01 Solution Architecture

European Ecosystem for Digital Identity (EESDI)

Architecture & IT Security

Version	Date	Author	Comment
[redacted]	[redacted]	[redacted]	Initial version
[redacted]	[redacted]	[redacted]	Updated Architectural Overview

Introduction

The purpose of this document is to give an high level overview of the solution architecture describing the fundamental solution components and concepts required to enable SSI use cases. The technology selection and architecture definition for the EESDI solution was driven by:

- Non-functional and functional requirement identified in alignment with Bundeskanzleramt and Partners, as well as
- The key architectural system qualities as performance, capacity, maintainability, extensibility, availability and serviceability, security and usability.

This documentation additionally provides guidance for all use cases based on the EESDI solution as well as operations and development.

Table of Contents

Confluence Page	Short Description	Owner
Architectural Templates	Describes architectural templates on how and when to implement certain solutions to specific problem statements.	[redacted]
Non-functional Requirements	Describes the key non-functional requirements motivating the solution design.	[redacted]
Architectural Decisions	Describes the key architectural decision justifying the EESDI solution design.	[redacted]
Participation Patterns	Describes different patterns for participating with the EESDI solution.	[redacted]
Security Architecture	Describes the security aspects of the EESDI solution.	[redacted]
Data Model	Describes the data models of sovereign credentials like basis-id and mobile driver license.	[redacted]

Architectural Goals

Before going into the depth of the EESDI Solution Architecture, we define the critical non-functional requirements that, grouped by architectural system qualities, drove the development of this architecture.

Support Openness

The EESDI is intended mid to long-term to grow freely with joining companies implementing new use cases. Different participation patterns like tenancy and SaaS (see also Participation Patterns) will be supported. In any case the solution must be based on open standards and technologies which can be freely used without vendor lock-ins by any interested company.

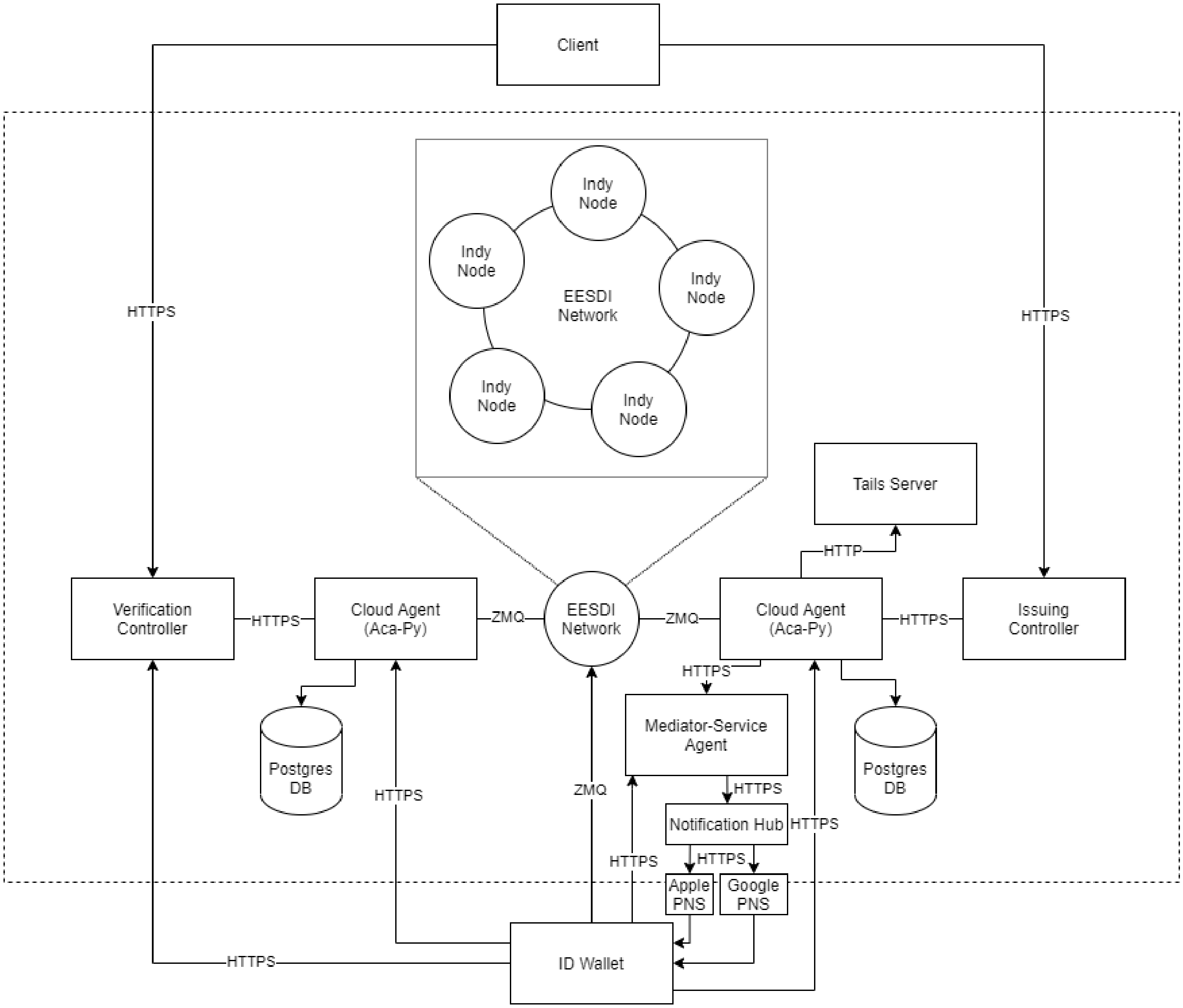
Support Scalability

The EESDI solution must be horizontally and vertically scalable, leveraging the power of cloud environments.

Architecture Overview

- working version -

The following diagram illustrates the key components building the EESDI solution. This diagram intentionally omits aspects on the infrastructure components like firewall, api gateways and load balancers as this is specific to the deployment of the individual building blocks.



EESDI Network

The EESDI Network consists of (currently 5) indy nodes operated by a network of node providers and implements a Verifiable Data Registry using a distributed ledger. The ledger stores the following key components.

- *Credential Schema Definitions (SCHEMA):* Describing the data template for a credential, defining what attributes names to expect.
- *Public Keys (CLAIM_DEF)*: Defines what public DIDs are permissioned to issue a credential for a particular schema definition.
- *Public DIDs (NYM, ATTRIB):* A DID (identity) stored to the ledger, describing the role of a network participant. (User, Trustee, Endorser, Steward, Network Monitor)
- *Revocation Data (REVOC_REG_DEF, REVOC_REG_ENTRY):* If revocation is used for a credential, particular revocation data is stored next to any issued credential. An update will be performed if the schema changes or the credential gets revoked.

Controller

The controller integrates with the cloud agent implementing the use case specific issuing and verification process. These components are generally customarily developed / deployed per use case according to the Architectural Templates.

Agents

A agent has the following two responsibilities in the solution design.

1. *Wallets:* Managing the wallets for identity owners.
2. *Messaging:* Communicating peer-to-peer with other agents to form connections and exchange credentials. This is done using a decentralized, secure messaging protocol DIDComm.

Mediator-Service Agent

In case the communication between two agents involves an edge device (like a phone) the mediator-service agent is used to proxy the communication between edge device to an agent.

Cloud Agent

The cloud agent is a regular software component run on a back-end server to communicate with a controller service. Additionally a cloud agent may be configured with a **Tails Server**, which will manage revocation entries for issued credentials.

ID Wallet

The ID Wallet stores the verifiable credentials issued by issuing controllers and responds to proof presentation requests, by created a zero knowledge proof of issued claims. It communicates with its mediator-service agent to form connections and credentials exchanges.

Client

The Client is the service which initiates the credential issuing respectively verification. It eventually receives the response of the process and may integrate that into its legacy processes. This client may be a ERP system or a front-end application, etc.

Appendix

Used Aries Standards

The EESDI Solution Architecture is based on the following RFCs.

RFC	Short Description
0005 DIDComm	An interaction protocol between agents and agent-like things
0008 MessageID and Threading	A RFC based on Service Decoractors to track threading data. It is primarily used to correlate asynchronous messages.
0015 ACKs	A RFC used to acknowledge proof presentations.
0036 Issue Credential Protocol 1.0	The protocol used to issue credentials.
0095 Basic Message	A protocol describing a stateless, easy to support user message protocol. It has a single message type used to communicate. Primarily used to submit additional data / attachments.
0037 Present Proof Protocol 1.0	The protocol used to verify credentials.
0056 Service Decorator	Additional service decorators used to specify service endpoints for connection-less proof requests.
0160 Connection Protocol	A protocol describing how to establish a connection between agents - used for issuing credentials.
0046 Mediators and Relays	A concept how an ssi mediator provides mediator and relay functionality for edge devices.
RFC 0268 Unified Didcomm Agent Deeplinking	A set of specifications for mobile agents to standardize around to provide better interoperable support for DIDCOMM compliant messages.

Glossary

Architectural Decisions

Version	Date	Author	Comment
0.1	09.08.2021	[redacted]	Initial version
0.2	19.08.2021	[redacted]	Added ARC001 Verification as a Service Decision and ARC002 Verify Service Endpoint
0.3	24.08.2021	[redacted]	Added ARC003 Multiple Proof Requests, ARC004 Publish Tails Files and ARC005 Revocation
0.4	31.08.2021	[redacted]	Added ARC006 Publish Docker Images
0.5	16.09.2021	[redacted]	Added ARC007 Container Orchestration
0.6	24.09.2021	[redacted]	Added ARC008 and ARC009
0.7	29 Oct 2021	[redacted]	ARC009: Reviewed and incorporated BKa feedback
0.8	05 Nov 2021	[redacted]	Added "ARC011 Classes of VCs"
0.9	16 Nov 2021	[redacted]	Added ARC012 and ARC013 as a TODO.
1.0	22 Nov 2021	[redacted]	Deleted "ARC011 Classes of VCs" (moved to dedicated page); Renamed ARC012 to ARC011 and ARC013 to ARC012; Updated ARC009 with final decision
1.1	01 Dec 2021	[redacted]	Added ARC013
1.2	11 Feb 2022	[redacted]	Initial ARC014

Introduction

The architectural decisions describe technical problem statements, different options and a justification for a selected solution approach. These decisions are proposed by the owner to the Architecture Board upon which a decision will be taken according to the following status flow.

TODO DRAFT IN REVIEW APPROVED

Status	Description
TODO	Architectural Decision is created, but not documented yet.
DRAFT	Architectural Decision has been drafted, but not finished yet. Work In Progress.
IN REVIEW	Architectural Decision has been finished and waiting for review in the next Architecture Review Meeting.
APPROVED	Architectural Decision has been approved by the Architecture Board.
REJECTED	Architectural Decision has been rejected by the Architectural Board.
SUPERSEDED	Architectural Decision has been superseded by another Architectural Decision
DEPRECATED	Architectural Decision is deprecated and not relevant anymore.

Architectural Decisions

The following table compiles a list of architectural decisions for the EESDI Solution Architecture.

ARC001 Verification as a Service

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
The verification of SSI credentials is implemented the same way for each and every verification controller. Starting a new use case always requires to build / copy the exact same code for executing the verification process. This results in more maintainability efforts and less flexibility in adapting the overall process.	A **General Purpose Verification API Service** could be built abstracting the verification process into a simple REST interface. This service would have the responsibility to adhere to the EESDI Architectural Templates for verification and returns the result to any use case-specific service for further processing. This service could be generally hosted and simplify the onboarding of any new use case especially if only verification is desired. **Benefits:** Reusable for multiple use cases, even a combination with Use Case Specific Verification API Service would be possible.		ART005 Verification Integration Pattern Participation Patterns	IN REVIEW	19.08.2021
	A **Use Case Specific Verification API Service** could be built implementing a specific endpoint for e.g. a mobile driver license verification. This service would have the responsibility to adhere to the EESDI Architectural Templates for verification and offers a use case-specific verification and validation. e.g. applying the business logic for validating the mobile driver's license or vaccination status. **Benefits**: The onboarding can happen on a use case rather than on a technical platform, allowing the integration to bring value faster.				

ARC002 Verify Service Endpoint

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Created At	Last Update
A connection-less proof request is presented to the ID wallet with a display of two custom service decorators, service endpoint and host name. These annotations can be freely chosen by the verification service, without any further verification. As a result the user is less protected against phishing attacks.	**Trusted Verifier:** Instead of putting the service endpoint and hostname into service decorators freely chosen by the verification controller, rather query the ATTRIB to the public ID for the agent. This implies that verifiers' DIDs need to be registered on the ledger. The Holder can additionally be warned / informed about the trustworthiness of the verifier. **Benefits**: The network would attest certain verifiers to be trustworthy. Allowing the user to have more trust in the proof request validity.		ART001 Verify SSI Credentials	DRAFT	19.08.2021	01.10.2021
	Connection-based Verification: Connection-less proof requests are generally not allowed, ensuring that all verified have to initially build a trusted connection to the Holder. The DID of the verifier is protected through a TLS certificate, which is validated during the verification process. The ID Wallet will show whether or not the certificate is trusted - analog to a regular HTTPS connection to a website) **Benefits**: Additional security through a verified and trusted connection. **Disadvantages**: • Static QR codes are not possible anymore as a dynamic QR code needs to be generated for every verification process. • An additional step for verification is required to complete the verification process. *Please note, this disadvantage only applies if there isn't any existing connection there yet.* • A did pair will have to be created for every verification. As shown in the performance tests - this will have an impact on the performance. see also 01 Analysis Network Performance & Security					

ARC003 Multiple Proof Requests

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
Multiple proof requests need to be presented to the ID wallet at the same time, so the holder can choose which one to answer. Currently, this is not supported by aca-py out of the box.	**Option 1:** Create an **array of connectionless proof requests** and put it as a payload for the wallet. The wallet checks which of the requests it can answer, and asks the user to choose one of them. **Benefits:** Simplicity: Straightforward to implement on the controller side as we already use connectionless proof requests; on the wallet side, any of the evaluated approaches require the same implementation changes. **Disadvantages:** The controller needs to care for creating many proof requests and deleting the unused ones.	**Option 1:** The **array of connectionless proof requests avoids** modifications of the aca-py should be avoided. The use of new protocols (e.g., exchanging multiple proof requests through basic messaging) or custom modifications of aca-py would cause unnecessary overhead.	ART002 Multiple Proof Requests (without Connection) ART001 Verify SSI Credentials	APPROVED	03.09.2021
	Option 2: Connection-full multiple proof requests by adapting the aca-py request-proof protocol. If a connection has already been established, a proof request can be created via the */request-proof /send-request* admin-API endpoint (RFC 0037). **Benefits:** Allows for connection-ful multiple proof requests. **Disadvantages:** Since the proof request is created and sent by the aca-py, no direct adaptation to its structure can be made. In order to ensure the desired functionality, an adaptation to the source code of the aca-py has to be made.				
	Option 3: Basic messaging. As another alternative, to send custom content, basic messaging (RFC 0095) can be used. In its most basic way, it allows one party to send a string (in which e.g. a JSON document may be encoded) to another party. Hence, it could be used to model our multiple proof requests protocol. **Benefits:** - **Disadvantages:** The handling of additional events like this have to be implemented in the wallet.				

ARC004 Publish Tails Files

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
The tails files need to be published so an ID-wallet can download it when creating the first proof with a verifiable credential.	**Option 1:** Publish the tails files on the **indy-tails-server**. **Benefits:** • Seamless integration with the aca-py: Once the aca-py is given the --tails-server-base-url flag, it will automatically take care for creating and configuring a revocation registry in the right way • Can be started with the right environment variables with docker **Disadvantages:** Yet another third-party image for quite limited functionality	**Option 1**: The **indy-tails-server** is most convenient because it allows for automating more steps.	IG004 Tails Server Installation IG001 ACA-PY Installation and Setup, Testnetwork Configuration	APPROVED	03.09.2021
	Option 2: Publish the tails files on a highly available file store like Github **Benefits:** Simplicity: Straightforward to implement on the controller side; and the things that need to be implemented on the wallet side will need to be implemented with any approach. **Disadvantages:** Requires manual uploading of the tails file and creation and modification of the revocation registry				

ARC005 Revocation

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
The issuer needs to store information about which credentials have been issued to revoke the right ones in case it is needed.	**Option 1:** For each credential, map its holder's universally unique identifier (UUID) to the verifiable credential's revocation ID and the associated revocation registry ID. **Benefits:** The holder's UUID is not personally identifiable without additional information, and the revocation ID and the revocation registry ID are not sensitive (the holder never shares their revocation ID when creating proofs, and the revocation registry ID gives herd privacy). Credential exchange records can be deleted. **Disadvantages:** This is not a standalone solution; the revocation process relies on a third party that maps the UUID to holder information and that asks for revoking the verifiable credential issued to the holder with this UUID.	**Option 1:** The option that only used the **holder's UUID**, the credential's **revocation ID** and the associated **revocation registry ID** avoids the need of protecting yet another store of personal information in the database. It also allows to delete the credential exchange records from the aca-py, which is good from a performance perspective and avoids persisted sensitive information there, too.	ART007 Revoke Credential ART004 Issue SSI Credentials	APPROVED	21.09.2021
	Option 2: Not deleting credential exchange records in the aca-py or storing holder-related information (e.g., the issued attributes or the exact time of issuance) in the issuer's database. **Benefits:** This approach would allow to search for specific credentials with specific attributes and to delete them. **Disadvantages:** In the first case, there will be sensitive information persisted in the aca-py, which is bad from a security perspective and for performance in the long run. In the second case, there will be sensitive information in the controller database.				

ARC006 Publish Docker Images

Problem Statement	Options		Decision / Rationality / Implications	Related Documents	Status	Last Update
We're publishing the code along with all dockerfiles and scripts necessary to build the docker images to public open source repositories. It makes deployments easier when we publish the images associated with a release to a public images repository.	**Option 1:** Publish images publicly There are several Providers for Public image repos, the restriction, Disadvantages and Benefits are outlined in the attached Document **Benefits:** No key management There is an official channel for publishing open Source images along with the code **Disadvantages:** Anybody can check out and run the images, there must be no security issues or specific infrastructure related configuration exposed on the images (e.g. dev network genesis URL as standard configuration)	**1.1** Dockerhub **1.2** Github Packages **1.3** Quay.io	**Option 1.2:** Github packages offer a docker repository attached to the my-digi-id organization on Github, which is the Official Repository for code sharing for the SSI Projects. As with all options, it is free for public repositories and has no Pull Limits attached. **Implications:** Images must never contain any standard configuration that connects to an existing endpoint. Images must never be built with secret default configurations. Enforce Secret Detection on the pre-merge pipelines	packages evaluation.xlsx	APPROVED	21.09.2021

	Option 2: Keep images private **Advantages:** We control who has access to the images **Disadvantages:** Everybody has access to the source code and dockerfiles, the images can be built from source, it is just Security by Obscurity	IBM Cloud CR				

ARC007 Container Orchestration

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
Which container orchestrator are we considering as a deployment target? By defining a concise target environment we can make sure to ensure interoperability, as well as provide properly tested install/config /operation guidelines.	Option 1: **Docker Swarm** **Advantages:** - Smoothly integrates with Docker toolset, easy bring-up. **Disadvantages:** - Docker EE (including Swarm) was bought by Mirantis, Swarm is not strategic anymore, focus of Mirantis is on Kubernetes based orchestrators - Lower market adoption, weaker market offering and future strategy see e.g. this Forrester study from 2020. - No as-a-service provider for Docker Swarm. - Even Docker Desktop (still with Docker Inc) has started to include a Kubernetes distribution.			DRAFT	16.09.2021
	Option 2: **Plain Kubernetes** **Advantages:** - Broad Kubernetes Ecosystem. - Governed by Cloud Native Computing Foundation (90 certified Kubernetes distributions) - Market-leading orchestration service, partly proprietary as-a-service offerings from AWS, Microsoft Azure, Google Cloud Platform, IBM Cloud and more. - Very good for stateless and stateful services compared to Docker Swarm. **Disadvantages:** - Open Source updates are frequent and require careful patching. - Lot of integrations need to be built on top of the core orchestration engine to provide a platform.				
	Option 3: **OKD** OKD is the upstream open source Origin Community Distribution of Kubernetes that powers Red Hat OpenShift. **Advantages:** **Disadvantages:**				

	Option 3: **Red Hat OpenShift** Red Hat OpenShift is a downstream enterprise-level production product based on the OKD open-source product. **Advantages:** • Red Hat OpenShift is part of the Cloud Native Computing Foundation (CNCF) Certified Kubernetes program, ensuring compatibility and interoperability between your container workloads. • Used in the past very successfully to introduce container technology to more security-sensitive industries like Finance / Banking. • Strategic as-a-service offerings from Google, Microsoft, IBM, Amazon, but also very strong on-prem customer base, therefore allows for portability across public cloud hypervisors and on-prem environments. • Pre-Integrates a lot of components that are missing from Plain Kubernetes, for example, Ingress Routing, DNS, Load Balancing, Logging, Operators, Object Storage abstraction, CI/CD Pipelines, Service Mesh, Marketplace, a very good Web Console, Platform Monitoring+Alerts, proper Role-Based Access Control and much more. • Red Hat is besides Google since 2014 major Kubernetes open-source committer. **Disadvantages:** • Red Hat OpenShift is not for free, but one can leverage OKD if needed. Most as-a-service offerings rely though on OpenShift instead of OKD due to the enterprise-level support.				

ARC008 Self Attested Attributes

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
Some attributes can not be attested by any official institution like favorite color or clothing size. However, a use case may be interested in getting additional attributes of a user's identity in a self-attested fashion without the requirement of an issued credential.	**Option 1**: The attribute is simply stored as a regular record into the SQLite Database of the ID Wallet. Upon request, the corresponding attribute is fetched from the database and matched to the self-attested attribute request. The self-attested attribute is a mechanism that can be used within a regular proof request leaving the restrictions section empty. **Option 2**: A general-purpose issuing service is set up allowing any holder to make claims about his own identity. e.g. favorite color or clothing size. The attributes would then be issued as regular credentials and can be verified as such.	**Option 1**: Saving the attributes as credentials will not deliver any additional trust and might even imply a higher trust level than actually given. Using a regular proof request with empty restrictions sections allows for a lean solution. **Implications**: It needs to be clarified how self-attested attributes can be standardized so that attribute names can be shared across multiple different use cases.	Self-attested claims registry	IN REVIEW	24.09.2021

ARC009 Validation of Technical Attributes (TVA)

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update / By
Data attributes like an email or a phone number, which must be provided for signing up to services in both the digital and physical space each and every time, would benefit from being technically validated and, optionally, issued as a Verifyable Credential. Various techniques for validating given data attributes can be applied, in order to balance convenience, security, and feasibility.	**Option 1**: Use self-attested attributes for any attributes (including technical validatable) where there is no issuer for a credential. **Option 2**: An ecosystem entity offers a service for TVAs such as email addresses and phone numbers through a 3P attestation service, using challenge-response (double opt-in) operations and issue credentials after successful attestation (Class 3). A lesser grade attestation (e.g. purely syntactical) may be performed and issued as Class 2 credentials. Caveat: Issued credentials only attest for the user having access to the phone/mailbox at the time of attestation. As this potentially opens up attack vectors, those credentials need to be classified accordingly (i.e. Class 2) **Option 3:** Issue credentials based on the user's identity (Basis-ID) by looking up regulatory data registries, such as KYC databases at MNO providers. Such credentials are being treated as Class 3 / 4 (depending on the reputation of the issuer) and may be named Elevated Technically Verified Attributes (eTVA).	**Options 4:** The credibility of the EESDI network may be jeopardized if self-attested attributes such as email or phone numbers are treated as "trustworthy" credentials. However, for certain use cases without the need for "secure" credentials, the trade-off between self-attested (convenience) vs. assurance is acceptable - the verifier takes the risk. Verifiers have to define the classification of requested attributes based on their needs. Based on the attested attribute, the sensitivity of the use case, and the characteristics of issued credentials (i.e. revocation, re-validation), it has been decided that ▪ technically (syntactically) attested will be used for the EESDI ecosystem. **Option 2 & 3** are potential extensions to further improve the quality of self-attested attributes:	ARC008 Self Attested Attributes ART010 Issuing verified e-mail addresses & phone numbers Classes of Verifiable Credentials Self-attested claims registry	APPROVED	12 Jan 2022 [redacted]

	Option 4: Option 1 with additional plausibility checks (i.e. RegEx for email addresses and phone numbers)	- 3P challenge-response attestation, and - regulated data registry lookups			

ARC010 Static VS Dynamic QR Codes

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Created At	Last Update
QR codes if statically generated can be misused if e.g. a photo is taken and shared on social media. A solution may be required to address this possibility of exploitation.				TODO	06.10.2021	06.10.2021

ARC011 Hybrid Vs Native Development

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
For now we developed our application for the iOS and Android operating system in Xamarin.	**Option 1:** Develop our application nativ for Android and iOS. This leads to several actions we have to take care of. - We have to develop the whole application again - We have to make sure to use the Aries SDK. There is no out of the box SDK for Android and iOS - We have to make sure that an upgrade from the "old" version to the new one is possible for our customer	[redacted]		TODO	17.11.2021
	Option 2: Further development in Xamarin.				

ARC012 DLT vs. PKI

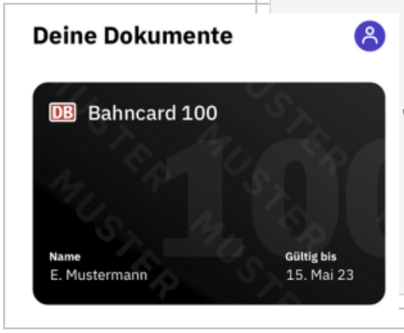
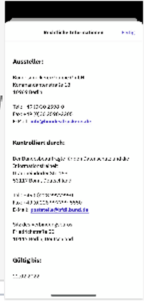
Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
				TODO	17.11.2021

ARC013 Verification Callback

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update
Callbacks after verification need to be secured against spoofing. Therefore, we have different options with different pros and cons.		**Option 4/5 have been selected**	ART009 Verification Callback	APPROVED	01 Dec 2021

ARC014 Issuer metadata in credentials

Problem Statement	Options	Decision / Rationality / Implications	Related Documents	Status	Last Update

It has been raised as a requirement to visualize information about the issuer of a credential in ID Wallet. Such information may be: - Company Logo - Legal Name - Address - Attested by Examples:  Deine Dokumente DB Bahncard 100 Name E. Mustermann Gültig bis 15. Mai 23  Aussteller: Kontrolliert durch: Gültig bis: V allback.pptx	ARC014 WG - Issuer metadata in credentials	DRAFT	11 Feb 2022

Architectural Templates

Version	Date	Author	Comment
0.1	14.07.2021	[redacted]	Initial version
0.2	30.07.2021	[redacted]	added ART007
0.3	02.08.2021	[redacted]	updated ART004 row
0.4	10.08.2021	[redacted]	updated ART007 to revoke credential
0.5	27.08.2021	[redacted]	updated Revoke Credential, added ART009
0.6	24.09.2021	[redacted]	added ART010
0.7	28.09.2021	[redacted]	approved ART003
0.8	03 Feb 2022	[redacted]	ART009 in review

Table of Contents

#	Title	Confluence Page	Short Description	Owner	Status
ART 001	Verify SSI Credentials	ART001 Verify SSI Credentials	A template for a simple SSI credential verification.	[redacted]	APPROVED
ART 002	Multiple Proof Requests	ART002 Multiple Proof Requests (without Connection)	A template on how to perform a proof request of multiple credentials definitions	[redacted]	APPROVED
ART 003	Device Binding Verification	ART003 Device Binding Verification	A template on how to verify a device coupling challenge.	[redacted]	APPROVED
ART 004	Issue SSI Credentials	ART004 Issue SSI Credentials	A template for a simple SSI credential issuing.	[redacted]	IN REVIEW
ART 005	Verification Integration Pattern	ART005 Verification Integration Pattern	A template to generically integrate third-party proof request originators.	[redacted]	APPROVED
ART 006	Self Attested Attributes	ART006 Self-Attested attributes	A self-attested attribute is a simple request for presentation without restrictions.	[redacted]	IN REVIEW
ART 007	Revoke Credential	ART007 Revoke Credential	A template on how to revoke an issued credential.	[redacted]	IN REVIEW
ART 008	Report Problem		A template on how to perform problem reporting during issuing and verification		TODO
ART 009	Verification Callback	ART009 Verification Callback	A template on how to callback a mobile app once the presentation process has been completed within the ID Wallet.	[redacted]	IN REVIEW
ART 010	Verify Email & Phone Number	ART010 Verify Email & Phone Number	A template on how to verify technical attributes which are self-attested, and externally asserted can	[redacted]	IN REVIEW
ART 011	Trusted Verifier		A template to describe how to improve the trust relationship between holder and verifier	[redacted]	TODO

Status Flow

TODO DRAFT IN REVIEW APPROVED

Purpose

The architectural templates describe patterns to implement solutions for specific problem statements.

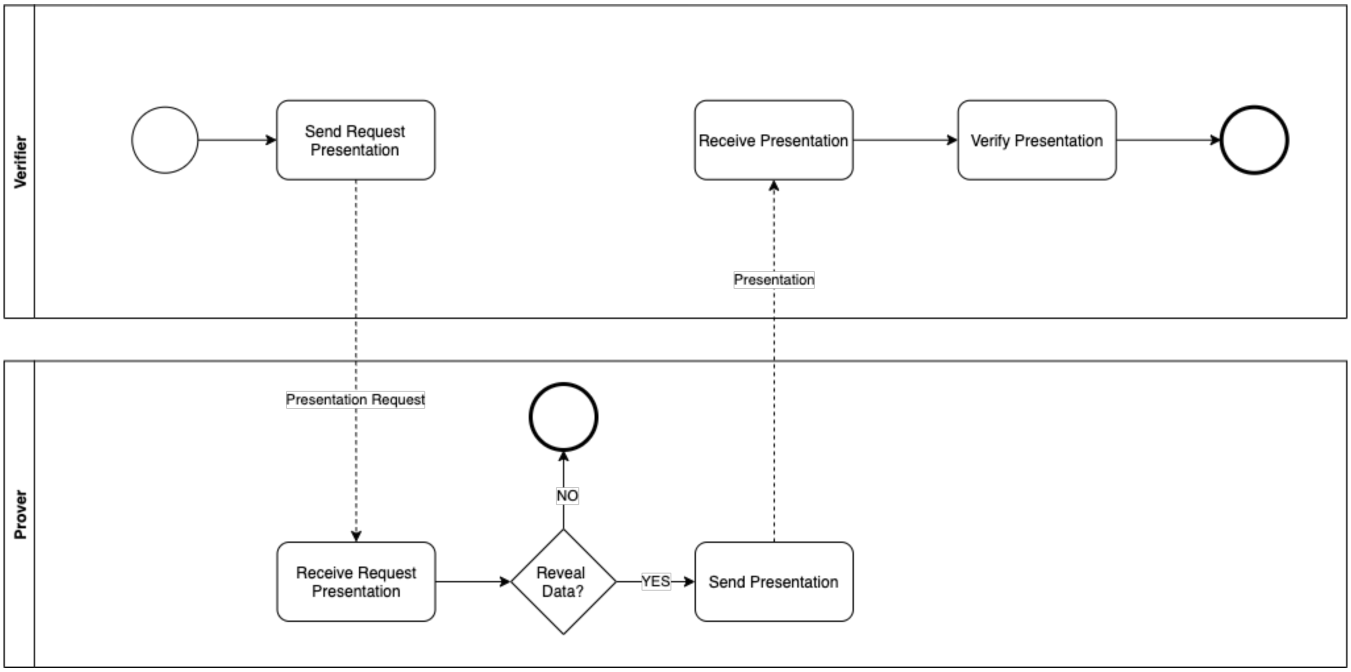
ART001 Verify SSI Credentials

Version	Date	Author	Comment	Status
0.1	14.07.2021	[redacted]	Initial version	DRAFT
0.2	16.07.2021	[redacted]	Extracted verification integration pattern into dedicated architecture template ART005 and updated documentation accordingly.	DRAFT
0.3	09.08.2021	[redacted]	Set document status to in review	IN REVIEW
0.4	23.08.2021	[redacted]	Incorporated Feedback	IN REVIEW
0.5	25.08.2021	[redacted]	Some editorial changes and additional information	IN REVIEW
1.0	30.08.2021	[redacted]	Incorporated Review Feedback Proof-reading, minor typo fixes	APPROVED
1.1	02.09.2021	[redacted]	Adding detailed sequence diagram	APPROVED

Table of Contents

Present Proof Protocol

This architectural template describes how SSI credentials are verified using the RFC 0037 Present Proof and RFC 0056 Service Decorator. This documentation is based on the *"Request Proof - Verifier to Prover"* flow. The diagram below illustrates the choreography implemented from the RFC 0037 Present Proof Protocol 1.0. (refer to Present Proof Choreography Diagram.)



The following steps describe the high-level interaction flow of requesting, presenting, and verifying a proof request.

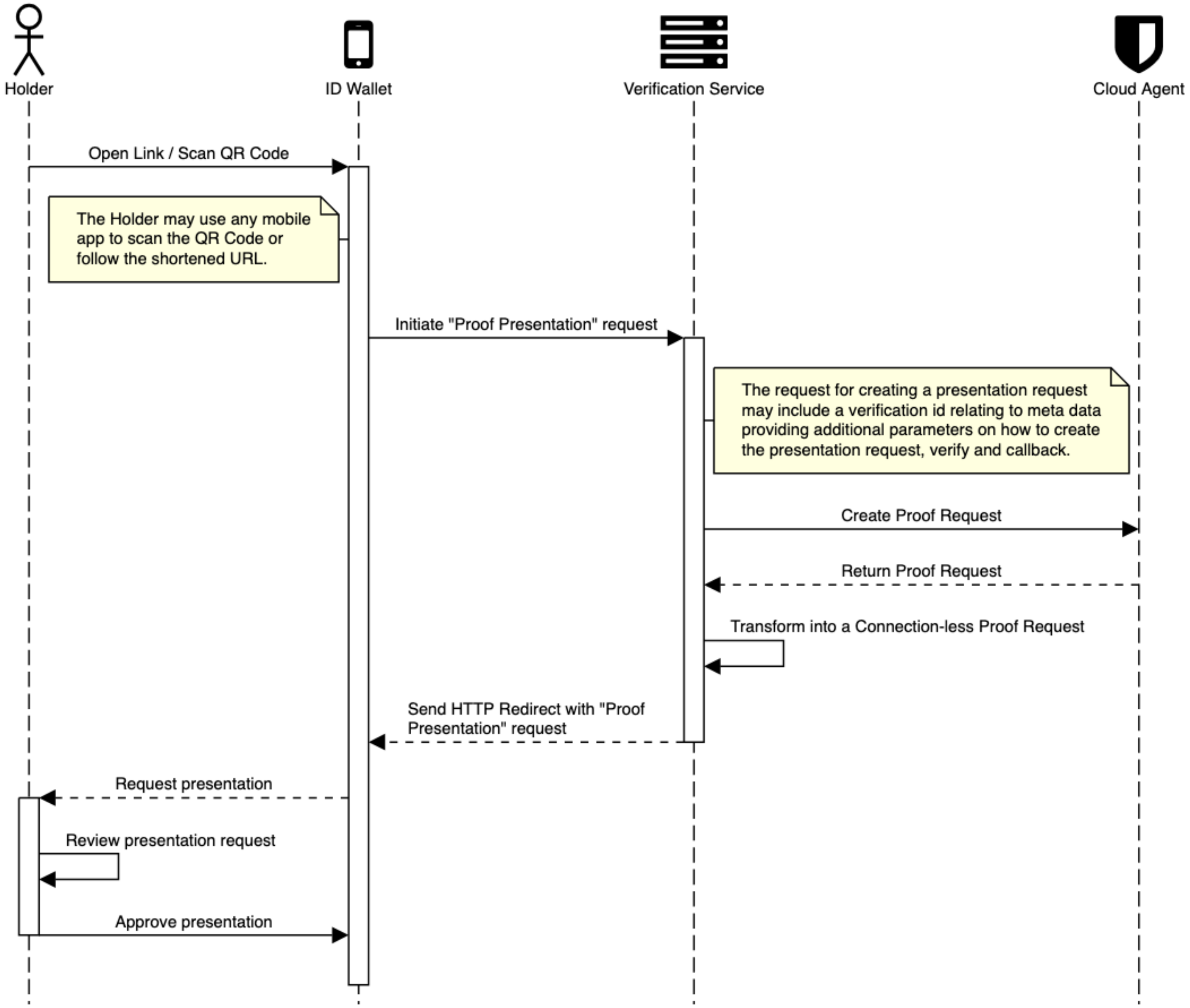
- The Verifier (Verification Service) provides a static QR code to the Holder, encoding an HTTP endpoint to request the Verifier (Verification Service) to initiate the proof request.
- This is triggered by the Holder (User) after performing an HTTP call to the shortened URL from the Prover (ID Wallet) - see also section **Request Presentation**.
- The presentation request is returned to the Prover (ID Wallet).
- The Prover (ID Wallet) collects the Holder's (Users) approval for revealing the requested attributes. In case the Holder (User) rejects that request - the process stops here. *Please note, there is currently no problem report flow implemented as suggested by the RFC 0037.*
- After approving the presentation request, the presentation (proof) is constructed and submitted to the Verifier (Verification Service).
- Finally, the Verifier (Verification Service) verifies the presentation.

The following sections dive into more details on how a presentation is requested and how it is verified.

Request Presentation

Although it would be possible to encode the complete connection-less proof request in a QR code, the result would be very complex (several kB). Thus only state-of-the-art mobile phones would be able to successfully scan the QR code. In order to accommodate that limitation, a shortened URL in accordance with RFC 0434 URL Shortening is generated and encoded into the QR code submitted to the Holder (User). This URL contains an HTTP endpoint that needs to be opened by the Holder's (User's) mobile phone. This can either be done directly through scanning the QR code (via the ID Wallet app or any other QR scanning app) or by following the HTTP endpoint through the mobile devices' browser.

The following diagram illustrates the sequence of creating a presentation request after the Holder (User) has received the QR code or deep-link to present a proof.


Holder
ID Wallet
Verification Service
Cloud Agent
Open Link / Scan QR Code
The Holder may use any mobile app to scan the QR Code or follow the shortened URL.
Initiate "Proof Presentation" request
The request for creating a presentation request may include a verification id relating to meta data providing additional parameters on how to create the presentation request, verify and callback.
Create Proof Request
Return Proof Request
Transform into a Connection-less Proof Request
Send HTTP Redirect with "Proof Presentation" request
Request presentation
Review presentation request
Approve presentation

Shortened URL

The shortened URL is either encoded through a QR code or represented as a URL. The following HTTP endpoint illustrates an example.

Shortened URL Example

```
https://verification.example.cloud/api/proof
```

Please note that the deep link may include a verificationId query parameter linking the proof request to previously exchanged metadata. see also ART005 Verification Integration Pattern for more details.

Create Presentation Request

Based on the shortened URL, the ID Wallet mobile app (or any other app) performs a corresponding HTTP call. The verification service receives that request and creates the Connection-less Proof Request in the following steps.

1. *Create Proof Request:* The verification service performs an HTTP call against the cloud agent (ACA-Py) REST API and creates a proof request, including what attributes or predicates to request for presentation by the Holder (User). This request is performed against the `/present-proof/create-request` endpoint. see also ACAPY Swagger API - Dev Environment as reference. The cloud agent (ACA-Py) creates an exchange id and returns the V10PresentationExchange model (see Appendix).

2. *Create Connection-less Proof Request:* After the presentation exchange record has been created, it needs to be transformed into a Connection-less Proof Request.

   Example

```
{
    "@type": "https://didcomm.org/present-proof/1.0/request-presentation",
    "~service": {
        "recipientKeys": ["8HH5gYEeNc3z7PYXmd54d4x6qAfCNrqQqEB3nS7Zfu7K"], // the public key of the
receiving cloud agent.
        "routingKeys": [], // not used
        "serviceEndpoint": "https://example.com/endpoint" // the public endpoint of the cloud agent.
    },
    "~thread": {
        "thid": "98fd8d72-80f6-4419-abc2-c65ea39d0f38" // a correlation id used to link verification flow
steps.
    },
    "request_presentations~attach": {
       "@id":"libindy-request-presentation-0",
       "mime-type":"application/json",
        "data": {
         "base64":"eyJuYW1lIjogIlByb29mIHJlcXV...=="
        }
    }
}
```

The Connection-less Proof request implements the RFC 0056 Service Decorator and RFC 0008 Threading shown in the example above in the tags "~service" and "~thread".

See also Request Presentation, and V10PresentationCreateRequest example in Appendix.

ⓘ **Use Range Proofs Where Possible**

For data minimization reasons, where possible, range proofs should be used to check, for example, whether the date of birth in a credential implies legal age or an issuance/validity date implies non-expiration. See also ART004 Issue SSI Credentials on examples where range proofs make sense and how they need to be prepared on issuance through integer-valued attributes. Range proofs are performed through filling the "requested_predicates" section in the V10PresentationCreateRequest JSON body (see Appendix).

1. Finally, the Connection-less Proof Request is converted into a didcomm URI and returned as an HTTP redirect to the ID Wallet. This presentation request is then shown to the Holder (User) for approval. *Please note, that this is enforced by the deeplink protocol (didcomm) which will be registered on the ID Wallet to process the redirect.*

Proof Presentation Redirect

```
didcomm://example.org?
m=ew0KICAgICJAdHlwZSI6ICJodHRwczovL2RpZGNvbW0ub3JnL3ByZXNlbnQtcHJvb2YvMS4wL3JlcXVlc3QtcHJlc2VudGF0aW9uIiw
NCiAgICAifnNlcnZpY2UiOiB7DQogICAgICAgICJyZWNpcGllbnRLZXlzIjogWyI4SEg1Z1lFZU5jM3o3UFlYbWQ1NGQ0eDZxQWZDTnJx
UXFFQjNuUzdaZnU3SyJdLCAvLyB0aGUgcHVibGljIGtleSBvZiB0aGUgcmVjZWl2aW5nIGNsb3VkIGFnZW50Lg0KICAgICAgICAicm91d
GluZ0tleXMiOiBbXSAvLyBub3QgdXNlZA0KICAgICAgICAic2VydmljZUVuZHBvaW50IjogImh0dHBzOi8vZXhhbXBsZS5jb20vZW5kcG
9pbnQiIC8vIHRoZSBwdWJsaWMgZW5kcG9pbnQgb2YgdGhlIGNsb3VkIGFnZW50Lg0KICAgIH0sDQogICAgIn50aHJlYWQiOiB7DQogICA
gICAgICJ0aGlkIjogIjk4ZmQ4ZDcyLTgwZjYtNDQxOS1hYmMyLWM2NWVhMzlkMGYzOCIgLy8gYSBjb3JyZWxhdGlvbiBpZCB1c2VkIHRv
IGxpbmsgdmVyaWZpY2F0aW9uIGZsb3cgc3RlcHMuDQogICAgfSwNCiAgICAicmVxdWVzdF9wcmVzZW50YXRpb25zfmF0dGFjaCI6IHsgL
i4uIH0NCn0=
```

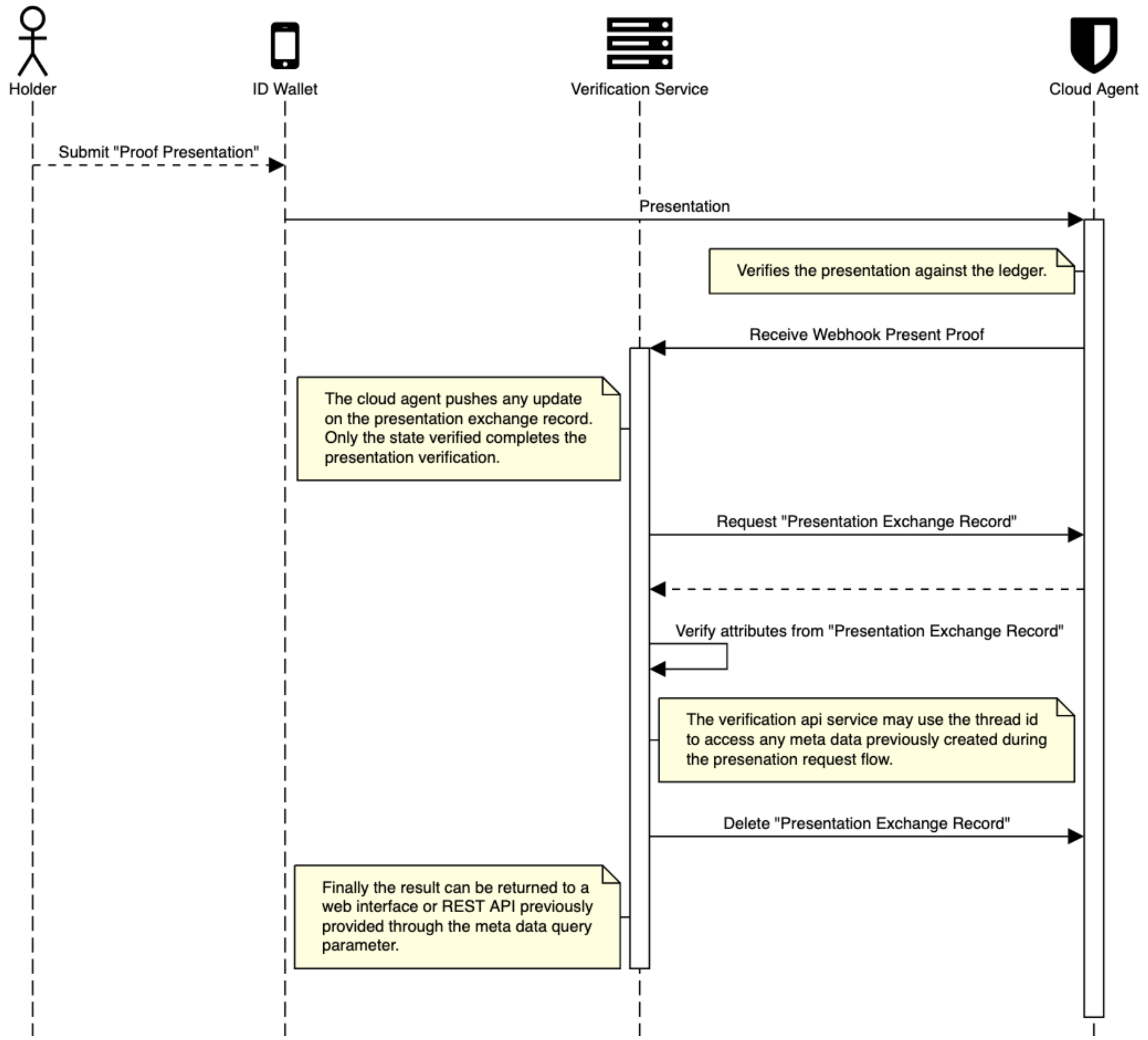
ⓘ **Redirect Location**

Please note that the redirect location is actually ignored by the id wallet. Only the "m" query parameter is decoded (base64) and processed. The proof presentation is submitted to the agent specified on the "serviceEndpoint" annotation.

See also example at Proof Service - Reference Implementation

Presentation

The following diagram illustrates the sequence of how a presentation gets verified through the system.


Holder
ID Wallet
Verification Service
Cloud Agent
Submit "Proof Presentation"
Presentation
Verifies the presentation against the ledger.
Receive Webhook Present Proof
The cloud agent pushes any update on the presentation exchange record. Only the state verified completes the presentation verification.
Request "Presentation Exchange Record"
Verify attributes from "Presentation Exchange Record"
The verification api service may use the thread id to access any meta data previously created during the presenation request flow.
Delete "Presentation Exchange Record"
Finally the result can be returned to a web interface or REST API previously provided through the meta data query parameter.

Create Presentation

After the presentation request gets approved by the user (Holder) the id wallet creates a presentation for the proof request and follows the redirect provided by the verification service - see also Presentation. Again the Presentation implements the RFC 0056 Service Decorator and RFC 0008 Threading linking the presentation to a unique thread id. Afterward, the presentation is sent to the service endpoint specified in the ~service.serviceEndpoint tag and encrypted using the public key specified in the *~service.recipientKeys*.

Verify Presentation

After receiving the presentation from the ID Wallet the cloud agent verifies the proof presentation against the ledger and informs the verification service through webhooks about any state change (`presentation_received` and `verified)` of the presentation exchange record. Only after the presentation has been verified the proof presentation can be trusted.

> ⓘ In order to receive notifications on received and verified presentations, the verification service needs to implement a webhook on `/topic /present_proof` - accessible by the cloud agent. The webhook endpoint must be secured, as otherwise fake webhooks could be sent. The ACA-Py allows for specifying an API key for webhooks through appending a #<<API_KEY>> to the --webhook-url parameter, see IG001 ACA-PY Installation and Setup, Testnetwork Configuration.

The following checks are performed by the Cloud Agent to verify the proof presentation.

- **Restrictions**: Involved schemas, credential definitions, public DIDs exist on-chain and match the expected ones specified in the restriction in the proof request.

- **Signatures**: Proof Presentation Attributes are signed with the signing keys specified in the credential definitions on the ledger.
- **Predicate Proofs**: e.g. >= 18
- **Revocation**: Verify proof of non-recovation by reading the revocation registry state (accumulator) from the ledger.
- **Link Secret**: The Holder knew the link secret contained in the verifiable credentials involved in the blinded form.

Once the presentation has been verified, the business logic for verifying the requested attributes or predicates can be applied. At this point, the verification finishes, and the calculated result can be pushed to the request originator.

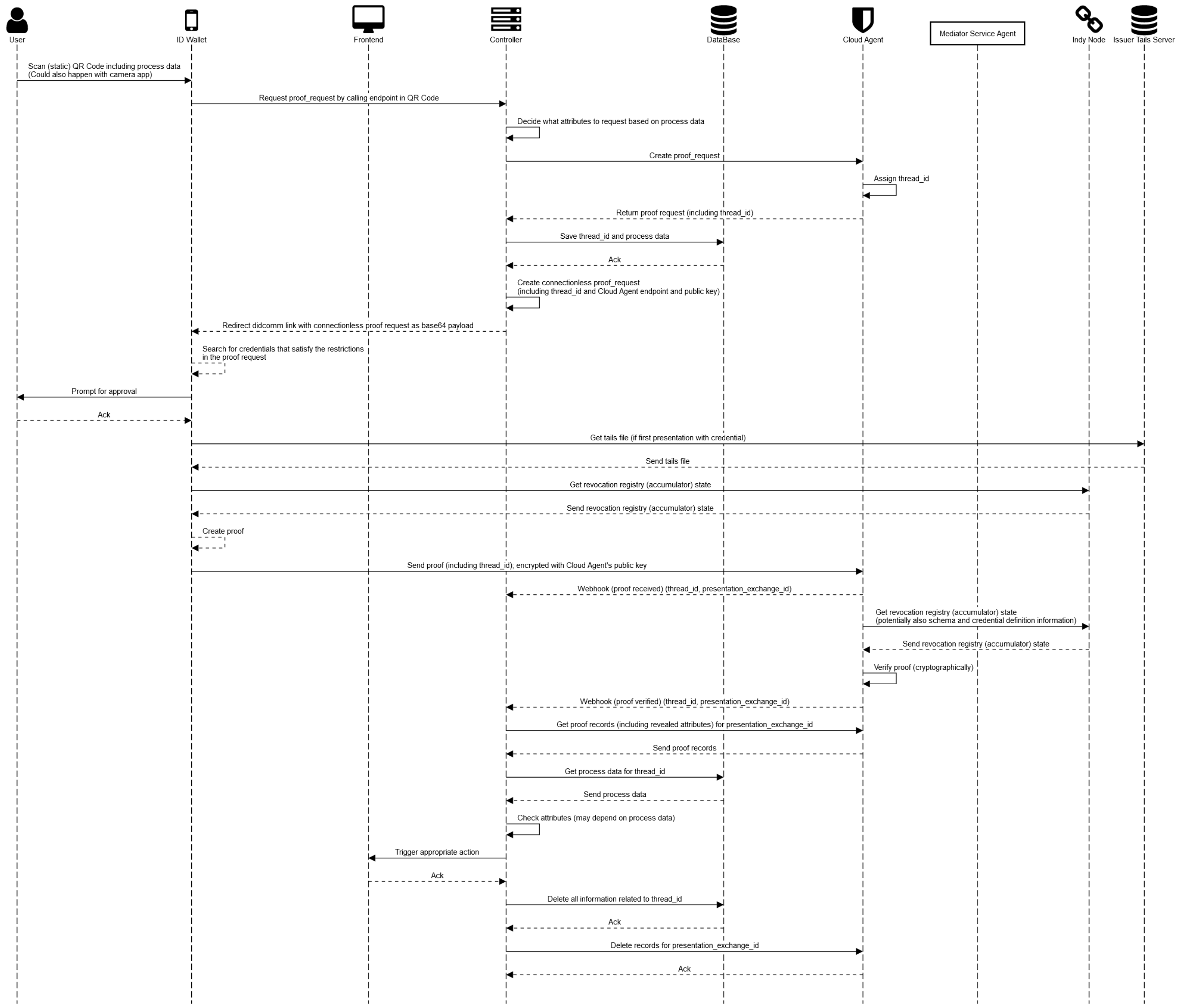
Open Topics

- Implement RFC 0035 Report Problem for error handling.
- Add service endpoint verification - check if the requestor is a trusted participant on the network. see also ARC002 Verify Service Endpoint

Appendix

```
{
"comment": "string",
"proof_request": {
  "name": "Proof request",
  "non_revoked": {
    "to": 1623763074
  },
  "nonce": "1234567890",
  "requested_attributes": {
    "masterId": {
    "names": ["firstName", "familyName", "addressStreet", "addressZipCode", "addressCountry", "addressCity",
"dateOfExpiry", "dateOfBirth"],
    "non_revoked": {
       "to": 1623763074
    },
    "restrictions": [{ "masterId": "XwQCiUus8QubFNJPJD2mDi:3:CL:29:masterID Dev" }]
    }
  },
  "version": "1.0"
  },
  "trace": false
}
```

```
{
  "auto_present": false,
  "connection_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "created_at": "2021-06-15 13:17:54Z",
  "error_msg": "",
  "initiator": "self",
  "presentation": {},
  "presentation_exchange_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "presentation_proposal_dict": {},
  "presentation_request": {},
  "presentation_request_dict": {},
  "role": "verifier",
  "state": "request-sent",
  "thread_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "trace": true,
  "updated_at": "2021-06-15 13:17:54Z",
  "verified": "true"
}
```


User
ID Wallet
Frontend
Controller
DataBase
Cloud Agent
Mediator Service Agent
Indy Node
Issuer Tails Server
Scan (static) QR Code including process data
(Could also happen with camera app)
Request proof_request by calling endpoint in QR Code
Decide what attributes to request based on process data
Create proof_request
Assign thread_id
Return proof request (including thread_id)
Save thread_id and process data
Ack
Create connectionless proof_request
(including thread_id and Cloud Agent endpoint and public key)
Redirect didcomm link with connectionless proof request as base64 payload
Search for credentials that satisfy the restrictions
in the proof request
Prompt for approval
Ack
Get tails file (if first presentation with credential)
Send tails file
Get revocation registry (accumulator) state
Send revocation registry (accumulator) state
Create proof
Send proof (including thread_id); encrypted with Cloud Agent's public key
Webhook (proof received) (thread_id, presentation_exchange_id)
Get revocation registry (accumulator) state
(potentially also schema and credential definition information)
Send revocation registry (accumulator) state
Verify proof (cryptographically)
Webhook (proof verified) (thread_id, presentation_exchange_id)
Get proof records (including revealed attributes) for presentation_exchange_id
Send proof records
Get process data for thread_id
Send process data
Check attributes (may depend on process data)
Trigger appropriate action
Ack
Delete all information related to thread_id
Ack
Delete records for presentation_exchange_id
Ack

- working version -

ART002 Multiple Proof Requests (without Connection)

Version	Date	Author	Comment	Status
0.1	14.07.2021	[redacted]	Initial version	DRAFT
0.2	21.07.2021	[redacted]	Adaption to other ART formats	DRAFT
0.3	06.08.2021	[redacted]	Minor formatting changes	IN REVIEW
0.4	24.08.2021	[redacted]	Improvements, moving architectural decision parts to Architectural Decisions#ARC003MultipleProofRequests	IN REVIEW
1.0	25.08.2021	[redacted]	Reworked the sequence diagram and the considerations.	APPROVED
1.1	02.09.2021	[redacted]	Transferring ownership to [redacted]	APPROVED

Table of Contents

Why are Multiple Proof Requests Needed?

Hyperledger Aries and compatible wallets allow to specify different restrictions for credentials from which attributes can be revealed. This allows for selective disclosure, combining several verifiable credentials. However, there are cases in which

- the verifier expects an attribute that is named differently on a multitude of credentials that can be chosen according to the restrictions, for example, "lastName" on the German Base-ID or "familyName" on a foreign Base-ID
- where some credentials/attributes are optional as they can also be revealed in another way. For example, showing a vaccination certificate may be manadory, but the holder may additionally reveal a digital Base-ID so skip the examination of the physical ID-card

This illustrates that there is a need for the verifier to send multiple proof requests at a time, and the prover should have the opportunity to select one of the proof requests and to answer it. However, so far there is no protocol to support this. One could of course process multiple requests sequentially, but this is extremely inconvenient and not user-friendly.

Multiple Proof Requests Protocol

This architectural template describes how multiple proof requests can be sent from the aca-py to the wallet app. Since so far Aries does not provide a standard for this purpose (see RFC 0037 Present Proof), the aca-py does not implement the functionality, and existing approaches have to be modified with a custom standard.

> Please note: To fulfill those modified requests, the wallet app to be used must also be adjusted.

Redirect w/ deeplink

As decided in Architectural Decisions#ARC003MultipleProofRequests, we send the connectionless proof requests via a redirect with a deeplink. This is the most flexible approach as it is already implemented and has a structure that can easily be adapted to an array, and consists of a small modification of the connectionless proof request that is used in many use cases, combining RFC 0034 URL Shortening with RFC 0037 (for details, see ART001 Verify SSI Credentials).

Hence, we suggest going for this approach. This only requires small changes on the verification backend controller: In the current verification implementation (see ART001 Verify SSI Credentials), the connectionless proof request that is redirected when scanning the verification QR code with the holder's ID wallet app contains the following information (in base64-encoded, serialized form):

```
{
    "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/present-proof/1.0/request-presentation",
    "@id": "4545138c-c93a-417c-8c1d-2e525932c7d2",
    "request_presentations~attach": {
        "@id": "libindy-request-presentation-0",
        "mime-type": "application/json",
        "data": {
            "base64": "eyJuYW1lIjogIlByb29mIHJlcXV...=="
        }
    },
    "~service": {
        "recipientKeys": [
            "BhcfSLbJQp7z4AwfBgnWZBCvpY7pAB2r4o2bMz7ZKtwG"
        ],
        "routingKeys": [
        ],
        "serviceEndpoint": "http://158.177.245.253:10000"
     },
    "~thread": {
        "thid": "4545138c-c93a-417c-8c1d-2e525932c7d2",
        "sender_order": 0,
        "received_orders": {}
    }
}
```

This JSON is built via retrieving id/thid ("thread id") according to RFC 0011 and data/base64 from the response of an API call to */request-proof/create-request*, according to RFC 0034 and RFC 0037.

Our solution approach is

1. to replace this single JSON object with an array of JSON objects that represent the various proof requests and
2. to add an integer-valued ~order key (ranging from 0 to n-1) to indicate the preference/order in the user dialog in the wallet (which may not be determined by the sequence in the array of JSON objects). This can be understood as a custom decorator, following RFC 0011.

This represents a custom modification of the standard. In parallel, we bring this suggestion into the working group.

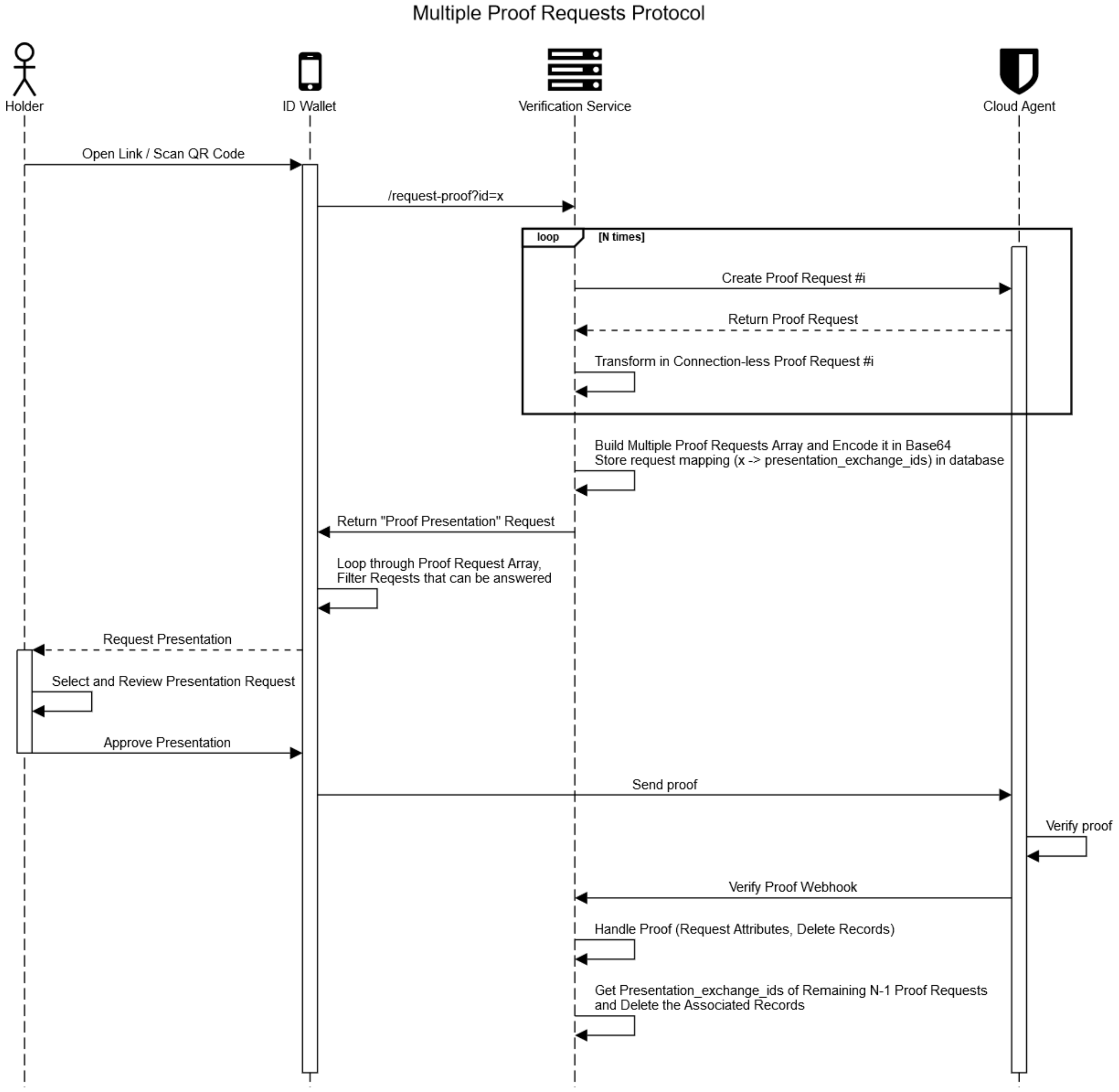
Example:

```
[
    {
        "@type":"did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/present-proof/1.0/request-presentation",
        "@id":"4545138c-c93a-417c-8c1d-2e525932c7d2",
        "request_presentations~attach": {
            "@id":"libindy-request-presentation-0",
            "mime-type":"application/json",
            "data": {
                "base64":"eyJuYW1lIjogIlByb29mIHJlcXV...=="
            }
        },
        "~service": {
            "recipientKeys": [
                "BhcfSLbJQp7z4AwfBgnWZBCvpY7pAB2r4o2bMz7ZKtwG"
            ],
            "routingKeys": [
            ],
            "serviceEndpoint":"http://158.177.245.253:10000"
        },
        "~thread":{
            "thid":"4545138c-c93a-417c-8c1d-2e525932c7d2",
            "sender_order": 0,
            "received_orders": {
            }
        },
        "~order": 0
    },
    {
        "@type":"did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/present-proof/1.0/request-presentation",
        "@id":"4545138c-c93a-417c-8c1d-2e525932c7d2",
        "request_presentations~attach": {
            "@id":"libindy-request-presentation-0",
            "mime-type":"application/json",
            "data": {
                "base64":"eyJuYW1lIjogIlByb29mIHJlcXV...=="
            }
        },
        "~service":{
            "recipientKeys": [
                "BhcfSLbJQp7z4AwfBgnWZBCvpY7pAB2r4o2bMz7ZKtwG"
            ],
            "routingKeys": [
            ],
            "serviceEndpoint":"http://158.177.245.253:10000"
        },
        "~thread": {
            "thid": "4545138c-c93a-417c-8c1d-2e525932c7d2",
            "sender_order": 0,
            "received_orders": {
            }
        },
        "~order": 1
    }
]
```

This approach comes with one inefficiency: For creating an **array of n** connectionless proof requests, the aca-py API endpoint *request-proof/create-request* needs to be invoked **n** times. We will need to determine whether this causes considerable overhead, but we expect that this is not the case as on the side of the aca-py, no new keypairs or local DIDs need to be generated, so essentially the aca-py just needs to create a new thread-id that contains the proof request JSON (attributes and restrictions that are required).

The controller hence repeats the procedure for creating a connectionless proof request as described in ART001 Verify SSI Credentials and builds the array of jsons for the multiple proof requests. This array is encoded in base64, attached as payload to an URL (d_m=<<base64-encoded) and forwarded to the mobile wallet (e.g. through URL-shortening or a redirect). The wallet will decode the payload and realize that there is an array of proof requests, iterate through all of them and check which of the proof requests it can satisfy. If there is exactly one proof request that can be satisfied, it will prompt the user like in the standard case; otherwise, the user will be able to choose from a dropdown menu which combination of credentials he or she wants to choose. It remains to be discussed whether the verifier should be able to attach some description to the proof request, for example, through the "comment" in the proof request that is encoded in "request_presentations~attach"."data"."base64". After the user has selected exactly one proof request that he or she wants to answer, the wallet proceeds like in the standard case and sends the proof back to the aca-py, where it is verified in the standard procedure. The controller then needs to take care that non-used presentation exchange records that were created when generating the proof request are deleted at some time so the aca-py does not need to persist unnecessary information.

In a sequence diagram, the process would look as follows:


Multiple Proof Requests Protocol
Holder
ID Wallet
Verification Service
Cloud Agent
Open Link / Scan QR Code
/request-proof?id=x
loop
[N times]
Create Proof Request #i
Return Proof Request
Transform in Connection-less Proof Request #i
Build Multiple Proof Requests Array and Encode it in Base64
Store request mapping (x -> presentation_exchange_ids) in database
Return "Proof Presentation" Request
Loop through Proof Request Array,
Filter Reqests that can be answered
Request Presentation
Select and Review Presentation Request
Approve Presentation
Send proof
Verify proof
Verify Proof Webhook
Handle Proof (Request Attributes, Delete Records)
Get Presentation_exchange_ids of Remaining N-1 Proof Requests
and Delete the Associated Records

Cleanup Presentation Exchange Records

With this design, there will be many unused presentation_exchange_records that are initialized by the cloud agent but that are never used. In the good case, 1 out of N proof requests is answered, in the bad case, 0 out of N proof requests is answered. To keep the load on the cloud agent as low as possible and to reduce the storage capacity to the minimum, presentation_exchange_records need to be deleted as soon as it becomes clear that they are not relevant anymore. A two-step process can achieve this:

- Have the controller store all thread-ids that are aggregated in one proof request array when it is created, and as soon as a proof response arrives with one specific thread id, delete all the remaining proof request preparations in the cloud agent mapped to the verification ID. As the "standard process" involves deleting the presentation_exchange_records for the proof request that is answered, too, this achieves that all records associated with proof requests that are successfully answered will be deleted on a continuous basis.
- Additionally, to remove proof requests that are never answered, periodically (e.g., every 10 minutes) there should be a job that queries the cloud agent's proof request records and deletes all of them that have a creation date that is older than a specific threshold after which a proof request should be answered (e.g., 10 minutes).

ART003 Device Binding Verification

Version	Date	Author	Comment	Status
0.1	15 Jul 2021	[redacted]	Initial version	DRAFT
0.2	21 Sep 2021	[redacted]	Outline, Verification flow Diagram, Drafts for EXT01, EXT02	DRAFT
0.3	22 Sep 2021	[redacted]	Sample Payloads & Pseudo-Code, minor wording changes, release for review	IN REVIEW
1.0	28 Sep 2021	[redacted]	Incorporating reviewer feedback	APPROVED

Table of Contents

Abstract

This document is an extension to ART001 Verify SSI Credentials, describing additional strategies to tie an issued credential to a device, leveraging hardware security functionalities.

Key takeaways

- An **issuer** creates credentials, which are signed and cryptographically coupled with a device.
- A **holder's** issued credentials can only be stored and presented with the device, credentials got issued for.
- A **verifier** *may* leverage the additional information for performing the verification of a presented credential (i.e. optionally verify the hardware proof, presented by the holder).
- This mechanism decreases the probability of credentials being copied onto other devices. The processes of re-issuing credentials in case of device losses and revocation are not affected. Device-bound credentials can only be issued on one device at a time.
- The addition of **Multi-Factor Authentication (MFA)** increases security, that only an authenticated credential owner can unlock and present a given credential.

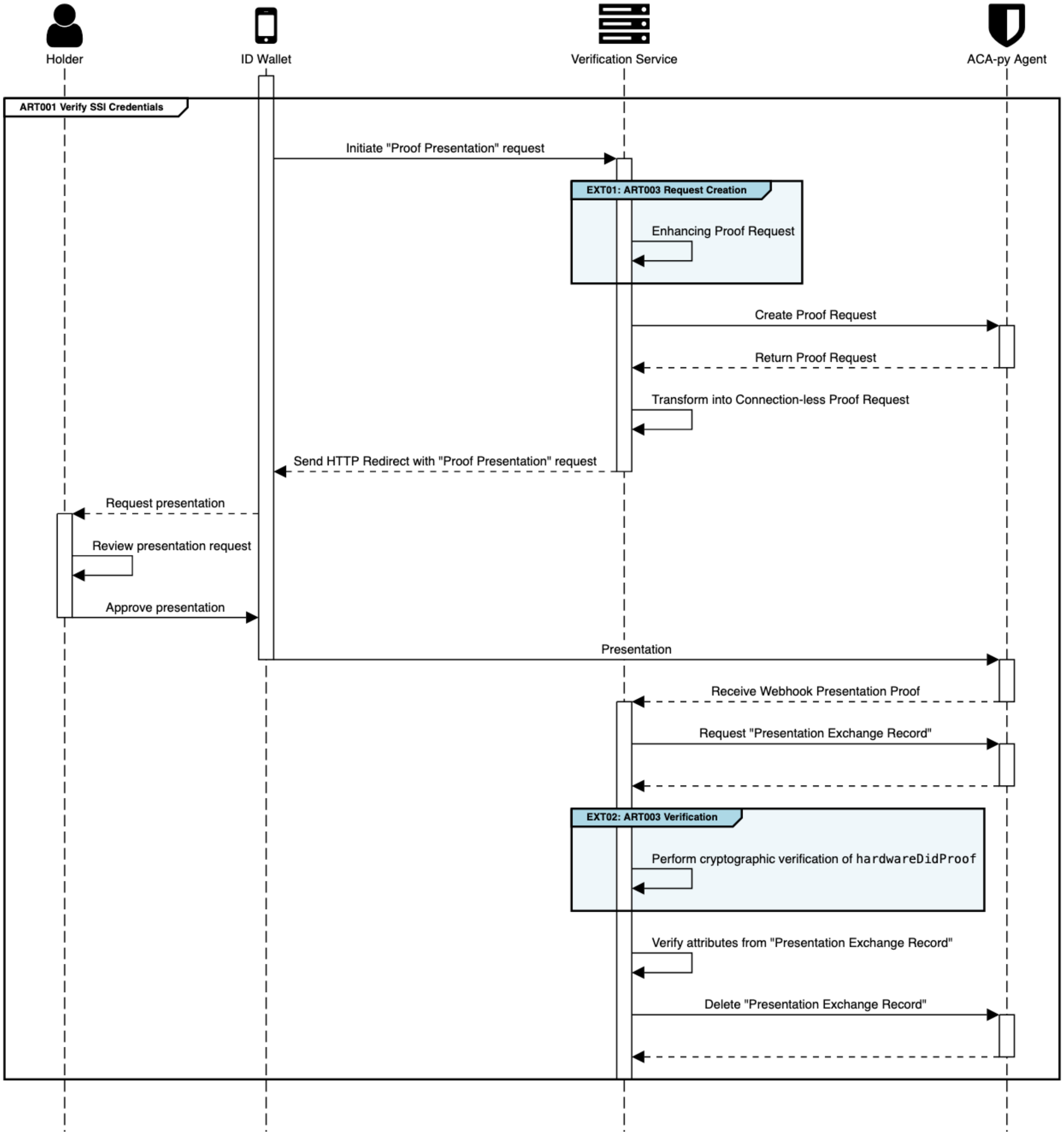
Out of scope

This template focuses on the **verification** of device-bound credentials. The issuing process (ART004 Issue SSI Credentials) as well as the details of how credentials are cryptographically signed and persisted on the mobile devices are not being covered here.

Solution Design

The following diagram illustrates how the ART001 Verify SSI Credentials is extended for the verification of device-bound credentials. Please note, that device binding is only necessary for sovereign credentials like the basis-id or digital driver license an is not mandatory for any other non-sovereign credential.

Verification flow for device-bound credentials (ART001 extension)


Holder
ID Wallet
Verification Service
ACA-py Agent
ART001 Verify SSI Credentials
Initiate "Proof Presentation" request
EXT01: ART003 Request Creation
Enhancing Proof Request
Create Proof Request
Return Proof Request
Transform into Connection-less Proof Request
Send HTTP Redirect with "Proof Presentation" request
Request presentation
Review presentation request
Approve presentation
Presentation
Receive Webhook Presentation Proof
Request "Presentation Exchange Record"
EXT02: ART003 Verification
Perform cryptographic verification of hardwareDidProof
Verify attributes from "Presentation Exchange Record"
Delete "Presentation Exchange Record"

Source: Gitlab

EXT01: ART003 Request Creation

1. Generate 80-bit nonce, set attribute `nonce`
2. Request schema attribute `hardwareDid`
3. Request self-attested attribute `hardwareDidProof`

Sample Proof Request for device-bound credentials

```
{
    "name": "Sample Proof Request for device-bound credentials",
    "nonce": "309831176642550496318983",
    "requested_attributes": {
        "checkin": {
            "names": [
                "firstName",
                "familyName",
                "addressStreet",
                "addressZipCode",
                "addressCountry",
                "addressCity",
                "dateOfExpiry",
                "dateOfBirth",
                "hardwareDid"
            ],
            "non_revoked": {
                "to": 1628846466
            },
            "restrictions": [{
                "cred_def_id": "MGfd8JjWRoiXMm2YGL4SGj:3:CL:43:Basis-ID Testnetzwerk"
            }]
        },
        "hardwareDidProof": {
            "name": "hardwareDidProof"
        }
    },
    "requested_predicates": {
    },
    "version": "0.1"
}
```

EXT02: ART003 Verification

To cryptographically verify the received hardwareDidProof attribute, proofing that the presented credential has been issued for the holder's device only, a series of instructions have to be executed:

1. Read attribute **`nonce`** and transform its value into **`hashedNonce`** by
    a. First, perform a Base64 decoding operation,
    b. second, appending `0x2` in the end, and
    c. third, perform a `SHA256` operation.
2. Read attribute **`hardwareDidProof`** and perform a Base64 decoding operation.
3. Read attribute **`hardwareDid`** and
    a. Extract the public key from its prefix **`did:key:`** , and
    b. perform a Base58 decoding operation.
4. Initialize an elliptic curve decoder, using **`secp256r1`**
5. Decode point for **`hardwareDidKey`** on elliptic curve
6. Generate public key from the decoded point, using the algorithm **`EC`**
7. Validate signature, using the algorithm **`SHA256withECDSA`**
   `Public Key + hashedNonce == hardwareDidProof`

Pseudo code example

```
// Read attribute nonce and transform its value into hashedNonce
decodedNonce = Base64.getDecoder().decode(nonce); // nonce = 4243494866520320281930 32
decodedNonce +=
'0x2';

hashedNonce = MessageDigest.getInstance("SHA-256").digest(decodedNonce);

// Read attribute hardwareDidProof and perform a Base64 decoding operation
Base64.getDecoder().decode(hardwareDIDProof); // hardwareDIDProof = MEQCIGBNInSnPwBCP9m0c4WY9xt+aCQgJvBKSuYw2f6q
/87eAiAwWbIdq4AZC8Q3X42v97eJhCJ9iCqxZMZmFdwgYYzdSw==

// Read attribute hardwareDid and extract the public key from its prefix did:key:
idx = hardwareDid.lastIndexOf(":"); // hardwareDID = did:key:zDnaevDEspHKH2XvV5WpMpi4rcZ2yjPoZkv5jeB2j6dHbG5LE
b58Did = hardwareDid.substring(idx + 2);
Base58.decode(b58Did);

// Initialize an elliptic curve decoder, using secp256r1
spec = ECNamedCurveTable.getParameterSpec('secp256r1');

// Decode point for hardwareDidKey on elliptic curve
ECPointUtil.decodePoint(ecparams.getCurve(), hardwareDIDKey);

// Generate public key from the decoded point, using the algorithm EC
KeyFactory.getInstance('EC').generatePublic(new ECPublicKeySpec(point, ecparams));

// Validate signature, using the algorithm SHA256withECDSA
ecdsaVerify = Signature.getInstance(SIGNATURE_ALGORITHM);
ecdsaVerify.initVerify(publicKey);
ecdsaVerify.update(hashedNonce);
ecdsaVerify.verify(hardwareDidProofValue);
```

Appendix

References

- ART001 Verify SSI Credentials
- ART006 Self-Attested attributes

Cryptography

Both attributes, `hardwareDid` and `hardwareDidProof` are leveraging on:

- NIST P256 / nist-p256r1
- SHA256withECDSA
- Key generation
  - Algorithm: Elliptic Curve
  - Signature: SHA256withECDSA
  - Device owners must provide their PIN to access the generated keys
  - Parameters being used:

Digest	DigestSha256
AlgorithmParameterSpec	secp256r1
SetRandomizedEncryptionRequired	false
SetUserAuthenticationValidityDurationSeconds (target API Level < 30)	-1
SetIsStrongBoxBacked	true, if available

- Key verification (Android only)

attestationVersion	>= 2
attestationSecurityLevel	TrustedEnvironment (1) or StrongBox (2)

keymasterSecurityLevel	TrustedEnvironment (1) or StrongBox (2)

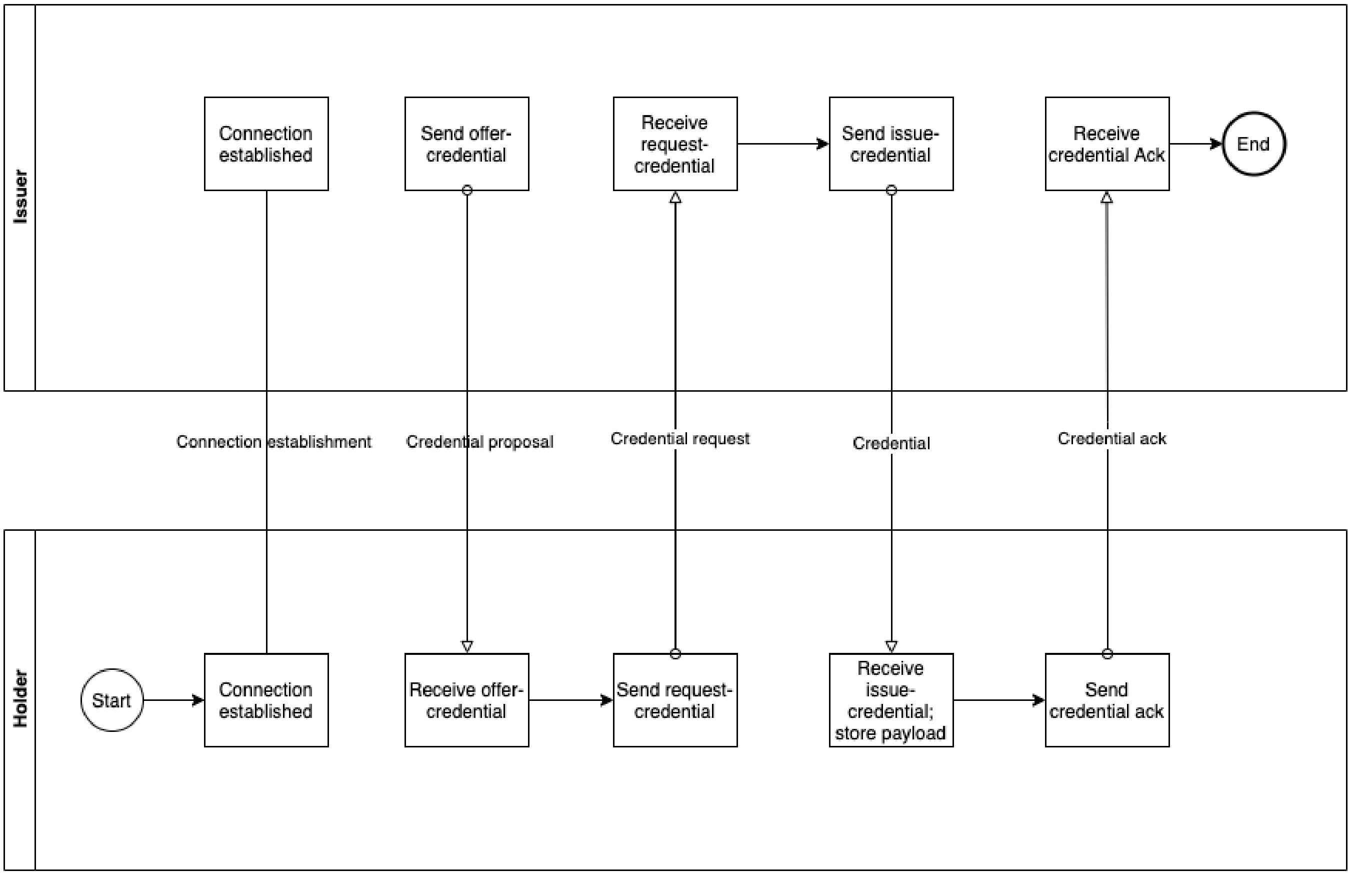
ART004 Issue SSI Credentials

Version	Date	Author	Comment	Status
0.11	14.09.2021	[redacted]	Compressed revision history	DRAFT
0.11	16.09.2021	[redacted]	Improved formatting and structure	DRAFT
0.11	29.09.2021	[redacted]	Added 2 separated diagram for issue and offer credentials. Ready for review and approval	DRAFT

Table of Contents

Introduction

This architectural template describes how SSI credentials are issued using the RFC0160 Connection Protocol and the RFC0036 Issue Credential Protocol 1.0. The diagram below illustrates the choreography implemented from the RFC0160 and RFC0036 (refer to Issue Credential Choreography Diagram).



Source

The following steps describe the high-level interaction flow of establishing a connection and issuing a credential.

- The Issuer (Issuing Service) gets a connection invitation from its Cloud Agent and delivers it to the Holder (User) who is to be issued a credential
- The Holder (User) accepts the invitation with their ID Wallet
- As soon as the Issuing Service gets a notification about the successful connection with the Holder (User) from their Cloud Agent, the issuance process is initiated.
- The issuance process starts with a credential offer that the Holder (User) needs to accept with their ID Wallet; if so, the user requests a credential (giving the issuer the blinded link secret for signing) and then receives the credential from the issuer
- The user then accepts the credential and the wallet stores it.
- After that, the Issung Service deletes sensitive information and only stores information that is necessary for later revocation.

Establishing a connection

Before a credential can be issued to the Holder (user) a connection between the Issuer and the Holder has to be established (see invitation example **ConnectionInvitationResponse** in the appendix). This is achieved through an end-to-end encrypted communication channel according to the RFC0160 Connection Protocol. A connection invite includes the following key attributes.

- **Endpoint**: The endpoint of the issuers agent.
- **Public Key**: The public key (or verkey) of the issuers agent.

This information may be provided via a link or encoded within a QR code contained in an e-mail or on a poster in the issuer's office.

It contains the inviting Cloud Agent's IP address and endpoint ("serviceEndpoint") and public key ("recipientKeys[-1]") either of the public DID or a newly generated peer DID) for end-to-end encrpytion. The "label" will be displayed on the accepting user's ID wallet; the "connection_id" is only for internal lookup and recognition. The issuer can add a key value pair "imageUrl": "http://link-to-image.{png, jpg}" to display the issuer's icon in the ID wallet. In this case, the invitation_url needs to be recreated, as the playload is the base64-encoded invitation json. For the invitation_url, only the base64 payload after c_i= is relevant for the wallet; the first part is ignored. It is possible to convert it to a link in the browser that the user can click on their mobile phone by redirecting the invitation_url where http:// is replaced by didcomm:// (similar to the process in ART001 Verify SSI Credentials).

The Cloud Agent will send three webhooks to the Issuing service; after creating the invitation, when getting a response from the ID Wallet, and when the connection has been established successfully. An example for this webhooks can be found in **establishConnectionWebhook**. The "state" will first be "invitation", then "request", then "response". When "state" is equal to "response", the connection is successfully established and the Issuer Controller can trigger the issuance of the credential.

Issue Credential Protocol

Once the connection is established, the issuer can offer the user a credential, and if the prospective holder accepts, the issuer signs the credential and sends it to the user, who stores the credential in their ID wallet.

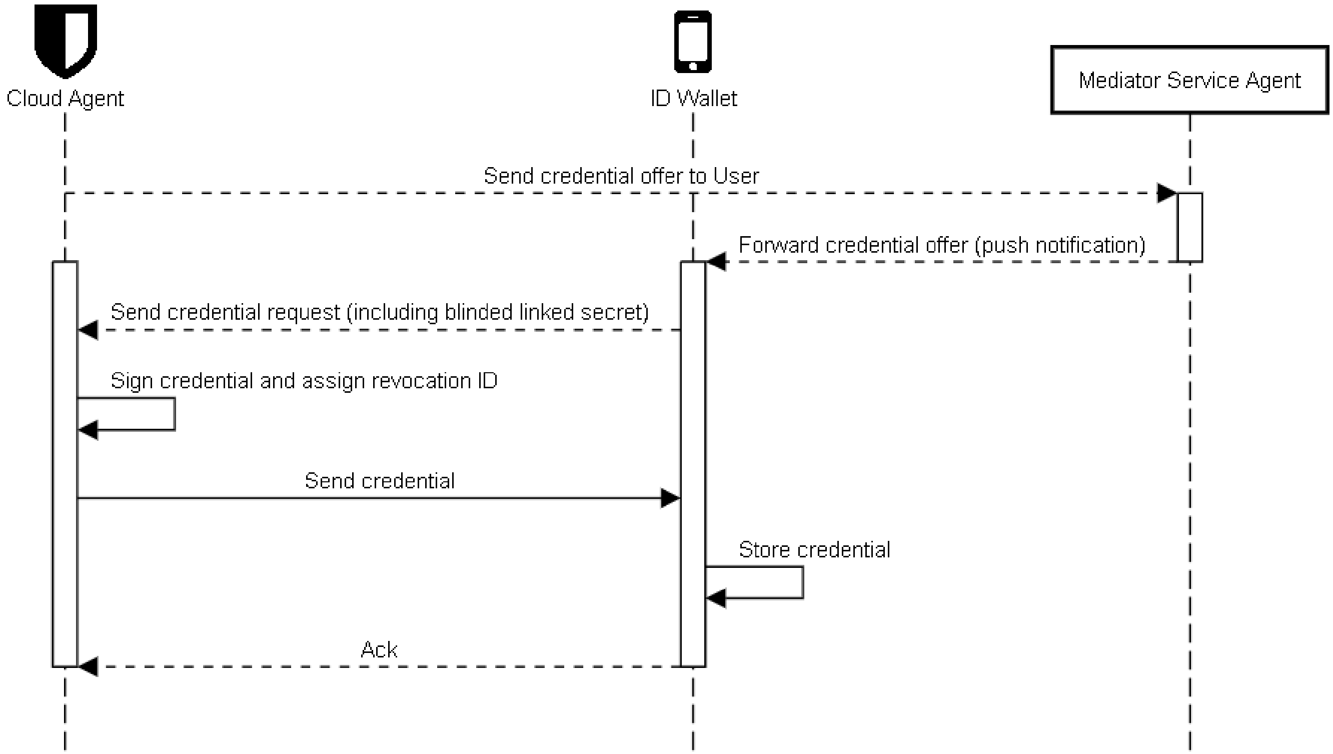
This follows the RFC0036 Issue Credential Protocol 1.0. Issuing the credential consists of multiple steps (three-round interaction). The reason is that the issuer cannot just send the credential to the holder because for holder binding, the holder's blinded link secret needs to be included. Consequently, the holder first gets a credential offer from the issuer; the holder then sends a "credential request" to the issuer, because he or she can get the credential. This process, however, is fully automated by using the Cloud Agent's issue-credential/send endpoint. The only action necessary from the issuer backend is to call the agent endpoint **issue-credential/send.** An example for the associated json body is given in **IssueCredentialRequest**. From the user perspective, the credential only needs to be accepted when prompted.

Offer Credential

This section starts when the cloud agent sends the credential offer to the user via a push notification

into the ID wallet. The credential can be stored once accepted and the process ends with the cloud agent receiving an Ack

message of the transaction.


Cloud Agent
ID Wallet
Mediator Service Agent
Send credential offer to User
Forward credential offer (push notification)
Send credential request (including blinded linked secret)
Sign credential and assign revocation ID
Send credential
Store credential
Ack

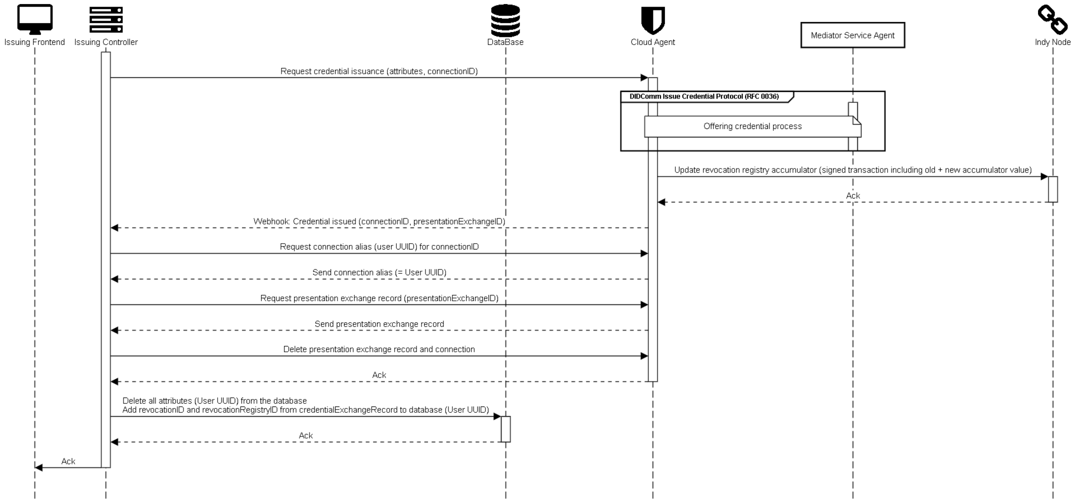
Issue Credential

This section starts when the issuing controller sends the request to the cloud agent which triggers the

DIDcomm protocol (RFC 0036) and updates the registry with the new accumulator value.

Finally, both systems will interchange the associated packages.

When the process is finished the issuing controller sends an Ack message to the issuing frontend .


Issuing Frontend
Issuing Controller
DataBase
Cloud Agent
Mediator Service Agent
Indy Node
Request credential issuance (attributes, connectionID)
DIDComm Issue Credential Protocol (RFC 0036)
Offering credential process
Update revocation registry accumulator (signed transaction including old + new accumulator value)
Ack
Webhook: Credential issued (connectionID, presentationExchangeID)
Request connection alias (user UUID) for connectionID
Send connection alias (= User UUID)
Request presentation exchange record (presentationExchangeID)
Send presentation exchange record
Delete presentation exchange record and connection
Ack
Delete all attributes (User UUID) from the database
Add revocationID and revocationRegistryID from credentialExchangeRecord to database (User UUID)
Ack
Ack

Issue SSI Credentials end to end

The issuance of the credential needs to be integrated with a database that holds the users to which a credential is to be issued and the corresponding attributes. The issuer also needs to persist information to be able to revoke credentials later on, this needs to be balanced with data minimization (the issuance components should store as least sensitive information as possible).

On a more detailed level, the issuance workflow looks as follows:

- The issuer stores information about users who should be issued a credential. This includes a user UUID in the legacy system that the issuer is connected to, the corresponding attributes as well as contact information (e.g., an e-mail address)
- Create a QR code from a connection invitation for a user and put the state to "in progress". The QR code needs to be delivered to the user, e.g., via an institutional e-mail address or an in-person visit. The connection must also allow for correlating the connection with the employee UUID. This is possible, for example, by using the UUID as alias when creating the connection invitation, or by mapping the connection_id that is generated when creating the connection invitation to the user UUID in the database
- Once the connection has been established, the attributes need to be taken from the database (lookup with UUID) and an issue-credential request must be sent to the agent
- When the credential has been successfully issued, the issuer controller is notified via a webhook ("state": "credential_issued"). The issuer can then retrieve the credential's revocation ID ("revocation_id") and revocation registry ID ("revoc_reg_id") which is important for later revocation. The issuer can also get these two values from querying the agent's credential_exchange_records with the "credential_exchange_id" from the webhok.
- Once the credential has been issued, all information from the database except for the mapping user UUID (revocation_id, rev_reg_id) can be deleted. Also the credential_exchange_record and connection should be deleted, and optionally the connection. This not only avoids the storage of sensitive information in multiple places but also keeps the Cloud Agent's performance stable in the long run.
- For details on how the revocation works, see ART007 Revoke Credential.

Prepare the Opportunity for Range Proofs on Issuance

Many credentials have attributes that are in some way numeric and for which on verification, range proofs are a desirable feature to support data minimization. Examples for this include

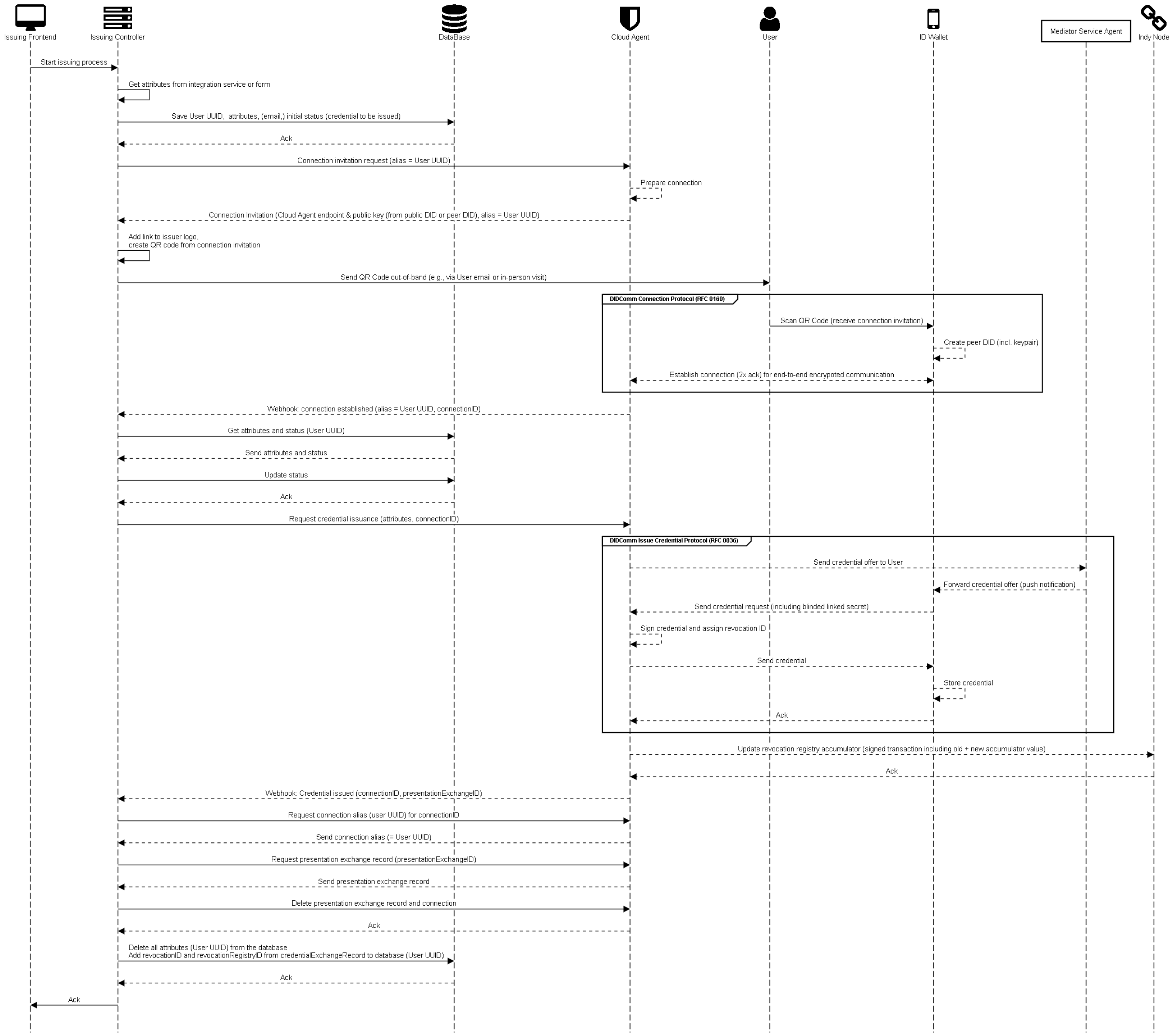
- a date of birth for later proofs of legal age
- a date for issuance and/or validity for later proofs of non-expiration
- a naturally ordered category, for example, "bronze, silver, gold, platinum", for proving at least a certain status

In this case, it is strongly recommended to convert the corresponding attribute into an integer value in a way that preserves ordering, as only range proofs for integer-valued attributes are supported by the Hyperledger Indy/Aries stack as well as many other specialized cryptographic proof protocols. The most natural, order-preserving transformation for dates would be a UNIX-timestamp, potentially "offset" to allow for dates earlier than 1970-01-01 to be represented. For "bronze, silver, gold, platinum", one could use a transformation to "1, 2, 3, 4".

Additional considerations:

- Human readability: Another order-preserving transformation for dates would be dd.mm.yyyy yyyymmdd, e.g., 20200906 for the 6th of September of 2020. This makes the date "closer" to human readable as long as wallets display the attribute value directly and no metadata how or what to display is in the credential
- Flexibility: Maybe at a later stage, information needs to get more detailed - imagine adding hours and seconds on a date or a new category "wood". In this case, take this into account from the beginning, for example, for a date, use yyyymmdd0000000000 to be able to add miliseconds later on, or "100, 1000, 10000, 100000" to represent "bronze, silver, gold, platinum", as this allows to later insert new intermediate categories without a need to change the schema and credential definition.

An even more detailed sequence diagram is illustrated in the following.


Issuing Frontend
Issuing Controller
DataBase
Cloud Agent
User
ID Wallet
Mediator Service Agent
Indy Node
Start issuing process
Get attributes from integration service or form
Save User UUID, attributes, (email,) initial status (credential to be issued)
Ack
Connection invitation request (alias = User UUID)
Prepare connection
Connection Invitation (Cloud Agent endpoint & public key (from public DID or peer DID), alias = User UUID)
Add link to issuer logo,
create QR code from connection invitation
Send QR Code out-of-band (e.g., via User email or in-person visit)
DIDComm Connection Protocol (RFC 0160)
Scan QR Code (receive connection invitation)
Create peer DID (incl. keypair)
Establish connection (2x ack) for end-to-end encrypoted communication
Webhook: connection established (alias = User UUID, connectionID)
Get attributes and status (User UUID)
Send attributes and status
Update status
Ack
Request credential issuance (attributes, connectionID)
DIDComm Issue Credential Protocol (RFC 0036)
Send credential offer to User
Forward credential offer (push notification)
Send credential request (including blinded linked secret)
Sign credential and assign revocation ID
Send credential
Store credential
Ack
Update revocation registry accumulator (signed transaction including old + new accumulator value)
Ack
Webhook: Credential issued (connectionID, presentationExchangeID)
Request connection alias (user UUID) for connectionID
Send connection alias (= User UUID)
Request presentation exchange record (presentationExchangeID)
Send presentation exchange record
Delete presentation exchange record and connection
Ack
Delete all attributes (User UUID) from the database
Add revocationID and revocationRegistryID from credentialExchangeRecord to database (User UUID)
Ack
Ack

© IBM Deutschland AG

References

- Issue Corporate ID
- Github hyperledger
- Aries RFCs workspace
- Aries-rfcs - Issuing credential
- Aries RFC 0015: ACKs
- Repository with above diagram source code

Appendix

ConnectionInvitationResponse

```
{
    "connection_id": "95e765ab-58d4-4db7-9b7a-35f9b1163dcf",
    "invitation": {
        "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/invitation",
        "@id": "36c3c4b1-19a5-4b80-b5e9-b15f94156cc4",
        "recipientKeys": [
            "XwyCNesDcmziaJ4iwFwDGgucYJVr62PDwNA91HX268p"
        ],
        "label": "IBM-Deutschland",
        "serviceEndpoint": "http://158.177.245.253:11000"
    },
    "invitation_url":  "http://158.177.245.253:11000?c_i=eyJAdHlwZSI6ICJkaWQ6c292OkJ6Q2JzTlloTXJqS..."
}
```

EstablishConnectionWebhook

```
{
    "their_role": "invitee",
    "accept": "auto",
    "state": "invitation",
    "invitation_mode": "once",
    "connection_protocol": "connections/1.0",
    "created_at": "2021-09-02 08:26:14.621224Z",
    "rfc23_state": "invitation-sent",
    "invitation_key": "56RefdTqFwpuYbk2rD51DcNpVWTPMdWSeyfh9TrrqLiP",
    "updated_at": "2021-09-02 08:26:14.621224Z",
    "routing_state": "none",
    "connection_id": "d736be2e-5c39-418c-9e4e-7b6ec056eb32"
}
```

IssueCredentialRequest

```
{
        "auto_remove": true,
        "comment": "string",
        "connection_id": "d736be2e-5c39-418c-9e4e-7b6ec056eb32",
        "cred_def_id": "Th7MpTaRZVRYnPiabds81Y:3:CL:8:default",
        "credential_proposal": {
                "@type": "issue-credential/1.0/credential-preview",
                "attributes": [
                        {
                                "name": "attribute1",
                                "value": "attributeValue1"
                        },
                        {
                                "name": "attribute2",
                                "value": "attributeValue2"
                        }
                ]
        },
        "issuer_did": credDefId.split(":")[0],
        "schema_id": "Th7MpTaRZVRYnPiabds81Y:2:MySchema:1.0",
        "schema_issuer_did": "Th7MpTaRZVRYnPiabds81Y",
        "schema_name": "MySchema",
        "schema_version": "1.0",
        "trace": false
}
```

IssueCredentialWebhook

IssueCredentialWebhook

```
{
  "credential_definition_id": "Th7MpTaRZVRYnPiabds81Y:3:CL:8:default",
  "credential_request": {
    "prover_did": "LCiGyab9u8ECgxDwvLWYBJ",
    "cred_def_id": "Th7MpTaRZVRYnPiabds81Y:3:CL:8:default",
    "blinded_ms": {
      "u":
"522892906443137588307028784653437530620754473691935311488458214035724284684125148431797325073540705265141091844
4312805473430272894985304352214338899494170270936550871792512378582146082259703907532348274288744249457608011498
0015350800591551009963581839925043170928749129006075654873119624669385997277281215308797568574651536302126134186
4444920400531776815378847345638453167059930344840071714706936776490242384183988481049465059811046029822825498134
8325370856169256677523480308232639085754520563474841783864817359601946677419215026953196838881210476787804973632
8097528565193009188367778443910980926433140268490148844472",
      "ur": "1 0BE1499739708D86A71E82608A43519940E772785D52D2B296D23F5739B7E2B9 1
04672589404EBE6C971A4960D7903306A8990453585FCFF78F720DE2026AEE2C 2
095E45DDF417D05FB10933FFC63D474548B7FFFF7888802F07FFFFFF7D07A8A8",
      "hidden_attributes": [Array],
      "committed_attributes": {}
    },
    "blinded_ms_correctness_proof": {
      "c": "115443037101094571308881598488252450770004155391801142890833530166781904865398",
      "v_dash_cap":
"216924218217301903047125659251435067038840750728355725157852431851142476616206960659907670284462669409691545029
1085985848589938353850527743068469890515475379576826600817673961906694553708631443770809527946721013803283347087
2119714301784248592698808430335348145519694291733232447994292625313062360536227284582862406021525452437413477852
7525696220036860664256412272336966961646668121893063362989639956526271314055084159924874558400245003979023347773
3342690933541196986696114164874263452200806279060324773065976412434691660142568150708154907323137908163518282827
8357271105545724843204046918031631204492069053741860401798611067109531170470753820210524502686274362989336339551
7848909026377453877486830469058983445525055485",
      "m_caps": "[Object]",
      "r_caps": {}
    },
    "nonce": "851594135276976202890412"
  },
  "created_at": "2021-09-02 12:54:12.338321Z",
  "auto_issue": true,
  "auto_offer": false,
  "state": "credential_issued",
  "schema_id": "Th7MpTaRZVRYnPiabds81Y:2:MySchema:1.0",
  "thread_id": "eed6f6f4-5889-4c62-8c0d-dd3ac6fa1561",
  "credential_proposal_dict": {
    "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/1.0/propose-credential",
    "@id": "dcba4cfa-60ed-468b-914a-d7cbac0c3b61",
    "schema_version": "1.0",
    "cred_def_id": "Th7MpTaRZVRYnPiabds81Y:3:CL:8:default",
    "schema_id": "Th7MpTaRZVRYnPiabds81Y:2:MySchema:1.0",
    "credential_proposal": {
      "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/1.0/credential-preview",
      "attributes": [Array]
    },
    "comment": "string",
    "schema_issuer_did": "Th7MpTaRZVRYnPiabds81Y",
    "schema_name": "MySchema",
    "issuer_did": "Th7MpTaRZVRYnPiabds81Y"
  },
  "initiator": "self",
  "credential_offer_dict": {
    "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/1.0/offer-credential",
    "@id": "eed6f6f4-5889-4c62-8c0d-dd3ac6fa1561",
    "~thread": {},
    "credential_preview": {
      "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/1.0/credential-preview",
      "attributes": [Array]
    },
    "offers~attach": [ [Object] ],
    "comment": "string"
  },
  "credential_offer": {
    "schema_id": "Th7MpTaRZVRYnPiabds81Y:2:MySchema:1.0",
    "cred_def_id": "Th7MpTaRZVRYnPiabds81Y:3:CL:8:default",
    "nonce": "994592954586458448602049",
    "key_correctness_proof": {
      "c": "96309727364538150943916096018081057463770662850061805621619285784987065395433",
      "xz_cap":
"214172595805056308695020525056439787211403110915263408936155152656454286382634788646092235110899457523320639459
0512626247239222867956706946423779471338979291686346090683523710454023036606394652980322575926703859860233651260
7945924924228656426887301407700779471984734195435875671750373321658194628800714151621289867619428429429848408009
6170287690706758117842536996307326356348695837267652375280084522023442985380759577074469169171020404156104166473
7252055755455243284142486741574142949843332894357403101362126565188835228519561003247543593157057009016228751912
5297169702663258511770892195353506109791047521412914906047935603749602556006557780184453290767776544812665625722
31280698128786698989781",
      "xr_cap": [Array]
    }
  },
  "credential": {
    "schema_id": "Th7MpTaRZVRYnPiabds81Y:2:MySchema:1.0",
    "cred_def_id": "Th7MpTaRZVRYnPiabds81Y:3:CL:8:default",
    "rev_reg_id": "Th7MpTaRZVRYnPiabds81Y:4:Th7MpTaRZVRYnPiabds81Y:3:CL:8:default:CL_ACCUM:98ec7cc1-d35f-44e1-
a7ae-417c0a27a710"
  },
  "role": "issuer",
  "connection_id": "2efcd8ba-3537-496f-ad41-41d0ed07fecb",
  "revoc_reg_id": "Th7MpTaRZVRYnPiabds81Y:4:Th7MpTaRZVRYnPiabds81Y:3:CL:8:default:CL_ACCUM:98ec7cc1-d35f-44e1-
a7ae-417c0a27a710",
  "trace": false,
  "updated_at": "2021-09-02 12:54:31.353152Z",
  "revocation_id": "1",
  "credential_exchange_id": "a1f5c166-1536-49af-814a-6394742f9ebb",
  "auto_remove": true
}
```

ART005 Verification Integration Pattern

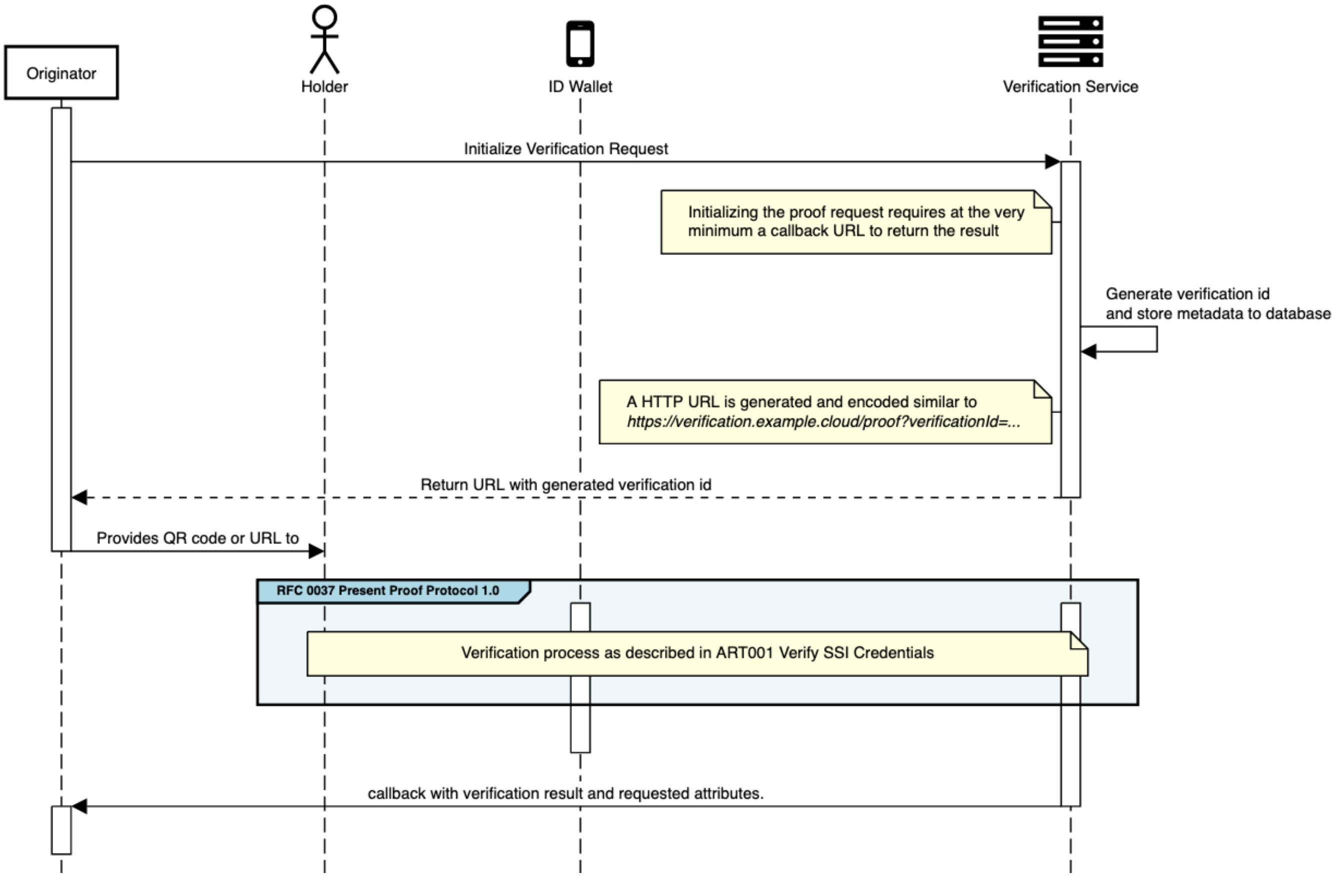
Version	Date	Author	Comment	Status
0.1	16.07.2021	[redacted]	Initial version	DRAFT
0.2	09.08.2021	[redacted]	Set document status to in review	IN REVIEW
0.3	19.08.2021	[redacted]	Added verification id to proof response schema and documented verification request response schema	IN REVIEW
1.0	07.09.2021	[redacted]	Incorporated review remarks	APPROVED

Table of Contents

Introduction

This document outlines a verification integration pattern describing how a third party back-end system can consume a SSI verification as a service. The aim is to abstract the complexities of implementing the Present Proof Protocol as described in ART001 Verify SSI Credentials with potential extensions of ART002 Multiple Proof Requests (without Connection) or ART003 Device Binding Verification. In the following the third party back-end system will be referred to as the *"Verification Request Originator"* or simply *"Originator".*

The following diagram illustrates the touch points of integrating with the Verification API Service. Firstly, inbound initializing the proof request meta data and secondly, outbound informing the originator about the proof request result. *Please note, that the actual sequence for creating and verifying the proof presentation has been left out here, as this is described in ART001 Verify SSI Credentials.*


Originator
Holder
ID Wallet
Verification Service
Initialize Verification Request
Initializing the proof request requires at the very minimum a callback URL to return the result
Generate verification id and store metadata to database
A HTTP URL is generated and encoded similar to *https://verification.example.cloud/proof?verificationId=...*
Return URL with generated verification id
Provides QR code or URL to
RFC 0037 Present Proof Protocol 1.0
Verification process as described in ART001 Verify SSI Credentials
callback with verification result and requested attributes.

Initialize Verification Request

Initially the Originator has to initialize a proof request. This is done by a simple HTTP call providing the meta data of the intended proof request. The following list provides examples of different kind of proof requests initialized through the meta data.

- ***Personalized Proof Request:*** A proof request providing firstname, lastname and birthdate in the meta data, allowing the verification to check whether the proof was provided by the requested person.
- ***Timed Proof Request:*** A proof request providing a valid until marking until when the proof request should be valid.
- ***Self Attested Proof Request:*** A proof request providing additional attributes self attested (without credential definition restriction) by the Holder.

ⓘ A combination of all of these different types of proof request is possible.

The following list describes the key aspects of initializing a proof request. (see also Appendix for the Meta Data Schema Definition)

- The HTTP endpoint is protected by an API key separately shared with the *Verification Request Originator* system.
- The HTTP call has to include the Meta Data within the POST body. *Please refer to the Appendix for the Meta Data Schema Definition*
- The meta data will be stored under a uniquely generated (UUIDv4) verification id.
- An HTTP error (400) is returned if the provided meta data is not compliant with the *Meta Data Schema Definition.*

Initialize Proof Request Example

```
curl -X POST -H 'X-API-KEY: 123123' https://ssi.verification.example.com/api/v1/init -d '{ "callbackUrl":
"https://callback.originator.com/49f03689-2ff3-4339-9a02-6a5acf1062b3" }'
```

Example - Successful verification request initialization

```
{
  "uri": "didcomm://ssi.example.com/proof",
  "verificationId": "936d6626-1b85-497e-b64e-518195b4747f"
}
```

see also Appendix for Verification Request Schema Definition.

Shortened URL Generation

The shortened URL is either encoded through a QR code or represented as a URL. The following shortened URL illustrates an example.

Shortened URL Example

```
https://verification.example.cloud/api/proof?verificationId=936d6626-1b85-497e-b64e-518195b4747f
```

The link consists of the following parts

- **Endpoint**: "verification.example.cloud/api/proof" The endpoint under which the presentation can be requested.
- **Query Parameter:** The query parameter *"verificationId"* which has been generated upon proof request initialization, correlating the received meta data to that unique proof request.

Please note, the query parameter is optional for ART001 Verify SSI Credentials but will always be provided if this integration pattern is implemented.

Request Presentation

This section describes the **extensions** to be implemented on the request presentation phase described in ART001 Verify SSI Credentials. Upon receiving a *Create Presentation Request* with a verification id, the Verification API Service queries the corresponding meta data from the database and performs the following evaluation.

1. **Timeout Check**: If valid until has been provided, the request is checked if it hasn't timed out yet. An HTTP error (408) is returned in case the QR is not valid anymore.
2. **Self Attested Attributes**: If self attested attributes have been provided, the proof request will additionally add the provided attributes to the presentation request without credential definition restrictions.

Callback

Once the proof presentation has been verified the Verification API Service parses the thread id (correlated with the verification id) from the *Presentation Exchange* and queries the corresponding meta data from the database. Based on the meta data, the Verification API Service may perform additional business logic for verification. Finally the response will be constructed according to the *Verification Response Schema Definition* and POSTED to the callbackURL found in the meta data.

Example - Successful verification, with the requested properties in the data tag.

```
{
  "code": 200,
  "verificationId: "b9525c1e-588d-4d9d-8325-8655c9fced29",
  "verified": true,
  "data": {
    "property1": "abc",
    "property2": "cde",
  },
  "message": ""
}
```

Error Handling

The following error scenarios may occur during the process of verifying the credential. This chapter describes how the Proof Request Originator will be informed about an error state.

- The response schema is based on *verification.response.schema.json* (see appendix)
- User errors are within the 4xx error range.
- System errors are within the 5xx error range.
- A successful verification evaluation is indicated with a 200 status code. However, the `data.verified` body has to be evaluated.
- In case of an error 4xx or 5xx, the returned body will be empty.

Examples

- The Holder rejects to share their credentials: In case the user rejects to share their credential, the process will stop. As of today, the ID Wallet app does not provide any feedback information to the Verification Service. The Verification Service system implements a timeout, though, in case no response is received from the wallet within 120 seconds, it will presume the proof request failed and will POST the following body to the `callbackURL`.

  ```
  { "code": 408, "message": "Request Timeout Error" }
  ```
- The Holder does not respond to the approval request: The user may remain inactive on the ID Wallet app and does not continue the process. The Verification Service will timeout after 120 seconds and presume the proof request failed. The following body with be send as POST to the `callbackURL`.

  ```
  { "code": 408, "message": "Request Timeout Error" }
  ```

Meta Data Invalidation

Any initialized metadata can be invalidated using the generated verification id. The following curl request illustrates an example how the API is to be used.

Initialize Proof Request Example

```
curl -X POST -H 'X-API-KEY: 123123' https://verification.example.com/api/v1/invalidate?verificationId=936d6626-1b85-497e-b64e-518195b4747f
```

The following list describes the key aspects of invalidating a verification request.

- The HTTP endpoint is protected by an API key separately shared with the *Verification Request Originator* system.
- The corresponding meta data will be deleted from the database upon invalidation, thus failing any proof presentation request afterwards.

Open Topics

- Define how to link thread id and verification id. Thread id is generated by ACAPY and is presentation request specific. There could be a 1:n relationship as the verification id (meta data) is used multiple times.
- Improve documentation on the Schema Definitions
- Describe Architectural Template (ART006) for creating proof requests with self attested attributes.

Appendix

```
{
    "$schema": "http://json-schema.org/draft-07/schema",
    "$id": "http://example.com/verification.request.response.schema.json",
    "type": "object",
    "title": "The verification request response schema definition",
    "description": "The response schema definition return upon a successful verification request.",
    "default": {},
    "examples": [
        {
            "uri": "didcomm://ssi.example.com/proof",
            "verificationId": "936d6626-1b85-497e-b64e-518195b4747f"
        }
    ],
    "required": [
        "uri",
        "verificationId"
    ],
    "properties": {
        "uri": {
            "$id": "#/properties/uri",
            "type": "string",
            "title": "The uri schema",
            "description": "The uri to redirect to the ID Wallet.",
            "default": "",
            "examples": [
                "didcomm://ssi.example.com/proof"
            ]
        },
        "verificationId": {
            "$id": "#/properties/verificationId",
            "type": "string",
            "title": "The verificationId schema",
            "description": "The verification id linked to the initialized meta data. This needs to be added as
query parameter to the uri. e.g. didcomm://ssi.example.com/proof?verificationId=936d6626-1b85-497e-b64e-
518195b4747f.",
            "default": "",
            "examples": [
                "936d6626-1b85-497e-b64e-518195b4747f"
            ]
        }
    },
    "additionalProperties": true
}
```

```
{
    "$schema": "http://json-schema.org/draft-07/schema",
    "$id": "http://example.com/verification.request.metadata.schema.json",
    "type": "object",
    "title": "The meta data schema definition",
    "description": "The general schema for exchanging meta data for a proof request processing.",
    "default": {},
    "examples": [
        {
            "callbackURL": "https://callback.example.com/123123",
            "validUntil": "2021-07-18T18:25:43.511Z",
            "selfAttested": [
                "email",
                "purposeOfStay"
            ],
            "data": {}
        }
    ],
    "required": [
        "callbackURL",
    ],
    "properties": {
        "callbackURL": {
            "$id": "#/properties/callbackURL",
            "type": "string",
            "title": "The callbackURL schema",
            "description": "An explanation about the purpose of this instance.",
            "default": "",
            "examples": [
                "https://callback.example.com/123123"
            ]
        },
        "validUntil": {
            "$id": "#/properties/validUntil",
            "type": "string",
            "title": "The validUntil schema",
            "description": "An explanation about the purpose of this instance.",
            "default": "",
            "examples": [
                "2021-07-18T18:25:43.511Z"
            ]
        },
        "selfAttested": {
            "$id": "#/properties/selfAttested",
            "type": "array",
            "title": "The selfAttested schema",
            "description": "An explanation about the purpose of this instance.",
            "default": [],
            "examples": [
                [
                    "email",
                    "purposeOfStay"
                ]
            ],
            "additionalItems": true,
            "items": {
                "$id": "#/properties/selfAttested/items",
                "anyOf": [
                    {
                        "$id": "#/properties/selfAttested/items/anyOf/0",
                        "type": "string",
                        "title": "The first anyOf schema",
                        "description": "An explanation about the purpose of this instance.",
                        "default": "",
                        "examples": [
                            "email",
                            "purposeOfStay"
                        ]
                    }
                ]
            }
        },
        "data": {
            "$id": "#/properties/data",
            "type": "object",
            "title": "The data schema",
            "description": "An explanation about the purpose of this instance.",
            "default": {},
            "examples": [
                {}
            ],
            "required": [],
            "additionalProperties": true
        }
    },
    "additionalProperties": true
}
```

```
{
    "$schema": "http://json-schema.org/draft-07/schema",
    "$id": "http://example.com/verification.response.schema.json",
    "type": "object",
    "title": "verification response schema",
    "description": "The verification response schema describing the response payload after verification.",
    "default": {},
    "examples": [
        {
            "code": 200,
            "verificationId": "b9525c1e-588d-4d9d-8325-8655c9fced29"
            "verified": true,
            "data": {
                "property1": "abc"
            },
            "message": ""
        }
    ],
    "required": [
        "code"
    ],
    "properties": {
        "code": {
            "$id": "#/properties/code",
            "type": "integer",
            "title": "The code schema",
            "description": "Represents the status code for the verification process. These codes are aligned
on the http status codes, hence 200 OK, 400 Bad Request, 500 Internal Server Error",
            "default": 200,
            "examples": [
                200, 400, 500
            ],
           "minimum": 100,
           "maximum": 511
        },
        "verificationId": {
            "$id": "#/properties/verificationId",
            "type": "strings",
            "title": "The verification id schema.",
            "description": "The verification id referenced with the initialized verification request.",
            "default": "",
            "examples": [
                "b9525c1e-588d-4d9d-8325-8655c9fced29"
            ]
        },
              "verified": {
          "$id": "#/properties/verified",
          "type": "boolean",
          "title": "The verified schema",
          "description": "Represents the verification process result. true means the holder has successfully
proofed, that he is in possession of the requested attributes.",
          "default": false,
          "examples": [
            true
          ]
        },
        "data": {
            "$id": "#/properties/data",
            "type": "object",
            "title": "The data schema",
            "description": "The data payload, only provided if the code is 200, may contain any use case
specific data.",
            "default": {},
            "examples": [
                {
                    "property1": "abc"
                }
            ],
            "required": [
            ],
            "properties": {

            },
            "additionalProperties": true
        },
        "message": {
            "$id": "#/properties/message",
            "type": "string",
            "title": "The message schema",
            "description": "Contains the error message in case of an response code 400 or 500.",
            "default": "",
            "examples": [
                ""
            ]
        }
    },
    "additionalProperties": true
}
```

ART006 Self-Attested attributes

Version	Date	Author	Comment	Status
0.1	07.09.2021	[redacted]	Initial version	DRAFT
0.2	05.10.2021	[redacted]	Added definition, presentation preview, and schema example	IN REVIEW
0.3	04 Jan 2022	[redacted]	Added self-attested attributes registry. Review & minor editorial changes.	IN REVIEW

Table of Contents

Abstract

The Aries RFC 0037 describes a self-attested attribute as an attribute that does not come from a credential, and hence any attribute specification without the `cred_def_id` key.

Design Decisions

What is the level of credentials assurance?

For further information, please refer to Classes of Verifiable Credentials

We conclude that discussed self-attested attributes shall be clustered under verifiable credentials as **Class 2**.

Presentation Preview

This is not a message but an inner object for other messages in this protocol. It is used to construct a preview of the data for the presentation. Its schema follows:

JSON-Object

```
{
    "@type": "https://didcomm.org/present-proof/1.0/presentation-preview",
    "attributes": [
        {
            "name": "<attribute_name>",
            "cred_def_id": "<cred_def_id>",
            "mime-type": "<type>",
            "value": "<value>",
            "referent": "<referent>"
        },
        // more attributes
    ],
    "predicates": [
        {
            "name": "<attribute_name>",
            "cred_def_id": "<cred_def_id>",
            "predicate": "<predicate>",
            "threshold": <threshold>
        },
        // more predicates
    ]
}
```

An attribute specification must specify a value, a `cred_def_id`, or both:

- if `value` is present and `cred_def_id` is absent, the preview proposes a **self-attested attribute**;
- if `value` and `cred_def_id` are both present, the preview proposes a verifiable claim to reveal in the presentation;
- if `value` is absent and `cred_def_id` is present, the preview proposes a verifiable claim not to reveal in the presentation.

Schema

The self-attested attributes could be clustered in schemas that normalized naming. Using a normalized naming, for example, JSON Web Token Claims RFC7519, the ecosystem can benefit by using the same standard for all use cases to which it applies.

Refer to the registry of selected claims that are used in the EESDI ecosystem: Self-attested claims registry

Below is an example of self-attested attributes used in the Access Management use case.

Guest Credential Self-Attested Attributes

```
"data": {
        "attr_names": [
          "license_plate_number",
                  "company_address",
          "picture"
        ],
        "name": "Guest-Self-Attested-Cred",
        "version": "1.1"
      }
```

ART007 Revoke Credential

Version	Date	Author	Comment	Status
0.1	25.08.2021	[redacted]	Initial version	DRAFT
0.2	30.08.2021	[redacted]	Reworked version, adding detailed sequence diagram	IN REVIEW
0.2	02.09.2021	[redacted]	Transferring ownership to [redacted]	IN REVIEW

Table of Contents

Revocation Process

Revoking a verifiable credential with the aca-py is quite simple, provided the credential issuance has been set up correctly (ART004 Issue SSI Credentials) with a revocation registry and a tails server (IG004 Tails Server Installation).

As described in the architectural decision (Architectural Decisions#ARC005Revocation), we assume that the controller of the issuer has stored a mapping of a holder UUID to the tuple of (revocation ID, revocation registry ID). Consequently, the revocation happens through a simple POST request to /revocation/revoke with the following body:

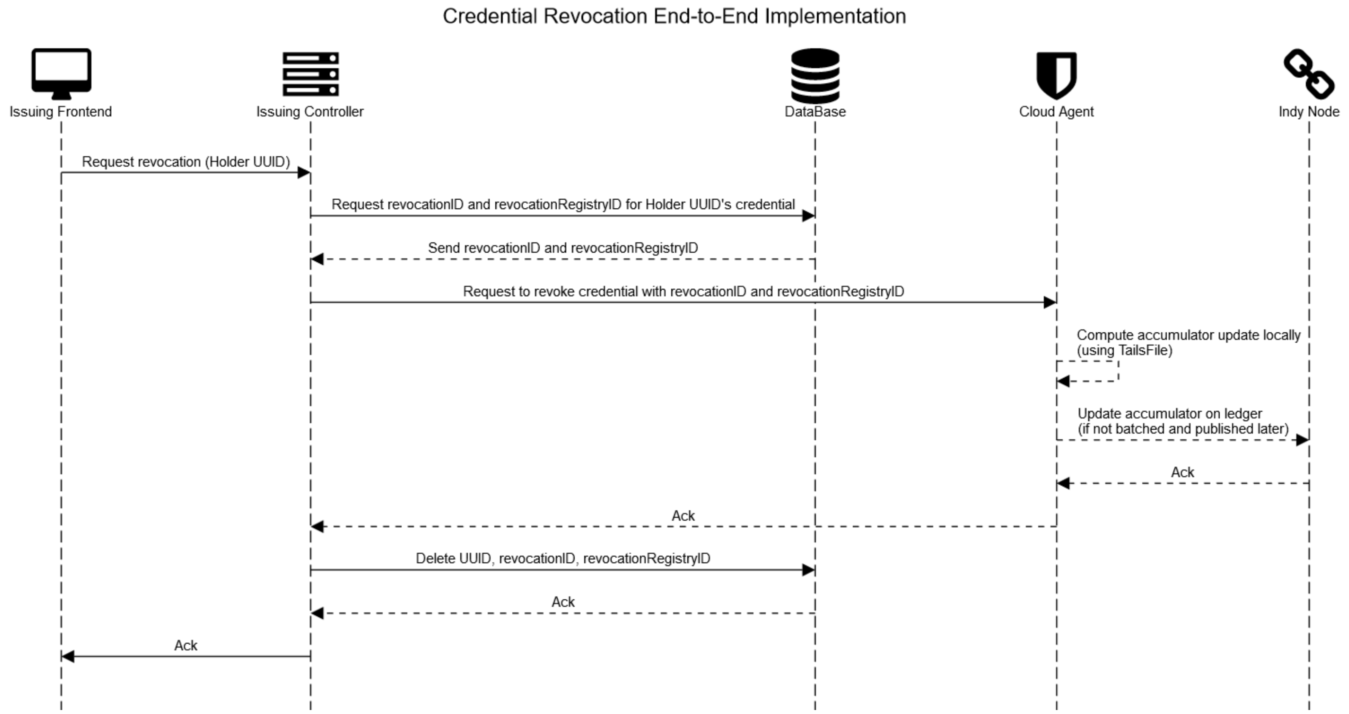
```
{
    "cred_rev_id": "<<CREDENTIAL REVOCATION ID>>"
    "rev_reg_id": "<<CREDENTIAL REVOCATION REGISTRY ID>>",
    "publish": true
}
```

By setting publish to true, the accumulator on the Indy blockchain is immediately updated; when setting publish to false, the new accumulator is only cached on the Cloud Agent and published on flushing it later on.

INFO

The revocation registry ID can also be drawn from (i) the issuer's institutional wallet or (ii) be taken from detailed information about the associated credential definition (GET /credential-definition/{cred_def_id}). Both cred_rev_id (=revocation_id) and rev_reg_id (=revocation registry id) can also be taken from the issue-credential webhooks that are delivered during the issuance process.

Detailed Sequence Diagram


Credential Revocation End-to-End Implementation
Issuing Frontend
Issuing Controller
DataBase
Cloud Agent
Indy Node
Request revocation (Holder UUID)
Request revocationID and revocationRegistryID for Holder UUID's credential
Send revocationID and revocationRegistryID
Request to revoke credential with revocationID and revocationRegistryID
Compute accumulator update locally
(using TailsFile)
Update accumulator on ledger
(if not batched and published later)
Ack
Ack
Delete UUID, revocationID, revocationRegistryID
Ack
Ack

Open Issues:

Currently, on every revocation, there is a new transaction written to the Indy ledger. At scale, when there are many revocations by many issuers, this may at some stage cause performance issues (for write performance metrics of Hyperledger Indy networks with different networks sizes, see the DLPS paper).

In this case, the above process should likely be modified by setting "publish" to false. Once per epoch, e.g., once a day, the request POST `/issue-credential/publish-revocations` should be called with an empty body to settle the pending revocation on the ledger. See also these instructions on revocation by the aca-py community. In other words, if `published=true` is used, the revocation records will be cached on the aca-py until flushed by calling `/issue-credential/publish-revocations`.

ART009 Verification Callback

Version	Date	Author	Comment	Status
0.1	08 Nov 2021	[redacted]	Initial version	DRAFT
0.2	16 Nov 2021	[redacted]	Removed ART005 dependency. Changed to connection-based and spoofing-resistant concept.	DRAFT
0.3	01 Dec 2021	[redacted]	Changed to CallbackDecorator concept	APPROVED

Table of Contents

Introduction

SSI verification is often part of a larger process flow. To make it as easy as possible for the user to carry out this overall process, as few manual interactions as possible should be required and process steps should mesh as intuitively together as possible. This specification describes a way to allow for calling a return address (callback), so that the process does not end after the user (Holder) has completed responding to the proof. Not only is it possible to return to an app (via deep link), but also to return to a web page (in the browser), which can, for example, continue the further process flow by using a session ID contained in the URL.

> ⚠ Please note that using http(s) callback URL does not necessarily open the browser in which the user originated from, but that the system decides which browser to open. This is usually the configured default browser. Therefore, the session should be based on a session ID contained in the callback URL, since a session cookie may not be available in the opened browser context.
>
> **Important:** This stands in conflict with requirement 3.1.1 "Verify the application never reveals session tokens in URL parameters." of the OWASP Application Security Verification Standard in version 4.0.3.

This ART introduces a new CallbackDecorator that has to be added to the Presentation Request (spec/present-proof/1.0/request-presentation). It is compatible with SSI verifications as defined in ART001 and ART005 for both connectionbased and connectless (with service decorator based on RFC 0056) proof requests.

Process aborts that may happen during the process (CBF-XX) are reflected in chapter **Issue & Error handling**.

CallbackDecorator

The CallbackDecorator adds meta information to the Presentation Request JSON object similar to RFC 0056.

It has to be added with the key *~callback*.

The callback data itself has to be defined in the following format:

JSON Format

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "title": "Callback URLs",
  "examples": [
    {
                "success_url": "https://example.org/?cbid=7512-4124214-2414124-ABC&status=OK",
            "failed_url": "https://example.org/?cbid=7512-4124214-2414124-ABC&status=FAILED"
        },
    {
                "success_url": "https://example.org/?cbid=7512-4124214-2414124-ABC&status=OK"
        },
    {
                "success_url": "app-abc://example.org/?verification=true"
        }
  ],
  "properties": {
          "success_url": {
            "type": "string",
              "format": "uri"
    },
    "failed_url": {
            "type": "string",
        "format": "uri"
    }
  },
  "required": [
    "success_url"
  ]
}
```

This data has to be base64 encoded and signed as defined in RFC0017 - Signing Attachments which allows adding signatures based on the IETF RFC 7515 standard for JSON Web Signatures (JWS).

The raw data (payload) is included as *$.['~callback']['data'].['base64']* (JSONPath).

Final Request Presentation Example

```
{
    "@type": "https://didcomm.org/present-proof/1.0/request-presentation",
    "@id": "<uuid-request>",
    "comment": "some comment",
    "request_presentations~attach": [
        {
            "@id": "libindy-request-presentation-0",
            "mime-type": "application/json",
            "data":  {
                "base64": "<bytes for base64>"
            }
        }
    ],
    "~service": {
        "endpointName": "<endpoint-url>"
    },
    "~thread": {
                ...
    },
    "~callback": {
        "mime-type": "application/json",
        "data": {
            "base64": "eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ... (bytes omitted to shorten)",
            "jws": {
                // payload: ...,  <-- omitted: refer to base64 content when validating
                "header": {
                    "kid": "did:key:z6MkmjY8GnV5i9YTDtPETC2uUAW6ejw3nk5mXF5yci5ab7th"
                },
                "protected": "eyJhbGciOiJFZERTQSIsImlhdCI6MTU4Mzg4... (bytes omitted)",
                "signature": "3dZWsuru7QAVFUCtTd0s7uc1peYEijx4eyt5... (bytes omitted)"
            }
        }
    }
}
```

Validate the CallbackDecorator

In case of an existing CallbackDecorator the Edge Agent (i. e. Wallet App) has to validate the signature of the given data.

Callback URLs **must not** be used when the signature validation failed.

Callback URLs **must not** be used when the signer differs from the signed service decorator, especially the service endpoint url.

- **Signature could be cached on Verifier side (no need to recompute for every verification)**
- Decision needed: Sign with TLS private keys **or with DID private key?**
- **Signature of callback decorator is primarily motivated due to connection less proof requests without integrity checks.**
    - **What is the future of connection less proof requests? any decorator would required special care for connection less scenarios (MITM)**

Executing the callback

After sending the presentation to the Cloud Agent (or in case of an error, see chapter **Issue & Error handling**) the ID Wallet app has to open the callback URL (*p_callback_success* resp. *p_callback_failed*) if one was provided.

> Opening a callback URL requires a user prompt in order not to compromise the self-sovereignty of the user.
>
> This allows the user to decide for himself whether he wants to follow the redirect.

Issue & Error handling

Besides a failed verification (invalid proof (i.e. caused be revoked credentials) or inacceptable data) the process could also interrupted while reviewing /sending the presentiation request in the ID Wallet:

CBF-01 - User declines providing the requested attributes

If the user decides that he is not willing to provide the requested attributes from one or more of his credentials he can cancel the presentation review in the ID Wallet app.

Clicking on cancel will cause the app to call the *p_callback_failed* URL, if it was provided (see chapter Executing the callback).

CBF-02 - User does not have all credentials to comply with the restrictions of the proof request

If the user does not have all required credentials he will be unable to answer the presentation request in the ID Wallet app. If this is being identified the app will call the *p_callback_failed* URL, if it was provided (see chapter **Executing the callback**).

> ⚠ The *p_callback_failed* URL should instruct the user how he can get the necessary credentials to answer the proof request.

CBF-03 - ID Wallet is technically unable to answer the presentation request

In case of no internet connection, an unavailable Cloud Agent or other technical reasons that may lead to an error while sending the proof response to the Cloud Agent the *p_callback_failed* URL will be called, if it was provided (see chapter **Executing the callback**).

Open Topics

- Migrate to https://github.com/hyperledger/aries-rfcs/tree/main/concepts/0700-oob-through-redirect when available & secured against spoofing
- How should OWASP 3.1.1 conflict be handled? OWASP Application Security Verification Standard

☐ Clearify which key has to be used for the signature

ART010 Issuing verified e-mail addresses & phone numbers

Version	Date	Author	Comment	Status
0.1	24 Sep 2021	[redacted]	Initial version	DRAFT
0.2	28 Sep 2021	[redacted]	Initial conceptual flow	DRAFT
0.3	30 Sep 2021	[redacted]	Verified credentials classes, updated conceptual flow	DRAFT
0.4	05 Oct 2021	[redacted]	Design Decisions, proofreading, set to "in review"	IN REVIEW
0.5	01 Nov 2021	[redacted]	Incorporate Feedback from BKa: Harden phone number attestation, distinguish between class 3 + 4 attestation,	IN REVIEW
0.6	03 Jan 2022	[redacted]	Added Claims Format section, based on JWT Claims	IN REVIEW
1.0	24 Jan 2022	[redacted]	Aligning with ARC009, incorporating decision to launch with A1 and B1/B2 for future rollouts	IN REVIEW

Table of Contents

Abstract

Technical attributes like email addresses, phone numbers, and tax numbers, that are not yet issued as verifiable credentials, need to be moved to the EESDI network. Claimed attributes will be attested leveraging on commonly used challenge-response processes, such as sending a one-time token (OTP) to the claimed email address/phone number and confirmed by the holder, proving access to the inbox/phone, or by querying legacy backend system of records. In its simplest form, an "offline" regular expression (regex) check might be sufficient.

This template focuses on email addresses and phone numbers only, however, the design decision and conceptual flows are applicable to a variety of claims.

Design Decisions

What attestation strategies are being supported

A1

B1

B2

What is the level of credentials assurance?

Please refer to Classes of Verifiable Credentials.

Attestation Strategy	Level of Attestation
A1	Class 1
B1	Class 2 / 3
B2	Class 2 / 3

What changes are required from ID Wallet?

Attestation Strategy	Impact on ID Wallet
A1	1. Input form for a. email address b. phone number c. tax number 2. Verifying input against regex rules (refer to section Regular expressions) 3. Storing of above's attributes in a persistent on-device datastore
B1	1. Initiation of the challenge-response process with issuing controller 2. New credential issuing flow with an input form for a. email address b. phone number c. tax number 3. Attributes need indicators to reflect their status (not attested / pending attestation / successfully attested). 4. Verification form to input OTP from email or SMS
B2	No changes required

What are the requirements for 3P attestation providers?

GDPR / EU-based

Given the strict regulatory environment, paired with the processing of potential PII (and even SPII), we recommend partnering with an EU-based vendor, with services deployments within the EU and following GDPR regulations. This ensures that *data in transit* between issuing controllers and the 3P attestation providers remain in the EU.

Widely used and reputable

We strongly encourage thorough technical due diligence in the selection of the 3P attestation providers, for (non-exhaustive):

- Ecosystem (APIs, SDKs, Developer community)
- Offered SLAs
- HA / DR strategies; Multi-Zone availability
- Enterprise support desk

Range of attestation service offerings

To reduce the implementation effort for integrating and maintaining the 3P attestation services, we recommend providers who offer attestation for multiple credentials and channels, for example, email, phone number (via SMS / Voice / Messenger / Landline), residential addresses.

Shortlist (non-exhaustive, non-prioritized):

- Twilio
- Byteplant

Commercials

Enterprise-grade, predictable, tiered service plans, potentially leveraging on existing partnerships between ecosystem members and 3P attestation providers.

Which entity is issuing credentials and operating/maintaining the services?

Ecosystem members

Could self-attested attributes be issued in the same way?

Given that self-attested attributes are classified as **Class 2** and therefore not adhering to a 3P attestation process, the issuing and verification process described in ART006 Self-Attested attributes is not modified.

What are potential extensions and research topics?

- Regular re-verification (e.g. sending a verification to the email address or phone every *N* months):
  This could be achieved by adding a revocation time to the issued credentials so that the email address or phone number has to be revalidated regularly. This not only increases trust, quality, and security but also ensures that users keep their details up to date.

- Verified "Selfies": Issuing verifiable credentials for an attested portrait shot of the user.

- Schema definition: Can a generic schema serve as the foundation for a variety of attributes? Are there proposals or standards (e.g. in ToIP) for naming conventions, schema structures?

Claims Format

- working version -

The self-attested claims for email and phone numbers must adhere to the following naming convention:

Claim Name	Type	Claim Description	Example	Reference
`email`	string	Holders' self-attested email address	jane.doe@my-digi-id.com	RFC5322
`phone_number`	string	Holder's self-attested phone number MUST include the country code MUST NOT include spaces, dashes, parentheses MUST only include characters: 0-9, +	+4930867716259	E.164 RFC3966
`taxID`	string			

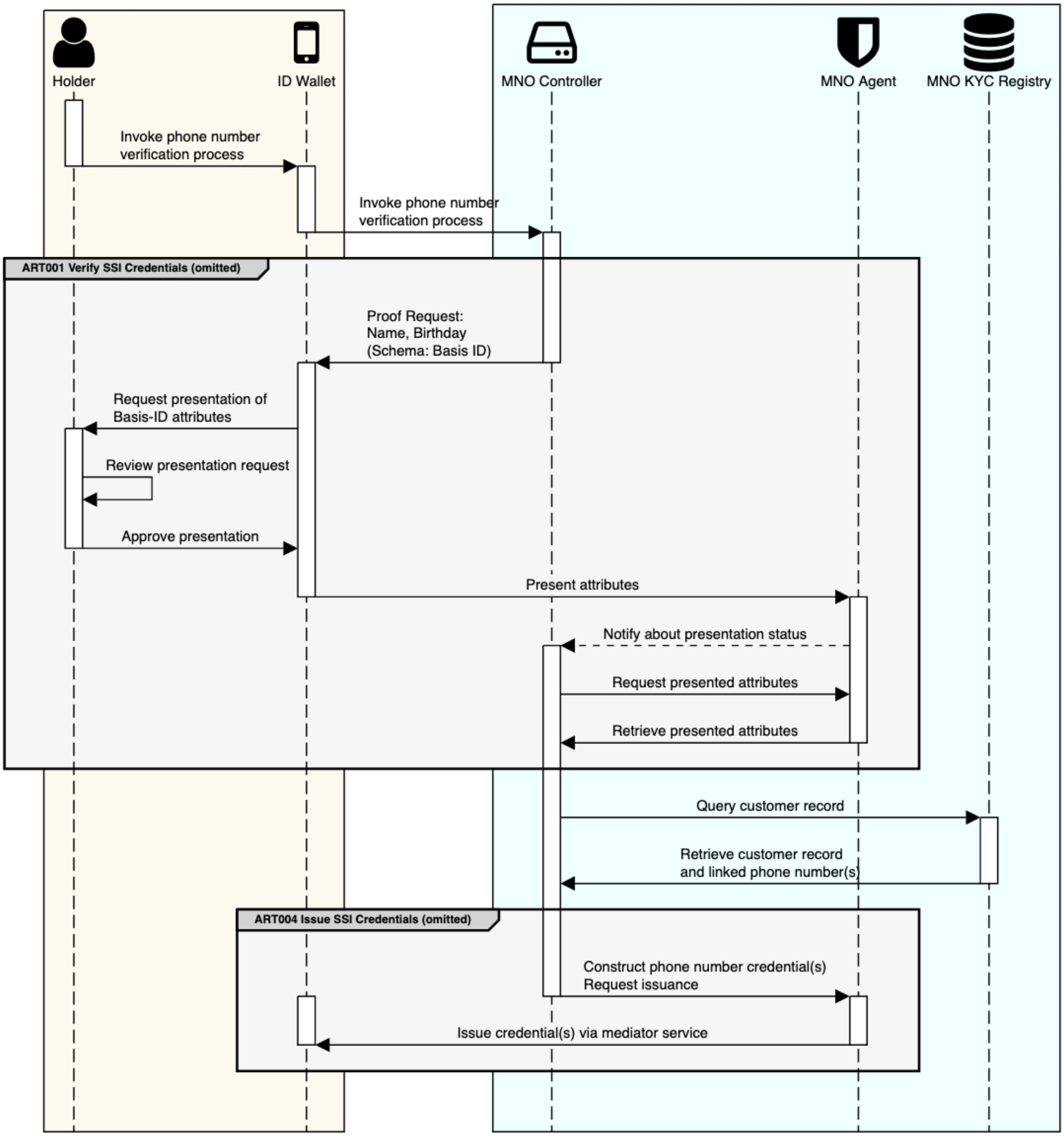
based on JSON Web Token Claims RFC7519 and OpenID Connect Core 1.0 Section 5.1

Regular Expressions

Claim Name	Type	Regular expression
`email`	string	
`phone_number`	string	
`taxID`	string	

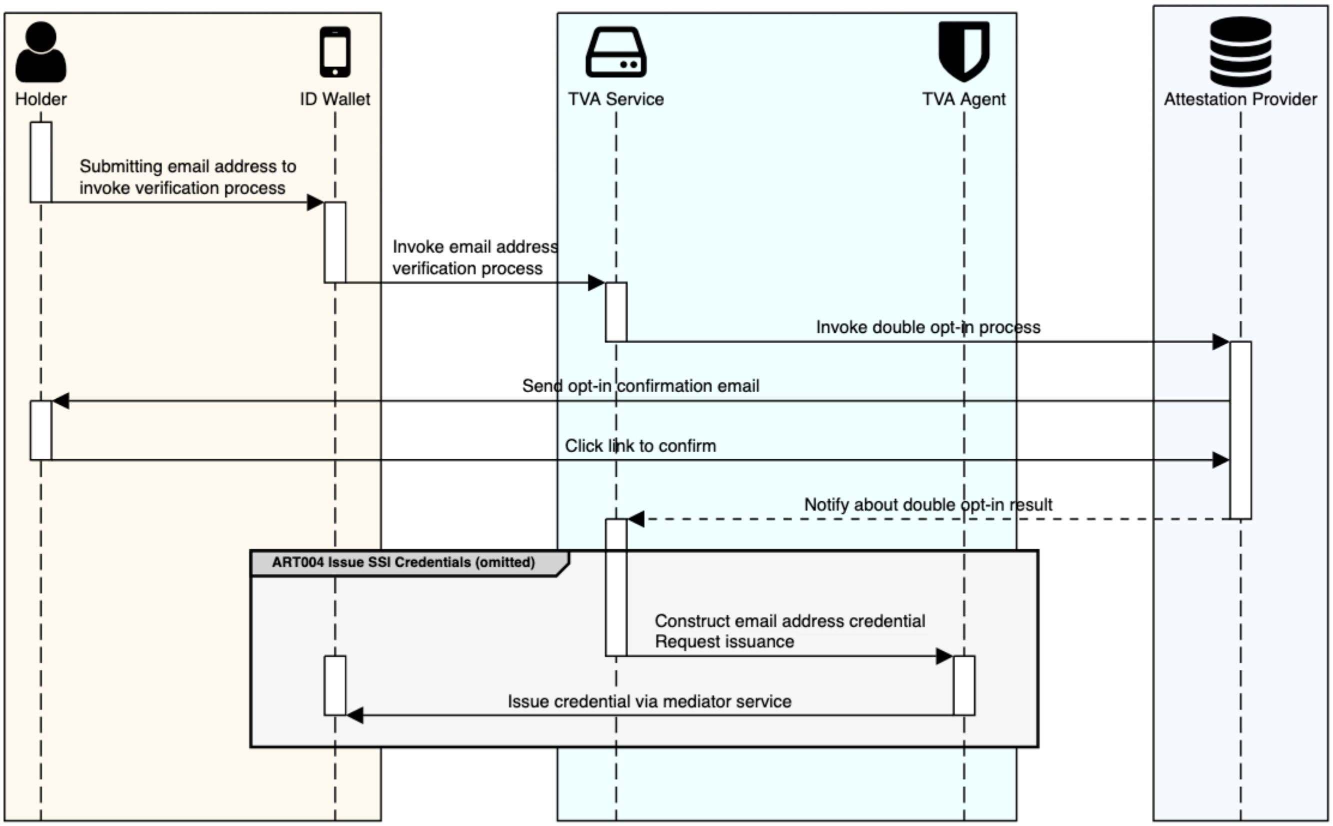
Conceptual Flows

Class 4 credential issuing: phone number


Holder
ID Wallet
MNO Controller
MNO Agent
MNO KYC Registry
Invoke phone number verification process
Invoke phone number verification process
ART001 Verify SSI Credentials (omitted)
Proof Request: Name, Birthday (Schema: Basis ID)
Request presentation of Basis-ID attributes
Review presentation request
Approve presentation
Present attributes
Notify about presentation status
Request presented attributes
Retrieve presented attributes
Query customer record
Retrieve customer record and linked phone number(s)
ART004 Issue SSI Credentials (omitted)
Construct phone number credential(s) Request issuance
Issue credential(s) via mediator service

Source: GitLab

Class 3 credential issuing: email address


Holder
ID Wallet
TVA Service
TVA Agent
Attestation Provider
Submitting email address to invoke verification process
Invoke email address verification process
Invoke double opt-in process
Send opt-in confirmation email
Click link to confirm
Notify about double opt-in result
ART004 Issue SSI Credentials (omitted)
Construct email address credential
Request issuance
Issue credential via mediator service

Source: GitLab

Classes of Verifiable Credentials

Version	Date	Author	Comment
0.1	22 Nov 2021	[redacted]	Initial version

Table of Contents

Motivation

Not all Verifiable Credentials (VC) are equal: Depending on the level of attestation and the reputation of the issuer, VCs can have different levels of trust.

The **Trust over IP (ToIP)** foundation suggests "*various classes of verifiable credentials*" (Source), which in summary are:

- Class 1 - Untrusted Credentials: Little or no confidence in the asserted identity's validity
- Class 2 - Minimum Internet Grade Credentials: Some confidence in the asserted identity's validity
- Class 3 - Asset Value Grade Credentials: High confidence in the asserted identity's validity
- Class 4 - High Assurance Grade Credentials: Very high confidence in the asserted identity's validity

Classifications in the EESDI ecosystem

Classification	Examples	Trust Assurance Scheme	Mapped Level to other Standards
Class 1	• Self-Attested Attributes	No trust assurance scheme	• NIST 800-63-3: **IAL1, AAL1, FAL1** • PCTF: **Level 1** • eIDAS: **Low** • Vectors of Trust: **P0, C0 Ma, Aa**
Class 2	• Proof of Employment • TVA (A1: Plausibility checks only)	Self-Asserted by ecosystem roles	• NIST 800-63-3: **IAL2, AAL1, FAL1** • PCTF: **Level 2** • eIDAS: **Between low and substantial** • Vectors of Trust: **P2, Ce, Mb, Ab?**
Class 3	• TVA (A2/B1: challenge-response)	Asserted by ecosystem roles and attested by an independent third-party	• NIST 800-63-3: **IAL2, AAL2, FAL2** • PCTF: **Level 3** • eIDAS: **Substantial** • Vectors of Trust: **P2, Cf, Mc, Ac?**
Class 4	• Basis ID • Digital Driver's license • TVA (B2: regulated registry)	Asserted by ecosystem roles and attested by an independent third-party and certified by a recognized certification body	• NIST 800-63-3: **IAL3, AAL3, FAL3** • PCTF: **Level 4** • eIDAS: **High** • Vectors of Trust: **P3, Cf, Mc, Ad?**

Data Model

Version	Date	Author	Comment
0.1	19.10.2021	[redacted]	Initial version for BasisID and Digital Driving License
0.2	03.11.2021	[redacted]	Approved Basis-ID data model
0.3	26.11.2021	[redacted]	Version 1.0 for Digital Driving License ready for review

Table of Contents

#	Title	Confluence Page	Short Description	Owner	Status
DM001	Basis ID	DM001 - Basis ID	Credential definition and schema for the BasisID	[redacted]	APPROVED
DM002	Digital Driving License	DM002 - Digital Driving License	Credential definition and schema for the Digital Driving License	[redacted]	IN REVIEW

Status Flow

TODO DRAFT IN REVIEW APPROVED

DM001 - Basis ID

Version	Date	Author	Comment	Status
0.1	19.10.2021	[redacted]	Initial version	DRAFT
0.2	20.10.2021	[redacted]	Ready for review	IN REVIEW
0.3	02.11.2021	[redacted]	Fixed Test Basis id Schema and Cred Def ID	IN REVIEW
0.4	03.11.2021	[redacted]	Approved	APPROVED

Schema Definitions

Credential Name	Ledger	Schema Version	Schema ID	Claim Definition ID	Last updated on	From DID
Basis-ID	EESDI-Pilot-Ledger	1.1	Vq2C7Wfc44Q1cSroPuXaw2:2:Basis-ID:1.1 https://eesdi.esatus.com/tx/EESDI-Pilot-Ledger/domain/1128	Vq2C7Wfc44Q1cSroPuXaw2:3:CL:1128:Basis-ID https://eesdi.esatus.com/tx/EESDI-Pilot-Ledger/domain/1164	2021-07-29T14:09:27.000Z	Vq2C7Wfc44Q1cSroPuXaw2
Basis-ID	EESDI-Test-Ledger	1.1	MGfd8JjWRoiXMm2YGL4SGj:2:Basis-ID:1.1 https://eesdi-test.esatus.com/tx/EESDI-Test-Ledger/domain/43	MGfd8JjWRoiXMm2YGL4SGj:3:CL:43:Basis-ID Testnetzwerk https://eesdi-test.esatus.com/tx/EESDI-Test-Ledger/domain/44	2021-07-26T09:53:53.000Z	MGfd8JjWRoiXMm2YGL4SGj

Basis-ID Attributes - Schema Version 1.1

Attribute	Description	Type	Format	Corresponding eID field (BSI-TR-03127_1-30.pdf)	Mandatory	Nullable
firstName	Specifies the first name of the person	String	example: "Ingrid"	DG4	Y	n
addressStreet	Specifies the street for the address	String	The place of residence is usually stored as a structured address (structure dPlace according [TR03110] consisting of the country code [ISO3166], street with house number, place of residence and postal code. If the holder of the ID card has his/her residence outside Germany, the text “keine Hauptwohnung in Deutschland” [No habitual place of residence in Germany] (noPlaceInfo according to [[TR03110]) is stored instead. Example: addressStreet = "Waßmannsdorfer Chaussee 90" addressCity = "Berlin" addressZipCode = "13439" addressCountry = "DE"	DG17	y	n
addressCity	Specifies the city for the address	String			y	n
addressZipCode	Specifies the ZIP code for the address	String			y	n
addressCountry	Specifies the country for the address	String			y	n
dateOfExpiry	The expiry date for the (physical) ID Card	Date	int (YYYYMMDD)	DG3	y	n
documentType	Specifies the type of the document	String	Predefined list of codes "ID": Identity Card "AR", "AS", "AF": Residence Permit "UB": eID card for Union citizens	DG1	y	n

hardwareDid	Hardware did used to enforce device binding.	String	`did:key:<>`	-	y	n
academicTitle	The academic title of the person	String	example "*Dr.*"	DG7	n	y
placeOfBirth	The place of the person's birth	String	Place of birth as unformatted text. example: "Berlin"	DG9	y	n
familyName	Actual surname of the person, i.e. after getting married	String	example: "Kayser"	DG5	y	n
nationality	Nationality of the holder	String	In ID cards issued before November 1, 2019, the data group is empty. German = "D"	DG10	n	y
birthName	Surname of the person given upon their birth	String	example: "Schmidt" Cards issued until Q2/2012 (PA: Serial Number 112) and Q4/2014 (eAT: Serial Number 138) contain no or empty birthName.	DG13	n	y
dateOfBirth	Date of birth of the person	Date	int (YYYYMMDD) The complete date of birth is not known for all holders of ID documents. It contains the date of birth in YYYYMMDD format; unknown parts are filled with blanks. With regard to the special age verification function (section 4.4.3.4), the date which is the latest possible date of birth according to the known partial dates is stored internally (for instance, 31 December of the year if only the year of birth is known). This ensures that even with an incomplete date of birth age verification will only supply a positive result if the holder is definitely of the desired age	DG8	y	n

Additional documentation:

https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03127/tr-03127.html

BSI-TR-03127_1-30.pdf

BSI-TR-03127_en.pdf

DM002 - Digital Driving License

Version	Date	Author	Comment	Status
0.1	19.10.2021	[redacted]	Initial version	DRAFT
1.0	26.11.2021	[redacted]	Version One	IN REVIEW

Schema Definitions

Schema Name	Ledger	Schema Version	Schema ID	Credential Definition ID	Last Updated On	From DID
Digitaler Führerschein	EESDI-Pilot-Ledger	0.3	KqtBRiQSyWqnzaxN3u2d7G:2:Digitaler Fuehrerschein:0.3 https://eesdi.esatus.com/tx/EESDI-Pilot-Ledger/domain/1260	KqtBRiQSyWqnzaxN3u2d7G:3:CL:1260: Digitaler Führerschein https://eesdi.esatus.com/tx/EESDI-Pilot-Ledger/domain/1269	2021-08-25T09:40:11.000Z	KqtBRiQSyWqnzaxN3u2d7G
Digitaler Führerschein	EESDI-Test-Ledger	0.3	9hsRe5jdzAbbyLAStV6sPc:2:Digitaler Führerschein Test:0.3 https://eesdi-test.esatus.com/tx/EESDI-Test-Ledger/domain/970	9hsRe5jdzAbbyLAStV6sPc:3:CL:970: Digitaler Führerschein Test https://eesdi-test.esatus.com/tx/EESDI-Test-Ledger/domain/972	2021-10-06T14:08:19.000Z	9hsRe5jdzAbbyLAStV6sPc

Digital Driving License Attributes - Schema Version 0.3

Attribute	Description	Type	Format	Mandatory	Nullable	Corresponding Field as in Anlage 8 FeV
firstName	Specifies the first name of the person	String	example: "Ingrid"	y	n	1. Name, Doktorgrad
familyName	Actual surname of the person, i.e. after getting married	String	example: "Kayser"	y	n	2. Vorname
academicTitle	The academic title of the person	String	example "*Dr.*"	n	y	1. Name, Doktorgrad
dateOfBirth	Date of birth of the person	Date	int (YYYYMMDD) The complete date of birth is not known for all holders of ID documents. It contains the date of birth in YYYYMMDD format; unknown parts are filled with blanks. With regard to the special age verification function (section 4.4.3.4), the date which is the latest possible date of birth according to the known partial dates is stored internally (for instance, 31 December of the year if only the year of birth is known). This ensures that even with an incomplete date of birth age verification will only supply a positive result if the holder is definitely of the desired age	y	y	3. Geburtsdatum und -ort
placeOfBirth	The place of the person's birth	String	Place of birth as unformatted text. Example: "Berlin"	y	n	3. Geburtsdatum und -ort
issuingEntity	The authority which has issued the driving license	String	Example: "Landratsamt"	y	n	4c. Name der Ausstellungsbehörde
dateOfIssuance	The date in which the driving license was issued	Date	int (YYYYMMDD) A field for entering the date of issue of the driver's license for one or more classes The date of issue of one or more driving license classes can also be entered on this field by specifying the number 10. In these cases, reference is made to this from the specific driving license class "*dateOfIssuance" field by means of "*)".	y	n	4a. Ausstellungsdatum gemäß § 24a

generalRestrictions	Restrictions which apply that apply to all issued driving license classes	String	Restrictions, requirements and additional information are to be entered in the form of key numbers (see Anlage 9 FeV). If they refer to individual driving license classes, they are to be entered in the specific restriction field in the line of the relevant driving license class. Those that apply to all issued driving license classes are to be noted in this field. The harmonized key numbers of the European Union consist of two digits (main key numbers). Sub-codes consist of a main key number (first part) and two digits and / or letters (second part). The first and second parts are separated by a point. The second part can contain further digits / letters with certain encodings. National codes consist of three digits. They are only valid in Germany. The key figures to be entered must fully record the restrictions, requirements and additional information. The use of sub-keys is mandatory for the main key numbers 44, 50, 51, 70, 71 and 79. Accumulations are to be separated by commas and alternatives by slashes. Harmonized key figures are to be listed before the national ones. When a driver's license is issued, the holder must be informed about the meaning of the key numbers entered.	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9. Beschränkungen und Zusatzangaben (einschließlich Auflagen), die nur für eine Fahrerlaubnisklasse gelten, werden in der Zeile der jeweiligen Klasse vermerkt. Solche, die für alle Fahrerlaubnisklassen gelten, werden in der letzten Zeile der Spalte ausgewiesen.
id	Number of the driving license	String	Made up of the authority code of the driving license authority, a driving license number to be continuously assigned by the latter as well as a check digit and the number of the copy of the document	y	n	5. Nummer des Führerscheins, die sich aus dem Behördenschlüssel der Fahrerlaubnisbehörde, einer von dieser fortlaufend zu vergebenden Fahrerlaubnisnummer sowie einer Prüfziffer und der Nummer der Ausfertigung des Dokuments zusammensetzt.
hardwareDid	Hardware did used to enforce device binding.	String	`did:key:<>`	y	n	-
licenseCategoryAM_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryAM_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryA1_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryA1_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryA2_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.

licenseCategoryA2_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryA_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryA_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryB1_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryB1_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryBE_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryBE_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryB_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryB_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryC1_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.

licenseCategoryC1_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryC1_DateOfExpiry	The validity date of limited driving license classes	Date	int (YYYYMMDD)	y	y	11. Das Gültigkeitsdatum befristet erteilter Fahrerlaubnisklassen.
licenseCategoryC1E_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryC1E_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryC1E_DateOfExpiry	The validity date of limited driving license classes	Date	int (YYYYMMDD)	y	y	11. Das Gültigkeitsdatum befristet erteilter Fahrerlaubnisklassen.
licenseCategoryCE_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryCE_Restrictions	Restrictions for this specific Driving License Class	String	Restrictions and additional information (including requirements) on the issued driving license classes in encrypted form in accordance with Anlage 9 FeV. Restrictions and additional information (including requirements) that only apply to one driving license class are noted in the line for the respective class. Those that apply to all driving license classes are shown in the generalRestrictions form.	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryCE_DateOfExpiry	The validity date of limited driving license classes	Date	int (YYYYMMDD)	y	y	11. Das Gültigkeitsdatum befristet erteilter Fahrerlaubnisklassen.
licenseCategoryC_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryC_DateOfExpiry	The validity date of limited driving license classes	Date	int (YYYYMMDD)	y	y	11. Das Gültigkeitsdatum befristet erteilter Fahrerlaubnisklassen.
licenseCategoryC_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9

licenseCategoryDE_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryDE_DateOfExpiry	The validity date of limited driving license classes	Date	int (YYYYMMDD)	y	y	11. Das Gültigkeitsdatum befristet erteilter Fahrerlaubnisklassen.
licenseCategoryDE_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryD1E_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryD1E_DateOfExpiry	The validity date of limited driving license classes	Date	int (YYYYMMDD)	y	y	11. Das Gültigkeitsdatum befristet erteilter Fahrerlaubnisklassen.
licenseCategoryD1E_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryD1_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryD1_DateOfExpiry	The validity date of limited driving license classes	Date	int (YYYYMMDD)	y	y	11. Das Gültigkeitsdatum befristet erteilter Fahrerlaubnisklassen.
licenseCategoryD1_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryD_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryD_DateOfExpiry	The validity date of limited driving license classes	Date	int (YYYYMMDD)	y	y	11. Das Gültigkeitsdatum befristet erteilter Fahrerlaubnisklassen.

licenseCategoryD_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryL_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryL_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryM_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryM_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9
licenseCategoryT_DateOfIssuance	The date of issue of the driving license for the respective driving license class	Date	int (YYYYMMDD) See "dateOfIssuance"	y	y	10. Das Datum der Erteilung der Fahrerlaubnis der jeweiligen Fahrerlaubnisklasse.
licenseCategoryT_Restrictions	Restrictions for this specific Driving License Class	String	Same as in "generalRestrictions"	y	y	12. Beschränkungen und Zusatzangaben (einschließlich Auflagen) zu den erteilten Fahrerlaubnisklassen in verschlüsselter Form gemäß Anlage 9

References:

Anlage 8 FeV , Anlage 8 FeV - Einzelnorm.pdf

Anlage 9 FeV , Anlage 9 FeV - Einzelnorm.pdf

Non-functional Requirements

Version	Date	Author	Comment
0.1	09.08.2021	[redacted]	Initial version

Table of Contents

Introduction

This chapter identifies the document and the (sub) system to which it relates, describes the contents of the document, and states its purpose.

Purpose

The purpose of the non-functional requirements document is:

- To specify required properties of the system.
- To define constraints on the system.
- To enable early system sizing and estimates of cost.
- To assess the viability of the proposed system.
- To give a basis for designing the operational models.
- To provide input to the component design

Description

The non-functional requirements specify the service levels the system must satisfy, the required non run-time properties of the system, and the constraints to which the system must conform. Non-functional requirements may apply to the system as a whole, to parts of the system, or to particular use cases.

Service Level Requirements (SLRs)

SLR are run-time properties which the system as a whole, or parts of the system, must satisfy. SLR includes: Capacity and Performance, Availability, Security and System Management.

Capacity and Performance (Volumetrics)

This chapter describes the capacity and performance levels the system must satisfy.

ID	Title
BKAVP-CAP-001	

Availability

This chapter describes the general availability requirements the system must satisfy.

ID	Title
BKAVP-AVA-001	Services must be able to gracefully handle the unavailability of other services.

Security

This chapter specifies the security requirements of the system defining its needed ability to safeguard data against loss or exposure, and to resist disruption by outside partners.

ID	Title
BKAVP-SEC-001	Services must be GDPR compliant

Scalability

This paragraph specifies the scalability requirements of the system.

ID	Title
BKAVP-SCA-001	Services must scale horizontally.

System Management

This chapter specifies the system management requirements of the system and concerns the maintenance and administration of the software and hardware systems.

ID	Title
BKAVP-SMG-001	It must be economically feasible to install and update the solution components.

Non-run-time Requirements

This chapter specifies the non-run-time property requirements of the system.

Portability

This paragraph specifies the portability requirements of the system.

ID	Title
BKAVP-POR-001	The solution must be based on open standards supported by multiple programming languages.

Maintainability

This paragraph specifies the maintainability requirements of the system.

ID	Title
BKAVP-MAI-001	It must be economically feasible to enhance the solution.
BKAVP-MAI-002	Source code must be version controlled.

System Constraints

This chapter specifies the business and technical constraints and standards to which the system must conform

Business Constraints

This paragraph specifies the business constraints that the system must satisfy.

ID	Title
BKAVP-BCT-001	

Technical Standards

This paragraph specifies the technical standards to which the system must conform.

ID	Title
BKAVP-TSD-001	Data must be stored and processed in UTF-8.
BKAVP-TSD-002	Date and time must be stored in UTC.

Technical Constraints

This paragraph specifies the technical constraints that the system must satisfy.

ID	Title
BKAVP-TCT-001	

Participation Patterns

Version	Date	Author	Comment
0.1	09.08.2021	[redacted]	Initial version

Table of Contents

Introduction

This chapter describes the different patterns for participating on the EESDI solution platform. The following table describes an overview of the different options for participants to interact with the EESDI network.

#	Option	Short Description	Indy Node	Cloud Agent	Verification Controller	Issuing Controller	Tails Server	Client	Implications
001	Node Provider	The participant operates an indy node within the network as steward.	x	x	x	x	x	x	+ Improves network integrity - High entry barrier, as SLA contracts needs to be signed and deep technical understanding of hyperledger indy is required.
002	Issuer Node Tenant	The participant does not run an indy node, but leverages the existing nodes in the EESDI network for issuing. The participant's agent needs to get author permissions to write transactions.		x		x	x*	x	+ Simple reuse of an existing network. - Participant still requires an official onboarding on the EESDI network as author. - SSI know how required in order to correctly implement the issuing controller according to the architectural templates. - Evolving the EESDI solution platform gets harder, as third party vendors may be dependent on existing solution concepts. ** Tails Server only needed if revocation is used.*

003	Verifier Node Tenant	The participant does not run an indy node, but leverages the existing nodes in the EESDI network for verification.		x	x			x	+ Simple reuse of an existing network + Ready to go solution without onboarding on the EESDI network required. - SSI know how required in order to correctly implement the verification controller according to the architectural templates. - Evolving the EESDI solution platform gets harder, as third party vendors may be dependent on existing solution concepts.
004	Cloud Agent Tenant	The participant does not run an indy node nor a cloud agent, but leverages existing components centrally. Cloud agent is configured for tenants.			x	x		x	+ In case of issuing no additional onboarding to the EESDI network required. - Saas Provider needs to be trusted, as all data is flowing over the provider - SSI know how required in order to correctly implement the verification controller according to the architectural templates.
005	SaaS	The participant uses a centrally hosted API for verifying and/or issuing credentials.						x	+ Easy adopting through state of the art REST APIs. + No SSI know how required. - SaaS Provider needs to be trusted, as all data is flowing over the provider.

Physical Operational Model

Unknown macro: 'gliffy'

Verifier Node Tenant

Version	Date	Author	Comment
0.1	25.08.2021	[redacted]	Initial version

Table of Contents

Introduction

The participant does not run an indy node, but leverages the existing nodes in the EESDI network for verification. In this participation pattern a partner is responsible for running the following solution components (see also Architecture Overview for more details)

- *Client:* The integration components to the verification controller. This may be a dedicated service, a plugin, a front end or directly the Legacy System.
- *Verification Controller*: The component which is adhering to the Architectural Templates. This component may very well be a simple module of the client component instead of a dedicated service.
- *Cloud Agent (ACA-Py)*: The agent to communicate with the EESDI Network.

However there are different scenarios how the implementation is performed. The following table provides an overview of the different models.

#	Title	Scenario	Support	Examples	Implications
1	**Customized**	The participant develops the client and the verification controller components. No implementation (besides reference implementation is provided by the Bundeskanzleramt)	- Installation & Configuration Guidelines for running an ACA-Py instance. - Architectural Templates describing the solution concept and approach for the verification controller. - ART001 Verify SSI Credentials - ART002 Multiple Proof Requests (without Connection) - ART003 Device Binding Verification - ART005 Verification Integration Pattern - Reference Implementation for the verification controller https://github.com/My-DIGI-ID/ddl-verification-controller.	- BMW, VW and BOSCH for the Mobile Drivers License.	+ Less effort in bug fixing or maintenance + Only network support required - Less flexibility in progressing standards used by the EESDI network.
2	**Adopted**	The participant develops the integration components (client) and uses the reference implementation provided by the Bundeskanzleramt.	- Installation & Configuration Guidelines for running an ACA-Py instance. - Reference Implementation for the verification controller. (not yet open source) - API Documentation of the verification controller (needed for integration with the client) - Installation & Configuration Guidelines for running the verification controller (+ any additional use case specific components, e.g. keycloak or mongodb).	- BAMF, BWI for the Access Management use case	+ More flexibility in progressing standards used by the EESDI network. + Simpler implementation effort on the partner side. (no specific SSI know how required) - More effort in bug fixing and maintenance. - Requirements have to be aligned with multiple partners.
3	**Product**	The participant does no development at all. All solution components (client and verification controller) are provided by the Bundeskanzleramt.	- Installation & Configuration Guidelines for running an ACA-Py instance. - Installation & Configuration Guidelines for running the verification controller (+ any additional use case specific components, e.g. keycloak or mongodb) and client (user interface, connectors e.g. email) components. - User Handbook	- Lindner, Steigenberger and Motel One for the Hotel Checkin use case - BOSCH, Deutsche Bahn, Deutsche Telekom, VW for the Access Management use case.	++ More flexibility in progressing standards used by the EESDI network. + No Implementation effort on the partner side. -- More effort in bug fixing and maintenance - Requirements have to be aligned with multiple partners.

Self-attested claims registry

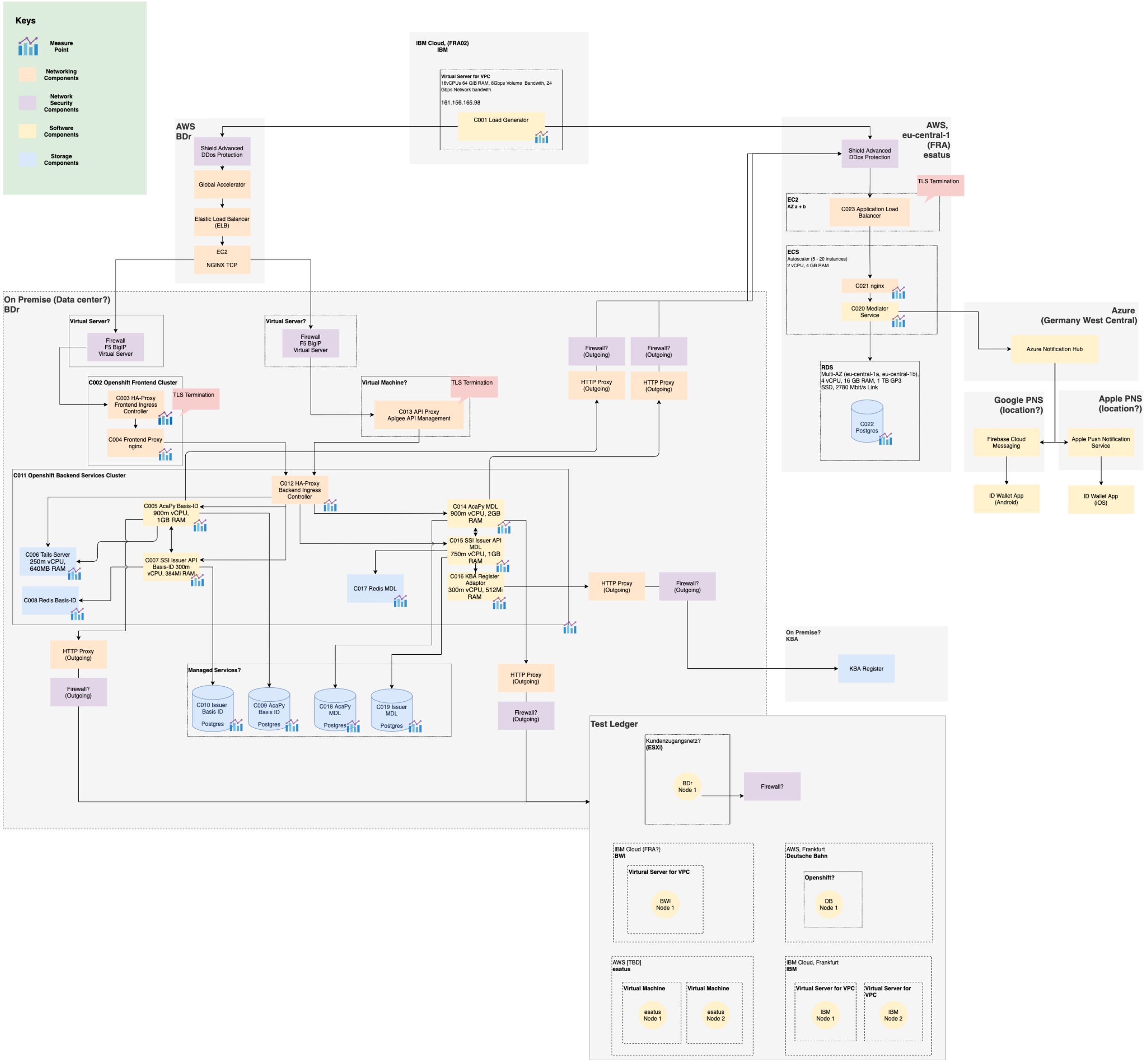
Version	Date	Author	Comment
0.1	04 Jan 2022	[redacted]	Initial version: email, phone_number
0.2	17 Jan 2022	[redacted]	Added: taxID

Following a list of self-attested attributes that have been selected for, and are supported by, the EESDI ecosystem.

The claim names are based on schema.org, JSON Web Token Claims RFC7519 and OpenID Connect Core 1.0 Section 5.1, where available.

Claim Name	Type	Claim Description	Example	Reference	Since
`email`	string	Holders' self-attested email address	jane.doe@my-digi-id.com	RFC5322 schema.org	04 Jan 2022
`phone_number`	string	Holder's self-attested phone number MUST include the country code MUST NOT include space, dash, parentheses MUST only include characters: 0-9, +	+4930867716259	E.164 RFC3966	04 Jan 2022
`taxID`	string	TIN / UID / AHV / NIF / ... Country format independent CAN be numeric and alpha numeric MUST NOT include space, dash, slash MUST include characters: 0-9, A-Z	2202081508156	schema.org	17 Jan 2022

Target Operational Model



Components Description

ID	Component	Description
C001	Load Generator	Generates traffic to perform load and performance tests. Logs test execution data.
C002	OpenShift Frontend Cluster	Openshift cluster used as a frontend layer exposing services to the external. Actually handling the Basis ID incoming requests (DDL is passing though APIGEE instead)

C003	HA Proxy Frontend Ingress Controller	Kubernetes ingress controller: it configures a HAProxy instance to route incoming requests from an external network to the in-cluster applications
C004	Frontend Proxy	Reverse Proxy for the frontend Opeshift cluster.
C005	Aca-Py Basis-Id	Hyperledger Aries cloud agent instance, running aca-py open source implementation. Implements standard DIDCOMM protocol and connects to its wallet (on C009), the tails server (C006), to a mediator agent (C020) and to the nodes on the Hyperledger Indy network. Communicates with a controller (C007) via webhooks (outgoing) and by exposing admin APIs (incoming requests)
C006	Tails Server	Stores tails files for credential revocation information
C007	SSI Issuer API Basis-Id	Implements the business logic for the issuing process. Controls the aca-py instance via the Admin APIs and receives events via webhooks
C008	Redis Basis-Id	Data structure store for the SSI Issuer API Basis-Id (C007) business logic
C009	Aca-Py Basis-Id Postgres	Storage of the aca-py wallet for the Aca-Py Basis-Id (C005)
C010	Aca-Py Basis-Id Issuer Postgres	Storage for the agent controller (C007)
C011	OpenShift Backend Services Cluster	Openshift cluster hosting the components for the backend services. This cluster is shared across multiple use cases (not only Basis ID and DDL)
C012	HA Proxy Backend Ingress Controller	Kubernetes ingress controller: it configures a HAProxy instance to route incoming requests from an external network to the in-cluster applications
C013	Apigee API Management	API management layer for the MDL issuing services
C014	Aca-Py MDL	Hyperledger Aries cloud agent instance, running aca-py open source implementation. Implements standard DIDCOMM protocol and connects to its wallet (on C018), to a mediator agent (C020) and to the nodes on the Hyperledger Indy network. Communicates with a controller (C015) via webhooks (outgoing) and by exposing admin APIs (incoming requests)
C015	SSI Issuer API MDL	Implements the business logic for the issuing process. Controls the aca-py instance via the Admin APIs and receives events via webhooks
C016	KBA Register Adaptor	Enables the integration with the Kraftfahrt-Bundesamt (KBA) register, holding the Central Driving License Register (Zentrales Fahrerlaubnisregister, ZFER), which includes credential revocation information consumable at the application layer by the SSI Issuer API MDL component (C015)
C017	Redis MDL	Data structure store for the SSI Issuer API MDL (C015) business logic
C018	Aca-Py MDL Postgres	Storage of the aca-py wallet for the Aca-Py MDL (C014)
C019	Aca-Py MDL Issuer Postgres	Storage for the agent controller (C007)
C020	Mediator Service	Custom built agent implementing SSI protocols plus specific services for the ID Wallet app and the issuing process
C021	Mediator Nginx	Reverse proxy for the Mediator Service (C020) incoming requests
C022	Mediator Postgres	Storage for the Mediator Service (C020)
C023	Application Load Balancer	Load Balancer for the Mediator Service (C020) incoming requests