# vLEI 2025 Hackathon Workshop

## Open Loop Verification Architecture with the vLEI

New York City Hackathon Workshop

November 3rd, 2025

Author(s) – vLEI Development Team – Kent Bull, Esteban Garcia

# Agenda

1. (2-5m) Welcome and Introductions
2. (2-5m) Workshop Intro
3. (30-45m) Module 1 - vLEI OOR Permissioned Action
    1. vLEI Issue-Hold-Verify Fundamentals
    2. Verifying a vLEI Official Organizational Role (OOR) ACDC
4. (45m) Module 2 - vLEI-based smart contract access - CCID by Chainlink
    1. Turning a vLEI into a CCID
    2. vLEI-based CCID smart contract access
5. (10m) Hackathon Feedback Gathering
6. (10m-end) Hackathon Retrospective and Lessons Learned

# Introductions!

# Workshop - Module 1 vLEI OOR Permissioned Action

- Outcome: Triggering a protected action with an OOR ACDC presentation
- Step 1: Infrastructure Setup (deploy.sh)
  - Witnesses, Schema Server, KERIA, Wallet, Verifier, Resource Svr
- Step 2: Set up GEDA and QVI - delegation
- Step 3: Challenge and Response between GEDA and QVI
- Step 4: QVI credential issuance and presentation
- Step 5: LE AID setup, LE ACDC issuance and presentation
- Step 6: Person AID setup, OOR issuance and presentation, ECR issue
- Step 7: Bonus: Trigger protected action

# CLONE ME
# https://github.com/GLEIF-IT/vlei-hackathon-2025-workshop

github.com/GLEIF-IT/vlei-hackathon-2025-workshop

GLEIF-IT / **vlei-hackathon-2025-workshop**

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

**vlei-hackathon-2025-workshop** Public

Edit Pins | Watch 0

master | 1 Branch | 0 Tags | Go to file t | Add file | Code

**kentbull** add challenge and response code

| | |
|---|---|
| config | Presenting QVI cre |
| images | Add readme and in |
| sig-wallet | add challenge and |
| task-data | GEDA delegation s |
| task-scripts | add challenge and |

Local | Codespaces

Clone

HTTPS | SSH | GitHub CLI

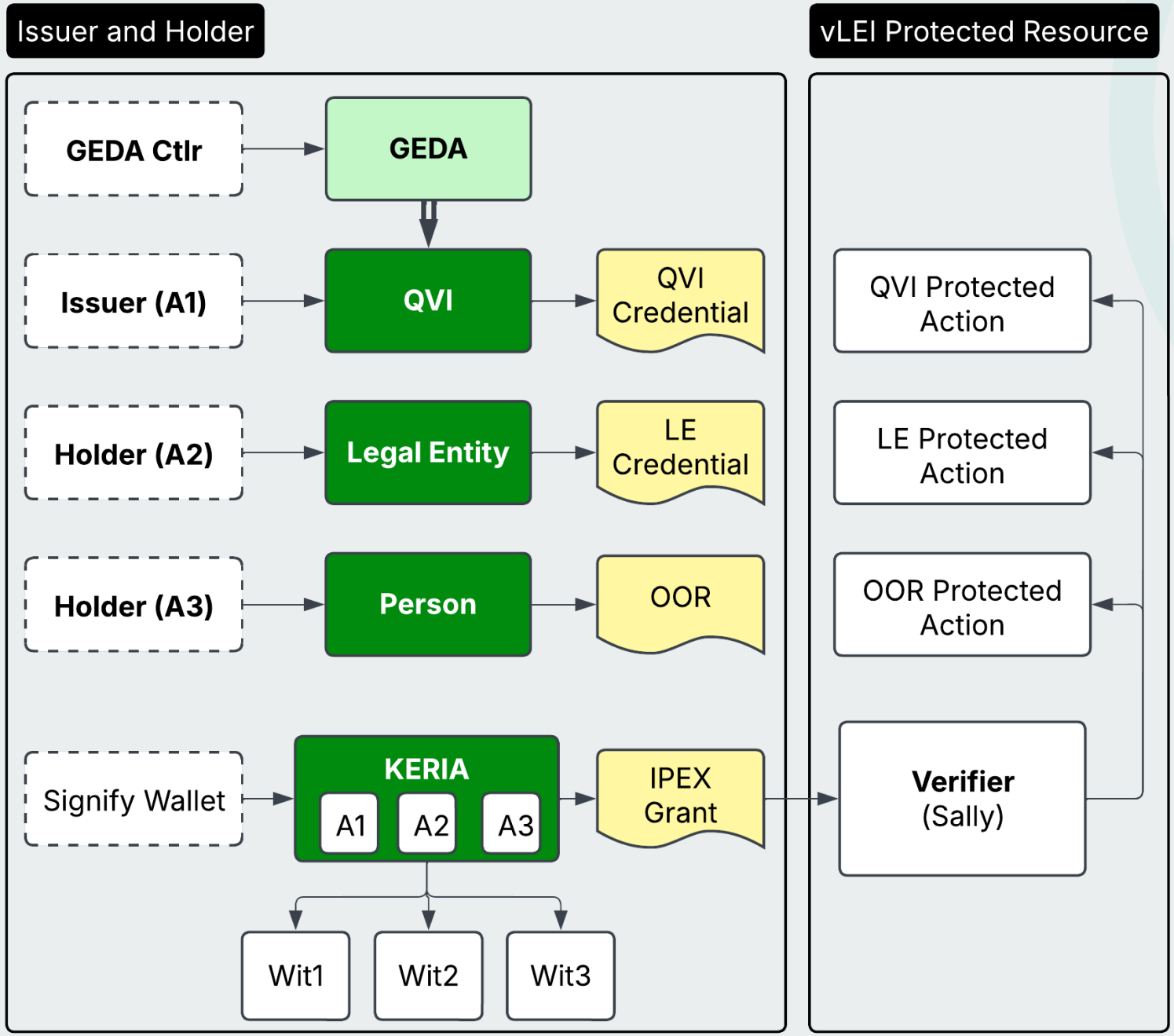
git@github.com:GLEIF-IT/vlei-hackathon-202

Use a password-protected SSH key.

Open with GitHub Desktop

GLEIF

# End Goal:

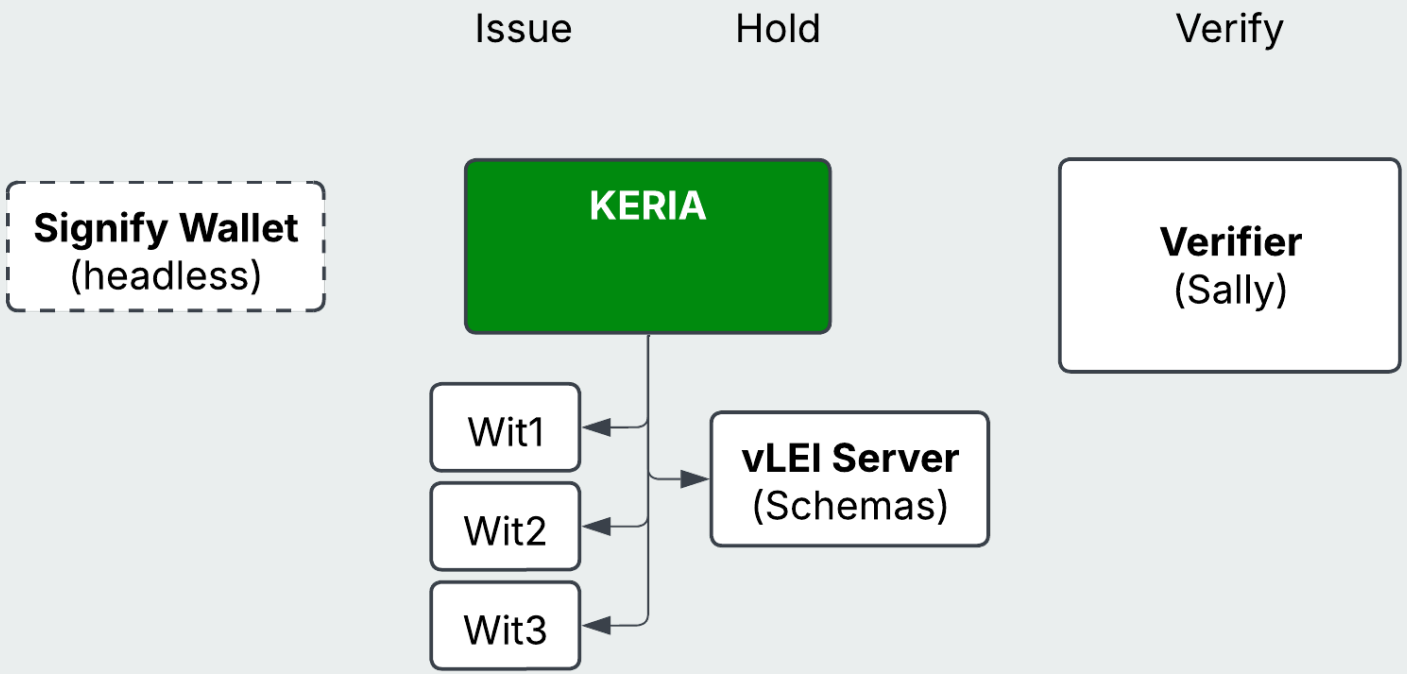

Issuer and Holder
vLEI Protected Resource
GEDA Ctlr
GEDA
Issuer (A1)
QVI
QVI Credential
Holder (A2)
Legal Entity
LE Credential
Holder (A3)
Person
OOR
Signify Wallet
KERIA
A1
A2
A3
IPEX Grant
Wit1
Wit2
Wit3
QVI Protected Action
LE Protected Action
OOR Protected Action
Verifier (Sally)

# Step 1: Infrastructure Setup

**docker compose build**
- builds the gleif/wkshp-tsx-shell image

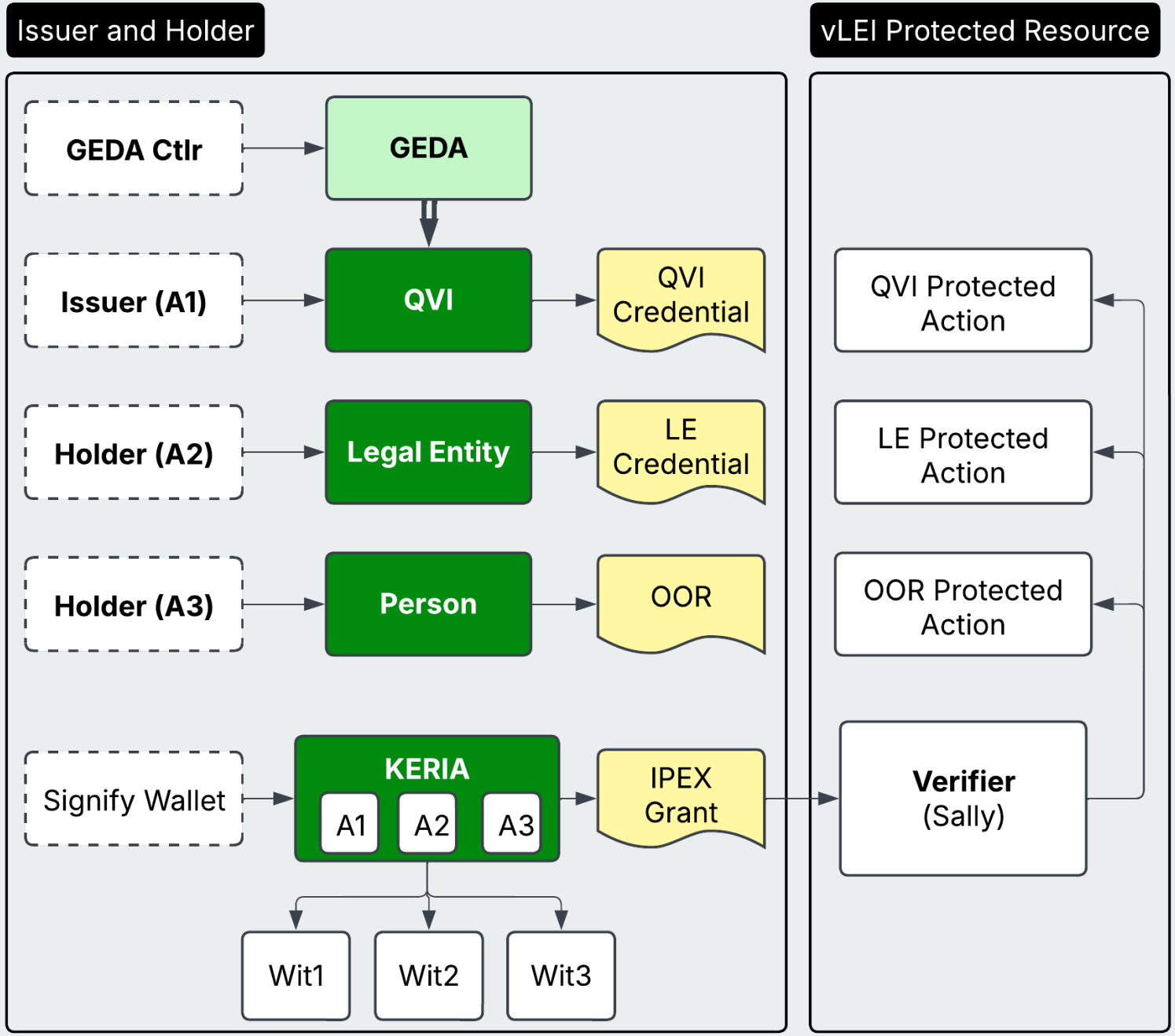
**./stop.sh -** to clear them out for a restart

**./deploy.sh -** sets up the following components



## Step SKIP: Skip to the end

**./run-all.sh** - Run all of the step scripts



# Script Path Notice

For all further scripts, assume they are run from

**`./task-scripts`**

Example:

**`geda/geda-aid-create.sh`**

would be run from

**`./task-scripts/geda/geda-aid-create.sh`**

# Creation of the GLEIF AID and QVI AID

# Step 2: Set up GEDA and QVI - delegation

**geda/geda-aid-create.sh**

**- create identifier for the GLEIF root of trust external delegate**

**verifier/recreate-with-geda-aid.sh**

**- restart sally with the new GEDA AID**

**qvi/qvi-aid-delegate-create.sh**

**- create the QVI identifier delegated from the GEDA**

**geda/geda-delegate-approve.sh**

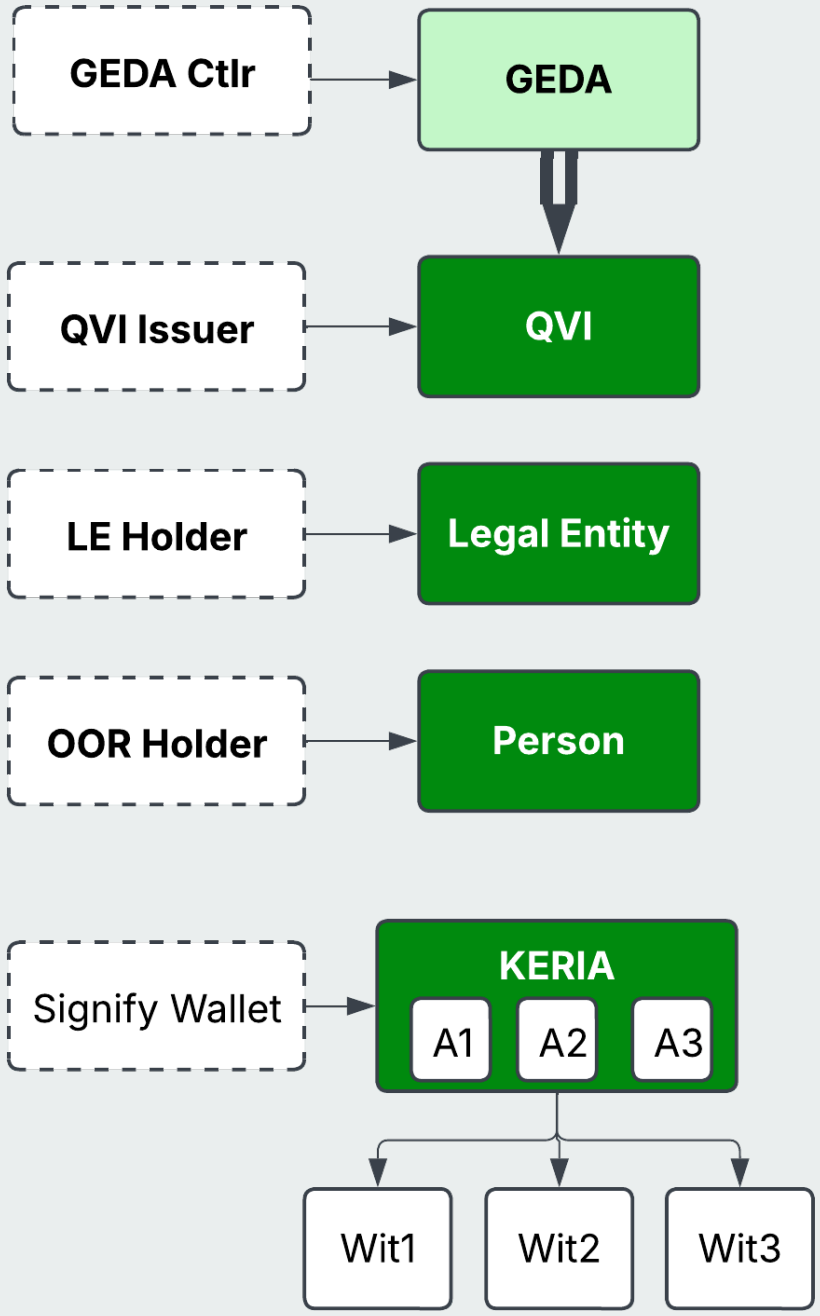
**- approve the delegation request from the QVI**

**qvi/qvi-aid-delegate-finish.sh**

**- QVI completes delegation by refreshing key state**

**geda/geda-oobi-resolve-qvi.sh**

**- GEDA resolves OOBI key state to get QVI mailbox endpoint (Location Scheme records) for subsequent communication**

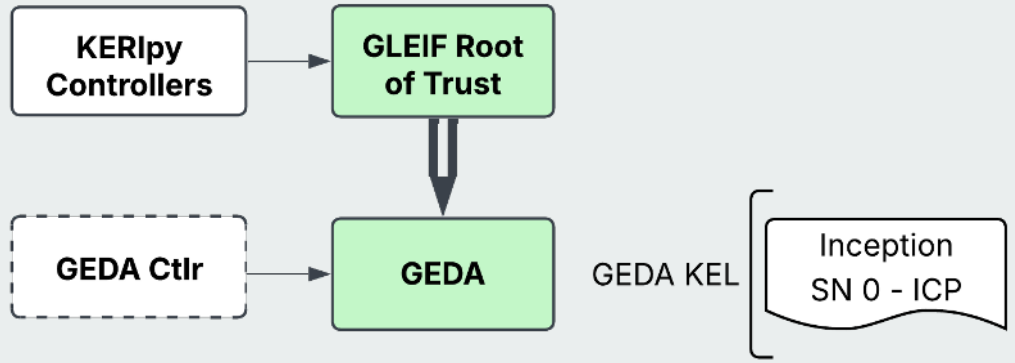


# Step 2 part 1/6: geda/geda-aid-create.sh - create AID for the GLEIF External Delegated AID

- Creates a single signature identifier (AID) for the:
  - GLEIF External Delegated AID (GEDA)
- In production, this is delegated from the GLEIF Root of Trust
- Will approve a delegation request from the QVI

```
const client = await getOrCreateClient(qviPasscode, env);
const gedaInfo: any = await createAid(client, 'geda');
```

```
...
export async function createAid(client: SignifyClient, name: string) {
...
client.identifiers().create(name);
...
client.identifiers()
        .addEndRole(name, 'agent', client!.agent!.pre);
```




GEDA AID

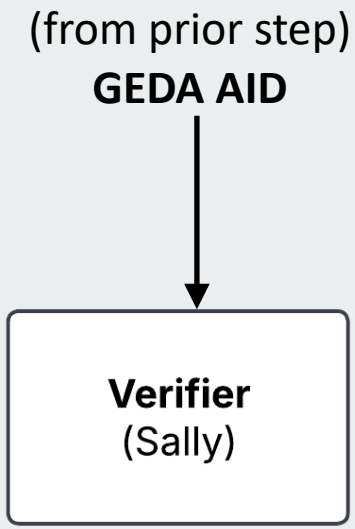
# Step 2 part 2/6: verifier/recreate-with-geda-aid.sh - update Verifier with new GEDA AID

- **Restarts the verifier (sally) with the new GEDA AID**
  - **In the `—auth` startup parameter**

```
sally server start \
  --name verifier \
  --alias verifier \
  --web-hook http://resource:9923 \
  --auth EDDe8pD24aqd0dCZTQHaGpfcluPFD2ajGIY3ARgE5DD \
  ...
```

**AID of QVI credential issuer must match this AID.**

**This Auth AID is validated when presenting a QVI credential.**




```
export GEDA_PRE="EDDe8pD24aqd0dCZTQHaGpfcluPFD2ajGIY3ARgE5DD"

docker compose -f docker-compose.yml down verifier
docker compose -f docker-compose.yml up -d verifier --wait
```

# Step 2 part 3/6: qvi/qvi-aid-delegate-create.sh - QVI requests AID delegation from GEDA AID

**Create the QVI identifier delegated from the GEDA**

**First, resolve OOBI of the delegator so it is known**

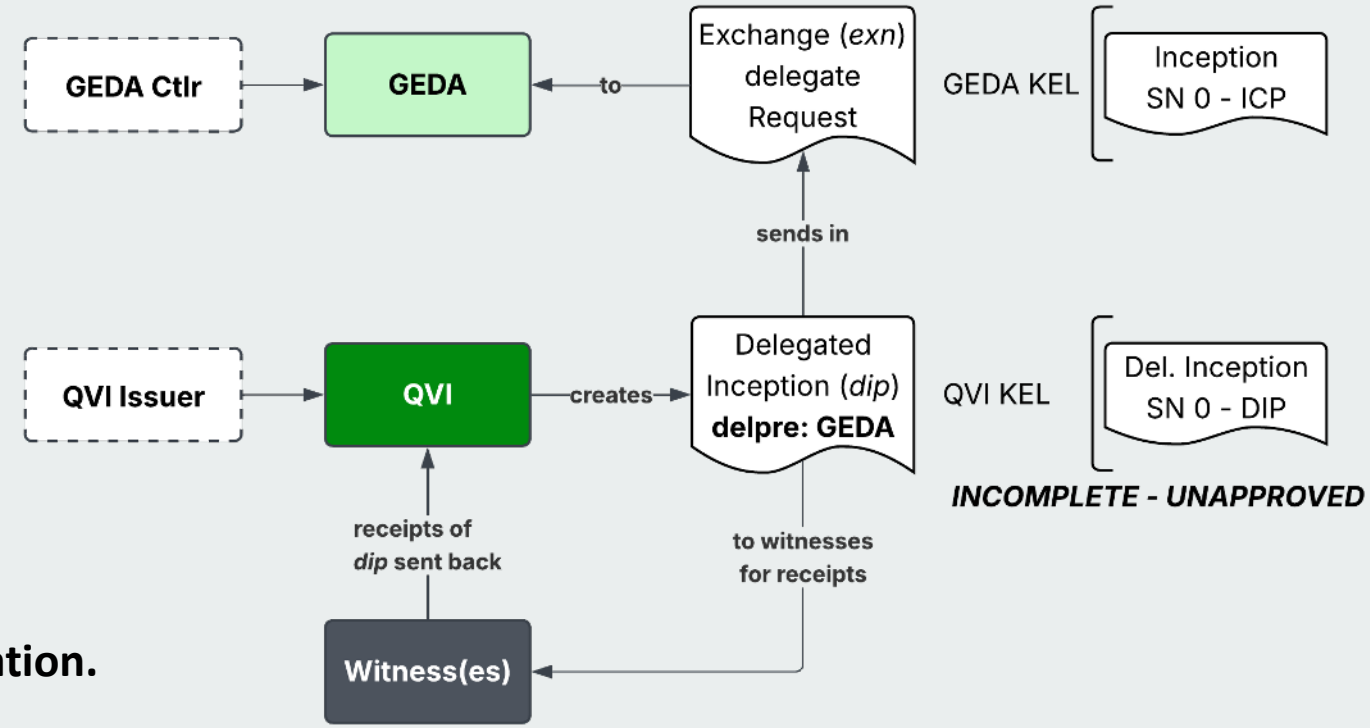
```
await resolveOobi(dgtClient, delOobi, dgrName)
```

**Then create the delegated inception event (*dip*)**

```
await dgtClient.identifiers().create(dgtName, {
        delpre: delPre
    });
```

**The Exchange (*exn*) message containing the dip is automatically sent to the delegator.**

**Next step is for the delegator to approve the delegation.**



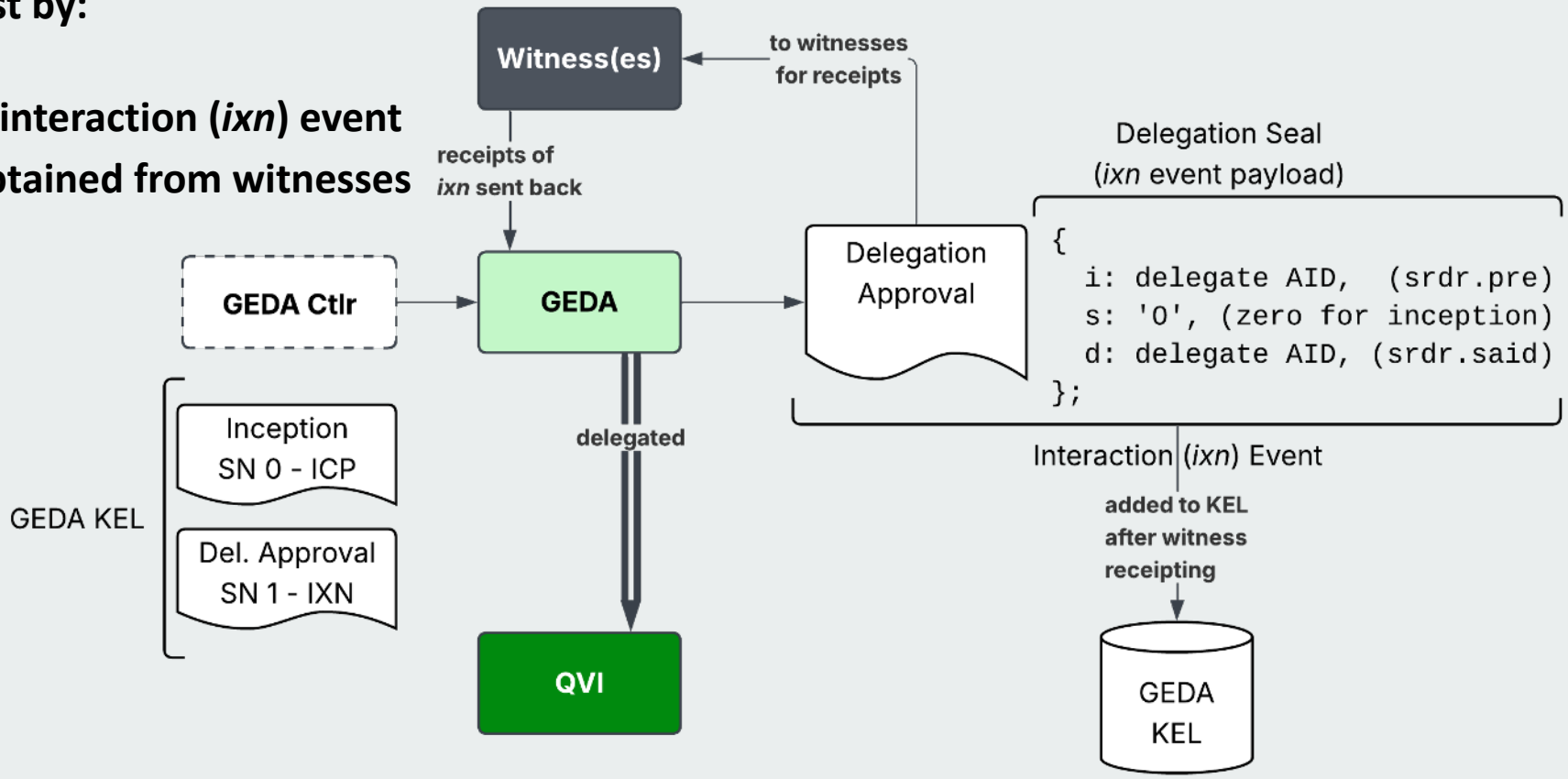
# Step 2 part 4/6: geda/geda-delegate-approve.sh - Anchors delegation approval seal

**Delegation request arrives in an exchange (*exn*) message.**

**GEDA approves the delegation request by:**

**1) creating a seal payload**
**2) anchoring that seal to its KEL in an interaction (*ixn*) event**
**3) any needed witness receipts are obtained from witnesses**

```
const anchor = {
        i: dgtPre,
        s: '0',
        d: dgtPre,
    };

    await dgrClient
        .delegations()
        .approve(dgrName, anchor);
```



# Step 2 part 5/6: qvi/qvi-aid-delegate-finish.sh - Discover GEDA Anchor

**QVI completes delegation by:**

**1) Refreshing GEDA KEL key state**

**2) Process GEDA KEL to discover delegation approval**

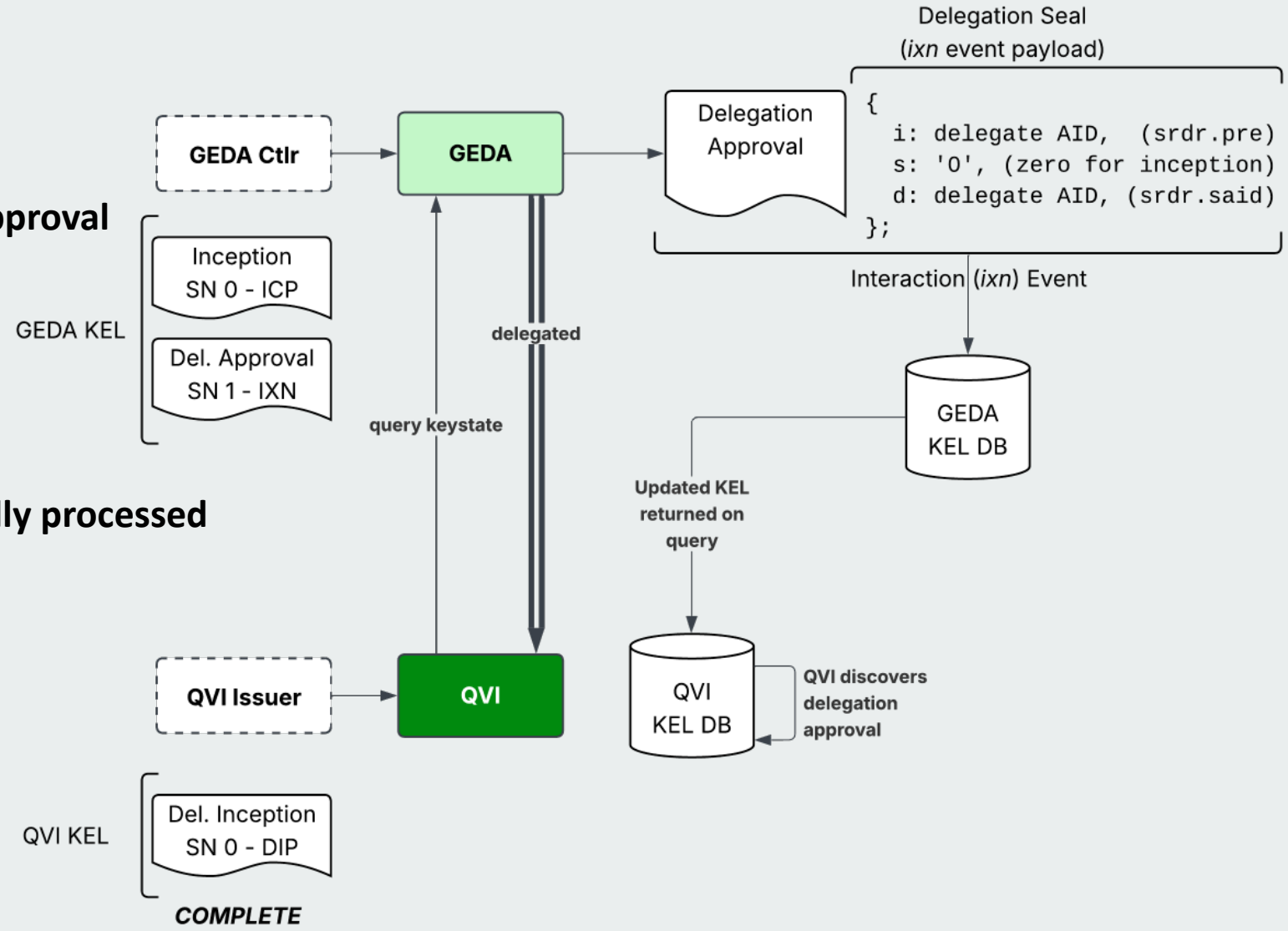
**- Delegation Seal in the interaction event**

**3) Complete the QVI AID setup**

```
await dgtClient.keyStates().query(delPre, '1');
```

**…delegation seal is picked up and automatically processed**

**…AID setup resumes (endpoint role)…**



# Step 2 part 6/6: geda/geda-oobi-resolve-qvi.sh - GEDA gets QVI AID key state, URL, roles,

**GEDA resolves QVI OOBI to**
**1) See where on the internet to send messages to the QVI**

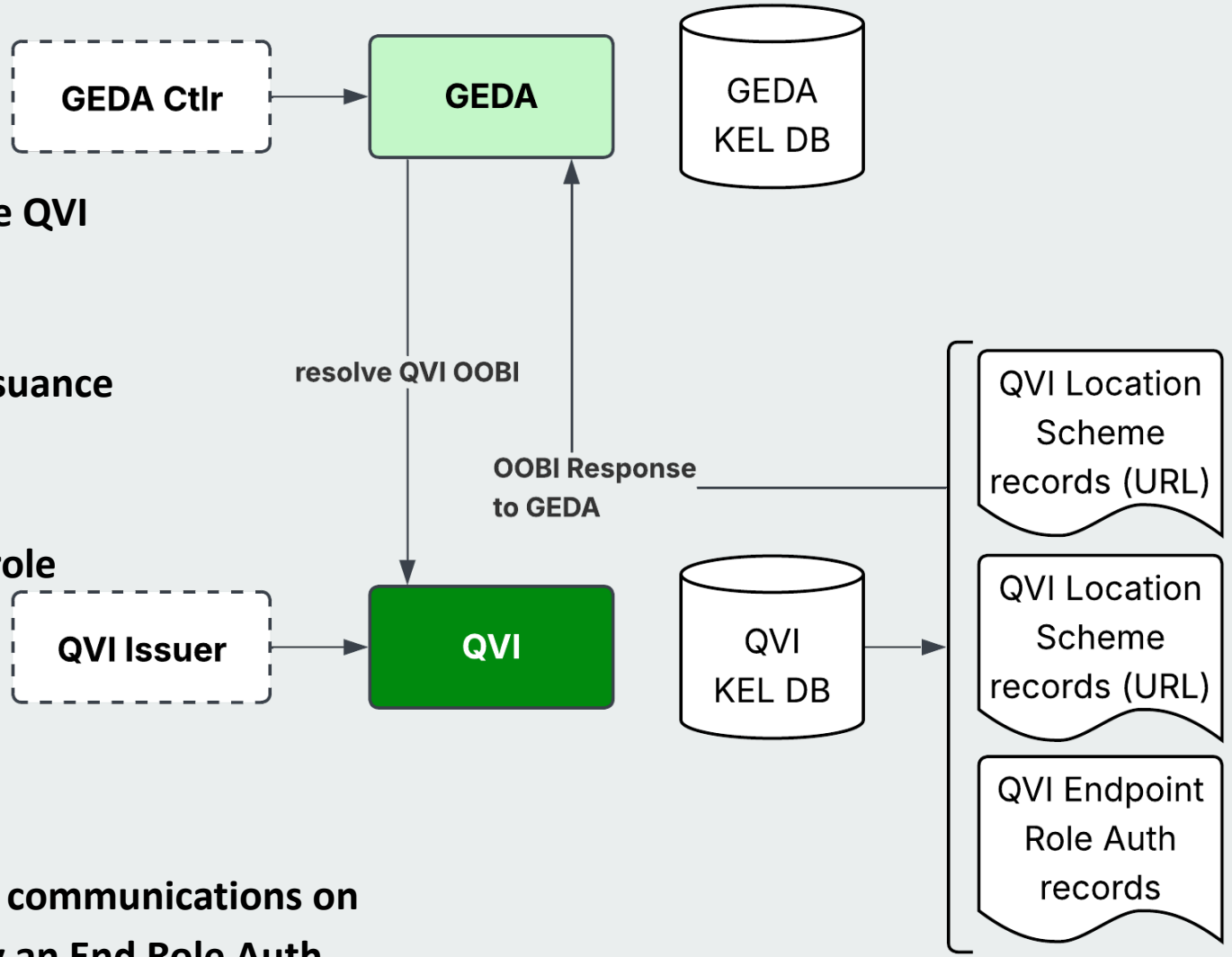
**This is essential so that the GEDA can:**
**- communicate with the QVI during QVI credential issuance**

**Endpoint Role Authorization records:**

**- designate an AID as authorized to perform a given role for an AID such as an agent, mailbox, or witness**
**- in this case authorize the KERIA agent to act as a mailbox (comms relay) for the QVI AID.**

**Location Scheme records:**

**- show the public URLs an AID declares it can receive communications on**
**- usually paired with the infrastructure authorized by an End Role Auth**



# Step 3: Challenge and Response between GEDA and QVI

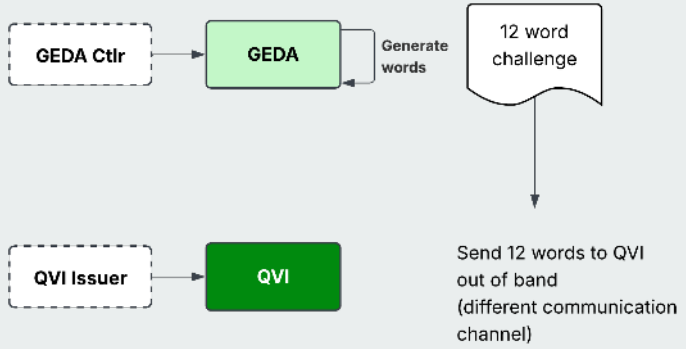
**Three step challenge and response process GEDA -> QVI**
**geda/geda-challenge-qvi.sh**
**- generate challenge words, share out of band**

**qvi/qvi-respond-geda-challenge.sh**
**- send signed exchange (exn) message with words back**

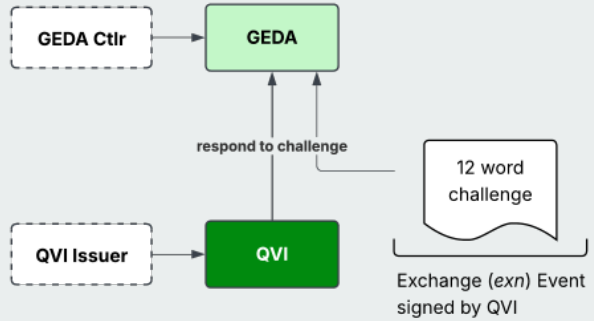
**geda/geda-verify-qvi-response.sh**
**- receive exn message with words, compare to local words**

**And then the same process in reverse QVI-> GEDA**
**qvi/qvi-challenge-geda.sh**
**geda/geda-respond-qvi-challenge.sh**
**qvi/qvi-verify-geda-response.sh**

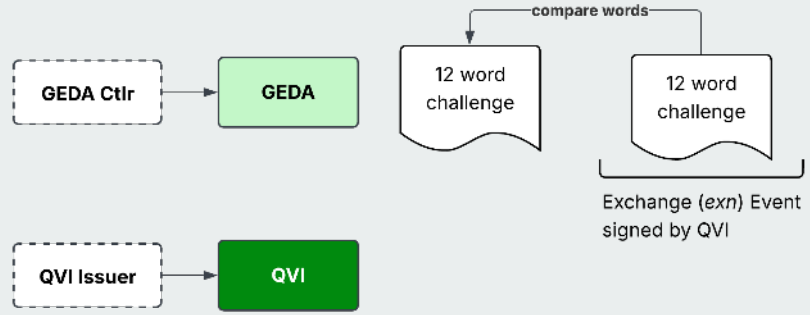
1




2




3



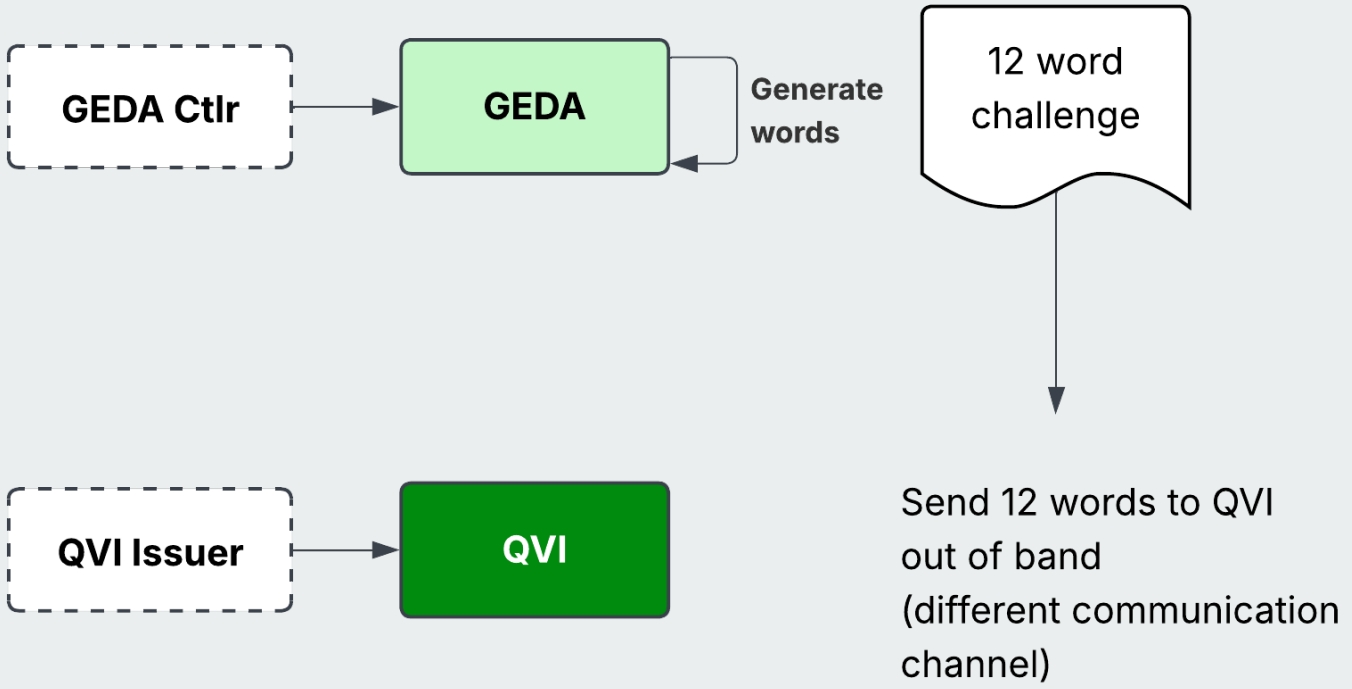
# Challenge Response Process for Mutual Control Verification

# Step 3 part 1/3: geda/geda-challenge-qvi.sh

**1) generate challenge words**

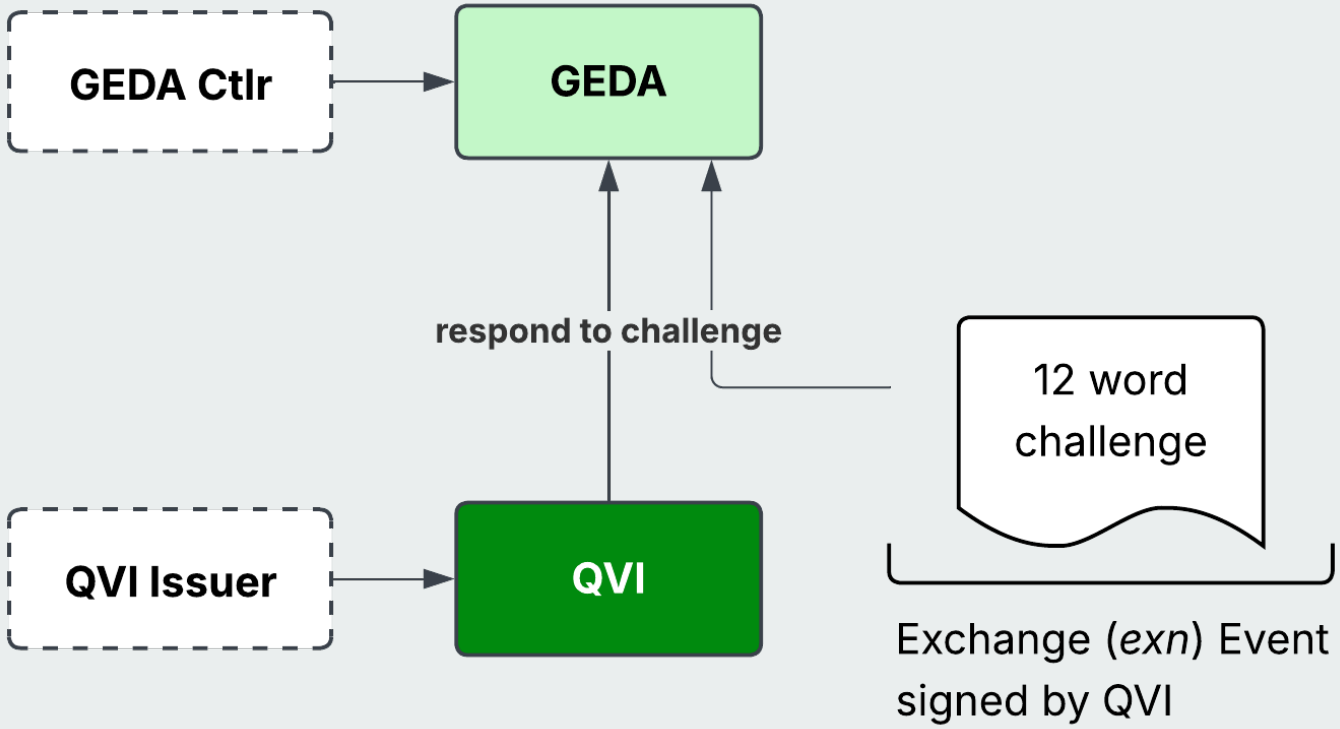
**2) share out of band**

**No in-band, agent to agent (mailbox to mailbox) communication has occurred yet.**



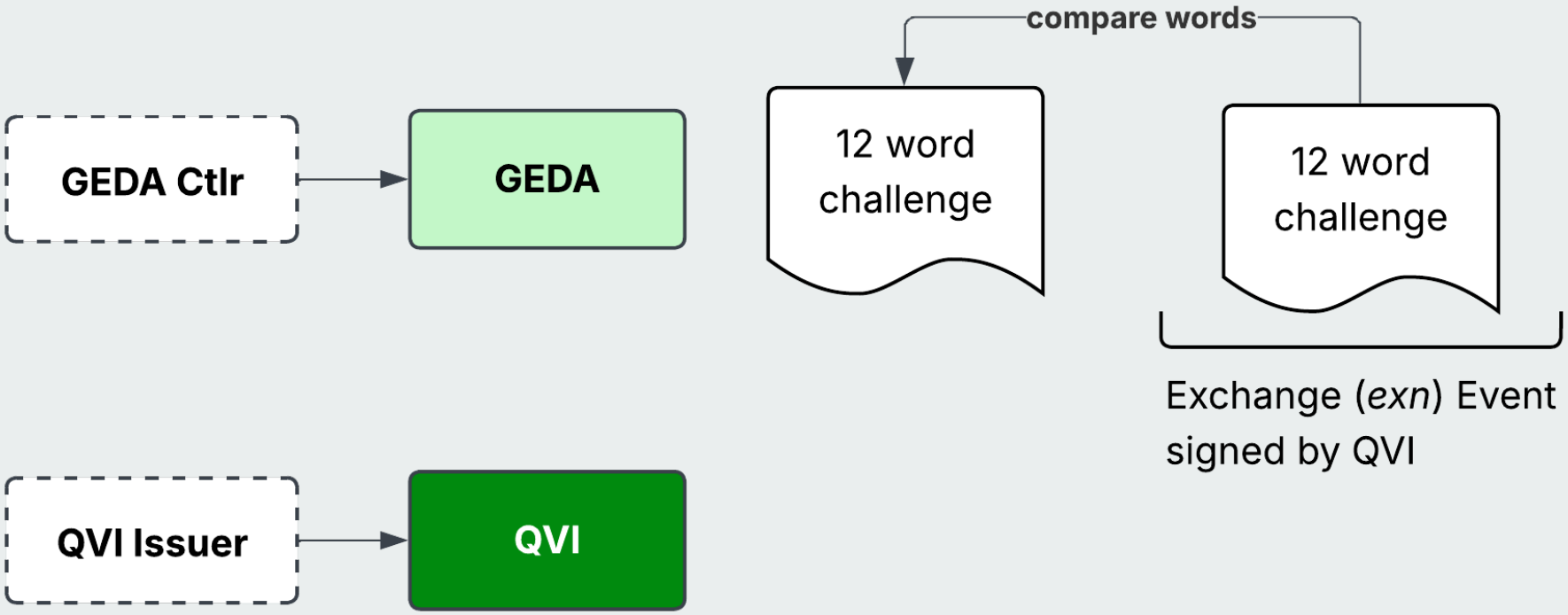
# Step 3 part 1/3: qvi/qvi-respond-geda-challenge.sh

**send signed exchange (exn) message with words back**



# geda/geda-verify-qvi-response.sh

**receive exn message with words, compare to local words**



# Issuance and presentation of the QVI credential

# Step 4: QVI credential issuance and presentation

**For Issuance:**

**geda/geda-registry-create.sh**

**- Create GEDA's ACDC registry for QVI ACDC credentials**

**geda/geda-acdc-issue-qvi.sh**

**- Issue the QVI credential from GEDA (IPEX Grant)**

**qvi/qvi-acdc-admit-qvi.sh**

**- Admit the QVI credential as the QVI (IPEX Admit)**

**For Presentation:**

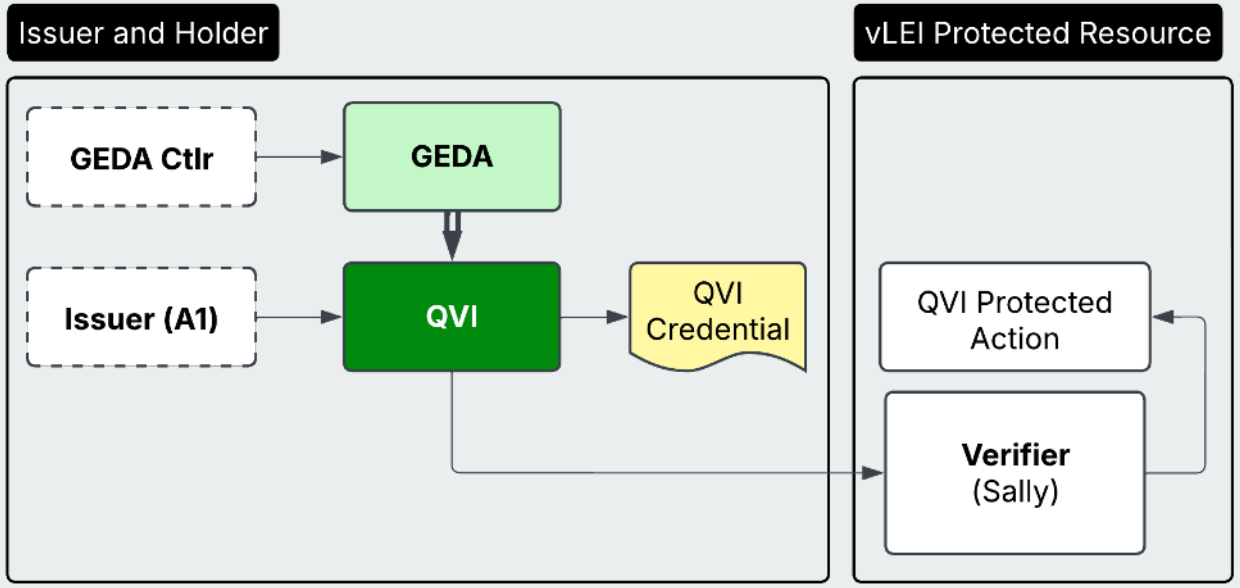
**qvi/qvi-oobi-resolve-verifier.sh**

**- Connect to Verifier by resolving its OOBI**

**qvi/qvi-acdc-present-qvi.sh**

**- Present the QVI credential to the Verifier**

**- As an IPEX Grant operation (IPEX Grant)**



# QVI Credential Issuance

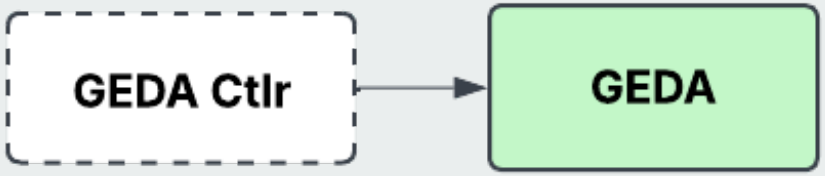
Creating and issuing the QVI credential from the GEDA to the QVI

verifiable Legal Entity Identifier

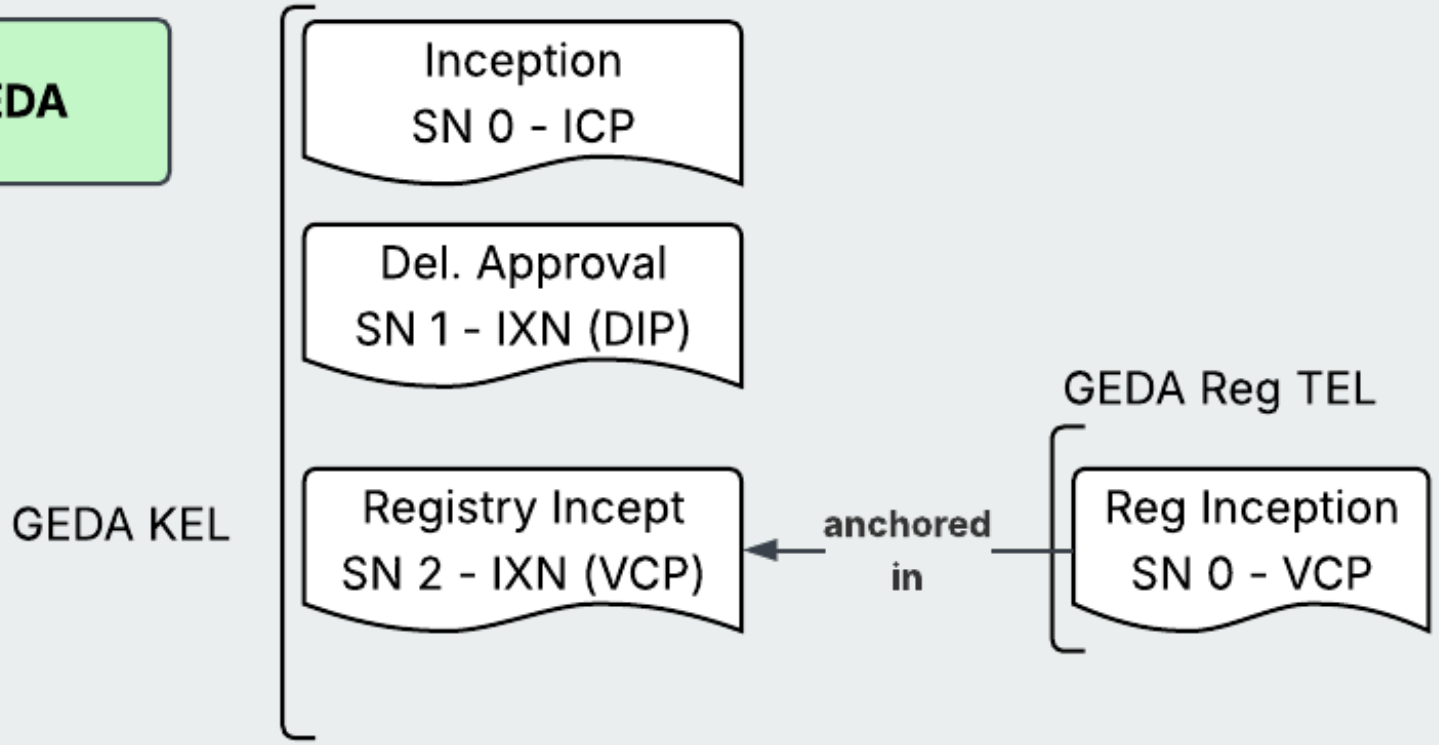
**PROTECTED**

# Step 4 part 1/6: geda/geda-registry-create.sh - Create ACDC Registry

**Create GEDA's ACDC registry for QVI ACDC credentials**

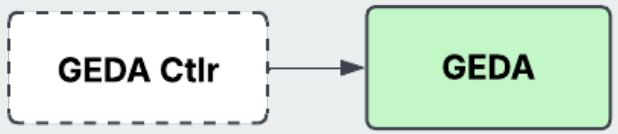



```
await client
            .registries()
            .create({ name: aidName, registryName:
registryName });
```

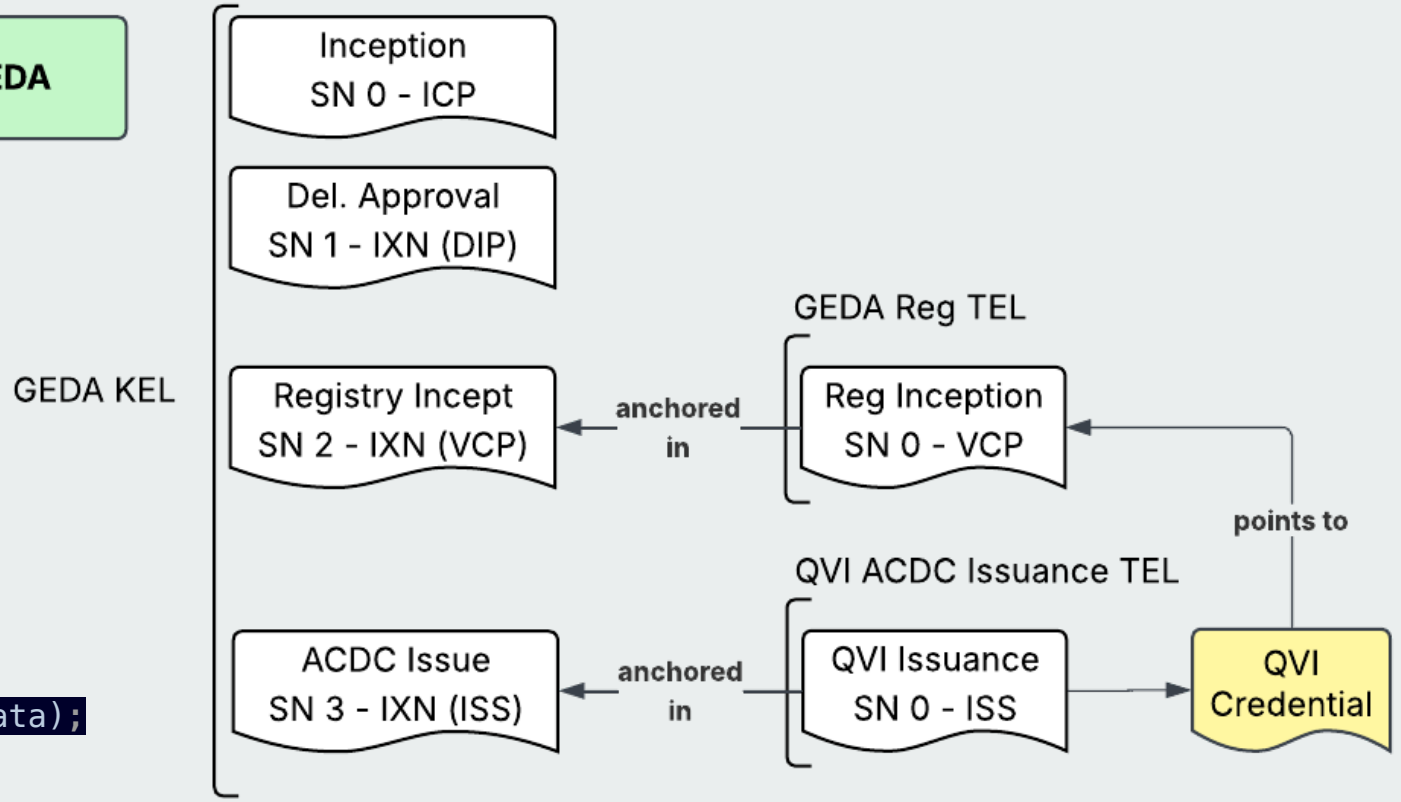


# Step 4 part 2/6: geda/geda-acdc-issue-qvi.sh - Create QVI ACDC credential

## Issue the QVI credential from GEDA




```
    const subject: CredentialSubject = {
        i: hldPrefix,
        dt: createTimestamp(),
        ...credData,
    };
    const issData: CredentialData = {
        i: issAid.prefix,
        ri: issRegistry.regk,
        s: schema,
        a: subject,
        e: credEdge,
        r: credRules,
    };
    const issResult = await
        issClient.credentials().issue(issAidName, issData);
```



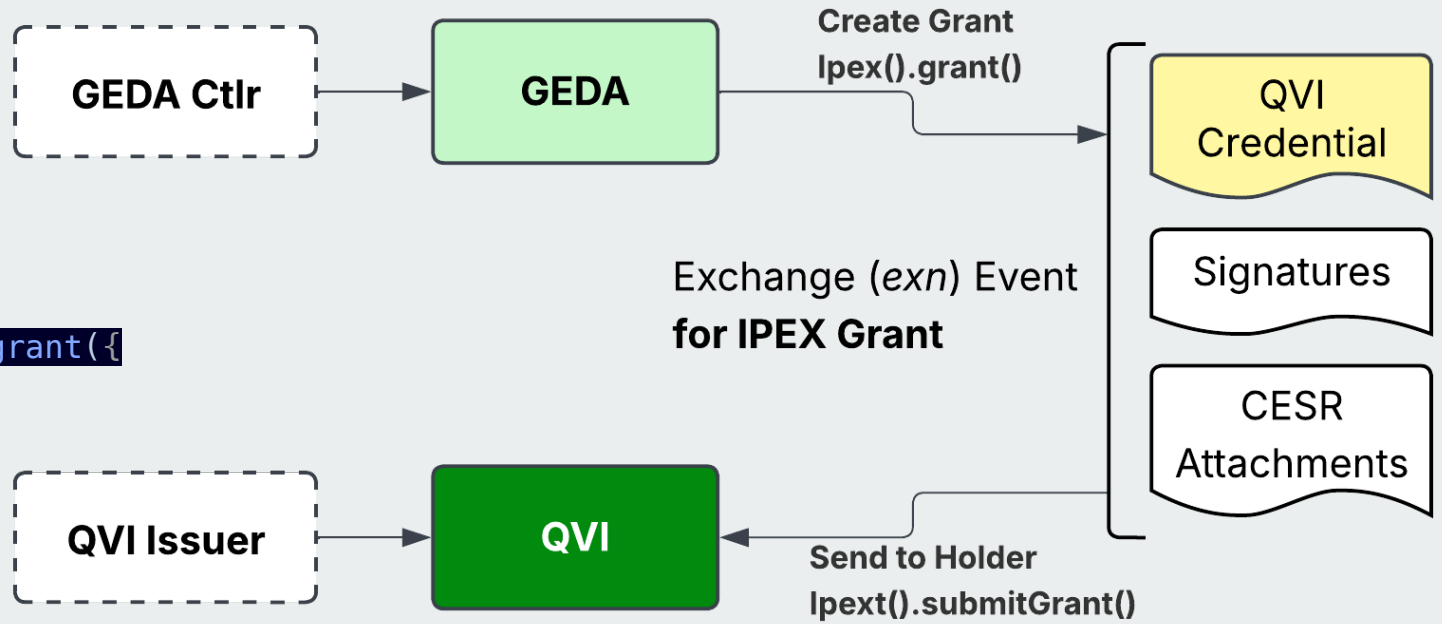
# Step 4 part 3/6: geda/geda-acdc-issue-qvi.sh - Send (Present) QVI ACDC to Holder (QVI)

**Send the QVI ACDC to the Holder (QVI) with IPEX Grant**

**The IPEX Grant operation includes the QVI ACDC in the CESR stream sent to the recipient.**

```
const [grant, gsigs, end] = await client.ipex().grant({
            senderName: issName,
            recipient: issueeAid,
            datetime: dt,
            acdc: acdc,
            anc: anc,
            iss: iss,
        });
```

```
    await client
        .ipex()
        .submitGrant(issName, grant, gsigs, end,
[issueeAid]);
```



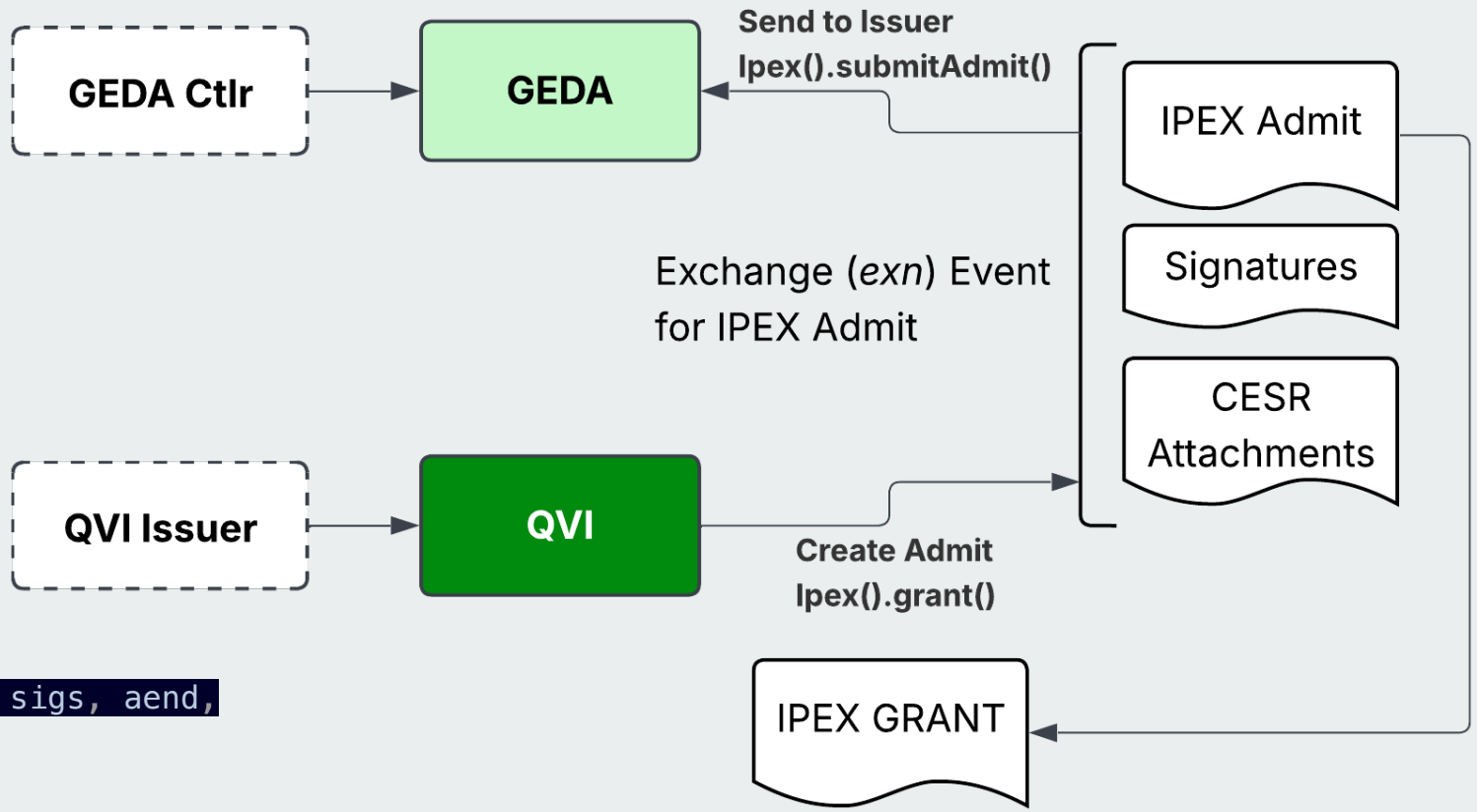
# Step 4 part 4/6: qvi/qvi-acdc-admit-qvi.sh - Admit QVI credential as the QVI

**Admit the QVI credential as the QVI (IPEX Admit)**

**The IPEX Admit points back to**

**the IPEX Grant.**

```
await client.ipex().admit({
            senderName: senderAidAlias,
            message: message,
            grantSaid: grantSaid,
            recipient: recipientAidPrefix,
            datetime: createTimestamp(),
        });

await client
            .ipex()
            .submitAdmit(senderAidAlias, admit, sigs, aend,
[recipientAidPrefix]);
```



# QVI Credential Presentation

Presenting the QVI credential from the QVI to the Verifier

verifiable Legal
Entity Identifier

**PROTECTED**

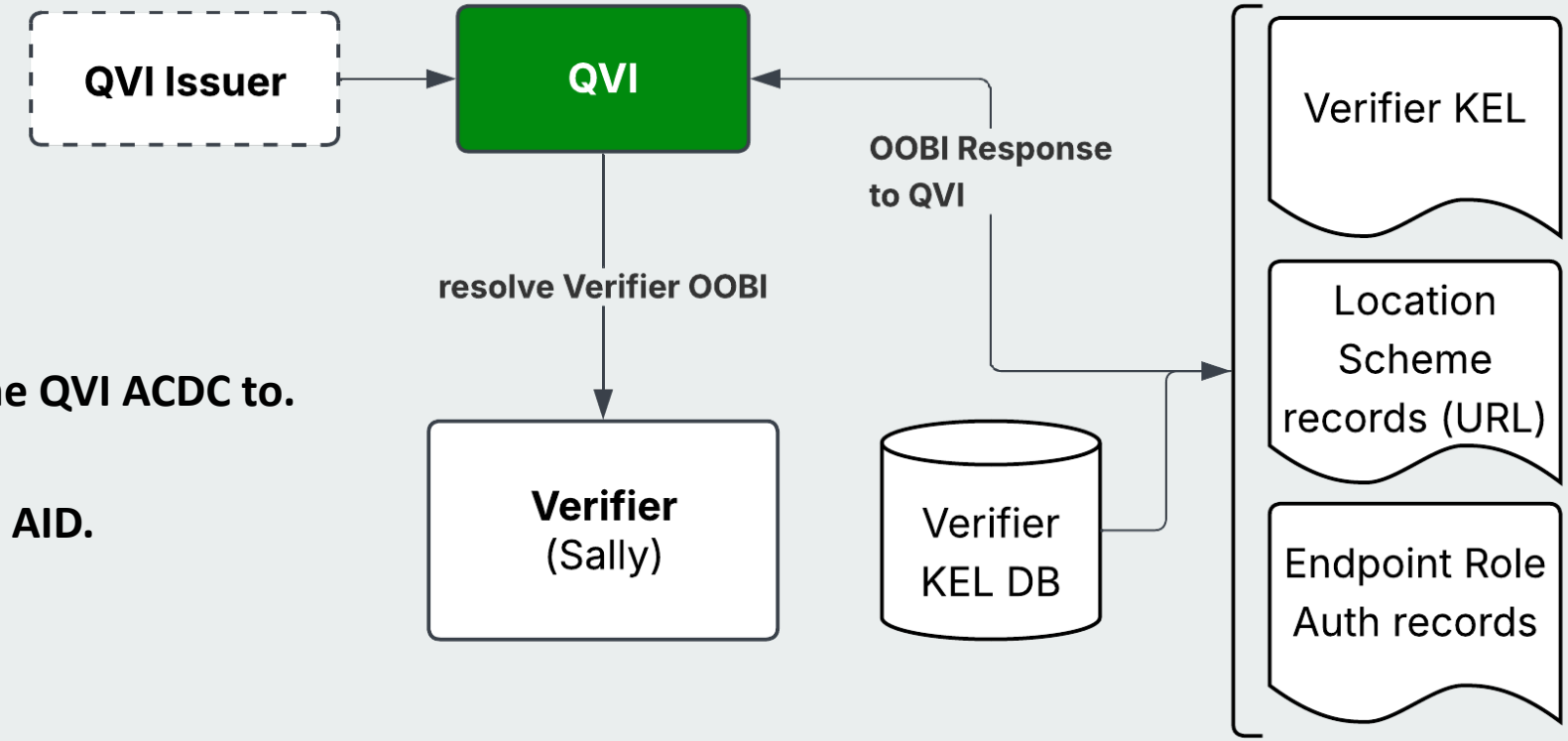
# Step 4 part 5/6: qvi/qvi-oobi-resolve-verifier.sh - Connect QVI to Verifier for presentation

**Connect to Verifier by resolving its OOBI**

**The QVI KERIA Agent needs the KEL of the Verifier including:**
**- Location Scheme records**
**- Endpoint Role Authorization records**

**So that it can know where to present the QVI ACDC to.**

**Presentations in KERI occur from AID to AID.**

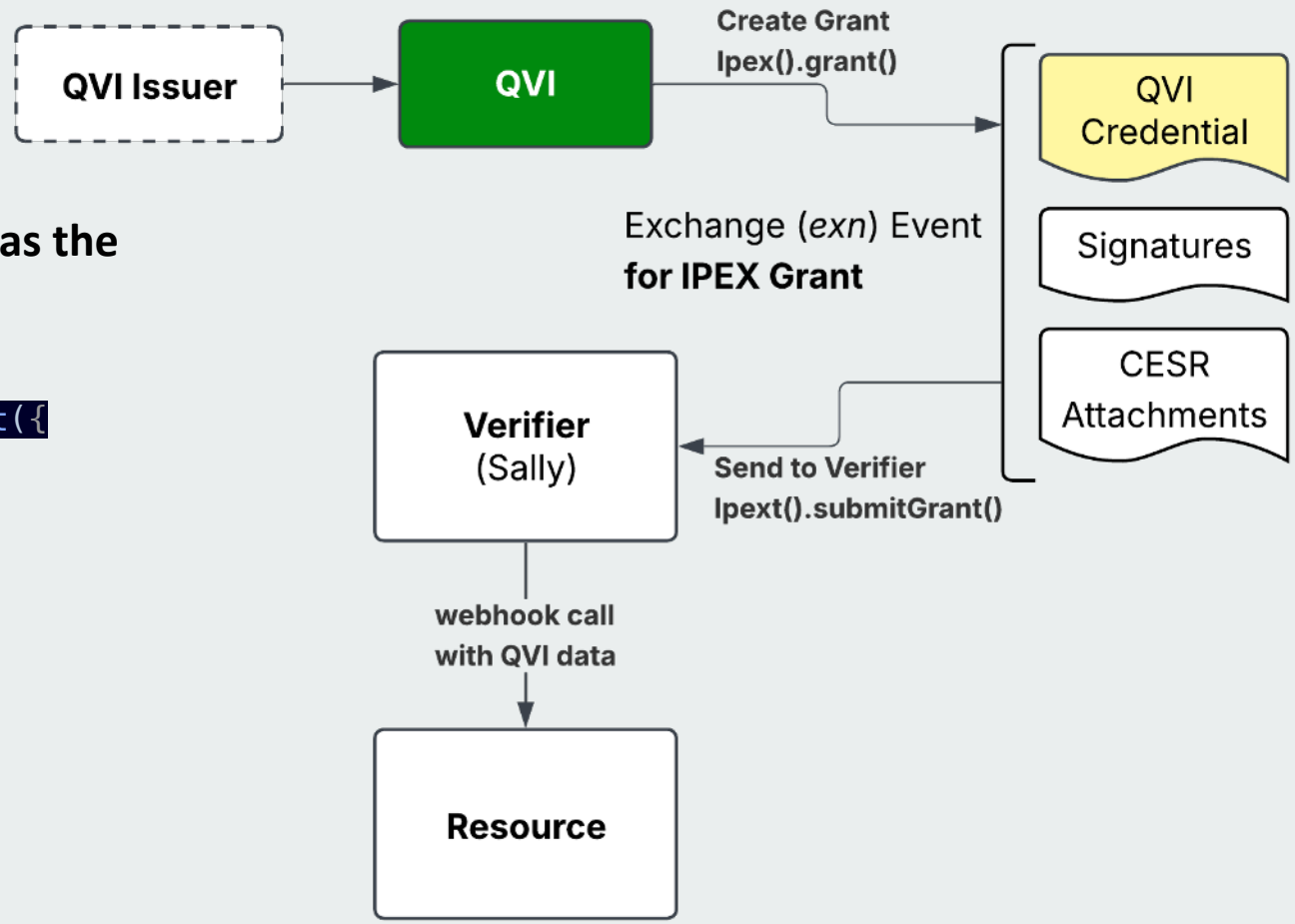


# Step 4 part 6/6: qvi/qvi-acdc-present-qvi.sh - Present QVI credential to verifier

**Present the QVI credential to the Verifier.**

**This uses an IPEX Grant operation with the verifier as the recipient.**

```
const [grant, gsigs, end] = await client.ipex().grant({
            senderName: issName,
            recipient: issueeAid,
            datetime: dt,
            acdc: acdc,
            anc: anc,
            iss: iss,
        });

    await client
        .ipex()
        .submitGrant(issName, grant, gsigs, end,
[issueeAid]);
```



# Issuance and presentation of the Legal Entity credential

# Step 5: LE AID setup, LE ACDC issuance and presentation

**Identifier Setup:**

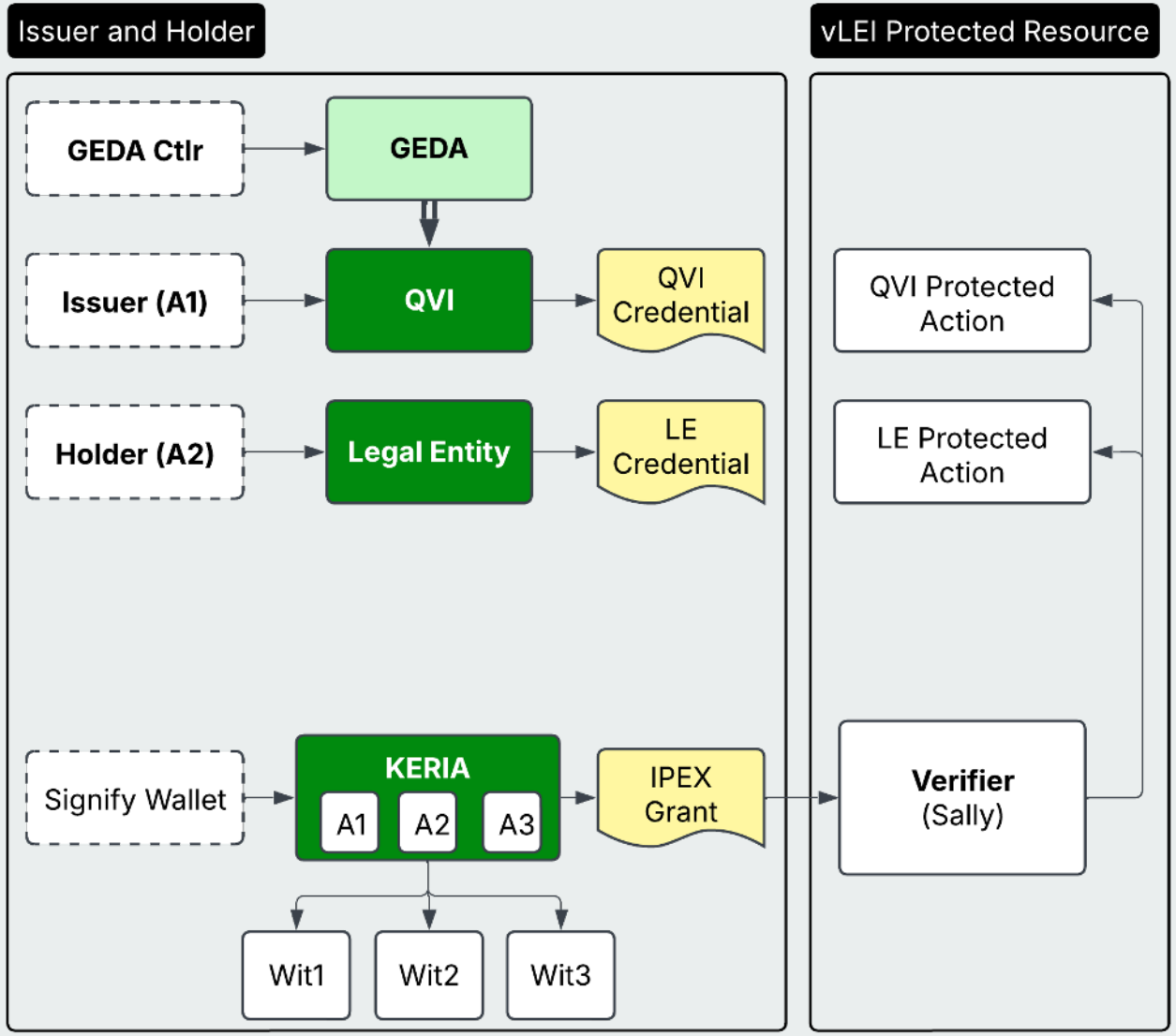
**le/le-aid-create.sh - create Legal Entity AID**
**le/le-oobi-resolve-qvi.sh - Connect LE to QVI with OOBI Resolve**
**qvi/qvi-oobi-resolve-le.sh - Connect QVI to LE with OOBI Resolve**

**For Issuance:**

**qvi/qvi-registry-create.sh**
**- Create QVI's ACDC registry for LE ACDC credentials**
**qvi/qvi-acdc-issue-le.sh**
**- Issue the LE credential from QVI (IPEX Grant)**
**le/le-acdc-admit-le.sh**
**- Admit the LE credential as the LE (IPEX Admit)**

**For Presentation:**

**le/le-oobi-resolve-verifier.sh**
**- Connect to Verifier by resolving its OOBI**
**le/le-acdc-present-le.sh**
**- Present the LE credential to the Verifier**
**- As an IPEX Grant operation (IPEX Grant)**



# LE Credential Issuance

Creating and issuing the LE credential from the QVI to the LE

verifiable Legal
Entity Identifier

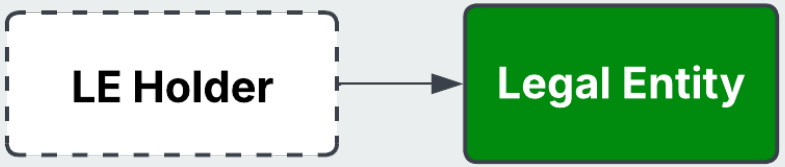
**PROTECTED**

# Step 5 part 1/8: le/le-aid-create.sh - create AID for the Legal Entity

- **Creates a single signature identifier (AID) for the legal entity (LE)**

```
const client = await getOrCreateClient(lePasscode, env);
const leInfo: any = await createAid(client, 'le');
```

```
...
export async function createAid(client: SignifyClient, name: string) {
...
client.identifiers().create(name);
...
client.identifiers()
        .addEndRole(name, 'agent', client!.agent!.pre);
```



# Step 5 part 2/8: le/le-oobi-resolve-qvi.sh - LE gets QVI AID key state, URL, roles,

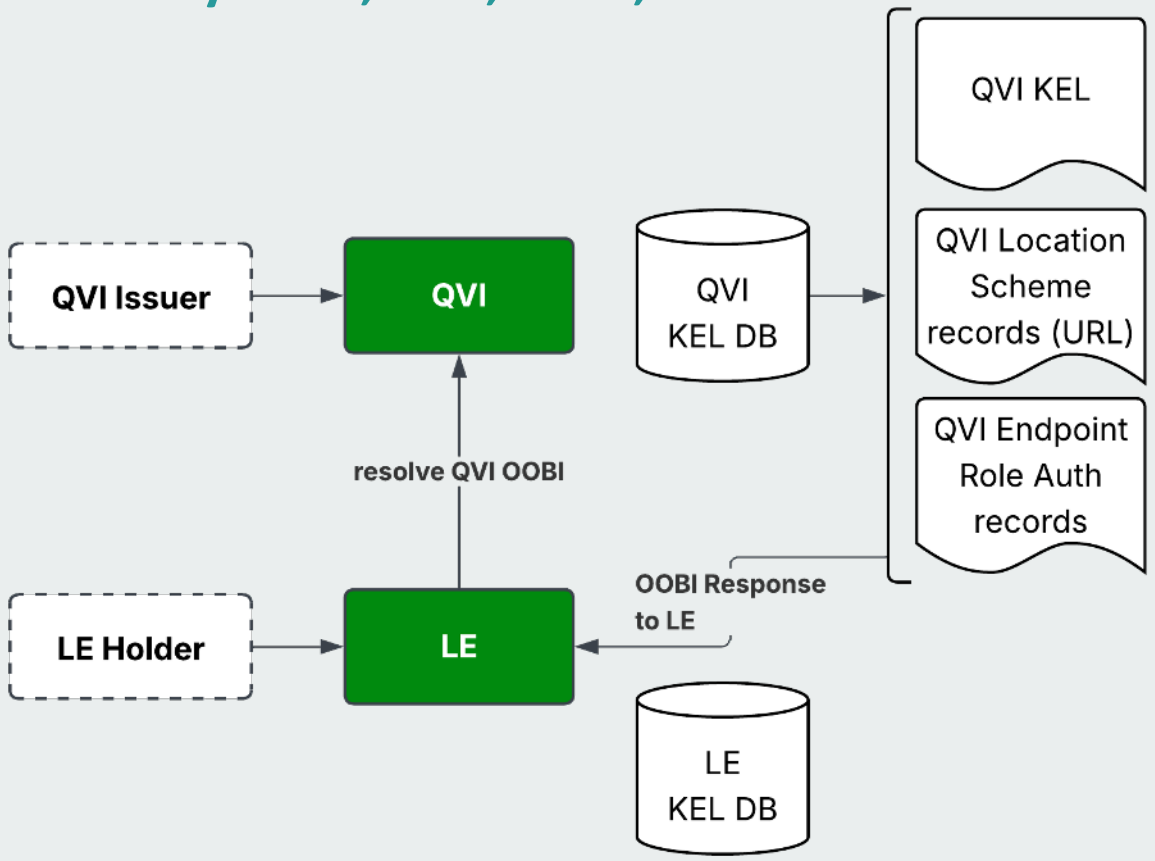
**LE resolves QVI OOBI to**
**1) See where on the internet to send messages to the QVI**

**This is essential so that the LE can:**
**- communicate with the QVI during LE credential issuance**

**Endpoint Role Authorization records:**

**- designate an AID as authorized to perform a given role for an AID such as an agent, mailbox, or witness**

**- in this case authorize the KERIA agent to act as a mailbox (comms relay) for the QVI AID.**




**Location Scheme records:**

**- show the public URLs an AID declares it can receive communications on**

**- usually paired with the infrastructure authorized by an End Role Auth**

# Step 5 part 3/8: qvi/qvi-oobi-resolve-le.sh - the reverse, QVI gets LE key state, URL, roles,

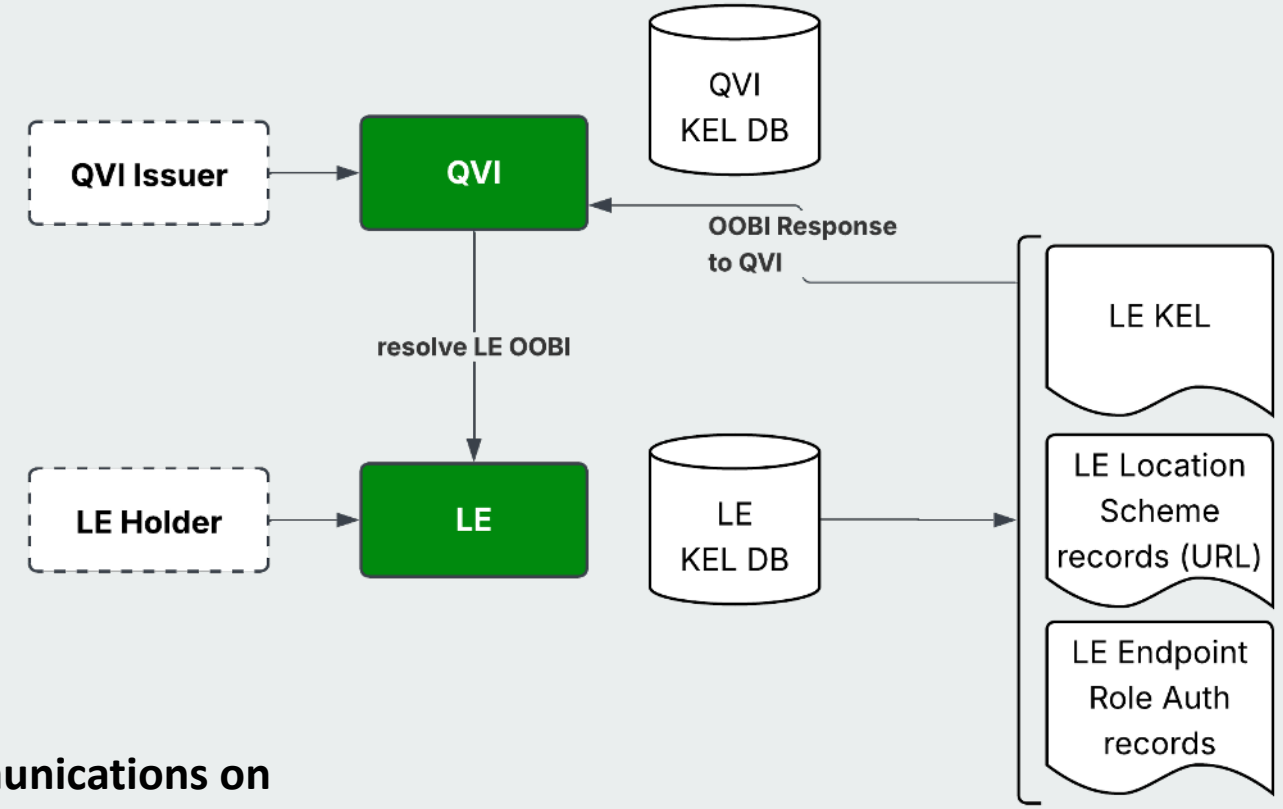
**LE resolves QVI OOBI to**
**1) See where on the internet to send messages to the QVI**

**This is essential so that the LE can:**
**- communicate with the QVI during LE credential issuance**

**Endpoint Role Authorization records:**

**- designate an AID as authorized to perform a given role for an AID such as an agent, mailbox, or witness**

**- in this case authorize the KERIA agent to act as a mailbox (comms relay) for the QVI AID.**



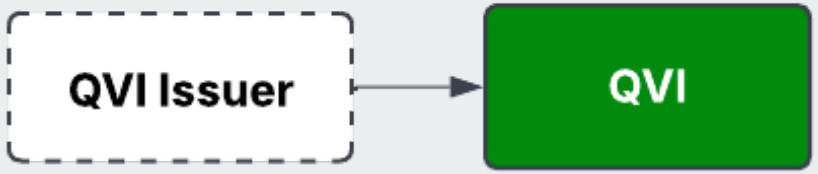

**Location Scheme records:**

**- show the public URLs an AID declares it can receive communications on**

**- usually paired with the infrastructure authorized by an End Role Auth**

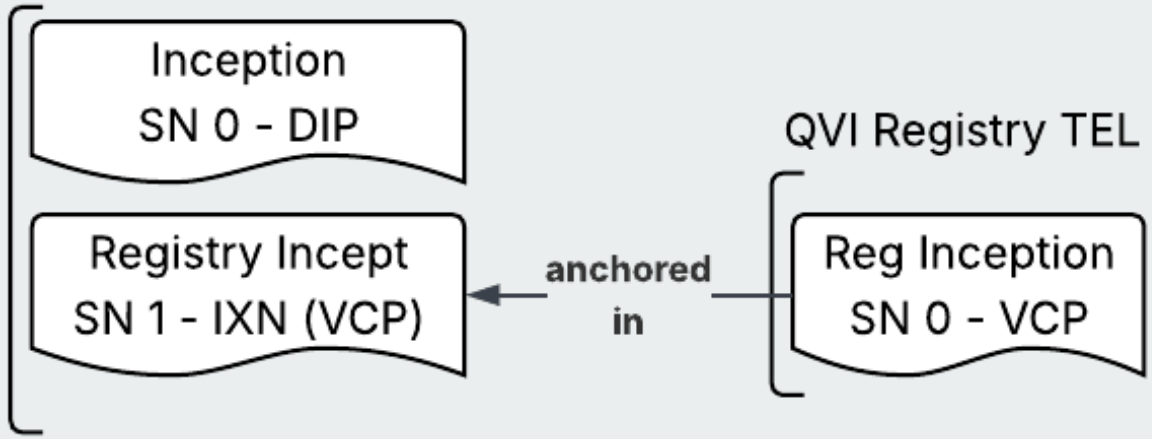
# Step 5 part 4/8: qvi/qvi-registry-create.sh - Create the QVI's ACDC registry

**Create QVI's ACDC registry for LE ACDC credentials**




```
await client
            .registries()
            .create({ name: aidName, registryName:
registryName });
```

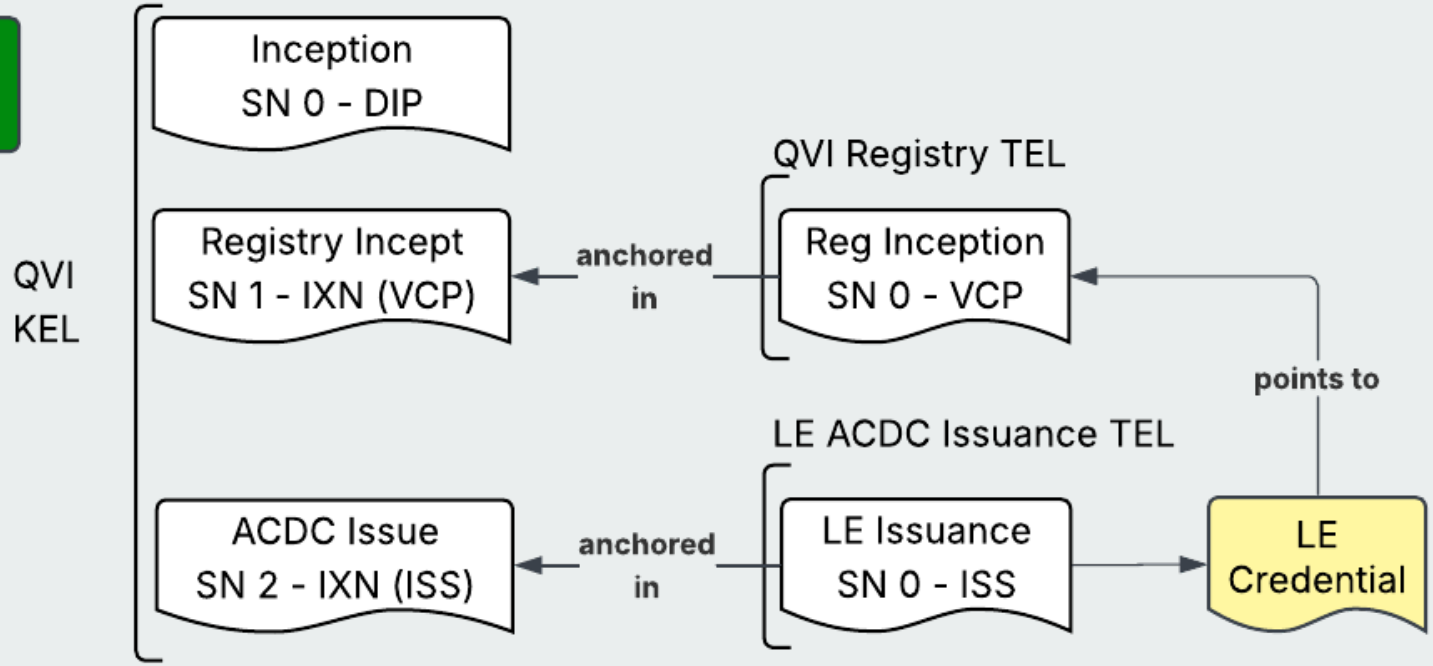
QVI KEL



# Step 5 part 5/8: qvi/qvi-acdc-issue-le.sh Part 1 - Create LE ACDC credential

## Issue the LE credential from QVI

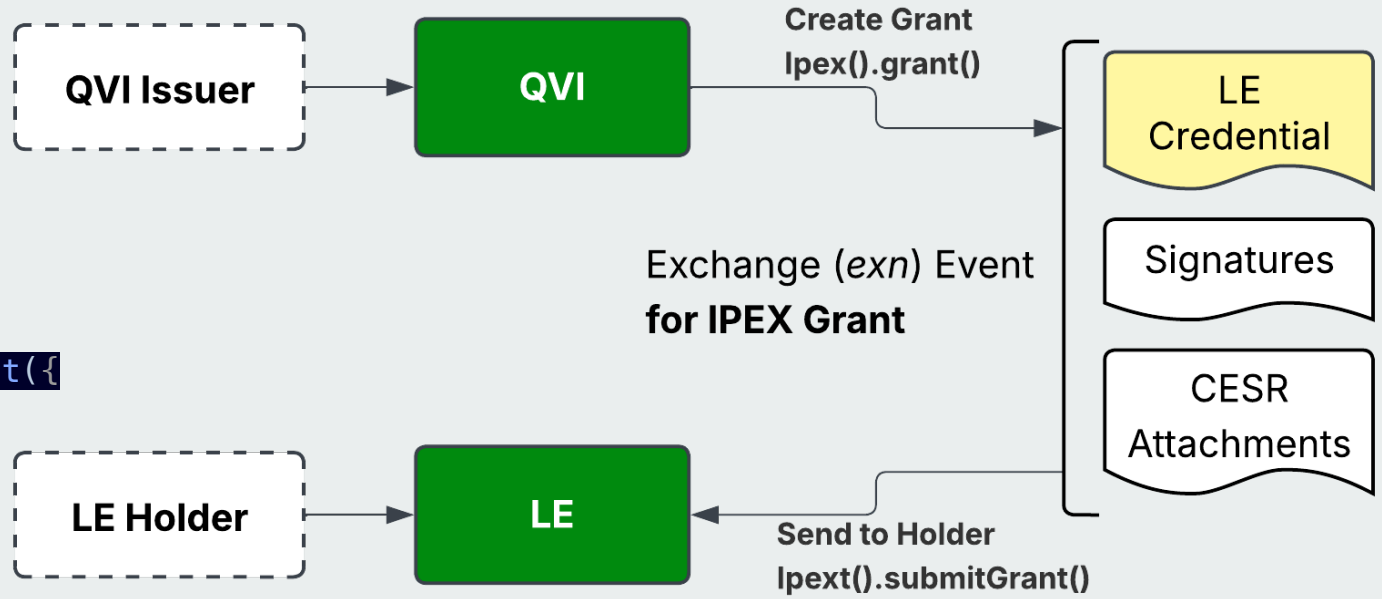

```
    const subject: CredentialSubject = {
        i: hldPrefix,
        dt: createTimestamp(),
        ...credData,
    };
    const issData: CredentialData = {
        i: issAid.prefix,
        ri: issRegistry.regk,
        s: schema,
        a: subject,
        e: credEdge,
        r: credRules,
    };
    const issResult = await
        issClient.credentials().issue(issAidName, issData);
```



# Step 5 part 5/8: qvi/qvi-acdc-issue-le.sh Part 2 - Send (Present) LE ACDC Holder (LE)

**Send the LE ACDC to the Holder (LE) with IPEX Grant**

**The IPEX Grant operation includes the LE ACDC in the CESR stream sent to the recipient.**




```
const [grant, gsigs, end] = await client.ipex().grant({
            senderName: issName,
            recipient: issueeAid,
            datetime: dt,
            acdc: acdc,
            anc: anc,
            iss: iss,
        });

    await client
        .ipex()
        .submitGrant(issName, grant, gsigs, end,
[issueeAid]);
```

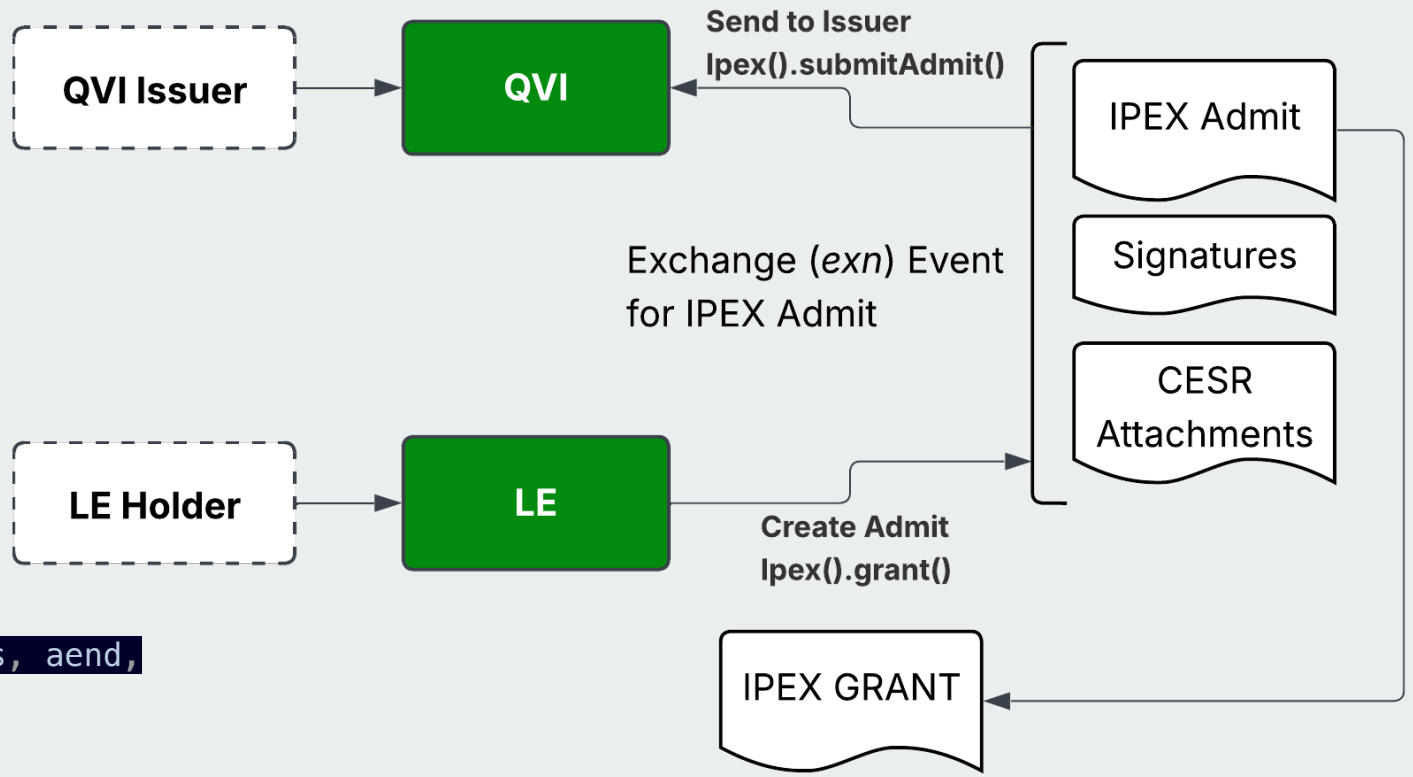
# Step 5 part 6/8: le/le-acdc-admit-le.sh - Admit LE credential as the legal entity

**Admit the QVI credential as the QVI (IPEX Admit)**

**The IPEX Admit points back to**

**the IPEX Grant.**

```
await client.ipex().admit({
            senderName: senderAidAlias,
            message: message,
            grantSaid: grantSaid,
            recipient: recipientAidPrefix,
            datetime: createTimestamp(),
        });

await client
            .ipex()
            .submitAdmit(senderAidAlias, admit, sigs, aend,
[recipientAidPrefix]);
```



# LE Credential Presentation

Presenting the LE credential from the LE to the Verifier

verifiable Legal
Entity Identifier

**PROTECTED**

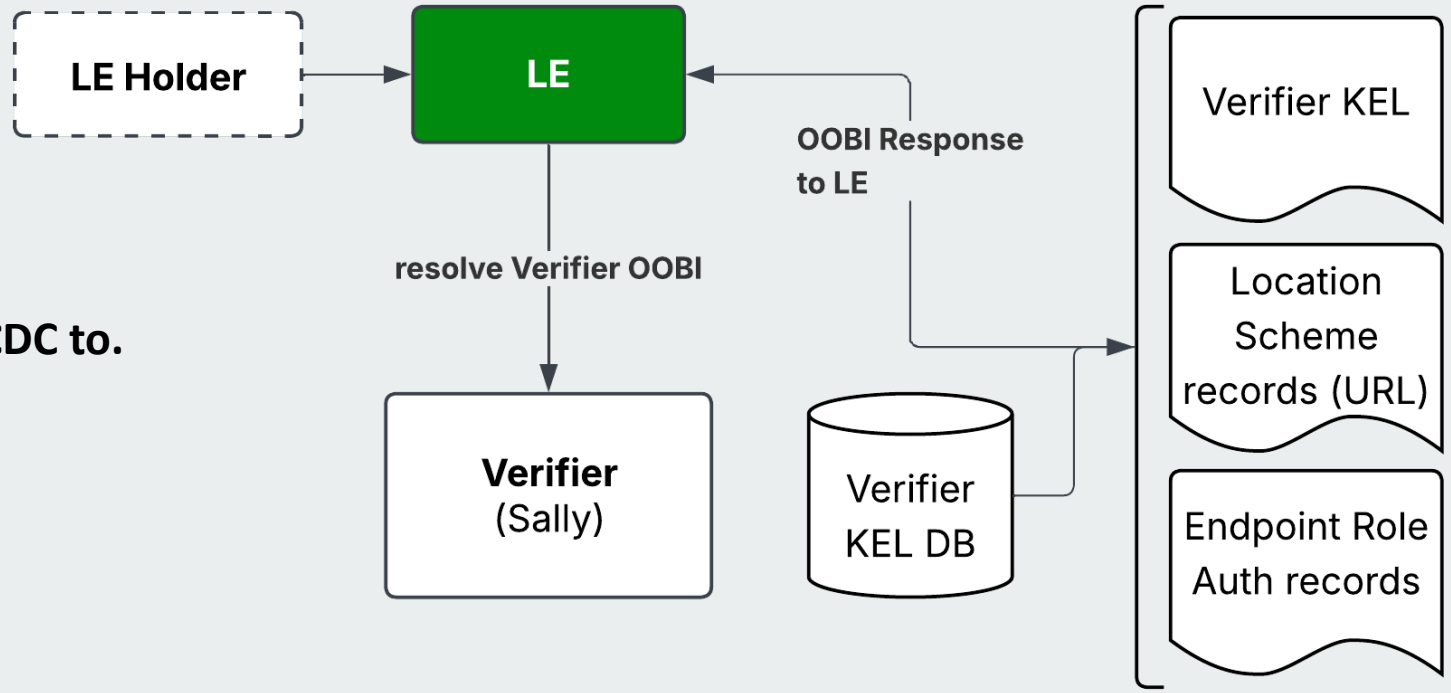
# Step 5 part 7/8: le/le-oobi-resolve-verifier.sh - Legal Entity connects to the Verifier

**Connect to Verifier by resolving its OOBI**

**The LE KERIA Agent needs the KEL of the Verifier including:**
**- Location Scheme records**
**- Endpoint Role Authorization records**

**So that it can know where to present the LE ACDC to.**

**Presentations in KERI occur from AID to AID.**

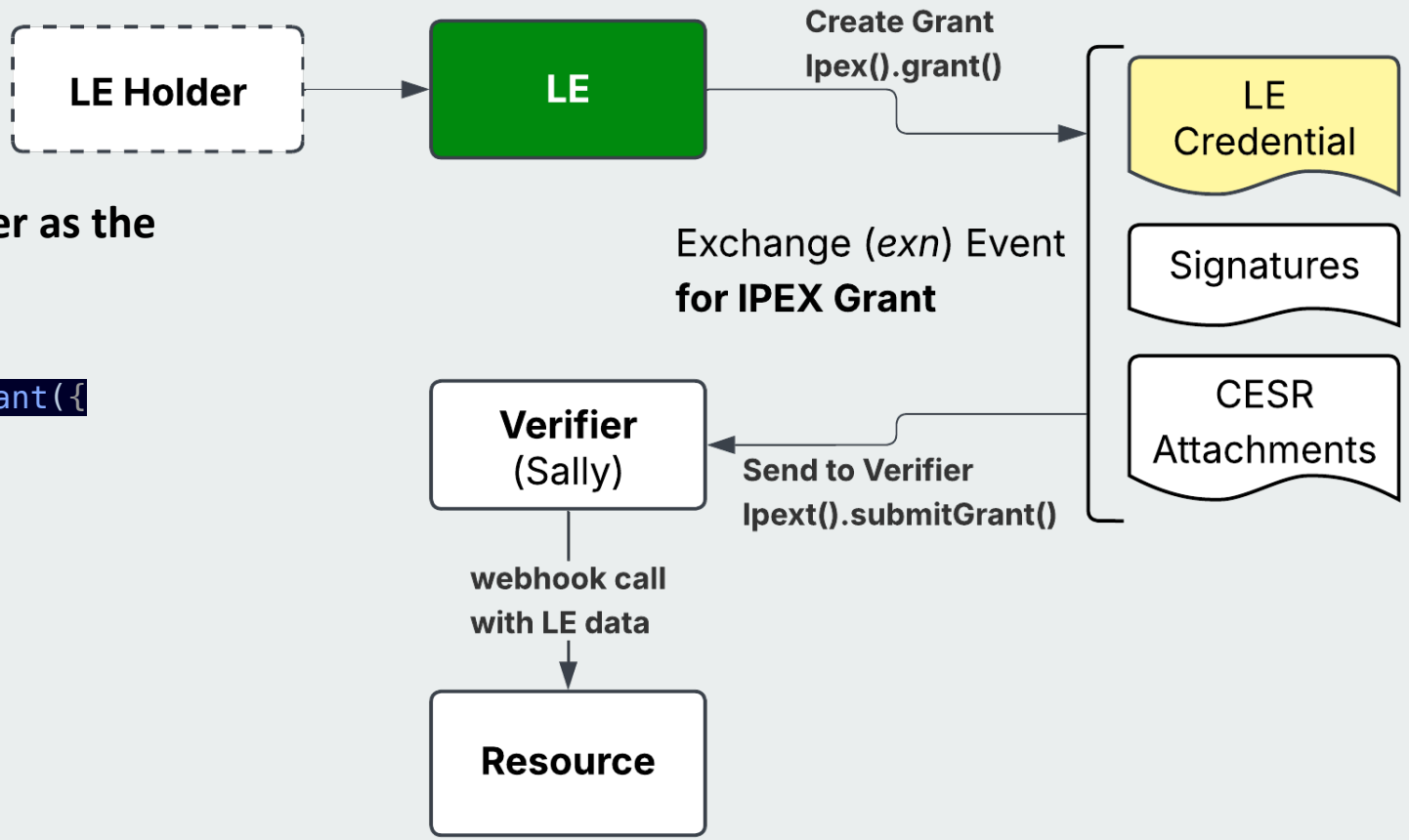


# Step 5 part 8/8: le/le-acdc-present-le.sh - Legal Entity presents LE ACDC to Verifier

**Present the LE credential to the Verifier.**

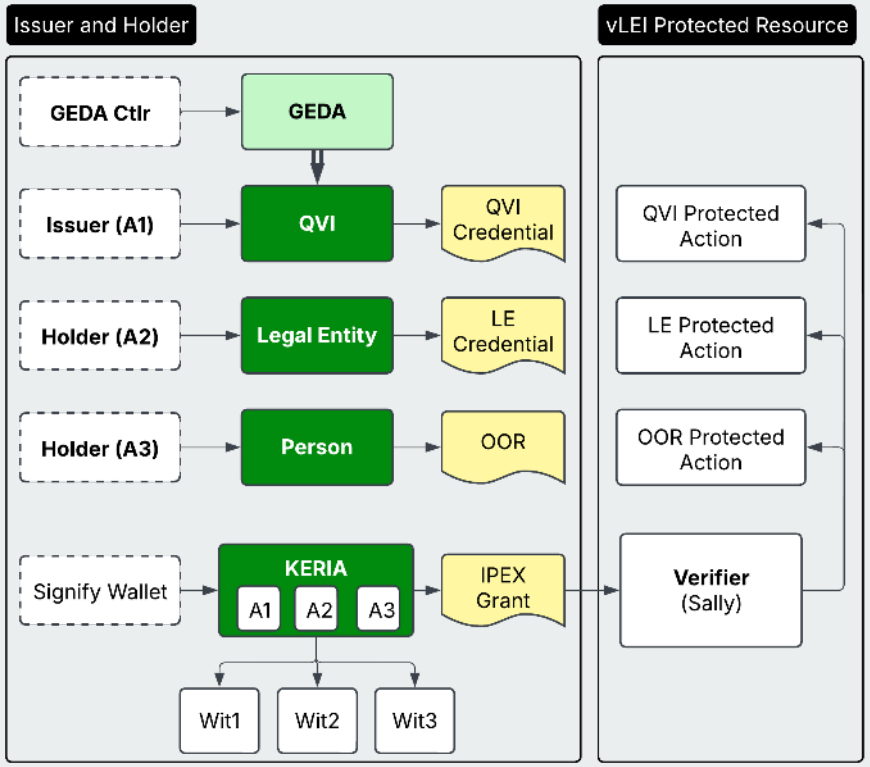
**This uses an IPEX Grant operation with the verifier as the recipient.**

```
const [grant, gsigs, end] = await client.ipex().grant({
            senderName: issName,
            recipient: issueeAid,
            datetime: dt,
            acdc: acdc,
            anc: anc,
            iss: iss,
        });

    await client
        .ipex()
        .submitGrant(issName, grant, gsigs, end,
[issueeAid]);
```



# Issuance and presentation of the Official Organizational Role credential

# Step 6: Person AID setup, OOR ACDC issuance and presentation




**Identifier Setup:**

person/person-aid-create.sh

person/person-oobi-resolve-le.sh

le/le-oobi-resolve-person.sh

qvi/qvi-oobi-resolve-person.sh

person/person-oobi-resolve-qvi.sh

person/person-oobi-resolve-verifier.sh

**For OOR Auth :**

le/le-registry-create.sh

- Create LE's registry for OOR Auth ACDC credentials

le/le-acdc-issue-oor-auth.sh

- Issue and send the OOR Auth credential from LE

qvi/qvi-acdc-admit-oor-auth.sh

- Admit the OOR Auth credential as the QVI

**For OOR:**

qvi/qvi-acdc-issue-oor.sh

- QVI issues the OOR credential to the Person

person/person-acdc-admit-oor.sh

- Admit the OOR credential as the person.

**For Presentation:**

person/person-acdc-present-oor.sh

- Present the OOR credential to the Verifier

# OOR Auth & OOR Credential Issuance

Creating and issuing the

- OOR Auth credential from the LE to the QVI
- OOR credential from the QVI to the Person

verifiable Legal Entity Identifier

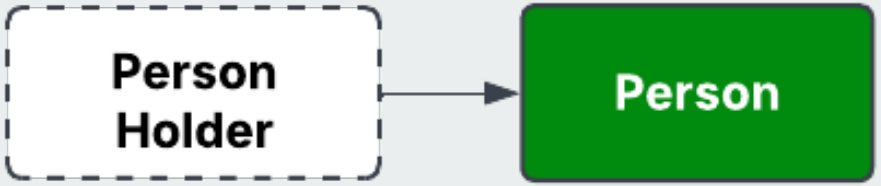
**PROTECTED**

# Step 6 part 1/12: person/person-aid-create.sh - create AID for the Person

- **Creates a single signature identifier (AID) for the person**

```
const client = await getOrCreateClient(personPasscode, env);
const leInfo: any = await createAid(client, 'person');
```

```
...
export async function createAid(client: SignifyClient, name: string) {
...
client.identifiers().create(name);
...
client.identifiers()
        .addEndRole(name, 'agent', client!.agent!.pre);
```



# Note for the end:

**Get the person AID from the step where you executed ./task-scripts/person/person-aid-create.sh**

**The console output looks like the following:**

```
Creating Person AID using SignifyTS and KERIA
Person info written to /task-data/person-*
   Prefix: ED7M2I8Zkk4DUHfxr_tihdovYdKfeHbcJ0gtpvI56NEZ
   OOBI: http://keria:3902/oobi/ED7M…
```

**The AID in this case would be:**
**ED7M2I8Zkk4DUHfxr_tihdovYdKfeHbcJ0gtpvI56NEZ**

**Save your Person AID for later in a note somewhere. We will use it to connect to the Chainlink demo.**

# Step 6 part 2/12: person/person-oobi-resolve-le.sh - Person gets LE key state

**Person resolves LE OOBI to**
**1) See where on the internet to send messages to the LE**

**This is essential so that the LE can:**
**- communicate with the LE during OOR Auth credential issuance**

**Endpoint Role Authorization records:**

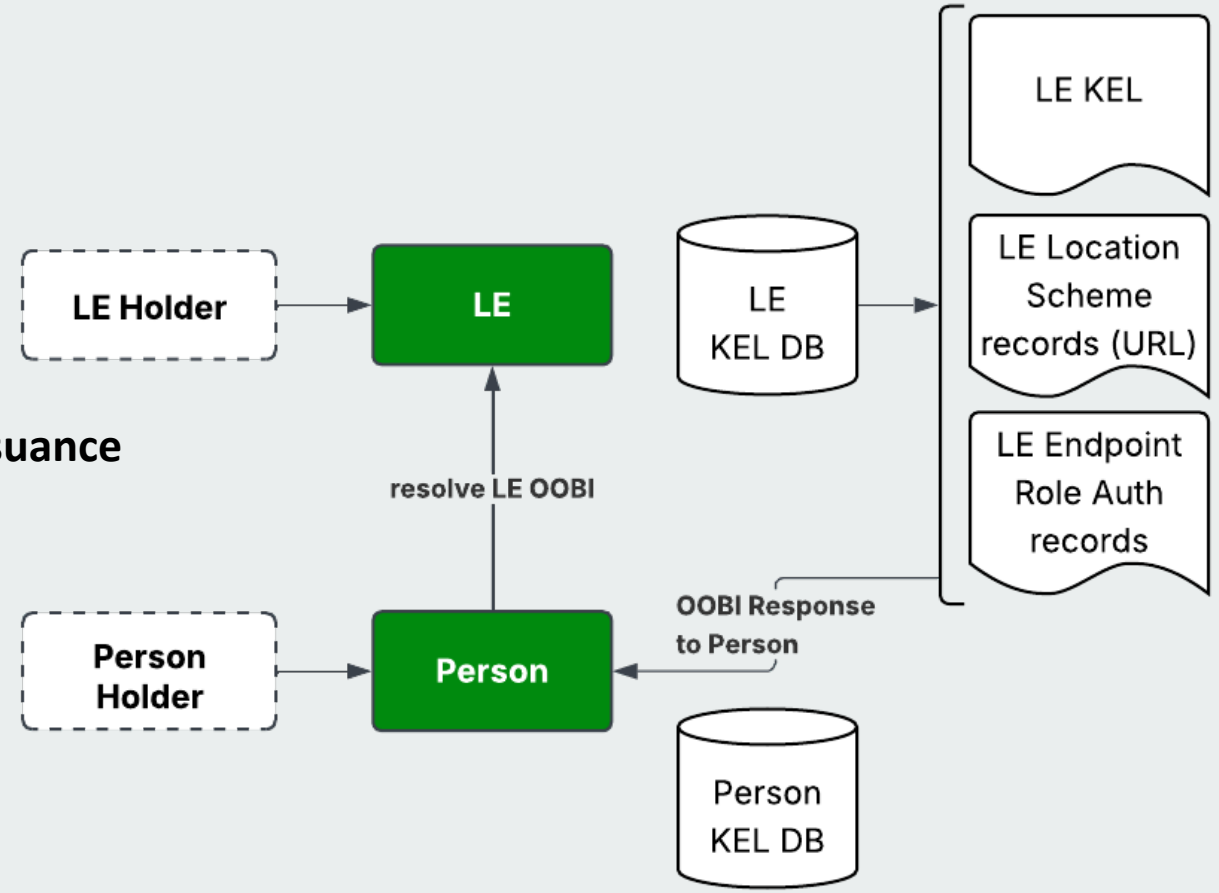
**- designate an AID as authorized to perform a given role for an AID such as an agent, mailbox, or witness**

**- in this case authorize the KERIA agent to act as a mailbox (comms relay) for the LE AID.**

**Location Scheme records:**

**- show the public URLs an AID declares it can receive communications on**

**- usually paired with the infrastructure authorized by an End Role Auth**



# Step 6 part 3/12: le/le-oobi-resolve-person.sh - LE gets Person key state, URL, roles,

**LE resolves Person OOBI to**
**1) See where on the internet to send messages to the Person**

**This is essential so that the LE can:**
**- communicate with the Person during OOR Auth credential issuance**

**Endpoint Role Authorization records:**

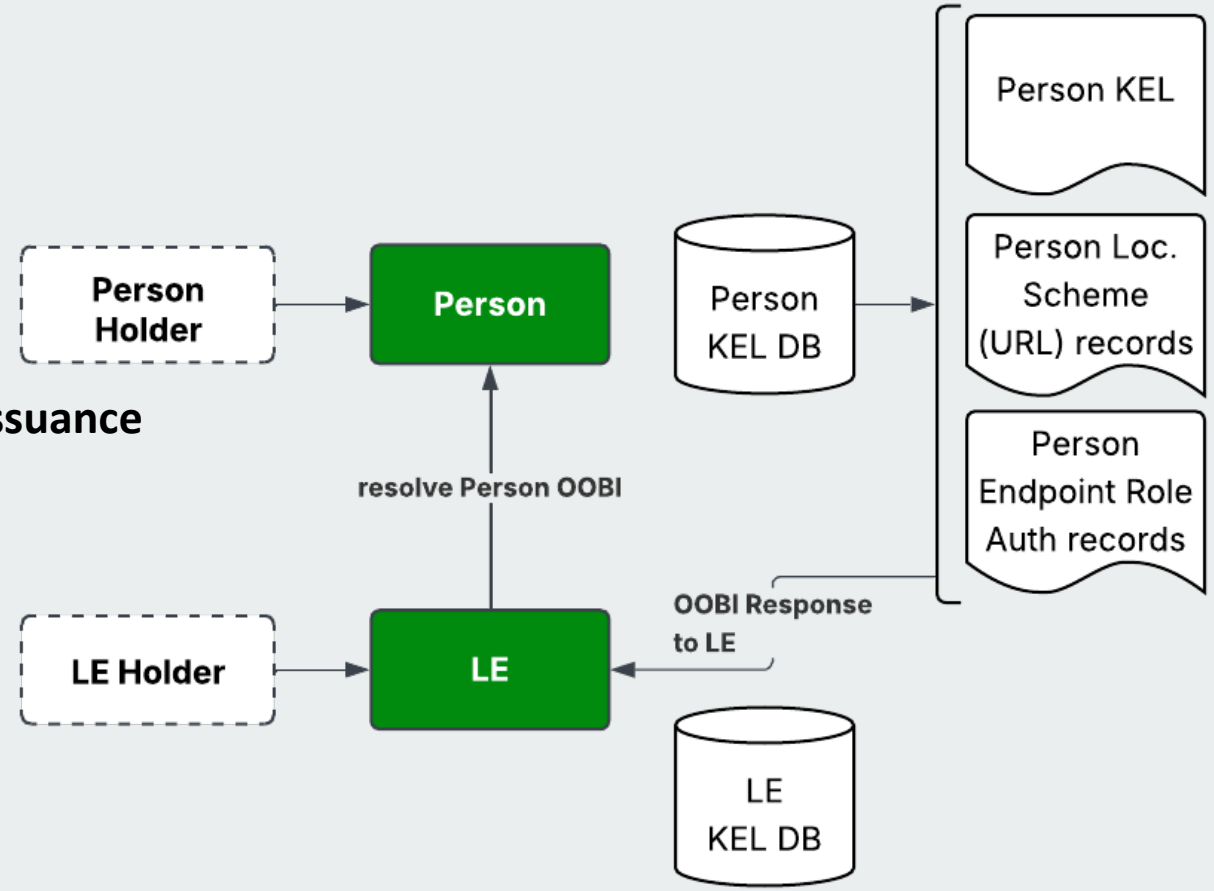
**- designate an AID as authorized to perform a given role for an AID such as an agent, mailbox, or witness**

**- in this case authorize the KERIA agent to act as a mailbox (comms relay) for the Person AID.**

**Location Scheme records:**

**- show the public URLs an AID declares it can receive communications on**

**- usually paired with the infrastructure authorized by an End Role Auth**



# Step 6 part 4/12: qvi/qvi-oobi-resolve-person.sh - QVI gets Person key state, URL, roles,

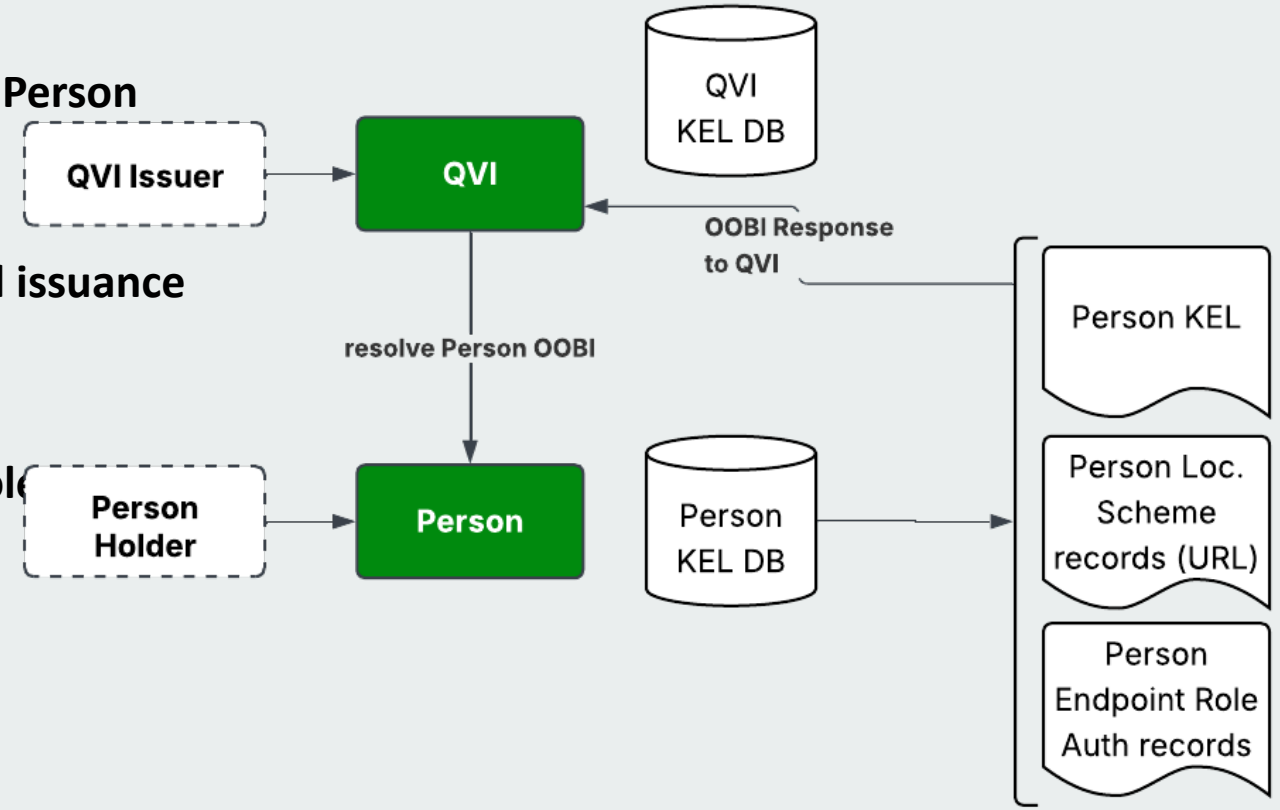
**QVI resolves Person OOBI to**
**1) See where on the internet to send messages to the Person**

**This is essential so that the QVI can:**
**- communicate with the Person during OOR credential issuance**

**Endpoint Role Authorization records:**

**- designate an AID as authorized to perform a given role for an AID such as an agent, mailbox, or witness**

**- in this case authorize the KERIA agent to act as a mailbox (comms relay) for the Person AID.**




**Location Scheme records:**

**- show the public URLs an AID declares it can receive communications on**

**- usually paired with the infrastructure authorized by an End Role Auth**

# Step 6 part 5/12: person/person-oobi-resolve-qvi.sh - Person gets QVI key state

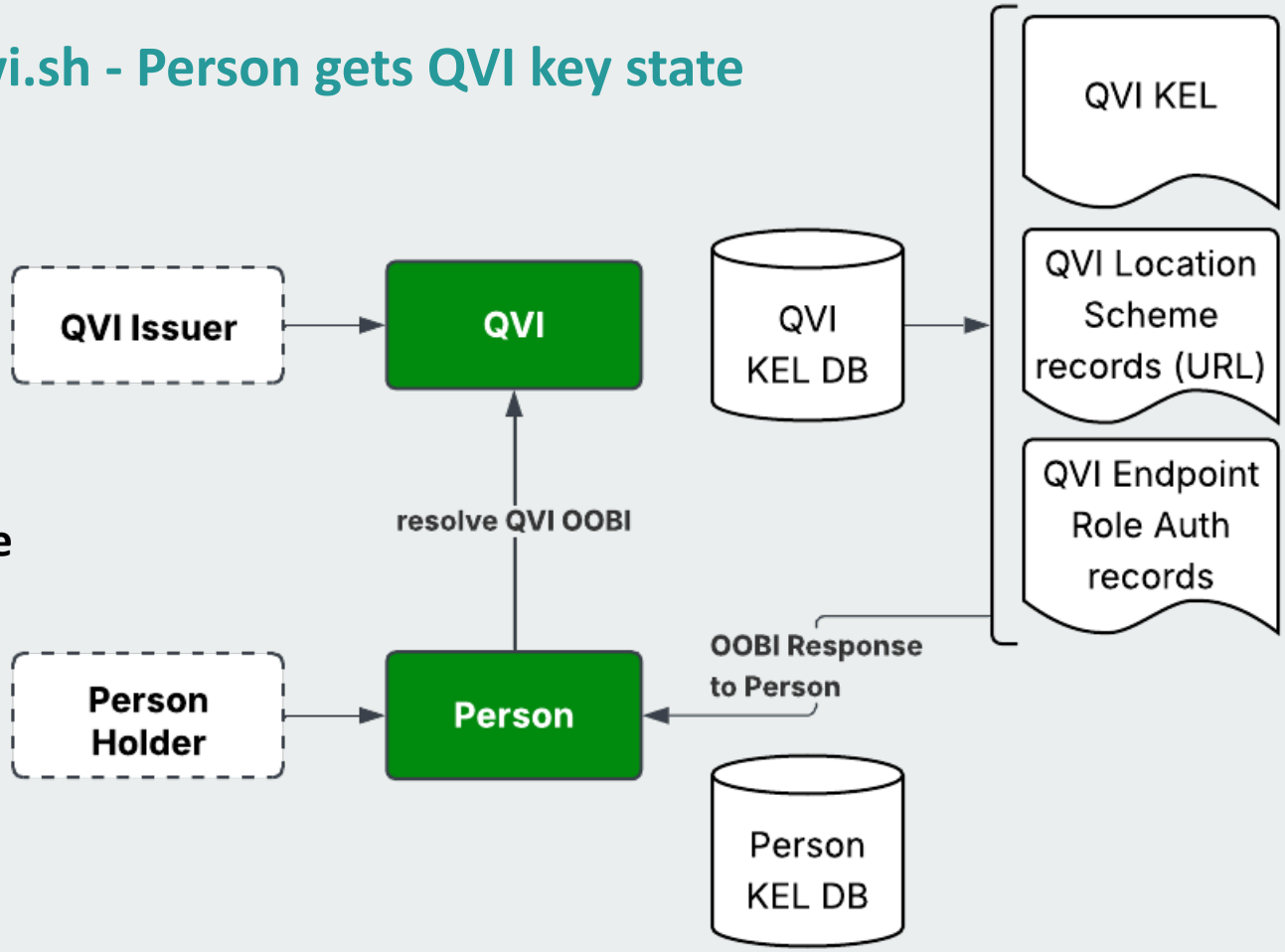
**Person resolves QVI OOBI to**
**1) See where on the internet to send messages to the QVI**

**This is essential so that the Person can:**
**- communicate with the QVI during OOR credential issuance**

**Endpoint Role Authorization records:**

**- designate an AID as authorized to perform a given role for an AID such as an agent, mailbox, or witness**

**- in this case authorize the KERIA agent to act as a mailbox (comms relay) for the QVI AID.**




**Location Scheme records:**

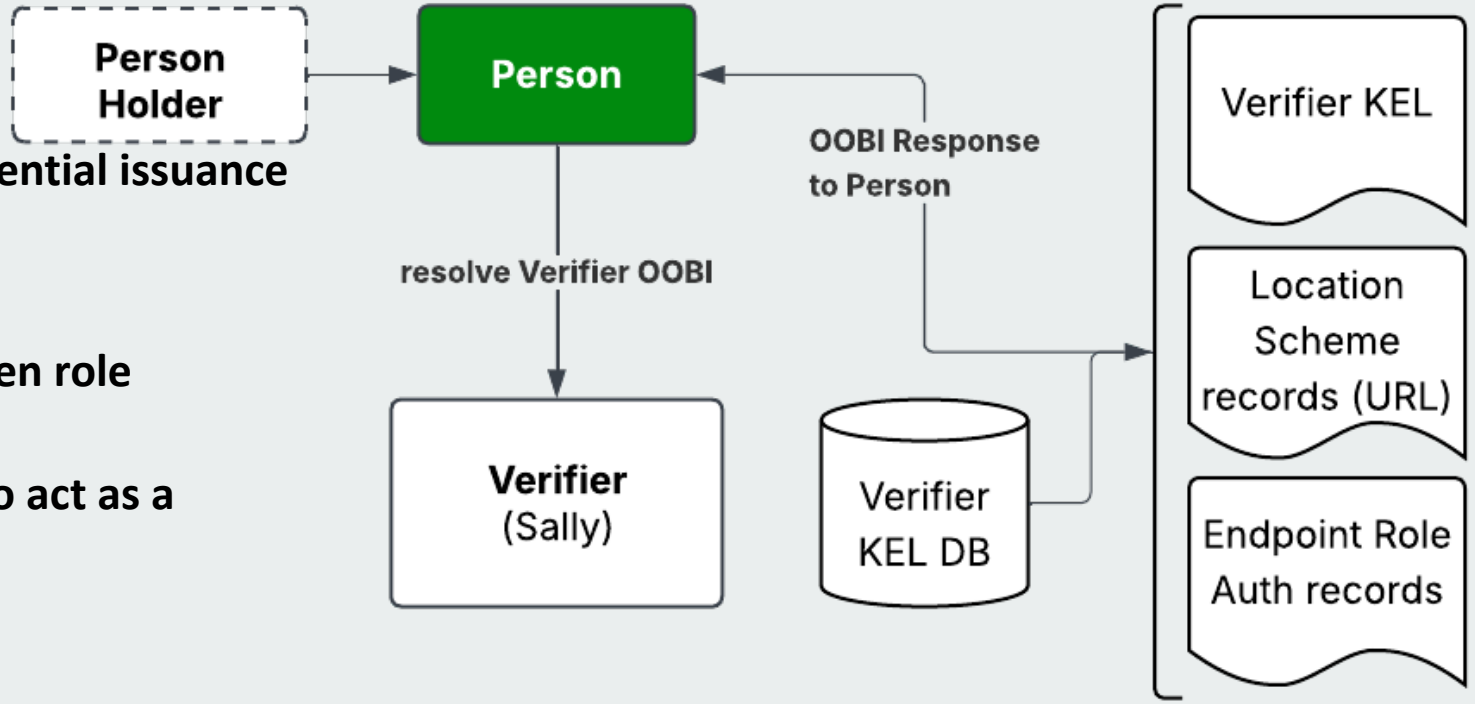
**- show the public URLs an AID declares it can receive communications on**

**- usually paired with the infrastructure authorized by an End Role Auth**

# Step 6 part 6/12: person/person-oobi-resolve-verifier.sh - Person gets Verifier key state

**Person resolves Verifier OOBI to**
**1) See where on the internet to send messages to the Verifier**

**This is essential so that the Person can:**
**- communicate with the Verifier during OOR credential issuance**




**Endpoint Role Authorization records:**

**- designate an AID as authorized to perform a given role for an AID such as an agent, mailbox, or witness**

**- in this case authorize the Verifier service itself to act as a mailbox (comms relay) for the Verifier AID.**

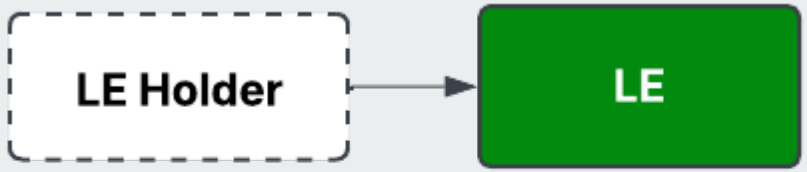
**Location Scheme records:**

**- show the public URLs an AID declares it can receive communications on**

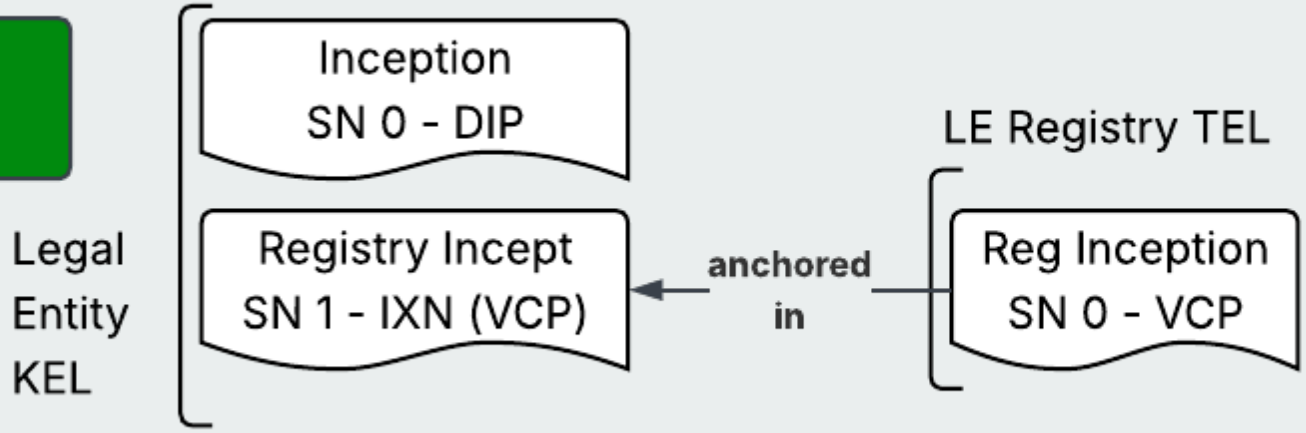
**- usually paired with the infrastructure authorized by an End Role Auth**

# Step 6 part 7/12: le/le-registry-create.sh - Create the Legal Entity's ACDC registry

**Create LE's ACDC registry for OOR Auth ACDC credentials**



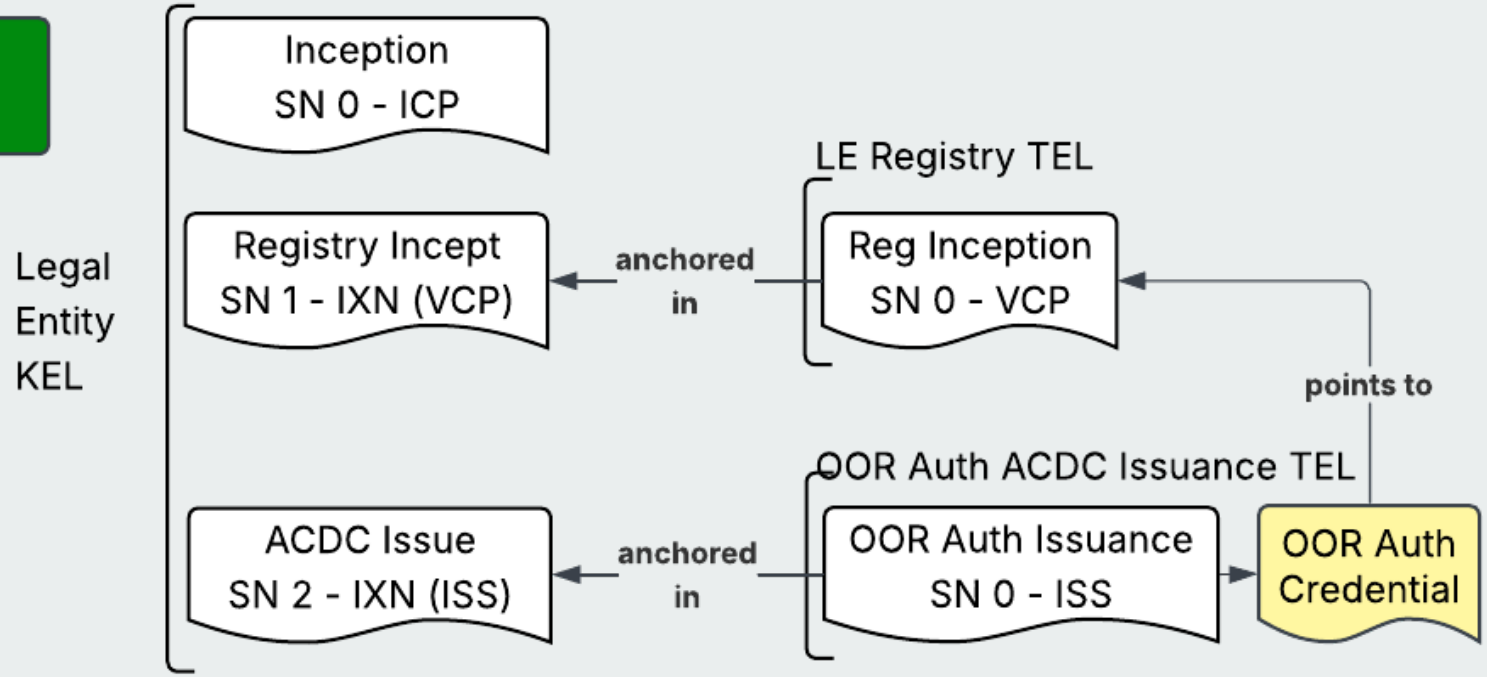

```
await client
            .registries()
            .create({ name: aidName, registryName:
registryName });
```



# Step 6 part 8/12: le/le-acdc-issue-oor-auth.sh Part 1 - Create OOR Auth ACDC credential

## Issue the OOR Auth credential from LE


```
    const subject: CredentialSubject = {
        i: hldPrefix,
        dt: createTimestamp(),
        ...credData,
    };
    const issData: CredentialData = {
        i: issAid.prefix,
        ri: issRegistry.regk,
        s: schema,
        a: subject,
        e: credEdge,
        r: credRules,
    };
    const issResult = await
        issClient.credentials().issue(issAidName, issData);
```



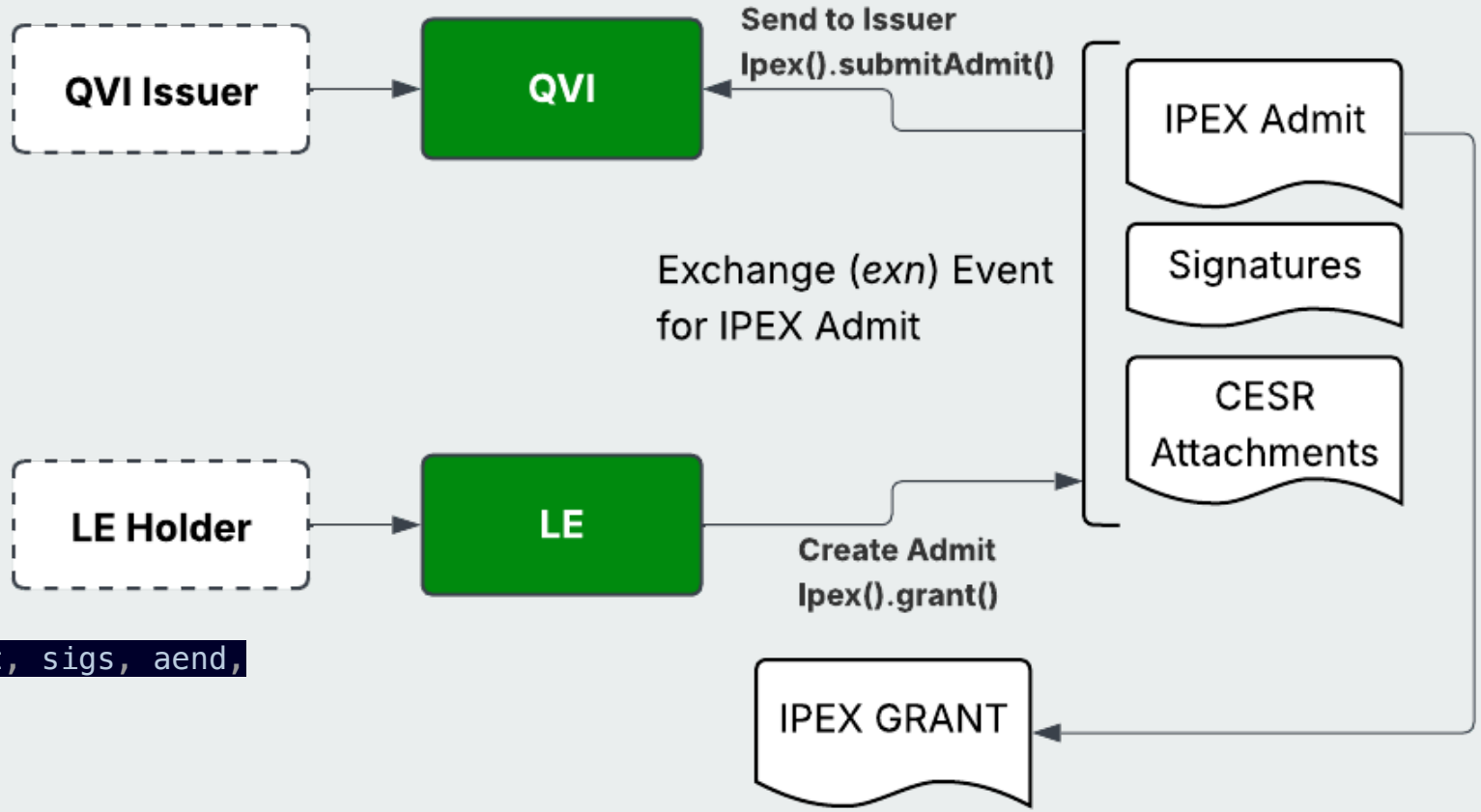
# Step 6 part 9/12: qvi/qvi-acdc-admit-oor-auth.sh - QVI admits OOR Auth ACDC

**Admit the OOR Auth credential as the QVI (IPEX Admit)**

**The IPEX Admit points back to**

**the IPEX Grant.**

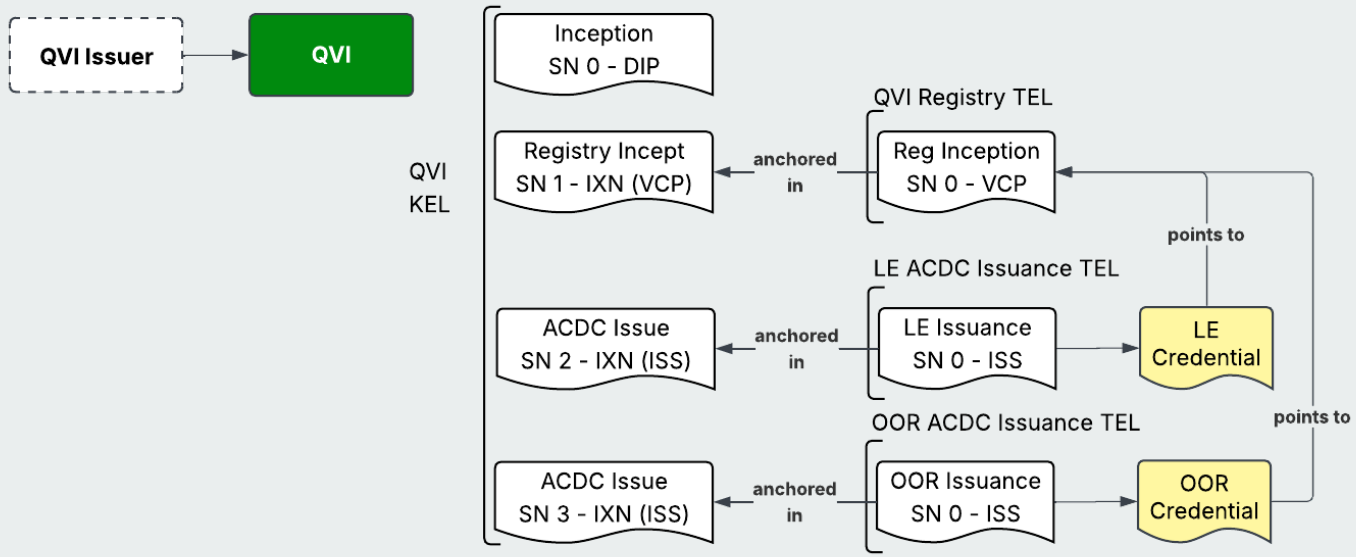
```
await client.ipex().admit({
            senderName: senderAidAlias,
            message: message,
            grantSaid: grantSaid,
            recipient: recipientAidPrefix,
            datetime: createTimestamp(),
        });

await client
            .ipex()
            .submitAdmit(senderAidAlias, admit, sigs, aend,
[recipientAidPrefix]);
```



# Step 6 part 10/12: qvi/qvi-acdc-issue-oor.sh - QVI issues the OOR to the Person and IPEX Grants

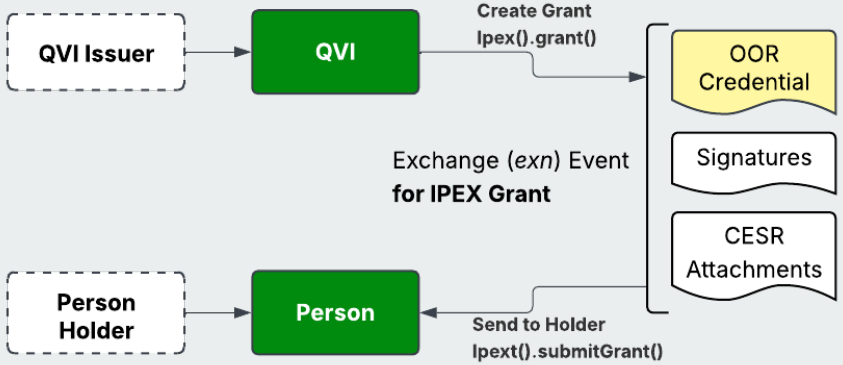
## Issue the OOR credential from the QVI to the Person

```
    const subject: CredentialSubject = {
        i: hldPrefix,
        dt: createTimestamp(),
        ...credData,
    };
    const issData: CredentialData = {
        i: issAid.prefix,
        ri: issRegistry.regk,
        s: schema,
        a: subject,
        e: credEdge,
        r: credRules,
    };
    const issResult = await
        issClient.credentials().issue(issAidName, issData);
```




## Send the OOR ACDC to the Person with IPEX Grant

```
const [grant, gsigs, end] = await client.ipex().grant({
            senderName: issName,
            recipient: issueeAid,
            datetime: dt,
            acdc: acdc,
            anc: anc,
            iss: iss,
        });

    await client
        .ipex()
        .submitGrant(issName, grant, gsigs, end, [issueeAid]);
```



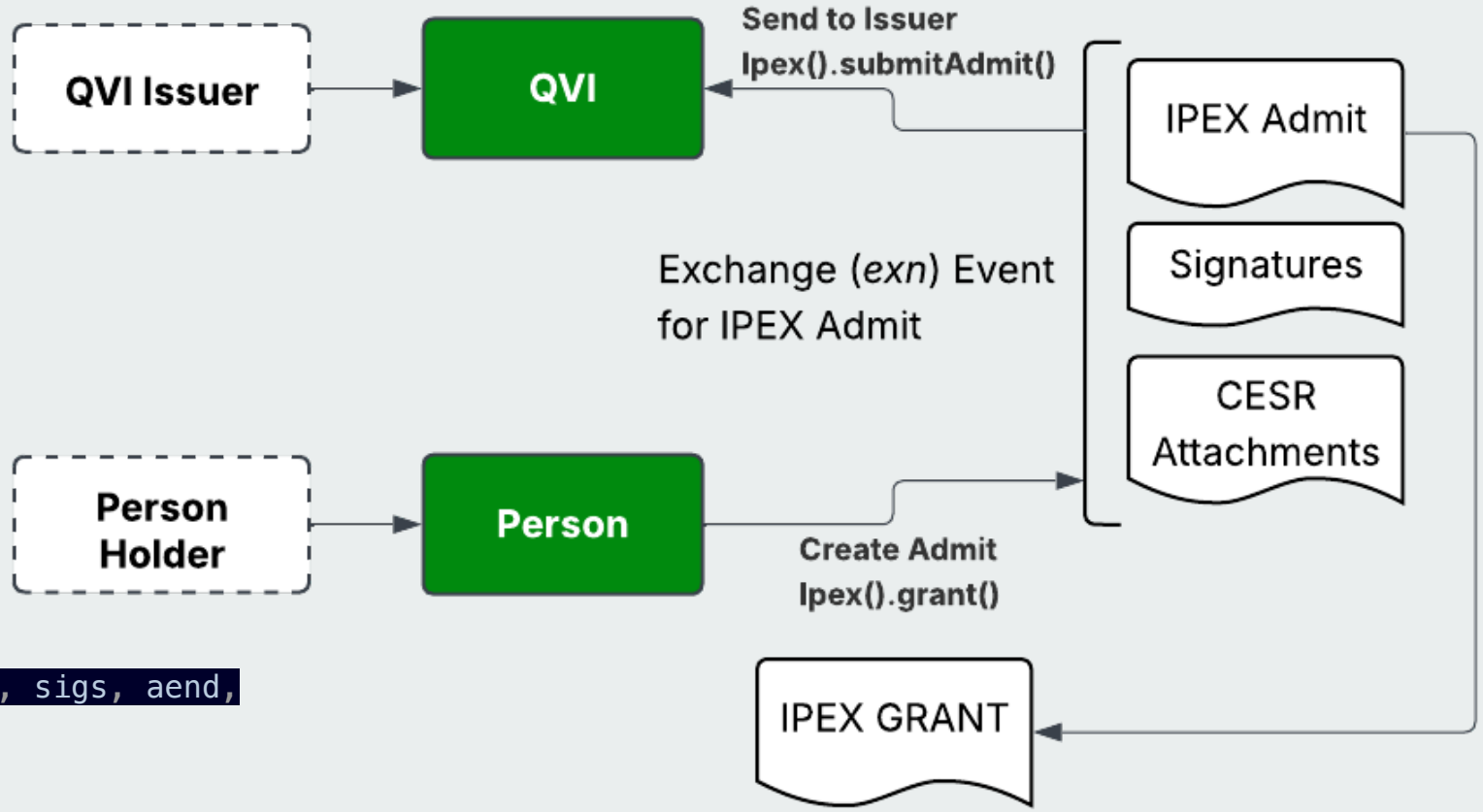
# Step 6 part 11/12: person/person-acdc-admit-oor.sh - Person admits OOR ACDC

**Admit the OOR credential as the Person (IPEX Admit)**

**The IPEX Admit points back to**

**the IPEX Grant.**

```
await client.ipex().admit({
            senderName: senderAidAlias,
            message: message,
            grantSaid: grantSaid,
            recipient: recipientAidPrefix,
            datetime: createTimestamp(),
        });

await client
            .ipex()
            .submitAdmit(senderAidAlias, admit, sigs, aend,
[recipientAidPrefix]);
```



# OOR Credential Presentation

Presenting the OOR credential from the Person to the Verifier

verifiable Legal Entity Identifier

**PROTECTED**

# Step 6 part ??/12: person/person-oobi-resolve-verifier.sh - Person connects to the Verifier

**This was already done as a part of setup!**

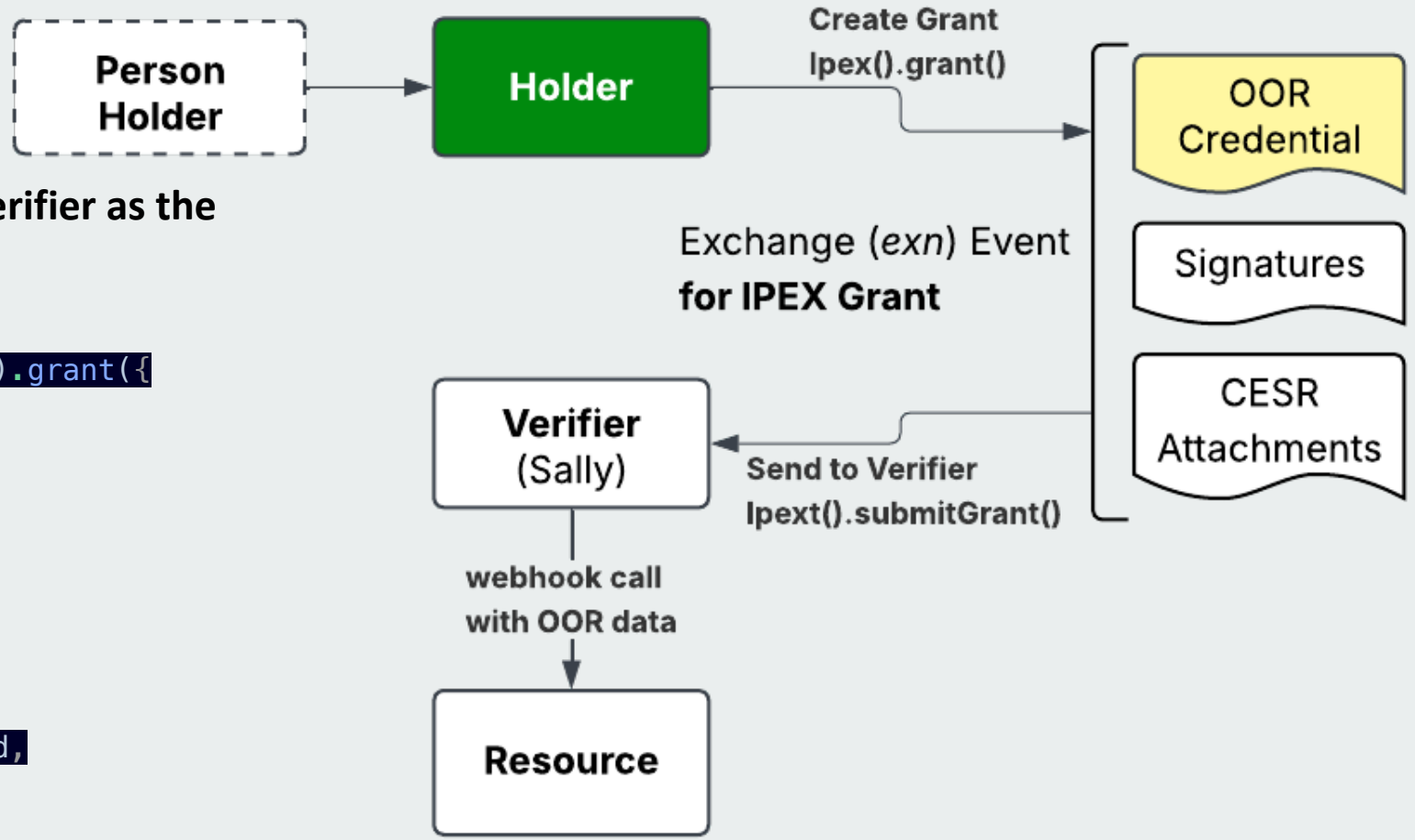
**Move along.**

# Step 6 part 12/12: person/person-acdc-present-oor.sh - Person presents OOR to Verifier

**Present the QVI credential to the Verifier.**

**This uses an IPEX Grant operation with the verifier as the recipient.**

```
const [grant, gsigs, end] = await client.ipex().grant({
            senderName: issName,
            recipient: issueeAid,
            datetime: dt,
            acdc: acdc,
            anc: anc,
            iss: iss,
        });

    await client
        .ipex()
        .submitGrant(issName, grant, gsigs, end,
[issueeAid]);
```



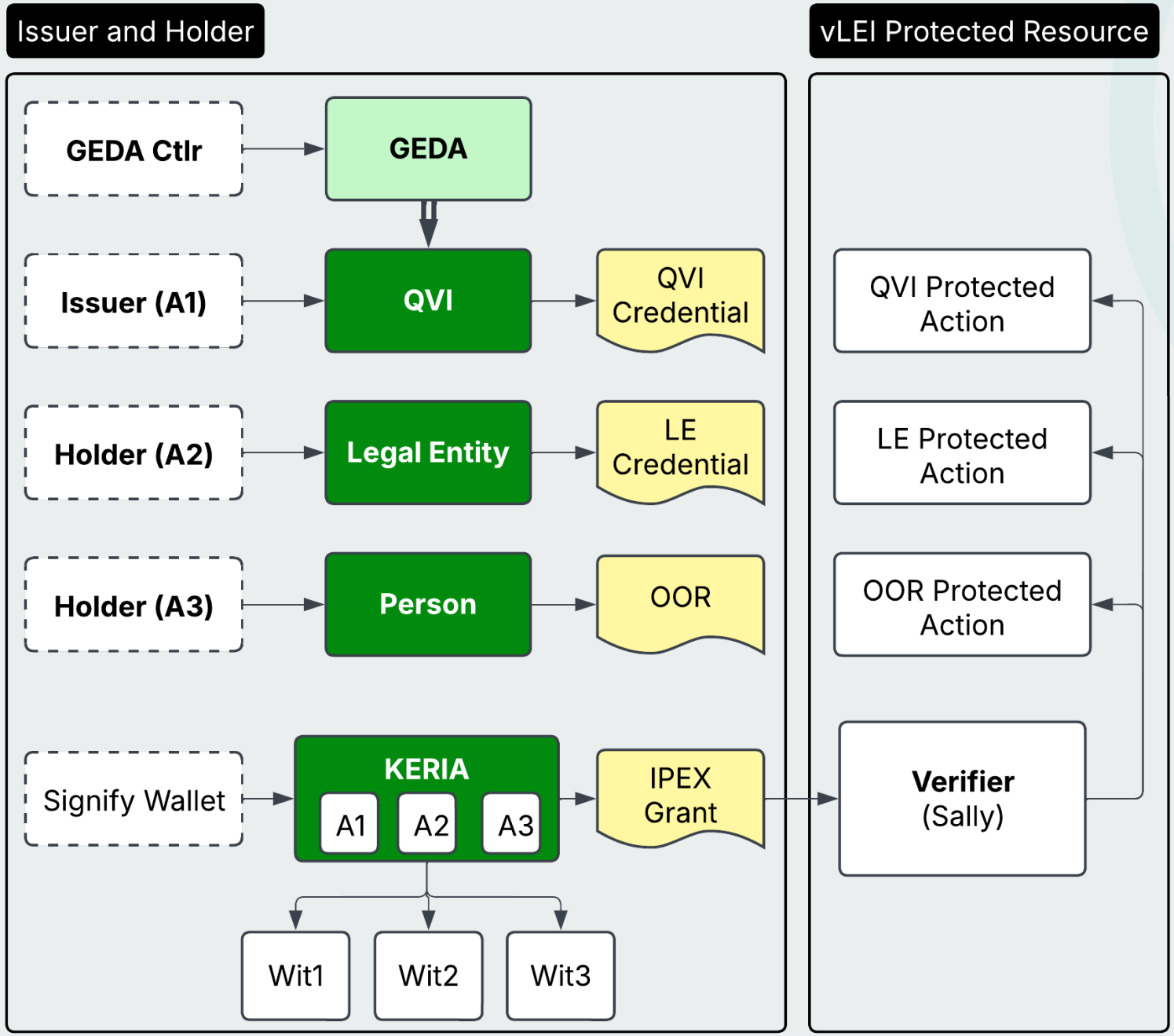
# What are the protected resources?

That is controlled by the Resource server in this instance.

Could be updating an ACL like in Chainlink's ACE.

Could be triggering some other ACDC dependent workflow.

# End Goal:


Issuer and Holder
vLEI Protected Resource
GEDA Ctlr
GEDA
Issuer (A1)
QVI
QVI Credential
Holder (A2)
Legal Entity
LE Credential
Holder (A3)
Person
OOR
Signify Wallet
KERIA
A1
A2
A3
IPEX Grant
Wit1
Wit2
Wit3
QVI Protected Action
LE Protected Action
OOR Protected Action
Verifier (Sally)

# REVIEW

- Created Identifiers
  - GEDA
  - QVI
  - Legal Entity
  - Person
- Created Credentials
  - QVI
  - LE
  - OOR Auth
  - OOR
- Presented Credentials to Verifier
    - QVI
    - LE
    - OOR
- Credentials triggered protected resource web hook calls

# Bonus: ECR Credential - Challenge for Audience

# Last Step - get your Person AID for Alice's CCID in the Chainlink module

**Get the person AID from the step where you executed**
**./task-scripts/person/person-aid-create.sh**

**The console output looks like the following:**

```
Creating Person AID using SignifyTS and KERIA
Person info written to /task-data/person-*
   Prefix: ED7M2I8Zkk4DUHfxr_tihdovYdKfeHbcJ0gtpvI56NEZ
   OOBI: http://keria:3902/oobi/ED7M…
```

**The AID in this case would be:**
**ED7M2I8Zkk4DUHfxr_tihdovYdKfeHbcJ0gtpvI56NEZ**

**In your .env file for your chainlink demo place this AID in Alice's CCID environment variable**

**export ALICE_CCID=`cast keccak "EFtkrGFObLcl6NHdfZBSnWnCjuca8pKzIj2mMw6TB8YJ"`**

# Workshop - Module 2 - vLEI-based smart contract access via CCID

- Outcome: Access a smart contract method with a CCID based on a vLEI
- Step 1: Infrastructure Setup
  - Witnesses
  - KERIA + Signify Headless Wallet
  - Verifier (Sally)
- Step 2: vLEI AID
- Step 3: Create CCID from vLEI AID
- Step 4: Access smart contract with CCID

- Link: https://github.com/smartcontractkit/chainlink-ace/blob/main/getting_started/advanced/GETTING_STARTED_ADVANCED.md

# Hackathon Feedback Gathering

- Specific Questions
- Was Testnet Helpful? If so, why?
- Were the vLEI Trainings helpful? How so?
- What was missing that would have been more helpful?
    - More or better docs?
    - Short guides or recipes?
    - Example Typescript wallet app?
    - Multisig or delegation guides?

# Hackathon Retrospective and Lessons Learned

- Start
- Stop
- Continue
- Lessons Learned about vLEI and KERI from your team’s experience

# Workshop End