Post-Quantum Encryption Benchmarks

Jordan Churi, *Electrical Engineering Student, California Polytechnic State University San Luis Obispo*

CONTENTS

LIST OF FIGURES

***Abstract*—Recent advancements in quantum computing bring the weaknesses in modern RSA encryption to the foreground. Shor's algorithm, though not implementable on today's quantum computers, shows that RSA asymmetric key encryption is not secure for the coming future. This flaw in the security has prompted the National Institute of Standards and Technology (NIST) to start a search for a new post-quantum encryption algorithm that will be resistant to future quantum computers. There are several implementations of preforming this encryption scheme. One promising technique is using lattices in an application called ring Learning with Errors (rLWE). Several algorithms have been submitted to NIST for post-quantum encryption. This paper covers the speed differences of different implementations of rLWE algorithms that have made it past the NIST round two post-quantum submissions on a desktop processor and an embedded system.**

***Index Terms*—Post-Quantum Cryptography, ring Learning with Errors**

I. INTRODUCTION

THIS post-quantum encryption is a multidisciplinary field, so this section will cover the motivations of why we are testing this encryption scheme and the backgrounds required to help understand the implications and results in this paper.

A. Motivation

1) Why Do We Need Asymmetric Encryption?: Asymmetric encryption is the type of encryption that is the encryption scheme used for public key exchange [1]. A public key exchange allows two parties who do not know each other to communicate securely without a common system to distribute keys [2]. Allowing different parties not to need a centralized authority to distribute keys to communicate securely is a fundamental part of the modern internet infrastructure. With asymmetric keys being used to create a secure channel in which to establish and more secure symmetric key protocol [3]. This type of encryption allows RSA is one of the standard asymmetric encryption algorithms. Looking at Fig: 1 shows how asymmetric encryption fits into communication with AES is the symmetric encryption algorithm. However, as mentioned before, there is a vulnerability in RSA encryption from quantum computers, which has led to this search for a new asymmetric post-quantum encryption standard.

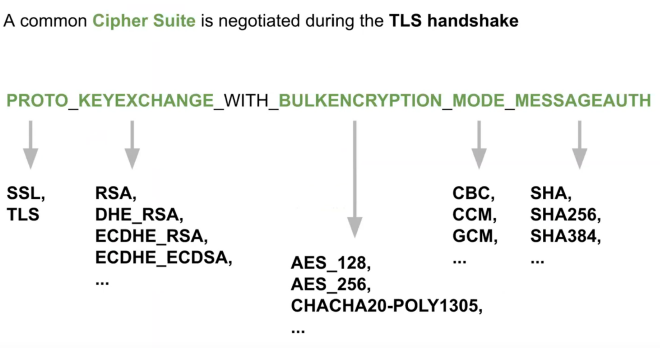

Fig. 1. TSL Handshake

2) Different Answers to the Same Problem: There are several different implementations of rLWE that have passed to the round to of the NIST post-quantum encryption search. However, the implementation and optimization of these algorithms are different for each of the encryption algorithms. This

difference in implementation causes a difference in performance between the encryption algorithms. With the increase of the internet of things (IoT) needing encryption, it is essential to make the algorithms as efficient as possible not just because of the number of devices and the limited resources that they have and inefficiencies scale.

3) Alternate Solutions: There are several other different types of post-quantum encryption problems that have been implemented other than rLWE. The two other major types of post-quantum encryption are code-based and multivariate [4]. Code-based encryption uses error correction codes to provide the security, and multivariate based encryption uses solving quadratic equations over a finite field [4]. Though these two implementations are valid and quantum secure enough to make it to round two of the NIST submissions, we chose to use rLWE. However, we selected the rLWE problem for several reasons. Another team is doing a hardware implementation of rLWE so we can compare hardware speedups in the future. Additionally, rLWE has a homomorphic property that allows for data to be manipulated without decrypting the data making it possible to use for machine learning in privacy critical applications like medicine as being able to manipulate and analyze data is essential but with current methods violates privacy of the patients.

B. Background

1) NISQ Era: When talking about post-quantum encryption, it is beneficial to mention a few words toward the current state of the field of quantum computing, as of June 2020. Currently, quantum computing can be said to be in the Noisey Intermediate Scale Quantum (NISQ) era. The NISQ era of quantum computing is characterized by quantum computers with 50 to 100 qubits that can perform some tasks that can outperform classical computers [5]. An example of this NISQ era quantum computer is Google's quantum supremacy experiment using their Sycamore quantum processor in October of 2019. Google's Sycamore processor was able to perform the Feynman-Schordinger algorithm in 200s that a classical computer would take 10,000 years to complete showing quantum supremacy [6]. However, there are some disputes by IBM on the implementation of classical computers. Though these NSIQ systems are not large enough and qubit coherence times are not longe enough to implement the quantum Fourier transform a significant part of Shor's algorithm that can break RSA encryption. However, it is a sign that now is the time to be transitioning over to a post-quantum encryption scheme before quantum computers can implement Shor's algorithm.

2) Shor's Algorithm: A few words should be said on the algorithm that has prompted this search for a new asymmetric encryption standard. Shor's algorithm was one of the first algorithms that showed the exponential speedups of a quantum computer. Shor's algorithm uses the quantum Fourier transform to factor the large prime numbers that compose the key for RSA encryption. The best classical implementation runs in superpolynomial time $O(exp(c(\ln n^{1/3}(\ln \ln n^{2/3}))))$ [7], whereas Shor's algorithm reduces that time complexity to subpolynomial time $O(((\log n)^2)(\log \log n)(\log \log \log n))$ [8] [7]. This speedup is what allows RSA to be considered ineffective at ensuring secure communication in a post-quantum world.

3) rLWE: As mentioned previously, ring learning with errors (rLWE) is one of the major categories in the NIST post-quantum encryption search. This section will go into some of the workings of this algorithm. The rLWE algorithm works by generating several n bit random arrays represented by s and e shown in Fig: 2. Then polynomial multiply the message A by s and add e [9]. This addition of e is the error in rLWE. An example of this can be seen in Fig: 2. All random arrays are Gaussian distributed random numbers. rLWE is the basis for several of the NIST round two submissions. The several round two submissions that we will be using are:

- New Hope [10]
- Kyber [11]
- NTRU [12]
- FrodoKEM [13]

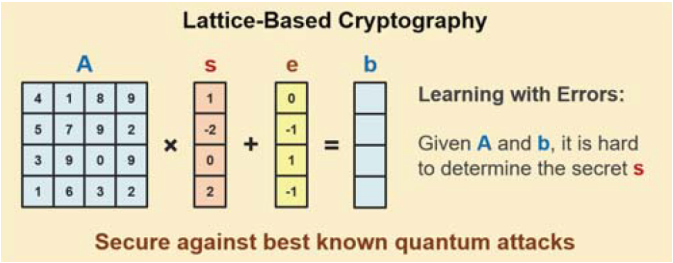

Fig. 2. rLWE Encryption

[14]

II. Design

To begin with, we had to figure out how to compare the various post-quantum encryption implementations. Thankfully, due to the NIST submission requirements, many of the quantum encryption algorithms had built-in speed tests for running in a Linux environment that returns the speed of specific tasks. However, this posed another problem because several of the intermediary steps, such as Numeric Theoretic Transform (NTT), are not provided in the speed tests. This difference in what is measured forced us to choose the common tests of Key Encapsulation, key decapsulation, and key pair generation. Additionally, another factor to consider is which key size that we will be testing. We used the maximum key size of each encryption scheme, which fulfills the NIST Level 5 requirement, which is equal to or exceeding the brute-force security of AES-256 and is roughly equivalent to RSA with a 2048 bit key.

The other factor to think about is the underlying dependencies of the post-quantum encryption algorithms. All the post-quantum encryption algorithms have dependencies in the OpenSSL library. This dependence is mainly in the random number generation for rLWE. This dependency means that many of the speed operations are dependent on the OpenSSL library. For this reason, we want to use the OpenSSL library to run speed tests on the pre-quantum RSA encryption algorithm to get a baseline to compare against the post-quantum algorithms.

However, OpenSSL is too large to run on embedded devices. To solve this problem, we planned to use the WolfSSL library and migrate the dependencies that require OpenSSL to WolfSSL to be able to run on smaller systems—then using the WolfSSL RSA benchmark to compare against the post-quantum encryption schemes. However, due to time constraints, this was not accomplished at the time of writing this paper. For more information, see the ”Future Steps” section.

OpenSSL does not return the results in the same format of the RSA speed results as in the NIST post-quantum encryption speed tests. OpenSSL returns the operation per second of the RSA encryption scheme. To use this data, we had to sum the total runtimes of each of the post-quantum encryption steps to create a total speed and then normalize it by the processor speed of the computer and by the operations per second of the RSA encryption test.

III. Implementation

To implement the post-quantum encryption benchmarks, the first step we need to implement is the post-quantum encryption. We do this by building different post-quantum encryption schemes. We accomplish this by running the make files provided in the various post-quantum encryption cloned git repositories. This makefile creates various binaries to run the speed tests. We then select the key length that fulfills the NIST Level 5 security specification. Then we plot the key encapsulation, decapsulation, and key pair gen. All of these values will be normalized with respect to the NewHope encryption scheme. There is no reason why NewHope was selected; it just happened to be the first algorithm that had been implemented.

Then we sum all these sections to get total CPU cycles that implementing each of these post-quantum encryption algorithms. These values are then normalized against RSA encryption speed in the library where the dependencies are built, see the Design section to see choosing the dependencies. We then convert the CPU cycles to match with RSA encryption units, so we get equivalent comparisons.

All this data will be explained and analyzed in the results section.

IV. Results

Fig: 3 shows the results of the key pair generation. As we can see from the graph, the fastest algorithm is the New Hope Encryption algorithm. Kyber closely follows it, and Frodo and NTRU are significantly slower. The main operation that influences speed in the key pair generation is producing the random polynomials for rLWE. My hypothesis why NTRU and Frodo are significantly slower than NewHope and Kyber is the use of precomputed values for the Gaussian random number generation.

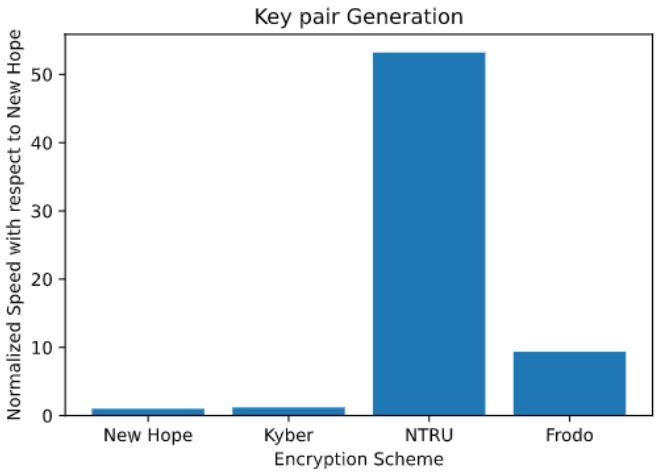

Fig. 3. Key Pair Generation

Next, in Fig: 4, we can see the speed to implement the Encapsulation (Encryption) of the post-quantum rLWE algorithms. Again the fastest algorithms are NewHope and Kyber, with the slowest being NTRU. With encapsulation, the primary operations that take time are the NTT operation and the hashing (SHAKE256). Some reasons for the increased speed of NewHope and Kyber is the precomputation of values in the NTT like precomputing in the Fast Fourier Transform (FFT). Additionally, the hashing operations in the functions are different in each of the implementations, which could also be a reason for this disparity in time to operate these different encryption algorithms.

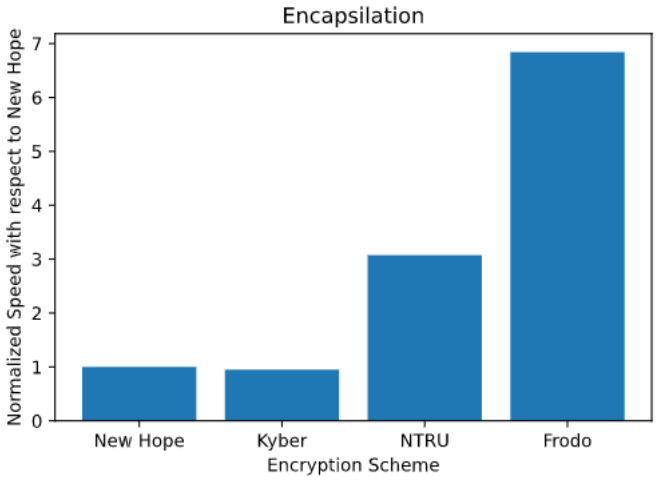

Fig. 4. Key Encapsulation

The third graph, Fig: 5, shows the decapsulation process of the post-quantum encryption algorithms. Decapsulation is incredibly similar to Encapsulation, with almost all the processes are the inverse of the Encapsulation process. So like Encapsulation, the functions dominating the Decapsulation process is NTT and hashing. The graph Fig: 5 looking almost identical to Fig: 4, and this is due to the similar operations between Encapsulation and decapsulation.

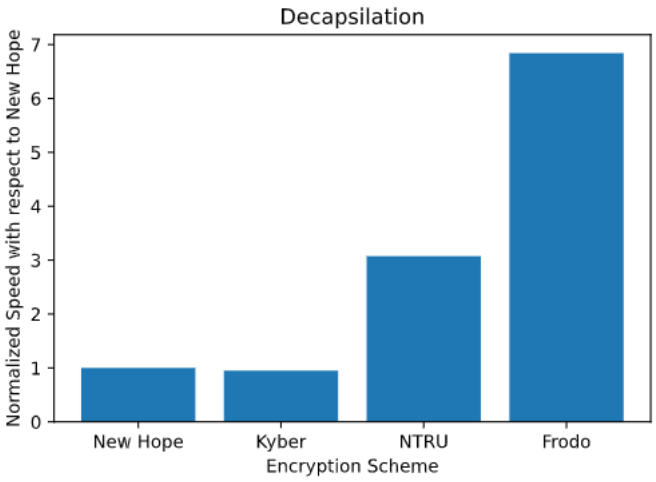

Fig. 5. Key Decapsulation

The final graph, Fig: 6 is the total amount of the different post-quantum encryption protocols. The difference in this graph is that all the values are normalized with respect to the RSA encryption speed from the OpenSSL encryption. As we can see, NewHope is the fastest of all of the post-quantum encryption algorithms and NTRU being the slowest. NewHope is faster than the RSA encryption algorithm. This faster speed is an auspicious feature for being the replacement of RSA for the post-quantum world.

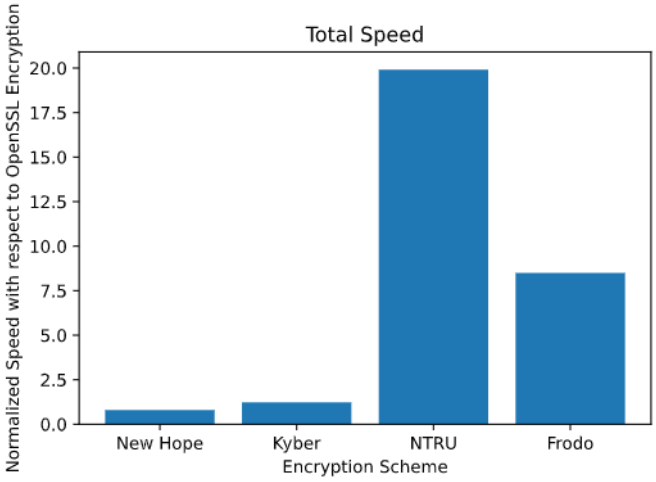

Fig. 6. Total Speed

V. Future Work

There are several advancements to this project that are poised to future work but unable to be completed due to time constraints and COVID19. The first next step that would be necessary to implement is converting OpenSSL dependencies in the post-quantum encryption schemes to the WolfSSL libraries to set up the implementation of embedded systems. This migration from OpenSSL to WolfSSL can be done using the WolfSSL compatibility layer that is designed to transfer OpenSSL to WolfSSL [15]. Additionally, there is documentation of implementing the NTRU encryption algorithm using WolfSSL that shows promise in speeding up NTRU [1].

The next future work is implementing post-quantum encryption on embedded devices. WolfSSL has built-in support for the Arduino, which seems to be a good candidate to start implementing this post-quantum encryption in an environment that is not a 64-bit machine [1]. For implementation on an MSP432P401R, though WolfSSL has support for TI RTOS, it is for the TIVA-C system and uses the NDK. I had not been able to find in the NDK for the MSP432P401R, which means that implementing WolfSSL on the system impossible.

VI. Conclusion

Overall, the world of post-quantum encryption has several promising contenders in the rLWE category to be the replacement for RSA encryption. However, as this project shows, the implementation of each of the different types of rLWE post-quantum encryption algorithms has different speeds and ways of implementing their rLWE. With the rise of fault-tolerant quantum computers obtaining a post-quantum encryption protocol is essential. Minimizing the speed of the encryption protocols will help reduce the energy costs when many devices are implementing this asymmetric encryption. From the post-quantum encryption algorithms that we have seen, the fastest is NewHope. However, Kyber is also another promising contender as even though it is not as fast as NewHope, it is close. Moving forward, this should provide a starting point to start implementing these post-quantum rLWE encryption algorithms on other devices like Arduino.

Appendix A
Repositories

The Git repositories are listed below:

- New Hope : https://github.com/newhopecrypto/newhope
- Kyber : https://github.com/pq-crystals/kyber.git
- NTRU : https://github.com/jschanck/ntru.git
- FrodoKEM : https://github.com/Microsoft/PQCrypto-LWEKE

Appendix B
Sr. Project Requirements

• Summary of Functional Requirements

Implement benchmarks of several post-quantum encryption protocols using rLWE.

• Primary Constraints

The limiting factors that impacted this project are the dependency on OpenSSL for all the different implementations of the post-quantum encryption algorithms. Other aspects that made this project challenging was trying to implement this on an MSP432P401R as I discovered the MSP432P401R was missing some development kits to implement these encryptions.

• Economic

The economic impacts of this project are enormous; encryption is the underpinning of all online systems like online banking or any online business. Though this project does not have a life cycle, these encryption protocols will change

as the NIST search for a post-quantum encryption standard continues. Additionally, at this time, any information towards selecting the candidates that make it to round three has implications on who shapes the future of encryption.

• Environmental

The ethical implication of this project is the energy usage depending on how much time the encryption takes to encode and decode data. This is because the longer it takes to run this program, the more energy it will use. This increase in time, especially when there are millions of devices can have a significant impact on the environment due to energy generation.

• Manufacturability

No issues with manufacturing as this project implemented in software.

• Sustainability

This project impacts the sustainable use of resources because making the most efficient encryption algorithm is vital to use the least amount of energy as possible. Additionally, selecting a good standard now is essential because, in the future, more secure cryptosystems are most likely to be built off of this encryption.

• Ethical

The ethical implication of this project is if there is a security flaw in the encryption algorithm and it was knowingly not mentioned, it would leave a backdoor into people's communication causing their privacy to be violated.

• Health and Safety

A safety concern with this project is the use of encryption for medical purposes.

• Social and Political

Some social and political issues associated with this project is that encryption is used for groups to communicate with each other without the view of external groups like governments and corporations. The use of encryption is critical for keeping personal data personal.

• Development

Some new things that I learned over the development of this project are Post-Quantum Encryption protocols, current asymmetric key exchange, and OpenSSL.

APPENDIX C
CODE

```
import pandas as pd
import matplotlib as mpl
from matplotlib import pyplot as plt


df = pd.read_excel('encryption_data.xlsx')


df.index = df['Test']

df_keypair = df.loc['keypair',:]
df_encaps = df.loc['encaps',:]
df_decaps = df.loc['encaps',:]

df_keypair.values



clk = 2.6e9

plt.bar(df_keypair.index[1:],df_keypair.values[1:]
plt.title('Key pair Generation')
plt.xlabel('Encryption Scheme')
plt.ylabel('Normalized Speed with respect
 ↪  to New Hope')

plt.bar(df_encaps.index[1:],df_encaps[1:]/df_encap
plt.title('Encapsilation')
plt.xlabel('Encryption Scheme')
plt.ylabel('Normalized Speed with respect
 ↪  to New Hope')


plt.bar(df_decaps.index[1:],df_decaps[1:]/df_decap
plt.title('Decapsilation')
plt.xlabel('Encryption Scheme')
plt.ylabel('Normalized Speed with respect
 ↪  to New Hope')


df.sum(axis=1,skipna=True)


a = df['New Hope'][df.index[3:]]


rsa_openssl = 973.5 /clk
b


b =[]
for i in range(1,len(a)):
    b.append( a[i]*rsa_openssl)


plt.bar(df_decaps.index[1:],b)
plt.title('Total Speed')
plt.xlabel('Encryption Scheme')
plt.ylabel('Normalized Speed with respect
 ↪  to Wolfssl Encryption')


b
```

ACKNOWLEDGMENT

The author would like to thank his senior project advisor Dr. Joseph Callenes-Sloan for providing direction and advice and being accommodating during these difficult times. I would also like to Dr. Andrew Danowitz and Dr. Paul Hummel for their advising as well. Finally, I would like to thank my colleagues, Michael Bauknecht, Brandon Nowark, and Vasanth Sadhassivan.

REFERENCES

[1] "How to get Started with wolfSSL."

[2] T. W. Edgar and D. O. Manz, "Chapter 2 - Science and Cyber Security," in *Research Methods for Cyber Security* (T. W. Edgar and D. O. Manz, eds.), pp. 33–62, Syngress, Jan. 2017.

[3] E. Conrad, S. Misenar, and J. Feldman, "Chapter 4 - Domain 3: Security Engineering (Engineering and Management of Security)," in *CISSP Study Guide (Third Edition)* (E. Conrad, S. Misenar, and J. Feldman, eds.), pp. 103–217, Boston: Syngress, Jan. 2016.

[4] robin.materese@nist.gov, "NIST Reveals 26 Algorithms Advancing to the Post-Quantum Crypto 'Semifinals'," Jan. 2019. Last Modified: 2019-01-31T16:27-05:00 Library Catalog: www.nist.gov.

[5] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. arXiv: 1801.00862.

[6] "Quantum Supremacy Using a Programmable Superconducting Processor." Library Catalog: ai.googleblog.com.

[7] Elisa Baumer, Jan-Grimo Sobez, and Stefan Tessarini, "Shor's Algorithm."

[8] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Comput.*, vol. 26, pp. 1484–1509, Oct. 1997. arXiv: quant-ph/9508027.

[9] "A Survey on Ring-LWE Cryptography."

[10] E. Alkim, R. Avanzi, J. Bos, L. Ducas, M. R. Albrecht, E. Orsini, V. Osheter, K. G. Paterson, G. Peer, and N. P. Smart, "Algorithm Specifications and Supporting Documentation," p. 48.

[11] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Algorithm Specications And Supporting Documentation," p. 37.

[12] C. Chen, O. Danba, J. Rijneveld, J. M. Schanck, P. Schwabe, W. Whyte, and Z. Zhang, "Algorithm Specications And Supporting Documentation," p. 41.

[13] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila, "Learning With Errors Key Encapsulation," p. 56.

[14] "Energy-Efficient Reconfigurable Quantum-Secure Lattice Cryptography Processor." Library Catalog: banerjeeutsav.github.io.

[15] "Migrating From OpenSSL to wolfSSL - wolfSSL," Feb. 2020. Library Catalog: www.wolfssl.com.