Citadel protocol specification

Dusk Network

November 2, 2023

Contents

1 Protocol overview

Citadel is a self-sovereign identity (SSI) protocol built on tope of Dusk that allows users of a given service to manage their digital identities in a fully transparent manner. More specifically, every user can know which information about them is shared with other parties, and accept or deny any request for personal information.

1.1 Properties

With Citadel, users of a service can request licenses that represent their *right* to use such a service. Citadel satisfies the following properties:

- *Proof of ownership*: users can prove that they own a valid license that allows them to use a certain service.
- *Proof of validity*: users with a valid license can prove that their license has not been revoked and is valid.
- *Unlinkability*: different services used by a same user cannot be linked from one another.
- *Decentralized session opening*: when users start using a service, the network learns that this happened and the license used to access to the service cannot be used again.
- *Attribute blinding*: users have the power to decide exactly what information they want to share and with whom.

1.2 The parties involved

Citadel involves three (potentially different) parties:

- The *user* is the person who interacts with the wallet and requests licenses in order to claim their right to make use of services.
- The *service provider* (SP) is the entity that offers a service to users. Upon verification that a service request from a user is correct, it provides such service.
- The *license provider* (LP) is the entity that receives requests for licenses from users, and upon acceptance, issues them. The LP can be the same SP entity or a different one.

1.3 Protocol flow [Missing explanation]

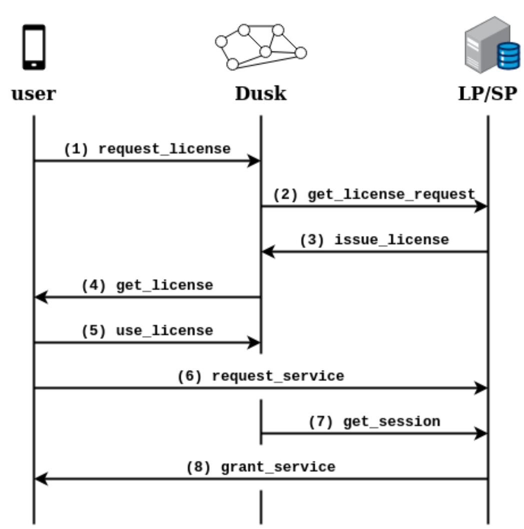

Figure 1: Overview of the protocol messages exchanged between the user, Dusk's network, and the LP/SP.

2 Building blocks

In this section, we present the static keys associated each party involved in the protocol, and also the structure of the elements involved.

Marta: I suggest to include a section with the details of the cryptographic elements included in this section (the jubjub group, the generators, etc.).

2.1 Cryptographic primitives

Marta: Hashing - it is going to be Poseidon everywhere.

2.2 Keys

Let $G, G' \leftarrow \mathbb{J}$ be two generators for the subgroup $\mathbb{J}$ of order t of the Jubjub elliptic curve. In Citadel, each party involved in the protocol holds a pair of static keys with the following structure:

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

We use the subindices $\mathsf{user}, \mathsf{SP}, \mathsf{LP}$ to indicate the owner of the keys, e.g. $\mathsf{pk}_{\mathsf{user}}$.

2.3 Elements involved

Marta: **Should this section be moved to 4. Implementation details?**

Here we describe the elements involved in Citadel. How they are used in the protocol is described in Section 3.

- *Request*: the structure of a request includes the encryption of a stealth address belonging to the user and where the license has to be sent to, and a symmetric key shared between the user and the LP.

Element	Type	Description
$(\mathsf{rpk}, R_{\mathsf{req}})$	StealthAddress	Stealth address for the LP.
enc	PoseidonCipher[6]	Encryption of a user's stealth address where the license has to be sent to and a symmetric key.
nonce	BlsScalar	Randomness needed to compute enc.

- *License*: asset that represents the right of a user to use a given service. A license has the following structure:

Element	Type	Description
$(\mathsf{lpk}, R_{\mathsf{lic}})$	StealthAddress	License stealth address of the user.
enc	PoseidonCipher[4]	Encryption of user attributes and signature of these attributes.
nonce	BlsScalar	Randomness needed to compute enc.
pos	BlsScalar	Position of the license in the Merkle tree of licenses.

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

  Marta: The session cookie is not a secret value, it is an struct. Does it refer to `session_id`?

  Marta: Clarify what the element `attr` is - is it a hash, an array?

Element	Type	Description
$\mathsf{pk}_{\mathsf{SP}}$	JubJubAffine	Public key of the SP.
r_{session}	BlsScalar	Randomness for computing the session hash.
session_id	BlsScalar	ID of a session opened using a license.
$\mathsf{pk}_{\mathsf{LP}}$	JubJubAffine	Public key of the LP.
attr	JubJubScalar	Attributes of the user.
c	JubJubScalar	Challenge value.
s_0	JubJubScalar	Randomness used to compute com_0^{hash}.
s_1	BlsScalar	Randomness used to compute com_1.
s_2	BlsScalar	Randomness used to compute com_2.

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

Marta: What does *validators* mean?

Marta: TODO - when we say *together with some randomness*, I would include the name of the random variable.

Marta: In com_0^{hash}, does the commitment also include some randomness?

Element	Type	Description
session_hash	BlsScalar	Hash of the SP's public key together with some randomness.
session_id	BlsScalar	ID of a session opened using a given license.
com_0^{hash}	BlsScalar	Hash of the public key of the LP.
com_1	JubJubExtended	Pedersen commitment of the attributes.
com_2	JubJubExtended	Pedersen commitment of the c value.

- *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:

Marta: TODO - when we say *together with some randomness*, I would include the name of the random variable.

Element	Type	Description
lpk	JubJubAffine	License public key of the user.
lpk′	JubJubAffine	A variation of the license public key of the user computed with a different generator.
$\mathsf{sig}_{\mathsf{lic}}$	Signature	Signature of the license attributes.
com_0^{hash}	BlsScalar	Hash of the LP's public key.
com_1	JubJubExtended	Pedersen commitment of the attributes.
com_2	JubJubExtended	Pedersen commitment of the c value.
session_hash	BlsScalar	Hash of the public key of the SP together with some randomness.
sig_session_hash	dusk_schnorr::Proof	Signature of the session hash signed by the user.
merkle_proof	PoseidonBranch	Membership proof of the license in the Merkle tree of licenses.

Marta: Add a section including the software that it is assumed each participant uses? For example, user makes use of wallet and does blockchain calls and queries. The blockchain stores a license contract that can be called blabla. The LP and SP software, etc. (see previous section 4.1 from Milosz).

3 Protocol

Marta: Change BLAKE2 to Poseidon and leave a footnote.

In this section, we describe the workflow of Citadel in detail.

1. (**user**) $\mathsf{request\_license}()$
   1.1. Compute a license stealth address $(\mathsf{lpk}, R_{\mathsf{lic}})$ belonging to the user, using the user's own public key, as follows.
      i. Sample r uniformly at random from $\mathbb{F}_t$.
      ii. Compute a symmetric Diffie–Hellman key $\mathsf{k} = rA_{\mathsf{user}}$.
      iii. Compute a one-time public key $\mathsf{lpk} = H^{\mathsf{BLAKE2b}}(\mathsf{k})G + B_{\mathsf{user}}$.
      iv. Compute $R_{\mathsf{lic}} = rG$.
   1.2. Compute the license secret key $\mathsf{lsk} = H^{\mathsf{BLAKE2b}}(\mathsf{k}) + b_{\mathsf{user}}$ and an additional key $\mathsf{k}_{\mathsf{lic}} = H^{\mathsf{Poseidon}}(\mathsf{lsk})G$.
   1.3. Compute the request stealth address $(\mathsf{rpk}, R_{\mathsf{req}})$ using the LP's public key, as follows.

      Marta: Consider using different letter instead of r...

      i. Sample r uniformly at random from $\mathbb{F}_t$.
      ii. Compute a symmetric Diffie–Hellman key $\mathsf{k}_{\mathsf{req}} = rA_{\mathsf{LP}}$.
      iii. Compute a one-time public key $\mathsf{rpk} = H^{\mathsf{BLAKE2b}}(\mathsf{k}_{\mathsf{req}})G + B_{\mathsf{LP}}$.
      iv. Compute $R_{\mathsf{req}} = rG$.
   1.4. Encrypt data using the key $\mathsf{k}_{\mathsf{req}}$: $\mathsf{enc} = \mathsf{Enc}_{\mathsf{k}_{\mathsf{req}}}((\mathsf{lpk}, R_{\mathsf{lic}})||\mathsf{k}_{\mathsf{lic}}; \mathsf{nonce})$.

      Marta: Include how the nonce is computed, if it is a random value as well.

   1.5. Send the following request to the network: $\mathsf{req} = ((\mathsf{rpk}, R_{\mathsf{req}}), \mathsf{enc}, \mathsf{nonce})$.
2. (**LP**) $\mathsf{get\_license\_request}()$

   The LP checks continuously the network to detect any incoming license requests addressed to them:
   2.1. Compute $\tilde{\mathsf{k}}_{\mathsf{req}} = a_{\mathsf{LP}} R_{\mathsf{req}}$.
   2.2. Check if $\mathsf{rpk} \stackrel{?}{=} H^{\mathsf{BLAKE2b}}(\tilde{\mathsf{k}}_{\mathsf{req}})G + B_{\mathsf{LP}}$.

   Marta: Include that if this is the case, the LP should decrypt the encrypted information to retrieve $\mathsf{lpk}, R_{\mathsf{lic}}, \mathsf{k}_{\mathsf{lic}}$.

   Marta: Is this done in $\mathsf{get\_license\_request}()$ or in next step?

3. (**LP**) $\mathsf{issue\_license}()$
   3.1. Upon receiving a request from a user, define a set of attributes attr associated to the license, and compute a digital signature as follows:
   $$\mathsf{sig}_{\mathsf{lic}} = \mathsf{sign\_single\_key}_{\mathsf{sk}_{\mathsf{SP}}}(\mathsf{lpk}, \mathsf{attr}).$$
   3.2. Encrypt the signature and the attributes using the license key:
   $$\mathsf{enc} = \mathsf{Enc}_{\mathsf{k}_{\mathsf{lic}}}(\mathsf{sig}_{\mathsf{lic}}||\mathsf{attr}; \mathsf{nonce}).$$

3.3. Send the following license to the network:

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

4. (**user**) get_license()

   In order to receive the license, the user must scan all incoming transactions the following way:

   4.1. Compute $\tilde{\mathsf{k}}_{\mathsf{lic}} = H^{\mathsf{BLAKE2b}}(\mathsf{lsk})G$.

   4.2. Check if $\mathsf{lpk} \stackrel{?}{=} H^{\mathsf{BLAKE2b}}(\tilde{\mathsf{k}}_{\mathsf{lic}})G + B_{\mathsf{user}}$,

   Marta: Same as before, we should include the step in which the user decyrpts the information associated to the license.

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:

   Marta: We should mention something about the *license contract* before this step. Maybe in Section 2 where elements are presented? Add a small section about Dusk's blockchain?

   - The user issues a transaction that calls the license contract, which includes a ZKP that is computed out of the gadget depicted in Figure **??**. Notice that here, the user signs session_hash using lsk. Likewise, the user here will need to compute $\mathsf{lpk}' = \mathsf{lsk}G'$.
   - 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 $\mathsf{session\_hash} = H^{\mathsf{Poseidon}}(\mathsf{pk}_{\mathsf{SP}} || r_{\mathsf{session}})$, and r_{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.

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

   Marta: Notation-wise: the acronym sc can be confused with the common abbreviation for `smart contract (sc)`, maybe use a different acronym?

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 $(\mathsf{attr}, \mathsf{pk}_{\mathsf{LP}}, c)$ included in the sc are correct.
   - Check whether the opening $(\mathsf{pk}_{\mathsf{SP}}, r_{\mathsf{session}})$ included in the sc matches the session_hash found in the session.
   - Check whether the openings $((\mathsf{pk}_{\mathsf{LP}}, \mathsf{s}_0), (\mathsf{attr}, \mathsf{s}_1), (c, \mathsf{s}_2))$ included in the sc match the commitments $(\mathsf{com}_0^{hash}, \mathsf{com}_1, \mathsf{com}_2)$ found in the session.

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.

4 Implementation details

Marta: Add Milosz figure here or in 3. Protocol?

References

[1] Benarroch, D., Campanelli, M., Fiore, D., Gurkan, K., Kolonelos, D.: Zero-knowledge proofs for set membership: efficient, succinct, modular. In: Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I. pp. 393–414. Springer (2021)

[2] Catalano, D., Fiore, D.: Vector commitments and their applications. In: Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16. pp. 55–72. Springer (2013)

[3] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PlonK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)

[4] Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: USENIX Security Symposium. vol. 2021 (2021)

[5] Khovratovich, D.: Encryption with Poseidon Available online: https://dusk.network/uploads/Encryption-with-Poseidon.pdf (accessed on 1 June 2022)

[6] Palatinus, M., Rusnak, P., Voisine, A., Bowe, S.: BIP-39: Mnemonic code for generating deterministic keys (2013), https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

[7] Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Advances in Cryptology—CRYPTO'89 Proceedings 9. pp. 239–252. Springer (1990)