**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 𝐸2 and a point 𝑄2 from the compressed isogeny zip using the public key 𝑝𝑘, 𝜙resp: 𝐸𝐴 → 𝐸2. Next, Verify the challenge isogeny 𝜙chall′: 𝐸2 → 𝐸1 by using the auxiliary data 𝑠 which means that it checks that the kernel point 𝐾chall, reconstructed from 𝑟, aligns with the hash of 𝑚𝑠𝑔 and the curve 𝐸1. Recompute the hash of 𝑚𝑠𝑔 and the 𝑗-invariant of 𝐸1 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 complements 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 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.