M M S EXPERIENCE BEYOND DIGITAL

November 2022

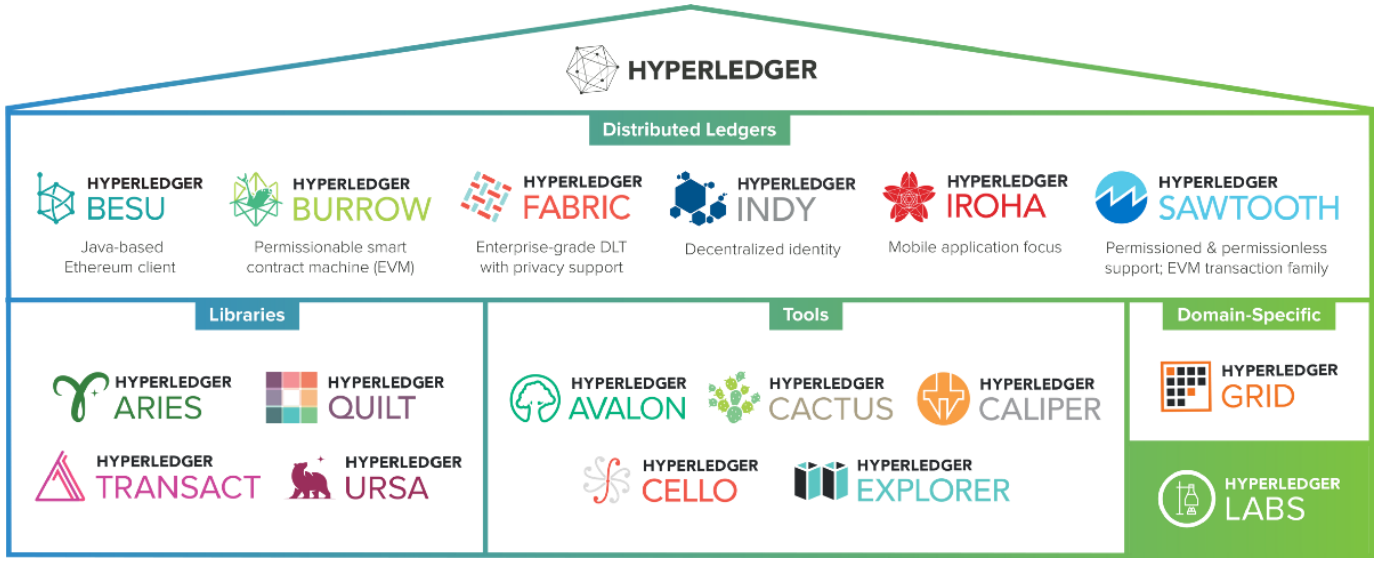
SSI Demo.

T Systems

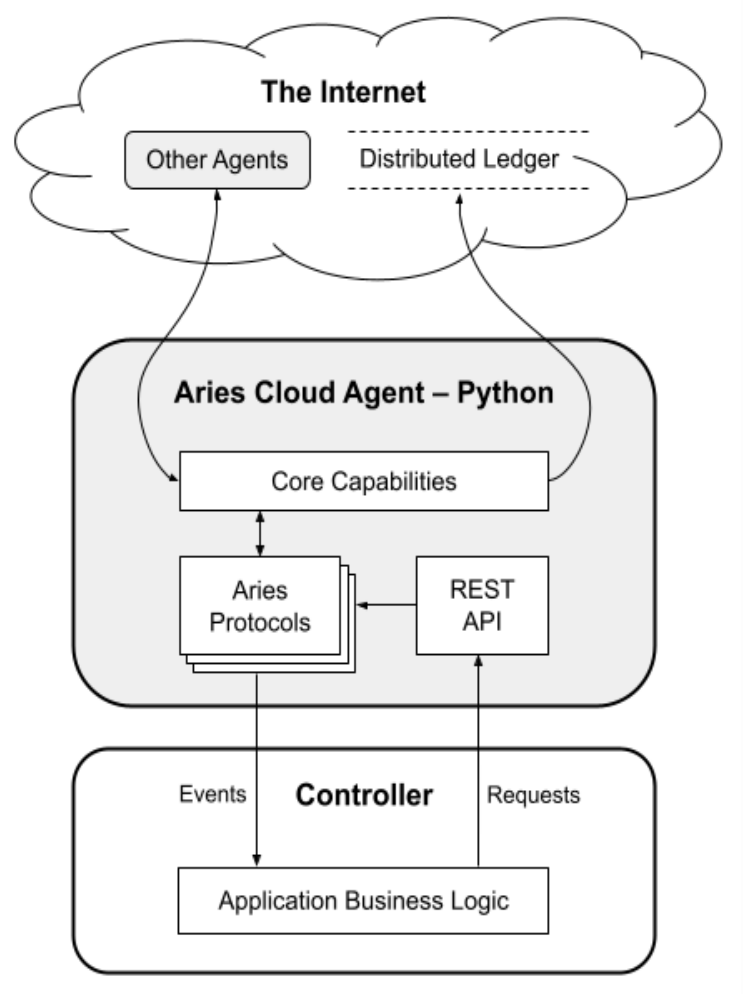
Danil Tolonbekov

The Hyperledger Frameworks and Tools

- **Hyperledger URSA**
  - a shared cryptographic library
- **Hyperledger INDY**
  - provides tools, libraries, and components for providing digital identities rooted on blockchains or other distributed ledgers
- **Hyperledger Aries**
  - creating, transmitting and storing verifiable digital credentials



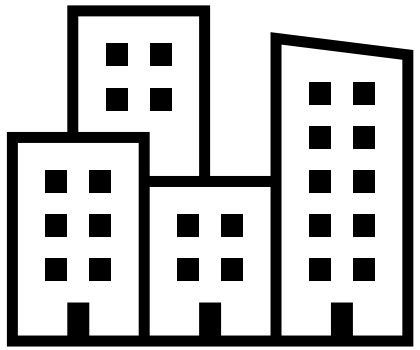
Understanding the Architecture[1]



- **Foundation for building Verifiable Credential (VC) ecosystems**
- **The "cloud" in the name means that ACA-Py runs on servers (cloud, enterprise, IoT devices, and so forth)**
- **Uses both Hyperledger Indy AnonCreds verifiable credential and the W3C Standard Verifiable Credential formats**

1 https://github.com/hyperledger/aries-cloudagent-python/tree/d78d4ea483e76c8033141e3c6c8e1a68e3a72096

Alice – Faber - ACME



Alice is a graduate student

Holder

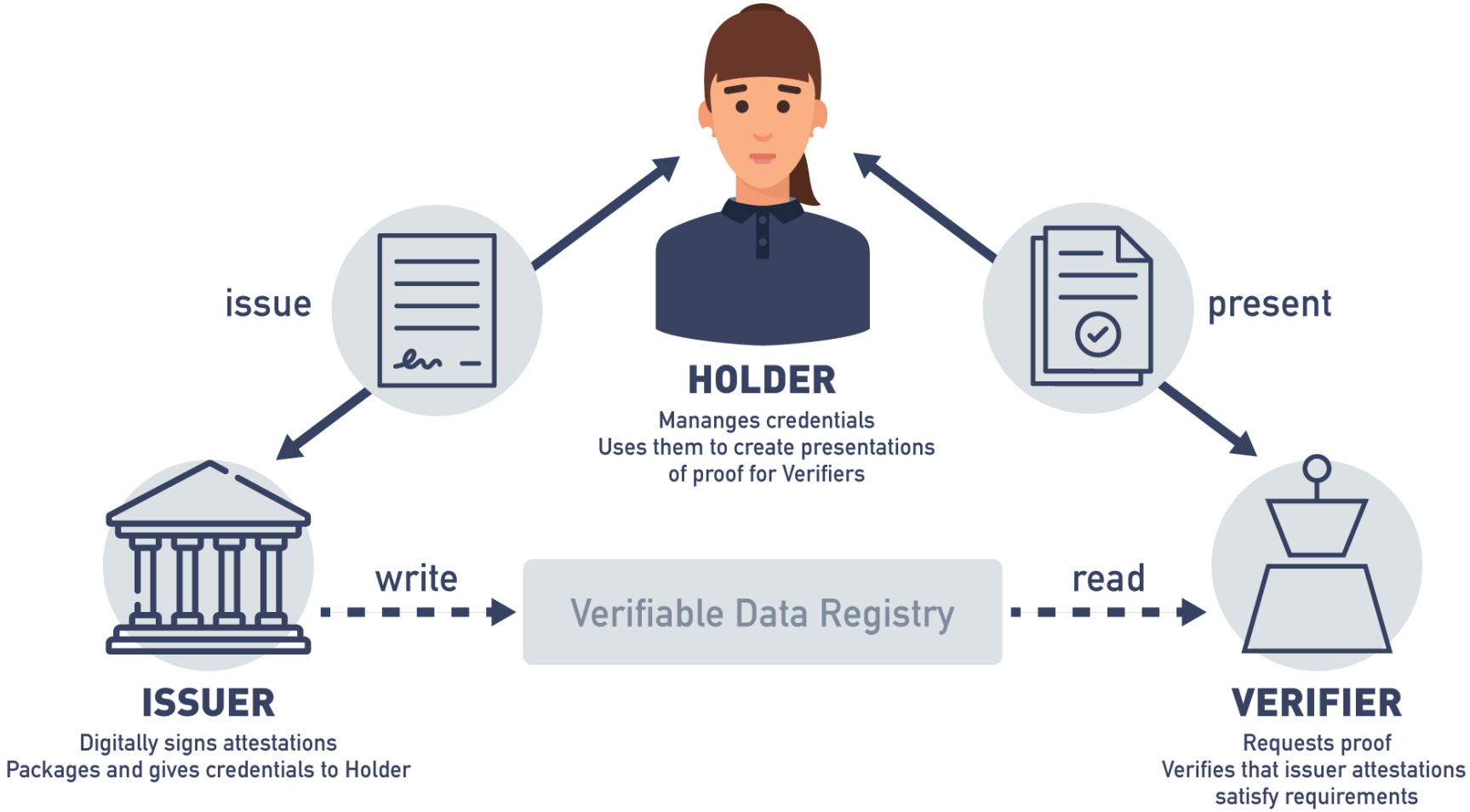
Faber College

Issuer

ACME is a company (MAANG)

Verifier

General workflow



Verifiable Data Registry

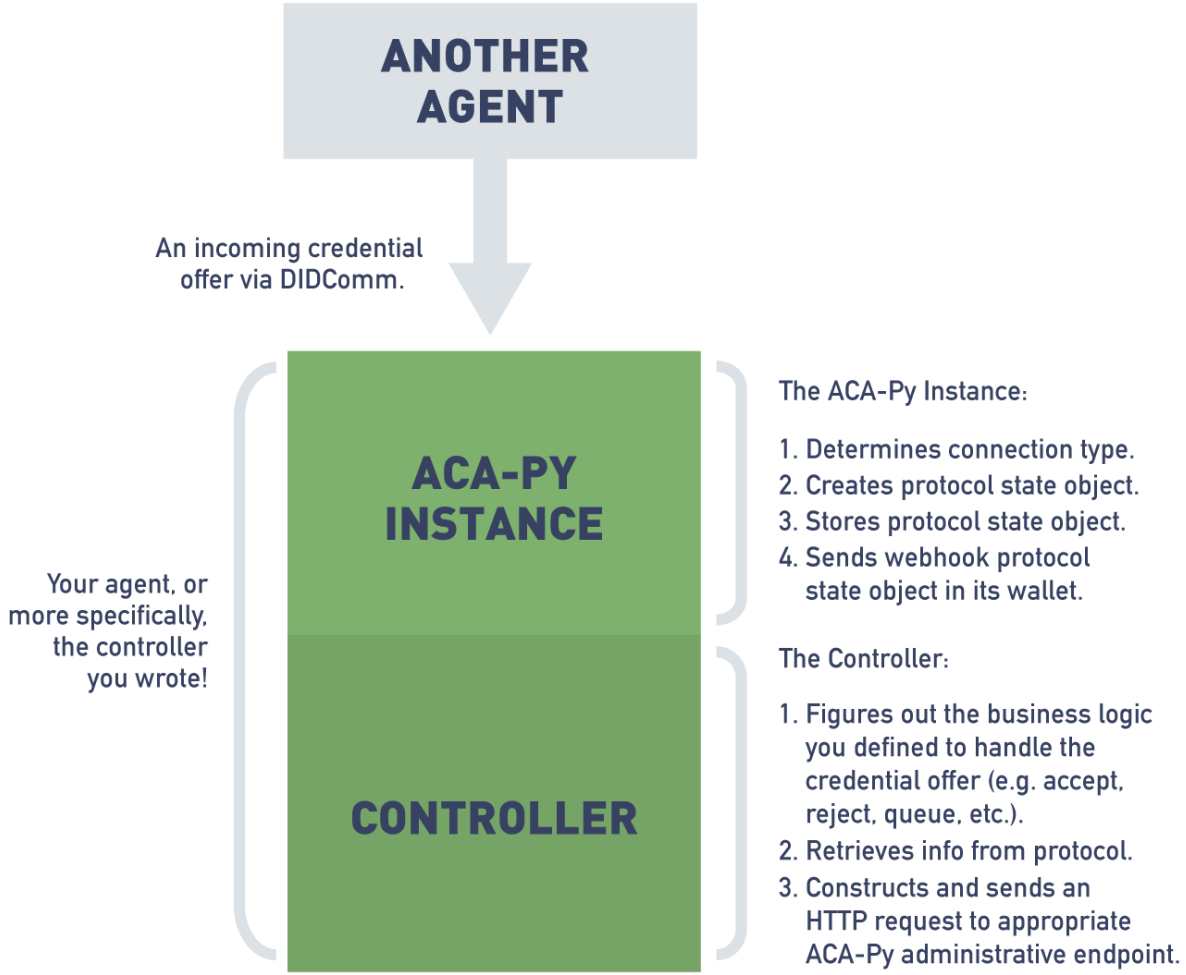
- **The primary purpose of the ledger is to be a place for a verifiable credential issuer to publish cryptographic keys and credential metadata so that a prover can produce a presentation that a verifier can cryptographically verify.**
- In theory, such information could be digitally published in other ways, but the attributes of a ledger are ideal for this purpose:
  - Data written to a distributed ledger (such as Indy) is immutable—it can't ever be changed.
  - Ledger data can't be removed
  - Multiple parties (that is, validators or miners) reach consensus on what is to be written to a ledger
  - The data is replicated across a set of independent parties and as such is highly available.

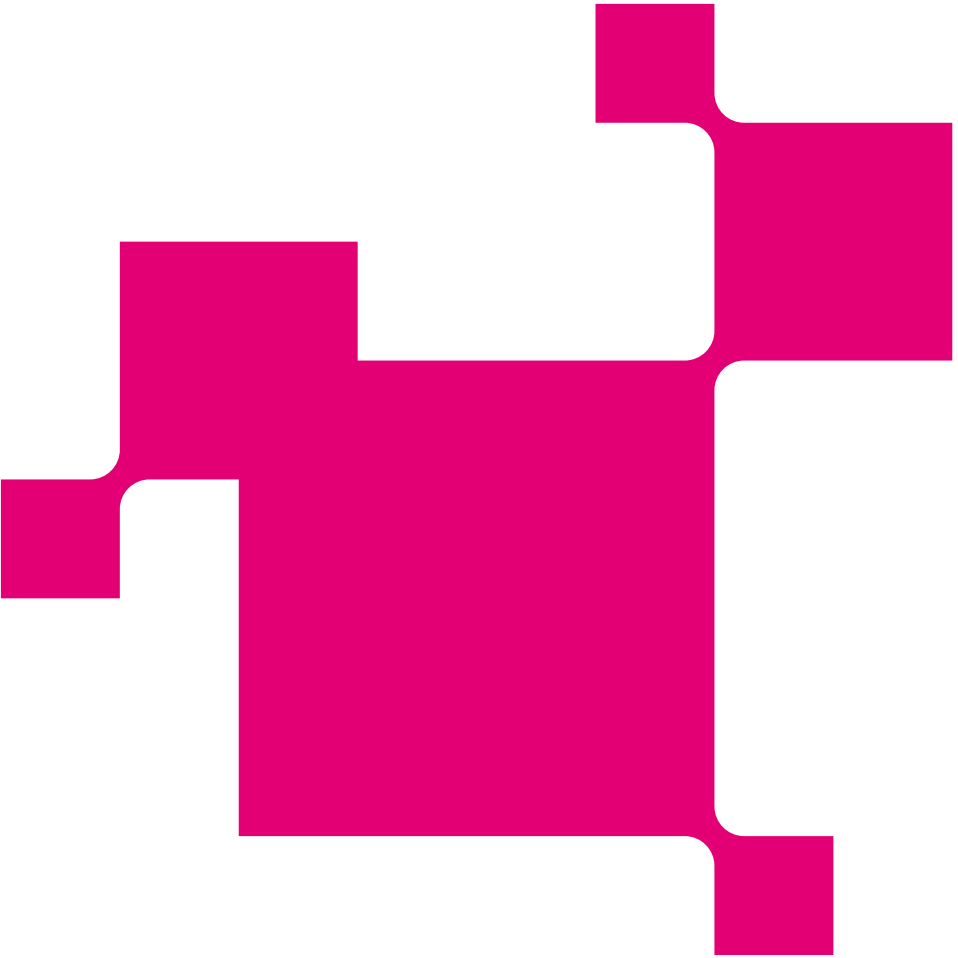
  - Genesis file (download, resolve)
  - test.bcovrin.vonx.io/genesis

Agent Start Up

… it needs to know:

- **The location of the genesis file(s) for the ledger(s) it will use (if any).**
  - --genesis-file <genesis-file>, ACAPY_GENESIS_FILE
- **If it needs objects (DIDs, schema, etc.) on the ledger, checking that they exist on ledger and in secure storage, and creating those objects if they don't exist.**
- **Transport (such as HTTP or web sockets) endpoints for messaging other agents.**
- **Storage options for keys and other data.**
- **Interface details between the agent framework and the controller for events and requests.**





DEMO

Issuer Initialization[1] (1)

- **Faber creates a wallet, with a Public DID as needed by a Verifiable Credential Issuer.**

```
created = await faber_agent.agent.register_or_switch_wallet(
    target_wallet_name,
    public_did=True,
    mediator_agent=faber_agent.mediator_agent,
)
```

- **In creating the Faber wallet, create Faber's DID**

```
if public_did:
    if cred_type == CRED_FORMAT_INDY:
        # assign public did
        new_did = await self.admin_POST("/wallet/did/create")
        self.did = new_did["result"]["did"]
        await self.register_did(
            did=new_did["result"]["did"], verkey=new_did["result"]["verkey"]
        )
        await self.admin_POST("/wallet/did/public?did=" + self.did)
```

1 https://github.com/hyperledger/aries-cloudagent-python/tree/d78d4ea483e76c8033141e3c6c8e1a68e3a72096/demo/runners

Issuer Initialization (2)

- **Call to self.seed to generate a random seed for the agent.**

```
rand_name = str(random.randint(100_000, 999_999))
self.seed = (
    ("my_seed_000000000000000000000000" + rand_name)[-32:]
    if seed == "random"
    else seed
)
```

- **Faber registers a schema and credential definition on the ledger.**

```
# create a schema and cred def for the new wallet
# TODO check first in case we are switching between existing wallets
if created:
    # TODO this fails because the new wallet doesn't get a public DID
    await faber_agent.create_schema_and_cred_def(
        schema_name=faber_schema_name,
        schema_attrs=faber_schema_attrs,
    )
```

Issuer Initialization (3)

```
# start the agents - faber gets a public DID and schema/cred def
await faber_container.initialize(
    schema_name="degree schema",
    schema_attrs=[
        "name",
        "date",
        "degree",
        "grade",
    ],
)
# Create a schema
schema_body = {
    "schema_name": schema_name,
    "schema_version": version,
    "attributes": schema_attrs,
}
schema_response = await self.admin_POST("/schemas", schema_body)
log_json(json.dumps(schema_response), label="Schema:")
schema_id = schema_response["schema_id"]
log_msg("Schema ID:", schema_id)
await asyncio.sleep(2.0)
```

- **Attributes in the schema Faber creates**

- **Method in agent.py to call ACA-Py to register the schema and cred def.**

Request From User to Issue Credential

- **Faber handles the request from the user to issue a credential**

```
offer_request = {
    "connection_id": faber_agent.agent.connection_id,
    "cred_def_id": faber_agent.cred_def_id,
    "comment": f"Offer on cred def id {faber_agent.cred_def_id}",
    "auto_remove": False,
    "credential_preview": cred_preview,
    "trace": exchange_tracing,
}
await faber_agent.agent.admin_POST(
    "/issue-credential/send-offer", offer_request
)
```

```
# define attributes to send for credential
faber_agent.agent.cred_attrs[faber_agent.cred_def_id] = {
    "name": "Alice Smith",
    "date": "2018-05-28",
    "degree": "Maths",
    "age": "24",
    "timestamp": str(int(time.time())),
}

cred_preview = {
    "@type": CRED_PREVIEW_TYPE,
    "attributes": [
        {"name": n, "value": v}
        for (n, v) in faber_agent.agent.cred_attrs[
            faber_agent.cred_def_id
        ].items()
    ],
}
```

Request From User to Send Proof Request

- **Faber handles the request from the user to request a proof from Alice.**

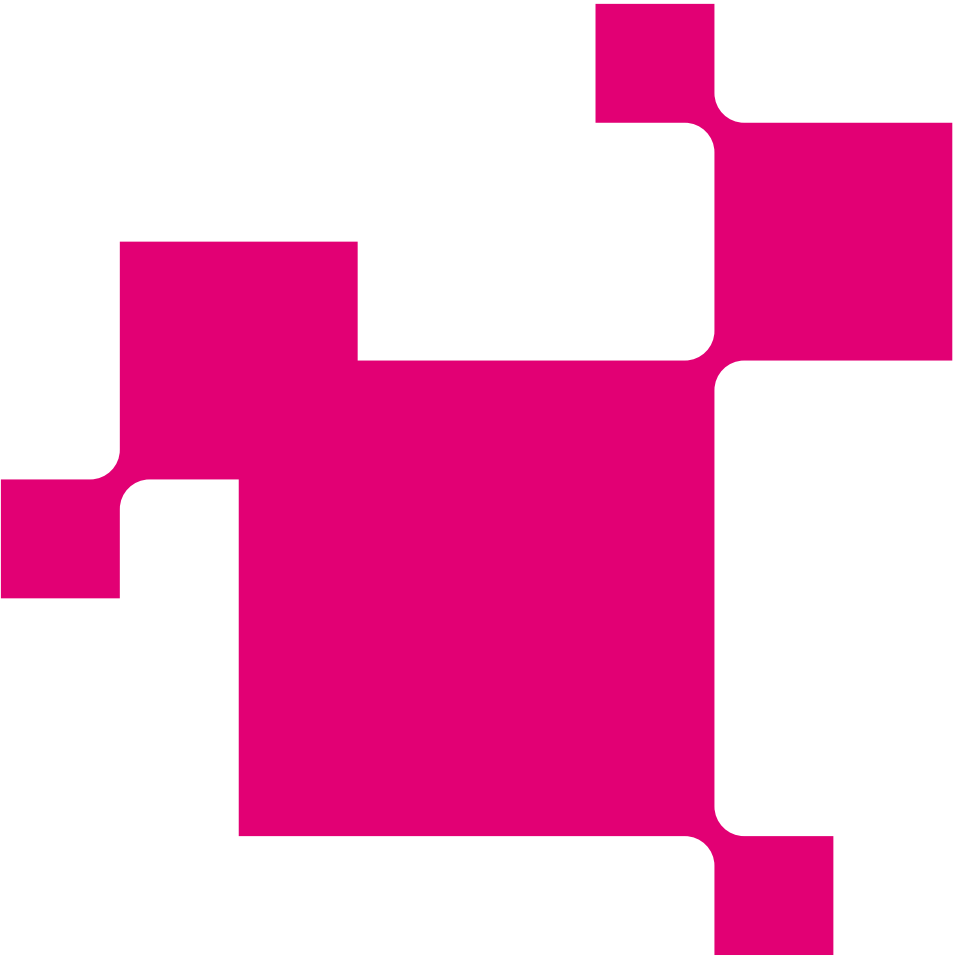
```
proof_request_web_request = {
    "connection_id": faber_agent.agent.connection_id,
    "proof_request": indy_proof_request,
    "trace": exchange_tracing,
}
await faber_agent.agent.admin_POST(
    "/present-proof/send-request", proof_request_web_request
)
```

```
indy_proof_request = {
    "name": "Proof of Education",
    "version": "1.0",
    "requested_attributes": {
        f"0_{req_attr['name']}_uuid": req_attr
        for req_attr in req_attrs
    },
    "requested_predicates": {
        f"0_{req_pred['name']}_GE_uuid": req_pred
        for req_pred in req_preds
    },
}
req_attrs = [
    {
        "name": "name",
        "restrictions": [{"schema_name": "degree schema"}],
    },
    {
        "name": "date",
        "restrictions": [{"schema_name": "degree schema"}],
    },
]
```

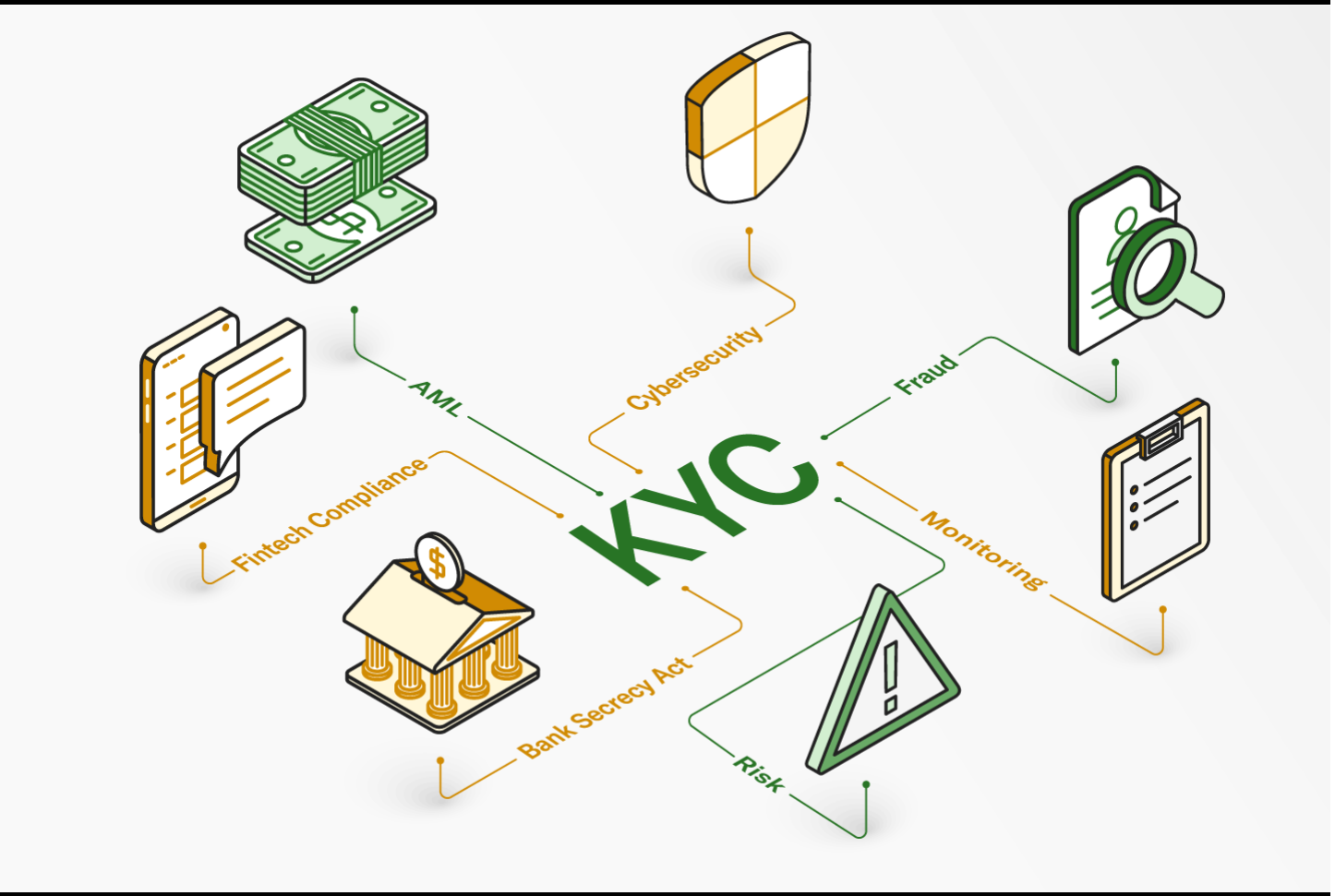
Credential Offer Received

```
state = message["state"]
credential_exchange_id = message["credential_exchange_id"]
prev_state = self.cred_state.get(credential_exchange_id)
if prev_state == state:
    return  # ignore
self.cred_state[credential_exchange_id] = state

self.log(
    "Credential: state = {}, credential_exchange_id = {}".format(
        state,
        credential_exchange_id,
    )
)

if state == "offer_received":
    log_status("#15 After receiving credential offer, send credential request")
    await self.admin_POST(
        f"/issue-credential/records/{credential_exchange_id}/send-request"
    )
```

- **Alice's agent uses the Agent container handler for a webhook notification related to the AIP 1.0 issue credential protocol.**



Know Your Clients (KYC)



KYC

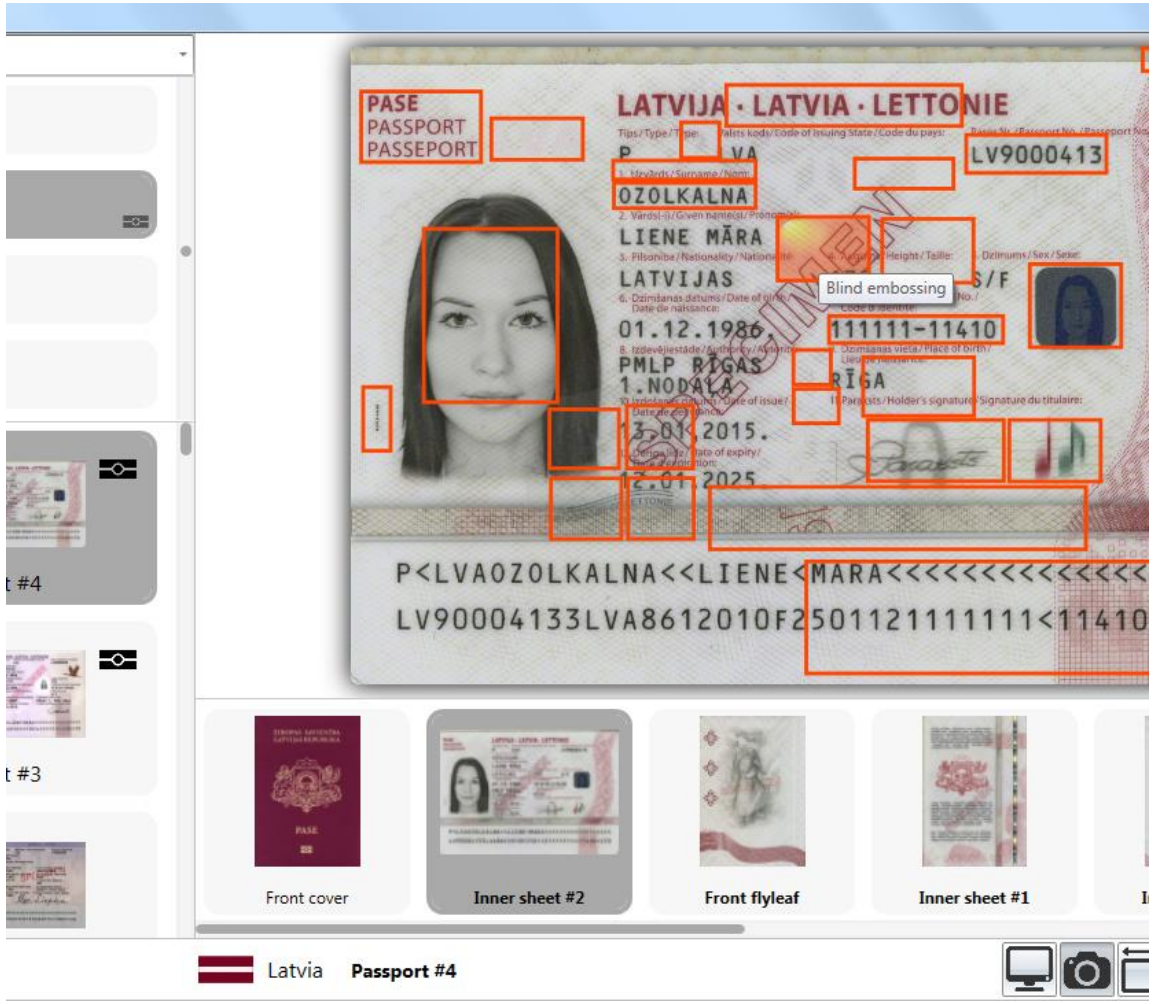
- KYC is the mandatory process of identifying and verifying the client's identity when opening an account and periodically over time.
- Identity verification practices to assess and monitor customer risk.
- A legal requirement intended as an anti-money laundering (AML) measure

Challenge! Structured Transparency!

- **How to protect the privacy of customers when on-boarding at a business, while simultaneously providing transparency to the business???**

- **The transparency enables a business to meet the know-your-customer (KYC) obligations they have under anti-money laundering and counter-terrorism financing regulation (AML/CTF)**

Bundling Problem



- While AML/CTF regulations usually require only specific data attributes (e.g. name, address, date of birth) of a customer to be verified for KYC purposes, often much more personal data is collected and stored by the regulated entity.

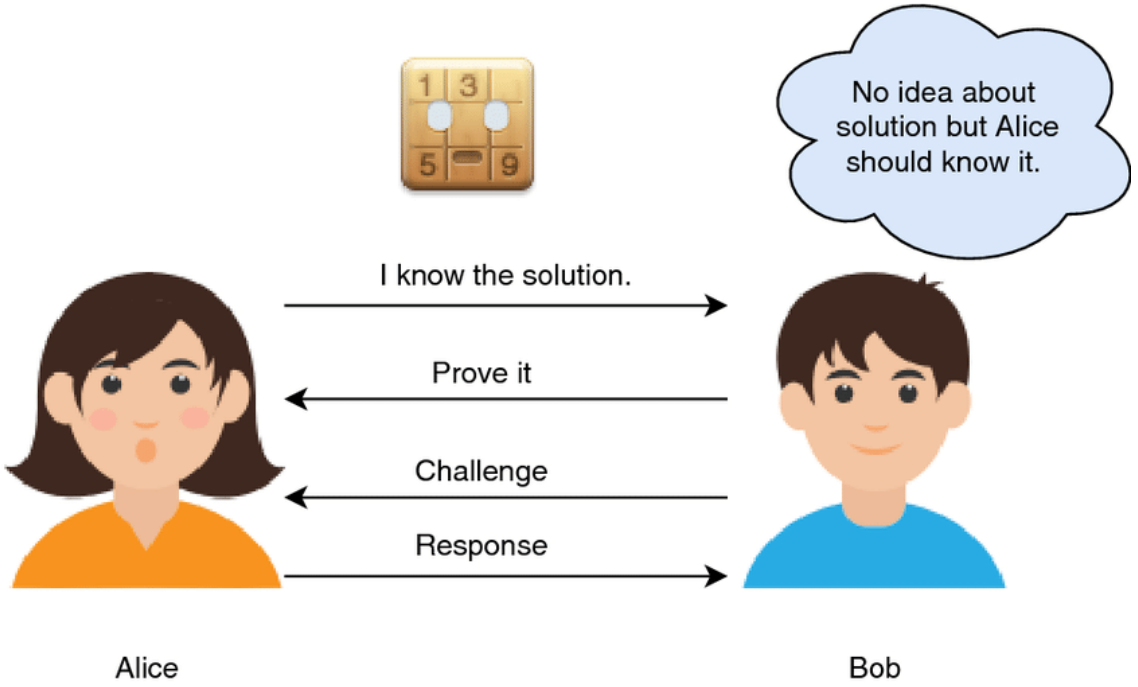
- Sometimes because copies are taken of full identity documents, revealing more attributes (i.e.: no elective disclosure), or because more data points are considered necessary to perform proper identity verification (i.e. to avoid false positives).

Zero-knowledge proof

A ZKP is a cryptographic method to prove to a party that you possess some knowledge without actually revealing the underlying information.

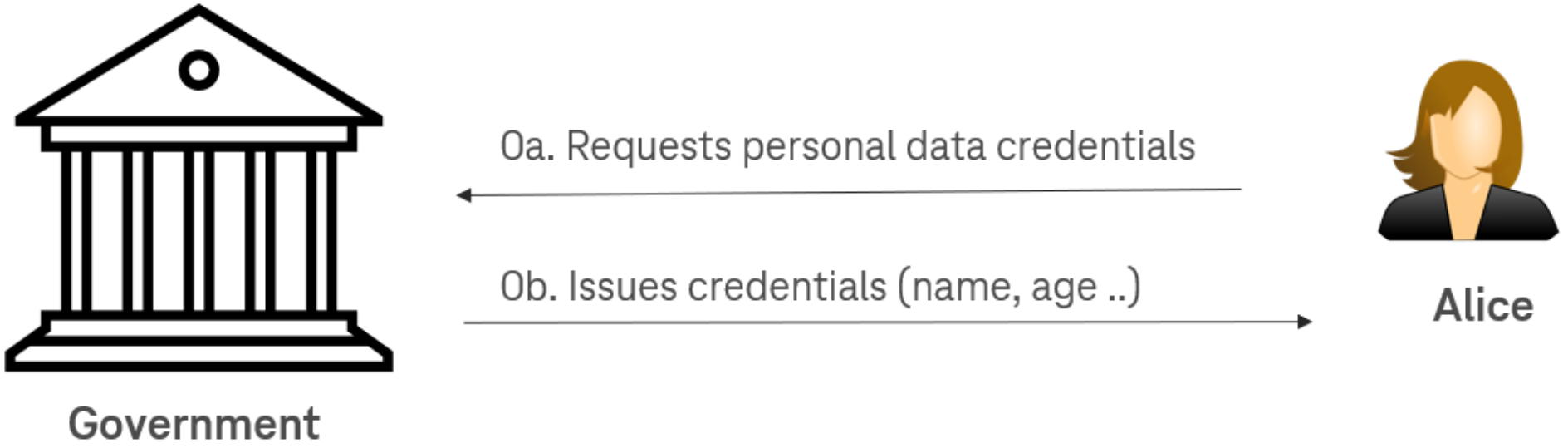
Combined, they are able to provide:

1) **Selective disclosure**
2) **Predicate proofs**
3) **Compound proofs**
4) **Non-correlating signatures**

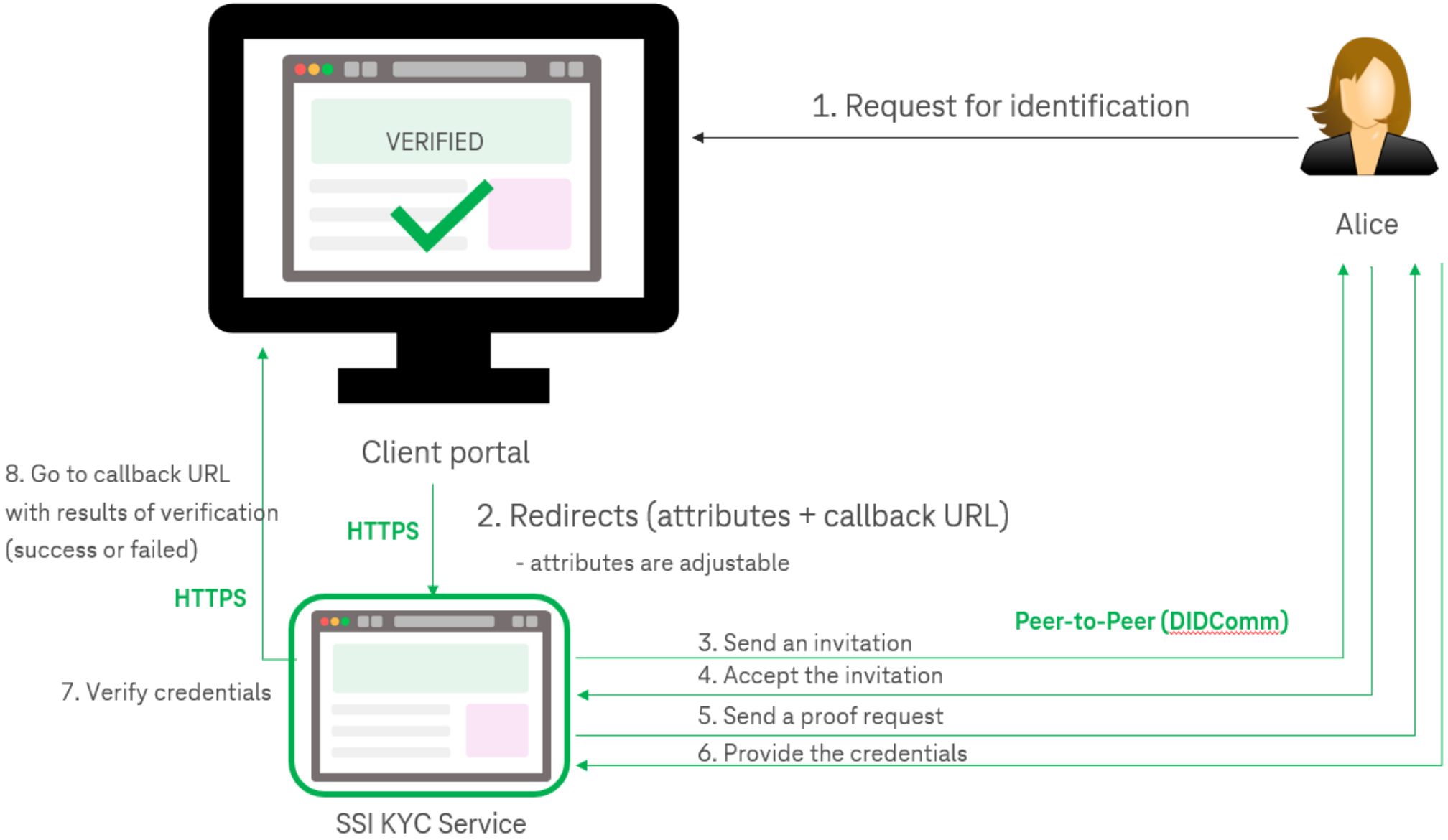


Initial state

- **User has already credentials (for example issued by the government) in his wallet**



KYC service


VERIFIED
1. Request for identification
Alice
Client portal
8. Go to callback URL
with results of verification
(success or failed)
HTTPS
2. Redirects (attributes + callback URL)
- attributes are adjustable
HTTPS
Peer-to-Peer (DIDComm)
3. Send an invitation
4. Accept the invitation
5. Send a proof request
6. Provide the credentials
7. Verify credentials
SSI KYC Service

Questions???