

Citadel Protocol Specification

Dusk Network

March 20, 2023

Contents

1	General Overview	2
1.1	What is Citadel	2
1.2	Document Organization	2
2	Definitions	2
2.1	The Types involved	2
2.2	The Roles involved	2
3	Protocol Workflow	2

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- **Proof of Validity:** users can prove ownership of a valid license, that has not been revoked.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 Types involved

- **Request:** TBD.
- **License:** TBD.

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

- **Session Cookie:** TBD.

2.2 The Roles involved

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

3 Protocol Workflow

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

1. **(user) send_note_license_req** : Compute a note public key ($\text{npk}_{\text{user}}, R_{\text{user}}$) belonging to the user, using the user's own public key, and also an additional key $k_{\text{user}} = H^{\text{Poseidon}}(\text{npk}_{\text{user}}, \text{nsk}_{\text{user}})$, by computing first the user's 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 $\text{mint_nft}(\text{npk}_{\text{SP}}, R_{\text{SP}}, \text{payload}_{\text{NFT}}, k_{\text{DH}})$, whose arguments are computed as follows:
 - $(\text{npk}_{\text{SP}}, R_{\text{SP}})$ is the SP's note public key, computed through his public key pk_{SP} .
 - $\text{payload}_{\text{NFT}} = (\text{npk}_{\text{user}}, R_{\text{user}}, k_{\text{user}})$.
 - k_{DH} is computed using the SP's public key.

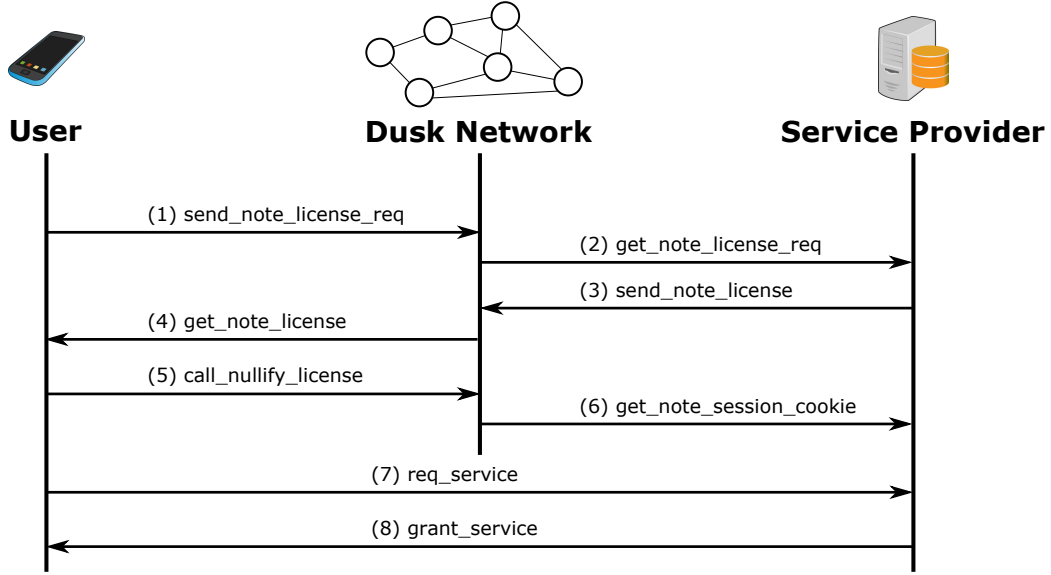


Figure 1: Overview of the protocol messages exchanged between the user, the Dusk Network, and the SP.

2. **(SP)** `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:

$$sig_{lic} = \text{sign_single_key}_{sk_{SP}}(npk_{user}, attr)$$

3. **(SP)** `send_note_license` : Set the $\text{payload}_{NFT} = \{sig_{lic}, attr\}$, and send the license to the user using the function $\text{mint_nft}(npk_{user}, R_{user}, \text{payload}_{NFT}, k_{user})$.

4. **(user)** `get_note_license` : Receive the note containing the license.

5. **(user)** `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 $sc = (s_0, s_1, s_2) \leftarrow \mathbb{F}_t$.
- The user creates a new NFT note where $\text{payload}_{NFT} = sc$, 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 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 nullifier_{lic} will be added to the Merkle tree of nullifiers.

6. **(SP)** `get_note_session_cookie` : Receive a note containing the session cookie sc .

7. **(user)** `req_service` : Request the service to the SP, establishing communication using a secure channel, and providing the tuple $(\text{tx_hash}, pk_{SP}, attr, c, sc)$.

8. **(SP)** `grant_service` : Grant or deny the service upon verification of the following steps:

- Check whether or not the values $(attr, pk_{SP}, c)$ are correct.
- Check whether or not the openings $((pk_{SP}, s_0), (attr, s_1), (c, s_2))$ match the commitments $\text{com}_0^{hash}, \text{com}_1, \text{com}_2$ found in the transaction tx_hash .

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 nullifier_{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.

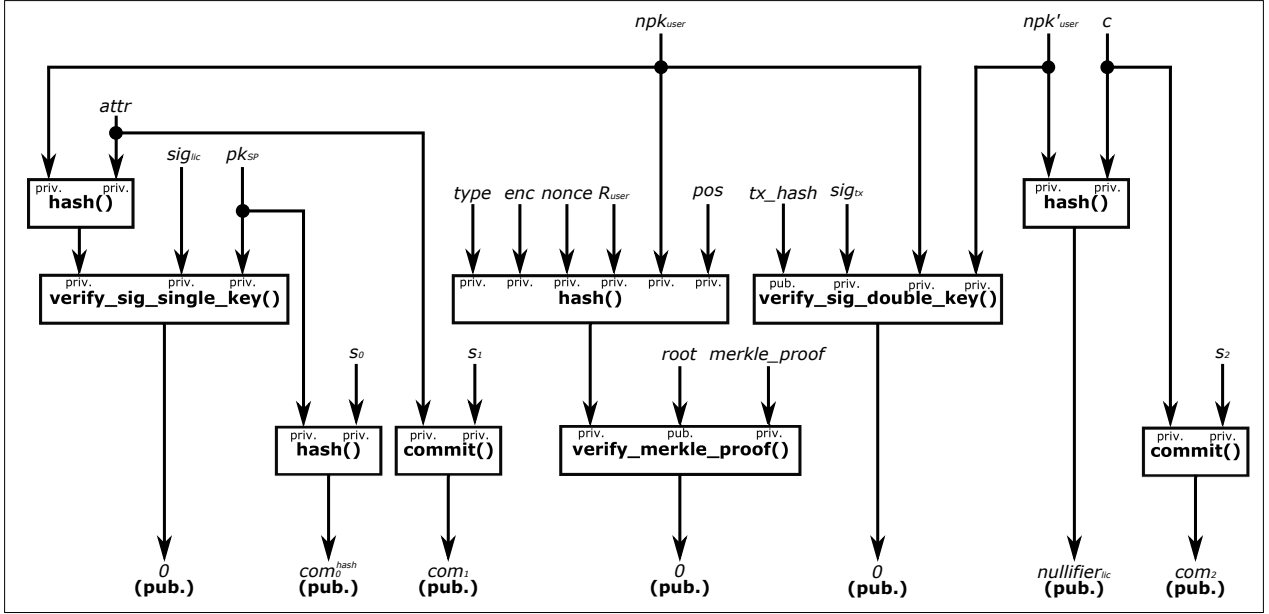


Figure 2: Arithmetic circuit for proving a license's ownership.