Citadel Protocol Specification

Dusk Network

April 11, 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- **Proof of Validity:** users can prove ownership of a valid license, that has not been revoked.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:** 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):** It provides the service upon verification that a service request is correct. It can be the same individual that the SP or another 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.
(rpk, R)	StealthAddress	It is a request stealth address of the SP.
enc	PoseidonCipher	It is a ciphertext of size 6.
nonce	BlsScalar	It is a 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.
(lpk, R)	StealthAddress	It is a license stealth address of the user.
enc	PoseidonCipher	It is a ciphertext of size 4.
nonce	BlsScalar	It is a randomness needed to compute enc.
pos	BlsScalar	It is the position of the license into a Merkle tree of licenses.

- **LicenseProverParameters:** A prover needs some auxiliary parameters to compute the proof that nullifies a license when desired to be spent. The structure is as follows:

Element	Type	Info.
lpk	JubJubAffine	The license public key.
lpk'	JubJubAffine	The license public key prime.
$\mathsf{sig}_{\mathsf{lic}}$	Signature	The signature of the license.
com_0^{hash}	BlsScalar	A hash commitment of the public key of the SP.
com_1	JubJubExtended	A Pedersen Commitment of the attributes.
com_2	JubJubExtended	A Pedersen Commitment of the c value.
session_hash	BlsScalar	The hash of the public key of the SSP plus some randomness.
sig_session_hash	dusk_schnorr::Proof	The signature of the session hash.
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 plus some randomness.
$\mathsf{nullifier}_{\mathsf{lic}}$	BlsScalar	The nullifier of a given license.
com_0^{hash}	BlsScalar	A hash commitment of the public key of the SP.
com_1	JubJubExtended	A Pedersen Commitment of the attributes.
com_2	JubJubExtended	A Pedersen Commitment of the c value.

- **Session Cookie:** 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.
$\mathsf{pk}_{\mathsf{SSP}}$	JubJubAffine	The public key of the SSP.
r	BlsScalar	Randomness for computing the session hash.
$\mathsf{nullifier}_{\mathsf{lic}}$	BlsScalar	The nullifier of a given license.
$\mathsf{pk}_{\mathsf{SP}}$	JubJubAffine	The public key of the SP.
attr	JubJubScalar	The attributes of the user.
c	JubJubScalar	The challenge value.
s_0	JubJubScalar	The randomness used to compute the commitment 0.
s_1	BlsScalar	The randomness used to compute the commitment 1.
s_2	BlsScalar	The randomness used to compute the commitment 2.

3 Protocol Workflow

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

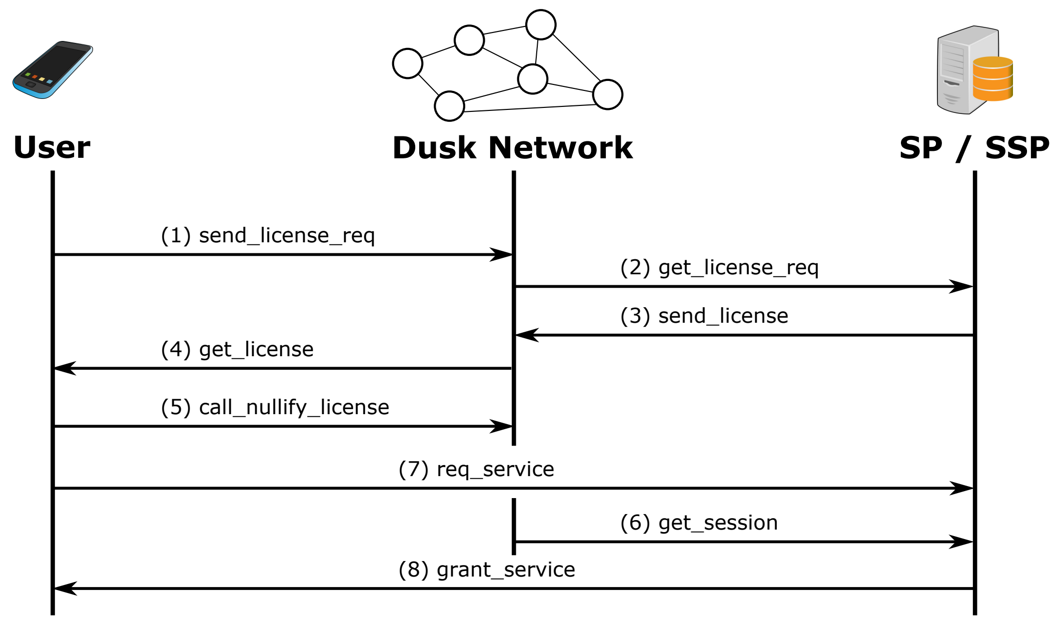

Figure 1: Overview of the protocol messages exchanged between the user, the Dusk Network, and the SP.

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

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

where

$$\mathsf{enc} = \mathsf{Enc}_{\mathsf{k_{DH}}}((\mathsf{lpk}, R)||\mathsf{k_{lic}}; \mathsf{nonce})$$

2. (**SP**) $\mathsf{get\_license\_req}$: Continuously check the network for incoming license requests.

3. (**SP**) $\mathsf{send\_license}$: Upon receiving the request from a user, define a set of attributes attr representing the license, and compute a digital signature as follows:

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

Then, send the following license to the network:

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

where

$$\mathsf{enc} = \mathsf{Enc}_{\mathsf{k_{lic}}}(\mathsf{sig_{lic}}||\mathsf{attr}; \mathsf{nonce})$$

4. (**user**) $\mathsf{get\_license}$: Receive the license by scanning the incoming transactions.

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 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:

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

   where $\mathsf{session\_hash} = H^{\mathsf{Poseidon}}(\mathsf{pk_{SSP}}||\mathsf{r})$.

6. (**user**) $\mathsf{req\_service}$: Request the service to the SSP, establishing communication using a secure channel, and providing the sc.

7. (**SSP**) $\mathsf{get\_session}$: Receive a $\mathsf{session}$ from the list of sessions, where $\mathsf{session.nullifier_{lic}} = \mathsf{sc.nullifier_{lic}}$.

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

   - Check whether or not the values $(\mathsf{attr}, \mathsf{pk_{SP}}, c)$ included in the sc are correct.
   - Check whether or not the opening $(\mathsf{pk_{SSP}}, \mathsf{r})$ included in the sc match the $\mathsf{session\_hash}$ found in the $\mathsf{session}$.
   - Check whether or not the openings $((\mathsf{pk_{SP}}, \mathsf{s_0}), (\mathsf{attr}, \mathsf{s_1}), (c, \mathsf{s_2}))$ included in the sc match the commitments $\mathsf{com}_0^{hash}, \mathsf{com}_1, \mathsf{com}_2$ found in the $\mathsf{session}$.

Furthermore, the SSP 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 $\mathsf{nullifier_{lic}}$. 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 SSP 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.

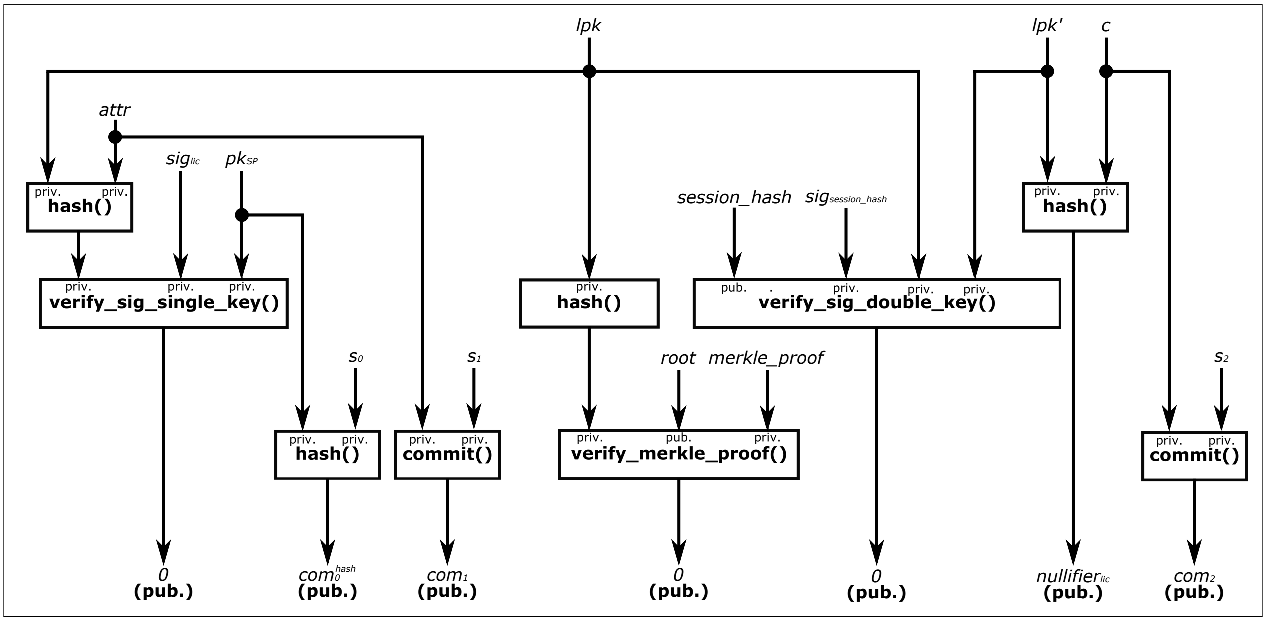

Figure 2: Arithmetic circuit for proving a license's ownership.