

# Citadel Protocol Specification

Dusk Network

April 19, 2023

## Contents

<b>1</b>	<b>General Overview</b>	<b>2</b>
1.1	What is Citadel . . . . .	2
1.2	Document Organization . . . . .	2
<b>2</b>	<b>Definitions</b>	<b>2</b>
2.1	The Roles Involved . . . . .	2
2.2	The Elements Involved . . . . .	2
<b>3</b>	<b>Protocol Workflow</b>	<b>3</b>

# 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- **Proof of Validity:** users can prove that a license has not been revoked and hence, it is a valid license.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:** no one can link any activity with other activities done in the network.
- **Decentralized Nullification:** when users use a license (i.e. they prove its ownership 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.
- **Service Provider (SP):** an entity offering an off-chain service that receives requests for licenses, and upon acceptance, issues them.
- **Session Service Provider (SSP):** the entity that provides the service upon verification that a service request is correct. The SSP may be the same as the SP 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.
$(rp_k, R)$ enc nonce	StealthAddress PoseidonCipher BlsScalar	It is a request stealth address of the SP. It is a ciphertext of size 6. Randomness needed to compute 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.
$(lp_k, R)$ enc nonce pos	StealthAddress PoseidonCipher BlsScalar BlsScalar	It is a license stealth address of the user. It is a ciphertext of size 4. Randomness needed to compute enc. 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 nullifies a license when used. Some of the items of this table are related to the session and session cookie elements. The structure is as follows:

Element	Type	Info.
lpk	JubJubAffine	The license public key.
lpk'	JubJubAffine	A variation of the license public key computed with a different generator.
sig <sub>lic</sub>	Signature	The signature of the license.
com <sub>0</sub> <sup>hash</sup>	BlsScalar	A hash commitment of the public key of the SP.
com <sub>1</sub>	JubJubExtended	A Pedersen Commitment of the attributes.
com <sub>2</sub>	JubJubExtended	A Pedersen Commitment of the $c$ value.
session_hash	BlsScalar	The hash of the public key of the SSP 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 SSP together with some randomness.
nullifier <sub>lic</sub>	BlsScalar	The nullifier of a given license.
com <sub>0</sub> <sup>hash</sup>	BlsScalar	A hash commitment of the public key of the SP.
com <sub>1</sub>	JubJubExtended	A Pedersen Commitment of the attributes.
com <sub>2</sub>	JubJubExtended	A Pedersen Commitment of the $c$ value.

- **SessionCookie:** a session cookie is a secret value known only by the user and the SSP. It contains a set of openings to a given set of commitments. The structure is as follows:

Element	Type	Info.
pk <sub>SSP</sub>	JubJubAffine	The public key of the SSP.
$r$	BlsScalar	Randomness for computing the session hash.
nullifier <sub>lic</sub>	BlsScalar	The nullifier of a given license.
pk <sub>SP</sub>	JubJubAffine	The public key of the SP.
attr	JubJubScalar	The attributes of the user.
$c$	JubJubScalar	The challenge value.
$s_0$	JubJubScalar	Randomness used to compute the com <sub>0</sub> <sup>hash</sup> .
$s_1$	BlsScalar	Randomness used to compute the com <sub>1</sub> .
$s_2$	BlsScalar	Randomness used to compute the com <sub>2</sub> .

### 3 Protocol Workflow

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

1. **(user)** send\_license\_req : compute a license stealth address (lpk,  $R$ ) belonging to the user, using the user's own public key, and also an additional key  $k_{lic} = H^{BLAKE2b}(lsk)G$ , by computing first the user's lsk. Then, compute the request stealth address (rp<sub>k</sub>,  $R$ ) and  $k_{DH}$  using the SP's public key. And finally send the following request to the network:

$$req = ((rp_k, R), enc, nonce)$$

,

where

$$enc = Enc_{k_{DH}}((lpk, R) || k_{lic}; nonce)$$

.

2. **(SP)** get\_license\_req : continuously check the network for incoming license requests.
3. **(SP)** send\_license : upon receiving a request from a user, define a set of attributes attr representing the license, and compute a digital signature as follows:

$$sig_{lic} = sign\_single\_key_{sk_{SP}}(lpk, attr)$$

.

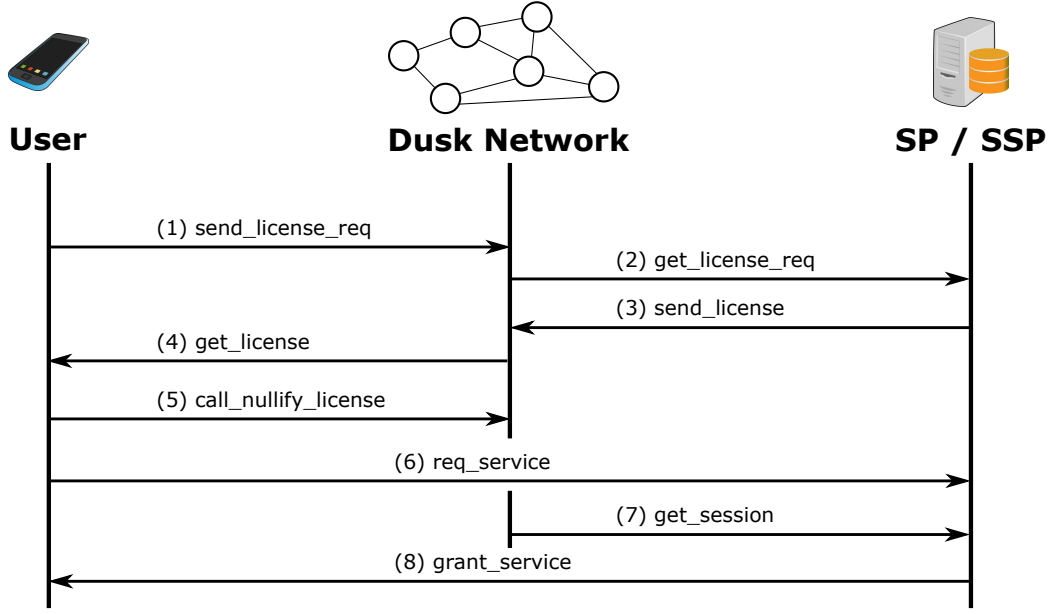


Figure 1: Overview of the protocol messages exchanged between the user, the Dusk Network, and the SP.

Then, send the following license to the network:

$$\text{lic} = ((\text{lpk}, R), \text{enc}, \text{nonce}, \text{pos})$$

,  
where

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

4. **(user)** `get_license` : receive the license by scanning the incoming transactions.
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 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:

$$\text{session} = \{\text{session\_hash}, \text{nullifier}_{\text{lic}}, \text{com}_0^{\text{hash}}, \text{com}_1, \text{com}_2\}$$

,  
where  $\text{session\_hash} = H^{\text{Poseidon}}(\text{pk}_{\text{SSP}} || r)$ .

6. **(user)** `req_service` : request the service to the SSP, establishing communication using a secure channel, and providing the `sc`.
7. **(SSP)** `get_session` : receive a `session` from the list of sessions, where  $\text{session.nullifier}_{\text{lic}} = \text{sc.nullifier}_{\text{lic}}$ .
8. **(SSP)** `grant_service` : grant or deny the service upon verification of the following steps:
  - Check whether the values  $(\text{attr}, \text{pk}_{\text{SP}}, c)$  included in the `sc` are correct.
  - Check whether the opening  $(\text{pk}_{\text{SSP}}, r)$  included in the `sc` matches the `session_hash` found in the `session`.
  - Check whether the openings  $((\text{pk}_{\text{SP}}, s_0), (\text{attr}, s_1), (c, s_2))$  included in the `sc` match the commitments  $(\text{com}_0^{\text{hash}}, \text{com}_1, \text{com}_2)$  found in the `session`.

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

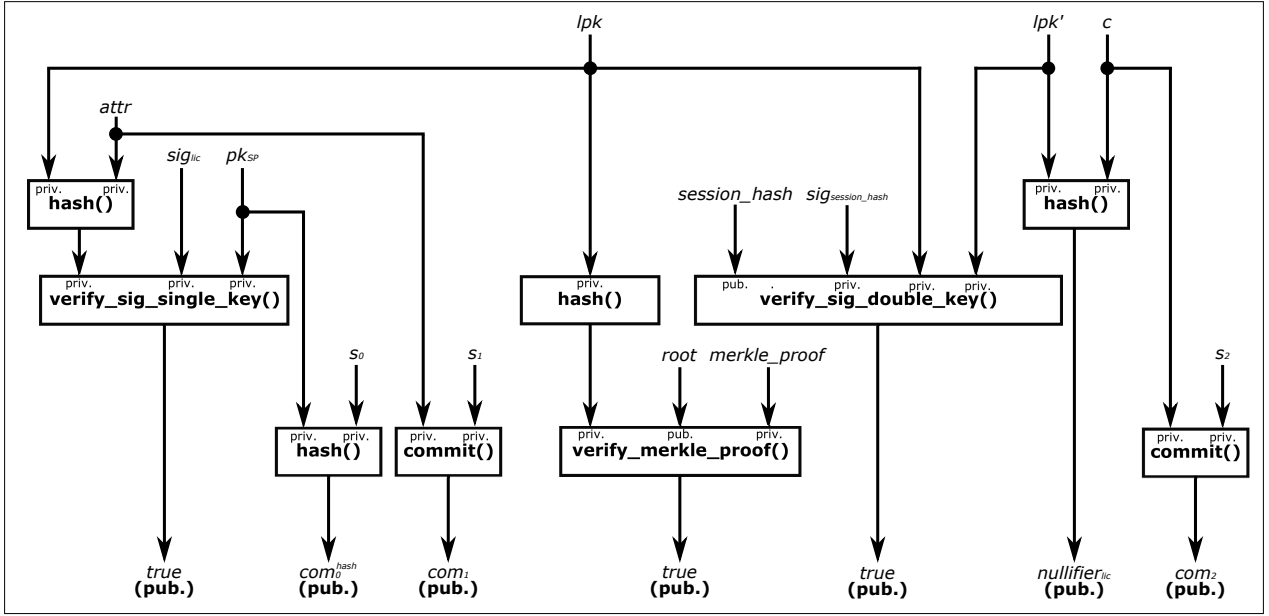


Figure 2: Arithmetic circuit for proving a license's ownership.

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