

## Chapter 3 - Signature

This chapter covers the heart of the  $\text{SQL}_{\text{SIGN}}$  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 - $\Sigma$ -protocols and the Fiat-Shamir Heuristic

The essence of  $\text{SQL}_{\text{SIGN}}$ 's practical operation (its functioning as a scheme) is that of an interactive proof of knowledge called a  $\Sigma$ -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  $\Sigma$ -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  $\Sigma$ -protocol over a set of  $(x, w)$  along with a security parameter  $\lambda$  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  $\Sigma$ -protocol consists of a three-way exchange of information:

- 1) Commitment - the prover runs  $P1(x, w) \rightarrow \text{com}$ . 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  $\lambda$ ,  $\text{chall}$ , 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  $P2(\text{chall}) \rightarrow \text{resp}$  and sends this response to the verifier.

Finally the verifier checks the trustworthiness of the prover by computing  $V(x, \text{com}, \text{chall}, \text{resp})$  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 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  $\lambda$ . 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  $P_2(x, w) \rightarrow \text{com}$  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,  $H(\text{com} \parallel \text{msg}) \rightarrow \text{chall}$ . The signer then generates a response from this challenge,  $P_2(\text{chall}) = \text{resp}$ . After all of this, independent verifiers can cross check this information to ascertain the validity of the signature on the data; it computes  $V(x, \text{com}, \text{chall}, \text{resp})$  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.

### Section 3.2 - Precomputation

We need some precomputed data before  $\text{SQL}_{\text{SIGN}}$  can run its algorithm.  $E_0$  is the special elliptic curve:  $y^2 = x^3 + x$ . When  $p \equiv 3 \pmod{4}$ , over  $F_{p^2}$ ,  $E_0$  is supersingular. We need a torsion value,  $T$ , that among other things is smooth so we can find efficiently find a basis for points in  $E_0[T]$  (recall the  $T$ -torsion subgroup of  $E_0$ , are the points on the curve satisfying  $[T]P = \infty$  and we can “easily” find a basis for smooth  $T$  via an algorithm similar to Pohlig-Hellman). We call this 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.

### Section 3.3 - Key generation

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

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

The signing key contains:

- 1) A quaternion,  $\alpha$ , that connects two ideals  $I_{\text{secret}}, J_{\text{secret}}$  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  $\varphi_{\text{secret}} : E_0 \rightarrow E_A$

- 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  $\varphi_{\text{secret}}$ .

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

After  $\varphi_{\text{secret}}$  is found,  $\alpha$  is conjugated. The algorithm Normalized takes in  $\varphi_{\text{secret}}$  and outputs  $E_A$  (the public key) and changes  $\varphi_{\text{secret}}$  so that it maps to the normalized elliptic curve  $E_A$  (i.e.  $\varphi_{\text{secret}}: E_0 \rightarrow E_A$ ). The basis  $B_{A,T}$  mentioned before is computed using the newly found  $\varphi_{\text{secret}}$ . 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  $\varphi_{\text{secret}}: E_0 \rightarrow E_A$  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  $\Sigma$ -protocol (that will be transformed into a Digital Signature Scheme by the Fiat-Shamir Transform) used in  $\text{SQL}_{\text{SIGN}}$  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,  $\text{End}(E)$ , 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:  $(f \circ g)(P) =$

$f(g(P))$ ; this is not necessarily commutative hence we consider rings here instead of fields. The endomorphism ring problem is to find  $\text{End}(E)$  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  $\text{End}(E_A)$  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  $\text{SQL}_{\text{SIGN}}$  protocol on a high level:

- 1) Commitment - the prover randomly generates  $(E_1, \text{End}(E_1))$  and sends  $E_1$  to the verifier.
- 2) Challenge - the verifier randomly generates an isogeny  $\phi_{\text{chall}} : E_1 \rightarrow E_2$  and sends  $\phi_{\text{chall}}$  to the prover.
- 3) Response - since the prover knows  $\text{End}(E_1)$  and now knows  $\phi_{\text{chall}} : E_1 \rightarrow E_2$ , they can compute  $\text{End}(E_2)$ . The prover can then use the knowledge of  $\text{End}(E_A)$  and newly computed  $\text{End}(E_2)$  to compute  $\phi_{\text{resp}} : E_A \rightarrow E_2$  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  $\phi_{\text{resp}}$  is indeed an isogeny from  $E_A$  to  $E_2$ . This (almost) works since the prover will likely have to use its knowledge of  $\text{End}(E_A)$  to compute  $\phi_{\text{resp}} : E_A \rightarrow E_2$ . The only problem is that if a dishonest prover generates  $E_1$  initially by choosing a random isogeny from the public key to  $E_1$  ( $\phi_{\text{cheat comm}} : E_A \rightarrow E_1$ ), when given  $\phi_{\text{chall}} : E_1 \rightarrow E_2$  in the challenge step, they can then simply compose the isogenies ( $E_A \rightarrow E_1 \rightarrow E_2$ ) and yield  $\phi_{\text{cheat resp}} : E_A \rightarrow E_2$ . To fix this, the verifier can choose a challenge  $\phi_{\text{chall}} : E_1 \rightarrow E_2$  where such a composition of isogenies to get from  $E_A$  to  $E_2$  is not possible.

One thing to note about  $\text{SQL}_{\text{SIGN}}$ 's specific protocol is that instead of viewing the keys, commitment, etc. as random tuples,  $(E, \text{End}(E))$ , we instead consider an  $(E_0, \text{End}(E_0))$  pair and then the keys are random isogenies  $\phi : E_0 \rightarrow E$ . 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  $\phi_{\text{com}} : E_0 \rightarrow E_1$  from a randomly generated ideal  $I_{\text{com}}$  with norm  $D_{\text{com}}$ . The Normalized function normalizes  $E_1$  (changes the Montgomery curve's 'A' coefficient value in its formula) and modifies  $\phi_{\text{com}}$  to map to it.  $B_{0, D_{\text{chall}}}$  where  $D_{\text{chall}}$  was precomputed is pushed through this isogeny to get  $B_{1, D_{\text{chall}}}$  which is written as  $(P_1, Q_1)$ .

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

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

## Bibliography

“Zero-knowledge proof.” Wikipedia, [https://en.wikipedia.org/wiki/Zero-knowledge\\_proof](https://en.wikipedia.org/wiki/Zero-knowledge_proof)

“Commitment scheme.” Wikipedia, [https://en.wikipedia.org/wiki/Commitment\\_scheme](https://en.wikipedia.org/wiki/Commitment_scheme)

“Fiat-Shamir heuristic.” Wikipedia,  
[https://en.wikipedia.org/wiki/Fiat%E2%80%93Shamir\\_heuristic](https://en.wikipedia.org/wiki/Fiat%E2%80%93Shamir_heuristic)