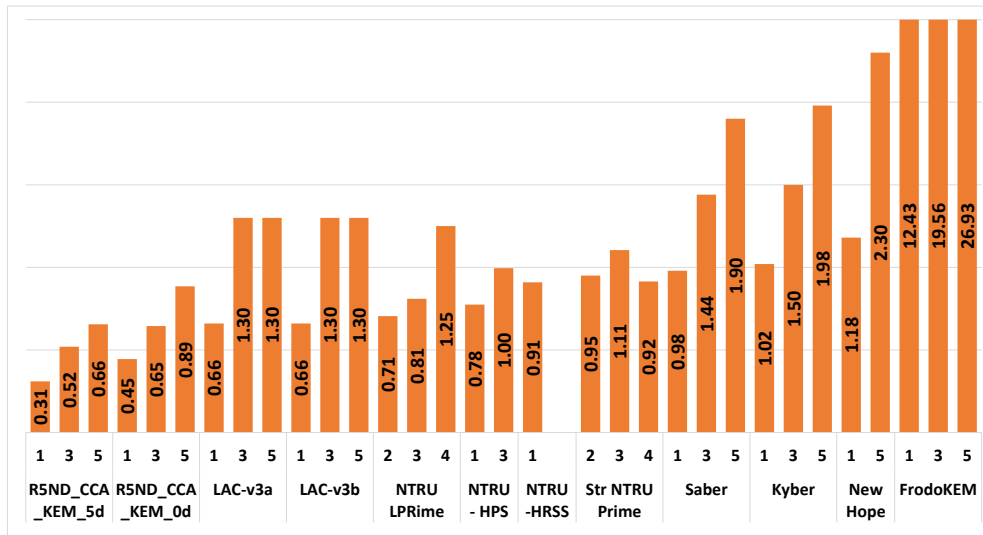
1123 Total execution times of our software/hardware implementations are summarized in Fig. 12  
1124 for encapsulation, and Fig. 13 for decapsulation.

1125 Rankings can be considered separately for three groups of parameter sets listed in Table 14,  
1126 with the security levels 1 and 2, 3 only, and 4 and 5, respectively. Only the first group  
1127 contains all 12 investigated algorithms. In the second group, NTRU-HRSS and NewHope  
1128 are missing, and in the third group, NTRU-HRSS and NTRU-HPS are not represented. In  
1129 Figs. 12 and 13, KEMs are arranged according to their ranking for security levels 1 and 2.

1130 Each execution time is separated into three components: the execution time in hardware  
1131 (i.e., in the hardware accelerator located in programmable logic of Zynq UltraScale+ SoC  
1132 FPGAs), the time required to transfer data and control between the processor and the  
1133 hardware accelerator, and the execution time in software (i.e., in ARM Cortex-A53). For  
1134 encapsulation, at least the function `randombytes()` is assumed to be executed in software  
1135 to generate a seed for a deterministic random bit generator (typically based on SHAKE)  
1136 implemented in hardware. For decapsulation, no internal function of KEM has to be  
1137 executed in software. We treat implementation as a software/hardware implementation  
1138 even if the operation of the processor is limited only to sending KEM inputs to and  
1139 receiving KEM outputs from the hardware accelerator. Hence, our software/hardware  
1140 implementations of LAC-v3b, R5ND\_CCA\_KEM\_5d, NewHope, Kyber, LAC-v3a, and  
1141 R5ND\_CCA\_KEM\_0d, which have the shortest execution times of encapsulation and  
1142 decapsulation are based on the pure hardware implementations of KEMs, described in  
1143 Section 5.1. The ranking of these six KEMs is similar, but not identical to the ranking  
1144 of their corresponding hardware implementations. The small changes in rankings come



(a)



(b)

**Figure 11:** (a) Public Key and (b) Private Key transfer latency ( $\mu s$ ) of SW/HW co-design on Zynq-Ultrascale+

1145 from small differences in the transfer time and execution time in software, as well as from  
 1146 the different maximum clock frequency of hardware accelerators when implemented in  
 1147 programmable logic of Zynq UltraScale+ rather than Artix-7 or Virtex-7 (as in Figs. 6  
 1148 and 7). Overall, however, not leaving any operation (other than `randombytes()`) in  
 1149 software gives these 6 KEMs enough advantage to outperform all six remaining schemes.

1150 FrodoKEM is by far the slowest KEM, and it cannot outperform any other scheme even if  
 1151 100% of its operations are moved to hardware. For encapsulation, NTRU-HPS, Streamlined  
 1152 NTRU Prime, and NTRU LPrime are also very unlikely to move in ranking ahead of any  
 1153 of the first six schemes, because even after reducing their execution time in software to  
 1154 zero and making the transfer time similar to the transfer time of the first six schemes (i.e.,  
 1155 in the range of 6.2-7.0  $\mu s$ ), their execution times would exceed the overall time for the  
 1156 KEM at position 6, R5ND\_CCA\_KEM\_0d.

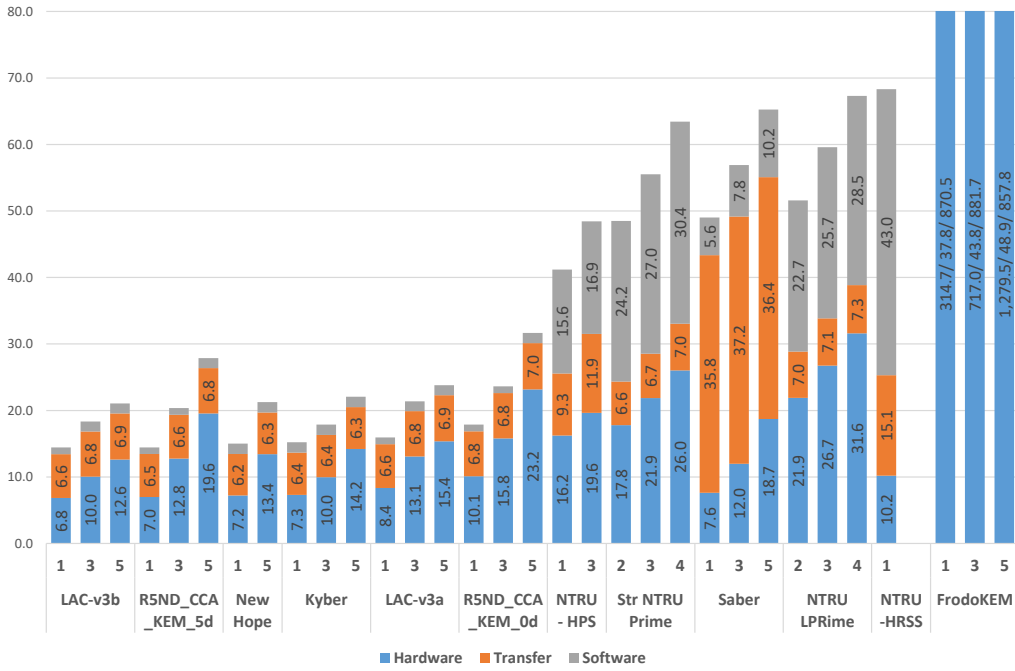


Figure 12: Encapsulation: Total Execution Time in Software/Hardware [μs]

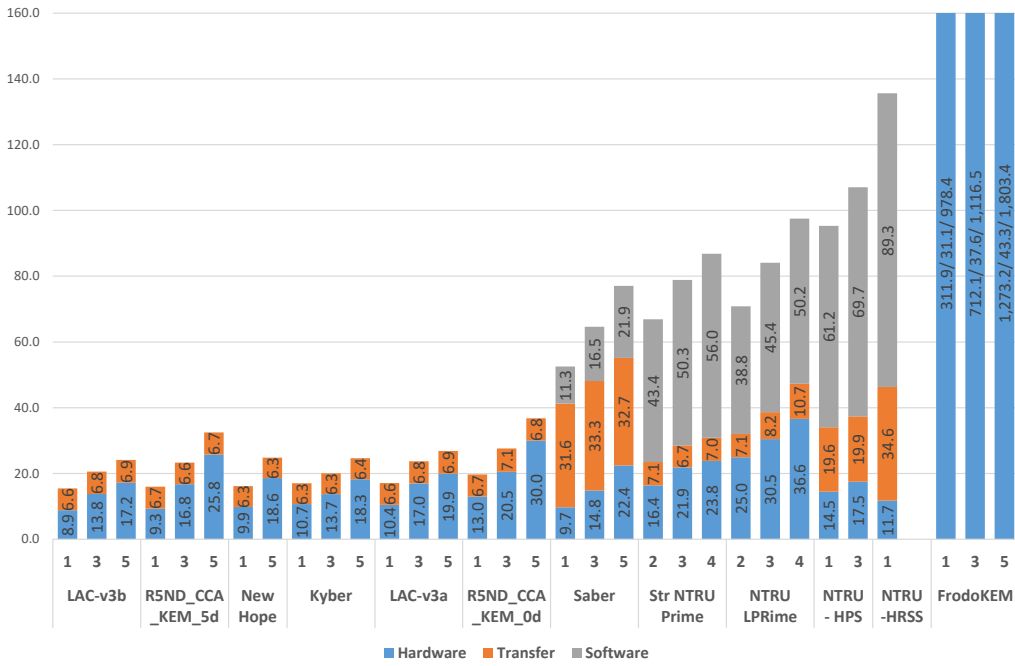


Figure 13: Decapsulation: Total Execution Time in Software/Hardware [μs]

1157 For Saber and NTRU-HRSS, it is too early to make such a judgment. However, the  
 1158 presented results at least reveal some potential weaknesses of these two algorithms (from  
 1159 the point of view of ease of their software/hardware partitioning), which can be observed  
 1160 by analyzing their profiling results, summarized in Tables 38 and 34. For NTRU-HRSS,

1161 even after moving to hardware its four most time-consuming operations, the software still  
 1162 amounts to a significant percentage of the total execution time. In Saber, even after moving  
 1163 to hardware its five most time-consuming operations, transfer time still dominates the  
 1164 total execution time. This last point can be reinforced by analyzing Table 24. According  
 1165 to this table, our software/hardware implementation of Saber has the largest number of  
 1166 transfers between the processor and the accelerator (6). Disregarding FrodoKEM, which  
 1167 is very slow in hardware already, NTRU-HRSS is the only other algorithm that requires  
 1168 more than one transfer during the encapsulation.

**Table 24:** Data Transfer Summary for SW-HW co-designs

Algorithms	Encapsulation			Decapsulation		
	Count	Load (bytes)	Return (bytes)	Count	Load (bytes)	Return (bytes)
KYBER512	1	32	768	1	736	32
KYBER768	1	32	1,120	1	1,088	32
KYBER1024	1	32	1,600	1	1,568	32
R5ND_1KEM_0d	1	16	740	1	724	16
R5ND_3KEM_0d	1	24	1,103	1	1,079	24
R5ND_5KEM_0d	1	32	1,509	1	1,477	32
R5ND_1KEM_5d	1	16	620	1	604	16
R5ND_3KEM_5d	1	24	934	1	910	24
R5ND_5KEM_5d	1	32	1,285	1	1,253	32
LightSaber-KEM	6	1,600	704	5	1,216	896
Saber-KEM	6	2,272	960	5	1,216	1,280
FireSaber-KEM	6	2,976	1,216	5	1,600	1,664
Frodo-640	4	19,400	22,032	3	9,768	22,008
Frodo-976	4	31,472	31,328	3	15,816	31,304
Frodo-1344	4	42,780	43,136	3	21,728	43,104
LAC-128-v3a	1	16	736	1	704	32
LAC-192-v3a	1	32	1,384	1	1,352	32
LAC-256-v3a	1	32	1,496	1	1,464	32
LAC-128-v3b	1	16	736	1	704	32
LAC-192-v3b	1	32	1,384	1	1,352	32
LAC-256-v3b	1	32	1,496	1	1,464	32
NEWHOPE512	1	32	960	1	928	32
NEWHOPE1024	1	32	1,856	1	1,824	32
ntruhs2048677	1	720	1,372	2	2,318	1,408
ntruhs4096821	1	864	1,708	2	2,906	1,692
ntruhrss701	2	1,024	1,436	3	2,854	1,512
kem/sntrup653	1	40	1,448	1	912	72
kem/sntrup761	1	40	1,664	1	1,048	72
kem/sntrup857	1	40	1,856	1	1,192	72
kem/ntrulpr653	1	40	1,584	1	1,040	72
kem/ntrulpr761	1	40	1,800	1	1,176	72
kem/ntrulpr857	1	40	1,992	1	1,320	72

1169 For decapsulation, the execution time in software is eliminated entirely for the first six  
 1170 KEMs in the ranking. The transfer time is similar for all algorithms from that group.  
 1171 The transfer time dominates the execution time in Saber, because, as shown in Table 24,  
 1172 five transfers are required, more than for any other algorithm. The transfer time is also

1173 unusually long for NTRU-HRSS and NTRU-HPS (with 3 and 2 transfers, respectively). As  
1174 a result, it might be too early to judge whether Saber, NTRU-HRSS, and NTRU-HPS can  
1175 be made as efficient as the first six KEMs, after moving all their operations to hardware.  
1176 On the other hand, for NTRU LPrime and Streamlined NTRU Prime there is already a  
1177 strong indication that these algorithms will not be able to move ahead of any of the first  
1178 six KEMs, even if implemented entirely in hardware. Finally, FrodoKEM is by far the  
1179 slowest algorithm out of all 12 implemented in this study.

## 1180 5.4 Use of High-Level Synthesis

1181 A traditional approach to high-level synthesis is based on starting from the existing  
1182 implementation in C, C++, or System C, and then introducing modifications aimed at:

- 1183 • inferring the desired interface
- 1184 • optimizing speed
- 1185 • minimizing resource utilization.

1186 In the case of PQC candidates, a starting point is naturally determined by either reference  
1187 implementation or the best portable implementation written entirely in C, such as those  
1188 described in Section 4.5, used as a starting point for the software/hardware co-design.

1189 Traditionally, a significant percentage of all modifications amounts to guiding the synthesis  
1190 tool toward the desired outcome using the C language pragmas, ignored by traditional  
1191 high-level language compilers, but treated as directives by a given synthesis tool. This  
1192 approach is used, in particular, in two popular HLS tools targeting FPGAs, Vivado HLS  
1193 and LegUp. For example, there are over 20 *pragma* directives in the current version of  
1194 Vivado HLS. Their different combinations lead to different hardware architectures. The  
1195 impact of a particular *pragma* directive is heavily dependent on the code structure and the  
1196 algorithm. Some directives may have no impact at all; others may dramatically change  
1197 the speed vs. cost trade-off. Exploring all possible combinations is often unrealistic.  
1198 Additionally, in many cases, code refactoring may give better results than an optimal  
1199 choice and placement of directives.

1200 The first attempt at applying HLS to benchmarking PQC schemes was reported in [14].  
1201 Only a few directives aimed at accomplishing unrolling of loops and pipelining were applied.  
1202 The authors also attempted to synthesize the C code of the entire algorithm. Taking into  
1203 account the limited availability of RTL code, no comparison with the equivalent RTL code  
1204 was attempted.

1205 An outcome of this approach is quite clearly illustrated in Table 25. The differences in  
1206 obtained results are huge, although probably not that surprising, taking into account  
1207 the almost complete reliance on tools in [14]. The HLS-based designs are bigger than  
1208 RTL-based designs in terms of the number of LUTs by a factor of at least 144 for Kyber,  
1209 14.5 for NewHope, and 7.5 for Classic McEliece. These factors are obtained by dividing  
1210 the area of the decapsulation/decryption unit in the HLS-based approach by the area of  
1211 the combined unit, capable of performing key generation, encapsulation/encryption and  
1212 decapsulation/decryption in the RTL approach. Thus, if units with the same functionality  
1213 were compared, the ratios could be even higher. The HLS to RTL ratios of the encapsula-  
1214 tion/encryption times are 10.5 for Kyber, 93 for NewHope, and 700 for Classic McEliece.  
1215 For decapsulation/(decryption+encryption), the corresponding ratios are 36 for Kyber,  
1216 741 for NewHope, and 551 for Classic McEliece. Overall in terms of the latency times  
1217 area product, the HLS-based designs are three orders of magnitude worse. Additionally,  
1218 a significant difference in specific ratios for Kyber and NewHope, combined with the almost  
1219 identical performance and resource usage of RTL designs, indicates that the approach  
1220 pursued in [14] cannot correctly predict the relative ranking of PQC candidates unless the  
1221 differences among them are truly enormous.



**Table 25:** Comparison between HLS-based designs from [14] and the corresponding RTL-based designs. All results for Virtex-7 FPGAs. TW denotes this work.

Design	Max. Freq.	Freq. Ratio	LUT	LUT Ratio	FF	FF Ratio	DSP	BR AM	Key Generation			Encaps./Enc.			Decaps./Enc. (CPA)		
									cycles	$\mu s$	cycles	cycl. ratio	$\mu s$	ratio	cycles	cycl. ratio	$\mu s$
<b>Kyber-512 CCA-KEM</b>																	
[14]	67	0.27	1,307,815	95.15	11,699	1.05	-	-	-	31,669	10.57	475.0	43,018	9.79	645.3	35.97	
<b>TW</b>	245		1,977,896	143.90	194,126	17.48	8	14	2,160	8.8	2,995	12.2	4,395		17.9		
<b>NewHope-512 CCA-KEM</b>																	
[14]	67	0.23	136,457	12.03	25,639	2.90	0	0	-	307,847	87.01	4,617.7	721,986	149.51	10,829.8	661.58	
<b>TW</b>	295		164,937	14.54	28,999	3.28	4	10.0	2,122	7.2	3,538	12.0	4,829		16.4		
<b>mceliece6960119 CPA-PKE</b>																	
[14]	100	0.77	840,430	7.19	60,270	0.32	-	-	-	3,787,729	699.75	37,877.3	10,659,024	424.07	106,590.2	550.85	
<b>[75]</b>	130		870,908	7.45	79,962	0.42	0	607.0	974,306	7,500.4	5,413	41.7	25,135		193.5		

**Table 26:** Comparison between HLS and RTL method for NTT implementations.

Method	DSP	BR AM	LUT	FF	Slice	Max. Freq.	Latency (cycles)
NewHope-512 with 1 NTT module							
HLS	4	3	1,181	1,403	239	454	3,247
RTL	4	3	1,040	940	190	476	3,247
HLS/RTL	1.00	1.00	1.14	1.49	1.26	0.95	1.00
NewHope-1024 with 1 NTT module							
HLS	4	5	1,110	1,342	219	455	6,266
RTL	4	5	842	803	170	476	6,266
HLS/RTL	1.00	1.00	1.32	1.67	1.29	0.96	1.00
Kyber512 with 2 NTT modules							
HLS	24	7	2,325	2,346	430	455	1,271
RTL	24	5	2,040	3,223	433	500	1,271
HLS/RTL	1.00	1.40	1.14	0.73	0.99	0.91	1.00
Kyber768 with 3 NTT modules							
HLS	36	11	5,379	4,043	1,074	416	1,271
RTL	36	7.5	3,054	5,098	637	500	1,271
HLS/RTL	1.00	1.47	1.76	0.79	1.69	0.83	1.00
Kyber1024 with 4 NTT modules							
HLS	48	14	7,111	5,457	1,374	416	1,271
RTL	48	10	4,055	6,803	960	500	1,271
HLS/RTL	1.00	1.40	1.75	0.80	1.43	0.83	1.00

1222 As a result, a substantially different approach was needed to overcome this inefficiency.  
 1223 This approach was demonstrated by our group in [21], [23], [56], and [57]. First, HLS is  
 1224 combined with software/hardware co-design. This way, only the most time-consuming  
 1225 operations (and preferably a single operation) needs to be offloaded to hardware. These  
 1226 operations can be identified using techniques described in Section 4.6. Secondly, these  
 1227 critical operations are described using block diagrams. Third, the block diagrams are  
 1228 translated into HLS-ready C code, written from scratch, and enhanced with HLS directives  
 1229 encoded using pragmas. The designer then debugs the code using a C testbench, which is  
 1230 much easier to develop and easier to use than HDL testbench. When the code is determined  
 1231 to be functionally correct, it is passed through synthesis. If the number of clock cycles  
 1232 is different from the expected number obtained from the analysis of the block diagram,  
 1233 additional pragmas need to be added, or the code needs to be refactored (rewritten) to  
 1234 make it more suitable for HLS tools. For example, a programmer may apply explicit  
 1235 function sharing or eliminate dependencies preventing multiple operations from executing  
 1236 in parallel. These optimizations continue at least until the required number of clock cycles  
 1237 is reached. They may also be applied to reduce the number of specific logic resources, such  
 1238 as LUTs, DSP units, or BRAMs, as well as to increase the maximum clock frequency.  
 1239 Below, we demonstrate the application of this approach to the preliminary software/hardware  
 1240 implementations of two Round 2 candidates: NewHope and Kyber. In both implemen-  
 1241 tations, we decided to offload to hardware only the most time-consuming operation, the  
 1242 Number Theoretic Transform (NTT). This operation was first expressed using a detailed  
 1243 block diagram, presented in [57]. Then, the described above methodology was followed. In  
 1244 parallel, optimized RTL implementation was developed for the purpose of evaluating the  
 1245 quality of our HLS design. The obtained results are summarized in Table 26.  
 1246 These results indicate that our primary goal of reaching the same number of clock cycles  
 1247 as that obtained using the RTL approach was accomplished. At the same time, the clock  
 1248 frequency was lower by up to 17%, and the number of LUTs, flip-flops, and slices higher

1249 by up to 76%, 67%, and 69%, respectively.

1250 Overall, the RTL- and HLS-based approaches to the design of a hardware accelerator for  
 1251 NTT led to almost the same total speed-up of the software/hardware implementation.  
 1252 At the same time, the development time was several times shorter for the HLS-based  
 1253 approach.

1254 The disadvantage of our approach is the need for a detailed block diagram, which requires  
 1255 either hardware expertise within a team of HLS programmers or collaboration with a group  
 1256 of hardware designers. Additionally, most of the HLS-ready code needs to be written from  
 1257 scratch. The reduction in clock frequency plays a secondary role, as it typically does not  
 1258 significantly affect the overall speed-up of the software/hardware implementation over the  
 1259 portable C code. Similarly, for high-speed implementations, the exact resource utilization  
 1260 plays a secondary role and does not affect the ranking of candidates.

## 1261 5.5 Results for Software Implementations Optimized Using NEON In- 1262 structions of ARM

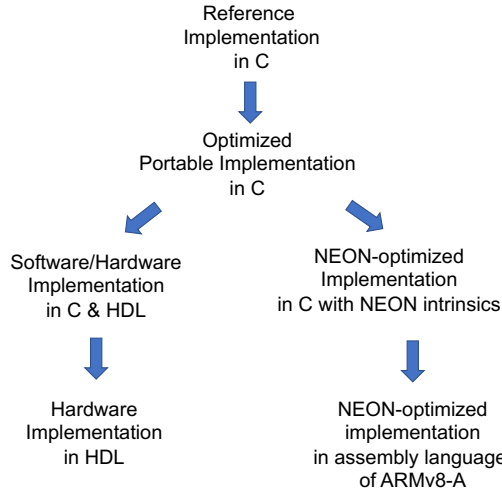


Figure 14: Types of optimized implementations.

Table 27: Comparison between Software Implementations using NEON instructions and Software-Hardware co-designs.

Algorithm	SW-Ref		SW-Neon		SW-HW		Ref/Neon		Ref/SW-HW		Neon/SW-HW	
	Encaps.	Decaps.	Encaps.	Decaps.	Encaps.	Decaps.	Encaps.	Decaps.	Encaps.	Decaps.	Encaps.	Decaps.
	$\mu s$	$\mu s$	$\mu s$	$\mu s$	$\mu s$	$\mu s$	Ratio	Ratio	Ratio	Ratio	Ratio	Ratio
NewHope-1024	723.5	891.2	338.3	363.6	21.3	24.9	2.1	2.5	34.0	35.8	<b>15.92</b>	<b>14.61</b>
ntruhrss701	3,060.0	8,896.0	207.0	311.0	68.3	135.6	14.8	28.6	44.8	65.6	<b>3.03</b>	<b>2.29</b>
ntruhs2048677	3,134.0	8,281.0	483.0	256.0	41.2	95.3	6.5	32.3	76.0	86.9	<b>11.72</b>	<b>2.68</b>
ntruhs4096821	4,504.0	12,125.0	598.0	327.0	48.4	107.1	7.5	37.0	93.0	113.2	<b>12.35</b>	<b>3.05</b>
LightSaber	503.1	629.4	141.8	151.0	49.0	52.5	3.5	4.1	10.3	12.0	<b>2.89</b>	<b>2.87</b>
Saber	965.3	1158.4	235.4	250.8	56.9	64.7	4.1	4.6	16.9	17.9	<b>4.13</b>	<b>3.87</b>
FireSaber	1573.9	1835.6	349.9	374.3	65.2	77.1	4.5	4.9	24.1	23.8	<b>5.36</b>	<b>4.85</b>

1263 On the selected platform, Zynq UltraScale+, a reference implementation of a PQC scheme  
 1264 in C can be optimized using several approaches shown in Fig. 14. First, basic optimizations  
 1265 may be still possible in C without affecting the portability of the code. From here, two  
 1266 divergent paths are worth investigating. First, Optimized Portable Implementation in  
 1267 C can be turned into a Software/Hardware Implementation in C and HDL, using the  
 1268 methodology described in this paper in Sections 4.3, 4.4, 4.6, 4.7. After all C functions are  
 1269 moved to hardware, this implementation becomes a pure hardware implementation in HDL.

1270 An alternative path is based on the use of SIMD instructions of ARMv8-a, referred to as  
1271 NEON instructions. These instructions can be called from C using the so-called intrinsics.  
1272 NEON intrinsics are function calls that the compiler replaces with an appropriate NEON  
1273 instruction or a sequence of NEON instructions. Intrinsics provide almost as much control  
1274 as writing assembly language but leave the allocation of registers to the compiler [6].  
1275 Operations that cannot take advantage of vector instructions are left in C. This path  
1276 can be further extended into pure assembly language code. This code may consist of  
1277 hand-coded NEON assembly language instructions, as well as remaining (so-called scalar)  
1278 assembly language instructions of ARMv8-a.

1279 We have developed NEON-optimized implementations in C with NEON intrinsics for 4  
1280 investigated KEMs, representing 3 Round 2 PQC candidates, namely NewHope, NTRU-  
1281 HPS, NTRU-HRSS, and Saber. Our starting point consisted of optimized implementations  
1282 of these algorithms, targeting Intel and AMD processors, using AVX2 (Advanced Vector  
1283 Extensions 2). In Table 27, we compare the performance of NEON-optimized software  
1284 implementations (based on intrinsics) with the performance of our software/hardware  
1285 implementations.

1286 Our software/hardware implementations appear to be superior for all investigated candi-  
1287 dates and parameter sets. Compared to the software implementation based on NEON  
1288 intrinsics, the execution times for NewHope-1024 are over 14.5 times smaller in soft-  
1289 ware/hardware. For NTRU-HPS software/hardware implementation is about 8 times  
1290 faster for encapsulation and 3.5-4.0 times faster for decapsulation. Saber has the ratios  
1291 approximately the same for encapsulation and decapsulation. However, the advantage of  
1292 software/hardware increases with the increase in the security level. Finally, NTRU-HRSS  
1293 has the smallest ratios in the range of 1.8-1.9.

1294 Multiple conference papers have been devoted to the NEON-based implementation of a  
1295 single public-key cryptosystem [15, 65, 66, 10, 51, 69, 67]. These papers demonstrate  
1296 that developing optimized implementations based on NEON intrinsics, hand-coded NEON  
1297 assembly language code, and hand-coded ARMv8-A RISC assembly language code is at  
1298 least as complex and labor intensive as the development of optimized software/hardware  
1299 implementations.

1300 The advantages of NEON-based implementations include a) software-only paradigm - no  
1301 need for expertise in hardware and knowledge of HDLs, b) NEON vector instructions run  
1302 at a higher clock frequency than an FPGA-based hardware accelerator, c) using the NEON  
1303 co-processor involves minimal (if any) transfer overhead, d) the NEON co-processor can be  
1304 potentially reused for non-cryptographic operations, such as signal and image processing.  
1305 The primary advantages of the software/hardware implementations are: a) Programmable  
1306 logic is much more powerful and less restrictive than the NEON co-processor in terms  
1307 of the number and type of operations that can be executed in parallel. As a result, a  
1308 higher overall speed-up is accomplished. b) Hardware written in HDL is likely to be more  
1309 portable than software written in the assembly language of a particular processor. In  
1310 particular, our software/hardware implementations can be ported to any other modern  
1311 SoC FPGA, assuming that the amount of the required hardware resources does not exceed  
1312 the capabilities of the programmable logic of a given SoC device.

## 1313 **6 Comparison with performance of the AVX2-optimized** 1314 **software implementations**

1315 In Table 28, we compare the performance of our software/hardware implementations,  
1316 running on Zynq UltraScale+, with the performance of the best software implementations  
1317 available to date, running on Intel Xeon E3-1220 v3 (3.1 GHz).

1318 When comparing these implementations, one needs to keep in mind that the software

**Table 28:** Comparison of the GMU software/hardware implementations, running on Zynq UltraScale+, with the software implementations in supercop-20200525 running on Intel Xeon E3-1220 v3 (3100MHz)

Algorithm	median cycles	SW (us)	SW/HW (us)	Ratio
<b>Encapsulation</b>				
<b>Level 1 &amp; 2</b>				
ntruhrss701	26116	8.4	68.3	0.12
ntruhs2048677	35352	11.4	41.2	0.28
kyber512	44404	14.3	15.2	0.94
sntrup653	46620	15.0	48.5	0.31
lightsaber2	67568	21.8	49.0	0.44
ntrulpr653	69400	22.4	51.6	0.43
lac128	82684	26.7	15.9	<b>1.67</b>
r5nd1kem0d	89500	28.9	16.7	<b>1.73</b>
newhope512cca	109040	35.2	15.0	<b>2.34</b>
r5nd1kem5d	122492	39.5	13.8	<b>2.85</b>
frodokem640shake	4529184	1,461.0	1,223.0	<b>1.19</b>
<b>Level 3</b>				
ntruhs4096821	43100	13.9	48.4	0.29
sntrup761	48780	15.7	55.5	0.28
ntrulpr761	72372	23.3	59.6	0.39
kyber768	74040	23.9	17.9	<b>1.34</b>
saber2	115948	37.4	56.9	0.66
lac192	158628	51.2	21.4	<b>2.39</b>
r5nd3kem5d	209572	67.6	19.2	<b>3.52</b>
r5nd3kem0d	317244	102.3	21.9	<b>4.67</b>
frodokem976shake	9467152	3,053.9	1,642.5	<b>1.86</b>
<b>Level 4 &amp; 5</b>				
sntrup857	60668	19.6	63.4	0.31
ntrulpr857	91416	29.5	67.3	0.44
kyber1024	103936	33.5	22.1	<b>1.52</b>
firesaber2	175844	56.7	65.2	0.87
lac256	188244	60.7	23.8	<b>2.55</b>
newhope1024cca	201772	65.1	21.3	<b>3.06</b>
r5nd5kem5d	368004	118.7	26.0	<b>4.57</b>
r5nd5kem0d	392492	126.6	29.2	<b>4.34</b>
frodokem1344shake	16379980	5,283.9	2,186.2	<b>2.42</b>
<b>Decapsulation</b>				
<b>Level 1 &amp; 2</b>				
kyber512	37600	12.1	17.1	0.71
r5nd1kem0d	43000	13.9	19.3	0.72
sntrup653	59324	19.1	66.9	0.29
ntruhs2048677	62004	20.0	95.3	0.21
r5nd1kem5d	63624	20.5	15.7	<b>1.31</b>
ntruhrss701	63632	20.5	135.6	0.15
lightsaber2	69508	22.4	52.5	0.43
ntrulpr653	82732	26.7	70.9	0.38
lac128	105388	34.0	17.1	<b>1.99</b>
newhope512cca	109728	35.4	16.1	<b>2.19</b>
frodokem640shake	4494652	1,449.9	1,321.3	<b>1.10</b>
<b>Level 3</b>				
sntrup761	59120	19.1	78.9	0.24
kyber768	63916	20.6	20.1	<b>1.03</b>
ntruhs4096821	79448	25.6	107.1	0.24
ntrulpr761	85908	27.7	84.1	0.33
r5nd3kem5d	117028	37.8	22.8	<b>1.65</b>
saber2	118848	38.3	64.7	0.59
r5nd3kem0d	156692	50.5	27.0	<b>1.87</b>
lac192	243008	78.4	23.7	<b>3.30</b>
frodokem976shake	9380108	3,025.8	1,866.2	<b>1.62</b>
<b>Level 4 &amp; 5</b>				
sntrup857	80904	26.1	86.8	0.30
kyber1024	91628	29.6	24.7	<b>1.20</b>
ntrulpr857	112116	36.2	97.5	0.37
firesaber2	182136	58.8	77.1	0.76
r5nd5kem0d	193228	62.3	35.9	<b>1.73</b>
newhope1024cca	206248	66.5	24.8	<b>2.68</b>
r5nd5kem5d	209136	67.5	31.7	<b>2.13</b>
lac256	377784	121.9	26.9	<b>4.54</b>
frodokem1344shake	16312844	5,262.2	3,119.9	<b>1.69</b>

1319 portions of our implementations are written in portable C and run on a much less powerful  
1320 processor, ARM Cortex-A53, at the frequency of 1.2 GHz. Hardware portions run in the  
1321 programmable logic of Zynq UltraScale+, at a frequency specific to each algorithm, listed  
1322 in Table 23, varying between 200 MHz for NTRU-HPS and NTRU-HRSS, through 322  
1323 MHz for Saber, until 490 MHz for NewHope. Even the frequency of NewHope is over  
1324 6 times smaller than the frequency of Intel Xeon. Additionally, all compared software  
1325 implementations are optimized using AVX2 instructions, which let them take advantage of  
1326 the parallelism present in each algorithm.

1327 Under these circumstances, it is no surprise that Zynq UltraScale+ can outperform Intel  
1328 Xeon only when its software/hardware implementation is fully optimized by moving all  
1329 operations other than `randombytes()` to programmable logic. Such implementations of  
1330 Kyber, LAC, NewHope, and Round5, outperform the best software implementations for  
1331 both encapsulation and decapsulation. For encapsulation, the speed-ups vary from 1.34 for  
1332 Kyber at level 3 to 4.67 for Round5 without error correction for level 3. For decapsulation,  
1333 the speed-ups vary from 1.03 for Kyber at level 3 to 4.54 for LAC at level 5. The only  
1334 exceptions are: Kyber at level 1, which reaches only the ratio of 0.94 for encapsulation,  
1335 and 0.71 for decapsulation, and Round5 without error correction, which reaches only the  
1336 ratio of 0.72 for level 1.

1337 Somewhat surprisingly, also our software/hardware implementation of FrodoKEM out-  
1338 performs the best software implementation, even though the percentage of operations  
1339 offloaded to hardware in FrodoKEM is the smallest among all implemented KEMs, as  
1340 shown in Figs. 9 and 10.

## 1341 7 Conclusions

1342 In this paper, we first reviewed the previous work on hardware and software/hardware im-  
1343 plementations of Round 2 PQC schemes. Out of 26 candidates, six - NewHope, CRYSTALS-  
1344 Kyber, FrodoKEM, Saber, Round5, and SIKE - received the highest coverage in terms of  
1345 the number of implementations and related publications. All of them have both high-speed  
1346 and lightweight implementations reported. Candidates with the Register-Transfer Level  
1347 (RTL) high-speed implementations and no lightweight implementations include LAC, Clas-  
1348 sic McEliece, Picnic, and Rainbow. The publications on BIKE focused on key generation  
1349 and decoding but did not report results for the entire KEM or PKE. Candidates with at  
1350 least one software/hardware lightweight implementation but no RTL high-speed implemen-  
1351 tations include LEDAcrypt, CRYSTALS-DILITHIUM, and qTESLA. The coverage of the  
1352 following candidates was limited to High-Level Synthesis implementations: SPHINCS+  
1353 and MQDSS. We are not aware of any publications on hardware or software/hardware im-  
1354 plementations of Three Bears, HQC, NTS-KEM (before the merger with Classic McEliece),  
1355 ROLLO, RQC, FALCON, GeMSS, and LUOV. With a few exceptions, the majority of  
1356 lightweight implementations were software/hardware implementations based on RISC-V.  
1357 The lattice-based family received by far the most extensive coverage. The following  
1358 candidates from other families were shown competitive to lattice-based cryptography in  
1359 terms of speed: for encryption and key exchange: Classic McEliece, for digital signatures  
1360 Picnic and Rainbow. However, all of them were investigated primarily from the point of  
1361 view of high-speed implementations.

1362 In terms of the comparison of the lattice-based schemes, the previous publications were  
1363 somewhat inconclusive. The largest differences were demonstrated in studies targeting ASIC  
1364 implementations. These studies indicated the significant advantage of Kyber and NewHope  
1365 over LAC and FrodoKEM, in terms of both the execution times of encapsulation and  
1366 decapsulation, as well as power consumption and energy usage. However, the advantage over  
1367 LAC could have been caused by devoting more effort and silicon area to the implementation  
1368 of NTT (which benefits only NewHope and Kyber) vs. the implementation of vector

1369 arithmetic, units used to speed up LAC. The benchmarking of lattice-based signature  
1370 schemes was limited to CRYSTALS-DILITHIUM and qTESLA. The conclusions were  
1371 complicated by the withdrawal of heuristic parameters of qTESLA by the submitters on  
1372 Aug. 20, 2019, and very limited coverage of the remaining parameter sets.

1373 Due to the timing constraints, in our study, we decided to focus on 12 CCA-secure Key  
1374 Encapsulation Mechanisms (KEMs) representing 8 out of 9 lattice-based key exchange  
1375 schemes (all except Three Bears). Taking into account that even for this subset of  
1376 candidates, the development of full RTL implementations appeared to be beyond the  
1377 capabilities of a single group, we investigated the use of two techniques to speed up the  
1378 development process: software/hardware co-design and High-Level Synthesis. A hybrid  
1379 of these two approaches, with some modifications to the traditional HLS methodology,  
1380 appeared to give quite promising results. However, we eventually devoted most of our  
1381 effort to software/hardware co-design based on the merger of the RTL HDL code and  
1382 optimized C code.

1383 Unlike other groups, we applied software/hardware co-design to high-speed rather than  
1384 lightweight implementations, which led to the choice of Xilinx Zynq UltraScale+, a state-of-  
1385 the-art SoC FPGA family, as our primary platform. The differentiating factor is that this  
1386 platform includes a hardwired ARM Cortex-A53 processor operating at the frequency of  
1387 1.2 GHz and a significant amount of programmable logic supporting hardware accelerators  
1388 operating at the clock frequencies up to 500 MHz. Still, our designs remained almost  
1389 completely portable due to leaving the software portion in C and modeling hardware  
1390 portion in hardware description languages, such as VHDL, Verilog, and Chisel.

1391 The detailed design methodology is described in this paper, and the corresponding code  
1392 required to build a generic benchmarking platform, suitable for performing timing measure-  
1393 ments of hardware and software/hardware co-designs is available for other groups to adopt.  
1394 It is also our intention to make our implementations of PQC candidates open-source after  
1395 the corresponding publications are accepted to peer-reviewed conferences or journals.

1396 Our software/hardware co-design approach was successfully applied to all 12 mentioned  
1397 above KEMs. For each KEM, multiple parameters sets, typically corresponding to three  
1398 security levels, were supported. In order to determine FPGA resources required for each  
1399 parameter set individually, the choice between parameter sets is performed during logic  
1400 synthesis rather than at the run time.

1401 For all algorithms other than FrodoKEM, the percentage of the original execution time in  
1402 software taken by operations offloaded to hardware exceeded 97.4% for decapsulation and  
1403 96.9% for encapsulation. For FrodoKEM operations taking at least 94% of the execution  
1404 time in software were offloaded to hardware. Significant speed-ups ranging between 7.6  
1405 and 111.1 were obtained versus a portable implementation in C, running on ARM Cortex-  
1406 A53. More importantly, even when four KEMs representing three candidates - NewHope,  
1407 NTRU, and Saber - were optimized in software by using NEON intrinsics, corresponding  
1408 to special SIMD instructions of ARM, our software/hardware implementations maintained  
1409 the lead by a factor varying between 1.81 for encapsulation in NTRU-HRSS up to 16.27  
1410 for encapsulation in NewHope. Finally, as an ultimate test, our implementations were  
1411 compared with the software implementations optimized using AVX2 vector instructions,  
1412 running on Intel Xeon E3-1220 v3, with the frequency 3.1 GHz. For each security level,  
1413 between 4 and 6 software/hardware implementations, running on Zynq UltraScale+ with  
1414 the 1.2 GHz ARM core were superior than the corresponding AVX2 implementations.

1415 For each candidate, an attempt was made to offload as many as possible operations to  
1416 hardware. For 50% of investigated KEMs, this percentage reached 100%. Thus, the  
1417 corresponding implementations could be treated as hardware implementations, assuming  
1418 that a random seed (of the size of 16, 24, or 32 bytes) was transferred to the hardware  
1419 module during encapsulation. KEMs implemented using this approach included Kyber,  
1420 LAC (v3a and v3b), NewHope, and Round5 (with and without error-correcting code).

1421 Their code was benchmarked using Artix-7 and Virtex-7 FPGAs.

1422 In terms of both the execution times and resource utilization, Round5 with an error-  
1423 correcting code (R5ND\_5d) outperformed Round5 without an error-correcting code  
1424 (R5ND\_0d). Similarly, LAC-v3b appeared superior over LAC-v3a in terms of both  
1425 speed and use of FPGA resources. Then, when the best representatives of four candidates  
1426 - Kyber, LAC, NewHope, and Round5 - were compared, the following conclusions could  
1427 be drawn. The execution times of these candidates were extremely close to one another.  
1428 For encapsulation, the execution times were within 10% from one another at the security  
1429 level 5, within 22% at the security level 3, and within 32% at the security level 1. For  
1430 decapsulation, the largest differences were 26% at level 5, 22% at level 3, and 48% at  
1431 level 1. In multiple instances, just a change of an FPGA family from low-cost Artix-7 to  
1432 high-performance Virtex-7 caused a significant change in the rankings, even though the  
1433 HDL code remained exactly the same. As a result, we must conclude that the differences  
1434 among these candidates in terms of speed are too small to give preference to any particular  
1435 candidate. These results contradict one of the earlier reports placing LAC well behind  
1436 NewHope and Kyber.

1437 In terms of resource utilization, a small advantage belongs to NewHope and Kyber. Both  
1438 of them use fewer LUTs and flip-flops than LAC and Round5, and their use of DSP units  
1439 and BRAMs, although slightly higher, is very moderate. Additionally, both NewHope and  
1440 Kyber use almost the same amount of resources independently of the security level. In  
1441 the case of both LAC and Round5, resource usage increases sharply with the increase in  
1442 security level. The former property appears to be an advantage for applications requiring  
1443 support for the highest or all security levels. In particular, the  $k$ -in-1 designs, which  
1444 support all  $k$  security levels and allow modifying them at run time, typically have only  
1445 slightly higher resource utilization than that for the maximum security level. Thus, the  
1446 flat dependence of the resource utilization on the security level implies a potential for very  
1447 cost-effective  $k$ -in-1 designs. At the same time, this potential should still be confirmed  
1448 through complete designs.

1449 Independently of our own results, the results reported in [63] and summarized in Table 8  
1450 indicate that Saber—when implemented entirely in hardware—can reach the same perfor-  
1451 mance level as the KEMs ranked on the first six positions in our study. It may also require  
1452 fewer resources than at least Round5. With its resource utilization almost independent of  
1453 the security level, it naturally allows 2-in-1 and 3-in-1 designs, supporting multiple security  
1454 levels with a relatively small overhead in terms of speed and resource usage as compared  
1455 to the implementation for the lowest security level. Similarly, estimates reported in [32]  
1456 indicate that the performance of ThreeBears is likely to be very similar to that of Kyber  
1457 and NewHope.

1458 For the remaining 5 KEMs, representing FrodoKEM, NTRU, and NTRU Prime, the  
1459 conclusions could be drawn only by comparing their software/hardware implementations  
1460 and contrasting them with the corresponding software/hardware implementations of  
1461 Kyber, LAC, NewHope, and Round5. In this case, all KEMs were implemented in Zynq  
1462 UltraScale+. Hardware accelerators were assumed to be preloaded with appropriate public  
1463 and private keys. Encapsulation started from generating 16-32 random bytes in software  
1464 and passing these bytes to the hardware accelerator. Decapsulation started by sending the  
1465 ciphertext to the hardware accelerator. Both operations ended when the shared secret was  
1466 available in the memory of the processor core. Our evaluation revealed that FrodoKEM was  
1467 by at least an order of magnitude slower than the remaining investigated KEMs. Ranking of  
1468 the remaining candidates in hardware could not be determined conclusively based on their  
1469 software/hardware co-design rankings. Software/hardware co-designs of Saber, NTRU-  
1470 HRSS, and NTRU-HPS in particular, and somewhat less likely of Streamlined NTRU  
1471 Prime and NTRU LPRime, could be possibly still significantly improved by offloading  
1472 more operations to hardware, up to the level of bypassing at least one of the first six



1473 candidates in the ranking.

1474 This pitfall of software/hardware co-designs was identified early on during the benchmarking  
1475 process. It could have been overcome only if candidates were significantly different from  
1476 the point of view of their hardware efficiency. Such large differences were not identified  
1477 in the case of the mentioned above five lattice-based KEMs. Consequently, the only way  
1478 to overcome this inherent weakness of the software/hardware methodology, when applied  
1479 to this particular set of candidates, is to move all (or almost all) remaining operations of  
1480 these algorithms to hardware. Doing that is, however, impractical at this point due to the  
1481 timeline imposed by NIST.

1482 At the same time, taking into account that moving more operations of these KEMs to  
1483 hardware can only increase the resource usage of the corresponding hardware accelerators,  
1484 it is still fair to compare their resource utilization with those of Kyber and NewHope. For  
1485 NTRU-HPS and NTRU-HRSS, the concern is a large number of DSP units, exceeding  
1486 700 for NTRU-HRSS and 800 for NTRU-HPS at the security level 3. For Streamlined  
1487 NTRU Prime and NTRU LPrime, the only concern is a relatively large number of LUTs,  
1488 clearly exceeding that of LAC-v3b and approaching or exceeding that of Round5 with an  
1489 error-correcting code (R5ND\_5d).

1490 Still, it is up to NIST and the cryptographic community to decide whether such relatively  
1491 small differences in the hardware efficiency of lattice-based candidates should play any role  
1492 in the Round 3 down-selection process.

## 1493 8 Future Work

1494 Future work will depend on the number and type of candidates qualified for Round 3.  
1495 Based on the lessons learned from Round 2, the following adjustments may be advisable:

- 1496 • More focus on hardware implementations vs. software/hardware implementations.  
1497 Software/hardware implementations may still be helpful for lightweight implementa-  
1498 tions with a clear resource utilization threshold. In these implementations, moving  
1499 more operations to hardware may be prohibited by exceeding the resource budget.
- 1500 • More focus on comparisons across families, rather than within the same family.  
1501 Round 2 designs illustrate substantial similarities between candidates belonging to  
1502 the same family but give a hint of more profound differences among representatives  
1503 of different families.
- 1504 • More hardware platforms to focus on. The larger the spectrum of platforms, the  
1505 higher certainty that the reported rankings are not artifacts of a particular platform  
1506 and will carry over to future generations of integrated circuits. For FPGAs and  
1507 SoC FPGAs, benchmarking should target families of at least two major vendors,  
1508 Xilinx and Intel. For ASIC implementations, different standard-cell libraries should  
1509 be considered. ASIC studies are particularly challenging, as they are more time-  
1510 consuming and costlier. However, they are indispensable as they may lead to different  
1511 conclusions than those obtained from FPGA investigations.
- 1512 • More work on optimized software implementations targeting vector instructions of  
1513 embedded processors, such as RISC-V and ARM (including NEON instructions).
- 1514 • Investigation of lightweight implementations protected against side-channel and fault  
1515 attacks should be conducted by multiple groups, serving interchangeably as attackers  
1516 and defenders.
- 1517 • Trade-offs among speed, area, power, energy, and resistance against side channel  
1518 attacks should be thoroughly studied, especially for lightweight implementations.

## References

- 1519
- 1520 [1] Erdem Alkim et al. *ISA Extensions for Finite Field Arithmetic - Accelerating Kyber*  
1521 *and NewHope on RISC-V*. Cryptology ePrint Archive 2020/049. Jan. 2020.
- 1522 [2] Erdem Alkim et al. *NewHope - Algorithm Specifications and Supporting Documenta-*  
1523 *tion Version 1.1*. Tech. rep. Apr. 2020.
- 1524 [3] Michał Andrzejczak. “The Low-Area FPGA Design for the Post-Quantum Cryptog-
- 1525 raphy Proposal Round5”. In: *2019 Federated Conference on Computer Science*  
1526 *and Information Systems*. Vol. 18. Leipzig, Germany, Sept. 2019, pp. 213–219. DOI:  
1527 [10/ggbsbd](https://doi.org/10/ggbsbd).
- 1528 [4] Michal Andrzejczak, Farnoud Farahmand, and Kris Gaj. “Full Hardware Imple-
- 1529 mentation of the Post-Quantum Public-Key Cryptography Scheme Round5”. In:  
1530 *2019 International Conference on ReConFigurable Computing and FPGAs (ReCon-*  
1531 *Fig)*. Cancun, Mexico: IEEE, Dec. 2019, pp. 1–2. ISBN: 978-1-72811-957-1. DOI:  
1532 [10.1109/ReConFig48160.2019.8994765](https://doi.org/10.1109/ReConFig48160.2019.8994765).
- 1533 [5] Nicolas Aragon et al. *BIKE: Bit Flipping Key Encapsulation: Submission for Round*  
1534 *3 Consideration*. Tech. rep. May 2020.
- 1535 [6] ARM. *Neon Intrinsics Reference*. [https://developer.arm.com/architectures/](https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics)  
1536 [instruction-sets/simd-isas/neon/intrinsics](https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/intrinsics). 2020.
- 1537 [7] ARM. *Neon Programmer’s Guide for Armv8-A*. [https://developer.arm.com/](https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/neon-programmers-guide-for-armv8-a)  
1538 [architectures/instruction-sets/simd-isas/neon/neon-programmers-guide-](https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/neon-programmers-guide-for-armv8-a)  
1539 [for-armv8-a](https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/neon-programmers-guide-for-armv8-a). 2020.
- 1540 [8] Roberto Avanzi et al. *CRYSTALS-KYBER: Algorithm Specifications And Supporting*  
1541 *Documentation (Version 2.0)*. Tech. rep. Apr. 2019.
- 1542 [9] Reza Azarderakhsh et al. “Key Compression for Isogeny-Based Cryptosystems”. In:  
1543 *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptogra-*  
1544 *phy - AsiaPKC ’16*. Xi’an, China: ACM Press, 2016, pp. 1–10. ISBN: 978-1-4503-4286-5.  
1545 DOI: [10/ggbsbz](https://doi.org/10/ggbsbz).
- 1546 [10] Reza Azarderakhsh et al. *NEON PQCrypto: Fast and Parallel Ring-LWE Encryption*  
1547 *on ARM NEON Architecture*. Cryptology ePrint Archive 2015/1081. Nov. 2015.
- 1548 [11] Brian Baldwin et al. “FPGA Implementations of the Round Two SHA-3 Candidates”.  
1549 In: *2010 International Conference on Field Programmable Logic and Applications,*  
1550 *FPL 2010*. Milan, Italy, Aug. 2010, pp. 400–407. ISBN: 978-1-4244-7842-2. DOI:  
1551 [10/bmn2zv](https://doi.org/10/bmn2zv).
- 1552 [12] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. “Sapphire: A  
1553 Configurable Crypto-Processor for Post-Quantum Lattice-Based Protocols”. In: *IACR*  
1554 *Transactions on Cryptographic Hardware and Embedded Systems 2019.4* (Aug. 2019).  
1555 DOI: [10.13154/tches.v2019.i4.17-61](https://doi.org/10.13154/tches.v2019.i4.17-61).
- 1556 [13] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. *Sapphire: A*  
1557 *Configurable Crypto-Processor for Post-Quantum Lattice-Based Protocols (Extended*  
1558 *Version)*. Cryptology ePrint Archive 2019/1140. Sept. 2020.
- 1559 [14] Kanad Basu et al. *NIST Post-Quantum Cryptography- A Hardware Evaluation Study*.  
1560 Cryptology ePrint Archive 2019/047. May 2019.
- 1561 [15] Daniel J Bernstein and Peter Schwabe. “NEON Crypto”. In: *Cryptographic Hardware*  
1562 *and Embedded Systems - CHES 2012*. Vol. 7428. LNCS. Leuven, Belgium, Sept. 2012,  
1563 pp. 320–339. DOI: [10.1007/978-3-642-33027-8\\_19](https://doi.org/10.1007/978-3-642-33027-8_19).
- 1564 [16] *CAESAR: Competition for Authenticated Encryption: Security, Applicability, and*  
1565 *Robustness - Web Page*. <https://competitions.cr.yp.to/caesar.html>. 2019.

- 1566 [17] Cryptographic Engineering Research Group (CERG) at George Mason University.  
1567 *Hardware Benchmarking of CAESAR Candidates*. <https://cryptography.gmu.edu/athena/index.php?id=CAESAR>. 2019.  
1568
- 1569 [18] Viet B Dang et al. “Implementing and Benchmarking Three Lattice-Based Post-  
1570 Quantum Cryptography Algorithms Using Software/Hardware Codesign”. In: *2019*  
1571 *International Conference on Field Programmable Technology, FPT 2019*. Tianjin,  
1572 China: IEEE, Dec. 9-13, 2019, pp. 206–214. DOI: [10.1109/ICFPT47387.2019.00032](https://doi.org/10.1109/ICFPT47387.2019.00032).
- 1573 [19] Rami Elkhatib, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. *Efficient and*  
1574 *Fast Hardware Architectures for SIKE Round 2 on FPGA*. Cryptology ePrint Archive  
1575 2020/611. May 2020.
- 1576 [20] Farnoud Farahmand. *Benchmarking Setup for Software/Hardware Implementations*  
1577 *of PQC Schemes*. Sept. 2019.
- 1578 [21] Farnoud Farahmand et al. “Evaluating the Potential for Hardware Acceleration of  
1579 Four NTRU-Based Key Encapsulation Mechanisms Using Software/Hardware Code-  
1580 sign”. In: *10th International Conference on Post-Quantum Cryptography, PQCrypto*  
1581 *2019*. LNCS. Chongqing, China: Springer, May 2019.
- 1582 [22] Farnoud Farahmand et al. “Minerva: Automated Hardware Optimization Tool”. In:  
1583 *2017 International Conference on ReConfigurable Computing and FPGAs, ReConFig*  
1584 *2017*. Cancun: IEEE, Dec. 2017, pp. 1–8.
- 1585 [23] Farnoud Farahmand et al. “Software/Hardware Codesign of the Post Quantum  
1586 Cryptography Algorithm NTRUEncrypt Using High-Level Synthesis and Register-  
1587 Transfer Level Design Methodologies”. In: *29th International Conference on Field*  
1588 *Programmable Logic and Applications, FPL 2019*. Barcelona, Spain: IEEE, Sept.  
1589 2019, pp. 225–231. ISBN: 978-1-72814-884-7. DOI: [10.1109/FPL.2019.00042](https://doi.org/10.1109/FPL.2019.00042).
- 1590 [24] Ahmed Ferozpur and Kris Gaj. “High-Speed FPGA Implementation of the NIST  
1591 Round 1 Rainbow Signature Scheme”. In: *2018 International Conference on ReCon-*  
1592 *Figurable Computing and FPGAs (ReConFig)*. Cancun, Mexico: IEEE, Dec. 2018,  
1593 pp. 1–8. ISBN: 978-1-72811-968-7. DOI: [10/ggbsdm](https://doi.org/10/ggbsdm).
- 1594 [25] Ahmed Ferozpur et al. *Hardware API for Post-Quantum Public Key Cryptosystems*.  
1595 GMU Report. Fairfax, VA: George Mason University, Apr. 2018.
- 1596 [26] Tim Fritzmman, Georg Sigl, and Johanna Sepúlveda. *RISQ-V: Tightly Coupled*  
1597 *RISC-V Accelerators for Post-Quantum Cryptography*. Cryptology ePrint Archive  
1598 2020/446. Apr. 2020.
- 1599 [27] Tim Fritzmman et al. “Towards Reliable and Secure Post-Quantum Co-Processors  
1600 Based on RISC-V”. In: *2019 Design, Automation Test in Europe Conference Exhibi-*  
1601 *tion (DATE)*. Mar. 2019, pp. 1148–1153. DOI: [10.23919/DATE.2019.8715173](https://doi.org/10.23919/DATE.2019.8715173).
- 1602 [28] Kris Gaj. “Challenges and Rewards of Implementing and Benchmarking Post-  
1603 Quantum Cryptography in Hardware”. In: *2018 Great Lakes Symposium on VLSI,*  
1604 *GLSVLSI 2018*. Chicago, IL, USA: ACM Press, 2018, pp. 359–364. ISBN: 978-1-4503-  
1605 5724-1. DOI: [10/ggbscs](https://doi.org/10/ggbscs).
- 1606 [29] Kris Gaj, Ekawat Homsirikamol, and Marcin Rogawski. “Fair and Comprehensive  
1607 Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3  
1608 Candidates Using FPGAs”. In: *Cryptographic Hardware and Embedded Systems,*  
1609 *CHES 2010*. Vol. 6225. LNCS. Santa Barbara, CA, Aug. 2010, pp. 264–278. ISBN:  
1610 978-3-642-15030-2 978-3-642-15031-9. DOI: [10.1007/978-3-642-15031-9\\_18](https://doi.org/10.1007/978-3-642-15031-9_18).

- 1611 [30] Kris Gaj et al. “ATHENa - Automated Tool for Hardware Evaluation: Toward Fair  
1612 and Comprehensive Benchmarking of Cryptographic Hardware Using FPGAs”. In:  
1613 *2010 International Conference on Field Programmable Logic and Applications, FPL*  
1614 *2010*. Milan, Italy: IEEE, Aug. 2010, pp. 414–421. ISBN: 978-1-4244-7842-2. DOI:  
1615 [10/d2bzw2](https://doi.org/10/d2bzw2).
- 1616 [31] Kris Gaj et al. *Comprehensive Evaluation of High-Speed and Medium-Speed Imple-*  
1617 *mentations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs*. Cryptology  
1618 ePrint Archive 2012/368. 2012.
- 1619 [32] Mike Hamburg. *ThreeBears Estimates on ASIC*. June 2020.
- 1620 [33] E. Homsirikamol et al. *Implementer’s Guide to Hardware Implementations Compliant*  
1621 *with the CAESAR Hardware API*. GMU Report. Fairfax, VA: GMU, 2016.
- 1622 [34] Ekawat Homsirikamol and Kris Gaj. “Hardware Benchmarking of Cryptographic  
1623 Algorithms Using High-Level Synthesis Tools: The SHA-3 Contest Case Study”. In:  
1624 *Applied Reconfigurable Computing - ARC 2015*. Vol. 9040. LNCS. Cham: Springer  
1625 International Publishing, 2015, pp. 217–228. ISBN: 978-3-319-16213-3 978-3-319-  
1626 16214-0. DOI: [10.1007/978-3-319-16214-0\\_18](https://doi.org/10.1007/978-3-319-16214-0_18).
- 1627 [35] Ekawat Homsirikamol and Kris Gaj. “Toward a New HLS-Based Methodology for  
1628 FPGA Benchmarking of Candidates in Cryptographic Competitions: The CAESAR  
1629 Contest Case Study”. In: *2017 International Conference on Field Programmable*  
1630 *Technology, FPT 2017*. Melbourne, Australia: IEEE, Dec. 2017, pp. 120–127. ISBN:  
1631 978-1-5386-2656-6. DOI: [10/ggbsf4](https://doi.org/10/ggbsf4).
- 1632 [36] Ekawat Homsirikamol, Panasayya Yalla, and Farnoud Farahmand. *Development*  
1633 *Package for Hardware Implementations Compliant with the CAESAR Hardware API*.  
1634 <https://cryptography.gmu.edu/athena/index.php?id=CAESAR>. 2016.
- 1635 [37] Ekawat Homsirikamol et al. *CAESAR Hardware API*. Cryptology ePrint Archive  
1636 2016/626. 2016.
- 1637 [38] James Howe. “Optimised Lattice-Based Key Encapsulation in Hardware”. In: *Second*  
1638 *NIST Post-Quantum Cryptography Standardization Conference 2019*. Aug. 2019,  
1639 p. 13.
- 1640 [39] Jingwei Hu et al. “Lightweight Key Encapsulation Using LDPC Codes on FPGAs”.  
1641 In: *IEEE Trans. Comput.* 69.3 (Mar. 2020). ISSN: 0018-9340, 1557-9956, 2326-3814.  
1642 DOI: [10.1109/TC.2019.2948323](https://doi.org/10.1109/TC.2019.2948323).
- 1643 [40] Jingwei Hu et al. “Optimized Polynomial Multiplier Over Commutative Rings on  
1644 FPGAs: A Case Study on BIKE”. In: *2019 International Conference on Field-*  
1645 *Programmable Technology (ICFPT)*. Tianjin, China: IEEE, Dec. 2019, pp. 231–234.  
1646 ISBN: 978-1-72812-943-3. DOI: [10.1109/ICFPT47387.2019.00035](https://doi.org/10.1109/ICFPT47387.2019.00035).
- 1647 [41] Arpan Jati et al. *SPQCop: Side-Channel Protected Post-Quantum Cryptoprocessor*.  
1648 Cryptology ePrint Archive 2019/765. June 2019.
- 1649 [42] Daniel Kales et al. “Efficient FPGA Implementations of LowMC and Picnic”. In: *The*  
1650 *Cryptographers’ Track at the RSA Conference 2020, CT-RSA 2020*. San Francisco:  
1651 Springer, Feb. 2020.
- 1652 [43] Matthias J. Kannwischer et al. *Pqm4 - Post-Quantum Crypto Library for the {ARM}*  
1653 *{Cortex-M4}*. <https://github.com/mupq/pqm4>. 2019.
- 1654 [44] Jens-Peter Kaps et al. “Lightweight Implementations of SHA-3 Candidates on  
1655 FPGAs”. In: *12th International Conference on Cryptology in India, Indocrypt 2011*.  
1656 Vol. 7107. LNCS. Chennai, India, Dec. 2011, pp. 270–289. ISBN: 978-3-642-25577-9  
1657 978-3-642-25578-6. DOI: [10.1007/978-3-642-25578-6\\_20](https://doi.org/10.1007/978-3-642-25578-6_20).

- 1658 [45] Miroslav Knezevic et al. “Fair and Consistent Hardware Evaluation of Fourteen Round  
1659 Two SHA-3 Candidates”. In: *IEEE Transactions on Very Large Scale Integration*  
1660 *(VLSI) Systems* 20.5 (May 2012), pp. 827–840. ISSN: 1063-8210, 1557-9999. DOI:  
1661 [10/ctjzhr](https://doi.org/10.1109/ctjzhr).
- 1662 [46] B. Koziel, R. Azarderakhsh, and M. M. Kermani. “A High-Performance and Scalable  
1663 Hardware Architecture for Isogeny-Based Cryptography”. In: *IEEE Transactions on*  
1664 *Computers* 67.11 (Nov. 2018), pp. 1594–1609. ISSN: 0018-9340. DOI: [10/gff4vv](https://doi.org/10.1109/gff4vv).
- 1665 [47] Brian Koziel et al. “NEON-SIDH: Efficient Implementation of Supersingular Isogeny  
1666 Diffie-Hellman Key Exchange Protocol on ARM”. In: *Cryptology and Network*  
1667 *Security*. Ed. by Sara Foresti and Giuseppe Persiano. Vol. 10052. Cham: Springer  
1668 International Publishing, 2016, pp. 88–103. ISBN: 978-3-319-48964-3 978-3-319-48965-  
1669 0. DOI: [10.1007/978-3-319-48965-0\\_6](https://doi.org/10.1007/978-3-319-48965-0_6).
- 1670 [48] Brian Koziel et al. “Post-Quantum Cryptography on FPGA Based on Isogenies on  
1671 Elliptic Curves”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers*  
1672 64.1 (Jan. 2017), pp. 86–99. ISSN: 1549-8328, 1558-0806. DOI: [10/gd89pp](https://doi.org/10.1109/gd89pp).
- 1673 [49] Brian Koziel et al. “SIKE’d Up: Fast Hardware Architectures for Supersingular  
1674 Isogeny Key Encapsulation”. en. In: *IEEE Transactions on Circuits and Systems I:*  
1675 *Regular Papers* (May 2020), pp. 1–13. ISSN: 1549-8328, 1558-0806. DOI: [10.1109/  
1676 TCSI.2020.2992747](https://doi.org/10.1109/TCSI.2020.2992747).
- 1677 [50] Weiqiang Liu et al. “High Performance Modular Multiplication for SIDH”. In: *IEEE*  
1678 *Trans. Comput.-Aided Des. Integr. Circuits Syst.* (2019), pp. 1–1. ISSN: 0278-0070,  
1679 1937-4151. DOI: [10.1109/TCAD.2019.2960330](https://doi.org/10.1109/TCAD.2019.2960330).
- 1680 [51] Patrick Longa. *FourQNEON: Faster Elliptic Curve Scalar Multiplications on ARM*  
1681 *Processors*. Cryptology ePrint Archive 2016/645. July 2016.
- 1682 [52] Xianhui Lu et al. *LAC: Practical Ring-LWE Based Public-Key Encryption with*  
1683 *Byte-Level Modulus*. Cryptology ePrint Archive 2018/1009. Dec. 2019.
- 1684 [53] Pedro Maat C. Massolino et al. “A Compact and Scalable Hardware/Software Co-  
1685 Design of SIKE”. In: *IACR Transactions on Cryptographic Hardware and Embedded*  
1686 *Systems* (Mar. 2020), pp. 245–271. ISSN: 2569-2925. DOI: [10.13154/tches.v2020.  
1687 i2.245-271](https://doi.org/10.13154/tches.v2020.i2.245-271).
- 1688 [54] Jose Maria Bermudo Mera et al. *Compact Domain-Specific Co-Processor for Ac-*  
1689 *celerating Module Lattice-Based Key Encapsulation Mechanism*. Cryptology ePrint  
1690 Archive 2020/321. Mar. 2020, p. 15.
- 1691 [55] Richard Newell. *Survey of Notable Security-Enhancing Activities in the RISC-V*  
1692 *Universe*. 17th International Workshop on Cryptographic Architectures Embedded  
1693 in Logic Devices, CryptArchi 2019. Pruhonice, Czech Republic, June 2019.
- 1694 [56] Duc Tri Nguyen, Viet B. Dang, and Kris Gaj. “A High-Level Synthesis Approach  
1695 to the Software/Hardware Codesign of NTT-Based Post-Quantum Cryptography  
1696 Algorithms”. In: *2019 International Conference on Field-Programmable Technology*  
1697 *(ICFPT)*. Tianjin, China: IEEE, Dec. 2019, pp. 371–374. ISBN: 978-1-72812-943-3.  
1698 DOI: [10.1109/ICFPT47387.2019.00070](https://doi.org/10.1109/ICFPT47387.2019.00070).
- 1699 [57] Duc Tri Nguyen, Viet B Dang, and Kris Gaj. “High-Level Synthesis in Implementing  
1700 and Benchmarking Number Theoretic Transform in Lattice-Based Post-Quantum  
1701 Cryptography Using Software/Hardware Codesign”. In: *16th International Symposi-*  
1702 *um on Applied Reconfigurable Computing, ARC 2020*. Apr. 2020.
- 1703 [58] NIST. *PQC - API Notes*. 2017.
- 1704 [59] David Patterson and Andrew Waterman. *The RISC-V Reader: An Open Architecture*  
1705 *Atlas*. Book version: 0.0.1. Strawberry Canyon LLC, Oct. 2017.

- 1706 [60] Andrew H Reinders et al. *Efficient BIKE Hardware Design with Constant-Time*  
1707 *Decoder*. Cryptology ePrint Archive 2020/117. Feb. 2020.
- 1708 [61] Jan Richter-Brockmann and Tim Güneysu. *Folding BIKE: Scalable Hardware Im-*  
1709 *plementation for Reconfigurable Devices*. Cryptology ePrint Archive 2020/897. July  
1710 2020.
- 1711 [62] Vincent Rijmen, Antoon Bosselaers, and Paulo Barreto. *Optimized ANSI C Code*  
1712 *for the Rijndael Cipher (Now AES), Rijndael-Alg-Fst.c, v3.0*. Dec. 2000.
- 1713 [63] Sujoy Sinha Roy and Andrea Basso. *High-Speed Instruction-Set Coprocessor for*  
1714 *Lattice-Based Key Encapsulation Mechanism: Saber in Hardware*. Cryptology ePrint  
1715 Archive 2020/434. Apr. 2020.
- 1716 [64] Markku-Juhani O. Saarinen. *Pqcbench*. <https://github.com/mjosaarinen/pqcbench>.  
1717 2019.
- 1718 [65] Hwajeong Seo et al. “Montgomery Modular Multiplication on ARM-NEON Revisited”.  
1719 In: *International Conference on Information Security and Cryptology - ICISC 2014*.  
1720 Vol. 8949. LNCS. Cham: Springer International Publishing, 2014, pp. 328–342. ISBN:  
1721 978-3-319-15942-3 978-3-319-15943-0. DOI: [10.1007/978-3-319-15943-0](https://doi.org/10.1007/978-3-319-15943-0).
- 1722 [66] Hwajeong Seo et al. *Parallel Implementation of Number Theoretic Transform*. Crypt-  
1723 ology ePrint Archive 2015/1024. Oct. 2015.
- 1724 [67] Hwajeong Seo et al. *SIDH on ARM: Faster Modular Multiplications for Faster Post-*  
1725 *Quantum Supersingular Isogeny Key Exchange*. Cryptology ePrint Archive 2018/700.  
1726 July 2018.
- 1727 [68] Douglas Stebila and Michele Mosca. *Liboqs - Master Branch*. [https://github.com/](https://github.com/open-quantum-safe/liboqs)  
1728 [open-quantum-safe/liboqs](https://github.com/open-quantum-safe/liboqs). 2019.
- 1729 [69] Silvan Streit and Fabrizio De Santis. “Post-Quantum Key Exchange on ARMv8-A:  
1730 A New Hope for NEON Made Simple”. In: *IEEE Transactions on Computers* 67.11  
1731 (Nov. 2018), pp. 1651–1662. ISSN: 0018-9340, 1557-9956, 2326-3814. DOI: [10/gff3sc](https://doi.org/10/gff3sc).
- 1732 [70] FrodoKEM Submission Team. *Round 2 Submissions - FrodoKEM Candidate Submis-*  
1733 *sion Package*. [https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/](https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions)  
1734 [Round-2-Submissions](https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions). Apr. 2019.
- 1735 [71] NTRU Prime Submission Team. *Round 2 Submissions - NTRU Prime Candidate*  
1736 *Submission Package*. [https://csrc.nist.gov/Projects/Post-Quantum-Cryptog-](https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions)  
1737 [raphy/Round-2-Submissions](https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions). Apr. 2019.
- 1738 [72] NTRU Submission Team. *Round 2 Submissions - NTRU Candidate Submission*  
1739 *Package*. Apr. 2019.
- 1740 [73] Round5 Submission Team. *Round 2 Submissions - Round5 Candidate Submission*  
1741 *Package*. [https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/](https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions)  
1742 [Round-2-Submissions](https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions). Apr. 2019.
- 1743 [74] Saber Submission Team. *Round 2 Submissions - Saber Candidate Submission Package*.  
1744 Apr. 2019.
- 1745 [75] Wen Wang, Jakub Szefer, and Ruben Niederhagen. “FPGA-Based Niederreiter  
1746 Cryptosystem Using Binary Goppa Codes”. In: *9th International Conference on*  
1747 *Post-Quantum Cryptography, PQCrypto 2018*. Ed. by Tanja Lange and Rainer  
1748 Steinwandt. Vol. 10786. LNCS. Fort Lauderdale, Florida: Springer International  
1749 Publishing, Apr. 2018, pp. 77–98. ISBN: 978-3-319-79062-6 978-3-319-79063-3. DOI:  
1750 [10.1007/978-3-319-79063-3\\_4](https://doi.org/10.1007/978-3-319-79063-3_4).
- 1751 [76] Wen Wang et al. *Parameterized Hardware Accelerators for Lattice-Based Cryptography*  
1752 *and Their Application to the HW/SW Co-Design of qTESLA*. Cryptology ePrint  
1753 Archive 2020/054. Apr. 2020.

- 1754 [77] Wen Wang et al. “XMSS and Embedded Systems - XMSS Hardware Accelerators  
1755 for RISC-V”. In: *Selected Areas in Cryptography – SAC 2019*. Vol. 11959. LNCS.  
1756 Waterloo, Ontario, Canada: Springer, 2019, pp. 523–550.
- 1757 [78] Andrew Waterman and Krste Asanovic. *The RISC-V Instruction Set Manual .*  
1758 *Volume I: Unprivileged ISA v2.2*. Tech. rep. 20190608-Base-Ratified. June 2019,  
1759 p. 236.
- 1760 [79] Andrew Waterman and Krste Asanovic. *The RISC-V Instruction Set Manual, Volume*  
1761 *II: Privileged Architecture, v1.12*. Tech. rep. June 2019, p. 113.
- 1762 [80] Guozhu Xin et al. “VPQC: A Domain-Specific Vector Processor for Post-Quantum  
1763 Cryptography Based on RISC-V Architecture”. In: *IEEE Transactions on Circuits*  
1764 *and Systems I: Regular Papers* 67.8 (Aug. 2020), pp. 1–13. ISSN: 1558-0806. DOI:  
1765 [10.1109/TCSI.2020.2983185](https://doi.org/10.1109/TCSI.2020.2983185).
- 1766 [81] Neng Zhang et al. “Highly Efficient Architecture of NewHope-NIST on FPGA Using  
1767 Low-Complexity NTT/INTT”. In: *IACR Transactions on Cryptographic Hardware*  
1768 *and Embedded Systems* (Mar. 2020), pp. 49–72. ISSN: 2569-2925. DOI: [10.13154/  
1769 tches.v2020.i2.49-72](https://doi.org/10.13154/tches.v2020.i2.49-72).

## A Results of Profiling

**Table 29:** Results of profiling FrodoKEM

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			Software/Hardware		
<b>FrodoKEM1344 - Encapsulation</b>					
1. frodo_mul_add_sa_plus_e	58,577.48	94.36	1.1 frodo_mul_add_sa_plus_e	1,328.39	60.76
2. Shake128 and frodo_sample_n x3	1,416.27	2.28	1.2 Shake128 and frodo_sample_n		
3. frodo_mul_add_sb_plus_e	654.64	1.05	1.3 frodo_mul_add_sb_plus_e		
4. Shake256	569.60	0.92	1.4 Shake256		
5. frodo_pack	386.22	0.62	2. frodo_pack	386.22	17.67
6. frodo_unpack	276.00	0.44	3. frodo_unpack	276.00	12.62
Others	195.62	0.32	Others	195.62	8.95
Total	62,075.83	100.00	Total	2,186.23	100.00
<b>FrodoKEM1344 - Decapsulation</b>					
1. frodo_mul_add_sa_plus_e	58,754.02	94.19	1.1 frodo_mul_add_sa_plus_e	1,316.52	42.20
2. Shake128 and frodo_sample_n x3	883.14	1.42	1.2 Shake128 and frodo_sample_n		
3. frodo_unpack x3	765.56	1.23	1.3 frodo_mul_add_sb_plus_e		
4. frodo_mul_add_sb_plus_e	649.68	1.04	1.4 Shake256		
5. frodo_mul_bs	507.08	0.81	2. frodo_unpack x3	765.56	24.54
6. Shake256	286.64	0.46	3. frodo_mul_bs	507.08	16.25
Others	530.74	0.85	Others	530.74	17.01
Total	62,376.86	100.00	Total	3,119.90	100.00
<b>FrodoKEM976 - Encapsulation</b>					
1. frodo_mul_add_sa_plus_e	31,430.38	90.82	1.1 frodo_mul_add_sa_plus_e	760.74	46.32
2. Shake128 and frodo_sample_n x3	1,410.18	4.07	1.2 Shake128 and frodo_sample_n		
3. frodo_mul_add_sb_plus_e	472.16	1.36	1.3 frodo_mul_add_sb_plus_e		
4. Shake256	414.11	1.20	1.4 Shake256		
5. frodo_pack	357.58	1.03	2. frodo_pack	357.58	21.77
6. frodo_unpack	297.73	0.86	3. frodo_unpack	297.73	18.13
Others	226.40	0.65	Others	226.40	13.78
Total	34,608.54	100.00	Total	1,642.45	100.00
<b>FrodoKEM976 - Decapsulation</b>					
1. frodo_mul_add_sa_plus_e	31,441.14	90.74	1.1 frodo_mul_add_sa_plus_e	749.76	40.18
2. Shake128 and frodo_sample_n x3	1,410.86	4.07	1.2 Shake128 and frodo_sample_n		
3. frodo_unpack x3	594.63	1.72	1.3 frodo_mul_add_sb_plus_e		
4. frodo_mul_add_sb_plus_e	471.29	1.36	1.4 Shake256		
5. frodo_mul_bs	368.32	1.06	2. frodo_unpack x3	594.63	31.86
6. Shake256	208.83	0.60	3. frodo_mul_bs	368.32	19.74
Others	153.51	0.44	Others	153.51	8.23
Total	34,648.58	100.00	Total	1,866.22	100.00
<b>FrodoKEM640 - Encapsulation</b>					
1. frodo_mul_add_sa_plus_e	13,794.27	85.19	1.1 frodo_mul_add_sa_plus_e	352.52	28.82
2. Shake128 and frodo_sample_n x3	1,002.40	6.19	1.2 Shake128 and frodo_sample_n		
3. frodo_mul_add_sb_plus_e	309.68	1.91	1.3 frodo_mul_add_sb_plus_e		
4. Shake256	215.55	1.33	1.4 Shake256		
5. frodo_pack	291.83	1.80	2. frodo_pack	291.83	23.86
6. frodo_unpack	277.26	1.71	3. frodo_unpack	277.26	22.67
Others	301.38	1.86	Others	301.38	24.64
Total	16,192.37	100.00	Total	1,222.99	100.00
<b>FrodoKEM640 - Decapsulation</b>					
1. frodo_mul_add_sa_plus_e	13,793.01	85.18	1.1 frodo_mul_add_sa_plus_e	342.95	25.95
2. Shake128 and frodo_sample_n x3	1,002.85	6.19	1.2 Shake128 and frodo_sample_n		
3. frodo_unpack x3	548.74	3.39	1.3 frodo_mul_add_sb_plus_e		
4. frodo_mul_add_sb_plus_e	309.21	1.91	1.4 Shake256		
5. frodo_mul_bs	242.40	1.50	2. frodo_unpack x3	548.74	41.53
6. Shake256	108.93	0.67	3. frodo_mul_bs	242.40	18.35
Others	187.23	1.16	Others	187.23	14.17
Total	16,192.37	100.00	Total	1,321.32	100.00



**Table 30:** Results of profiling Kyber

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
<b>Software</b>			<b>Software/Hardware</b>		
<b>Kyber CCA-KEM 1024 - Encapsulation</b>					
1. indcpa_enc	736.7	93.55	1.1. indcpa_enc	20.5	93.18
2. hash	49.3	6.26	1.2. hash		
3. randombytes	1.5	0.19	2. randombytes	1.5	6.82
Total	787.5	100.00	Total	22.0	100.00
<b>Kyber CCA-KEM 1024 - Decapsulation</b>					
1. indcpa_enc	734.2	76.99	1.1 indcpa_enc	24.7	100.00
2. indcpa_dec	191.7	20.10	1.2 indcpa_dec		
3. hash & verify	27.7	2.90	1.3 hash & verify		
Total	953.7	100.00	Total	24.7	100.00
<b>Kyber CCA-KEM 768 - Encapsulation</b>					
1. indcpa_enc	496.3	92.48	1.1. indcpa_enc	16.3	91.58
2. hash	38.9	7.24	1.2. hash		
3. randombytes	1.5	0.28	2. randombytes	1.5	8.42
Total	536.7	100.00	Total	17.8	100.00
<b>Kyber CCA-KEM 768 - Decapsulation</b>					
1. indcpa_enc	493.2	73.60	1.1 indcpa_enc	20.1	100.00
2. indcpa_dec	154.6	23.07	1.2 indcpa_dec		
3. hash & verify	22.3	3.33	1.3 hash & verify		
Total	670.1	100.00	Total	20.1	100.00
<b>Kyber CCA-KEM 512 - Encapsulation</b>					
1. indcpa_enc	302.8	91.19	1.1. indcpa_enc	13.7	90.10
2. hash	27.8	8.36	1.2. hash		
3. randombytes	1.5	0.45	2. randombytes	1.5	9.90
Total	332.0	100.00	Total	15.2	100.00
<b>Kyber CCA-KEM 512 - Decapsulation</b>					
1. indcpa_enc	298.5	68.93	1.1 indcpa_enc	17.1	100.00
2. indcpa_dec	117.9	27.22	1.2 indcpa_dec		
3. hash & verify	16.7	3.85	1.3 hash & verify		
Total	433.0	100.00	Total	17.1	100.00

**Table 31:** Results of profiling LAC-v3a

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			Software/Hardware		
<b>LAC-v3a-256 - Encapsulation</b>					
1. pke_enc_seed	901.0	99.42	1.1 pke_enc_seed	22.3	93.69
2. hash_to_k	2.3	0.26	1.2 hash_to_k		
3. random_bytes	1.5	0.17	1.3 Others		
Others	1.4	0.16	2. random_bytes	1.5	6.31
Total	906.3	100.00	Total	23.8	100.00
<b>LAC-v3a-256 - Decapsulation</b>					
1. pke_enc_seed	901.1	65.37	1.1 pke_enc_seed	26.9	100.00
2. pke_dec	472.4	34.27	1.2 pke_dec		
3. hash_to_k	2.3	0.17	1.3 hash_to_k		
Others	2.6	0.19	1.4 Others		
Total	1,378.4	100.00	Total	26.9	100.00
<b>LAC-v3a-192 - Encapsulation</b>					
1. pke_enc_seed	558.5	99.06	1.1 pke_enc_seed	19.9	92.99
2. hash_to_k	2.3	0.41	1.2 hash_to_k		
3. random_bytes	1.5	0.27	1.3 Others		
Others	1.5	0.26	2. random_bytes	1.5	7.01
Total	563.8	100.00	Total	21.4	100.00
<b>LAC-v3a-192 - Decapsulation</b>					
1. pke_enc_seed	558.7	71.44	1.1 pke_enc_seed	23.7	100.00
2. pke_dec	218.6	27.95	1.2 pke_dec		
3. hash_to_k	2.3	0.30	1.3 hash_to_k		
Others	2.5	0.31	1.4 Others		
Total	782.1	100.00%	Total	23.7	100.00
<b>LAC-v3a-128 - Encapsulation</b>					
1. pke_enc_seed	328.6	98.73	1.1 pke_enc_seed	14.9	93.73
2. hash_to_k	2.3	0.69	1.2 hash_to_k		
3. random_bytes	1.0	0.30	1.3 Others		
Others	0.9	0.27	2. random_bytes	1.0	6.27
Total	332.8	100.00	Total	15.9	100.00
<b>LAC-v3a-128 - Decapsulation</b>					
1. pke_enc_seed	328.5	71.01	1.1 pke_enc_seed	17.1	100.00
2. pke_dec	130.3	28.17	1.2 pke_dec		
3. hash_to_k	2.3	0.50	1.3 hash_to_k		
Others	1.5	0.32	1.4 Others		
Total	462.6	100.00	Total	17.1	100.00

**Table 32:** Results of profiling LAC-v3b

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
<b>Software</b>			<b>Software/Hardware</b>		
<b>LAC-v3b-256 - Encapsulation</b>					
1. pke_enc_seed	861.8	99.05	1.1 pke_enc_seed	19.6	92.88
2. hash_to_k	2.3	0.27	1.2 hash_to_k		
3. random_bytes	1.5	0.17	1.3 Others		
Others	4.4	0.51	2. random_bytes	1.5	7.12
Total	870.0	100.00	Total	21.1	100.00
<b>LAC-v3b-256 - Decapsulation</b>					
1. pke_enc_seed	861.6	64.69	1.1 pke_enc_seed	24.1	100.00
2. pke_dec	463.3	34.78	1.2 pke_dec		
3. hash_to_k	2.3	0.17	1.3 hash_to_k		
Others	4.7	0.35	1.4 Others		
Total	1,331.9	100.00	Total	24.1	100.00
<b>LAC-v3b-192 - Encapsulation</b>					
1. pke_enc_seed	522.7	98.54	1.1 pke_enc_seed	16.8	91.82
2. hash_to_k	2.3	0.44	1.2 hash_to_k		
3. random_bytes	1.5	0.28	1.3 Others		
Others	3.9	0.74	2. random_bytes	1.5	8.18
Total	530.4	100.00	Total	18.3	100.00
<b>LAC-v3b-192 - Decapsulation</b>					
1. pke_enc_seed	522.8	70.59	1.1 pke_enc_seed	20.6	100.00
2. pke_dec	211.2	28.51	1.2 pke_dec		
3. hash_to_k	2.3	0.31	1.3 hash_to_k		
Others	4.4	0.59	1.4 Others		
Total	740.7	100.00	Total	20.6	100.00
<b>LAC-v3b-128 - Encapsulation</b>					
1. pke_enc_seed	309.4	98.34	1.1 pke_enc_seed	13.4	93.07
2. hash_to_k	2.3	0.73	1.2 hash_to_k		
3. random_bytes	1.0	0.32	1.3 Others		
Others	1.9	0.61	2. random_bytes	1.0	6.93
Total	314.6	100.00	Total	14.4	100.00
<b>LAC-v3b-128 - Decapsulation</b>					
1. pke_enc_seed	309.3	70.25	1.1 pke_enc_seed	15.5	100.00
2. pke_dec	126.3	28.69	1.2 pke_dec		
3. hash_to_k	2.3	0.52	1.3 hash_to_k		
Others	2.3	0.53	1.4 Others		
Total	440.2	100.00	Total	15.5	100.00

**Table 33:** Results of profiling NewHope

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
<b>Software</b>			<b>Software/Hardware</b>		
<b>NewHope CCA-KEM 1024 - Encapsulation</b>					
1. cpapke_enc	668.3	91.33	1.1. cpapke_enc	19.7	92.92
2. hash	62.0	8.47	1.2. hash		
3. randombytes	1.5	0.20	2. randombytes	1.5	7.08
Total	731.7	100.00	Total	21.2	100.00
<b>NewHope CCA-KEM 1024 - Decapsulation</b>					
1. cpapke_enc	660.7	74.01	1.1 cpapke_enc	24.8	100.00
2. cpapke_dec	193.9	21.72	1.2 cpapke_dec		
3. hash & verify	38.2	4.27	1.3 hash & verify		
Total	892.7	100.00	Total	24.8	100.00
<b>NewHope CCA-KEM 512 - Encapsulation</b>					
1. cpapke_enc	316.9	89.60	1.1. cpapke_enc	13.5	89.98
2. hash	35.3	9.97	1.2. hash		
3. randombytes	1.5	0.42	2. randombytes	1.5	10.02
Total	353.6	100.00	Total	15.0	100.00
<b>NewHope CCA-KEM 512 - Decapsulation</b>					
1. cpapke_enc	311.8	72.92	1.1 cpapke_enc	16.1	100.00
2. cpapke_dec	93.3	21.82	1.2 cpapke_dec		
3. hash & verify	22.5	5.26	1.3 hash & verify		
Total	427.5	100.00	Total	16.1	100.00

**Table 34:** Results from profiling NTRU

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			Software/Hardware		
<b>NTRU HPS 4096821- Encapsulation</b>					
1. poly_Rq_mul	3,954.9	92.29	1.1 poly_Rq_mul	29.8	63.77
2. owcpa_samplemsg	251.4	5.87	1.2 owcpa_samplemsg		
3. shake256	54.3	1.27	1.3 shake256		
4. sha3_256	7.6	0.18	1.4 sha3_256		
Others	16.9	0.39	Others	16.9	36.23
Total	4,285.1	100.00	Total	46.7	100.00
<b>NTRU HPS 4096821 - Decapsulation</b>					
1. poly_S3_mul	3,972.1	33.15	1.1 poly_Rq_mul	37.4	34.92
2. poly_Sq_mul	3,960.3	33.05	1.2 poly_S3_mul		
3. poly_Rq_mul	3,955.4	33.01	1.3 poly_Sq_mul		
4. poly_S3_frombytes x2	31.1	0.26	1.4 sha3_256 x2		
5. sha3_256 x2	24.2	0.20	2. poly_S3_frombytes x2	31.1	29.01
Others	38.6	0.32	Others	38.6	36.07
Total	11,981.7	100.00	Total	107.1	100.00
<b>NTRU HPS 2048677- Encapsulation</b>					
1. poly_Rq_mul	2,692.6	90.92	2.1 poly_Rq_mul	26.0	62.44
2. owcpa_samplemsg	199.8	6.75	2.2 owcpa_samplemsg		
3. shake256	45.9	1.55	2.3 shake256		
4. sha3_256	7.5	0.25	2.4 sha3_256		
Others	15.6	0.53	Others	15.6	37.56
Total	2,961.5	100.00	Total	41.6	100.00
<b>NTRU HPS 2048677 - Decapsulation</b>					
1. poly_S3_mul	2,706.8	33.11	1.1 poly_Rq_mul	34.1	35.77
2. poly_Sq_mul	2,693.2	32.94	1.2 poly_S3_mul		
3. poly_Rq_mul	2,693.1	32.94	1.3 poly_Sq_mul		
4. poly_S3_frombytes x2	25.9	0.32	1.4 sha3_256 x2		
5. sha3_256	20.6	0.25	2. poly_S3_frombytes x2	25.9	27.14
Others	35.3	0.43	Others	35.3	37.10
Total	8,174.9	100.00	Total	95.3	100.00
<b>NTRU-HRSS - Encapsulation</b>					
1. poly_Rq_mul	2,886.1	97.36	1. poly_lift	27.9	42.74
2. shake256	24.2	0.82	2.1 poly_Rq_mul	22.3	34.12
3. poly_lift	27.9	0.94	2.2 owcpa_samplemsg		
4. sha3_256	7.6	0.26	2.3 shake256		
5. owcpa_samplemsg	3.5	0.12	2.3 sha3_256		
Others	15.1	0.51	Others	15.1	23.14
Total	2,964.5	100.00	Total	65.3	100.00
<b>NTRU-HRSS - Decapsulation</b>					
1. poly_S3_mul	2,900.8	33.00	1.1 poly_Rq_mul	46.4	34.17
2. poly_Sq_mul	2,890.7	32.89	1.2 poly_S3_mul		
3. poly_Rq_mul	2,886.6	32.84	1.3 poly_Sq_mul		
4. poly_lift	27.2	0.31	1.4 sha3_256		
5. sha3_256	22.3	0.25	2. poly_lift	27.2	20.05
Others	62.1	0.71	Others	62.1	45.78
Total	8,789.8	100.00	Total	135.6	100.00

**Table 35:** Results of profiling NTRULPrime

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			Software/Hardware		
<b>NTRULPrime857 - Encapsulation</b>					
1. Rq_mult_small x2	1,448.9	70.32	1.1 Short_fromlist	69.6	70.97
2. Short_fromlist	261.6	12.69	1.2 Expand x2		
3. Expand x2	245.4	11.91	1.3 Hash X4		
4. Hash x4	60.7	2.95	1.4 Rq_mult_small x2		
5. crypto_decode_857x1723	28.5	1.38	1.5.crypto_encode_857x1723round		
6. crypto_encode_857x1723round	4.7	0.23	1.6 Others		
Others	10.9	0.53	2. crypto_decode_857x1723	28.5	29.03
Total	2,060.5	100.00	Total	98.0	100.00
<b>NTRULPrime857 - Decapsulation</b>					
1. Rq_mult_small x3	2,173.5	78.00	1.1 Short_fromlist	47.3	48.52
2. Short_fromlist	261.4	9.38	1.2 expand x2		
3. expand x2	246.5	8.85	1.3 Rq_mult_small x3		
4. crypto_decode_857x1723 x2	50.2	1.80	1.4 Hash x3		
5. Hash x3	34.0	1.22	1.6 crypto_encode_857x1723round		
6. crypto_encode_857x1723round	4.8	0.17	1.7 Others		
Others	16.3	0.58	2. crypto_decode_857x1723 x2		
Total	2,786.6	100.00	Total	97.5	100.00
<b>NTRULPrime761 - Encapsulation</b>					
1. Rq_mult_small x2	1,169.5	68.62	1.1 Short_fromlist	66.0	67.07
2. Short_fromlist	226.2	13.27	1.2 Expand x2		
3. Expand x2	214.8	12.60	1.3 Hash X4		
4. Hash X4	54.4	3.19	1.4 Rq_mult_small x2		
5. crypto_decode_761x1531	25.7	1.51	1.5 crypto_encode_761x1531round		
6. crypto_encode_761x1531round	4.2	0.25	1.6 Others		
Others	9.5	0.56	2. crypto_decode_761x1531	32.4	32.93
Total	1,704.4	100.00	Total	98.4	100.00
<b>NTRULPrime761 - Decapsulation</b>					
1. Rq_mult_small x3	1,753.6	76.60	1.1 Short_fromlist	71.4	54.20
2. Short_fromlist	225.9	9.87	1.2 expand x2		
3. Expand x2	214.9	9.39	1.3 Rq_mult_small x3		
3. crypto_decode_761x1531 x2	45.4	1.98	1.4 Hash x3		
4. Hash x3	31.0	1.35	1.5 crypto_encode_761x1531round		
5. crypto_encode_761x1531round	4.3	0.19	1.6 Others		
Others	14.2	0.62	2. crypto_decode_761x1531 x2		
Total	2,289.4	100.00	Total	131.7	100.00
<b>NTRULPrime653 - Encapsulation</b>					
1. Rq_mult_small x2	934.0	67.19	1.1 Short_fromlist	58.6	67.90
2. Short_fromlist	190.0	13.67	1.2 Expand x2		
3. Expand x2	183.6	13.21	1.3 Rq_mult_small x2		
4. Hash x4	48.2	3.46	1.4 Hash x4		
5. crypto_decode_653x1541	22.7	1.64	1.5 crypto_encode_653x1541round		
6. crypto_encode_653x1541round	3.6	0.26	1.6 Others		
Others	8.1	0.58	2. crypto_decode_653x1541	27.7	32.10
Total	1,390.2	100.00	Total	86.3	100.00
<b>NTRULPrime653 - Decapsulation</b>					
1. Rq_mult_small x3	1,400.9	75.50	1.1 Short_fromlist	64.2	55.60
2. Short_fromlist	187.9	10.13	1.2 Expand x2		
3. Expand x2	183.7	9.90	1.3 Rq_mult_small x3		
4. crypto_decode_653x1541 x2	38.8	2.09	1.4 Hash x3		
5. Hash x3	27.6	1.49	1.5 crypto_encode_653x1541round		
6. crypto_encode_653x1541round	3.7	0.20	1.6 Others		
Others	12.9	0.70	2. crypto_decode_653x1541 x2		
Total	1,855.5	100.00	Total	115.5	100.00

**Table 36:** Results of profiling Streamlined NTRU Prime

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			Software/Hardware		
<b>StreamlinedNTRUPrime857 - Encapsulation</b>					
1. Rq_mult_small	724.5	63.36	1.1 crypto_sort_uint32	60.1	62.95
2. crypto_sort_uint32	259.7	22.71	1.2 Hash x5		
3. Hash x5	71.9	6.29	1.3 Rq_mult_small		
4. Rq_decode	30.4	2.66	1.4 Round_and_encode		
5. Round_and_encode	8.2	0.72	1.5 Others		
Others	48.8	4.27	2. Rq_decode	35.4	37.05
Total	1,143.5	100.00	Total	95.5	100.00
<b>StreamlineNTRUPrime857 - Decapsulation</b>					
1. R3_mult	1,019.4	39.43	1.1 Hash x4	57.3	48.39
2. Rq_mult_small x2	1,448.9	56.05	1.2 Rq_mult_small x2		
3. Hash x4	42.1	1.63	1.3 R3_mult		
4. Rq_decode	27.9	1.08	1.4 Others		
5. Rounded_decode	28.1	1.09	2. Rq_decode	33.0	27.90
Others	18.9	0.73	3. Rounded_decode	28.1	23.71
Total	2,585.2	100.00	Total	118.4	100.00
<b>StreamlineNTRUPrime761 - Encapsulation</b>					
1. Rq_mult_small	584.6	61.65	1.1 crypto_sort_uint32	56.3	63.97
2. crypto_sort_uint32	223.8	23.61	1.2 Hash x5		
3. Hash x5	62.5	6.59	1.3 Rq_mult_small		
4. Rq_decode	27.0	2.85	1.4. Round_and_encode		
5. Round_and_encode	7.7	0.81	1.5 Others		
Others	42.7	4.50	2. Rq_decode	31.7	36.03
Total	948.2	100.00	Total	88.0	100.00
<b>StreamlineNTRUPrime761- Decapsulation</b>					
1. R3_mult	816.2	39.06	1.1 Hash x4	53.3	59.35
2. Rq_mult_small x2	1,169.4	55.96	1.2 Rq_mult_small x2		
3. Hash x4	35.8	1.72	1.3 R3_mult		
4. Rq_decode	24.5	1.17	1.4 Others		
5. Rounded_decode	25.8	1.24	2. Rq_decode	32.3	35.97
Others	17.8	0.85	3. Rounded_decode	4.2	4.68
Total	2,089.6	100.00	Total	89.8	100.00
<b>StreamlineNTRUPrime653 - Encapsulation</b>					
1. Rq_mult_small	467.0	60.19	1.1 crypto_sort_uint32	52.3	65.60
2. crypto_sort_uint32	185.5	23.90	1.2 Hash x5		
3. Hash x5	54.8	7.06	1.3 Rq_mult_small		
4. Rq_decode	24.2	3.11	1.4 Round_and_encode		
5. Round_and_encode	6.4	0.82	1.5 Others		
Others	38.2	4.92	2. Rq_decode	27.4	34.40
Total	775.9	100.00	Total	79.7	100.00
<b>StreamlineNTRUPrime653- Decapsulation</b>					
1. R3_mult	617.3	37.58	1.1 Hash x4	51.0	63.88
2. Rq_mult_small x2	933.7	56.85	1.2 Rq_mult_small x2		
3. Hash x4	35.7	2.17	1.3 R3_mult		
4. Rq_decode	21.0	1.28	1.4 Others		
5. Rounded_decode	22.4	1.37	2. Rq_decode	25.2	31.61
Others	12.4	0.76	3. Rounded_decode	3.6	4.51
Total	1,642.5	100.00	Total	79.8	100.00

**Table 37:** Results of profiling Round5

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			Software/Hardware		
<b>R5ND_CCA_5KEM_0d - Encapsulation</b>					
1. r5_cpa_pke_encrypt	290.8	86.42	1.1. r5_cpa_pke_encrypt	30.1	95.20
2. hash	44.2	13.13	1.2. hash		
3. randombytes	1.5	0.45	2. randombytes	1.5	4.80
Total	336.5	100.00	Total	31.6	100.00
<b>R5ND_CCA_5KEM_0d - Decapsulation</b>					
1. r5_cpa_pke_encrypt	287.1	69.05	1.1 r5_cpa_pke_encrypt	36.8	100.00
2. r5_cpa_pke_decrypt	83.6	20.11	1.2 r5_cpa_pke_decrypt		
3. hash & verify	45.1	10.84	1.3 hash & verify		
Total	415.8	100.00	Total	36.8	100.00
<b>R5ND_CCA_3KEM_0d - Encapsulation</b>					
1. r5_cpa_pke_encrypt	211.4	86.27	1.1. r5_cpa_pke_encrypt	22.6	95.72
2. hash	32.6	13.32	1.2. hash		
3. randombytes	1.0	0.41	2. randombytes		
Total	245.0	100.00	Total	23.6	100.00
<b>R5ND_CCA_3KEM_0d - Decapsulation</b>					
1. r5_cpa_pke_encrypt	208.2	67.27	1.1 r5_cpa_pke_encrypt	27.6	100.00
2. r5_cpa_pke_decrypt	67.5	21.80	1.2 r5_cpa_pke_decrypt		
3. hash & verify	33.9	10.94	1.3 hash & verify		
Total	309.5	100.00	Total	27.6	100.00
<b>R5ND_CCA_1KEM_0d - Encapsulation</b>					
1. r5_cpa_pke_encrypt	133.9	86.69	1.1. r5_cpa_pke_encrypt	16.9	94.46
2. hash	19.6	12.67	1.2. hash		
3. randombytes	1.0	0.64	2. randombytes		
Total	154.5	100.00	Total	17.9	100.00
<b>R5ND_CCA_1KEM_0d - Decapsulation</b>					
1. r5_cpa_pke_encrypt	130.3	67.61	1.1 r5_cpa_pke_encrypt	19.7	100.00
2. r5_cpa_pke_decrypt	41.7	21.62	1.2 r5_cpa_pke_decrypt		
3. hash & verify	20.8	10.77	1.3 hash & verify		
Total	192.7	100.00	Total	19.7	100.00
<b>R5ND_CCA_5KEM_5d - Encapsulation</b>					
1. r5_cpa_pke_encrypt	372.0	91.87	1.1. r5_cpa_pke_encrypt	26.4	94.55
2. hash	31.4	7.76	1.2. hash		
3. randombytes	1.5	0.38	2. randombytes		
Total	404.9	100.00	Total	27.9	100.00
<b>R5ND_CCA_5KEM_5d - Decapsulation</b>					
1. r5_cpa_pke_encrypt	372.0	69.24	1.1 r5_cpa_pke_encrypt	32.5	100.00
2. r5_cpa_pke_decrypt	132.3	24.63	1.2 r5_cpa_pke_decrypt		
3. hash & verify	32.9	6.12	1.3 hash & verify		
Total	537.2	100.00	Total	32.5	100.00
<b>R5ND_CCA_3KEM_5d - Encapsulation</b>					
1. r5_cpa_pke_encrypt	214.3	88.83	1.1. r5_cpa_pke_encrypt	19.3	95.04
2. hash	25.9	10.75	1.2. hash		
3. randombytes	1.0	0.42	2. randombytes		
Total	241.3	100.00	Total	20.4	100.00
<b>R5ND_CCA_3KEM_5d - Decapsulation</b>					
1. r5_cpa_pke_encrypt	214.2	67.19	1.1 r5_cpa_pke_encrypt	23.3	100.00
2. r5_cpa_pke_decrypt	78.5	24.63	1.2 r5_cpa_pke_decrypt		
3. hash & verify	26.1	8.18	1.3 hash & verify		
Total	318.8	100.00	Total	23.3	100.00
<b>R5ND_CCA_1KEM_5d - Encapsulation</b>					
1. r5_cpa_pke_encrypt	111.7	88.59	1.1. r5_cpa_pke_encrypt	13.5	93.15
2. hash	13.4	10.62	1.2. hash		
3. randombytes	1.0	0.79	2. randombytes		
Total	126.1	100.00	Total	14.4	100.00
<b>R5ND_CCA_1KEM_5d - Decapsulation</b>					
1. r5_cpa_pke_encrypt	111.8	64.72	1.1 r5_cpa_pke_encrypt	16.0	100.00
2. r5_cpa_pke_decrypt	46.7	27.06	1.2 r5_cpa_pke_decrypt		
3. hash & verify	14.2	8.22	1.3 hash & verify		
Total	172.7	100.00	Total	16.0	100.00



**Table 38:** Results of profiling for Saber

Function	Time [us]	Time [%]	Function	Time [us]	Time [%]
Software			Software/Hardware		
<b>FireSaber - Encapsulation</b>					
1. MatrixVectorMul	815.84	69.08%	1.1 MatrixVectorMul	55.09	84.43%
2. InnerProduct	204.44	17.31%	1.2 InnerProduct		
3. GenMatrix	92.93	7.87%	1.3 GenMatrix		
4. Hash	45.10	3.82%	1.4 Hash		
5. GenSecret	12.50	1.06%	1.5 GenSecret		
Others	10.16	0.86%	Others	10.16	15.57%
Total	1,180.97	100.00%	Total	65.25	100.00%
<b>FireSaber - Decapsulation</b>					
1. MatrixVectorMul	816.34	59.31%	1.1 MatrixVectorMul	55.14	71.55%
2. InnerProduct x2	408.14	29.65%	1.2 InnerProduct x2		
3. GenMatrix	92.99	6.76%	1.3 GenMatrix		
4. Hash	24.49	1.78%	1.4 Hash		
5. GenSecret	12.53	0.91%	1.5 GenSecret		
Others	21.92	1.59%	Others	21.92	28.45%
Total	1,376.41	100.00%	Total	77.06	100.00%
<b>Saber - Encapsulation</b>					
1. MatrixVectorMul	458.94	63.55%	1.1 MatrixVectorMul	49.15	86.36%
2. InnerProduct	153.19	21.21%	1.2 InnerProduct		
3. GenMatrix	53.29	7.38%	1.3 GenMatrix		
4. Hash	37.98	5.26%	1.4 Hash		
5. GenSecret	10.97	1.52%	1.5 GenSecret		
Others	7.76	1.07%	Others	7.76	13.64%
Total	722.13	100.00%	Total	56.91	100.00%
<b>Saber - Decapsulation</b>					
1. MatrixVectorMul	458.98	52.93%	1.1 MatrixVectorMul	48.15	74.47%
2. InnerProduct x2	306.52	35.35%	1.2 InnerProduct x2		
3. GenMatrix	53.29	6.15%	1.3 GenMatrix		
4. Hash	20.87	2.41%	1.4 Hash		
5. GenSecret	11.00	1.27%	1.5 GenSecret		
Others	16.51	1.90%	Others	16.51	25.53%
Total	867.17	100.00%	Total	64.66	100.00%
<b>LightSaber - Encapsulation</b>					
1. MatrixVectorMul	203.70	54.55%	1.1 MatrixVectorMul	43.36	88.49%
2. InnerProduct	102.26	27.38%	1.2 InnerProduct		
3. GenMatrix	23.67	6.34%	1.3 GenMatrix		
4. Hash	27.31	7.31%	1.4 Hash		
5. GenSecret	10.86	2.91%	1.5 GenSecret		
Others	5.64	1.51%	Others	5.64	11.51%
Total	373.44	100.00%	Total	49.00	100.00%
<b>LightSaber - Decapsulation</b>					
1. MatrixVectorMul	204.43	43.44%	1.1 MatrixVectorMul	41.27	78.55%
2. InnerProduct x2	204.80	43.52%	1.2 InnerProduct x2		
3. GenMatrix	23.67	5.03%	1.3 GenMatrix		
4. Hash	15.55	3.30%	1.4 Hash		
5. GenSecret	10.83	2.30%	1.5 GenSecret		
Others	11.27	2.40%	Others	11.27	21.45%
Total	470.55	100.00%	Total	52.54	100.00%

## 1771 B Pseudocode of Implemented Algorithms

1772 Below we show the pseudocode of all implemented KEMs, with parts offloaded to hardware  
1773 marked with the gray background.

### 1774 B.1 FrodoKEM

---

**Algorithm 1** Pseudocode of FrodoKEM.Encaps [70]

---

**Input:** Public key  $pk = seed_A || b \in \{0, 1\}^{len_{seed_A} + D \cdot n \cdot \bar{n}}$ .

**Output:** Ciphertext  $c_1 || c_2 \in \{0, 1\}^{(\bar{m} \cdot n + \bar{m} \cdot \bar{n})D}$  and shared secret  $SS \in \{0, 1\}^{len_{ss}}$ .

---

- 1: Choose a uniformly random key  $\mu \leftarrow_s U(\{0, 1\}^{len_\mu})$
  - 2: Compute  $\mathbf{pkh} \leftarrow \text{SHAKE}(pk, len_{pkh})$
  - 3: Generate pseudorandom values  $seed_{SE} || k \leftarrow \text{SHAKE}(\mathbf{pkh} || \mu, len_{seed_{SE}} + len_k)$
  - 4: Generate pseudorandom bit string  $(r^{(0)}, r^{(1)}, \dots, r^{(2\bar{m}n + \bar{m}\bar{n} - 1)}) \leftarrow \text{SHAKE}(0x96 || seed_{SE}, 2\bar{m}n + \bar{m}\bar{n} \cdot len_x)$
  - 5: Sample error matrix  $\mathbf{S}' \leftarrow \text{Frodo.SampleMatrix}((r^{(0)}, r^{(1)}, \dots, r^{(\bar{m}n - 1)}), \bar{m}, n, T_x)$
  - 6: Sample error matrix  $\mathbf{E}' \leftarrow \text{Frodo.SampleMatrix}((r^{(\bar{m}n)}, r^{(\bar{m}n + 1)}, \dots, r^{(2\bar{m}n - 1)}), \bar{m}, n, T_x)$
  - 7: Generate  $\mathbf{A} \leftarrow \text{Frodo.Gen}(seed_A)$
  - 8: Compute  $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
  - 9: Compute  $\mathbf{c}_1 \leftarrow \text{Frodo.Pack}(\mathbf{B}')$
  - 10: Sample error matrix  $\mathbf{E}'' \leftarrow \text{Frodo.SampleMatrix}((r^{(2\bar{m}n)}, r^{(2\bar{m}n + 1)}, \dots, r^{(2\bar{m}n + \bar{m}\bar{n} - 1)}), \bar{m}, \bar{n}, T_x)$
  - 11: Compute  $\mathbf{B} \leftarrow \text{Frodo.Unpack}(\mathbf{c}_1, n, \bar{n})$
  - 12: Compute  $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
  - 13: Compute  $\mathbf{C} \leftarrow \mathbf{V} + \text{Frodo.Encode}(\mu)$
  - 14: Compute  $\mathbf{c}_2 \leftarrow \text{Frodo.Pack}(\mathbf{C})$
  - 15: Compute  $\mathbf{ss} \leftarrow \text{SHAKE}(\mathbf{c}_1 || \mathbf{c}_2 || \mathbf{k}, len_{ss})$
  - 16: **return** ciphertext  $(\mathbf{c}_1 || \mathbf{c}_2)$  and shared secret  $\mathbf{ss}$
-

**Algorithm 2** Pseudocode of FrodoKEM.Decaps [70]

**Input:** Ciphertext  $\mathbf{c}_1 || \mathbf{c}_2 \in \{0, 1\}^{\bar{m} \cdot n + \bar{m} \cdot \bar{n}}^D$ , secret key  $sk' = (s || seed_A || \mathbf{b}, \mathbf{S}, \mathbf{pkh}) \in \{0, 1\}^{len_s + len_{seed_A} + D \cdot n \cdot \bar{n}} \times Z_q^{n \times \bar{n}} \times \{0, 1\}^{len_{pkh}}$ .  
**Output:** Shared secret  $\mathbf{ss} \in \{0, 1\}^{len_{ss}}$ .

---

```

1:  $\mathbf{B}' \leftarrow \text{Frodo.Unpack}(c_1)$ 
2:  $\mathbf{C} \leftarrow \text{Frodo.Unpack}(c_2)$ 
3: Compute  $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S}$ 
4: Compute  $\mu' \leftarrow \text{Frodo.Decode}(\mathbf{M})$ 
5: Parse  $pk \leftarrow seed_A || \mathbf{b}$ 
6: Generate pseudorandom values  $seed_{SE'} || k' \leftarrow \text{SHAKE}(\mathbf{pkh} || \mu', len_{seed_{SE'}} + len_k)$ 
7: Generate pseudorandom bit string  $(r^{(0)}, r^{(1)}, \dots, r^{(2\bar{m}n + \bar{m}\bar{n} - 1)}) \leftarrow \text{SHAKE}(0x96 || seed'_{se}, 2\bar{m}n + \bar{m}\bar{n}.len_x)$ 
8: Sample error matrix  $\mathbf{S}' \leftarrow \text{Frodo.SampleMatrix}((r^{(0)}, r^{(1)}, \dots, r^{(\bar{m}n - 1)}), \bar{m}, n, T_x)$ 
9: Sample error matrix  $\mathbf{E}' \leftarrow \text{Frodo.SampleMatrix}((r^{(\bar{m}n)}, r^{(\bar{m}n + 1)}, \dots, r^{(2\bar{m}n - 1)}), \bar{m}, n, T_x)$ 
10: Generate  $\mathbf{A} \leftarrow \text{Frodo.Gen}(seed_A)$ 
11: Compute  $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$ 
12: Sample error matrix  $\mathbf{E}'' \leftarrow \text{Frodo.SampleMatrix}((r^{(2\bar{m}n)}, r^{(2\bar{m}n + 1)}, \dots, r^{(2\bar{m}n + \bar{m}\bar{n} - 1)}), \bar{m}, \bar{n}, T_x)$ 
13: Compute  $\mathbf{B} \leftarrow \text{Frodo.Unpack}(\mathbf{b}, n, \bar{n})$ 
14: Compute  $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$ 
15: Compute  $\mathbf{C}' \leftarrow \mathbf{V} + \text{Frodo.Encode}(\mu')$ 
16: if  $\mathbf{B}' || \mathbf{C} = \mathbf{B}'' || \mathbf{C}'$  then
17:   return shared secret  $\mathbf{ss} \leftarrow \text{SHAKE}(c_1 || c_2 || k', len_{ss})$ 
18: else
19:   return shared secret  $\mathbf{ss} \leftarrow \text{SHAKE}(c_1 || c_2 || s, len_{ss})$ 
20: end if

```

---

1775

**B.2 LAC****Algorithm 3** LAC-CCA Key Generation [52]

**Input:** Random  $seed$

**Output:**  $pk = (seed_{\vec{a}}, \vec{b}), sk = (\vec{s}, pk)$

---

```

1:  $(seed_{\vec{a}}, seed_{\vec{s}}, seed_{\vec{e}}) \leftarrow \text{GenSeed}(seed)$ 
2:  $\vec{a} \leftarrow \text{UniformSampl}(seed_{\vec{a}}) \in R_q$ 
3:  $\vec{s} \leftarrow \text{CBDSampl}(seed_{\vec{s}})$ 
4:  $\vec{e} \leftarrow \text{CBDSampl}(seed_{\vec{e}})$ 
5:  $\vec{b} \leftarrow \vec{a}\vec{s} + \vec{e} \in R_q$ 
6:  $pk = (seed_{\vec{a}}, \vec{b})$ 
7:  $sk = (\vec{s}, pk = (seed_{\vec{a}}, \vec{b}))$ 

```

---

**Algorithm 4** LAC-CCA Encapsulation [52]**Input:**  $pk = (seed_{\vec{a}}, \vec{b})$ , message  $\vec{m}$ **Output:** A ciphertext  $\vec{c}$  and a session key  $\vec{s}\vec{s}$ 


---

```

1:  $seed \leftarrow \text{GenSeed}(\vec{m}, pk)$ 
2:  $seeds \leftarrow \text{GenSeed}(seed)$ 
3:  $\vec{a} \leftarrow \text{UniformSampl}(seed_{\vec{a}}) \in R_q$ 
4:  $(\vec{r}, \vec{e}_1, \vec{e}_2) \leftarrow \text{CBDSampl}(seeds)$ 
5:  $\vec{c}_1 \leftarrow \vec{a}\vec{r} + \vec{e}_1 \in R_q$ 
6:  $\vec{m}' \leftarrow \text{BCH.Enc}(\vec{m}) \in \{0, 1\}^{l_v}$ 
7:  $\vec{m}'' \leftarrow \text{D2.Enc}(\vec{m}') \in \mathbb{Z}_q^{l_v}$ 
8:  $\vec{c}_2 \leftarrow \text{Compress}((\vec{b}\vec{r})_{l_v} + \vec{e}_2 + \vec{m}'')$ 
9:  $\vec{c} \leftarrow (\vec{c}_1, \vec{c}_2)$ 
10:  $\vec{s}\vec{s} \leftarrow \text{Hash}(\vec{m}, \vec{c})$ 

```

---

**Algorithm 5** LAC-CCA Decapsulation [52]**Input:**  $sk = (\vec{s}, pk = (seed_{\vec{a}}, \vec{b}))$ ,  $\vec{c} = (\vec{c}_1, \vec{c}_2)$ **Output:** A session key  $\vec{s}\vec{s}$ 


---

```

1:  $\vec{u} \leftarrow \vec{c}_1\vec{s} \in R_q$ 
2:  $\vec{m}'' \leftarrow \text{Decompress}(\vec{c}_2) - (\vec{u})_{l_v} \in \mathbb{Z}_q^{l_v}$ 
3:  $\vec{m}' \leftarrow \text{D2.Dec}(\vec{m}'') \in \{0, 1\}^{l_v}$ 
4:  $\vec{m} \leftarrow \text{BCH.Dec}(\vec{m}')$ 
5:  $seed \leftarrow \text{GenSeed}(\vec{m}, pk)$ 
6:  $seeds \leftarrow \text{GenSeed}(seed)$ 
7:  $\vec{a} \leftarrow \text{UniformSampl}(seed_{\vec{a}}) \in R_q$ 
8:  $(\vec{r}, \vec{e}_1, \vec{e}_2) \leftarrow \text{CBDSampl}(seeds)$ 
9:  $\vec{c}_1 \leftarrow \vec{a}\vec{r} + \vec{e}_1 \in R_q$ 
10:  $\vec{m}' \leftarrow \text{BCH.Enc}(\vec{m}) \in \{0, 1\}^{l_v}$ 
11:  $\vec{m}'' \leftarrow \text{D2.Enc}(\vec{m}') \in \mathbb{Z}_q^{l_v}$ 
12:  $\vec{c}_2 \leftarrow \text{Compress}((\vec{b}\vec{r})_{l_v} + \vec{e}_2 + \vec{m}'')$ 
13:  $\vec{c} \leftarrow (\vec{c}_1, \vec{c}_2)$ 
14: if  $\vec{c} = \vec{c}$  then
15:    $\vec{s}\vec{s} \leftarrow \text{Hash}(\vec{m}, \vec{c})$ 
16: else
17:    $\vec{s}\vec{s} \leftarrow \text{Hash}(\text{Hash}(sk), \vec{c})$ 
18: end if

```

---

1776 **B.3 Kyber**


---

**Algorithm 6** Pseudocode of `KYBER.CCAKEM.Enc(pk)` [8]

---

**Input:** Public key  $pk \in \mathcal{B}^{12.k.n/8+32}$ **Output:** Ciphertext  $c \in \mathcal{B}^{d_u.k.n/8+d_v.n/8}$ **Output:** Shared key  $K \in \mathcal{B}^*$ 

```

1:  $m \leftarrow \mathcal{B}^{32}$ 
2:  $m \leftarrow H(m)$ 
3:  $(\bar{K}, r) := G(m || H(pk))$ 
4:  $c := \text{KYBER.CPAPKE.Enc}(pk, m, r)$ 
5:  $K := \text{KDF}(\bar{K} || H(c))$ 
6: return  $(c, K)$ 

```

---



---

**Algorithm 7** Pseudocode of `KYBER.CCAKEM.Dec(c, sk)` [8]

---

**Input:** Ciphertext  $c \in \mathcal{B}^{d_u.k.n/8+d_v.n/8}$ **Input:** Secret key  $sk \in \mathcal{B}^{24.k.n/8+96}$ **Output:** Shared key  $K \in \mathcal{B}^*$ 

```

1:  $pk := sk + 12.k.n/8$ 
2:  $h := sk + 24.k.n/8 + 32 \in \mathcal{B}^{32}$ 
3:  $z := sk + 24.k.n/8 + 64$ 
4:  $m' := \text{KYBER.CPAPKE.Dec}(c, (u, v))$ 
5:  $(\bar{K}', r') := G(m' || h)$ 
6:  $c' := \text{KYBER.CPAPKE.Enc}(pk, m', r')$ 
7: if  $c = c'$  then
8:   return  $K := \text{KDF}(\bar{K}' || H(c))$ 
9: else
10:  return  $K := \text{KDF}(z || H(c))$ 
11: end if
12: return  $K$ 

```

---

**Algorithm 8** Pseudocode of KYBER.CPAPKE.Enc( $pk, m, r$ ): encryption [8]**Input:** Public key  $pk \in \mathcal{B}^{12.k.n/8+32}$ **Input:** Message  $m \in \mathcal{B}^{32}$ **Input:** Random coins  $r \in \mathcal{B}^{32}$ **Output:** Ciphertext  $c \in \mathcal{B}^{d_u.k.n/8+d_v.n/8}$ 

```

1:  $N := 0$ 
2:  $\hat{t} := \text{Decode}_{12}(pk)$ 
3:  $\rho := pk + 12.k.n/8$ 
4: for  $i$  from 0 to  $k - 1$  do
5:   for  $j$  from 0 to  $k - 1$  do
6:      $\hat{A}^T[i][j] := \text{Parse}(\text{XOF}(\rho, i, j))$ 
7:   end for
8: end for
9: for  $i$  from 0 to  $k - 1$  do
10:   $r[i] := \text{CBD}_\eta(\text{PRF}(r, N))$ 
11:   $N := N + 1$ 
12: end for
13: for  $i$  from 0 to  $k - 1$  do
14:   $e_1[i] := \text{CBD}_\eta(\text{PRF}(r, N))$ 
15:   $N := N + 1$ 
16: end for
17:  $e_2 := \text{CBD}_\eta(\text{PRF}(r, N))$ 
18:  $\hat{r} := \text{NTT}(r)$ 
19:  $u := \text{NTT}^{-1}(\hat{A}^T \circ \hat{r}) + e_1$ 
20:  $v := \text{NTT}^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + \text{Decompress}_a(\text{Decode}_1(m), 1)$ 
21:  $c_1 := \text{Encode}_{d_u}(\text{Compress}_a(u, d_u))$ 
22:  $c_2 := \text{Encode}_{d_v}(\text{Compress}_a(v, d_v))$ 
23: return  $c = (c_1 || c_2)$ 

```

**Algorithm 9** Pseudocode of KYBER.CPAPKE.Dec( $sk, c$ ): decryption [8]**Input:** Secret key  $sk \in \mathcal{B}^{12.k.n/8}$ **Input:** Ciphertext  $c \in \mathcal{B}^{d_u.k.n/8+d_v.n/8}$ **Output:** Message  $m \in \mathcal{B}^{32}$ 

```

1:  $u := \text{Decompress}_a(\text{Decode}_{d_u}(c), d_u)$ 
2:  $v := \text{Decompress}_a(\text{Decode}_{d_v}(c + d_u.k.n/8), d_v)$ 
3:  $\hat{s} := \text{Decode}_{12}(sk)$ 
4:  $m := \text{Encode}_1(\text{Compress}_a(v - \text{NTT}^{-1}(\hat{s}^T \circ \text{NTT}(u)), 1))$ 
5: return  $m$ 

```

1777 **B.4 NewHope****Algorithm 10** Pseudocode of NEWHOPE-CCA-KEM Encapsulation [2]

---

```

1: function NEWHOPE-CCA-KEM.ENCAPS( $pk$ )
2:    $coin \xleftarrow{\$} \{0, \dots, 255\}^{32}$ 
3:    $\mu \leftarrow \text{SHAE256}(32, coin) \in \{0, \dots, 255\}^{32}$ 
4:    $K || coin' || d \leftarrow \text{SHAE256}(96, \mu || \text{SHAE256}(32, pk)) \in \{0, \dots, 255\}^{32+32+32}$ 
5:    $c \leftarrow \text{NEWHOPE-CPA-PKE.ENCRYPT}(pk, \mu, coins')$ 
6:    $ss \leftarrow \text{SHAKE256}(32, K || \text{SHAKE256}(32, c || d))$ 
7:   return ( $\bar{c} = c || d, ss$ )

```

---

**Algorithm 11** Pseudocode of NEWHOPE-CCA-KEM Decapsulation [2]

---

```

1: function NEWHOPE-CCA-KEM.DECAPS( $\bar{c}, \bar{sk}$ )
2:    $c || d \leftarrow \bar{c} \in \{0, \dots, 255\}^{3n/8+7n/4+32}$ 
3:    $sk || pk || h || s \leftarrow \bar{sk} \in \{0, \dots, 255\}^{7n/4+7n/4+32+32+32}$ 
4:    $\mu' \leftarrow \text{CPA.Decrypt}(c, sk)$ 
5:    $K' || coin'' || d' \leftarrow \text{SHAKE256}(96, \mu' || h) \in \{0, \dots, 255\}^{32+32+32}$ 
6:   if  $c = \text{NewHope-CPA-PKE.Encrypt}(pk, \mu', coin'')$  and  $d = d'$  then
7:      $fail \leftarrow 0$ 
8:   else
9:      $fail \leftarrow 1$ 
10:   $K \leftarrow K'$ 
11:   $K \leftarrow s$ 
12:  return ( $ss = \text{SHAKE256}(32, K_{fail} || \text{SHAKE256}(32, c || d))$ )

```

---

**Algorithm 12** Pseudocode of NEWHOPE-CPA-PKE Encryption [2]

---

```

1: function NEWHOPE-CPA-PKE.ENCRYPT( $pk \in \{0, \dots, 255\}^{7n/4+32}, \mu \in \{0, \dots, 255\}^{32}$ ,
2:    $coin \in \{0, \dots, 255\}^{32}$ )
3:    $(\hat{b}, publicseed) \leftarrow \text{DecodePk}(pk)$ 
4:    $\hat{a} \leftarrow \text{GenA}(publicseed)$ 
5:    $s' \leftarrow \text{BitRev}(\text{Sample}(coin, 0))$ 
6:    $e' \leftarrow \text{BitRev}(\text{Sample}(coin, 1))$ 
7:    $e'' \leftarrow \text{Sample}(coin, 2)$ 
8:    $\hat{t} \leftarrow \text{NTT}(s')$ 
9:    $\hat{u} \leftarrow \hat{a} \circ \hat{t} + \text{NTT}(e')$ 
10:   $v \leftarrow \text{Encode}(\mu)$ 
11:   $v' \leftarrow \text{NTT}^{-1}(\hat{b} \circ \hat{t}) + e'' + v$ 
12:   $h \leftarrow \text{Compress}(v')$ 
13:  return  $c = \text{EncodeC}(\hat{u}, h)$ 

```

---

**Algorithm 13** Pseudocode of NEWHOPE-CPA-PKE Decryption [2]

---

```

1: function NEWHOPE-CPA-PKE.DECRYPT( $c \in \{0, \dots, 255\}^{7n/4+3n/8}, sk \in \{0, \dots, 255\}^{7n/4}$ )
2:    $(\hat{u}, h) \leftarrow \text{DecodeC}(c)$ 
3:    $\hat{s} \leftarrow \text{DecodePoly}(sk)$ 
4:    $v' \leftarrow \text{Decompress}(h)$ 
5:    $\mu = \text{Decode}(v' - \text{NTT}^{-1}(\hat{u} \circ \hat{s}))$ 
6:   return  $\mu$ 

```

---

1778

**B.5 NTRU-HPS and NTRU-HRSS****Algorithm 14** Pseudocode of NTRU KEM Encapsulate( $h$ ) [72]

---

```

1:  $coins \leftarrow_{\$} \{0, 1\}^{256}$ 
2:  $(r, m) \leftarrow \text{Sample\_rm}(coins)$ 
3:  $c \leftarrow \text{Encrypt}(h, (r, m))$ 
4:  $k \leftarrow H_1(r, m)$ 
5: return  $(c, k)$ 

```

---

**Algorithm 15** Pseudocode of NTRU KEM Decapsulate( $(f, f_p, h_q, s), c$ ) [72]

---

```

1:  $(r, m, fail) \leftarrow \text{Decrypt}((f, f_p, h_q), c)$ 
2:  $k_1 \leftarrow H_1(r, M)$ 
3:  $k_2 \leftarrow H_2(s, c)$ 
4: if  $fail = 0$  then
5:   return  $k_1$ 
6: else
7:   return  $K_2$ 
8: end if

```

---



---

**Algorithm 16** Pseudocode of NTRU DPKE Encrypt( $h, (r, m)$ ) [72]

---

```

1:  $m' \leftarrow \text{Lift}(m)$ 
2:  $c \leftarrow (r \cdot h + m') \bmod (q, \Phi_1 \Phi_n)$ 
3: return  $c$ 

```

---



---

**Algorithm 17** Pseudocode of NTRU DPKE Decrypt( $(f, f_p, h_p), c$ ) [72]

---

```

1: if  $c \neq 0 \pmod{(q, \Phi_1)}$  then
2:   return  $(0, 0, 1)$ 
3: end if
4:  $a \leftarrow (c \cdot f) \bmod (q, \Phi_1 \Phi_n)$ 
5:  $m \leftarrow (a \cdot f_p) \bmod (3, \Phi_n)$ 
6:  $m' \leftarrow \text{Lift}(m)$ 
7:  $r \leftarrow ((c - m') \cdot h_q) \bmod (q, \Phi_n)$ 
8: if  $(r, m) \in \mathcal{L}_r \times \mathcal{L}_m$  then
9:   return  $(r, m, 0)$ 
10: else
11:   return  $(0, 0, 1)$ 
12: end if

```

---

1779

## B.6 Streamlined NTRU Prime and NTRULPRime

---

**Algorithm 18** Pseudocode of Encapsulation in Streamlined NTRU Prime and NTRULPRime [71]

---

**Input:**  $\underline{K} \in \text{PublicKeys}$ .

**Output:**  $\text{Ciphertexts}' \times \text{SessionKeys}' = \underline{\text{Ciphertexts}} \times \text{Confirm} \times \text{SessionKeys}'$ .

---

```

1: Decode  $\underline{K}$ , obtaining  $K \in \text{PublicKeys}$ .
2: Generate a uniform random  $r \in \text{Inputs}$ .
3: Encode  $r$  as a string  $\underline{r} \in \text{Inputs}$ .
4: Compute  $c = \text{Encrypt}(r, K) \in \text{Ciphertexts}$ .
5: Encode  $c$  as a string  $\underline{c} \in \text{Ciphertexts}$ .
6: Compute  $C = (\underline{c}, \text{HashConfirm}(\underline{r}, \underline{K})) \in \text{Ciphertexts} \times \text{Confirm}$ 
7: Return  $(C, \text{HashSession}(1, \underline{r}, C))$ .

```

---

---

**Algorithm 19** Pseudocode of Decapsulation in Streamlined NTRU Prime and NTRUL-PRime [71]

---

**Input:**  $C = (\underline{c}, \gamma) \in \text{Ciphertexts} \times \text{Confirm}$  and  $(\underline{k}, \underline{K}, \rho) \in \text{SecretKeys} \times \text{PublicKeys} \times \text{Inputs}$

**Output:** SessionKeys

---

- 1: Decode  $\underline{c}$ , obtaining  $c \in \text{Ciphertexts}$ .
  - 2: Decode  $\underline{k}$ , obtaining  $k \in \text{SecretKeys}$ .
  - 3: Compute  $r' = \text{Decrypt}(c, k) \in \text{Inputs}$ .
  - 4: Encode  $r'$  as a string  $\underline{r}' \in \text{Inputs}$ .
  - 5: Compute  $c' = \text{Encrypt}(r', K) \in \text{Ciphertexts}$ .
  - 6: Encode  $c'$  as a string  $\underline{c}' \in \text{Ciphertexts}$ .
  - 7: Compute  $C' = (\underline{c}', \text{HashConfirm}(\underline{r}', \underline{K})) \in \text{Ciphertexts} \times \text{Confirm}$
  - 8: If  $C' = C$  then return  $\text{HashSession}(1, \underline{r}, C)$ . Otherwise return  $\text{HashSession}(0, \rho, C)$ .  
(The choice between these two outputs is secret information.)
- 

**Algorithm 20** Pseudocode of Encryption in Streamlined NTRU Prime [71]

---

**Input:**  $r \in \text{Inputs}$  and  $K = h \in \text{PublicKeys}$ .

**Output:**  $c \in \text{Ciphertexts}$ .

---

- 1: Compute  $hr \in \mathcal{R}/q$ .
  - 2: **Return**  $c = \text{Round}(hr)$ .
- 

**Algorithm 21** Pseudocode of Decryption in Streamlined NTRU Prime [71]

---

**Input:**  $c \in \text{Ciphertexts}$  and  $k = (f, v) \in \text{Short} \times \mathcal{R}/3$ .

**Output:**  $r' \in \text{Inputs}$ .

---

- 1: Compute  $3fc \in \mathcal{L}/q$ .
  - 2: View each coefficient of  $3fc \in \mathcal{R}/q$  as an integer between  $-(q-1)/2$  and  $(q-1)/2$ , and then reduce modulo 3, obtaining a polynomial  $e \in \mathcal{R}/3$ .
  - 3: Multiply by  $v \in \mathcal{R}/3$ .
  - 4: Lift  $ev \in \mathcal{R}/3$  to a small polynomial  $r' \in \mathcal{R}$ .
  - 5: **Return**  $r'$  if  $r'$  has weight  $w$ . Otherwise output  $(1, 1, \dots, 1, 0, 0, \dots, 0)$ .
- 

**Algorithm 22** Pseudocode of Encryption in NTRU LPRime Expand [71]

---

**Input:**  $r \in \text{Inputs}$  and  $K = (S, A) \in \text{Seeds} \times \text{Rounded}$ .

**Output:**  $c \in \text{Ciphertexts}$ .

---

- 1: Compute  $G = \text{Generator}(S)$ .
  - 2: Generate a uniform random  $b \in \text{HashShort}(r)$ .
  - 3: Compute  $bG$  in  $\mathcal{R}/q$ .
  - 4: Compute  $bA$  in  $\mathcal{R}/q$ .
  - 5: Compute  $T = (T_0, T_1, \dots, T_{I-1}) \in (\mathbb{Z}/\tau)^I$  as follows:  $T_j = \text{Top}((bA)_j + r_j(q-1)/2)$ .
  - 6: **Return**  $c = (\text{Round}(bG), T) \in \text{Ciphertexts}$ .
- 

**Algorithm 23** Pseudocode of Decryption in NTRU LPRime Expand [71]

---

**Input:**  $c = (B, T) \in \text{Rounded} \times (\mathbb{Z}/\tau)^I$  and  $k = a \in \text{SecretKeys}$ .

**Output:**  $r' \in \text{Inputs}$ .

---

- 1: Compute  $aB$  in  $\mathcal{R}/q$ .
  - 2: Compute  $(r'_0, r'_1, \dots, r'_{I-1}) \in \{0, 1\}$  as follows. View  $\text{Right}(T_j) - (aB)_j + 4w + 1 \in (\mathbb{Z}/q)$  as an integer between  $-(q-1)/2$  and  $(q-1)/2$ . Then  $r'_j$  is the sign bit of this integer: 1 if the integer is negative, otherwise 0.
  - 3: **Return**  $r' = (r'_0, r'_1, \dots, r'_{I-1}) \in \text{Inputs}$
-

**B.7 Round5****Algorithm 24** Pseudocode of `r5_cca_kem_encapsulate(pk)` [73]**Parameters:** Integers  $p, t, q, n, d, \bar{m}, \bar{n}, \mu, b, k, f, \tau; \xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$ **Input:**  $pk \in \{0, 1\}^k \times R_{n,p}^{d/n \times \bar{n}}$ **Output:**  $ct = (\tilde{U}, v, g) \in R_{n,p}^{\bar{m} \times d/n} \times Z_t^\mu \times \{0, 1\}^k, k \in \{0, 1\}^k$ 

- 
- 1:  $m \xleftarrow{\$} \{0, 1\}^k$
  - 2:  $(L, g, \rho) = G(m || pk)$
  - 3:  $(\tilde{U}, v) = r5\_cpa\_pke\_encrypt(pk, m, \rho)$
  - 4:  $ct = (\tilde{U}, v, g)$
  - 5:  $k = H(L || ct)$
  - 6: **return**  $(ct, k)$
- 

**Algorithm 25** Pseudocode of `r5_cca_kem_decapsulate(ct, sk)` [73]**Parameters:** Integers  $p, t, q, n, d, \bar{m}, \bar{n}, \mu, b, k, f, \tau; \xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$ **Input:**  $ct = (\tilde{U}, v, g) \in R_{n,p}^{\bar{m} \times d/n} \times Z_t^\mu \times \{0, 1\}^k, sk = (sk_{CPA-PKE, y, pk}) \in \{0, 1\}^k \times \{0, 1\}^k \times (\{0, 1\}^k \times R_{n,p}^{d/n \times \bar{n}})$ **Output:**  $k \in \{0, 1\}^k$ 

- 
- 1:  $m' = r5\_cpa\_pke\_decrypt(sk_{CPA-PKE, (\tilde{U}, v)})$
  - 2:  $(L', g', \rho') = G(m' || pk)$
  - 3:  $(\tilde{U}', v') = r5\_cpa\_pke\_encrypt(pk, m', \rho')$
  - 4:  $ct' = (\tilde{U}', v', g')$
  - 5: **if**  $(ct = ct')$  **then**
  - 6:     **return**  $k = H(L' || ct)$
  - 7: **else**
  - 8:     **return**  $k = H(y || ct)$
  - 9: **end if**
-

**Algorithm 26** Pseudocode of `r5_cpa_pke_encrypt(pk)` [73]**Parameters:** Integers  $p, t, q, n, d, \bar{m}, \bar{n}, \mu, b, k, f, \tau; \xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$ **Input:**  $pk = (\sigma, B) \in \{0, 1\}^k \times R_{n,p}^{d/n \times \bar{n}}, m, \rho \in \{0, 1\}^k$ **Output:**  $ct = (\tilde{U}, v) \in R_{n,p}^{\bar{m} \times d/n} \times Z_t^\mu$ 

- 1:  $A = f_{d,n}^{(\tau)}(e)$
- 2:  $R = f_R(\rho)$
- 3:  $U = R_{q \rightarrow p, h_2}(\langle A^T R \rangle_{\Phi_{n+1}})$
- 4:  $\tilde{U} = U^T$
- 5:  $v = \langle R_{p \rightarrow t, h_2}(\text{Sample}_\mu(\langle B^T R \rangle_\xi)) \rangle_t$
- 6:  $ct = (\tilde{U}, v)$
- 7: **return**  $ct$

**Algorithm 27** Pseudocode of `r5_cpa_pke_decrypt(sk, ct)` [73]**Parameters:** Integers  $p, t, q, n, d, \bar{m}, \bar{n}, \mu, b, k, f, \tau; \xi \in \{\Phi_{n+1}(x), x^{n+1} - 1\}$ **Input:**  $sk \in \{0, 1\}^k, ct = (\tilde{U}, v) \in R_{n,p}^{\bar{m} \times d/n} \times Z_t^\mu$ **Output:**  $\hat{m} \in \{0, 1\}^k$ 

- 1:  $v_p = \frac{p}{t}b$
- 2:  $S = f_s(sk)$
- 3:  $U = \tilde{U}^T$
- 4:  $y = R_{p \rightarrow b, h_3}(v_p - \text{Sample}_\mu((S^T(U + h_4 J))_\xi))$
- 5: **return**  $\hat{m}$

1781

**B.8 Saber****Algorithm 28** Pseudocode of `Saber.KEM.Encaps(pk = (seedA, b))` [74]

- 1:  $m \leftarrow u(\{0, 1\}^{256})$
- 2:  $(\hat{K}, r) = g(F(pk), m)$
- 3:  $c = \text{Saber.PKE.Enc}(pk, m; r)$
- 4:  $K = H(\hat{K}, c)$
- 5: **return**  $(c, K)$

---

**Algorithm 29** Pseudocode of Saber.KEM.Decaps ( $sk = (s, z, pkh), pk = (seed_A, b), c$ ) [74]

---

```

1:  $m' = \text{Saber.PKE.Dec}(s, c)$ 
2:  $(\hat{K}, r') = g(pkh, m')$ 
3:  $c' = \text{Saber.PKE.Enc}(pk, m'; r')$ 
4: if  $c = ct'$  then
5:   return  $K = H(\hat{K}', c)$ 
6: else
7:   return  $K = H(z, c)$ 
8: end if

```

---

**Algorithm 30** Pseudocode of Saber.PKE.Enc ( $pk = (seed_A, b), m \in R_2; r$ ) [74]

---

```

1:  $A = \text{gen}(seed_A) \in R_q^{l \times l}$ 
2:  $(\hat{K}, r') = g(pkh, m')$ 
3: if  $r$  is not specified then
4:    $r = u(\{0, 1\}^{256})$ 
5: end if
6:  $s' = \beta_\mu(R_q^{l \times l}; r)$ 
7:  $b' = ((As' + h) \bmod q) \gg (\epsilon_q - \epsilon_p) \in R_p^{l \times l}$ 
8:  $v' = b^T(s' \bmod p) \in R_p$ 
9:  $c_m = (v' + h_1 - 2^{\epsilon_p - 1} m \bmod p) \gg (\epsilon_q - \epsilon_T) \in R_T$ 
10: return  $c := (c_m, b')$ 

```

---

**Algorithm 31** Pseudocode of Saber.PKE.Dec ( $sk = s, c = (c_m, b')$ ) [74]

---

```

1:  $v = b'^T(s \bmod p) \in R_p$ 
2:  $m' = ((v - 2^{\epsilon_p} - \epsilon_T c_m + h_2) \bmod p) \gg (\epsilon_p - 1) \in R_2$ 
3: return  $m'$ 

```

---

## 1782 C Speed-ups Compared to the Best Portable Implementations in C

1783  
1784 In Table 39, for each investigated KEM and each major operation (Encapsulation and  
1785 Decapsulation), we list the total execution time in software (for the optimized portable  
1786 software implementations in C running on ARM Cortex-A53 of Zynq UltraScale+ MPSoC),  
1787 the total execution time in software and hardware (after offloading the most time-consuming  
1788 operations to hardware), and the obtained speed-up. The ARM processor runs at 1.2 GHz,  
1789 DMA for the communication between the processor and the hardware accelerator at 200  
1790 MHz, and the hardware accelerators at the maximum frequencies, specific for the RTL  
1791 implementations of each algorithm, listed in Table 23. All execution times were obtained  
1792 through experimental measurements using the setup shown in Fig. 1. The speed-up for  
1793 the software part offloaded to hardware itself is given in the column Accel. Speed-up. This  
1794 speed-up is a ratio of the execution time of the accelerated portion in software (column  
1795 Accel. SW [ms]) and the execution time of the accelerated portion in hardware, including  
1796 all overheads (column Accel. HW [ms]). The last column indicates how big percentage  
1797 of the software-only execution time was taken by an accelerated portion of the program.  
1798 Links to the underlying software implementations are summarized in Table 22.

**Table 39:** Speed-ups Compared to the Best Portable Implementations in C

Algorithm	Parameter Set	Total SW [ms]	Total SW/HW [ms]	Total Speed-up	Accel. SW [ms]	Accel. HW [ms]	Accel. Speed-up	SW part Sped up by HW [%]
<b>Encapsulation</b>								
FrodoKEM	Frodo-640	16.192	1.223	13.2	15.322	0.353	43.5	94.62
FrodoKEM	Frodo-976	34.609	1.642	21.1	33.727	0.761	44.3	97.45
FrodoKEM	Frodo-1344	62.076	2.186	28.4	61.218	1.328	46.1	98.62
Kyber	Kyber_512	0.327	0.015	21.5	0.326	0.014	23.9	99.52
Kyber	Kyber_768	0.533	0.018	29.8	0.531	0.016	32.5	99.71
Kyber	Kyber_1024	0.784	0.022	35.5	0.783	0.021	38.2	99.80
LAC-v3a	LAC-128-v3a	0.333	0.016	22.0	0.332	0.015	22.2	99.70
LAC-v3a	LAC-192-v3a	0.564	0.021	28.0	0.562	0.020	28.3	99.73
LAC-v3a	LAC-256-v3a	0.906	0.024	41.0	0.905	0.022	40.6	99.83
LAC-v3b	LAC-128-v3b	0.315	0.014	23.0	0.314	0.013	23.4	99.68
LAC-v3b	LAC-192-v3b	0.530	0.018	31.0	0.529	0.017	31.4	99.72
LAC-v3b	LAC-256-v3b	0.870	0.021	44.0	0.868	0.020	44.4	99.83
NewHope	NewHope_512	0.348	0.015	23.1	0.346	0.013	25.7	99.55
NewHope	NewHope_1024	0.723	0.021	34.0	0.722	0.020	36.7	99.78
Round5	R5ND-CCA1KEM0d	0.155	0.018	8.6	0.154	0.017	9.1	99.36
Round5	R5ND-CCA3KEM0d	0.245	0.024	10.4	0.244	0.023	10.8	99.59
Round5	R5ND-CCA5KEM0d	0.337	0.032	10.6	0.335	0.030	11.1	99.55
Round5	R5ND_CCA1KEM5d	0.126	0.014	8.7	0.125	0.013	9.3	99.21
Round5	R5ND_CCA3KEM5d	0.241	0.020	11.9	0.240	0.019	12.4	99.58
Round5	R5ND_CCA5KEM5d	0.405	0.028	14.5	0.403	0.026	15.3	99.62
Saber	LightSaber-KEM	0.373	0.049	7.6	0.368	0.043	8.5	98.49
Saber	Saber-KEM	0.722	0.057	12.7	0.714	0.049	14.5	98.93
Saber	FireSaber-KEM	1.181	0.065	18.1	1.171	0.055	21.3	99.14
NTRU LPRime	kem/ntrulpr653	1.390	0.052	26.9	1.367	0.029	47.4	98.36
NTRU LPRime	kem/ntrulpr761	1.704	0.060	28.6	1.679	0.034	49.6	98.49
NTRU LPRime	kem/ntrulpr857	2.061	0.067	30.6	2.032	0.039	52.3	98.62
NTRU-HPS	ntruhs2048677	2.961	0.041	71.9	2.946	0.026	115.2	99.47
NTRU-HPS	ntruhs4096821	4.285	0.048	88.5	4.268	0.032	135.4	99.61
NTRU-HRSS	ntruhrss701	2.964	0.068	43.4	2.921	0.025	115.4	98.55
Str NTRU Prime	kem/sntrup653	0.776	0.049	16.0	0.752	0.024	30.9	96.89
Str NTRU Prime	kem/sntrup761	0.948	0.056	17.1	0.921	0.029	32.3	97.15
Str NTRU Prime	kem/sntrup857	1.143	0.063	18.0	1.113	0.033	33.7	97.34
<b>Decapsulation</b>								
FrodoKEM	Frodo-640	16.192	1.321	12.3	15.214	0.343	44.4	93.96
FrodoKEM	Frodo-976	34.649	1.866	18.6	33.532	0.750	44.7	96.78
FrodoKEM	Frodo-1344	62.377	3.120	20.0	60.573	1.317	46.0	97.11
Kyber	Kyber_512	0.428	0.017	25.1	0.428	0.017	25.1	100.00
Kyber	Kyber_768	0.666	0.020	33.2	0.666	0.020	33.2	100.00
Kyber	Kyber_1024	0.950	0.025	38.5	0.950	0.025	38.5	100.00
LAC-v3a	LAC-128-v3a	0.463	0.017	27.1	0.463	0.017	27.1	100.00
LAC-v3a	LAC-192-v3a	0.782	0.024	32.9	0.782	0.024	32.9	100.00
LAC-v3a	LAC-256-v3a	1.378	0.027	51.3	1.378	0.027	51.3	100.00
LAC-v3b	LAC-128-v3b	0.440	0.015	28.4	0.440	0.015	28.4	100.00
LAC-v3b	LAC-192-v3b	0.741	0.021	36.0	0.741	0.021	36.0	100.00
LAC-v3b	LAC-256-v3b	1.332	0.024	55.2	1.332	0.024	55.2	100.00
NewHope	NewHope_512	0.426	0.016	26.4	0.426	0.016	26.4	100.00
NewHope	NewHope_1024	0.891	0.025	35.9	0.891	0.025	35.9	100.00
Round5	R5ND_CCA1KEM0d	0.193	0.020	9.8	0.19273	0.020	9.8	100.00
Round5	R5ND_CCA3KEM0d	0.309	0.028	11.2	0.30946	0.028	11.2	100.00
Round5	R5ND_CCA5KEM0d	0.416	0.037	11.3	0.4158	0.037	11.3	100.00
Round5	R5ND_CCA1KEM5d	0.173	0.016	10.8	0.17267	0.016	10.8	100.00
Round5	R5ND_CCA3KEM5d	0.319	0.023	13.7	0.31877	0.023	13.7	100.00
Round5	R5ND_CCA5KEM5d	0.537	0.033	16.5	0.53721	0.033	16.5	100.00
Saber	LightSaber-KEM	0.471	0.053	9.0	0.459	0.041	11.1	97.60
Saber	Saber-KEM	0.867	0.065	13.4	0.851	0.048	17.7	98.10
Saber	FireSaber-KEM	1.376	0.077	17.9	1.354	0.055	24.6	98.41
NTRU LPRime	kem/ntrulpr653	1.856	0.071	26.2	1.817	0.032	56.7	97.91
NTRU LPRime	kem/ntrulpr761	2.289	0.084	27.2	2.244	0.039	58.1	98.02
NTRU LPRime	kem/ntrulpr857	2.787	0.098	28.6	2.736	0.047	57.8	98.20
NTRU-HPS	ntruhs2048677	8.175	0.095	85.8	8.114	0.034	238.0	99.25
NTRU-HPS	ntruhs4096821	11.982	0.107	111.9	11.912	0.037	318.6	99.42
NTRU-HRSS	ntruhrss701	8.790	0.136	64.8	8.700	0.046	187.7	98.98
Str NTRU Prime	kem/sntrup653	1.643	0.067	24.6	1.599	0.023	68.2	97.36
Str NTRU Prime	kem/sntrup761	2.090	0.079	26.5	2.039	0.029	71.5	97.59
Str NTRU Prime	kem/sntrup857	2.585	0.087	29.8	2.529	0.031	82.0	97.84