CHAPTER 1

Seraphis knowledge proofs

The idea is to provide a comprehensive set of tools to prove knowledge of a specific information. This can be used, for instance, for the purpose of proving/verifying that: a payment was done, a person has a certain available balance, a person owns a certain address and etc. These tools open the path for a complete audit framework.

1.1 Enote ownership proof

The idea is to prove knowledge of an enote by verifying equation 1.1.

$$K_o = k_g^o G + k_x^o X + k_u^o U + K_1 \tag{1.1}$$

where,

$$k_g^o = \mathcal{H}_n(\text{“jamtis\_sender\_extension\_g”}, K_1, q, C) \tag{1.2}$$
$$k_x^o = \mathcal{H}_n(\text{“jamtis\_sender\_extension\_x”}, K_1, q, C) \tag{1.3}$$
$$k_u^o = \mathcal{H}_n(\text{“jamtis\_sender\_extension\_u”}, K_1, q, C) \tag{1.4}$$

1.1.1 Prover

The prover provides:

- q - Sender-receiver-secret
- K_1 - Jamtis-address-spend-key
- K_o - Enote onetime-address

1.1.2 Verifier

The verifier gets the recorded commitment amount on chain for the enote being proved and he reconstructs the enote from equation 1.1.

1.2 Enote amount proof

The idea is to prove knowledge of the amount in an enote by verifying equation 1.5.

$$C = aG + bH \tag{1.5}$$

where G and H are generators.

1.2.1 Prover

The prover provides:

- a - Blinding factor
- b - Amount
- C - Public Commitment

1.2.2 Verifier

The verifier simply verifies if the equation 1.5 holds.

1.3 Enote sent proof

The idea is to prove that an enote was sent to a certain address or received by a certain addess. This proof is the equivalent of the OutProofs and InProofs in the legacy protocol. There are two ways of creating them and in both cases the prover has to provide the sender-receiver-secret described by equation 1.6[1]:

[1] The domain separator here can be “jamtis_sr_secret_plain” for non self-send enotes or one of the self-send variations of domain separator like: “jamtis_selfsend_dummy”, “jamtis_selfsend_change” and “jamtis_selfsend_self_spend”

$$q = \mathcal{H}_{32}(\text{“domain-separator”}, K_d, K_e, \text{input\_context}) \tag{1.6}$$

Notice that

$$K_d = r * K_2 = k_{fr} * K_e \tag{1.7}$$

where K_d is the sender-receiver Diffie-Hellman derivation. r is the enote ephemeral-private-key. K_2 is the public address K_2 of the receiver. k_{fr} is the find-received private key. K_e is the enote ephemeral-public key.

- The sender of enote K_o to address K_1 wants to prove that he sent this enote so he has to provide q (sender-receiver-secret) and the only way for him to do it is by knowing the enote ephemeral-private-key r and the address of the recipient K_2 to generate K_d. After that he can calculate q using equation 1.6.
- The receiver of enote K_o to address K_1 wants to prove that he received this enote so he has to provide q (sender-receiver-secret) and the only way for him to do it is by knowing the find-received private key k_{fr} and the enote ephemeral-public key (which is public knowledge) K_e to generate K_d. After that he can calculate q using equation 1.6.

1.3.1 Prover

The prover provides:

- *enote_ownership_proof* - Enote Ownership Proof
- *enote_amount_proof* - Enote Amount Proof

1.3.2 Verifier

The verifier gets both proofs and the corresponding enote recorded on-chain and checks if the proofs are valid as explained in the previous sections.

1.4 Address ownership proof

The idea is to prove that someone knows the master and view-balance private keys (k_m and k_{vb}) corresponding to a certain address K_1.

The proof works for either K_1 and $K_1^{base}(K_s)$ defined by equations 1.8.

$$K_1 = k_g^j G + k_x^j X + k_u^j U + K_s \tag{1.8}$$
$$K_s = k_{vb} X + k_m U \tag{1.9}$$

1.4.1 Prover

The prover does the following:

1. Generates a scalar $k_g^{offset} = \mathcal{H}_n(K_1)$.
2. Generates a new address $K_o^{new} = k_g^{offset} G + K_1$.
3. Makes a composition proof on K_o^{new} using a message agreed by the verifier.
4. Publishes the proof and the corresponding key image.

1.4.2 Verifier

The verifier gets the composition_proof and the key image, he calculates k_g^{offset} and he gets the corresponding address being proved (K_1). Finally he performs a composition proof on the new address K_o^{new}. Notice that it works because the goal of the composition proofs is to prove knowledge of (a, b, c) in the equation $K = aA + bB + cC$ without revealing (a, b, c) in the bases (A, B, C).

1.5 Address index proof

The idea is to prove that someone knows the index from which a jamtis address was built by proving knowledge of the generator *gen* defined by equation 1.10.

$$gen = \mathcal{H}_{32}[k_{ga}](j) \tag{1.10}$$

Notice that the terms from equation 1.8 can be described by equations 1.12-1.14.

$$\begin{align}
k_g^j &= \mathcal{H}_n(\text{“jamtis\_spendkey\_extension\_g”}, K_s, j, generator) \tag{1.11}\\
k_x^j &= \mathcal{H}_n(\text{“jamtis\_spendkey\_extension\_x”}, K_s, j, generator) \tag{1.12}\\
k_u^j &= \mathcal{H}_n(\text{“jamtis\_spendkey\_extension\_u”}, K_s, j, generator) \tag{1.13}\\
& \tag{1.14}
\end{align}$$

1.5.1 Prover

The prover does the following:

1. Calculates the generator (since he knows k_{ga} and j).
2. Publishes K_s, K_1, *gen* and j.

1.5.2 Verifier

The verifier tries to reconstruct the address K_1 and checks if it matches the original one.

1.6 Enote key image proof

The idea is to prove that a key image corresponds to a certain enote (onetime-address).

1.6.1 Prover

The prover does the following:

1. Calculates the key image corresponding to the enote.
2. Calculates the composition proof.
3. Publishes the composition proof.

1.6.2 Verifier

The verifier gets the onetime-address, the supposed key-image corresponding to it and verifies the composition proof. If it is valid then the key image corresponds to that onetime-address and the prover knows all the openings.

1.7 Transaction funded proof

The idea is to prove that someone funded a particular transaction without exposing the transaction inputs.

1.7.1 Prover

The prover does the following:

1. Generates a random t_k^{new}.
2. Calculates $K_o^{new} = t_k^{new} G + K_o$.
3. Make a composition proof on K_o^{new} using a message agreed by the verifier.
4. Publishes the composition proof and K_o^{new}.

1.7.2 Verifier

Verifies the composition proof together with the message they agreed on.

1.8 Reserve proof

The idea is to prove that someone has a certain unspent amount available to spend.

1.8.1 Prover

The prover does the following for each enote that he wants to expose the key image and the amount:

1. Generates an ownership proof.
2. Generates an amount proof.
3. Generates a key image proof.
4. Publishes the all the 3 proofs, the corresponding enotes, key images and ledger indices.

1.8.2 Verifier

Verifies that enotes are on-chain, verifies that key images are not on-chain yet and for each enote, he verifies the ownership proof, amount proof and key image proof.

1.9 Enote unspent proof

The idea here is the same given at ZtM2 for the UnspentProofs. The prover will make a proof showing whether the enote being tested was spent or not in that ring.

Therefore this proof may be valid only for a specific period, e.g. until the end of month May of 2023, as the prover can only provide proofs until a specific date as he does not know when his enotes will be used as a decoy in other transactions in the future.

Also, the unspent proofs can be very resource demanding as the verifier can't believe the prover and he has to scan the blockchain looking for all the transactions that an enote have appeared. Some research is being done in this regard.

1.9.1 Prover

The main idea of the proof is to not leak the key image of the original enote represented by equation 1.15 while at the same time verifying whether this key image corresponds to the enote being tested represented by equation 1.16.

$$\tilde{K} = \frac{(k_u^o + k_u^j + k_m)}{(k_x^o + k_x^j + k_{vb})}U = \frac{k_b}{k_a}U \tag{1.15}$$

$$K_o = k_a X + k_b U + k_c G \tag{1.16}$$

where the terms k_a,k_b and k_c are defined as in equation 1.17-1.19.

$$k_a = k_x^o + k_x^j + k_{vb} \tag{1.17}$$
$$k_b = k_u^o + k_u^j + k_m \tag{1.18}$$
$$k_c = k_g^o + k_g^j \tag{1.19}$$

This can be achieved by doing the following for each enote and key image being tested:

1. Exposing $k_a X$, $k_b U$, $k_c G$ and $k_a \tilde{K}^? = (k_x^o + k_x^j + k_{vb})\tilde{K}^?$.
2. Making a Schnorr proof on k_a in the bases $(X, \tilde{K}^?)$.
3. Making a Schnorr proof on k_b in the base U.
4. Making a Schnorr proof on k_c in the base G.

1.9.2 Verifier

The verifier must do the following steps for each enote and key image being tested:

1. Checks if $K_o = k_a X + k_b U + k_c G$.
2. Verify the Schnorr proof on $(X, \tilde{K}^?)$.
3. Verify the Schnorr proof on U.
4. Verify the Schnorr proof on G.
5. Checks if $k_a \tilde{K}^? \neq k_b U$. If they are the same then the enote was spent in that transaction.

1.9.3 Why it works?

Rearranging the key image (linking tag) equation:

- $k_a \tilde{K}^? = k_b * U$

Since we know the key image being tested ($\tilde{K}^?$), we can make a proof showing that we know $k_a \tilde{K}^?$ and $k_b U$ without revealing k_a and k_b and compare both.

The advantage of using this proof instead of reserve proofs or MProve+ is the privacy gain that one has by not exposing the key image (and letting the verifier know when you spend a transaction in the future). The disadvantage is the complexity from the verifier side as he has to scan the whole blockchain for a period of time to test all enotes being used as a decoy.