An analysis of LAC, a suite of lattice based post-quantum cryptosystems

CSCI 762 - Spring 2018

Aziel T. Shaw
ats9095@rit.edu

May 4, 2018

Contents

Abstract

The threat of quantum computing is steadily becoming bigger and bigger. NIST have recognized this and are holding a competition to find the best cryptographic schemes most resilient towards attacks from quantum computers. For the first round of this competition they have 69 entries to analyze and test. This paper analyzes and discusses an entry in this competition, LAC, which has four Lattice based cryptosystem schemes which promise a high resistance to most known attacks while also providing high flexibility in parameters.

In particular this paper focuses on looking at all four of LACs sub-cryptosystems titled as LAC.CPA, LAC.CCA, LAC.KE, and LAC.AKE. LAC.CPA is a Public Key encryption scheme and LAC.KE, a passively secure key exchange protocol directly converted from LAC.CPA. LAC.CCA is a secure key encapsulation method that is based on LAC.CPA. LAC.AKE is an authenticated key exchange protocol based on LAC.CPA & LAC.CCA. This paper mainly covers the public key schema, followed by a brief discussion on all of the subsystems in how they're designed, with their inputs and outputs. The correctness of the algorithms, and how well they perform compared to other, more current non-quantum safe algorithms. After describing the subsystems the Quantum resistance of the algorithms will be discussed, along with LACs advantages and disadvantages, followed by the Conclusion.

1 Introduction

This section will introduce the background behind LAC. It will discuss the NIST competition and the reasoning for needing this set of cryptosystems. This section will briefly cover the Shors algorithm before introducing the LAC cryptosystems, and some of the introductory information needed to help fully understand why LAC is considered a solid contender for the NIST competition [5].

1.1 NIST Competition

Recently quantum computers are becoming more and more likely as the amount of resources that have gone into researching them has also been increasing. This creates a need for cryptosystems which are secure against the kinds of attacks that quantum computers are being ever more proven that they can do. The threat of quantum computers being more mainstream and readily available puts the security of current cyber-communications at risk. The National Institute of Security and Technology (NIST) is holding a competition to find the cryptosystems most resistant to quantum computing. It is still unclear if a large-scale quantum computer can be built, but the risk is high enough that the security experts at NIST have started this competition [5]. A good example of the need for post-quantum cryptography is Shors algorithm named after mathematician Peter Shor. Shors algorithm solves large integer factorization in polynomial time [4]. This is a big deal as it makes RSA and other public key cryptosystem trivial to crack.

1.2 Introduction to LAC

LAC is a group of four quantum-safe algorithms based off of Lattices which are explained in more detail at Section 2. The main cryptosystem, titled LAC.CPA is the main system that the other 3 schemes are based off of, as shown in Figure 1. LAC includes a key generation algorithm which is used to create keys for the various LAC systems, but specifically for the public key encryption scheme. The public key encryption scheme (LAC.CPA) is the main schema that lays the foundation for the other three sub-schemes. The second algorithm is the secure key encapsulation mechanism called LAC.CCA. LAC.CCA is created when you take LAC.CPA and apply the Fujisaki-Okamoto transformation to the public key scheme. The Fujisaki-Okamoto transform modifies a weaker public-key encryption scheme into a strong scheme that is IND-CCA secure. It is a well known transformation which allows the increase of security among various public key cryptosystems [6].

The next scheme in this suite of post-quantum lattice based cryptoschemes is the passively secure unauthenticated key exchange protocol LAC.KE, based directly off of LAC.CPA. The final crypto scheme that's discussed in both this paper and the submission paper [7] is the authenticated

key exchange protocol based off of the public key encryption scheme (LAC.CPA), and the secure key encapsulation mechanism (LAC.CCA).

It should also be noted that LAC.CPA and LAC.CCA both adhere to different IDE security standards. There are a few different "levels" of security varying schemes can have. These levels of security include IND-CPA, IND-CCA, and IND-CCA2 [2]. These different standards show a cryptosystems *cipher-text indistinguishability* property. The higher the indistinguishability, the more secure the cryptosystem is from a malicious party to be able to tell any similarities between cipher-texts based on their plain-texts [2]. The LAC public key cryptosystem is IND-CPA secure. IND-CPA means that if an attacker is going to try an defeat the cryptosystem, the attacker has a negligibly better chance to "hack" into the system than the person would if they were to just randomly guess [10]. The second IND security standard mentioned in this paper is the LAC Key Encapsulation algorithm which is CCA secure [7]. For something to be CCA secure it needs to be indistinguishable to random noise, and assuming the attacker has access to an "oracle", it will still be provably secure among these conditions [10].

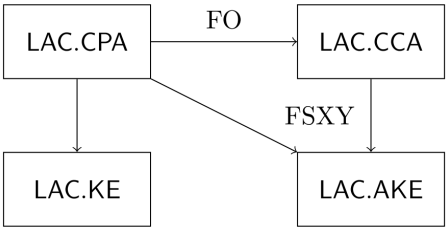

Figure 1: Architecture of LAC.

2 Lattices: definitions and problems

2.1 Lattices

A lattice is the linear combination of independent vectors (basis vectors). It is a subset of the vector space with the operations of addition, subtraction, and multiplication by an integer. A lattice L is formulated by:

$$L = \sum_{i=1}^{n} (b_i * v_i), v_i \in Z$$

where b_i is a basis.

A lattices' basis is not unique. There are infinitely many bases for a given lattice. A basis is considered good if its vectors are almost orthogonal, otherwise it is considered bad. Good bases are usually long, and bad bases are usually short. So in order to increase security and ensure the basis is a good one, algorithms must ensure that bases are given enough high dimensional vectors.

2.2 Shortest Vector Problem

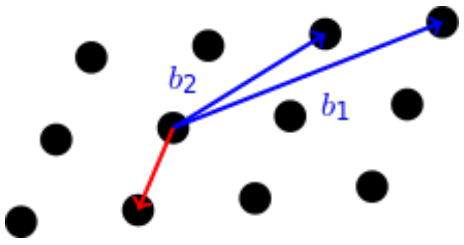

Figure 2: SVP Problem [8]

The shortest vector problem asks to find the shortest non-zero vector in a lattice L given by basis B. We know this is hard as given a basis with many vectors of high dimensionality, it is very difficult to make a combination and prove that it is the smallest non-zero vector. Since, you have to compare your vector to all other possible combinations of all the other vectors in order to possibly find a vector smaller than any of the others.

2.3 Learning With Errors

$$\begin{aligned} < s, a_1 > &\approx_\chi b_1 (\mod q) \\ < s, a_2 > &\approx_\chi b_2 (\mod q) \\ &\cdot \\ &\cdot \end{aligned}$$

Given a basis B, and a linear combination of B plus some small noise, the Learning With Errors problem attempts to distinguish the resulting linear combination plus noise from a completely random vector. This is an extension of the Learning From Parity with Errors problem.

The learning with errors (LWE) problem is provably very hard for both traditional computer and Quantum computers to solve. LWE is resistant to quantum computers as there is no *known* algorithm that is able to solve the LWE problem in polynomial time.

2.4 Ring Learning With Errors

- Learning with Errors is proven to be just as hard as the worse-case lattice problems.
- LWE is inefficient because of the inherent quadratic overhead.
- In order to make this efficient we bound the problem within a polynomial ring. This is proven under worst-case assumptions on ideal lattices to be just as hard as LWE.

Ring Learning With Errors is an extension of the Learning With Errors problem where you treat the vectors as mod n^{th} cyclotomic polynomials where n is a power of 2, and the polynomials are all elements of a Polynomial Ring. The polynomial ring that LAC uses is: $R_q = \mathbb{Z}_q[x]/(x^n + 1)$

Ring learning with errors (RLWE) takes the Learning with Errors problem and modifies it in such a way that it runs more efficiently on traditional computers. LWE operates in a potentially infinitely space, so if the vectors/points on the space get large enough it will take computers a very long time to compute things. When you convert to RLWE then you bound everything inside a range which allows computations to be much more efficient on computers. The reason this is allowed is because RLWE can be proven to be just as hard to break as LWE [7].

3 LAC Cryptosystems

This section goes into detail about the LAC cryptosystems and how they work beneath the hood. This section will start off with the design rationale before continuing into how the key generation works followed by the main cryptosystems public key Encryption/Decryption schemes. Following the Public Key scheme this paper will very briefly discuss the other three cryptosystems which are based off of the Public Key Scheme. As shown in Section 2 having a Lattice structure in your cryptosystem is very strong as the problems that come along with Lattices are very hard to crack.

3.1 Design Rationale

When the creators of LAC designed this cryptosystem they had criterion they desired to design LAC after. Part of this design required LAC to have mathematically proven levels of security based on worst-case hardness problems. In order to achieve these proven levels of security, LAC is designed around the polynomial learning with errors problem, which is discussed in Section 2.

The second point of design relates to security as well. It is a bit general as it is just about having resistance to all currently known attacks. This refers to both quantum and non-quantum

attacks. It is proven that Lattices have very hard problems, which both conventional computers and quantum computers can not solve easily.

There are three final criterion related to the design of LAC. The first two of which are about ensuring the cryptosystems all perform at a high speed, have small key sizes, and have small cipher-text sizes in order to keep the cryptosystem usable both currently and for a long period of time. The final design choice was chosen in order to help keep the cryptosystems flexible. This will allow the cryptosystems to be customizable depending on the needs of the user.

3.2 Key Generation

The Key Generation algorithm (as shown in algorithm 1) for LAC is pretty simple and straight forward. It starts by getting $seed_a$ from a random Gaussian distribution within all possible seeds in $\mathcal{S}$. This seed is used in future calculations.

Using the generated seed, get a vector $\mathbf{a}$ which is calculated based on a uniform distribution over the Ring, based on the generated $seed_a$. After this, pick a secret key $\mathbf{s}$ based on another random Gaussian distribution. Next, pick a random error $\mathbf{e}$ which will be used to add error into the calculations satisfying the LWE property. Finally, calculate $\mathbf{b}$, which is where the secret key $\mathbf{s}$ is combined with $\mathbf{a}$, and small error $\mathbf{e}$ to create a key (all bounded within the ring R_q). The reason this error is considered small is since you multiply the two known vectors $(\mathbf{a}, \mathbf{s})$ and add the error $\mathbf{e}$ which is not recorded, then the $\mathbf{e}$ will not be big enough to cause any issues as the biggest change comes from multiplying $\mathbf{a}$ & $\mathbf{s}$.

Algorithm 1 LAC Key Generation

Ensure: A pair of keys, public key and private key

1: **procedure** KeyGen( )
2: $seed_a \xleftarrow{\$} \mathcal{S}$ ▷ Pick a random number (seed)
3: $\mathbf{a} \leftarrow (U(R_q); seed_a) \in R_q$ ▷ Pick number a based on random seed
4: $\mathbf{s} \xleftarrow{\$} \Psi_\sigma^n$ ▷ Pick another random number
5: $\mathbf{e} \xleftarrow{\$} \Psi_\sigma^n$ ▷ Pick random error
6: $\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e} \in R_q$
7: **return** $(pk := (seed_a, \mathbf{b}), sk := s)$ ▷ The pair of public and private keys

3.3 Public Key Scheme

This section will cover the details relating to the public key encryption scheme contained within LAC.

3.3.1 Encryption

The encryption for the public key system (shown in algorithm 2) is pretty straight forward. It starts by passing in the public key *pk*, a message $\mathbf{m}$ and a *seed*. First, calculate $\mathbf{a}$ based on the same seed as was used to generate the $\mathbf{a}$ in the key generation. After calculating $\mathbf{a}$, encode the message using the **B**ose-**C**haudhuri-**H**ocquenghem (BCH) encoding (discussed more in section 4) to get c_m. Using the seed passed into the encryption, generate three vectors (r, e_1, e_2) over varying Gaussian distributions.

Finally, calculate the cipher-texts (c_1, c_2); starting with c_1, take the $\mathbf{r}$ generated, and multiply it with $\mathbf{a}$, add the error $\mathbf{e_1}$ just like how $\mathbf{b}$ is calculated in algorithm 1. The last cipher-text c_2 is calculated by multiplying the public key $\mathbf{b}$ with $\mathbf{r}$, adding the error $\mathbf{e_2}$ and then adding the bit multiplied by half of the LAC prime $\mathbf{q}$ (q is the largest prime number below 2^8 which is 251).

Algorithm 2 LAC.CPA Encryption

Ensure: A secure ciphertext **c**.

procedure ENCRYPTION($pk = (seed_a, \mathbf{b})$, $\mathbf{m} \in \mathcal{M}; seed \in \mathcal{S}$)
  $\mathbf{a} \leftarrow Samp(U(R_q); seed_a) \in R_q$ ▷ Choose **a** (same **a** as in ken gen)
  $c_m \leftarrow \text{ECCEnc}(\mathbf{m}) \in \{0,1\}^{l_v}$ ▷ Encode message using **BCH**
  $(r, e_1, e_2) \leftarrow Samp(\Psi_\sigma^n, \Psi_\sigma^n, \Psi_\sigma^{l_v}; \text{seed})$ ▷ Pick vectors over Gaussian distribution
  $c_1 \leftarrow \mathbf{ar} + e_1 \in R_q$ ▷ First part of ciphertext
  $c_2 \leftarrow (\mathbf{br})_{l_v} + e_2 + \bar{q} \cdot c_m \in \mathbb{Z}_q^{l_v}$ ▷ Second part of ciphertext
  return c:= $(c_1, c_2) \in R_q \times \mathbb{Z}_q^{l_v}$ ▷ Return ciphertext pair

3.3.2 Decryption

The decryption for the public key system (shown in algorithm 3) relies mainly on the BCH decoding all the bits correctly and that there are not more failed bits than allowed per each security level of LAC. It starts by generating **u**, which is taken from multiplying c_1 by the secret key s. Next, to get the encoded cipher-text, take $c_2 - \mathbf{u}$. This is where the decoding starts. If the bit being checked falls between the first quarter and the third quarter then that bit is a 1, otherwise the bit is a 0. This is where the error comes in, for the encryption, multiply the bit by half of the LAC prime (251). So if the error is too big it will mis-classify the bit. After all the bits are retrieved then the cipher-text is BCH decoded.

$$\begin{aligned} c'_m &= c_2 - (u)_{l_v} \in \mathbb{Z}_q^{l_v}, \\ &= ((as + e)r)_{l_v} + e_2 + \bar{q} \cdot c_m - ((ar + e_1)s)_{l_v}, \\ &= w + \bar{q}c_m \end{aligned}$$

As shown in the algorithm above, after the cipher-text is decrypted then get the result bit plus a little bit of error (shown as **w**). If the error (**w**) for that bit is too big then the bit being decrypted will be decrypted incorrectly. The specifics of this are discussed more in section 4.

Algorithm 3 LAC.CPA Decryption

Ensure: A correct plaintext **m**.

procedure DECRYPTION(sk = s, c = (c_1, c_2))
  $\mathbf{u} \leftarrow c_1 s \in R_q$
  $c'_m \leftarrow c_2 - (u)_{l_v} \in \mathbb{Z}_q^{l_v}$ ▷ Decrypt (Noise stays)
  for $i = 0$ to $l_v - 1$ **do**
    if $\frac{q}{4} \leq c'_{mi} < \frac{3q}{4}$ **then** ▷ If c between bounds
      $c_m i \leftarrow 1$ ▷ Bit is 1
    else
      $c_m i \leftarrow 0$ ▷ otherwise bit is 0
    end if
  end for
  $\mathbf{m} \leftarrow \text{ECCDec}(c_m)$ ▷ Decode message
  return m ▷ Return decrypted message

3.3.3 ECC Encoding/Decoding

The ECC Encoding is one of the most important parts of LAC. The specifics of why are discuss in section 4. The encoding (shown in Algorithm 4) works by padding zeros to the message (as to satisfy one of the requirements of BCH), followed by performing BCH encoding.

The decoding (shown in Algorithm 5) takes the cipher-text and removes the leading zeros before using the BCH Decoding algorithm to decode the message to the original message.

In an email correspondence between LAC and the NIST moderators, the moderators recommended that the authors used a different error correcting algorithm such as the binary

Goppa codes, as the maximum code length is longer, an additional information bit is able to be corrected, padding will no longer be needed, has been well researched and proven over the past 10 years, and can be easily implemented over the existing BCH decoding.

Algorithm 4 LAC.CPA Error Correction Encoding for Encryption

Ensure: An encoded $c_m \in \{0,1\}^{l_v}$.

1: **procedure** EncEncoding($m \in \mathcal{M}$)
2: $\{0,1\}^{n_e}$
3: $c_m \leftarrow \{0,1\}^{l_v}$
4: **return** c_m

Algorithm 5 LAC.CPA Error Correction Decoding for Decryption

Ensure: A bit string $\mathbf{m}$.

1: **procedure** DecDecoding($\mathbf{c_m} \in \{0,1\}^{l_v}$)
2: $\hat{\mathbf{c}} \leftarrow (\mathbf{c_m})_{n_e}$
3: $\mathbf{m} \leftarrow \text{BCHD}(\hat{\mathbf{c}}) \in \{0,1\}^{l_e}$
4: **return** $\mathbf{m}$

3.4 Other LAC Schemes

This section will briefly cover the other 3 post-quantum cryptoschemes under LACs umbrella. These cryptosystems are all based off the public key cryptosystem. All 3 in one form or another build off of the public key foundation usually with some kind of transform.

3.4.1 Key encapsulation mechanism

The first LAC.CPA extension is the secure key encapsulation method (LAC.CCA). The key exchange mechanism is generated by applying the Fujisaki-Okamoto transformation into the public key cryptosystem. This key exchange mechanism works very similar to the public key system, but with added features that make it IND-CCA secure. It does this by using hashes and a round of extra encryption to verify the encapsulation went smoothly. The performance of LAC.CCA is two to three times as slow as LAC.CPA due to the extra computations and steps taken to ensure the extra security.

3.4.2 Passively secure Key exchange protocol

LAC.KE is an unauthenticated key exchange protocol derived straight from LAC.CPA. This key exchange protocol uses a hash function with an output equal to the length of the session key. The hash functions used for this are SHA256, SHA384, and SHA512, depending on the LAC security level chosen. The authors note than the key encapsulation method is only passively secure when there is no key caching.

3.4.3 Authenticated Key exchange protocol

There is a second key exchange protocol within LAC called LAC.AKE. This key exchange protocol is the authenticated key exchange protocol. It is built off of LAC.CPA and LAC.CCA. The authenticated key exchange is considered to be Canetti-Krawczyk secure as well as resistant to attacks such as key compromise impersonations, and maximal exposure attacks [7].

4 Error Correcting Codes

Error correcting codes (ECC) are the key for this cryptosystem to work. This paper will not talk in detail about the specific ECC used in LAC, but instead will discuss ECC in general, why they

Algorithms	n	σ	δ	δ_e	δ_t	Δ
LAC128	512	1	$2^{-13.35}$	$2^{-13.17}$	$2^{-13.41}$	$2^{-239.6}$
LAC192	1024	1/2	$2^{-24.51}$	$2^{-24.44}$	$2^{-24.74}$	$2^{-253.8}$
LAC256	1024	1	$2^{-7.44}$	$2^{-7.34}$	$2^{-7.48}$	$2^{-115.4}$

Table 1: Failure probability of the public Key cryptosystem

are important, and how they are used to make LAC a strong candidate for the post-quantum cryptosystem competition.

Error correction in the public key cryptosystem is started in algorithm 4, where the message is padded with zeros to satisfy the parameters of the BCH encoding. It really comes into use in algorithm 5, where the leading zeros are unpadded for the BCH decoding and then computations are done to recover the bits.

Because error is introduced into the cipher-text through picking various error vectors (such as $\mathbf{e}$, $\mathbf{e_1}$, and $\mathbf{e_2}$), when the decryption happens in algorithm 3, the cipher-text might not be correctly retrieved. The *if* statement in line 5 of algorithm 3 is where the cipher-text is converted back into the message. If the error is not big then the bit will be decrypted correctly. If the error for that bit is too high, then the bit will not be decrypted correctly.

Table 1 talks about the failure probabilities of each of the LAC security levels. Table 1 shows a lot of important information regarding the error correction of LAC. For example σ shows the Gaussian distribution for each security level, and n shows the size of the vectors retrieved when getting the distribution. All the δ parameters show the probability of a single bit being unable to be properly recovered. δ shows the probability that a single bit will not be recovered, δ_e shows the Gaussian estimation of a bit being unable to be recovered (very similar results as to δ), and δ_t shows the results from the authors tests of bit corruption (based on $l_m \times 10^7$ long, random messages). The final, and probably the most important column is the Δ column. This column tells us the probability of a whole message being unable to be correctly decrypted for each security level. It should be noted that LAC128 can have at a maximum **30** bits be decrypted incorrectly, LAC192 can only have **14** fail, and LAC256 can have **57** bits be incorrect [7].

5 Quantum Resistance of LAC

The authors claim that LAC has high and concrete security. As LAC relies on the fact that LWE is a difficult problem it makes sense to prove that LWE is itself quantum resistant. There are allegedly some general algorithms that can be used to solve the LWE problem according to certain parameters [1, 3]. These papers discuss some reduction attacks based on the BKZ algorithm which computes vectors such that it becomes the smallest vector [9] (e.g. solves the smallest vector problem). These attacks that are based on BKZ are more powerful than an exhaustive search algorithms, algebraic algorithms, and combinational algorithms.

The authors discuss two simple but generic attacks that can be performed on LAC, and how well LAC resists these attacks. The first attack discussed is the Primal attack. A Primal attack is one where a specific SVP instance from LWE. The attack is only successful if the standard deviation multiplied by the$\sqrt{b}$ is less than the probability of bit failure multiplied by the LAC prime number (251). The second attack is the Dual attack. The Dual attack takes advantage of basis Lattices which are considered bad Lattices. BKZ is used in this attack to find a particular vector which can then break the decision LWE problem with an advantage greater than brute forcing and random guessing.

It should be noted that BKZ relies on reducing a lattice basis to its easiest form using an SVP oracle [9]. The reason these attacks are not viable is that it is extremely difficult to estimate the amount of calls needed to talk to the oracle. Therefore the fact these algorithms can be completed in polynomial time is considered *null*, and the algorithms still hold to the base hardness of the SVP problem. Table 2 shows the claimed security of LAC based on these two attacks. It should be noted that the actual security levels of LAC against these attacks are actually stronger

Algorithm	Claimed Security	Primal Attack		Dual Attack		Security Categories
		Classical	Quantum	Classical	Quantum	
LAC128	128	148	135	147	133	1, 2
LAC192	192	288	261	286	259	3, 4
LAC256	256	323	293	320	290	5

Table 2: Security claims of LAC.CPA

Categories	Key Generation		Encryption		Decryption	
	CPU Cycles	Times	CPU Cycles	Times	CPU Cycles	Times
LAC128	90686	29.25	152575	49.22	68285	22.03
LAC192	309216	99.75	410469	132.41	238268	76.86
LAC256	269827	87.04	513753	165.73	336207	108.45

Table 3: LAC public key Performance without AVX2 (times in microseconds)

than the claimed security. With the Quantum security only being slightly less secure than on a classical computer, but still stronger than the claimed security levels.

6 Performance

The authors of LAC wrote up up the schemes in C in order to test how well the cryptosystems would perform, as shown in table 3. Due to the vector calculations that LAC uses, there is a way to increase the performance of the cryptosystem depending on if the processors have the AVX2 (Advanced Vector Extensions) instruction sets. The AVX2 instruction set allows faster computations of vector math, which speeds up LAC.CPA by up to two to three times as fast. Most modern processors (Intel i3/i5/i7/i9 from 2013 and later, and AMD Excavator Processors and later) have the AVX2 instruction sets on them. The performance of LAC on processors with the AVX2 instruction set is shown in table 4.

7 Advantages and Disadvantages

LAC is designed to have a lot of reasons people would choose it over other systems. The first of which is that it is very fast. Especially on processors that support vector computations. LAC is also designed to be parallel by default. The authors wanted to ensure that LAC was also simple to understand. They achieved this by using a small modulus to reduce the size of keys and ciphertexts, and uses easy to understand computations. The final design category used by LAC was to allow it to be flexible. LAC allows you to change the strength of the error correcting code to help speed up, or slow down the processing at the cost of being able to retrieve your data. You are also able to increase the "bit strength" of the cryptosystems by changing the size of the errors, and distributions in order to increase the security and hardness of the cryptosystems.

However, not everything about LAC is good. The cryptosystems themselves have some attributes about them that make it not as good of a choose. The main and only one talked about in this paper is the fact that it is unable to be sped up by the Number Theoretic Transform (NTT) which is a specialization of the Fast Fourier Transformation (FFT), on processors that

Categories	Key Generation		Encryption		Decryption	
	CPU Cycles	Times	CPU Cycles	Times	CPU Cycles	Times
LAC128	38957	12.56	53357	17.21	27259	8.79
LAC192	114753	37.02	117749	37.98	54313	17.52
LAC256	78678	25.38	137258	44.28	133379	43.03

Table 4: LAC public key Performance with AVX2 (times in microseconds)

do not support vector calculations. NTT is mainly useful for polynomial multiplication within a polynomial ring. Such as the one that's used in LAC. Another disadvantage is that if a bad basis is chosen for the Lattice, then the cryptosystem is vulnerable to BKZ attacks [1, 3].

8 Conclusion

The LAC cryptosystems are well thought out and well constructed. The four cryptosystems all have their strengths and weaknesses in regards to security and speed. Each sub-system has an IDE standard of security showing that it reaches a certain level of security which is helpful in deciding whether or not it is worth using a particular cryptosystem. The public key cryptosystem seems to be very resistant to post-quantum attacks, mainly due to the high error introduced in the encryption, the high error correction coding redundancy, and its design based on LWE. Despite the high error introduced into the system, LAC is still able to correct any error introduced through using the carefully chosen BCH encoding.

LAC has many advantages, such as its design. The authors constructed it to be fast performing, and take up minimal space so if should run on most computers. As LAC is based on the LWE problem, which at the time of writing has no known polynomial time algorithms to solve LWE, both on classical and quantum computers. Overall the authors have a solution for most current risks towards todays standard cryptosystems. As LAC is well performing with a high level of quantum security, and provably correct to a certain level depending on the security level chosen thanks to the BCH encoding with the error, it may be able to make it further in the NIST competition.

References

[1] Albrecht, M. R., Player, R., and Scott, S. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046, 2015. `https://eprint.iacr.org/2015/046`.

[2] anonymous. Easy explanation of "ind-" security notions?, Jul 2015.

[3] Bindel, N., Buchmann, J., Göpfert, F., and Schmidt, M. Estimation of the hardness of the learning with errors problem with a restricted number of samples. Cryptology ePrint Archive, Report 2017/140, 2017. `https://eprint.iacr.org/2017/140`.

[4] Blanda, S. Shor's algorithm – breaking rsa encryption, Jun 2014.

[5] Division, C. S., Laboratory, I. T., of Standards, N. I., Technology, and of Commerce, D. Post-quantum cryptography | csrc, 2016.

[6] Hofheinz, D., Hövelmanns, K., and Kiltz, E. A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, Report 2017/604, 2017. `https://eprint.iacr.org/2017/604`.

[7] Lu, X., Liu, Y., Jia, D., Due, H., He, J., and Zhang, Z. Lac lattice-based cryptosystems, Dec 2017.

[8] Schmittner, S. Lattice problem, Apr 2018.

[9] van de Pol, J. The bkz algorithm, 2018.

[10] Xiao, L., and Yen, I.-L. *Security Analysis and Enhancement for Prefix-Preserving Encryption Schemes.*