# SQI$_{\text{SIGN}}$

Isogeny-Based Digital Signature Scheme

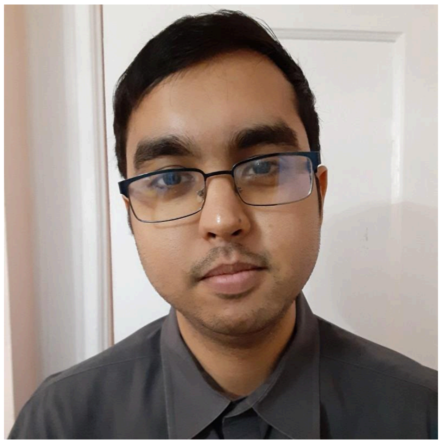
Osasere Imade
Abdul Mutallif
Anjiya Shrestha
Matthew Withee
Suresh Yhap

Final Research Paper
CS381/780 – Post Quantum Cryptography
Dr. Delaram Kahrobaei
Queens College, Fall 2024

https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/sqisign-spec-web.pdf

Matt Withee

Anjiya Shrestha


Suresh Yhap

Abdul Mutallif

Osasere Imade

# Chapter 1 – Introduction

As the promise of quantum computing draws closer and closer to reality, the safeguards in place today allowing society to protect itself in the digital realm are clearly becoming less sufficient. The rapidly emerging field of post-quantum cryptography has been building on the classical concepts and algorithms already in use while finding innovative new ways to defend against the threat of cyberattacks. Classical algorithms such as digital signature schemes are highly effective when used for defending against classical computers, but the computational power from a quantum attack would leave a huge amount of vulnerability, so innovations are required to prepare for the future. One such notable innovation in post-quantum cryptography consists of the development of an isogeny-based digital signature scheme. There are a few types of isogeny-based algorithms, but the one that will be discussed here is something called $\text{SQI}_{\text{SIGN}}$, which stands for Short Quaternion Isogeny-based Signature, a scheme where the digital signature is generated using isogenies between supersingular elliptic curves. This technique is gaining a lot of attention due to its efficiency in securing data while maintaining a compact signature size.

First, a brief review of digital signatures. A digital signature is a cryptography mechanism with three major security functions: authentication, integrity of information, and non-repudiation.

1. Authentication refers to the verification of the identity of the sender of information, achieved through the use of public key cryptography where the sender ‘signs’ a message with their private key and the message is then verified by the receiver using the sender’s public key.
2. The second function of a digital signature, integrity, refers to the process of ensuring that the message has not been altered in any way by an ‘in-between’ party.
3. Non-repudiation, the third security function of a digital signature, is a way to prove that someone sent the message, essentially ensuring that someone that sends a message can’t deny it later because they are the one that had the private key. These functions are the cornerstone of secure digital communication.

A digital signature is composed of two important mechanisms, a private key and a public key, both generated by the sender of the message, with the public key available for anyone that wants to view it.

Traditionally, the private and public keys are generated with algorithms such as Rivest-Shamir-Adleman (RSA) or Elliptic Curve Cryptography (ECC). In RSA, the private key consists of two large prime numbers and the sender encrypts either the message or, more commonly, a hash (summary) of the message. The public key has a mathematical relationship to both these prime numbers. Because factoring very large prime numbers is unrealistic for a modern computer, this algorithm ensures that the encryption remains secure. In ECC, the private key is a randomly chosen number and the public key is created by solving the Elliptic Curve Discrete Logarithm Problem to generate a number. The public key in ECC is linked to the private key. While ECC offers smaller key sizes than RSA (thereby making it a more efficient algorithm), it’s still vulnerable to a quantum attack with something like Shor’s algorithm able to solve the discrete logarithm problem at the center of its security. So, as technical advances in quantum computing become closer to reality, there is a need for algorithms that are quantum resistant.

One promising approach to a quantum resistant cryptography scheme comes in the form of what is called isogeny-based cryptography. An isogeny is a special function (essentially a map) that connects elliptic curves while preserving their group structure. Multiple points from the first curve can map to a single point on the target curve depending on the degree of the isogeny, but an isogeny must be surjective. Isogenies are useful in cryptography because computing them between elliptic curves can be very complicated, referred to as the Isogeny Problem. More specifically, many isogeny-based schemes (such as $\text{SQI}_{\text{SIGN}}$) rely on the difficulty of computing isogenies between *supersingular* elliptic curves, a problem that is very complex for even a quantum computer. And while it is achievable, reversing the process (finding the isogeny when you're only given the curves) is deemed virtually impossible, which is why it is deemed to be quantum-resistant.

There are many types of isogeny-based cryptography schemes, including Supersingular Isogeny Diffie-Hellman (a key exchange protocol that builds upon classical Diffie-Hellman, but incorporating points on the elliptic curves to compute a shared secret) and Supersingular Isogeny Key Encapsulation (a key encapsulation scheme that actually builds on SIDH by using a key encapsulation mechanism). These algorithms, like most isogeny-based algorithms, have small key sizes when compared to other quantum-resistant algorithms, and, at least in theory, they should be secure and efficient. However, both have been broken already by modern attacks that have shown weaknesses in their general structures.

While these algorithms have their own merits and shortcomings, this paper will discuss in detail a robust and promising new algorithm called $\text{SQI}_{\text{SIGN}}$, a digital signature scheme which relies on what's known as supersingular elliptic curves, which are elliptic curves with a special property where their associated endomorphic rings are a quaternion algebra.  Quaternion algebra will be broken down further later, but for now it can be described as a four-dimensional, non-commutative algebra that has more complex quadratic fields, making it more complex to be able to compute and resistant to attack, an important feature for a quantum-resistant algorithm. The private key is generated by randomly choosing one of these isogenies. A common analogy is to imagine standing at the beginning of a maze where you can see the, but you don't know the exact path to take through the maze until you start walking it. The path that you walk to finish the maze is essentially your private key. The public key would be a map that you made that shows you found the exit but doesn't show which path you walked to get there. What's happened is the isogeny has been applied to a target curve to generate the public key.

As expected, there are pros and cons to any digital signature scheme and SQIsign is no exception. A benefit to this scheme is that researchers already know the complexity of the attacks that can be made against it, making the level of security adjustment very straightforward and theoretically keeping the algorithm safe while maintaining efficiency. Another important feature (indeed, what could be considered the defining feature) of $\text{SQI}_{\text{SIGN}}$ is that the keys and signature are quite small when compared to other similar algorithms, making it highly efficient not only in terms of storage but also time it takes to send the message. However, while it is currently accepted that the endomorphism ring problem (the mathematical problem an attacker needs to solve to break the encryption) is extremely complex to solve, consideration must be given that someone could find a shortcut to solving it, making $\text{SQI}_{\text{SIGN}}$ obsolete.

When looked at with future demands in mind, it becomes increasingly obvious that $\text{SQI}_{\text{SIGN}}$, with its smaller key and signature sizes, is one of the most promising and exciting fields in post-quantum cryptography. In this paper, the mathematical properties of $\text{SQI}_{\text{SIGN}}$ will be examined in greater detail, including the generation of the keys, the process of signing and verification, as well as an analysis of the performance and security of the algorithm, while also discussing the potential challenges and shortcomings.

# Chapter 2 - Basic Operations

The secureness of the SQI$_{SIGN}$ Digital Signature scheme involves the difficulty of finding isogenies between supersingular elliptic curves that are defined over finite fields.

## 2.1 - Finite Fields

To define a field, we can build up from the essential element of algebraic structures, the set. A set is an unordered collection of distinct elements. If you add a binary operation on the elements and have the property of closure (applying the binary operation on two elements of the set results in an element of the same set) you end up with a magma. If the binary operation is associative (the order in which you evaluate the binary operations when there is a chain of the operation i.e. the placement of parentheses doesn't matter) then you form a semigroup. If the set has an identity element (applying the binary operation to any element along with the identity yields that same original element), the result is a monoid. Adding the existence of an inverse for every element (applying the binary operation to any element and its inverse yields the identity) forms a group. Finally, if you add commutativity (you can arrange the elements in any order when applying the binary operation to two elements) your result is an abelian group.

A field is a set enhanced with two binary operations (called addition and multiplication) such that the set paired with each of the two binary operations individually each form an abelian group (but for multiplication we exclude the additive identity, known as 0, because it has no multiplicative inverse in the set). Also, distributivity of multiplication across addition should hold ( $a(b + c) = ab + ac$). An example of a field is the rational numbers $\mathbb{Q}$ equipped with the usual notions of addition, subtraction (addition with the inverse), multiplication, and division (multiplication with the inverse). A finite field (aka a Galois field) is a field with finite order (the number of elements or cardinality of the underlying set is finite).

The finite fields considered in this paper are ones of prime order, $F_p$, or the square of a single prime order, $F_{p^2}$ where $p \equiv 3\ (mod\ 4)$. All finite fields are of prime power (a power of a single prime) order. The characteristic of a field is the minimum positive number of times you must add the multiplicative identity element to get the additive identity. An illustrative example is the field ($\mathbb{Z}_p$, +, ·) where the operations are done modulo $p$. The characteristic of this field is $p$ because adding 1 $p$ times yields 0 (because it equals p which is congruent to 0 mod p). The characteristic of a field $F_q$ where $q = p^r$ is $p$.

A quadratic residue is the remainder when a perfect square is reduced modulo $p$ (i.e. it is congruent to a square). To test if an element of the field $F_p$ = ($\mathbb{Z}_p$, +, ·) is a perfect square, that is, there is an element $b$ such that $b^2 \equiv a\ (mod\ p)$, then raising both sides of the congruence by $(p - 1) / 2$ yields $b^{p-1} \equiv a^{(p-1)/2}\ (mod\ p)$. By Fermat's little theorem which states that $c^{p-1} \equiv 1\ mod\ p$ where $c$ and $p$ are relatively prime means that $a^{(p-1)/2} \equiv 1\ mod\ p$. If this is not true, then $a$ wasn't a perfect square to begin with. In the special case where $p \equiv 3\ (mod\ 4)$, the positive square root is given by $\sqrt{a}\ =\ a^{(p+1)\,/\,4}$. A verifying example is if $p$ = 7, then $\sqrt{a}\ = a^{(7+1)/4}\ =\ a^{2}$. Testing if $a\ =\ 2$ is a square and if so what the square root is follows. $2^{(7-1)\,/\,2} \equiv 8 \equiv 1\ (mod\ 7)$ so 2 is a square mod 7. $\sqrt{2} \equiv 2^{(7+1)/4} \equiv 4\ (mod\ 7)$. Indeed, $4^{2} \equiv 16 \equiv 2\ (mod\ 7)$. A natural ordering of $F_p$'s elements is considered (0, then 1, then $\ldots\ p\ -\ 1$ ).

If $F_p$ was analogous to the real integers, $F_{p^2}$ is analogous to the complex integers. The imaginary unit, $i$ , is the root of the equation $i^{2}\ +\ 1\ =\ 0$. The multiplicative inverse of an element, denoted $1/(a\ +\ bi)$ can be written with a "real" denominator by multiplying the numerator and denominator by the complex conjugate, $a\ -\ bi$. A lexicographical ordering of $F_{p^2}$ (based on the real parts and then based on imaginary parts if the real parts are equal) is defined.

## **2.2 - Elliptic Curves**

The class of curve we are interested in for SQI$_{\text{SIGN}}$ are called elliptic curves. In particular, we are interested in so-called Montgomery curves over the field $F_q$ of the form:

$By^{2}\ =\ x^{3}\ +\ Ax^{2}\ +\ x$ where A and B are elements of the field that satisfy $B(A^{2}\ -\ 4) \neq 0$. In addition, we include a "point at infinity", ∞. Two curves are isomorphic (there is a bijective mapping between them) if the mapping is of the form: $(x,\ y) \rightarrow (D(x\ +\ R),\ Cy)$ i.e. there is only shifting and scaling performed. Two elliptic curves are quadratic twists of one another if $C\ =\ \sqrt{B/B'}$ where $B'$ is the leading coefficient on the $y^{2}$ term of the first curve and are isomorphic if $B/B'$ is a perfect square in $F_q$.

If N, the number of solutions to the elliptic curve, is congruent to: $1\ mod\ char(F_q)$, then the elliptic curve is called supersingular. We are concerned with the field $F_{p^2}$ where $p \equiv 3\ (mod\ 4)$ so for instance if $p\ =\ 7$, the curve is supersingular if it has 1, 8, 15, $\ldots$ many solutions. With B = 1, a supersingular curve has precisely $(p\ +\ 1)^{2}$ points. So for $p\ =\ 7$, there are 64 points (verifying this: $64 \equiv 1\ (mod\ 7)$ as expected).

Algorithm 1 (pg. 8) takes in an $A$ value (take B = 1 here for now) and outputs an $A'$ (which corresponds to a curve isomorphic to the curve corresponding to $A$) and the isomorphic mapping itself.

A very interesting fact is that we can define an addition operation that when paired with the set of points on the Montgomery curve form an abelian group! For an elliptic curve, if a line intersects it at two points, then it must intersect it at a third point given that tangent points are counted twice and the aforementioned “point at infinity” is the third point for vertical lines of intersection.

Addition can be defined as follows. The “point at infinity” serves as the additive identity so if $P$ is a point on the curve, $P + \infty = \infty + P = P$. Now to add two general points on the curve, draw a line of intersection through the two points, $P_1$and $P_2$. The third point of intersection, $P_3$, is defined as -($P_1 + P_2$). So the three points, $P_1$, $P_2$, -($P_1 + P_2$) add to “zero” which here is the additive identity, ∞. This summing to identity will always be true. Reflecting this point over the x-axis yields the additive inverse ($P_1 + P_2$), the desired sum (this is because a vertical line through -($P_1 + P_2$) and ($P_1 + P_2$) also intersect the point at infinity, ∞, and these three points should sum to “zero” as they do: -($P_1 + P_2$) + ($P_1 + P_2$) + ∞ = ∞ again because the point at infinity is the additive identity). The addition rule is closed because it always results in a point on the curve. The addition rule is commutative because the order of points of intersection we add doesn’t matter. Associativity turns out to also be true. Because of these properties, the points on the Montgomery curve with the defined addition operation truly do form an abelian group.

Point doubling can be achieved by drawing the tangent line to the curve at the desired point to be doubled, finding the third point of intersection (since the tangent point counts as two points), and reflecting as before. Note that by this definition a vertical tangent yield $P = - P$ (because this occurs on the x-axis) and $P + - P + \infty = \infty$ (by the addition law) so $[2]P = \infty$ (because $[2]P + \infty = [2]P$ since ∞ is the identity point). General scalar multiplication can be achieved with repeated addition, that is, summing copies of the point: $[k]P = P + P + ... + P$ k times, using doubling and addition as defined above. The order of a point is the smallest positive integer $m$ such that $[m]P = \infty$. The geometric definition of addition outlined here can be translated to a coordinate-based algebraic one using the formulas on (pg. 9).

We consider a subgroup of an elliptic curve $E$, defined over $F_{p^2}$ and integer $m$, called the $m$-torsion subgroup denoted $E[m]$ that consists of the points in $E$ that yield $[m]P = \infty$ . This means it consists of all points each with the property that when you add $m$ copies of the point using the line intersection and reflection method previously mentioned, you end up on a vertical line, that is, the sum is ∞. If $m^2$ divides the number of points on the curve, then $E[m]$ is isomorphic to $Z_m \times Z_m$ so it consists of $m^2$ points. There exist non-unique points $R$ and $S$ (that is, possibly multiple pairs exist) in $E[m]$ that generate the group $E[m]$. $(R, S)$ is called a basis for $E[m]$. Algorithm 3 on pg. 10 essentially finds $R$ and calls Algorithm 2 on pg. 10 which finds $S$ and returns the basis $(R, S)$.

Since we are working with repeated applications of an abelian group’s operator, we can naturally define a discrete logarithm problem. That is, given a point $P$ and the point $[k]P$ (with scalar multiplication as defined above), extract the value of $k$, the number of times the addition operator was applied. You can imagine the difficulty of this because as defined above, the sum of two points on an elliptic curve seem to have little relation to what the initial points were (so repeatedly applying this is like bouncing around the elliptic curve). All you have is the initial point ($P$) and the final point ($[k]P$) and you need to find out how many times, $k$, the operator was applied. For points of large prime order (that is, they bounce around a lot before reaching the "steady state" of ∞ through a vertical line), the DLP is believed to be difficult for classical computers because you can basically only do an exhaustive search (try all possible $k$).

However, for points of smooth order (that is, the order is a composite number with small prime factors), the problem is efficiently solvable. For a particular $m$-torsion subgroup where $m$ is a power of 2 or 3 (therefore the points in this subgroup $E[m]$ are of smooth order) where a basis of this group $(R,\ S)$ is known, any point on the elliptic curve in $E[m]$ can be represented as a "linear combination" of the basis points, that is, $P\ =\ [a]R\ +\ [b]S$. So, such points are characterized by two integers, $a$ and $b$, in a way similar to how vectors are described when the basis is implicitly understood. An algorithm based on the Pohlig-Hellman algorithm (which solves the DLP for a finite abelian group with a smooth number of elements) is used to find $a$ and $b$.

This analysis of supersingular elliptic curves defined over finite fields gives the background required to understand isogenies between elliptic curves and eventually the correspondence between quaternion ideals and isogenies.

## 2.3 – Isogenies

Remember, an isogeny is a special type of function that connects two elliptic curves in a way that preserves their algebraic structure. Given two elliptic curves $E_1$ and $E_2$ over a finite field $F_q$, an isogeny φ: $E_1 \rightarrow E_2$ is a non-constant map defined by rational functions (ratios of polynomials) with coefficients in $F_q$. This map satisfies $\varphi(\infty)\ =\infty$, where ∞ represents the point at infinity on the elliptic curve. An isogeny is also a group homomorphism which means it preserves the group operation (point addition) on the elliptic curves and when two curves are connected by an isogeny, they are said to be isogenous. Interestingly, two curves over $F_q$ are isogenous if and only if they have the same number of $F_q$ rational points, denoted f $\#E_1(F_q)$ = $\#E_2(F_q)$.

The kernel of an isogeny φ, denoted as ker(ϕ), includes all points on $E_1$ that get mapped to the point at infinity on $E_2$: ker(φ) = {P ∈ $E_1$| $\varphi(P)\ =\infty$ }. If you know a subgroup G of $E_1$ with size N, there is a unique isogeny φ of degree N with the subgroup G as its kernel, ker(φ) = G. Furthermore, For every isogeny φ: $E_1 \rightarrow E_2$, there exists a dual isogeny φ'$:E_2 \rightarrow E_1$ of the same degree and every isogeny has a dual isogeny φ'$:E_2 \rightarrow E_1$, which reverses the direction of the original map. The composition of the isogeny and its dual gives a scalar multiplication by N on the original curve: φ'◦ φ = [N].

An elliptic curve can be represented using a specific type of equation called a Weierstrass equation and the general form look like: $y^2 + a_1xy + a_3y =\ x^3 + a_2x^2 + a_4x + a_6$ where x and y are the variables that represent the coordinates of points on the curve and the numbers $a_1$,$a_2$,$a_3$,$a_4$,$a_6$ are constants (called coefficients) that come from the field and these coefficients define the specific shape and properties of the curve. One of the methods for computing isogenies is through Vélu's formulas which allow us to calculate the specific map between two elliptic curves, given the kernel of the isogeny. If we are given a point Q on $E_1$ that generate a subgroup of order N, Vélu's formulas provide us with the rational functions that define the isogeny φ and can be written as φ ((x, y)) = (f(x), y · g(x)).

f(x) and g(x) are ratios of polynomials, and their degrees are related to the degree of the isogeny and the degree of the isogeny related to the size of the kernel. Vélu's formulas are only practical to compute for relatively smaller values of N, so when N becomes large, especially if N is a composite number, we decompose the isogeny into smaller isogenies, each with a smaller prime degree. For example, if N has a prime factorization: N = $l_1^{e1}$ · $l_2^{e2}$... where $l_1$,$l_2$,… are primes, we can compute the isogeny as a composition of isogenies of degree $l_1$,$l_2$,… This method is feasible as long as the prime factors of N are sufficiently small.

### 2.3.1 - 2-Isogenies

A 2-isogeny is a specific type of isogeny with degree 2 which means that it maps points between two elliptic curves in such a way that the kernel of the isogeny contains exactly two points (including the identity point). Given an elliptic curve $E_{A,B}$ and a point Q of order 2 on this curve, we can generate a 2-isogeny φ: $E_{A,B} \rightarrow E_{A',B'}$ . The point Q generates the kernel of the isogeny, which is the set of points that are mapped to the point at infinity on the target curve. Consider a point Q of order 2 on an elliptic curve $E_{A,B}$. There are two cases: Case 1, When Q=(0,0), The isogeny φ and the new curves $E_{A',B'}$ are defined using specific formulas involving A and B. The formulas calculate φ (x,y) and update the curve coefficients to get (A′,B′). In Case 2, When Q=($x_Q$,0) with $x_Q \neq 0$, different formulas are used to compute φ (x,y) and (A′,B′), involving $x_Q$. In both cases, the point (0,0) on $E_{A',B'}$ lies in the kernel of the dual isogeny φ, which is needed for chaining isogenies. The dual isogeny is the inverse map of the isogeny, and its kernel consists of points that are mapped to the point at infinity.

### 2.3.2 - 4-isogenies

A 4-isogeny is a specific type of isogeny with degree 4, meaning that the kernel of the isogeny contains exactly four points, including the identity point. 4-isogenies are more complex than 2-isogenies but for efficiency purposes, some implementations may prefer to use 4-isogeny formulas more because they can be equivalent to combining two 2-isogenies formulas in a sequence. On page 12, Case 1: Q=(1,$y_Q$), then 2[Q]=(0,0), in this case, Q has the x-coordinate 1, and its second multiple is (0,0). This formula computes the new x- and y-coordinates of points and defines the image curve's parameters A′ and B′. The transformation is based on the elliptic curve's initial parameters and accounts for the 4-isogeny structure. The formula effectively transforms points on the original curve $E_{A,B}$ into points on the new curve $E_{A',B'}$', which has different parameters for A′ but the same B.On pg 12, Case 2: Q=(−1,$y_Q$),in this case, Q has the x-coordinate of −1, and this is similar to the first case, but the sign changes in the formulas so, we follow a similar process to derive the isogeny. This transformation also adjusts the point coordinates and the curve's parameters but in this case, the biggest difference here is that the formulas take into account the different x-coordinate of Q=(−1,$y_Q$).On pg 12, Case 3: Q=($x_Q$,$y_Q$) with $x_Q$ ≠ ±1. In this third case, the point Q has coordinates ($x_Q$,$y_Q$) and $x_Q$ is neither 1 nor -1. The new curve parameters in this case are (A′,B′)=(2−4xQ2,B)(A', B') = (A′,B′)=(2 −4$x_Q^2 x$,B). This formula is the most general and applies to points that are not at specific locations like 1 or −1 but it still serves the same purpose which is mapping points from one elliptic curve to another while preserving the group structure.

### 2.3.3 - Other odd-degree isogenies using Vélu's formulas

Odd-degree isogenies are those with a kernel of odd size. Let ℓ be an odd prime, and let Q be a point on the elliptic curve $E_{A,B}$ of order ℓ. This point Q generates the kernel of the isogeny φ:$E_{A,B} \rightarrow E_{A',B'}$, where $E_{A',B'}$ is the codomain curve. The main idea here is that the isogeny is determined by its kernel, and knowing the point Q allows us to compute the isogeny more clearly and to compute this isogeny, we define the polynomial $h_S$(x), which encodes the structure of the kernel. The polynomial is defined as the product, $h_S$(x) = s∈S Π (x − $x_{[s]Q}$), where S={1,2,…,(ℓ−1)/2}, and $x_{[s]Q}$ is the x-coordinate of the point $[s]Q$, which is the scalar multiple of Q and this polynomial collects all the important points in the kernel of the isogeny.The parameter A′ defines the new curve given by A′=2·1+d/1−d, where d, a value computed from the original elliptic curve parameter A is computed as d= (A-2/A+2)$^l$ · ($h_S$(1)/$h_S$(-1))$^8$. The parameter B remains unchanged so $, B'=B$ and it allow us to compute the new elliptic curve $E_{A',B'}$ that is isogenous to the original curve $E_{A',B'}$.

### 2.3.4 - Odd-degree isogenies using √élu

The square root Vélu algorithm, also called the √élu algorithm, computes isogenies of elliptic curves in time O~(ℓ) rather than O(ℓ), where ℓ is the degree. The main idea is to reindex the points in the kernel subgroup in a baby-step-giant-step manner, the baby-step giant-step algorithm computes discrete logarithms in a group of order q using O( √q) group operations and this algorithm breaks down the computation into smaller steps, making a faster evaluation of the kernel polynomial $h_S$(x). Instead of S={1,2,…,(ℓ−1)/2}, we use S={1,3,5,…,ℓ−2} redefining the index Sets I,J,K then continue by defining an index system (I,J) for S, where, I = {2m(2i+1) | 0 ≤ i ≤ m′} and J = {2j+1 | 0 ≤ j < m}. m is defined as m = √(ℓ−1/2) and m′ = 0 if m = 0 , if not then m′ = (ℓ+1)/4m and K = S\(I±J) , where I ± J = {i+j, i-j | i ∈ I, $i \in J$ }. The sizes of these sets are all in O(√ℓ), which makes the algorithm better.

Biquadratic polynomials are polynomials of degree 4 that can be written in the general form P(x)=a$x^4$+b$x^2$+c . The term a$x^4$ represents the highest degree (degree 4), and there are no $x^3$or $x^1$ terms, which differs the biquadratic polynomials from general quartic polynomials. To evaluating $h_S$(α) in Steps First, defining Biquadratic Polynomials $F_0$,$F_1$,$F_2$: $F_0$ ($x_1$,$x_2$)=($x_1$−$x_2$)$^2$ , $F_1$ ($x_1$,$x_2$) = 2(($x_1x_2$+1)($x_1$+$x_2$) + 2a$x_1x_2$), $F_2$ ($x_1$,$x_2$) = ($x_1x_2$-1). On pg 13, for polynomial $h_I$(x), this polynomial contains roots at the x-coordinates of multiples of Q in set I. For $D_j$(x), squares the differences, focusing roots at x = $x_{[j]Q}$ . For the Resultant Δ I,J , it measures the common factors between $h_I$(x) and $D_j$(x) and results in an element of $F_q$. For $E_j$(x), this incorporates the input $\alpha$ and the polynomials $F_0$,$F_1$,$F_2$. For Resultant R, ΔI,J is an element of $F_Q$ and contains information about $\alpha$. For $h_K$ and output $h_S$($\alpha$), this gives the value of $h_S$($\alpha$) needed for the isogeny computation.

When using efficient polynomial arithmetic, computing polynomials using product trees for efficient polynomial multiplication and remainder trees for multi-point evaluation speeds up polynomial and resultant computations. Product trees are used to multiply a series of polynomials efficiently and the polynomials are grouped in pairs and multiplied in parallel, reducing the number of steps. Remainder trees are used for multi-point evaluation which is a method that allows us to evaluate a polynomial at multiple points at once by breaking the evaluation into smaller steps which speeds up the calculation.

In $\text{SQI}_{\text{SIGN}}$, isogenies and methods like Vélu's and √élu's algorithms help create secure and practical post-quantum signatures. They rely on the difficulty of solving isogeny problems on supersingular elliptic curves, providing strong protection against quantum attacks.

## 2.4 – Quaternions and Ideals

A quaternion is a mathematical object, essentially a set of four numbers, used to represent rotations in three-dimensional space. Additionally, quaternion defines the quotient of two vectors in a three-dimensional space. A quaternion algebra over a field $K$ is a central simple algebra of dimension 4 over $K$. It is defined by two parameters $a,\ b \in K$ and it is thus denoted as $B\ =\ (\frac{a,b}{K})\ =\ K\ +Ki\ +\ Kj\ +Kk$ where the multiplication rules are:

$i^2\ =\ a,\ j^2\ =\ b,\ ij\ =\ k,\ ji\ =\ -\ k$. Quaternion algebras are generalizations of Hamilton's quaternions. There are four-dimensional spaces generated by four elements $\{1,\ i,\ j,\ k\}$ and non-commutative algebras with the multiplication rules defined previously.

### 2.4.3.1 - Finding a Short Basis

The short basis algorithm involves finding a short basis for a lattice with respect to $N_q(u)$ = $u_0^2$ + $qu_1^2$. Simply put, "the goal of lattice basis reduction is to find a basis with short, nearly orthogonal (perpendicular) vectors when given an integer lattice basis as input" (Wikipedia). The input for the algorithm includes two vectors $b_0$ and $b_1$, which form an integral lattice basis, along with a positive integer q. The output should be two vectors, $\beta_0$ and $\beta_1$, that form a short basis for the same lattice. The input must also meet these requirements: $N_q$ ($\beta_1$) ≤ $N_q$ ($\beta_0$). Now that we know what to input, let's talk about the process. The process starts by initializing and swapping the basis vectors to maintain the condition $N_q$ ($\beta_1$) ≤ $N_q$ ($\beta_0$). The algorithm then enters a loop, where it computes a value r that is rounded by dividing $N_q$($\beta_1$). The goal of the loop is to reduce the length of the vectors. The loop continues It updates the gamma vector by subtracting r$\beta_1$ from $\beta_0$, and if the norm of gamma is smaller than $\beta_1$, the vectors are updated. Finally, the loop terminates, and the short basis vectors $\beta_0$ and $\beta_1$ are returned.

**2.4.3.2 - Finding the Closest Vector**

After computing a reduced basis, the next step is to find a vector in the lattice that is closest to the target vector t. This is accomplished using Babai's nearest plane algorithm. The nearest plane algorithm was developed by L. Babai in 1986, and it "obtains a 2(2/√3) ^n approximation ratio, where n is the rank of the lattice" (Regev). In other words, the algorithm projects the target vector onto the reduced basis, and then adjusts the result to produce a vector c that is close to t with respect to the norm $N_q$.

**2.4.3.3 - Enumerating Close Vectors**

The final step is to enumerate short vectors in the lattice that are close to the target vector t, using the Fincke-Pohst algorithm. In other words, "the running time of enumeration algorithms greatly depends on the quality of the input lattice basis. So, suitably preprocessing the input lattice using a basis reduction algorithm is an essential part of lattice enumeration methods" (Lattice Cryptography). Furthermore, the algorithm takes as input a lattice L, a target vector t, and an initial close vector c. It then iterates through possible values for the coordinates x and y, generating vectors that are within bound B from the target vector. By updating both x and y within nested loops, the algorithm yields vectors that satisfy the distance condition. This process continues until either the specified number of tries is reached, or all close vectors are found.

**2.4.4.1 - Basic Quaternion Arithmetic**

Quaternions are generalizations of complex numbers, which are represented by the expression: α = a + bi +cj + dk / r. The basic arithmetic operations like addition and multiplication are computed by reducing common denominators where necessary. Specifically, multiplication follows the axioms: i^2 = -1, j^2 = -p, ij = -ji = k. We also have the conjugate operation where α' = a- bi - cj -dk / r. There's also the reduced trace that simplifies to: tr(α) = 2a / r. Finally, we have the reduced norm: nrd(α) = a^2 + b^2 + p (c^2 + d^2) / r^2.

**2.4.4.2 - Lattices**

A lattice is a collection of quaternions that form a grid-like structure. Additionally, lattices are defined by a basis of quaternions that are represented as columns of a matrix. You can also perform operations on lattices such as:

**Equality**: Whether two transactions are similar by comparing their HNF.

**Union and Intersection**: The union of two lattices is formed by merging their initial matrices into HNF. The segmentation of the dual network is obtained by computing the HNF.

**Multiplication**: In lattice multiplication, the base matrices are multiplied by the corresponding quaternary algebra generators.

**Containment**: This function ensures that an element is contained in the network by solving a system of linear equations.

**Index**: The index of one network in another is the mean value of their basis matrices.

**Right Transporter**: The right transporter of a lattice is a set of objects that, when connected by a grid, still belong to another grid. It is one of the most complex operations on networks. (SQI Sign)

### 2.4.5 - Quaternion orders and ideals

Quaternaries play an important role in algebra, and their structure includes some small orders and ideals. An order is a special type of lattice that forms a subring. Elements of an order O are integral because their norm and trace are integers. An order is maximal if it isn't contained in any larger order.

An ideal is a sublattice of an order. The left order of an ideal I is the set of elements that, when multiplied with I, still belong to I. Similarly, the right order is defined in the same way for the right side. A connecting ideal links two orders $O_L$ and $O_R$.

The norm of an ideal is the greatest common divisor (gcd) of the norms of its elements. This norm is an integer and can be computed from both the left and right orders. Any ideal can be expressed using the elements of its left order, and when two ideals are multiplied together, the result is another ideal. Ideals are multiplicative, and their norms follow this rule too. Two orders $O_1$ and $O_2$ are equivalent if there's a quaternion element that links them. In addition, two left ideals I and J are equivalent if they can be scaled into each other by multiplication. When this happens, the left and right orders of both ideals are also equal.

#### 2.4.5.1 - Basic Operations on Ideals

Now that we know about the importance of orders and ideals, it is also important to note that various operations can be performed on them. Simply put, ideals are used in lattices to perform various operations like equality, union, intersection, and multiplication.

**Left and Right Orders**: The left and right orders of an ideal determine how it relates to itself. These are the sets of elements that, when multiplied with the ideal, still belong to the ideal.

**Isomorphism of Ideals**: Two left ideals are isomorphic if there is a quaternion β that scales one ideal into the other, like I = Jβ. This can be calculated using the transporter of the ideals and checking their norms.

**Connecting Ideals**: To connect two orders $O_L$ and $O_R$, we compute a connecting ideal using the formula: I = ($O_L$ + $O_R$)N, where N is the norm of their intersection.

**Pullback and Pushforward Ideals**: If two ideals have coprime norms, the pullback combines them into a new ideal, while the pushforward reduces them using intersection and inversion.

#### 2.4.5.2 - Finding Equivalent Ideals of Small Norm

Sometimes, we need to find an equivalent ideal with a smaller norm, particularly when working with large ideals. When working with $\mathrm{SQI}_{\mathrm{SIGN}}$, we often need to find an equivalent ideal J that has the same structure as a given ideal I but with a different, smaller norm. This is achieved using the surjection: χI (α) = Iα' / nrd(I), where α is a quaternion, and the set of ideals J is equivalent to I. To do this, we use an algorithm, which constructs a new ideal J = χ(β) where β is formed from the reduced basis of I. The process repeats up to a set number of times to check if the norm of J is prime.

## **2.5 - Solving Norm Equations**

A norm is a function from a real or complex vector space to the non-negative real numbers that behave in a certain way like the distance from the origin. Various kinds of norms include the equivalent norm, the absolute value norm, and the Euclidean norm.

The basis for this is that solving norm equations in quaternion orders has applications in cryptography. There are three KLPT-based procedures for solving norm equations which are ***KeyGenKLPT*** which is used within a key generation, ***SpecialEichlerNorm*** which solves a norm equation $O$ and ***SigningKLPT*** which solves a norm equation in the left $O$-ideal which $O$ represents quaternion order. The left $O$-ideal is a non-zero subset $I \subseteq B\ such\ that\ OI \subseteq I\ where\ O\ is\ the\ maximal\ order.$

**KeyGenKLPT Algorithm**: Given a left $O$-ideal $I$ in a quaternion algebra $Bp,\ \infty$, the KLPT algorithm finds an equivalent ideal of a smaller norm

Input: A left ideal $I \subseteq O$ where $O$ is a maximal order in the quaternion algebra $Bp,\ \infty$.

Process: Find an equivalent ideal $J$ that $J$ has a smaller norm than $I$. The algorithm uses the reduction theory of quaternions to efficiently find $J$ by solving norm equations. Ensure that $J$ corresponds to an isogeny with a reasonable degree that makes further cryptographic computations feasible. Output: An equivalent ideal $J$ of a smaller norm.

The KLPT algorithm helps in minimizing computational overhead in constructing and verifying isogenies, making isogeny-based cryptographic schemes more practical.

**SpecialEichlerNorm Algorithm** explains the process of solving norm equations efficiently in any maximal order $O$ the idea behind the algorithm is to find $O1 \cong O\ such\ that\ I(O0,\ O1)$ satisfies the constraints on the input of EichlerNorm so we can solve in $(O0 \cap O_1)$ and then transport the output $O$ using the isomorphism between $O$ and $O_1$.

**The SigningKLPT algorithm** is done by the generalized KLPT algorithm. The output of a SigningKLPT is of a constant degree. Some randomization is also included to ensure a good distribution.

The process of solving norm equations in quaternion orders and ideals can be seen in various algorithms which include Cornacchia's algorithm, representing integers by special extremal order, and reduction to linear systems.

### 2.5.1.1 – Cornacchia's Algorithm

Cornacchia's algorithm allows us to efficiently solve norm equations of the form $x^2 + ny^2 = m$. The algorithm inputs $m$ which is an integer and gives a true or false value if a solution was found or not. The algorithm goes through a sequence of numbers to make sure $m$ is a prime if it is a prime, Cornacchia's algorithm would run and return the value of x and y when found.

### 2.5.1.2 - Representing integers by special extremal order

This is a follow-up from Cornacchia's algorithm. The idea of this algorithm is to find elements in a given norm. This process allows us to solve norm equations in the sub-order $R + jR \subseteq \vartheta_0$ where $R = Z[i]$ the quadratic substring of $\vartheta_0$ whose norm for is given by $F(t, x, y, z) = x^2 + y^2 + p(z^2 + w^2)$. The general form is to sample $z, w$ before using Cornacchia's to see if we can find $x, y$ the following: $x^2 + y^2 = T - p(z^2 + w^2)$. The algorithm returns a boolean if a solution was found.

### 2.5.1.3 - Reduction to linear systems

This algorithm in this case can be found in EichlerModConstraint and FindLinearCombination. Both algorithms use linear algebra in $Z / NZ$ to find a quaternion of a special form. In EichlerModConstraint, we consider quaternion elements $\alpha = x + yi + zj + wk$ where $[x, y, z, w] \in Z^4$ as integer column vectors while in FindLinearCombination, we compute a basis $<a_1, a_2, a_3, a_4> = O$ and compute vectors $[x, y, z, w] \in Z^4$.

The final step of solving the norm equation can be seen in the **Strong Approximation algorithm**. This algorithm solves norm equations inside $O0$. The algorithm finds the strong approximation mod $N\ in$ N of some $\mu_0 \in Oj_O$. This algorithm takes in an input of a prime number $N$, two values, $C, D \in Z$, the output is of the equation $\mu = \lambda\mu 0 + N\mu 1$ with $\mu_{0 = j(C + \omega d)\ where}$ $\mu_{1 \in O0}$ $such\ that\ n(\mu) \in N$ which then returns a value of $\mu = \lambda j(C + D\omega) + N(t + \omega x + j(y + \omega z))$.

The essence of solving norm equations and how these algorithms and equations are essential in cryptography is to provide for integrity and non-repudiation of cryptographic systems like in $SQI_{SIGN}$. Integrity provides us with the ability to make sure a message sent is a message received where nothing in the message changes. Non-repudiation provides us with the ability to verify whoever sends a digitally signed message cannot be in denial of the message coming from the person who sent it. Solving norm equations provides us with the ability to ensure that messages can be digitally secure and encoded which enhances the security of digital communications.

### 2.6.1 The Correspondence between Ideals and Isogenies

The correspondence between ideals and isogenies provides a bridge between algebra and geometry, specifically within elliptic curves.  Elliptic curves are described as “smooth, projective, algebraic curves of genus one, on which there is a specified point O" (Elliptic). These curves aren't just abstract mathematical objects, they have a group structure, meaning you can add points on the curve in a well-defined way, with O acting as the identity element. This structure makes them incredibly versatile, especially in applications like cryptography.

At the heart of this connection is the concept of an endomorphism. An endomorphism is a map of the elliptic curve. It takes points on the curve and maps them back onto the curve, all while preserving the group structure. The set of all such endomorphisms forms a mathematical structure called the endomorphism ring. This ring captures the symmetries and internal structure of the elliptic curve, and in the case of supersingular elliptic curves, it has a deep connection to quaternion algebras, which are special number systems that extend the idea of complex numbers.

Furthermore, there is a one-to-one correspondence between certain algebraic objects, called left ideals in the endomorphism ring, and geometric subgroups of the elliptic curve, which correspond to isogenies. Essentially, every left ideal maps to the kernel of an isogeny, and every isogeny maps back to an ideal. For instance, if you consider the supersingular elliptic curve y^2 = x^3 + x, its endomorphism ring naturally connects to these ideals, creating a seamless relationship between algebra and geometry.

### 2.6.2 Converting $O_0$ -Ideals to Isogenies

Kernels play a crucial role in the conversion of ideals and isogenies. Kernels are used in isogenies as they portray the set of points that show a change. If you know the kernel, you can construct the isogeny. We start with an ideal, and as the ideal acts on the elliptic curve, let’s say E1 is guided to E2, the kernel is generated as a subgroup of points of E1 that have vanished by the transformation, which are then mapped to the identity point 0 on E2 to preserve the group structure**.** This kernel then fully defines the isogeny.

So, looking at the Ideal to Isogeny algorithm, the first input is an ideal I, P prime D and Q prime D which are the transformed points of PD and QD after applying an isogeny. To simplify these steps in the algorithm, think of it as: first, compute the action of the ideal I, then, solve for the kernel generator, which is the starting point that generates the kernel using scalar multiplication; and finally, construct the isogeny using this kernel (SQI Sign). The kernel plays a big role because it is the central component that connects the ideal to the isogeny. In essence, this process provides a computationally practical method for bridging the gap between ideals and isogenies, which is crucial for protocols in post-quantum cryptography.

Additionally, constructing matrices from the ideal and determining its kernel, which gives the generator of the associated isogeny, is an important concept called the “push-through isogeny”. This takes a basis of E[D] and maps it to the image under a separable isogeny. The construction of this isogeny is efficient, as it simply reuses the coefficients calculated during the translation of the ideal. The Ideal to Isogeny algorithm uses this and simplifies this step further by directly generating the separable isogeny from its kernel.

### 2.6.3 Converting $O_0$ -Ideals of large

The process of converting $O_0$-ideals with large prime power norms to isogenies plays a critical role in the key generation and signing procedures of cryptographic systems like SQI Sign because it establishes the mathematical foundation for secure and efficient operations. This process is handled by the Ideal to Isogeny Eichler algorithm, which transforms an ideal into its corresponding isogeny. The procedure is divided into two main phases: one focused on quaternion operations and the other on elliptic curve operations.

In the first phase, quaternion calculations are used to reduce a sequence of ideals while calculating their norms and key isogeny parameters. This process leverages the algebraic properties of quaternions, which are described as "a four-dimensional associative normed division algebra over the real numbers" (Quaternion). Using the Ideal Step algorithm, the ideals are refined step by step, ensuring all necessary values are ready for the next stage. In the second phase, these simplified ideals are transformed into isogenies by using torsion bases and building separable isogenies. This phase relies on the groundwork laid by the quaternion calculations to ensure the process is both accurate and efficient.

The process also incorporates techniques like compression and endomorphism evaluation. Compression and Endomorphism evaluation plays a crucial role in ensuring efficient and practical use. First, let's talk about compression. Compression is all about reducing the size of the isogeny output. Isogenies often require a lot of data to represent. By compressing this information, we save memory and bandwidth, making it more efficient. For endomorphism evaluation, this step focuses on efficiently computing the action of an isogeny. These optimizations are critical for making the algorithm practical in post-quantum cryptography.

Ultimately, this process bridges the gap between quaternion-based ideal arithmetic and elliptic curve-based isogeny construction. It provides a scalable method for handling large prime power norms, which is essential for secure and efficient cryptographic applications.

### 2.6.4 Converting Isogenies to Ideals

Converting isogenies into their corresponding $O_0$ -ideals is a simpler process compared to the reverse. The goal is to find the ideal associated with an isogeny generated by a point P of order D. This involves a technique called decomposition, which plays a crucial role in translating an isogeny back into its ideal efficiently. Decomposition is particularly valuable in SQISign because it avoids computationally expensive operations, making the process both secure and practical. The reason for translating an isogeny into an ideal is that ideals offer a more algebraic and structured representation, making them easier to store, manipulate, and verify in cryptographic systems.

# Chapter 3 - Signature

This chapter covers the heart of the $\mathrm{SQI}_{\mathrm{SIGN}}$ Digital Signature scheme, that is, the key generation, signing, and verification steps. It builds upon the foundation laid out in our discussion of elliptic curves and quaternions in Chapter 2 - Basic Operations.

## Section 3.1 - Σ-protocols and the Fiat-Shamir Heuristic

The essence of $\mathrm{SQI}_{\mathrm{SIGN}}$'s practical operation (its functioning as a scheme) is that of an interactive proof of knowledge called a Σ-protocol. A proof of knowledge consists of a prover and a verifier. The prover wants the verifier to believe that the prover has certain knowledge of something. The easiest way for the prover to prove this would be for them to simply tell the verifier the information itself, thus proving that the prover knows the information. This however is not practical because the knowledge itself usually needs to be a well kept secret. In so-called, “zero-knowledge” proofs, the verifier should be able to, given a very small chance of error, correctly ascertain if the prover does or does not have the knowledge without the prover exposing the knowledge itself. In fact, no knowledge (other than whether or not the prover knows something) should be able to be ascertained by a dishonest verifier. A legitimate prover should be believed and an adversarial prover should not be believed (with a very small probability of error).

A Σ-protocol is a type of interactive proof of knowledge where the prover and the verifier communicate through a predefined and mutually accepted and understood protocol resulting in a decision made by the verifier. An NP-language is a language, L, (perhaps a set of solutions to a problem) in which it is possible to deterministically verify in polynomial time if an element, x, (a potential solution to the problem) belongs to L given that you provide a witness, w (a way to prove that $x \in L$). A prover that has knowledge of the (x, w) should be able to convince the verifier that it knows w. The Σ-protocol over a set of (x, w) along with a security parameter λ is a collection of probabilistic polynomial time algorithms, (P1 , P2 , V). A PPT algorithm as opposed to a deterministic algorithm uses randomness and has a small chance to be incorrect. P1 and P2 are assumed to be able to share a state without direct communication between them. V is deterministic. The process of a Σ-protocol consists of a three-way exchange of information:

1) Commitment - the prover runs $P1(x, w) \rightarrow com$. Essentially, the prover “commits” to a value that it can’t change later, while also hiding the knowledge, w, from the verifier.of
2) Challenge - the verifier tests (challenges) the prover by sending a bit string of length λ, chall, randomly chosen from a uniform distribution. Since the prover doesn’t know this ahead of time and must instead react to what the verifier provides, the prover cannot cheat by sending a precomputed response.
3) Response - the prover runs $P2(chall) \rightarrow resp$ and sends this response to the verifier.

Finally the verifier checks the trustworthiness of the prover by computing V(x, com, chall, resp) and outputting honest or dishonest.

We are interested in a non-interactive proof of knowledge unlike the interactive sigma protocol outlined so far. We would like to have a single signing stage by one party and a verifying stage by another party with nothing communicated between them other than the signed data (as opposed to having a commitment, challenge, response, and verification stages).

Essentially, we want to generate a Digital Signature Scheme from an interactive proof of knowledge. This is precisely what the Fiat-Shamir transform does.

First, (x, w) is generated given the security parameter λ. The signer alone has this pair as the private signing key. The public verification key, x, will be made available to anyone that seeks to verify the signature. Secondly, the signer will compute a commitment $P_2$(x, w) → com as it did before. However, it does not send this commitment to the verifier. Instead, it computes the challenge itself using a cryptographically secure, one-way hash function that acts as a random oracle (cannot be predicted but is still deterministic in that the same input produces the same output). The challenge is thus the output of the hash function when passed the commitment and the message, that is, H(com // msg) → chall. The signer then generates a response from this challenge, $P_2$(chall) = resp. After all of this, independent verifiers can cross check this information to ascertain the validity of the signature on the data; it computes V(x, com, chall, resp) and outputs whether or not the signature is valid (knowledge has been shown or not).

One thing to note is that this transformation completely relies on the assumption that the hash function is not predictable. If an antagonistic signer knew which challenge would be output from the hash function beforehand, it could fake a response without even computing the hash and so could falsely sign messages. Unsuspecting verifiers would then mark the signature as authenticated. But since the hash function gives very different outputs even if you slightly change either the commitment or the message, the signer cannot know what the challenge will be until it actually computes the hash function and thus the response will be legitimate. Thus the transformation is correct and complete.

## <u>**Section 3.2 - Precomputation**</u>

We need some precomputed data before $SQI_{SIGN}$ can run its algorithm. $E_0$ is the special elliptic curve: $y^2 = x^3 + x$. When $p \equiv 3 \ (mod\ 4)$, over $F_{p^2}$, $E_0$ is supersingular. We need a torsion value, T, that among other things is smooth. Given a basis for points in $E_0[T]$ (recall that this is the T-torsion subgroup of $E_0$, the points on the curve satisfying [T]P = ∞), we can “easily” find T if it is smooth via an algorithm similar to Pohlig-Hellman which efficiently solves the discrete logarithm problem in this case. We call the basis $B_{0,T}$. We also need the degree of the commitment and challenge isogenies (the size of their kernels) and a few other values.

## <u>**Section 3.3 - Key generation**</u>

The key generation process consists of taking in a base-1 number of length λ and outputs a public verifying key, pk, a random elliptic curve $E_A$, and a private signing key, sk, data required to compute End($E_A$).

The public key contains a semi-random curve $E_A$.

The signing key contains:

1) A quaternion, α, that connects two ideals $I_{secret}$ , Secret related to the signing key isogenies

2) A basis $B_{A,T}$ which is the image of $B_{0,T}$ (a basis of $E_0[T]$) after applying the degree power-of-2 isogeny $\varphi_{secret}: E_0 \rightarrow E_A$
3) A point $Q \in E_A[2^f]$ that generates the kernel of the dual of the last $2^f$-degree isogeny composed in $\varphi_{secret}$.

Algorithm 25 - SQIsign.KeyGen($1^\lambda$) outlines the key generation process. First, a random secret prime $D_{secret} \equiv 3 \ (mod\ 4)$ of $size \approx p^{1/4}$ is sampled. Using the FullRepresentInteger algorithm, an element of a maximal order $\gamma \in O_0$ is found. $O_0$ is isomorphic to the endomorphism ring (defined later) of $E_0$. Then an ideal of norm $D_{secret}$ is found using the equation $I_{secret} = O_0\ (\gamma(a\ +\ i)\ +\ O_0(D_{secret})$ where $a$ is a random positive scalar less than $D_{secret}$ and i is the standard imaginary quaternion element. α that connects $I_{secret}$ to an equivalent ideal $J_{secret}$ ($J_{secret}\ =\ I_{secret}\alpha$ ) is found using the KeyGenKLPT algorithm. $J_{secret}$ is found using the formula: $J_{secret}\ =\ \chi I_{secret}(\alpha)\ =\ I_{secret}\ *\ (\alpha/nrd(I_{secret}))$ where $nrd(I_{secret})$ is the norm of $I_{secret}$ (the gcd of the norms of its elements). IdealToIsogenyEichler attempts to find the isogeny $\varphi_{secret}$ corresponding to the ideal $J_{secret}$. The process mentioned thus far is repeated until a $\varphi_{secret}$ is found.

After $\varphi_{secret}$ is found, α is conjugated. The algorithm Normalized takes in $\varphi_{secret}$ and outputs $E_A$ (the public key) and changes $\varphi_{secret}$ so that it maps to the normalized elliptic curve $E_A$ (i.e. $\varphi_{secret}: E_0 \rightarrow E_A$). The basis $B_{A,T}$ mentioned before is computed using the newly found $\varphi_{secret}$. A point P that generates the points in the intersection of: a) the kernel of the secret isogeny and b) the points on $E_0$ of order $2^f$, is chosen. P is one of the points forming the basis of the $2^f$-Torsion subgroup (the points on $E_0$ such that $[2^f]P_0 = \infty$ for $P_0 \in E_0$). The other basis point Q is found using the CompleteBasis algorithm. The secret isogeny $\varphi_{secret}: E_0 \rightarrow E_A$ is then applied to this Q to get the final Q. The public key, $E_A$ is returned. The secret key, the tuple $(\alpha,\ B_{A,T},\ Q)$, is also returned.

## **Section 3.4 - Signing**

The Σ-protocol (that will be transformed into a Digital Signature Scheme by the Fiat-Shamir Transform) used in $\text{SQI}_{\text{SIGN}}$ in particular passes around elliptic curves and endomorphism rings of elliptic curves. As a review, given an elliptic curve E, an endomorphism is an isogeny φ : E → E (the domain equals the codomain). Recall that an isogeny is an onto mapping (the entire codomain is mapped to by elements of the domain) that has a finite kernel (the set of elements that map to the identity element; the familiar notion of the null space is the kernel of a matrix for instance). Since here we consider isogenies where the domain and codomain are both E, the mapping is also one-to-one (no two elements of the domain map to the same element in the codomain) and thus is a bijection (one-to-one correspondence). An endomorphism ring, End(E), is the set of all endomorphisms of E (essentially a set of functions) equipped with an addition and multiplication binary operators. Here addition of functions is defined: (f+g)(P) = f(P) + g(P) (P is a point on the curve). Multiplication of functions is defined as function composition: (f o g)(P) = f(g(P)); this is not necessarily commutative hence we consider rings here instead of fields. The endomorphism ring problem is to find End(E) given E. It is considered to be difficult to compute from E.

Naturally then, the Digital Signature uses $E_A$ as a public key and End($E_A$) as the private key (the knowledge). The prover tries to convince the verifier that they know the endomorphism ring of $E_A$. This is the $SQI_{SIGN}$ protocol on a high level:

1) Commitment - the prover randomly generates ($E_1$, End($E_1$)) and sends $E_1$ to the verifier.
2) Challenge - the verifier randomly generates an isogeny $\varphi_{chall} : E_1 \rightarrow E_2$ and sends $\varphi_{chall}$ to the prover.
3) Response - since the prover knows End($E_1$) and now knows $\varphi_{chall} : E_1 \rightarrow E_2$, they can compute End($E_2$). The prover can then use the knowledge of End($E_A$) and newly computed End($E_2$) to compute $\varphi_{resp} : E_A \rightarrow E_2$ and send it to the verifier.

The verifier, who knows $E_A$, since it's the public key, and $E_2$, since it generated an isogeny from $E_1$ as the challenge and was sent $E_1$ as the commitment, can check if $\varphi_{resp}$ is indeed an isogeny from $E_A$ to $E_2$. This (almost) works since the prover will likely have to use its knowledge of End($E_A$) to compute $\varphi_{resp} : E_A \rightarrow E_2$. The only problem is that if a dishonest prover generates $E_1$ initially by choosing a random isogeny from the public key to $E_1$ ($\varphi_{cheat\ comm} : E_A \rightarrow E_1$), when given $\varphi_{chall} : E_1 \rightarrow E_2$ in the challenge step, they can then simply compose the isogenies ($E_A \rightarrow E_1 \rightarrow E_2$) and yield $\varphi_{cheat\ resp} : E_A \rightarrow E_2$. To fix this, the verifier can choose a challenge $\varphi_{chall} : E_1 \rightarrow E_2$ where such a composition of isogenies to get from $E_A$ to $E_2$ is not possible.

One thing to note about $SQI_{SIGN}$'s specific protocol is that instead of viewing the keys, commitment, etc. as random tuples, (E, End(E)), we instead consider an ($E_0$, End($E_0$)) pair and then the keys are random isogenies $\varphi: E_0 \rightarrow E$. Also several ideals are involved. Here are the steps of the signing on a more detailed, lower level.

In the commitment stage, we generate the random isogeny $\varphi_{com} : E_0 \rightarrow E_1$ from a randomly generated ideal $I_{com}$ with norm $D_{com}$. The Normalized function normalizes $E_1$ (changes the Montgomery curve's coefficient on the $x^2$ term in its formula) and modifies $\varphi_{com}$ to map to it. $B_{0, D_{chall}}$ where $D_{chall}$ was precomputed is pushed through this isogeny to get $B_{1, D_{chall}}$ which is written as ($P_1$, $Q_1$).

The challenge phase randomly generates $\varphi_{chall} : E_1 \rightarrow E_2$ and hashes normalized $E_1$ with the message to be sent obtaining a scalar value \`a\`. Then a basis of $E_1[D_{chall}]$ is found yielding $R_1$ and $S_1$ and finally the challenge isogeny's kernel is computed: $K_{chall} = R_1 + [a]S_1$. $\varphi_{chall}$ and $E_2$ go through a similar normalizing process that $\varphi_{com}$ and $E_1$ went through. $K_{chall}$ is then expressed as a linear combination of ($P_1$, $Q_1$) and then using the KernelDecompositionToIdeal function, an ideal $I_{chall}$ is found. Finally, a generator of the challenge isogeny, Q, and scalar, r, are found such that $[r]Q = K_{chall}$.

The response phase finds an equivalent ideal of power-of-2 norm: $J \sim \underline{I_{secret}} \cdot I_{com} \cdot I_{chall}$.

$J$ is then converted into an isogeny. The final signature is this isogeny, the $r$ computed earlier, and an 's' that is part of the details of the challenge algorithm.

**Verification Process**
The verification algorithm confirms the validity of a digital signature by ensuring it corresponds to the given message and public key. Given a message (msg), a signature (σ), and the public key (pk), the algorithm confirms whether the signature matches the message and public key. It does so by recomputing the isogenies used in the signing process and verifying the consistency between the message, signature, and public key.

**Algorithm 27: SQISign.Verify**
Inputs: Msg is the original message, σ is the Signature which consists of compressed response data (zip), value (r), and parameters (s) and pk is the public key of the signer, which represents the curve $EA$.
Process: First, extract the compressed response isogeny zip, scalar r, and auxiliary data s from the signature σ and reconstructs the elliptic curve $E$2 and a point $Q$2 from the compressed isogeny zip using the public key $pk$, $\phi$resp: $EA \rightarrow E$2. Next, Verify the challenge isogeny $\phi$chall′: $E$2 $\rightarrow E$1 by using the auxiliary data $s$ which means that it checks that the kernel point $K$chall, reconstructed from $r$, aligns with the hash of $msg$ and the curve $E$1. Recompute the hash of $msg$ and the $j$-invariant of $E$1 which confirms that this hash matches the original challenge kernel used in signing.
Outputs: A Boolean indicating whether the signature is valid or not.
Purpose: The verification algorithm ensures message integrity and authenticity by validating the signature against the public key and the original message, so it ensures that the message was indeed signed by the holder of the corresponding private key.

**Compression and Decompression**
The signature process involves compressing both the response isogeny and the dual of the challenge isogeny. This compression minimizes the size of the signature while maintaining all necessary cryptographic information. Response Isogeny: It's broken down into smaller steps of a fixed length (for e.g.2^f) and each step is represented using two points P and Q, which define a structure called the kernel. Sometimes, P and Q are swapped based on a specific condition (indicated by a bit b) and these steps allow the process to compactly represent the transformation. Challenge Isogeny: It only involves a single transformation, but the degree Dchall isn't always a simple number (like a prime) so, it's broken into smaller factors (typically powers of 2 and 3) for easier processing.

**Algorithm 28: Decompressresp (E, s)**
Inputs: E is a normalized Montgomery curve representing the domain of the isogeny and s is compression data for the isogeny, consisting of b1: A bit indicating whether to swap basis points and (s1, s2, ..., se) which are scalars used to reconstruct the isogeny kernel.
Process: The algorithm begins by generating a torsion basis (P2f, Q2f) for the 2^f-torsion subgroup of the input curve E. The compression data s is decomposed into its components: b1 which determines whether to swap the torsion basis points (P2f, Q2f) and the remaining scalars

(s1, ..., se) guide the iterative reconstruction of the isogeny. For each scalar si: compute the kernel point K = P2f + [si]Q2f then construct the isogeny φ with this kernel and update the elliptic curve and basis points after applying φ.
After iterating through all scalars, the resulting curve E′ is normalized to a Montgomery form using Montgomery Normalize, ensuring compatibility and a specific generator point Q on E′ is chosen, ensuring a consistent and valid result for operations.
Outputs: E′ is the codomain curve of the isogeny, in normalized Montgomery form and Q is a point on E′ that generates the 2-isogeny φ′.
Purpose: This algorithm reconstructs the isogeny and its output curve from compressed data, and it is crucial for decompressing isogenies efficiently in isogeny-based cryptographic protocols, enabling verification and other cryptographic operations without requiring the full isogeny data upfront.

**Algorithm 29: DecompressAndCheckchall**
Inputs: E is a normalized Montgomery curve, and s is compression of the isogeny ϕ, consisting of bits b1, b2 and scalars s1, s2. Q is a point on E of order 2, r is scalar for verification and msg is the original signed message.
Process: First, compute torsion basis for E and use b1, b2 to adjust basis points. Reconstruct kernel points K2 and K3 using s1, s2 and build and normalize isogenies ϕ2 and ϕ3, then compose ϕ = ϕ3 ◦ ϕ2. Compute a point Q′ deterministically from s. Verify hash H(msg) aligns with r and ϕ(Q′).
Output: Boolean indicating if the signature is valid or not.
Purpose: This algorithm ensures the signature corresponds to the message and public key, validating authenticity.

**NIST Security Levels (NIST-I, NIST-III, NIST-V)**
A parameter set consists in a choice of prime p (the characteristic of the field), and a bound B on the prime factors of T. These levels are defined as parameter sets to match different security requirements based on cryptographic strength, typically measured against quantum computing attacks.
**NIST-I:**
Purpose: Suitable for systems requiring moderate security.
Prime Characteristic (p): A 197-bit prime represented as p {1973}.
Bound B: Value: 2000, used to restrict the search space for torsion points.
Factorization of T: T is decomposed into small primes to ease the computation.
Example: T = 2^36 · 3^6 · 7^4 · ... · 1973.

**NIST-III:**
Purpose: Provides a higher security level, typically equivalent to classical 192-bit security.
Prime Characteristic (p): A 441-bit prime p {7441}.
Bound B: Value: 48000, allows for computations over a larger set of torsion points.
Factorization of T: Example: T = 3^68 · 5^7 · 7^12 · ... · 47441.

**NIST-V:**

Purpose: Designed for the highest level of security, similar to 256-bit post-quantum security.
Prime Characteristic (p): A 1223-bit prime p {312233}.
Bound B: Value: 320000. Offers the broadest range of torsion points.
Factorization of T: Example: T = 3^72 · 5^7 · 7^16 · ... · 318233.

**Comparison of Levels:**

| Security Level | Prime Size (p) | Bound (B) | Factorization Complexity (T) |
|---|---|---|---|
| NIST-I | 197 bits | 2000 | Low |
| NIST-III | 441 bits | 48000 | Moderate |
| NIST-V | 1223 bits | 320000 | High |

**Binary Format**

The binary format specifies how cryptographic objects are encoded for transmission, ensuring compatibility across systems. Field Elements: Elements of Fp and Fp² are encoded in little-endian format with the real and imaginary parts concatenated for Fp². Integers and Quaternions: Integers are encoded in little-endian two's complement and quaternions are encoded as five integers. Elliptic Curve Points: Points are encoded by their x-coordinate; torsion bases are concatenations of P, Q, and P − Q. Keys and Signatures: Secret Keys encode the public key, quaternion α, and torsion basis components. Public Keys are encoded as the A-coefficient of the Montgomery curve. Signatures encode compressed isogeny, integer r, and tuple s. This encoding ensures efficient data transmission and storage.

# Chapter 5 – Parameter Search and Choices

This chapter focuses on the choice of parameter requirements, searches, and choices for the instantiations of SQIsign in the NIST-I, NIST-III, and NIST-V security levels. Concerning SQIsign, parameter requirements refer to the specific set of mathematical parameters chosen to optimize efficiency and security in the signature scheme. However, concerning SQIsign, the specific choice of parameters within the requirements defined in this research has no impact on the security of SQIsign. The efficiency of SQIsign is based on choosing a prime *p* of suitable size such that $p^2 - 1$ contains many smooth prime factors. There are five recommended sizes for security parameters λ. I will be giving three here and they include Prime size of $log_2(p)$ ; the degree $D_{com}$ of the commitment isogeny $\varphi_{com}$ of size approximately $2^{2\lambda} \approx p$; the degree of $D_{chall}$ of the challenge isogeny $\varphi_{chall}$ of size approximately $2^{\lambda} \approx p^{1/2}$. The parameter of our choice should mainly prioritize fast verification and in doing so maintain reasonable signing performance. Following this, we are going to look at suitable methods for finding parameters. One of which includes the **Sieve-and-boost.** In this method, we search for primes of the form $p = 2(2^{f'}3^{g'}x)^n - 1$ for a smooth number $x$ and for different values of $f' g'$ $such\ that\ 2^{nf'} . 3^{n^{g'}} \geq \sqrt{p}$ to ensure that $\varphi_{chall}$ can be computed as a chain of 2-isogenies and 3-isogenies. When searching for primes of bit size smaller than 256, 384, and 512 bits respectively, the search space for *x* is roughly of size $\frac{log_2 p}{2n}$ Bits. The algorithm starts by identifying smooth numbers in the search interval for *x* using the sieve implementation. For each smooth *x* and suitable values for *f', g'* we store the primes of the form $2(2^{f'}3^{g'}x)^{n_{|v|}} - 1$ then simultaneously computing the

B-smooth parts of $p^2 - 1$ by using a product tree. The numbers$n_{|v|} \in \{n_{I,} n_{III,} n_{IV}\}$ are chosen to give primes suitable for the tree security levels.

# Chapter 6 – Performance Analysis

The SQIsign library is built into several sub-modules that are linked to libraries supporting the NIST Signature API such as EC (Elliptic Curve) and isogeny computation, implementation of the KLPT algorithm, implementation of the ideal to isogeny algorithm also known as Id2iso, module for quaternion computation, arbitrary precision module based on GMP (GNU Multiple Precision Arithmetic Library), implementation of the signature key generation signing and verification protocols. These modules support flexibility development, allowing separate handling of different mathematical and computational tasks within the signature scheme.
The **optimized implementation** is the same as the **reference implementation**. The reference implementation is built with two configurations which are using the GMP system installation on Ubuntu 22.04LTS and using a custom-built version with disabled assembly code.

## Key and Signature Size

The SQIsign key and signature sizes are listed below for each security level. The key and signature size of SQIsign is critical for ensuring both security and efficiency. A higher key size shows a great level of resistance to attacks. A smaller signature size reduces the amount of data transferred which is important. Minimal key and signature sizes are most preferred in SQIsign to reduce memory and storage requirements.

TABLE 1. SQISIGN key and signature sizes in bytes for each security level.

| Parameter set | Public key | Secret key | Signature |
|---|---|---|---|
| NIST-I | 64 | 782 | 177 |
| NIST-III | 96 | 1138 | 263 |
| NIST-V | 128 | 1509 | 335 |

The **optimized implementation** is the same as the **reference implementation**. The reference implementation is built with two configurations which use the GMP system installation on Ubuntu 22.04LTS and a custom-built version with disabled assembly code.
The **performance evaluation** was performed on an Intel x86 64-bit CPU with results being the median of ten benchmark runs.

# Chapter 7 – Implementation Details

This section deals with implementation details and constants that are used in various algorithms and protocols. These constants drive the efficiency and accuracy of its algorithms. These constants help the system achieve reliable outcomes. An example is when searching for prime numbers within a fixed bit size. The KLPT_random_prime_attempts is used when looking for a random prime number of bit size *k.* We try at most *k*(KLPT_random_prime_attempts) random integers until a prime is found. This comes from the fact that a random number of bit size *k* has a probability$\frac{-ln\,2}{k}$ Of being prime. The KLPT_equiv_bound_coeff is a constant that is bound on the absolute value of the coefficients of the linear combination of the small basis used to find the equivalent ideal. Also, KLPT_equiv_num_iter is a constant which is the maximum

number of trials to find an equivalent ideal of prime norm. The selection of these constants is determined as thus: $Let\ B := KLPT\_equiv\_num\_iter \quad C := KLPT\_equiv\_num\_iter$ taking a value of *C* roughly equal to the size of the search space, so this means $C = (2B + 1)^3(B + 1) \approx 8B^4$. For a typical ideal, we have $D \approx p^{\frac{1}{2}}$ ; $we\ choose\ the\ value\ of\ B\ such\ that$ $(\frac{(1+2In\,2)}{log\,(p)})^c$ is negligible.
There is a constant KLPT_repres_num_gamma_trial that defines the maximum number of attempts made in the algorithm FullRepresentInteger defined as thus: KLPT_repres_num_gamma_trial = $2^{\text{KLPT\_gamma\_exponent\_center\_shift}}$.
Also, there are various constants in SigningKLPT including KLPT_signing_klpt_length; this constant gives the length of the output of the signing KLPT algorithm. Also we have the KLPT_signing_number_strong_approx which is the number of vectors tried for the strong approximation step inside the signing KLPT algorithm and we also have two other constants. These constants are fixed so that

$$2^{-\texttt{KLPT\_signing\_num\_gamma\_trial}} \leq \mathsf{negl},$$
$$-\log\left(1 - \frac{1}{\log(p)}\right)\left(\begin{array}{c} 2^{\texttt{KLPT\_gamma\_exponent\_center\_shift}} \\ -\texttt{KLPT\_signing\_num\_gamma\_trial} \end{array}\right) \geq -\log(\mathsf{negl}),$$
$$(1 - 4/(13\log(p)))^{\texttt{KLPT\_signing\_number\_strong\_approx}} \leq 1/64,$$
$$2^{-2\cdot\texttt{KLPT\_signing\_klpt\_length}-(15/4)\log(p)-25} \leq \mathsf{negl},$$

Furthermore, we have KeyGenKLPT, and the constants for these are: KLPT_keygen_length which is the constant of the length of the alternate secret key isogeny. We also have the **KLPT_Keygen_number_strong_approx** which is the constant of the number of vectors tried for the strong approximation step inside the KeyGenKLPT algorithm. This signing algorithm can be fixed the same way it was done previously (the signing KLPT algorithm).
The various constants for the SpecialEichlerNorm can be defined as follows: the **KLPT_eichler_num_alternate_order** which is the number of precomputed orders that we use can be fixed by setting it equal to 7. Also, we have the **KLPT_eichler_number_strong_approx** which is the maximum number of trials for the strong approximation step inside the Eichler norm algorithm; this parameter can be fixed by setting the it to the ceiling function of $10log(p)$. We also have the **KLPT_eichler_norm_equiv_ideal** which is the number of equivalent ideals tried before aborting; this parameter can be fixed by the ceiling function of $\frac{log\,p}{10}$. We have the **KLPT_eichler_number_mu_norm** which is equal to the ceiling function of $\frac{log(T)-\frac{5}{4}log\,p\,))}{log(3)}$ which can be fixed by the maximum number of target norms tried for $n\mu$.

# Chapter 8 – Heuristics

## 8.1 - Assumptions

SQI$_{\text{SIGN}}$ is built on a few critical assumptions: primality of certain parameters, quadratic residues and non-residues, random lattices, and the importance of the ideal size. Primality is used in several key places when discussing the security of the scheme, but perhaps the most important is in the construction of the finite field where the modulus *p* is a very large prime number, thereby ensuring that the field cannot be subjected to a factorization attack.

Tied together with primality is the idea of quadratic residuosity, where a quadratic residue is a number that can be expressed as the square of an integer within a given modular system. For example, if our prime number *p* is equal to 7, we would use modular arithmetic for the square numbers up to 7. This is going to give us a set of numbers that we can call residues. In this example, our residues would be the set {0,1,2,4} and the remaining set {3,5,6} we can refer to as non-residues. Both of these sets of numbers will be incorporated into $\text{SQI}_{\text{SIGN}}$ in different but important ways. The residues will ensure that the structure of the elliptic curves remains intact by ensuring that certain points fall on the curve, while the non-residues are used in the generation of the isogeny path. This provides an extra layer of security as an attacker would have a very tough time telling the difference between the residues and non-residues.

Another assumption we can make is that the lattices will be random. As has been discussed previously, lattices are used in key generation, signing, and transformations. Randomness of the lattices ensures unpredictability, making it impossible for an attack in which there is an attempt to reconstruct the lattices, ensuring the security of the private key.

The final assumption to be discussed is perhaps the most obvious, the notion of finding the optimal ideal sizes. If an ideal is small the security isn't as strong, but larger ideals will lead to greater computational overhead, potentially making the scheme far too expensive.

## 8.2 - Subroutines

The strength of a cryptographic protocol often lies in the intricacy of its subroutines, and $\text{SQI}_{\text{SIGN}}$ is no different. The core of the scheme involves the supersingular elliptic curves and isogeny paths, both of which involve very complex algebraic structures. There are several subroutines critical to the security of the protocol: RandomEquivalentPrimeIdeal, KeyGenKLPT, SpecialEichlerNorm, IdealToIsogenyEichler, and other further randomizations. We have discussed most of the mechanics of these subroutines already, so here the focus will be their contributions to the security of the scheme:

- o RandomEquivalentPrimeIdeal generates a random ideal equivalent to a given ideal, ensuring the randomness of the structure adds unpredictability to the protocol. This is a large part in what makes the scheme resistant to a lot of known attacks.
- o KeyGenKLPT, which we've covered a few times, ensures that the ideal corresponds to a valid isogeny. Without this subroutine, the private key could be invalid or easy to guess.
- o SpecialEichlerNorm is the subroutine that handles performance maintenance without sacrificing security.
- o IdealToIsogenyEichler is the subroutine that actually converts the ideals to isogenies, ensuring that the protocol preserves the security guarantees and works seamlessly.
- o Last, there are other randomizations that are used in $\text{SQI}_{\text{SIGN}}$ , making it even harder to attack any one part of the scheme.

# Chapter 9 – Security Analysis

## 9.1 – Security Reductions

There are a few properties that the sigma protocol must satisfy: correctness, special soundness, weak Honest-Verifier Zero Knowledge (wHVZK), Impersonation Under Passive

Attack (IMP-PA), and finally Existential Unforgeability Under Chosen Message Attacks (EUF-CMA) after the Fiat-Shamir Transform.

The correctness property says that the verifier will always accept a valid proof. The verifier function is V(x, comm, chall, resp)=1, where x is the public key, comm is the commitment, chall is a random challenge from the verifier, and resp is the response from the prover.

Special soundness means that if a prover can produce two valid responses to the same challenge, they must know the private key. However, if a cheating prover only knows the public key *x* and does not witness the private key *w,* they can't force the verifier to accept.

wHVZK ensures that even if the verifier is honest during the protocol, they can't get any additional information about the private key. This is only partially handled in $\mathrm{SQI}_{\mathrm{SIGN}}$ in the sense that the scheme doesn't create "fake" results as some other schemes do, but because there are so many isogenies created during the key generation it would be impossible to extract the real one.

IMP-PA security says that an attacker simply observing the protocol can't forge the signature or get information about the private key. There is a theorem that states that if a scheme satisfies both special soundness and wHVZK, it also satisfies IMP-PA.

There is another theorem that says any scheme that satisfies IMP-PA and undergoes a Fiat-Shamir transform, that scheme then satisfies EUF-CMA.. This says that even after observing signatures on other chosen messages, an attacker can't forge a valid signature for any new message.

**9.2 – Resistance to Known Attacks**

Carefully choosing certain parameters in the sigma protocol ensures that (so far) there is no way to break the security that $\mathrm{SQI}_{\mathrm{SIGN}}$ offers.

Theoretically, if one could compute the endomorphism ring or the isogeny path, they could break the protocol (this is true not just of $\mathrm{SQI}_{\mathrm{SIGN}}$ but any isogeny-based scheme). However, no such algorithm exists even in quantum computing.

Key recovery is also not possible as an attacker would have to either randomly guess the private key or reverse-engineer the isogeny from the destination curve (public key). Again, there is no algorithm that can reverse-engineer an isogeny, and while one could theoretically randomly guess the private key, the probability of this is close enough to zero that it is deemed quantum resistant.

Forgery is another attack that has been proven to be ineffective, as even if an attacker observes multiple valid signatures, the randomness and unpredictability in the protocol means they can't use any of the information to construct a signature for a new message. This is due to the fact that the signature generation depends on the private key and isogeny path, neither of which can be obtained from the public key or other signatures.

## Bibliography

Chavez-Saab, J., Santos, M. C.-R., De Feo, L., Eriksen, J. K., Hess, B., Kohel, D., Leroux, A., Longa, P., Meyer, M., Panny, L., Patranabis, S., Petit, C., Rodríguez Henríquez, F., Schaeffler, S., & Wesolowski, B. (2023, June 1). *SQIsign Algorithm Specifications and Supporting Documentation*. Version 1.0. SQIsign.org. https://sqisign.org

"Isogeny." *Wikipedia*, 10 Apr. 2023, en.wikipedia.org/wiki/Isogeny.

OpenAI. "ChatGPT ." *ChatGPT*, OpenAI, 2024, chatgpt.com/.

Pound, Mike. "What Are Digital Signatures? - Computerphile." *Www.youtube.com*, 2021, www.youtube.com/watch?v=s22eJ1eVLTU. Accessed 10 Oct. 2021.

PQShield. "PQC Scheme: SQISign." *YouTube*, 8 Nov. 2023, www.youtube.com/watch?v=9WfENxo82jg. Accessed 30 Sept. 2024.

Wikipedia Contributors. "Digital Signature." *Wikipedia*, Wikimedia Foundation, 4 June 2019, en.wikipedia.org/wiki/Digital_signature.

---. "Quaternion Algebra." *Wikipedia*, Wikimedia Foundation, 21 Feb. 2024, en.wikipedia.org/wiki/Quaternion_algebra. Accessed 30 Sept. 2024.

Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography* (3rd edition), (Chapman & Hall/CRC Cryptography and Network Security Series)(2020)

"Algebraic structure." *Wikipedia*, https://en.wikipedia.org/wiki/Algebraic_structure

"Field." *Wikipedia*, https://en.wikipedia.org/wiki/Field_(mathematics)

"Finite field." *Wikipedia*, https://en.wikipedia.org/wiki/Finite_field

"Elliptic curve." *Wikipedia*, https://en.wikipedia.org/wiki/Elliptic_curve

"Degree of a Polynomial." Wikipedia, 3 June 2024, en.wikipedia.org/wiki/Degree_of_a_polynomial#:~:text=Degree%204%20%E2%80%93%20quartic%20(or%2C.

Santos, Maria. "Maria Santos." PhD Student at UCL, Interested Particularly in Post-Quantum Cryptography and Applications., 7 Nov. 2020, www.mariascrs.com/2020/11/07/velus-formulas.html.

"Lattice Cryptography." *Enum*, cseweb.ucsd.edu/~daniele/LatticeLinks/Enum.html. Accessed 23 Sept. 2024.

"Lattice Reduction." *Wikipedia*, Wikimedia Foundation, 22 Jan. 2024, en.wikipedia.org/wiki/Lattice_reduction.

Regev, Oded. *Lecture 3 CVP Algorithm 1 the Nearest Plane Algorithm*, cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/cvp.pdf. Accessed 23 Sept. 2024.

"Norm". *Wikipedia,* https://en.wikipedia.org/wiki/Norm_(mathematics)

"Quaternion". *Wikipedia,* https://en.wikipedia.org/wiki/Quaternion

DRP, Turkey. Salim Erdem Kocak "Isogeny Based Cryptography and KLPT Algorithm". *YouTube,* uploaded by DRP Turkey, 5 Sept. 2024, https://www.youtube.com/watch?v=vNc9svEBCf0&list=WL&index=1&t=751s

Leroux, Antonin. *Quaternion Algebras and Isogeny-Based Cryptography*. 6 Sept. 2022, https://www.lix.polytechnique.fr/Labo/Antonin.LEROUX/manuscrit_these.pdf

"Zero-knowledge proof." Wikipedia, https://en.wikipedia.org/wiki/Zero-knowledge_proof

"Commitment scheme." Wikipedia, https://en.wikipedia.org/wiki/Commitment_scheme

"Fiat-Shamir heuristic." Wikipedia, https://en.wikipedia.org/wiki/Fiat%E2%80%93Shamir_heuristic