Verification of Post-Quantum Signature Algorithms using Quaternion Algebra

Elmir Dzaka, Fernando Araujo

May 5, 2024

Abstract

This project explores recreating and understanding a proposed post-quantum signature scheme that utilizes Schnorr's scheme, using a finite non-commutative associative algebras (FNAA) ring. The ring utilizes non-commutative quaternion algebra to generate and verify signatures developed from a hidden discrete logarithm problem (HDLP) basis and given a message. This project taught us how to create keys and signatures using an HDLP within an FNAA ring and how the security differs from practices we learned in class.

1 Introduction

In recent years, quantum computing has posed unprecedented challenges to conventional cryptographic systems, threatening the security of sensitive information across various domains. To mitigate this looming threat, researchers have turned their attention to post-quantum cryptography (PQC), aiming to develop cryptographic protocols resistant to attacks from quantum computers. This paper delves into utilizing Non-Commutative Finite Associative Algebra (FNAA) as a promising avenue for constructing robust signature schemes in PQC. FNAA offers a unique mathematical framework that diverges from classical cryptographic approaches, harnessing the algebraic properties of non-commutative structures to fortify cryptographic primitives by designing post-quantum cryptoschemes implementing what is known as the hidden discrete logarithm problem (HDLP) [1], [2].

The discrete logarithm problem (DLP) involves solving the equation $Y = G^x$ in a finite cyclic group (modulo a prime number), where x is unknown. In HDLP, the values of Y, G, or both are hidden, as this is possible if the cyclic group represents a subset of some set of algebraic elements [2], [3]. Therefore, FNAA implementation is used for this project. HDLP is a computationally difficult problem that is suitable for designing new public key cryptosystems, as this paper shows. Implementing these public keys is meant to compute private key 4D vectors to generate two digital signatures using Schnorr's scheme with an SHA-256 hash function, then verify the signatures [1]. In this project, we aim to verify a proposed digital signature scheme using 4-dimensional (4D) FNAAs as the algebraic support of an HDLP to design one of three digital signature algorithm schemes.

2 Preliminaries

The implementation of FNAAs to construct a post-quantum public-key cryptoschemes is based on some computationally tricky problems that are different from the factorization and DLP as quantum computers can break them in polynomial time [1], [4]. Therefore, it is keen to use an HDLP basis to construct public keys to resist quantum computer attacks.

To create post-quantum signature schemes using FNAAs, we need to understand what FNAAs are and how a ring can be made using non-commutative algebra. Non-commutative algebra is a set of algebras whose multiplication is not commutative. In other words, $a \circ b \neq b \circ a$; however, they are associative, $a \circ (b \circ c) = (a \circ b) \circ c$. When applied to cryptography schemes, each multiplication operation must be in a specific order and be derived from non-commutative properties. To allow for this to work, Hamiltonian Quaternion (denoted as $\mathbb{H}$) is used since they are associative [5].

A Quaternion is represented as a group of complex numbers. These complex numbers are a sum of real and imaginary parts $a \cdot 1 + b \cdot i$. Since Quaternions are non-commutative associative, they can also be represented as a linear combination $\mathbb{H} = a \cdot \mathbf{1} + b \cdot \mathbf{i} + c \cdot \mathbf{j} + d \cdot \mathbf{k} = (a, b, c, d)$. We can use these linear combinations to create keys and signatures in a cryptography scheme. The multiplication operations that can be performed are represented using a basis-vector multiplication table (BVMT) and seen in Table 1. Table 2 shows the same logic but derived within the referenced paper used for this project [6], [1]. Although never stated, the basis-vector multiplication derived in the referenced documents is just Quaternion algebra logic, as shown when comparing Tables 1 and 2.

Table 1: Quaternion BVMT [5]

$\circ$	1	$\mathbf{i}$	$\mathbf{j}$	$\mathbf{k}$
1	1	$\mathbf{i}$	$\mathbf{j}$	$\mathbf{k}$
$\mathbf{i}$	$\mathbf{i}$	-1	$\mathbf{k}$	$-\mathbf{j}$
$\mathbf{j}$	$\mathbf{j}$	$-\mathbf{k}$	-1	$\mathbf{i}$
$\mathbf{k}$	$\mathbf{k}$	$\mathbf{j}$	$-\mathbf{i}$	-1

Table 2: Non-Commutative BVMT [3]

$\circ$	$\overrightarrow{e}$	$\overrightarrow{i}$	$\overrightarrow{j}$	$\overrightarrow{k}$
$\overrightarrow{e}$	μe	μi	μj	μk
$\overrightarrow{i}$	μi	$-\mu^{-1}\lambda e$	k	$-\lambda j$
$\overrightarrow{j}$	μj	$-k$	$-\mu^{-1} e$	i
$\overrightarrow{k}$	μk	λj	$-i$	$-\mu^{-1}\lambda e$

When both tables are compared, it shows that the two BVMTs are almost identical; therefore, when performing the multiplication between two 4D vectors, we assumed that $\mu, \lambda = 1$. As a result, the multiplication outcome should be represented as follows:

$$a = a_0 \cdot \mathbf{1} + a_1 \cdot \mathbf{i} + a_2 \cdot \mathbf{j} + a_3 \cdot \mathbf{k} \qquad b = b_0 \cdot \mathbf{1} + b_1 \cdot \mathbf{i} + b_2 \cdot \mathbf{j} + b_3 \cdot \mathbf{k}$$

$$\begin{aligned} a \circ b = (a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3) \cdot \mathbf{1} + (a_0 b_1 + a1 b_0 + a_2 b_3 - a_3 b_2) \cdot \mathbf{i} \\ +(a_0 b_2 - a_1 b_3 + a_2 b_0 + a_3 b_1) \cdot \mathbf{j} + (a_0 b_3 + a_1 b_2 - a_3 b_1 + a_3 b_0) \cdot \mathbf{k} \end{aligned} \tag{1}$$

Various operations were used to describe quaternion algebra [5]; the one shown above is what is most important. Throughout this research, it is revealed that other FNAAs can also be used to perform the post-quantum signature generation and verification [6], but for this project, implementing quaternion algebra seemed to be the simple option.

3 Methodology

To create an implementation of the proposed post-quantum signature scheme, we used the Python module **pyquaternion** [7] to handle the non-commutative algebras needed to create the FNAA ring as described in the preliminary section.

3.1 Declaring the initial vectors and cyclic group

To begin, it is said in the research papers [1], [8] that when deciding the cyclic group for the operation, choosing a huge prime number q, however, for this project, we decided to choose $q = 7$ to avoid further complexity to the FNAA math, as suggested by Dr. Kalla.

Additionally, we must declare the initial vectors that will be used to form the HDLP basis of our public keys. These vectors include 1 non-invertible vector $N = (n_0, n_1, n_2, n_3)$ and 2 invertible vectors $A = (a_0, a_1, a_2, a_3)$ and $B = (b_0, b_1, b_2, b_3)$ of order q [1]. Rules of FNAA define non-invertible (2) and invertible (3) vectors as follows:

$$\textbf{Non-invertible: } n_0 \cdot n_1 = n_2 \cdot n_3, \tag{2}$$

$$\textbf{Invertible: } a_0 \cdot a_1 \neq a_2 \cdot a_3, \tag{3}$$

based on the information above, we've selected these initial vectors and value of q:

$$q = 7; \quad A = (1, 2, 3, 4); \quad B = (2, 3, 4, 5); \quad N = (2, 1, 1, 2).$$

Then, ensured that the following conditions were satisfied:

$$A \circ N \neq N \circ A; \quad B \circ N \neq N \circ B; \quad A \circ B \neq B \circ A$$

3.2 Generate the local and global units of the HDLP

When generating the local keys, the non-invertible vector N is a determining factor in developing the local units. This is because the majority of non-invertible vectors are locally invertible, meaning invertible relative to some local two-sided unit that acts in frame of some subset of the set of non-invertible vectors, including the fixed vector N [1]. The notion of local units is used in defining forms of HDLP for left-sided (L_N) and right-sided (R_N) units related to N:

$$XN = N \to L_N = (d, h, \frac{n_1(1-\lambda h)}{\mu n_3}, \frac{n_0(1-\mu d)}{\lambda n_2}), \quad d, h = 0, 1, ..., p-1. \tag{4}$$

$$NX = N \to R_N = (d, h, \frac{n_0(1-\mu d)}{\lambda n_3}, \frac{n_1(1-\lambda h)}{\mu n_2}), \quad d, h = 0, 1, ..., p-1. \tag{5}$$

where $p = 2q + 1$, d and h are non-negative independent variables within the range $p-1$. The various outcomes of vectors L_N and R_N contain a finite number of invertible $(\mathbf{p^2 - p})$ and non-invertible $(\mathbf{p})$ vectors, but to randomly choose a vector for signal generation can be based on the value of d, which can be discovered looping through the following equation:

$$E'_N = (d, \frac{\lambda n_1 - \mu n_0 + \mu^2 n_0 d}{\lambda^2 n_1}, \frac{n_0(1-\mu d)}{\lambda n_3}, \frac{n_0(1-\mu d)}{\lambda n_2}), \quad d = 0, 1, ..., p-1. \tag{6}$$

The unit E is known as the global and acts as the unit on every algebra element. The set of vectors coming from equation (6) contains a finite set of invertible $(\mathbf{p-1})$ vectors and only **1 non-invertible vector** , which can be computed using the following formula:

$$E''_N = (\frac{n_0}{\lambda n_1 + \mu n_0}, \frac{n_1}{\lambda n_1 + \mu n_0}, \frac{n_2}{\lambda n_1 + \mu n_0}, \frac{n_3}{\lambda n_1 + \mu n_0}); \tag{7}$$

however, we soon discovered that equation (7) did not work as intended. So, with further analysis of research articles [1], [2], [3], [8], we discovered that the hidden group of vectors determined by the fixed non-invertible vector N: L_N (4), R_N (5), E'_N (6), should have at least one matching vector to their rotation counterpart corresponding to the same independent variable d. (**NOTE:** L_N and R_N both contain $p^2 = 225$ quaternion vectors. Therefore, we must reduce the number of vectors to use.)

The rotation counterparts are a more simplified way of choosing a value d that leads to the invertible vector E''_N; therefore, the left-sided rotation (L'_N) and right-sided rotation (R'_N) units are vital in finding the correct generator value d to calculate the public keys. We can calculate the rotation units as follows:

$$L'_N = (d, \lambda^{-1} - \mu\lambda^{-1}d, \frac{n_1}{n_3}d, \frac{n_0(1-\mu d)}{\lambda n_2}), \quad d = 0, 1, ..., p-1. \tag{8}$$

$$R'_N = (d, \lambda^{-1} - \mu\lambda^{-1}d, \frac{n_0(1-\mu d)}{\lambda n_3}, \frac{n_1}{n_2}d), \quad d = 0, 1, ..., p-1. \tag{9}$$

To find the correct generator value d, the articles suggest that the local (L_N, L'_N, R_N, R'_N) and global (E'_N, E''_N) units should have at least one generator value that allows the local and global units to equal their corresponding rotation units [1]. We would utilize Python to loop through all p^2 vectors of L_N and R_N and compare them with L'_N and R'_N to ensure that at least one vector is matched to its counterpart for every value d.

Algorithm 1 Generator calculation d and h pairs using local units

procedure VERIFY_GENERATOR(N, q)
▷ Discovers pairs of d and h values where $L_N = L'_N$ and $R_N = R'_N$

$p = 2q + 1; \quad \lambda = 1; \quad \mu = 1$ ▷ Initialize p, λ, and μ
for h in range(p) **do**
  for d in range(p) **do**
    $L_N(N, d, h) \mod q$ ▷ Calculate L_N using equation (4)
    $L'_N(N, d) \mod q$ ▷ Calculate L'_N using equation (8)
    $R_N(N, d, h) \mod q$ ▷ Calculate R_N using equation (5)
    $R'_N(N, d) \mod q$ ▷ Calculate R'_N using equation (9)
    if $L_N == L'_N$ and $R_N == R'_N$ **then** ▷ If statement for comparison
      print(d, h) ▷ Output the values of d and h pairs
return $TRUE$

By implementing algorithm (1), we discovered that the d and h pairs were inversely proportional for every output. This isn't a significant discovery, as we only needed to ensure that the values of $d = 0, 1, ...p-1$ contribute to the findings discussed in the research articles [1], [2], that there exist one vector where $L_N(N, d, h) = L'_N(N, d)$ and

$R_N(N, d, h) = R'_N(N, d)$ for every value d. The same approach can be implemented for the global units, but only to find the non-invertible vector defined in the research article [1].

Algorithm 2 Generator calculation d using global units

procedure FIND_NON_INV_VEC(N, q)
▷ Discovers the generator d value where E'_N is non-invertible

$E''_N(N) \mod q$
▷ Calculate E''_N using equation (7) [**NOT USED** because $E'_N \neq E''_N$ for all d values]

$p = 2q + 1; \quad \lambda = 1; \quad \mu = 1$ ▷ Initialize p, λ, and μ
for d in range(p) **do**
  $E'_N(N, d) \mod q$ ▷ Calculate E'_N using equation (6)
  if $E'_N[0] \cdot E'_N[1] == E'_N[2] \cdot E'_N[3]$ **then** ▷ If statement for invertibility (2)
    return d ▷ Output the value of generator d
return -1 ▷ If no invertible vector if found, return -1

By implementing algorithm (2), we would find the value d that provides a non-invertible vector E'_N, which would have been calculated using equation (7), but because $E'_N \neq E''_N$, the equation wouldn't be helpful for our specific examples. However, since the condition that needed to be satisfied is defined by equation (2), we bypassed the need to calculate E''_N. We can now define our generator value $d = 3$ and use it to represent our local and global vectors ($L'_N(N, d = 3)$, $R'_N(N, d = 3)$, and $E'_N(N, d = 3)$) to generate the public keys.

3.3 Public and Private Key Generation

We can generate the public keys now that the local and global units have been successfully obtained. This process consists of computing three non-invertible vectors (Y, Z, T). The elements within the public key quaternions are expected to be in order q. In this process, we will generate a random integer $x < q$ that will be used as a secret key for the following algorithm. Additionally, when the public keys have been generated, two vectors will be calculated (J, U) and used for signature generation to represent the private keys. Algorithm (3) describes the process for public and private key generation needed to compute and verify digital signatures [1]. Within the test setup, we randomly generate the secret key x; therefore, results for the public and private keys may turn out differently every time the code is run.

Algorithm 3 Public and Private Key Vector Generation

procedure KEY_GEN(A, B, N, x, d) ▷ Generates the Public and Private Keys
  Generate a non-invertible vector N of order q.
  Generate two random invertible vectors A and B of order q satisfying:
    $A \circ N \neq N \circ A$, $B \circ N \neq N \circ B$, and $A \circ B \neq B \circ A$.
  Generate the local left-sided unit $L'_N(N, d)$ using equation (8).
  Generate the local right-sided unit $R'_N(N, d)$ using equation (9).
  Generate a random non-negative integer $x < q$.
  Compute the vector $Y = A \circ N^x \circ L'N \circ A^{-1}$.
  Compute the vector $Z = B \circ R'N \circ N \circ B^{-1}$.
  Generate the global unit $E''_N = E'_N(N, d)$ using equation (6).
  Compute the vector $T = B \circ E''N \circ A^{-1}$.
▷ Public Keys (Y, Z, T) have been generated for verification; now perform private key (J, U) generation.
  Compute the vector $J = B \circ R'_N$.
  Compute the vector $U = L'_N \circ A^{-1}$
▷ Vectors N, J, and U represent the private keys needed to compute digital signatures

3.4 Digital Signature Generation for a 4D FNAA

The signature generation scheme we will use for this project can be found in "*4.1.1. Signature generation algorithm A*" [1] that implements a variation of Schnorr's scheme using HDLP and FNAAs. To generate the digital signature, we need a couple of inputs. A message M that contains contents that we will hash, allowing us to calculate the first signature element v. We used the message: $M =$ **"Hello World"**. We must also generate another random integer $k < q$ to compute the vector V used to hash our message. These inputs are used to calculate the second signature element s, which uses the Schnorr scheme to output the two signature elements and the message for signature verification. Algorithm (4) best describes the process involving the private keys mentioned in section 3.3. After the algorithm has been used, we may proceed to the signature verification of the generated signature elements v and s.

Algorithm 4 Signature Generation Algorithm A

procedure SIG_GEN(M, k, N, J, U, x, q) ▷ Generates the Digital Signature (v, s)
  Select random an integer $k < q$;
  $V = J \circ N^k \circ U$; ▷ Compute vector V for hashing
  $v = F_h(M, V)$; ▷ Compute the 1^{st} signature element v.
▷ F_h performs a SHA-256 hash function.
  $s = k + x \cdot v \bmod q$; ▷ Compute the 2^{nd} signature element s.
  OUTPUT the signature (v, s) to the document M.

3.5 Digital Signature Verification for a 4D FNAA

The signature verification scheme we will use for this project can be found in "*4.1.2. Signature verification algorithm A*" [1] that verifies Schnorr's scheme of the previous algorithm. We must use the generated public keys mentioned in section 3.3 to verify the digital signature. The following algorithm uses the outputs from Algorithm (4) and utilizes the same hash function (SHA-256). Algorithm (5) best describes the process using the mentioned inputs to generate a new hash v' and compares itself to v. If $v = v'$, the signature is genuine; otherwise, the signature is incorrect and has been compromised.

Algorithm 5 Signature Verification Algorithm A

1: **procedure** SIG_VER(v, s, M, Y, T, Z) ▷ Verifies the Digital Signature (v, s)
2: $V' = Y^{-v} \circ T \circ Z^{s}$; ▷ Compute the vector V' for hashing
3: $v' = F_h(M, V')$; ▷ Compute the hash value v'
4: **if** $v' == v$; **then** ▷ Verify the condition statement
5: **return** $1, TRUE$ ▷ The signature is accepted as genuine
6: **else**
7: **return** $0, FALSE$ ▷ The signature is rejected

3.6 Signature Scheme Proof

Throughout this project, most of the work went into understanding the FNAA side of things and gaining the knowledge to perform the operations to achieve some results. The complex work that went into solving these algorithms and attempts to explain why the researchers did what they did makes this implementation post-quantum. It turns out that performing an HDLP basis is what makes the project post-quantum as it diverges into cyclic groups of primitive elements of $\mathbb{Z}_p$, similar to or precisely to Schnorr's Scheme discussed in class [9]. The following calculation is a proof of the digital signature scheme implemented for the project:

$$V' = (B \circ R'_N \circ N \circ B^{-1})^s \circ (T) \circ (A \circ N^x \circ L'_N \circ A^{-1})$$

$$= B \circ (R'_N \circ N)^s \circ B^{-1} \circ (B \circ E''_N \circ A^{-1}) \circ A \circ (N^x \circ L'_N)^{-v} \circ A^{-1}$$

$$= B \circ R'_N \circ N^s \circ N^{-vx} \circ L'_N \circ A^{-1}$$

$$= J \circ N^{k+vx} \circ N^{-vx} \circ U = J \circ N^k \circ U$$

$$\Rightarrow V' = V \Rightarrow v' = v.$$

4 Results

The results of this test setup come from *quaternion.py* file. The code can be found in the Github repository: https://github.com/fernarau9652/HW-Crypto-Project.

4.1 STEP 1: Verifying Vector Correctness

As section 3.1 mentions, we developed the test setup from three quaternion vectors, two invertible and one non-invertible of cyclic order q.

$$q = 7; \quad A = (1, 2, 3, 4); \quad B = (2, 3, 4, 5); \quad N = (2, 1, 1, 2).$$

To verify the correctness of these vectors, they must satisfy the following conditions:

$$\begin{aligned} A \circ N \mod 7 &\neq N \circ A \mod 7 \Rightarrow (3, 0, 0, 2) \neq (3, 3, 0, 4); \\ B \circ N \mod 7 &\neq N \circ B \mod 7 \Rightarrow (1, 4, 2, 6) \neq (1, 5, 4, 1); \\ A \circ B \mod 7 &\neq B \circ A \mod 7 \Rightarrow (6, 6, 5, 5) \neq (6, 1, 1, 0). \end{aligned}$$

These conditions have been met using equation (1) to prove non-commutative properties; now, we can develop the HDLP basis.

4.2 STEP 2: Verify generators d, h, for L_N and R_N

The HDLP basis can be formed using various algorithms discussed in this paper. Algorithm (1) compares the local units to derive non-invertible vectors for public keys and describes pairs of independent variables (d, h) that help us ensure public and private key generation is possible. It can be seen that the pairs of d and h are inversely pro-

$(1, 0)$	$(12, 3)$	$(8, 7)$	$(4, 11)$
$(0, 1)$	$(11, 4)$	$(7, 8)$	$(3, 12)$
$(14, 1)$	$(10, 5)$	$(6, 9)$	$(2, 13)$
$(13, 2)$	$(9, 6)$	$(5, 10)$	$(1, 14)$

Table 3: Pairs of (d, h) derived from Algorithm (1)

portional. The reason for this can be due to the properties of how equations (4) and (5) are relatively similar and differ from a change in the numerator. Not to mention that λ and μ both equal 1, which ultimately provided pairs derived from equations (8) and (9). However, the generation of d and h work and provides a motive to discover the actual generation integer for d.

4.3 STEP 3: Find unique Non-invertible vector

Given that there are generator pairs, we need to discover the only non-invertible vector shared between the global units E'_N and E''_N. Since it is discussed that E'_N contains at least one non-invertible vector, we must implement Algorithm (2) to find the generator value d, which will be implemented to calculate the final local and global units for the

development of the HDLP basis. Through implementation and utilization of equation (2), we found that $d = 3$ resulting in: $(\lambda, \mu = 1)$

$$E'_N(N,3) = (3, \frac{\lambda 1 - \mu 2 + \mu^2 2 \cdot 3}{\lambda^2 1}, \frac{2(1-\mu 3)}{\lambda 2}, \frac{2(1-\mu 3)}{\lambda 1}) \mod 7 = (3,5,5,3).$$

Which would have been different from how equation (7) is calculated:

$$E''_N(N) = (\frac{2}{\lambda 1 + \mu 2}, \frac{1}{\lambda 1 + \mu 2}, \frac{1}{\lambda 1 + \mu 2}, \frac{2}{\lambda 1 + \mu 2}) \mod 7 = (\frac{2}{3}, \frac{1}{3}, \frac{1}{3}, \frac{2}{3})$$

Although both results are different from each other, we can conclude to use the non-invertible vector E'_N as it follows the protocol discussed in the research articles [1], [2], and in section 3.2 of this paper.

4.4 STEP 4: Calculate and store L'_N, R'_N, and E''_N

Once the unique generator is found, as shown in sections 3.2, 4.2, and 4.3, L'_N and R'_N can be calculated. These vectors will represent the local left- and right-sided units needed for the HDLP logic to work along with the global unit E''_N. We reference Algorithm (3), steps 5 and 6, to calculate the local and global rotation vectors. Doing so will give us the following results:

$$L'_N(N,3) = (3,5,1.5,3); \quad R'_N(N,3) = (3,5,5,3); \quad E''_N = E'_N(N,3) = (3,5,5,3);$$

Looking at the results above, an interesting thing to note is that the local right-sided vector is the same numeric vector as the global-sided vector. We think this is because the global vector needs to represent correctness for local groups to ensure that the HDLP is satisfied. Essentially, the global represents a global solution that can be seen universally, and due to the HDLP, it can be either the left— or right-sided unit. Since the local group ordering is unknown globally, this adds security based on the HDLP definition.

4.5 STEP 5: Calculate Public Key vectors (Y, Z, T)

Once the local and global units are successfully generated, the public key can be generated as a tuple of vectors. As defined in section 3.3, a random integer $x < q$ is generated to use as the secret key to generate the set of tuples as defined in Algorithm (3). The tuple results can be seen below:

$$x = 2; \quad \textbf{x is randomly generated}$$

$$Y = A \circ N^x \circ L'_N \circ A^{-1} \implies (3, 1.867, 1.333, 2.067)$$

$$Z = B \circ R'_N \circ N \circ B^{-1} \implies (4, 0.296, 6.963, 4.852)$$

$$T = B \circ E''_N \circ A^{-1} \implies (4.867, 6.667, 5.4, 1.333)$$

4.6 STEP 6: Calculate Private Key vectors (J, U)

Once the public key is computed, a tuple will be calculated to create the private key described in section 3.3 and Algorithm (3). The results of the generated tuple are shown below:

$$J = B \circ R^{'}{}_{N} \implies (5, 6, 3, 2)$$
$$U = L^{'}{}_{N} \circ A^{-1} \implies (0.983, 0.067, 0.217, 6.3)$$

4.7 Signature Generation Results

Once the public and private keys are set, the signature can be generated given a message **M**. As described in section 3.4, a random integer $k < q$ is generated to compute a variable v by computing a hash. The hash algorithm used is SHA-256 since it is a widely known and reliable hashing algorithm. The resultant are two signature elements in the form of integers, (v, s). Doing so gives us the following vectors and hashed integers as described on Algorithm (4).

$$k = 3; \quad \textbf{k is randomly generated}$$
$$V = J \circ N^{k} \circ U \implies (1.33, 4.4, 0.333, 3.133)$$
$$v = F_h(M, V)) \implies 736182505895415059997644719 72....$$
$$s = k + x \cdot v \ mod \ q \implies 3$$

4.8 Signature Verification Results

Once the hash is complete, we get an encrypted signature (v, s) used to verify the validity of the given message **M**. After the signature is generated, the receiver of the message needs to verify the validity. This is done by implementing Algorithm (5). From the algorithm, we get another hashed integer v'. This integer is then compared to the hashed integer v to see if the integers are the same. If they are, then the message is considered valid. If not, then the message is considered invalid and discarded. The results for the verification are shown below:

$$V' = Y^{-v} \circ T \circ Z^{s} \implies (0, 0, 0, 0)$$
$$v' = F_h(M, V^{'}) \implies 104355530019801050157751961385123430398387 2812...$$
$$v' \neq v$$

As seen above, given our inputs, v and v' are not the same. This means that we either sent and decrypted the message incorrectly or there was an issue generating the keys given our chosen inputs. Either way, we are unfortunately getting an invalid message verification.

5 Challenges

We faced several challenges while developing this project. The first issue we ran into was understanding what non-commutative algebras are and how they contribute to developing post-quantum cryptography schemes. The problem we faced was that developing a ring within non-commutative algebra was complicated because commutativity and associativity are not guaranteed. As described in section 2, each operation needs to be done in a specific order derived from non-commutative properties. We tried to implement this logic in Singular but quickly figured out that forcing non-commutative rings in Singular is very complex and outside the project's scope into non-commutative algebras; we found the missing piece we were looking for. The papers we reference fail to reference where they derive their non-commutative logic and properties. We discovered they were referencing Quaternion logic and changed the variables and properties to make it seem different. Once we realized this logic, we found that Python is a common language used for non-commutative algebra logic and includes a package named **pyquaternion** that provided everything we needed.

Another challenge we faced was understanding where the security came from in the implemented post-quantum cryptography scheme. As described in sections 1 and 2, the HDLP allows the post-quantum cryptography scheme to work since DLP's are very vulnerable within post-quantum systems. At first, we knew that the HDLP was important, but we didn't see how the HDLP fixed the vulnerabilities that the DLP showed in post-quantum machines. As described in sections 2 and 3.1-3.3, the HDLP allows for generating both local and global units, which prevents a post-quantum machine from solving anything in polynomial time. This is due to the added security of not knowing some of the variables in the DLP equation at any given time. These principles allow the formulas and properties described in section 3 to work.

6 Conclusion

At the beginning of this project, we intended to perform PQC signature generation and verification through software and hardware implementation. However, after discussing with Prof. Kalla, it was clear that we didn't fully understand the project as we had never used FNAAs to construct post-quantum signature schemes before or learned of them during class. Therefore, we had to delve further into the conceptual computations before we could ever touch the hardware implementation of this project. We only implemented the software aspect, which ultimately did not work as the hash values $v' \neq v$ during the verification stage of this project. There could have been many aspects of this result. One reason could be that not all public keys were non-invertible due to the computation of the local and global units. Another reason could be that certain aspects of some algorithms could have been misinterpreted, such as the calculation of V' as it resulted in **zeros** for all elements in the quaternion vectors, therefore v' results in a hash different than v leading to the signature being rejected based on Algorithm 5. However, we learned many new concepts about using a different type of algebra that we've never used before to implement a new variation of the Schnorrs scheme for post-quantum signature verification. It was pretty challenging to understand at first. Still, with practice and additional readings, we can determine that the proposed post-quantum signature scheme would work as most of the conceptual aspects of developing the HDLP basis worked well. However, the hashing comparison could be worked on further, as it would have been nice to prove this variation of Schorr's scheme as the process represents the one shown during one of our lectures (Fig 1).

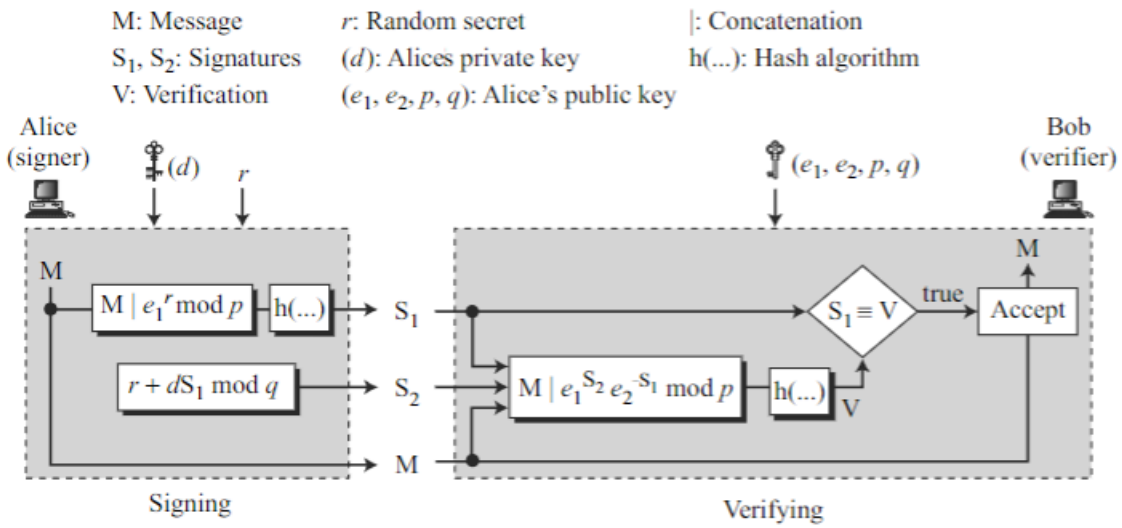

Figure 1: Digital Signature Algorithm using Schnorr's Scheme [9]

7 Contributions

Throughout this project, we worked closely to understand the various aspects of interpreting the use of FNAA in hardware cryptography signature schemes, as it is not easy to know within a couple of weeks. For instance, we all looked through research articles to understand HDLP and the use of FNAAs regarding PQC, tried to implement FNAA logic, wrote examples of quaternion multiplication by hand to ensure non-commutative properties are correct, read about quaternion Python libraries to verify correctness to external sources, developing code, putting together our report, and providing documentation. However, here are some ways we individually focused our work to add value to the team.

7.1 Elmir

Elmir contributed to the project by implementing and trying to understand the algorithms needed in both Python and Singular. These algorithms include public/private key generation, signature generation, and verification. Elmir also assisted in deriving the connections between the referenced papers and Quaternion logic in non-commutative algebras. Elmir also created connections between the post-quantum material described in Professor Kalla's course and the material shown in the referenced paper to compare and contrast different post-quantum schemes. Elmir also assisted with deriving the differences between DLP and HDLP and how that applies to post-quantum cryptography schemes.

7.2 Fernando

Fernando contributed to the project by indulging in the concepts of FNAA complexity and focusing on the conceptual aspects of why the HDLP basis should be considered in PQC. Additionally, Fernando assisted in implementing FNAA and discovered using Quaternion algebra instead of the FNAA examples in the research articles. Fernando constructed the idea that if the algebra chosen for this project meets the criteria of an FNAA, then it should still work and be implemented in various languages for testing. Besides conceptual help, Fernando helped in the software aspects by helping to code in Singular and Python to develop the correct implementation according to the research articles mentioned in this paper.

8 Final Recap

Overall, the final project successfully helps Elmir and Fernando learn a new way of thinking about cryptography schemes regarding non-commutative algebras. The project helped Elmir and Fernando dive into the intricate math that makes up both non-commutative and commutative cryptography schemes. From the deep dive, Elmir and Fernando mastered the math and reasoning behind different types of cryptographic schemes and learned that there are multiple different and contrasting ways of developing cryptography schemes.

References

[1] D. N. Moldovyan, A. A. Moldovyan, and N. A. Moldovyan, "Post-quantum signature schemes for efficient hardware implementation," in *Microprocessors and Microsystems*, 2021.

[2] A. A. Moldovyan and N. A. Moldovyan, "Post-quantum signature algorithms based on the hidden discrete logarithm problem," in *Computer Science Journal of Moldova*, vol. 26, no. 3, 2018, p. 78.

[3] D. N. Moldovyan, "Non-commutative finite groups as primitive of public key cryptosystems," in *Quasigroups and Related Systems*, 2010.

[4] S. Y. Yan, "Quantum Attacks on Public-Key Cryptosystems," in *Springer*, 2014.

[5] Wolfram MathWorld, "Quaternion." [Online]. Available: https://mathworld.wolfram.com/Quaternion.html

[6] D. N. Moldovyan, A. A. Moldovyan, and N. Sklavos, "Post-quantum signature schemes for efficient hardware implementation," in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2019, pp. 1–5.

[7] K. Wyann and et. al, "pyquaternion." [Online]. Available: https://kieranwynn.github.io/pyquaternion/#welcome

[8] D. N. Moldovyan, A. A. Moldovyan, and V. Shcherbacov, "On Defining 4-Dimensional Finite Non-Commutative Associative Algebras Over GF(2^s)," in *Proceedings of the Fifth Cconference of Mathematical Society of Moldova*, 2019.

[9] P. Kalla, "Digital Signature Algorithms." [Online]. Available: https://utah.instructure.com/courses/935869/files/157354728?module_item_id=23521977