Chapter 3 - Signature

This chapter covers the heart of the SQI_{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 SQI_{SIGN}'s practical operation (it's 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.
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.

The Σ-protocol used in SQI_{SIGN} in particular passes around elliptic curves and endomorphism rings of elliptic curves. Given an elliptic curve E, an endomorphism is an isogeny $\varphi : E \rightarrow E$ (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). 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. 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 Σ-protocol 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 actual protocol:

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 φ_{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 φ_{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.

We are interested in a non-interactive proof of knowledge unlike the interactive sigma protocol outlined so far. We would like 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 verification key, x, will be made available to anyone that seeks to verify the signature. Secondly, the signer will compute a commitment $P_2(x, w) \rightarrow com$ 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, $H(com \ // \ msg) \rightarrow chall$. 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).

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.