# Citadel Protocol Specification

Dusk Network

May 2, 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 built 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:** users can prove ownership of a license that allows them to use a given service.
- **Proof of Validity:** users can prove that a license has not been revoked and hence, it is a valid license.
- **Unlinkability:** activities from the same user in the network cannot be linked to each other by other parties.
- **Decentralized Session Opening:** when users use a license to open a session to use a service, everyone in the network learns that this happened, so it cannot be used again.
- **Attribute Blinding:** users can decide what information they want to share, hiding 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:** an entity that interacts with the wallet to request licenses and prove ownership of those.
- **License Provider (LP):** an entity that receives requests for licenses, and upon acceptance, issues them.
- **Service Provider (SP):** the entity that provides a service upon verification that a service request is correct. The SP may be the same as the LP entity or a different one.

## 2.2 The Elements Involved

- **Request:** a request includes the encryption of a stealth address belonging to the user, where the license has to be sent to, and a symmetric key. The structure is as follows:

| Element | Type | Info. |
|---|---|---|
| $(\mathsf{rpk}, R_{\mathsf{req}})$ | StealthAddress | It is a request stealth address of the LP. |
| $\mathsf{enc}$ | PoseidonCipher[6] | It is the encryption of a license stealth address of the user and a symmetric key. |
| $\mathsf{nonce}$ | BlsScalar | Randomness needed to compute $\mathsf{enc}$. |

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

| Element | Type | Info. |
|---|---|---|
| $(\mathsf{lpk}, R_{\mathsf{lic}})$ | StealthAddress | It is a license stealth address of the user. |
| $\mathsf{enc}$ | PoseidonCipher[4] | It is the encryption of some user attributes and its signature. |
| $\mathsf{nonce}$ | BlsScalar | Randomness needed to compute $\mathsf{enc}$. |
| $\mathsf{pos}$ | BlsScalar | It is the position of the license in the Merkle tree of licenses. |

- **LicenseProverParameters:** a prover needs some auxiliary parameters to compute the proof that proves the ownership of a license. Some of the items of this table are related to the session and session cookie elements. The structure is as follows:

| Element | Type | Info. |
|---|---|---|
| $\mathsf{lpk}$ | JubJubAffine | The license public key of the user. |
| $\mathsf{lpk}'$ | JubJubAffine | A variation of the license public key of the user computed with a different generator. |
| $\mathsf{sig}_{\mathsf{lic}}$ | Signature | The signature of the license attributes. |
| $\mathsf{com}_0^{hash}$ | BlsScalar | A hash of the public key of the LP. |
| $\mathsf{com}_1$ | JubJubExtended | A Pedersen Commitment of the attributes. |
| $\mathsf{com}_2$ | JubJubExtended | A Pedersen Commitment of the $c$ value. |
| session_hash | BlsScalar | The hash of the public key of the SP together with some randomness. |
| sig_session_hash | dusk_schnorr::Proof | The signature of the session hash signed by the user. |
| merkle_proof | PoseidonBranch | Membership proof of the license in the Merkle tree of licenses. |

- **Session:** a session is a public struct known by all the validators. The structure is as follows:

| Element | Type | Info. |
|---|---|---|
| session_hash | BlsScalar | The hash of the public key of the SP together with some randomness. |
| session_id | BlsScalar | The id of a session open using a given license. |
| $\mathsf{com}_0^{hash}$ | BlsScalar | A hash of the public key of the LP. |
| $\mathsf{com}_1$ | JubJubExtended | A Pedersen Commitment of the attributes. |
| $\mathsf{com}_2$ | JubJubExtended | A Pedersen Commitment of the $c$ value. |

- **SessionCookie:** a session cookie is a secret value 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:

| Element | Type | Info. |
|---|---|---|
| $\mathsf{pk}_{\mathsf{SP}}$ | JubJubAffine | The public key of the SP. |
| $r_{\mathsf{session}}$ | BlsScalar | Randomness for computing the session hash. |
| session_id | BlsScalar | The id of a session open using a given license. |
| $\mathsf{pk}_{\mathsf{LP}}$ | JubJubAffine | The public key of the LP. |
| attr | JubJubScalar | The attributes of the user. |
| $c$ | JubJubScalar | The challenge value. |
| $\mathsf{s}_0$ | JubJubScalar | Randomness used to compute the $\mathsf{com}_0^{hash}$. |
| $\mathsf{s}_1$ | BlsScalar | Randomness used to compute the $\mathsf{com}_1$. |
| $\mathsf{s}_2$ | BlsScalar | Randomness used to compute the $\mathsf{com}_2$. |

# 3 Protocol Workflow

In Citadel, each party involved in the protocol keeps static keys, as we detail now. Let $G, G' \leftarrow \mathbb{J}$ be two generators for the subgroup $\mathbb{J}$ of order $t$ of the Jubjub. The keys of each party as the following.

- *Secret key:* $\mathsf{sk} = (a, b)$, where $a, b \leftarrow \mathbb{F}_t$.
- *Public key:* $\mathsf{pk} = (A, B)$, where $A = aG$ and $B = bG$.

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

1. (**user**) request_license : compute a license stealth address $(\mathsf{lpk}, R_{\mathsf{lic}})$ belonging to the user, using the user's own public key, as follows.

   (a) Sample $r$ uniformly at random from $\mathbb{F}_t$.

   (b) Compute a symmetric Diffie–Hellman key $\mathsf{k} = rA_{\mathsf{user}}$.

   (c) Compute a one-time public key $\mathsf{lpk} = H^{\mathsf{BLAKE2b}}(\mathsf{k})G + B_{\mathsf{user}}$.

   (d) Compute $R_{\mathsf{lic}} = rG$.

   Compute also an additional key $\mathsf{k}_{\mathsf{lic}} = H^{\mathsf{BLAKE2b}}(\mathsf{lsk})G$, by computing first the license secret key $\mathsf{lsk} = H^{\mathsf{BLAKE2b}}(\mathsf{k}) + b_{\mathsf{user}}$. Then, compute the request stealth address $(\mathsf{rpk}, R_{\mathsf{req}})$ using the LP's public key, as follows.

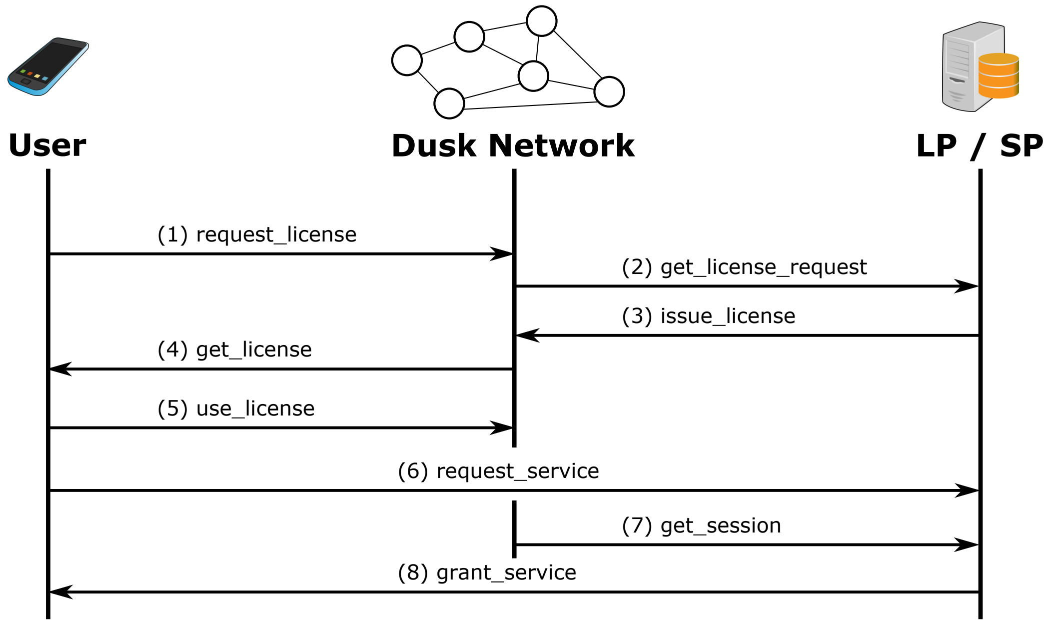


Figure 1: Overview of the protocol messages exchanged between the user, the Dusk Network, and the SP.

(a) Sample $r$ uniformly at random from $\mathbb{F}_t$.

(b) Compute a symmetric Diffie–Hellman key $\mathsf{k}_{\mathsf{req}} = rA_{\mathsf{LP}}$.

(c) Compute a one-time public key $\mathsf{rpk} = H^{\mathsf{BLAKE2b}}(\mathsf{k}_{\mathsf{req}})G + B_{\mathsf{LP}}$.

(d) Compute $R_{\mathsf{req}} = rG$.

And finally send the following request to the network:

$$\mathsf{req} = ((\mathsf{rpk}, R_{\mathsf{req}}), \mathsf{enc}, \mathsf{nonce}),$$

where

$$\mathsf{enc} = \mathsf{Enc}_{\mathsf{k}_{\mathsf{req}}}((\mathsf{lpk}, R_{\mathsf{lic}})||\mathsf{k}_{\mathsf{lic}}; \mathsf{nonce}).$$

2. (**LP**) get_license_request : continuously check the network for incoming license requests, by checking if $\mathsf{rpk} \stackrel{?}{=} H^{\mathsf{BLAKE2b}}(\tilde{\mathsf{k}}_{\mathsf{req}})G + B_{\mathsf{LP}}$, where $\tilde{\mathsf{k}}_{\mathsf{req}} = a_{\mathsf{LP}}R_{\mathsf{req}}$.

3. (**LP**) issue_license : upon receiving a request from a user, define a set of attributes attr representing the license, and compute a digital signature as follows:

$$\mathsf{sig}_{\mathsf{lic}} = \mathsf{sign\_single\_key}_{\mathsf{sk}_{\mathsf{SP}}}(\mathsf{lpk}, \mathsf{attr})$$

.

Then, send the following license to the network:

$$\mathsf{lic} = ((\mathsf{lpk}, R_{\mathsf{lic}}), \mathsf{enc}, \mathsf{nonce}, \mathsf{pos}),$$

where

$$\mathsf{enc} = \mathsf{Enc}_{\mathsf{k}_{\mathsf{lic}}}(\mathsf{sig}_{\mathsf{lic}}||\mathsf{attr}; \mathsf{nonce}).$$

4. (**user**) get_license : receive the license by scanning the incoming transactions, and checking if $\mathsf{lpk} \stackrel{?}{=} H^{\mathsf{BLAKE2b}}(\tilde{\mathsf{k}}_{\mathsf{lic}})G + B_{\mathsf{user}}$, where $\tilde{\mathsf{k}}_{\mathsf{lic}} = H^{\mathsf{BLAKE2b}}(\mathsf{lsk})G$.

5. (**user**) use_license : when using the license, open a session with a specific SP by executing a call to the license contract. The following steps are performed:

- The user issues a transaction that calls the license contract, which includes a ZKP that is computed out of the gadget depicted in Figure 2.
- The network validators will execute the smart contract, which verifies the proof. Upon success, the following session will be added to a shared list of sessions:

$$\mathsf{session} = \{\mathsf{session\_hash}, \mathsf{session\_id}, \mathsf{com}_0^{hash}, \mathsf{com}_1, \mathsf{com}_2\},$$

where $\mathsf{session\_hash} = H^{\mathsf{Poseidon}}(\mathsf{pk_{SP}}||r_{\mathsf{session}})$, and $r_{\mathsf{session}}$ is sampled uniformly at random from $\mathbb{F}_t$.

6. (**user**) request_service : request the service to the SP, establishing communication using a secure channel, and providing the session cookie that follows.

$$\mathsf{sc} = \{\mathsf{pk_{SP}}, r_{\mathsf{session}}, \mathsf{session\_id}, \mathsf{pk_{LP}}, \mathsf{attr}, c, \mathsf{s}_0, \mathsf{s}_1, \mathsf{s}_2\}$$

7. (**SSP**) get_session : receive a session from the list of sessions, where session.session_id = sc.session_id.
8. (**SSP**) grant_service : grant or deny the service upon verification of the following steps:
   - Check whether the values $(\mathsf{attr}, \mathsf{pk_{LP}}, c)$ included in the sc are correct.
   - Check whether the opening $(\mathsf{pk_{SP}}, r_{\mathsf{session}})$ included in the sc matches the session_hash found in the session.
   - Check whether the openings $((\mathsf{pk_{LP}}, \mathsf{s}_0), (\mathsf{attr}, \mathsf{s}_1), (c, \mathsf{s}_2))$ included in the sc match the commitments $(\mathsf{com}_0^{hash}, \mathsf{com}_1, \mathsf{com}_2)$ found in the session.

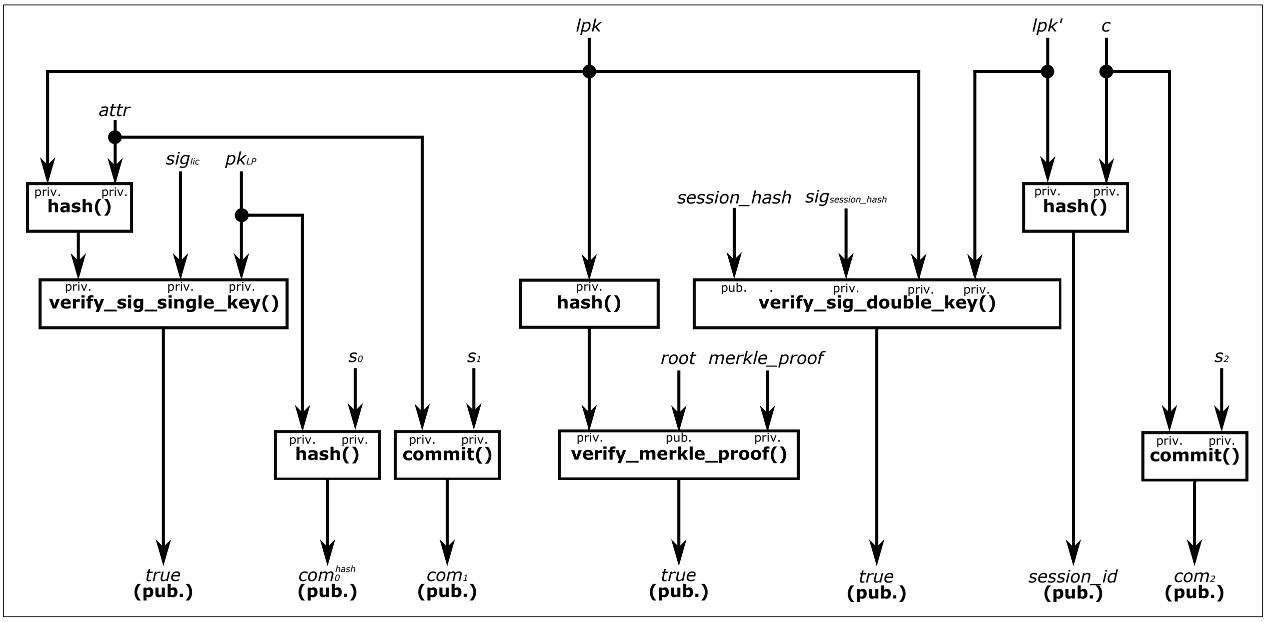


Figure 2: Arithmetic circuit for proving a license's ownership.

Furthermore, the SP might want to prevent the user from using the license more than once (e.g. this is a single-use license, like entering a concert). This is done through the computation of session_id. 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 can 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 can be reused only under certain conditions.