Citadel Protocol Specification

Dusk Network

March 22, 2023

Contents

1 General Overview

1.1 What is Citadel

A Self-Sovereign Identity (SSI) protocol serves the purpose of allowing users of a given service to manage their identities in a fully transparent manner. In other words, every user can know which information about them is shared with other parties, and accept or deny any request for personal information.

`Citadel` is a SSI protocol build on top of Dusk Network. Users of a service can get a *license*, which represents their *right* to use such a service. In particular, `Citadel` allows for the following properties:

- **Proof of Ownership:** a user of a service is able to prove ownership of a license that allows them to use such a service.
- **Proof of Validity:** users can prove ownership of a valid license, that has not been revoked.
- **Unlinkability:** no one can link any activity with other activities done in the network.
- **Decentralized Nullification:** when a user spends a license, everyone in the network learns that this happened, so it cannot be spent again.
- **Attribute Blinding:** the user is capable of deciding which information they want to leak, blinding any other sensitive information and providing only the desired one.

1.2 Document Organization

In Section 2 we define all the object types and entities involved in the protocol. In Section 3 we roll out the protocol with full details.

2 Definitions

2.1 The Roles involved

- **User:** TBD.
- **Service Provider:** TBD.

2.2 The Types involved

- **Request:** TBD.
- **License:** A license is an asset that represents the right of a user to use a given service. The structure is as follows:

$$\mathsf{L} = \{\mathsf{type}, \mathsf{pos}, \mathsf{nonce}, \mathsf{enc}, \mathsf{npk}, R\}.$$

  where

  - type can be transparent (0) or obfuscated (1).
  - pos is the position of the license into a Merkle tree of licenses.
  - nonce is a randomness needed to compute enc.
  - enc is a ciphertext of size 4.
  - (npk, R) is the note public key of the license's owner.

- **Session Cookie:** A session cookie is a secret value (thus, always obfuscated) known only by the user and the SP. It contains a set of openings to a given set of commitments. The structure is as follows:

$$= \{\mathsf{pos}, \mathsf{nonce}, \mathsf{enc}, \mathsf{npk}, \mathsf{R}\}.$$

  where

  - pos is the position of the session cookie into a Merkle tree of session cookies.
  - nonce is a randomness needed to compute enc.
  - enc is a ciphertext of size 3.
  - (npk, R) is the note public key of the SP.

3 Protocol Workflow

The workflow is depicted in Figure 1, and described with full details as follows.

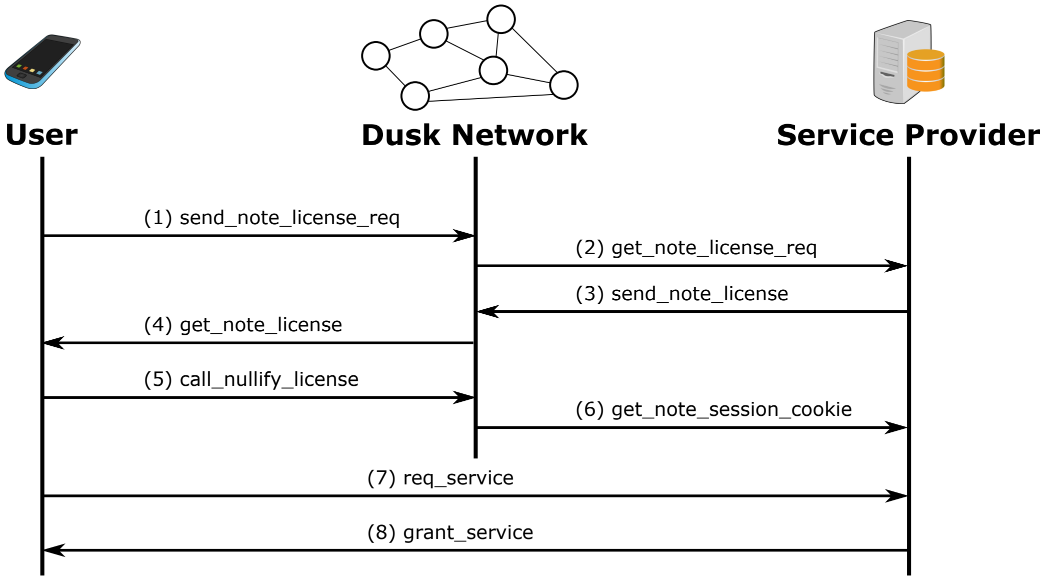

Figure 1: Overview of the protocol messages exchanged between the user, the Dusk Network, and the SP.

1. (**user**) $\mathsf{send\_note\_license\_req}$: Compute a note public key $(\mathsf{npk_{user}}, R_{\mathsf{user}})$ belonging to the user, using the user's own public key, and also an additional key $\mathsf{k_{user}} = H^{\mathsf{Poseidon}}(\mathsf{npk_{user}}, \mathsf{nsk_{user}})$, by computing first the user's $\mathsf{nsk_{user}}$. Then, send the required amount of Dusk coins to the SP, in order to pay for the service. Into the same transaction, send an NFT to the SP using the function $\mathsf{mint\_nft}(\mathsf{npk_{SP}}, R_{\mathsf{SP}}, \mathsf{payload_{NFT}}, \mathsf{k_{DH}})$, whose arguments are computed as follows:
   - $(\mathsf{npk_{SP}}, R_{\mathsf{SP}})$ is the SP's note public key, computed through his public key $\mathsf{pk_{SP}}$.
   - $\mathsf{payload_{NFT}} = (\mathsf{npk_{user}}, R_{\mathsf{user}}, \mathsf{k_{user}})$.
   - $\mathsf{k_{DH}}$ is computed using the SP's public key.

2. (**SP**) $\mathsf{get\_note\_license\_req}$: Continuously check the network for incoming license requests. Upon receiving the payment from a user, define a set of attributes *attr* representing the license, and compute a digital signature as follows:

$$\mathsf{sig_{lic}} = \mathsf{sign\_single\_key}_{\mathsf{sk_{SP}}}(\mathsf{npk_{user}}, \mathsf{attr})$$

3. (**SP**) $\mathsf{send\_note\_license}$: Set the $\mathsf{payload_{NFT}} = \{\mathsf{sig_{lic}}, \mathsf{attr}\}$, and send the license to the user using the function $\mathsf{mint\_nft}(\mathsf{npk_{user}}, R_{\mathsf{user}}, \mathsf{payload_{NFT}}, \mathsf{k_{user}})$.

4. (**user**) $\mathsf{get\_note\_license}$: Receive the note containing the license.

5. (**user**) $\mathsf{call\_nullify\_license}$: When desiring to use the license, nullify it by executing a call to the license contract. The following steps are performed:
   - The user sets a session cookie $\mathsf{sc} = (\mathsf{s_0}, \mathsf{s_1}, \mathsf{s_2}) \leftarrow \mathbb{F}_t$.
   - The user creates a new NFT note where $\mathsf{payload_{NFT}} = \mathsf{sc}$, and the SP is the receiver.
   - The user issues the transaction that includes the NFT described in the previous step, by calling the license contract. In this case, the $\mathsf{tx\_proof}$ is computed as done in the standard Phoenix model, but into the same circuit, the circuit depicted in Figure 2 is appended.
   - The network validators will execute the smart contract, which verifies the proof. Upon success, the NFT note will be forwarded, and the license nullifier $\mathsf{nullifier_{lic}}$ will be added to the Merkle tree of nullifiers.

6. (**SP**) $\mathsf{get\_note\_session\_cookie}$: Receive a note containing the session cookie sc.

7. (**user**) $\mathsf{req\_service}$: Request the service to the SP, establishing communication using a secure channel, and providing the tuple $(\mathsf{tx\_hash}, \mathsf{pk}_{\mathsf{SP}}, \mathsf{attr}, c, \mathsf{sc})$.

8. (**SP**) $\mathsf{grant\_service}$: Grant or deny the service upon verification of the following steps:

   - Check whether or not the values $(\mathsf{attr}, \mathsf{pk}_{\mathsf{SP}}, c)$ are correct.
   - Check whether or not the openings $((\mathsf{pk}_{\mathsf{SP}}, \mathsf{s}_0), (\mathsf{attr}, \mathsf{s}_1), (c, \mathsf{s}_2))$ match the commitments $\mathsf{com}_0^{hash}, \mathsf{com}_1, \mathsf{com}_2$ found in the transaction $\mathsf{tx\_hash}$.

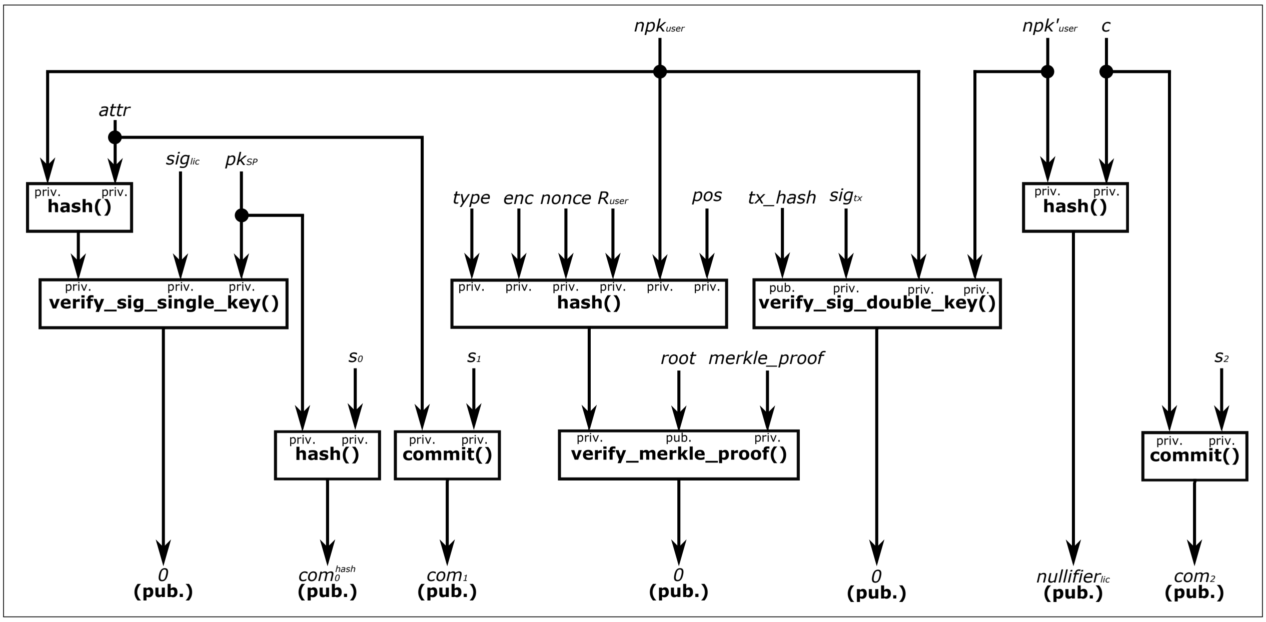

Figure 2: Arithmetic circuit for proving a license's ownership.

Furthermore, the SP might request the user to nullify the license they are using (i.e. this is a single-use license, like entering a concert). This is done through the computation of $\mathsf{nullifier}_{\mathsf{lic}}$. The deployment of this part of the circuit has two different possibilities:

- If we set $c = 0$ (or directly remove this input from the circuit), the license will be able to be used only once.

- If the SP requests the user to set a custom value for c (e.g. the date of an event), the license will be able to be reused only under certain conditions.