Evaluating KpqC Algorithm Submissions: Balanced and Clean Benchmarking Approach

Hyeokdong Kwon[1][0000−0002−9173−512X],
Minjoo Sim[1][0000−0001−5242−214X],
Gyeongju Song[1][0000−0002−4337−1843],
Minwoo Lee[2][0000−0002−2356−3055], and
Hwajeong Seo[2][0000−0003−0069−9061]

[1]Department of Information Computer Engineering,
Hansung University, Seoul (02876), South Korea,
[2]Department of Convergence Security,
Hansung University, Seoul (02876), South Korea,
`{korlethean, minjoos9797, thdrudwn98, minunejip, hwajeong84}@gmail.com`

Abstract. In 2022, a Korean domestic Post Quantum Cryptography contest called KpqC held, and the standard for Post Quantum Cryptography is set to be selected in 2024. In Round 1 of this competition, 16 algorithms have advanced and are competing. Algorithms submitted to KpqC introduce their performance, but direct performance comparison is difficult because all algorithms were measured in different environments. In this paper, we present the benchmark results of all KpqC algorithms in a single environment. To benchmark the algorithms, we removed the external library dependency of each algorithm. By removing dependencies, performance deviations due to external libraries can be eliminated, and source codes that can conveniently operate the KpqC algorithm can be provided to users who have difficulty setting up the environment.

Keywords: Benchmark · Cryptography Implementation · KpqC · Post Quantum Cryptography · Standardization.

1 Introduction

Quantum computers, first proposed by physicist Richard Feynman in 1981 [1], gradually began to materialize with the introduction of quantum algorithms by Professor David Deutsch in 1985 [2]. Quantum computers can execute quantum algorithms, which pose a significant threat to modern cryptosystems. One example is Grover's algorithm, an algorithm designed to locate specific data among n unsorted data entries [3]. A classical computer would require a brute force search of at most $\mathrm{O}(2^n)$ attempts. However, Grover's algorithm reduces the search time to a maximum of $\mathrm{O}(2^{n/2})$. This effectively halves the security of symmetric key algorithms and hash functions. Another powerful quantum algorithm is the Shor algorithm, which efficiently performs prime factorization [4]. The Shor algorithm can break public key algorithms such as RSA and ECC in polynomial time. While symmetric key algorithms can temporarily mitigate the threat posed by quantum computers by doubling the key length, public key algorithms lack such defenses. To address this issue, NIST initiated a Post Quantum Cryptography competition to foster the development, standardization, and distribution of Post Quantum Cryptography [5]. Similarly, a Post Quantum Cryptography contest was held in Korea, marking the beginning of the process to select a Korean standard. In this paper, we eliminate the dependencies of the algorithms submitted to the KpqC competition and present benchmark results in a standardized environment.

1.1 Contributions

- **Benchmark results in common development environments.** All algorithms in KpqC Round 1 come with detailed white papers that include performance measurements. The provided benchmarks, prepared by the development teams, are highly reliable. However, a challenge arises from the fact that the benchmark environments in the white papers differ. Therefore, conducting the benchmark in an environment with sufficient resources can provide advantageous results. To address this, we conducted a collective benchmark of all algorithms in a standardized environment, enabling a fair performance comparison among the algorithms.
- **Enhancing accessibility by eliminating working dependencies.** Many of the KpqC candidate algorithms rely on external libraries for their implementation. These dependencies offer significant benefits by providing preexisting modules for algorithm implementation, eliminating the need for separate implementation. However, from the perspective of downloading and using the source code, the absence of these dependencies can create complexity and hinder code operation. This issue is particularly challenging for novice users who may struggle with setting up the development environment. To address this, our focus was on removing the dependencies associated with the algorithms. This approach offers two key advantages.

  Firstly, it enables immediate use of the source code without the need to set up the environment. By distributing the source code in a runnable state, even users unfamiliar with environment setup can easily operate the code. This greatly enhances the accessibility of the source code and attracts a broader user base.

  Secondly, it ensures a fair benchmarking process. While most libraries used are likely to be the same, discrepancies can still arise due to differences in library versions during environment setup. By eliminating external library dependencies and replacing the necessary modules with identical source code, we can provide more accurate and equitable benchmark results.

The rest of the paper is structured as follows. Section 2 provides an overview of the Post Quantum Cryptography contest and the specific details of KpqC. Additionally, we introduce the PQClean project, which served as the inspiration for the research discussed in this paper. In Section 3, we detail the methodology employed to benchmark the KpqC algorithms and present the obtained benchmark results. Finally, Section 4 concludes the paper by summarizing the key findings and discussing potential directions for future research.

2 Related Works

2.1 NIST Standardization of Post Quantum Cryptography

The significance of Post Quantum Cryptography (PQC) has emerged as a means to ensure secure communication in the age of quantum computers. The United States National Institute of Standards and Technology (NIST) initiated the PQC Standardization Contest in 2016. In 2022, standard algorithms were selected, and Round 4 was conducted to determine additional standards [6]. Following the standard selection process, CRYSTALS-Kyber [7] was chosen as the Key Encapsulation Mechanism (KEM). In the Digital Signature category, the selected algorithms include CRYSTALS-Dilithium [8], Falcon [9], and SPHINCS+[10]. Round 4 is currently underway to select additional standards for the KEM category, with BIKE [11], Classic McEliece [12], and HQC [13] competing. Although SIKE [14] advanced to Round 4, it subsequently withdrew from the contest due to the discovery of a security vulnerability [15].

2.2 KpqC: Korea's Post Quantum Cryptography Standardization

KpqC, the domestic contest for standardizing Post Quantum Cryptography in Korea, took place at the end of 2021 [16]. The timeline of the KpqC competition is presented in Table 1. The results of Round 1 were announced at the end of 2022, with 16 algorithms passing the evaluation. Subsequently, the Round 2 results will be announced in December 2023, and the final standard will be selected in September 2024. KpqC has defined four evaluation criteria for assessing the algorithms.

The first criterion is safety. Algorithms must demonstrate their security and provide proof of their safety. It is crucial for these algorithms to ensure security not only on quantum computers but also on classical computers.

The second criterion is efficiency. The computational resources required to execute the algorithm should be reasonable, and the computation time should not be excessively long. Some algorithms may have a probability of decryption or verification failure due to their structure, but this failure probability should not hinder their practical use.

The third criterion is usability. The implemented algorithm should be capable of operating in various environments, ensuring its usability across different systems.

The final criterion is originality. The proposed algorithm should possess a creative and innovative structure, showcasing novel approaches in the field of Post Quantum Cryptography.

In Kpqc Round 1, a total of 7 algorithms were chosen in the Key Encapsulation Mechanism (KEM) category, and 9 algorithms were selected in the Digital

Table 1: Timeline of KpqC competetion.

Phase	Date
Announcement of holding KpqC	2021. 11.
Deadline for submitting candidate algorithms	2022. 10.
Announcement of Round 1 Results	2022. 12.
Scheduled date for announcement of Round 2 results	2023. 9.
Scheduled date of announcement of standard selection	2024. 09.

Signature category. The specific algorithms that passed the evaluation can be found in Table 2. Notably, among the selected algorithms, the Lattice-based algorithms demonstrate strong performance. This observation aligns with the NIST PQC standard, where three out of the four selected standards are also Lattice-based algorithms.

Table 2: KpqC Round 1 candidate algorithms.

Scheme	PKE/KEM	Digital Signature
Code-based	IPCC [17] Layered-ROLLO [19] PALOMA [20] REDOG [21]	Enhanced pqsigRM [18]
Lattice-based	NTRU+ [23] SMAUG [25] TiGER [27]	GCKSign [22] HAETAE [24] NCC-Sign [26] Peregrine [28] SOLMAE [29]
Multivariate Quadratic-based	-	MQ-Sign [30]
Hash-based	-	FIBS [31]
Zero knowledge-based	-	AIMer [32]

2.3 PQClean

A significant number of NIST's Post Quantum Cryptography algorithms rely on external libraries. Utilizing pre-existing implementations rather than building modules from scratch is more efficient in cryptographic algorithm implementation, resulting in the creation of dependencies on external libraries. While dependencies are convenient during development, they can pose inconvenience when using the implemented source code as the development environment must be set up accordingly. To address this issue, PQClean, a library introduced in [33], focuses on removing these dependencies to enable easy operation of Post Quantum Cryptography (PQC).

PQClean not only aims to make source code operation more convenient by eliminating library dependencies but also places emphasis on improving the overall quality of the source code. To achieve this, PQClean conducted approximately 30 implementation checklists. These checklists included verifying adherence to the C standard, ensuring consistency in compilation rules, minimizing Makefiles, and confirming the consistency of integer data. As a result, PQClean not only facilitates the convenient operation of source code by removing library dependencies but also provides clean source code for higher quality implementation.

Another advantage of PQClean is its ease of portability to other platforms or frameworks, as it does not form dependencies. For instance, pqm4 is a library that collectively executes NIST PQC on ARM Cortex-M4 and provides benchmark results [34]. This showcases the versatility and compatibility of PQClean in various computing environments.

3 KPQClean: Clean Benchmark on KpqC

Building upon the inspiration of PQClean, we undertook the KPQClean project for the KpqC competition. The initial phase involved working with a total of 16 candidate algorithms from KpqC Round 1. To present the results, we focused on removing external dependencies for each algorithm and conducting benchmarking.

The project proceeded in a systematic manner, starting with the removal of libraries and subsequently addressing the Makefile rules and benchmarking. During the library removal process, we carefully examined the dependencies present in each code. Most KpqC algorithms rely on OpenSSL [35] and utilize OpenSSL's AES for random number generation. To eliminate these dependencies, the code sections that made external library calls were removed. Consequently, the externally implemented algorithms can no longer be used in their original form. To ensure the operation of these algorithms, we directly integrated the source code that implements the algorithms. For instance, the AES algorithm requires the utilization of CTR-DRBG. We ported the AES code used by PQClean, making necessary modifications to the internal structure to ensure seamless functionality. This approach enables the development of code that operates independently by eliminating external library dependencies.

The next step involved writing a unified Makefile. Most KpqC candidate algorithms offer convenient compilation using gcc by providing compilation rules in the Makefile. However, variations in compilation rules across different algorithms can lead to discrepancies in performance measurements. To address this, we made efforts to create a standardized Makefile with consistent rules, thereby ensuring fair and comparable benchmarking results.

Lastly, the benchmarking process was conducted. A dedicated source code for benchmarking was prepared, compiled, and executed. The benchmarking environment resembled the specifications outlined in Table 3. The two devices use a Ryzen processor and an Intel processor, respectively, and the rest of the device specifications are almost identical. To obtain the measurements, each algorithm underwent 10,000 iterations, and the median number of clock cycles required for operation was calculated for each round. We applied -O2 as an optimization level option. However, most of the KpqC algorithms performed performance measurements on -O3. Therefore, -O3 performance was additionally measured to reflect the developer's intention.

Table 4 presents the benchmark results for the Key Encapsulation Mechanism (KEM) algorithms. Among the KEM candidates, SMAUG performed the best in the Keygen operation, NTRU+ excelled in Encapsulation, and PALOMA showcased the best performance in Decapsulation. However, it is worth noting that IPCC-1's Encapsulation measurement exhibited excessively slow performance, leading to the exclusion of its measurement as an error value. As a result, it was temporarily excluded from the benchmark.

One KEM algorithm REDOG, was temporarily excluded from the benchmarking process. REDOG presented a unique case as it was implemented in pure Python while all algorithm analyses were based on the C language. Consequently, REDOG was excluded from the benchmarking process as it deviated from the performance standards used for other algorithms.

Table 3: Benchmark environment.

	Environment 1	Environment 2
CPU	Ryzen 7 4800H	Intel i5-8259U
GPU	RTX 3060	Coffee Lake GT3e
RAM	16GB	16GB
OS	Ubuntu 22.04	Ubuntu 22.04
Compiler	gcc 11.3.0	gcc 11.3.0
Optimization level	-O2, -O3	-O2, -O3
Editor	Visual Studio Code	Visual Studio Code

Table 5 presents the benchmark results for the Digital Signature candidate algorithms. The performance measurements were conducted in the same benchmarking environment as the Key Encapsulation Mechanism (KEM) algorithms.

In the Digital Signature category, AIMer demonstrated excellent performance in the Keygen operation, while Peregrine showcased exceptional performance in Sign and Verification. However, accurate measurements for FIBS could not be obtained due to incomplete calculations, rendering its measurement inconclusive.

In common, the operation on the Intel processor tends to be somewhat faster than the operation on the Ryzen processor. This is because the Intel processor used in the experiment had better performance than the Ryzen processor. Also, for many algorithms, the performance difference between the -O2 and -O3 op-

Table 4: Benchmark result of KpqC KEM Round 1 Candidates. (Unit: clock cycles(algorithm speed), Strikethrough: Lack of consistency in benchmarks, ^{A}: AVX applied.)

	Environment 1 -O2			Environment 2 -O2		
Algorithm	Keygen	Encapsulation	Decapsulation	Keygen	Encapsulation	Decapsulation
IPCC-1	14,362,627	~~164,892,550~~	2,484,981	13,792,887	~~159,126,951~~	1,196,157
IPCC-3	14,170,647	898,710	2,619,570	13,754,219	870,059	1,235,991
IPCC-4	14,209,594	1,075,059	2,904,524	13,754,687	1,050,451	1,318,173
NTRU+-576^A	208,742	111,998	128,093	186,944	105,686	120,194
NTRU+-768^A	279,386	148,480	181,250	246,616	139,310	166,938
NTRU+-864^A	304,819	179,858	224,953	270,494	160,789	200,702
NTRU+-1152^A	444,744	223,619	278,690	698,490	202,678	257,114
PALOMA-128	125,800,419	510,922	35,496	118,204,341	499,914	39,724
PALOMA-192	125,360,779	514,228	34,220	118,310,371	499,302	38,846
PALOMA-256	125,294,065	510,284	34,713	118,366,206	503,814	43,174
SMAUG-128	171,477	154,483	178,205	158,149	164,598	196,470
SMAUG-192	250,096	229,999	277,298	244,736	225,490	272,132
SMAUG-256	479,138	385,178	438,364	435,790	411,917	465,572
TiGER-128	273,470	466,755	628,778	163,856	209,168	311,924
TiGER-192	288,550	518,491	674,192	171,578	214,126	312,702
TiGER-256	536,152	~~1,088,747~~	~~1,477,318~~	444,558	433,462	673,105
	Environment 1 -O3			Environment 2 -O3		
IPCC-1	13,940,097	~~160,111,204~~	~~16,360,164~~	12,643,392	~~145,233,220~~	1,159,273
IPCC-3	13,996,024	926,492	2,512,836	12,795,377	874,663	1,206,585
IPCC-4	13,989,832	1,106,031	2,714,531	13,078,917	1,037,485	1,310,503
NTRU+-576^A	202,652	110,026	121,742	177,748	102,296	111,820
NTRU+-768^A	270,512	146,566	174,435	239,546	137,135	161,970
NTRU+-864^A	297,192	168,113	204,537	260,672	153,481	186,386
NTRU+-1152^A	435,305	222,459	266,626	568,556	201,226	246,050
PALOMA-128	122,325,408	498,365	34,307	108,402,198	459,846	40,838
PALOMA-192	122,290,738	503,266	34,278	108,206,652	460,374	40,688
PALOMA-256	122,321,957	497,959	34,249	108,216,713	459,880	40,886
SMAUG-128	72,790	57,246	50,460	63,020	49,324	39,196
SMAUG-192	105,966	82,940	80,475	92,658	69,739	67,691
SMAUG-256	158,021	139,925	135,749	135,202	122,766	115,096
TiGER-128	65,482	48,749	51,214	62,490	45,398	53,248
TiGER-192	69,426	63,510	57,739	66,512	60,238	58,572
TiGER-256	81,316	87,551	93,090	78,772	82,776	89,902
Layered ROLLO I-128^A	285,940	83,346	788,104	203,181	66,529	558,503
Layered ROLLO I-192^A	320,958	136,503	518,491	227,813	102,758	671,605
Layered ROLLO I-256^A	687,721	201,913	1,014,203	375,056	136,052	1,245,346

tions is not noticeable. This is because each algorithm is well optimized and no further optimization is performed at the compiler level. Some algorithms perform better with the -O3 option. In this case, it can be said that the algorithms have a point where optimization is possible.

Table 5: Benchmark result of KpqC Digital Signature Round 1 Candidates. (Unit: clock cycles(algorithm speed), $_o$: original(NCCSign) $_c$: conserparam(NCCSign), Strikethrough: Lack of consistency in benchmarks, ^{A}: AVX applied.)

	Environment 1 -O2			Environment 2 -O2		
Algorithm	Keygen	Sign	Verification	Keygen	Sign	Verification
AIMer-I	145,058	3,912,361	3,669,834	145,566	3,691,256	3,713,173
AIMer-III	296,496	8,001,274	7,550,063	274,358	7,771,108	7,366,672
AIMer-V	710,442	18,068,276	17,415,022	790,456	18,394,069	17,662,359
GCKSign-II	179,771	601,707	176,987	171,176	640,093	167,116
GCKSign-III	186,673	649,049	183,367	173,252	698,964	168,824
GCKSign-V	252,822	917,415	277,733	248,629	945,815	273,631
HAETAE-II	798,312	4,605,461	147,494	700,875	4,173,002	142,584
HAETAE-III	~~1,533,941~~	~~11,474,155~~	257,926	~~1,352,577~~	10,615,663	250,534
HAETAE-V	846,713	3,902,298	305,428	752,413	3,418,728	311,986
MQSign-72/46	94,788,559	516,954	1,461,281	87,038,447	509,630	1,377,392
MQSign-112/72	488,913,828	1,493,703	5,211,909	448,271,119	1,472,032	4,808,216
MQSign-148/96	1,488,480,956	3,162,943	12,036,827	1,326,638,494	3,128,536	11,091,036
NCCSign-I$_o^A$	2,650,542	10,404,301	5,232,079	2,296,351	15,914,954	4,519,308
NCCSign-III$_o^A$	4,477,513	17,657,839	8,867,243	4,009,717	16,015,734	7,996,462
NCCSign-V$_o^A$	7,240,343	64,377,767	14,358,074	6,561,582	26,019,063	13,005,536
NCCSign-I$_c^A$	1,869,079	23,762,252	3,681,057	1,704,190	27,083,021	3,344,228
NCCSign-III$_c^A$	3,655,334	~~39,587,190~~	7,241,808	3,271,119	~~65,455,745~~	6,533,931
NCCSign-V$_c^A$	6,263,739	179,281,596	12,418,902	5,723,169	39,565,842	6,533,931
Peregrine-512	12,401,256	329,933	37,294	12,073,005	295,128	33,114
Peregrine-1024	39,405,505	709,848	80,243	38,493,479	640,132	71,246
Enhanced pqsigRM-612	6,013,112,315	7,210,560	2,223,401	4,961,556,899	7,505,040	2,125,125
Enhanced pqsigRM-613	58,238,108,879	1,864,512	1,053,034	74,021,054,015	2,113,913	1,126,131
SOLMAE-512^A	23,848,774	378,392	43,935	22,494,902	351,311	64,526
SOLMAE-1024^A	55,350,546	760,380	141,375	52,388,360	706,028	152,984

	Environment 1 -O3			Environment 2 -O3		
AIMer-I	145,986	3,878,272	3,672,923	133,130	3,960,345	3,747,101
AIMer-III	296,032	8,087,462	7,678,098	272,484	8,440,184	7,968,982
AIMer-V	713,922	17,983,857	17,361,691	643,253	17,998,305	17,373,174
GCKSign-II	164,836	537,675	159,674	175,993	597,712	172,893
GCKSign-III	166,199	581,189	161,646	183,987	698,941	179,608
GCKSign-V	231,797	895,549	279,009	238,884	928,251	262,868
HAETAE-II	688,083	3,429,265	131,805	672,901	3,334,242	126,972
HAETAE-III	~~1,329,157~~	~~8,734,670~~	228,578	~~1,291,292~~	~~8,261,232~~	227,780
HAETAE-V	723,318	2,790,612	272,542	719,708	2,627,334	270,600
MQSign-72/46	39,040,917	311,112	512,227	38,474,591	298,952	533,676
MQSign-112/72	115,942,827	669,465	1,143,296	117,049,542	650,928	1,120,124
MQSign-148/96	235,289,035	1,186,622	1,943,667	236,124,011	1,165,706	1,897,664
NCCSign-I$_o^A$	2,619,295	10,301,902	5,171,686	2,317,555	13,776,448	4,568,006
NCCSign-III$_o^A$	4,379,261	~~86,475,941~~	8,685,877	3,981,551	~~83,521,123~~	7,935,382
NCCSign-V$_o^A$	7,178,921	42,637,366	14,245,148	6,333,006	25,183,392	12,555,623
NCCSign-I$_c^A$	1,843,356	~~50,520,712~~	3,636,803	1,666,543	16,352,341	3,248,162
NCCSign-III$_c^A$	3,618,997	21,416,384	7,170,903	3,141,974	34,454,252	6,234,249
NCCSign-V$_c^A$	6,149,059	151,973,282	12,196,791	5,613,303	167,158,023	11,155,020
Peregrine-512	11,953,307	253,402	25,462	11,783,005	260,328	26,262
Peregrine-1024	38,366,232	535,920	53,621	38,493,479	551,168	55,654
Enhanced pqsigRM-612	6,139,551,981	4,610,319	2,278,806	4,702,612,115	4,732,706	2,064,731
Enhanced pqsigRM-613	54,994,439,928	714,647	225,577	71,111,088,778	923,513	417,658
SOLMAE-512^A	23,053,028	349,566	40,513	22,627,042	332,848	64,838
SOLMAE-1024^A	53,966,332	698,581	135,256	53,245,753	668,103	149,168

4 Conclusion

This paper presents a benchmarking effort conducted on the candidate algorithms of KpqC Round 1. To facilitate the benchmarking process, the KPQClean library was developed and is currently available on GitHub[1]. The primary objective of the KPQClean library is to remove dependencies in the KpqC candidate algorithms and provide benchmark results in a standardized environment. However, there are a few limitations that need to be addressed.

Firstly, there are algorithms that have not yet been measured or evaluated. While this issue exists, the goal is to address these gaps and provide benchmark results for all the algorithms. Efforts are ongoing to resolve these outstanding matters.

The second limitation involves the removal of remaining dependencies. Currently, KPQClean focuses on eliminating external library dependencies, but there are other dependencies such as dynamic allocation that still need to be addressed. The aim is to eliminate all dependencies to ensure a fully self-contained library.

Lastly, it is important to rectify any anomalies or unusual values in the measurement results to ensure that they align with normal benchmarking standards. While most algorithms exhibit consistent trends in their measurements, some algorithms may demonstrate anomalies that result in extremely slow performance. These issues will be addressed to provide accurate and reliable benchmark results. This is likely due to limitations of the benchmark method. To measure the performance of the algorithm, many iterations were performed and the median value of the values was used. During this process, the equipment may perform other calculations, and performance may deteriorate due to heat generation. Therefore, we devise a more sophisticated benchmark method.

The KPQClean project is an ongoing endeavor closely aligned with the KpqC Competition. The project aims to present benchmark results in a unified environment while also providing a more convenient PQC library. This endeavor seeks to generate increased interest among researchers and students in the field of KpqC, offering a comfortable and conducive environment for further study and exploration.

References

1. R. P. Feynman, “Simulating physics with computers,” in *Feynman and computation*, pp. 133–153, CRC Press, 2018.
2. D. Deutsch, “Quantum theory, the church–turing principle and the universal quantum computer,” *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.
3. L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.
4. P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

---

[1] `https://github.com/kpqc-cryptocraft/KPQClean`

5. K.-B. Jang and H.-J. Seo, "Quantum computer and standardization trend of nist post-quantum cryptography," in *Proceedings of the Korea Information Processing Society Conference*, pp. 129–132, Korea Information Processing Society, 2019.
6. NIST, "Round 4 submissions - post-quantum cryptography: Csrc," 2022.
7. R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber algorithm specifications and supporting documentation," *NIST PQC Round*, vol. 2, no. 4, pp. 1–43, 2019.
8. V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai, "Crystals-dilithium," *Algorithm Specifications and Supporting Documentation*, 2020.
9. T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon," *Post-Quantum Cryptography Project of NIST*, 2020.
10. D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 2129–2146, 2019.
11. N. Aragon, P. S. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Guneysu, C. A. Melchor, *et al.*, "Bike: bit flipping key encapsulation," 2017.
12. M. R. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen, *et al.*, "Classic mceliece," *National Institute of Standards and Technology, Tech. Rep*, 2020.
13. C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, and G. Zémor, "Hqc: hamming quasi-cyclic," *NIST Post-Quantum Standardization, 3rd Round*, 2021.
14. R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, *et al.*, "Supersingular isogeny key encapsulation," *Submission to the NIST Post-Quantum Standardization project*, vol. 152, pp. 154–155, 2017.
15. W. Castryck and T. Decru, "An efficient key recovery attack on sidh." Cryptology ePrint Archive, Paper 2022/975, 2022. `https://eprint.iacr.org/2022/975`.
16. K. team, "Kpqc competition round 1." https://kpqc.or.kr/competition.html, 2023. Accessed: 2023-04-07.
17. J. Ryu, Y. Kim, S. Yoon, J.-S. Kang, and Y. Yeom, "Ipcc-improved perfect code cryptosystems,"
18. J. Cho, J.-S. No, Y. Lee, Z. Koo, and Y.-S. Kim, "Enhanced pqsigrm: Code-based digital signature scheme with short signature and fast verification for post-quantum cryptography," *Cryptology ePrint Archive*, 2022.
19. C. Kim, Y.-S. Kim, and J.-S. No, "Layered rollo-i: Faster rank-metric code-based kem using ideal lrpc codes," *Cryptology ePrint Archive*, 2022.
20. D.-C. Kim, C.-Y. Jeon, Y. Kim, and M. Kim, "Paloma: Binary separable goppa-based kem1,"
21. J.-L. Kim, J. Hong, T. S. C. Lau, Y. Lim, C. H. Tan, T. F. Prabowo, and B.-S. Won, "Redog and its performance analysis," *Cryptology ePrint Archive*, 2022.
22. J. Woo, K. Lee, and J. H. Park, "Gcksign: Simple and efficient signatures from generalized compact knapsacks," *Cryptology ePrint Archive*, 2022.
23. J. Kim and J. H. Park, "Ntru+: Compact construction of ntru using simple encoding method," *Cryptology ePrint Archive*, 2022.
24. J. H. Cheon, H. Choe, J. Devevey, T. Güneysu, D. Hong, M. Krausz, G. Land, J. Shin, D. Stehlé, and M. Yi, "Haetae: Hyperball bimodal module rejection signature scheme,"

25. J. H. Cheon, H. Choe, D. Hong, J. Hong, H. Seong, J. Shin, and M. Yi, "Smaug: the key exchange algorithm based on module-lwe and module-lwr,"
26. K.-A. Shim, J. Kim, and Y. An, "Ncc-sign: A new lattice-based signature scheme using non-cyclotomic polynomials,"
27. S. Park, C.-G. Jung, A. Park, J. Choi, and H. Kang, "Tiger: Tiny bandwidth key encapsulation mechanism for easy migration based on rlwe (r)," *Cryptology ePrint Archive*, 2022.
28. E.-Y. Seo, Y.-S. Kim, J.-W. Lee, and J.-S. No, "Peregrine: Toward fastest falcon based on gpv framework," *Cryptology ePrint Archive*, 2022.
29. K. Kim, M. Tibouchi, A. Wallet, T. Espitau, A. Takahashi, Y. Yu, and S. Guilley, "Solmae algorithm specifications," 2022.
30. K.-A. Shim, J. Kim, and Y. An, "Mq-sign: A new post-quantum signature scheme based on multivariate quadratic equations: Shorter and faster," 2022.
31. S. Kim, Y. Lee, and K. Yoon, "Fibs: Fast isogeny based digital signature,"
32. S. Kim, J. Ha, M. Son, B. Lee, D. Moon, J. Lee, S. Lee, J. Kwon, J. Cho, H. Yoon, *et al.*, "The aimer signature scheme,"
33. M. J. Kannwischer, P. Schwabe, D. Stebila, and T. Wiggers, "Improving software quality in cryptography standardization projects," in *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 19–30, IEEE, 2022.
34. M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "pqm4: Testing and benchmarking nist pqc on arm cortex-m4," 2019.
35. J. Viega, M. Messier, and P. Chandra, *Network security with openSSL: cryptography for secure communications.* " O'Reilly Media, Inc.", 2002.