System concept for Project 2080069021

"DIGITAL IDENTITIES: HOTEL CHECK-IN PILOT"

Bundeskanzleramt

Author: IBM Deutschland GmbH
IBM-Allee 1
71139 Ehningen

Co-authors: Bundesdruckerei GmbH
esatus AG

Date: 28.4.2021 (Date of German Version, translated afterwards)

Contents

1 Purpose of this document

This document describes the basic architecture concept, identity life cycle, and security requirements for the SSI ecosystem. It focuses on the "hotel check-in" use case (basis ID and company ID) as it is currently being implemented by the federal government. This provides a basis for dialog about the intended purposes of SSI technology in the future and the use of the basis ID.

2 Introduction

As part of its digitization strategy, the Federal Republic of Germany will create an ecosystem for digital proofs of identity. The fundamental assumption behind the new ecosystem is that once a digital proof of identity is issued, the holder controls it. This concept is called "self-sovereign identity" (SSI). Similarly to haptic forms of documentation, digital proofs of identify are recognized as valid if it can be shown that they were issued by a trustworthy source. The use of materials that are hard to copy, watermarks, signatures, stamps, or seals are used to ensure that haptic forms of documentation are trustworthy. In the digital world, digital signatures can be used to show that a document is authentic. In the SSI content, such cryptographically verifiable certificates or proofs of identity – such as a digital driver's license, digital letter of reference, or digital ticket – are known as verifiable credentials (VCs). They are saved locally in the holder's wallet app and can be shown and checked from there as needed.

To demonstrate SSI's potential to enable secure, efficient identity management, the Federal Chancellery (BKAmt) has launched a pilot with diverse participants from the business community. The goals of the pilot are to first test the "hotel check-in" use case in practice and then use it as a basis for further use cases in order to show how decentralized digital proof of identity can be implemented. In the planned pilot, selected pilot employees from four selected companies will load two forms of VC – a basis ID that verifies that the identity card has been entered and a company ID containing their employer's company name and official billing address – into a wallet app (which is provided by the project but can be used beyond it) with standardized functions specifically designed for SSI and use these credentials to check in at selected hotels. For this purpose, selected locations from three hotel chains (Steigenberger, Motel One, Lindner) are taking part in the project.

The results of this project will provide the basis for other SSI use cases in an ecosystem planned to eventually be Europe-wide; these additional use cases will be developed in sector-specific task forces.

The network will employ distributed ledger/blockchain technology to ensure that entries remain unalterable and the infrastructure cannot be manipulated. This decentralized, open approach will also prevent a single organization from gaining control of the ecosystem or the data processed within it, as well as vendor lock-in. Furthermore, the architecture makes it possible for the holder of documentation to provide it to the verifier without the issuer's knowledge.

A conscious choice was made in favor of distributed architecture supported by blockchain technology to avoid having a single point of failure in the form of a central certificate authority and prevent dependence on a central body.

In the first step, the holder requests and receives the required VCs (basis ID and company ID) and saves them in the wallet app. With these VCs they can authenticate their identity to one of the participating hotels during check-in by scanning a QR code provided there with the wallet app and transferring the required and requested data from the two VCs to the hotel. This process eliminates the need to separately provide the company address for billing and to fill out a registration form.

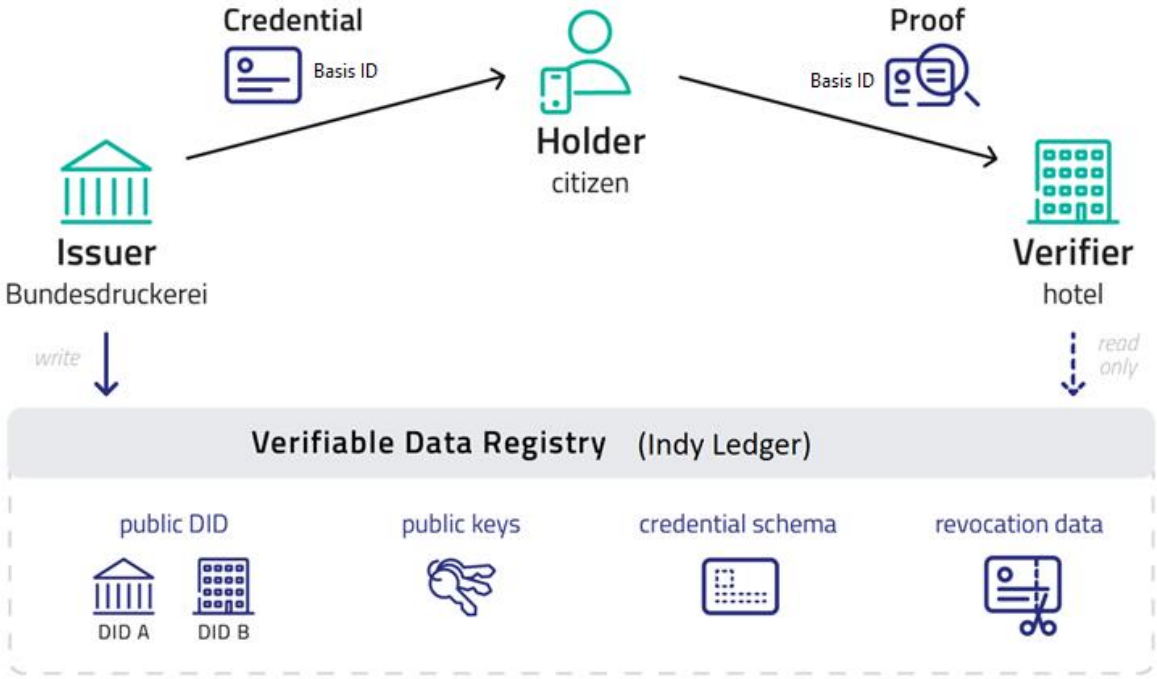


Exhibit 1: Process overview. Only the basis ID is shown in the illustration

2.1 Pilot scope and limitations

The scope of the pilot covers the use of the system described above for the "hotel check-in using the basis ID and company ID" use case by employees of the specified participating pilot companies for 6 months from the pilot start date (planned for May 17, 2021). The participating pilot companies at the start are Bosch, Deutsche Bahn, Deutsche Lufthansa, and BWI. The group of participating employees is limited by the companies in question and can be tracked. At all pilot companies, only company mobile phones managed by the companies themselves will be used for the use case. At the start of the pilot, the pilot companies will take organizational measures (in the form of instructions to their employees) to prevent other (e.g., private) smartphones from being used for the pilot instead of the specified company phones. Technical measures to ensure device security and organizational measures will exclude smartphones with operating systems older than Android V7 or iOS V13 from the pilot (see chapter 7.1.1).

The above-mentioned measures will take the form of corresponding instructions from the employers to their employees.

If additional participating pilot companies are given the right to use the existing system with their employees, they will be required implement the same organizational and technical measures described here.

2.2 System overview

The system consists of the following entities:

1. **Verifiable data registry:** A public permissioned blockchain network (Hyperledger Indy consisting of Indy nodes) for storing public key material and a decentralized verification infrastructure operated by authenticated, authorized node operators
2. **Issuer**: Issues identities to the holder in the form of credentials. Issuers are currently the Bundesdruckerei (for the basis ID) and the Deutsche Bahn, Bosch, Deutsche Lufthansa, and the BWI for the company ID.
3. **Holder:** Holders receive the credentials from issuers, save them in their wallet, and can present them to verifiers
4. **Verifier**: The verifier is the recipient of credentials and the identity attributes ("claims") that they contain or logical statements based on them and triggers the verification process through the

network to check the data's integrity and authenticity. The process from the scanning of the QR code to the check against the blockchain network takes place on the hotel servers.

The following entities are involved in the current "hotel check-in" pilot:

1. The Bundesdruckerei as the issuer of the basis ID
2. Deutsche Bahn, Bosch, Deutsche Lufthansa, and the BWI as issuers of the company ID
3. Hotels as verifiers of the IDs
4. Holders (company employees) with the wallet app on their smartphones
5. BWI GmbH, Deutsche Bahn AG, Bundesdruckerei GmbH, IBM Deutschland GmbH, and esatus AG as node operators that make the blockchain network available

Other technical components involved and contained in the system:

1. **Wallet app:** For storing, releasing, and transmitting credentials; operated by esatus AG
2. **Aries agents**: Software agents that provide SSI functionalities for sharing with other agents, for communication to make the verifiable data registry (Hyperledger Indy) available, and to enable communication with the wallet app; operated by all participants
3. **Tails server** for each issuer that makes the Tails files essential for revocation publicly available for download; operated by the issuers
4. **Mediator service** that acts as a notification service and inbox for incoming SSI-related DIDComm messages for mobile end devices (outgoing communication takes place directly with the relevant end points). Additionally packaged messages are placed in the corresponding individual inboxes there. After receiving such a message, the mediator service also sends a notification to the end device through an Azure Notification Hub. This notification only provides the information that a new message has been received and alerts the wallet to the incoming communication, without including the content or any reference to it. The mediator service itself cannot read the messages, which can only be decrypted with the keys for the connection in question on the end devices used to transmit the message. This service is operated by esatus.

Also see Exhibit 2: Components of the IT network.

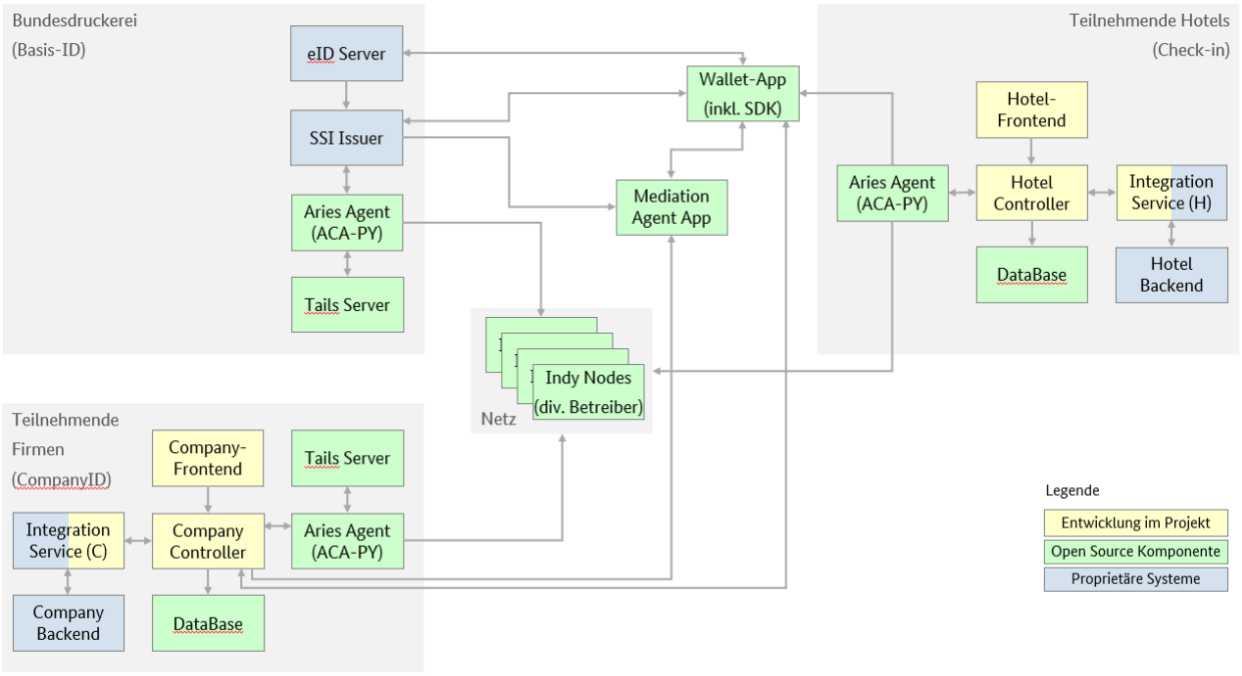


Exhibit 2: Components of the IT network

2.3 Security technology requirements

All node operators have an ISMS in place that meets the criteria laid out in ISO 27001.

3 Hyperledger Indy

The infrastructure involved uses Hyperledger Indy, a public-permissioned blockchain. The network consists of multiple network nodes (validator nodes) that can be operated by specified participants (stewards) as a permissioned network. The validator nodes build trust in one another through the consensus algorithm. With Hyperledger Indy, consensus is created with the plenum protocol: a Redundant Byzantine Fault Tolerance algorithm (RBFT), a redundant version of the PBFT, which has been evaluated as secure. RBFT provides parallel processing of PBFT instances, so the consensus it achieves provides security characteristics that at least match those of PBFT [3] (no separate proof of security exists for RBFT itself.) If the maximum number of malicious or faulty nodes in the network is F, consensus requires a network with N = 3F + 1 total nodes. In addition, up to 1/3 of the nodes can be temporarily down or unreachable without affecting the network's functionality. They can be updated by the working nodes later and participate in the consensus once again. Read access to the network is freely available and does not require authentication; this is analogous to the typical characteristics of a public blockchain. However, write access requires permission, which is only granted if both the author's signature and an endorser's

signature are provided. The signatures are checked by the validator nodes. Apart for the actual read and write access, the validator nodes are constantly in contact with one another and regularly perform an auditor transaction to monitor the consensus.

A network of 15-25 validator nodes makes sense and should be the goal as this size ensures good protection against outages and enables the consensus algorithm to work most efficiently. Based on our present knowledge, such a network could handle 50 to 100 transactions per second [2]. Currently just 5 validator nodes are operating; during the pilot the peak transaction load per second is expected to be in the low single digits. If more than 1/3 of the nodes stop working, the network switches into “read only” mode. Due to the lack of consensus, no new transactions are possible, but verifiable read access can still take place. In the pilot, roles are clearly assigned among the project participants. No nodes will be shut down.

Generally a governance organization exists to complement the trust-building roles of the technical basis. This organization represents the network and provides a legal framework to the real business world. It enters into contracts with people and organizations, obligating them to their technical roles and the resulting rights and responsibilities. These responsibilities primarily ensure the trust that the parties involved will act according to their best knowledge and conscience, taking all needed precautions to protect the network.

IBM will support the federal government to define a governance organization within three months after the hotel pilot goes live, align on it with network participants, and reach formal agreements with them.

For the transitional phase until the formal governance agreements are reached, the pilot participants have agreed to fulfill the roles described in 3.1 for at least the duration of the pilot. IBM will monitor role fulfillment and compliance with rights and responsibilities during this transitional phase and, if necessary, intervene in a guiding capacity with the goal of protecting the network and ensuring its stable operation. On behalf of and in consultation with the federal government, IBM thus assumes governance responsibility until the governance organization has been defined.

Hyperledger Indy underwent penetration testing in 2018 that examined both the node software (indy-node) and the consensus protocol (indy-plenum) [4]. Part of further planned pen tests will involve checking whether the findings from previous tests have been addressed.

For the pilot project system, Deutsche Bahn – representing the employers/issuers of the company IDs, conducted a further pen test of the externally accessible components of the government portal frontend,

Tails servers, and SSI agent. No weak points were identified and the application in its current configuration was found to provide a high level of security.

3.1 Roles

To enforce the rules of the permissioned network, an individual configuration and integrated system of rights and rules are saved in the ledger. The roles are:

Trustee: Trustees administer the rights of other members and can assign them roles (and the associated right) within the network. They also specify the set of rules and the detailed rights and requirements for each specific role. In this project, the trustees are: BWI GmbH, Deutsche Bahn AG, Bundesdruckerei GmbH, IBM Deutschland GmbH, esatus AG.

Steward: The stewards operate the network nodes (validator nodes). Each steward operates one of the nodes and has the right to change the associated information, such as the IP, port, and key. In this project, the stewards are: BWI GmbH, Deutsche Bahn AG, Bundesdruckerei GmbH, IBM Deutschland GmbH, esatus AG.

Endorser: Endorsers approve write access for regular data transactions. Their main task is to write identities without other roles, such as the issuers, onto the network by authorizing transactions with their signature. In this project, the endorsers are: BWI GmbH, Deutsche Bahn AG, Bundesdruckerei GmbH, IBM Deutschland GmbH, esatus AG.

Network monitor: The network monitor can read out stewards' metainformation in order to check and track the network's overall status and condition. The network monitor is not able to write or read security-relevant information; it is quasi just an "observer" and can only trigger other network participants, for example with automated e-mail notifications about irregularities or by contacting node stewards via an emergency contact list. The network monitor is part of the governance organization: for the pilot, IBM is handling the observer/trigger function of this role on behalf of the federal government. In this way, the principle of avoiding central control is upheld.

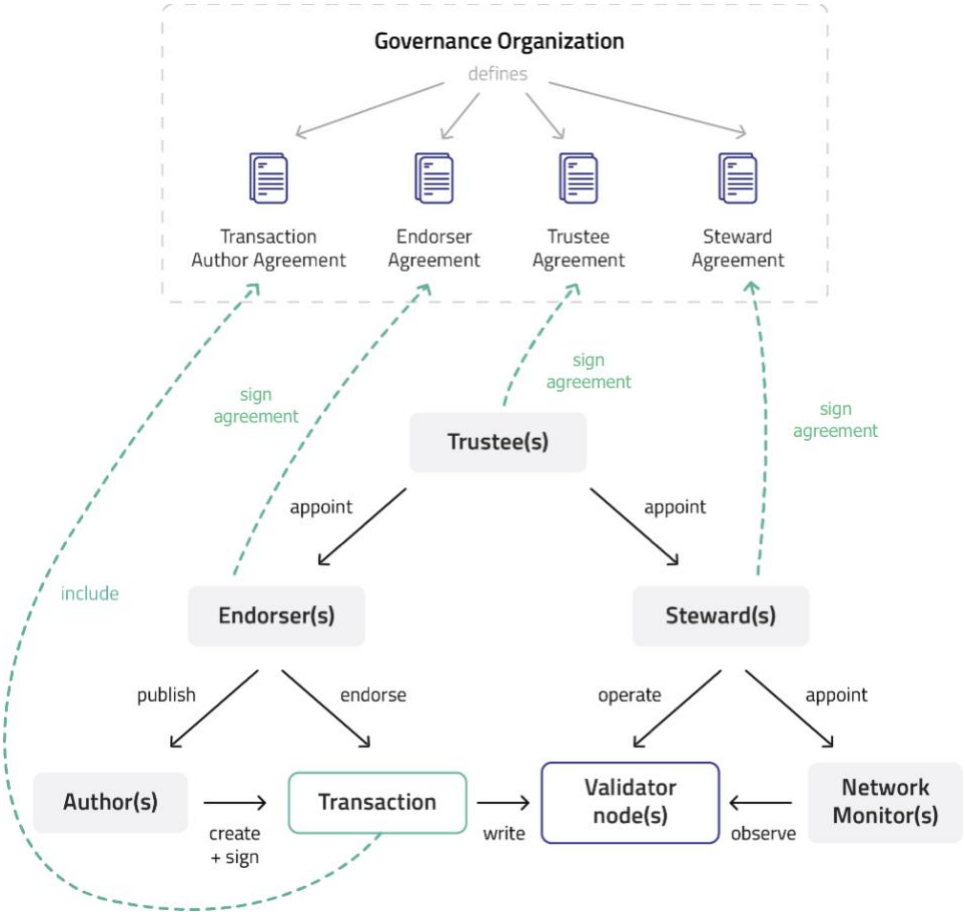


Exhibit 3: Interaction among involved roles

Each of these roles is a so-called decentralized identifier (DID) on the network. Transaction authorization can be verified through the nodes by validating the role of the DIDs and the signatures with the corresponding public key. Project members are identified and allocated to their roles using existing means of communication (Confluence for project with a secure HTTPS connection and user authorization via username/password or e-mail encryption/signature). Other communication also take place through these, especially the sharing of public keys. Within the project, all participants will be securely identified. For this purpose, the following process was established and applied; an analogous approach will be used for any changes that take place:

1. The federal government names and authorizes all companies participating in the pilot and informs the scrum master (an IBM employee assigned to this role). To provide this information at the start of the project, it is included in the project documentation; changes are shared by e-mail or in another written form.

2. Each pilot company names (or has already named) a contact person (and possibly one or more deputies) who are authorized to request access to Confluence for additional employees.
3. The scrum master receives e-mail requests to give additional people access to Confluence from either the federal government (for new companies joining the pilot) or from one of the contact persons described in point 2 above. The request includes the first and last name, e-mail address, and company affiliation of the employee to be authorized.
4. The scrum master reviews the authorization from the contact person and checks that the data on the person to be authorized is valid and complete.
5. If the check is successful, the scrum master uses the PMO dashboard to enter a new user with the following information:
    a. User type (IBM, external contractor, customer, external (IBM))
    b. First and last name
    c. E-mail address
    d. Individual tag
    e. Description
6. The scrum master specifies the access and assigns the user to a user group. The following user groups are available and have defined access rights:
    a. BKAmt-IBM
    b. BKAmt-IBM Lead
    c. BKAmt-Client
    d. BKAmt-IBM-Contractor
    e. BKAmt-PM
7. The new project member automatically receives a system-generated e-mail with self-registration instructions and can set an individual password.
8. The scrum master maintains a document with the people added and access rights that were issued.
9. Although the process described above covers new companies joining the project, such additions are not foreseen for the pilot.

The node operators – more precisely, the Hyperledger Indy software they use – enforce the rules. Node operators are not expected to join or leave the project during the pilot.

The specific rules on which and how many approvals (and therefore signatures) are needed to enter identities and roles is contained in the set of rules in the ledger. In addition to the rules for creating and changing identities, schemas, credential definitions, and revocation information, it also contains rules on the validator nodes and for changing the rules themselves. The terms of the rules can refer to one or more

roles and be combined with logical operations. Normally a governance organization sets the rules and the trustees technically implement them. Currently the standard set of rules is being used[1].

3.2 Creating the blockchain network

To create a network, each node operator first generates local cryptographic material in the form of a Boneh-Lynn-Shacham key pair (public and private key) for authenticating the node (required for consensus). Each node operator also generates a key pair (Ed25519) and a DID derived from it. The private key is administered by the Indy Node software contained at each node of the file system. This DID can be assigned a steward role (for all node operators, this takes place when they accept the genesis file; see below), which later enables the node operator to administer its network nodes. Each node operator's public keys (BLS and Ed25519) are shared with all other node operators, along with the IP address of the server on which the node will run and the two ports used for networking (for node-to-node and node-to-client communication) so that the nodes can link to one another and the network can begin. To do so, two genesis files are generated jointly (within the project's secure Confluence): The pool_transactions_genesis file describes the initial node operators (IP, port, and public keys) and the domain_transaction_genesis file describes the initial trustees and stewards. The genesis files and their hashes are shared with all node operators over a secure transport medium (via the project's Confluence with a secure HTTPS link and user authentication with username/password) and checked for accuracy (own information and keys are checked, with an additional review in the form of a SHA256 checksum). Then the genesis files serve as the configuration for the network launch. The nodes connect and create a consensus, and the network is ready for the first transaction. To provide additional authentication, the network launch is recorded on video; in this video, the project participants confirm their identities and the authenticity of their public keys and then start the network together. The genesis files are not signed; instead, the publication of the hashes in the project's Confluence ensures the network's integrity. For the pilot project, the genesis files for the various entities involved (issuers, wallets, verifiers) are integrated into the software during development ("hard-wired"), which ensures trustworthiness and integrity. In the future, genesis files and/or their hashes could be recorded in newspapers, federal bulletins, or committees' online directories. Each node operator activates the IP data packet filtering function (iptables) on the host system. As a result, incoming data packets are checked before they are transferred to the target application (Indy Node process). Outgoing data packets are also checked before they leave the node. A shared configuration and

[1] https://hyperledger-indy.readthedocs.io/projects/node/en/latest/auth_rules.html

overview of incoming and outgoing IP addresses is maintained in Confluence for all node operators. The filter blocks connection requests that do not originate from known IP addresses and does not allow them to link to the server.

All initial trustees and stewards are saved in the domain_transaction_genesis file, while all other entities and DIDs required for the pilot are added to the network separately when they are assigned their steward roles.

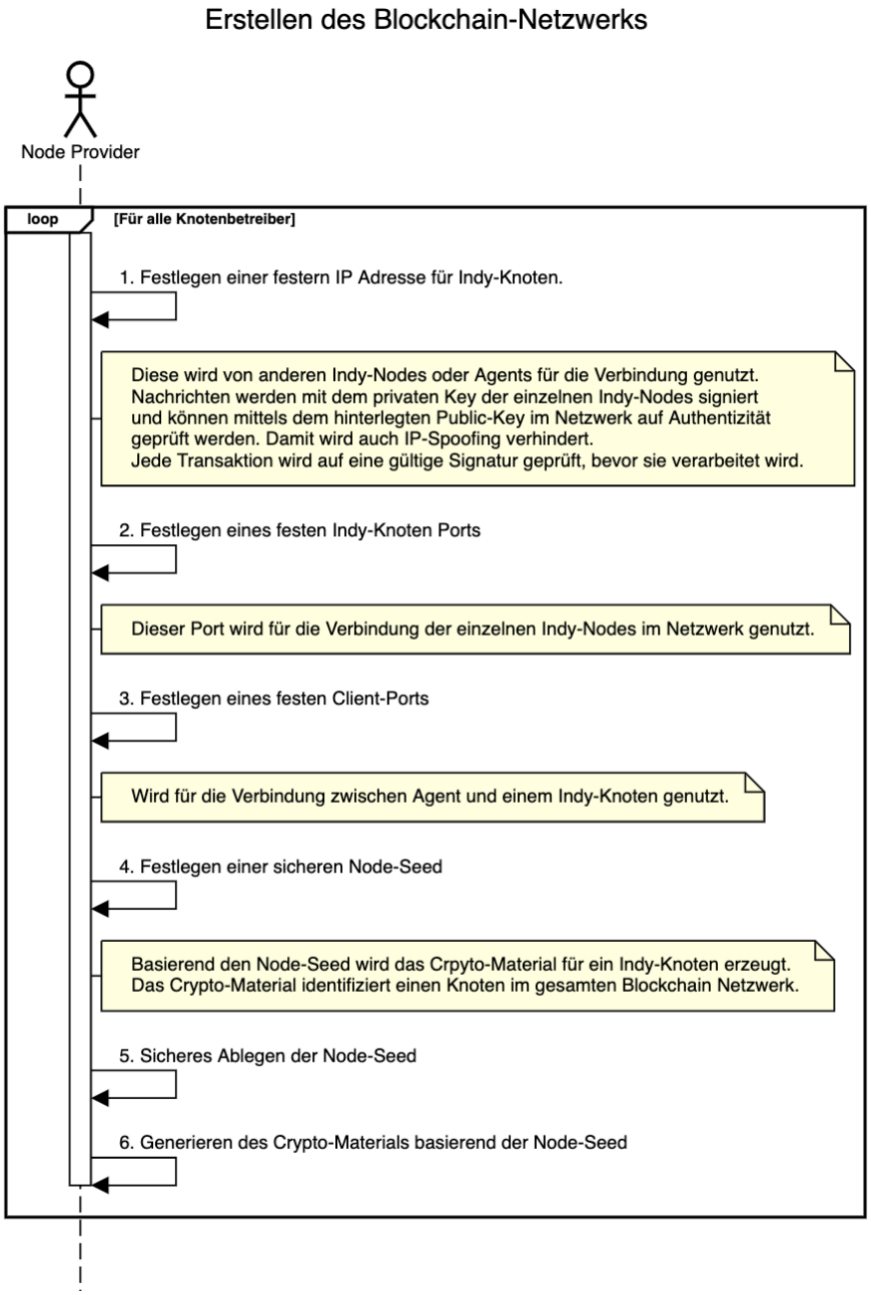


Exhibit 4: Creating the blockchain network

3.3 Onboarding a public DID

To make an initial transaction, an author (issuer, verifier, node operator added later) creates a key pair (Ed25519) and the DID that corresponds to the public key. The DID and the public key are packaged in an initial transaction. The author must digitally sign all further transactions with their private key so that the data's authenticity is assured and easy to check. The author sends the resulting transactions to multiple nodes. Transaction finality and consistency is reached as soon as the author receives F + 1 (F= maximum number of faulty nodes; see above) successful responses. Consensus on the data is ensured by using a Boneh–Lynn–Shacham (BLS) multisignature that unequivocally proves the consistency of the validator nodes so that a read request only needs to be made to a single node. A Merkle tree is used to save the transactions as an organized list. The current state proof and the auditor path for the Merkle tree can be used to check that the ledger status is correct. In turn, the authenticity of the validator nodes can be assured with their public keys from the genesis file. The genesis files are the primary trust anchors and are shared securely among all participants (as described above).

Most of the identities on the network (issuers or verifiers) are public institutions that store a DID and the matching public key only and have no further rights. This information is shown in the exhibit below (the main content of the transaction in "txn") and only relates to the author (company); no personal data is included. It is supplemented by metadata such as validation information for the validator nodes (auditPath, reqSignature) and information about the position in the blockchain (ledgerSize, seqNo).

```
{
  "auditPath": [
    "GDyUoeC5rbK9drQYDmrWEgZBHnj4jW1wXjWb74h3ttnv",
    "72Yze51yrDBkbFgR5ZdUJiLB9QbP9JTw8tKJbacayVKG",
    "D52hsZf4iH4Kp4x4eEp18FicbPNdirG9TL2cvat1eKvL",
    "2fKw6teotaep8GDZyzBbdHUsUXwXh8nxGHUkPnLhjU8j"
  ],
  "ledgerSize": 10,
  "reqSignature": {
    "type": "ED25519",
    "values": [
      {
        "from": "4cU41vWW82ArfxJxHkzXPG",
        "value": "2iFCpuo137eX73kDCr4Ts8WHEyRNu1wJRDUkhivxARMUZMGRSAwefGPvwB9giggrBZnhRvUGHm5FojkaFmdz4b88"
      }
    ]
  },
  "rootHash": "CLqaCCYRZv4cinYNYVmBowEeuPKaDQ3iyGno2eNSHCwa",
  "txn": {
    "data": {
      "alias": "Company",
      "dest": "JiVLsA5wxVnbHQ5s7pDN58",
      "role": "101",
      "verkey": "Af1wegtCKdqjjWqjdP89tkLGLhfM8oRoLWaHpn8wpRU8"
    },
    "metadata": {
      "digest": "442a28bd1b9947867fc65ec8ef62c2b21d465498224a729f322143bce3967351",
      "from": "4cU41vWW82ArfxJxHkzXPG",
      "payloadDigest": "15bab600e46d5119df83e75bb56469eb185baebcc5e1c6e597956ec4f83acf50",
      "reqId": 1612383608927929900
    },
    "protocolVersion": 2,
    "type": "1"
  },
  "txnMetadata": {
    "seqNo": 8,
    "txnId": "80f7aeea87d46e00d0516528924aa825658bb3c294b0c6ef4e5de4738455107d",
    "txnTime": 1612383611
  },
  "ver": "1"
}
```


Exhibit 5: Onboarding of a public DID; Indy-specific representation

3.4 Creating the schemas and credential definitions

In a next step (before the pilot begins), the issuers must define the VCs they issue.

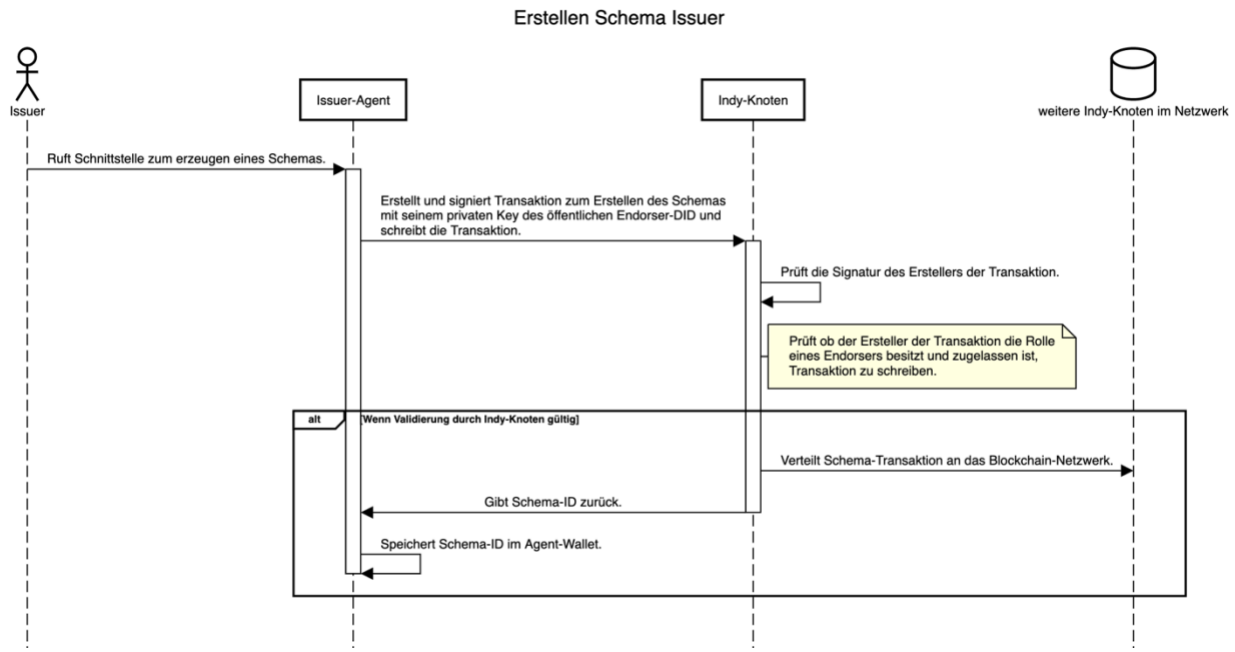


Exhibit 6: Sequence diagram – creating an issuer schema

First a fundamental schema ("template") is created on the Indy blockchain (by a steward). It contains a definition of the attributes VCs must include in order to conform to the schema. The schema for a basis ID, for example, contains a subset of the data fields from a German identity card (city, family name, birthplace, birth name, first name, date of birth, street address, country, expiration date, academic title, postal code). The name of the schema, the attributes it contains, and the issuer's DID are stored in the blockchain. The following exhibit shows the schema for a basis ID: In addition to the metadata described above, the form of the schema is provided in "txn"; specifically, its name (in the case of the basis ID) and the attributes that a basis ID must contain. This means that the schema also contains **no personal data**.

```
{
  "auditPath": [
    "GtBkPbR2wgGKuthxFj9SGZicEqXkEgYunRhD6CEGYrS3",
    "CWYPYX5TRgb3yGzVB7ztNgbxUjygjvPqwar9BqwAHMWE",
    "Bn1HMU1FpTWZmd3JPMP4bCcCzTMUP1wcxVdQbHB5Q2Ha"
  ],
  "ledgerSize": 29,
  "reqSignature": {
    "type": "ED25519",
    "values": [
      {
        "from": "XwQCiUus8QubFNJPJD2mDi",
        "value": "Wqz2dzouY5GcgEpoJLqTqtULJSiauCnnvtAsPGXCof3eDwjWxoVZ7YL8N4VLAErlgX7hTsLT4JHzRVTUhknm75w"
      }
    ]
  },
  "rootHash": "FtFcEWvc9AZagBey6ApTK1WZm6zPjSuPczyEpABj7HCJ",
  "txn": {
    "data": {
      "data": {
        "attr_names": [
          "addressZipCode",
          "academicTitle",
          "dateOfExpiry",
          "firstName",
          "familyName",
          "birthName",
          "addressCountry",
          "dateOfBirth",
          "placeOfBirth",
          "addressStreet",
          "addressCity"
        ],
        "name": "masterID",
        "version": "1.0"
      }
    },
    "metadata": {
      "digest": "d8dc8812cb4b73b8d25b55607985f6ddb8195239b2c466fab72523e0e97343ed",
      "from": "XwQCiUus8QubFNJPJD2mDi",
      "payloadDigest": "2a03105bcac1874a2c4b4ec245698a50f6a740944042479a04d9f282b17fc66a",
      "reqId": 1612453082228177400
    },
    "protocolVersion": 2,
    "type": "101"
  },
  "txnMetadata": {
    "seqNo": 29,
    "txnId": "XwQCiUus8QubFNJPJD2mDi:2:masterID:1.0",
    "txnTime": 1612453084
  },
  "ver": "1"
}
```


Exhibit 7: Creating a schema

Using the schema definition as a basis, a separate “credential definition” is then created by each participating issuer that wants to generate VCs based on a schema.

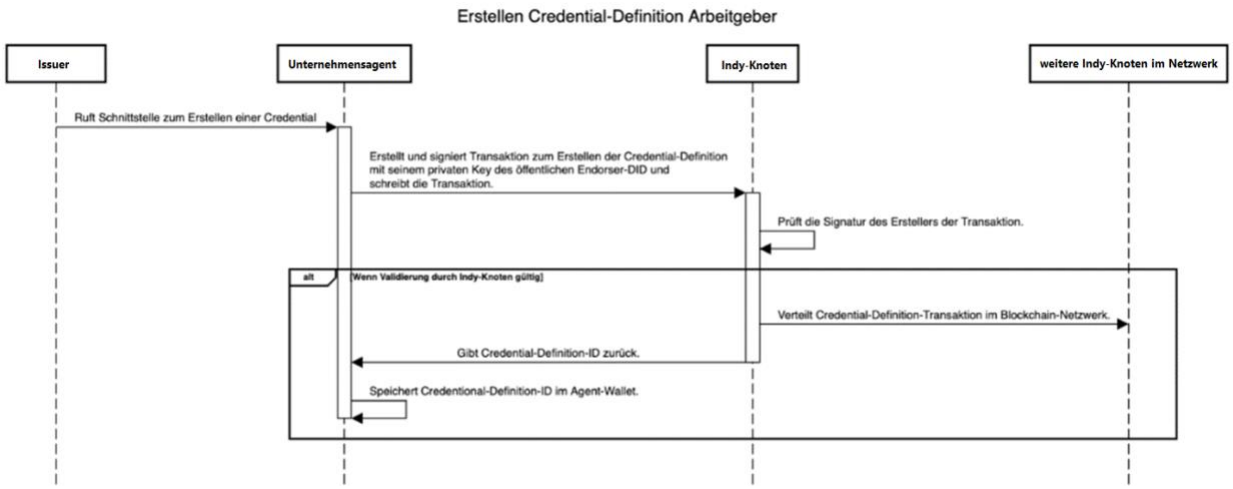


Exhibit 8: Sequence diagram – creating an employer's credential definition

In addition to the information in the schema to which it refers (referenced via the block number in which the schema was defined), the credential definition also contains its creator's DID (and, as a result, the public key). In addition, a large random number (2048 Bit) is generated for each type of attribute, which is needed later as a reference when holders and verifiers provide proof of the attributes (a so-called common reference string, required for the noninteractive zero-knowledge proofs that are used and part of the issuer's public key). This means that **no personal data** is contained in the credential definition either. The following exhibit shows an example of a credential definition for the basis ID as generated by the Bundesdruckerei.

```
{
  "auditPath": [
    "AzHyGe2gJk9LGdvXbGeVDZml91Sy7Ze7Yu4y3FExokzr",
    "GtBkPbR2wgGKuthxFj9SGZicEqXkEgYunRhD6CEGYrS3",
    "CWYPYX5TRgb3yGzVB7ztNgbxUjygjvPqwar9BqwAHMWE",
    "Bn1HMU1FpTWZmd3JPMP4bCcCzTMUP1wcxVdQbHB5Q2Ha"
  ],
  "ledgerSize": 30,
  "reqSignature": {
    "type": "ED25519",
    "values": [
      {
        "from": "XwQCiUus8QubFNJPJD2mDi",
        "value": "mjxwU1PHBLG6juvbVK7NKfv2MzEXJvSTEB5v7UMadhSBMtyezy6z21UxXTydEWYxtvQ7reXWqAspik442darH4V"
      }
    ]
  },
  "rootHash": "BohGJuyZmkXe9eFrAMdyPYmGHMGgX25K1J3kkYMTXN7H",
  "txn": {
    "data": {
      "data": {
        "primary": {
          "n": "840562701292560653023914371181975694221465010621576986698298406870232182967194431122063474869249
          "r": {
            "academictitle": "709122764334594511067197538962319465440932227597742774530130410619446365528909439
            "addresscity": "183195248843622939514051116276911457245907552392897628015956823794966009539797491766
            "addresscountry": "74259298020794733366790695863265679665861303633192331680593270731230458688525360
            "addressstreet": "194688873743903851819928177017169767995716546189085280619551712609664083006799100
            "addresszipcode": "609235921525986719986803721589082876487289590470936718725936570951961274219021177
            "birthname": "839216944518149523922576126495803180046545036377538189421330392117690852381506672383
            "dateofbirth": "215527628935379508348776796782682158690693530700849141419982020622312312679830120888
            "dateofexpiry": "486431228116713978427377547693790343782777860114060904297385991228395029124673869
            "familyname": "242813786382269302377029615577582543208022152102149858153973102408957223852289638419
            "firstname": "548399757135805729753684616549843739150443791419724648206956062845218771666443012942
            "master_secret": "806077577395143502308814747121978396707485829256593123955264855027606065563227716
            "placeofbirth": "294856775019554300672731363952960512503482934266499688611082896300431572964991952
          },
          "rctxt": "616907373077490893438330567517088394044763285810601971881408044450448806002443467392765995
          "s": "239293558625126098235679476186971846363889966503577196658813069332187922393956045041506472086
          "z": "292094834741225720265939351788492984399668375009337617165150315789273250242585307645388687785
        }
      },
      "ref": 29,
      "signature_type": "CL",
      "tag": "masterID Dev"
    },
    "metadata": {
      "digest": "117fc7af65cbbc74705fb82c5f8d58a73f9c2ffbe7c7a9afd1d2aadef6dfe3fd",
      "from": "XwQCiUus8QubFNJPJD2mDi",
      "payloadDigest": "0fed6da4a1e6484a0ac80f596093af234528db1a63bce7ebe45d4f83a489fb55",
      "reqId": 1612453139691817200
    },
    "protocolVersion": 2,
    "type": "102"
  },
  "txnMetadata": {
    "seqNo": 30,
    "txnId": "XwQCiUus8QubFNJPJD2mDi:3:CL:29:masterID Dev",
    "txnTime": 1612453142
  },
  "ver": "1"
}
```


Exhibit 9: Creating a credential definition

3.5 DIDComm

DIDComm (short for “DID communication”) is asynchronous, message-based, stateless communication between two (or more) agents and agent-link “things” in decentralized DID-based identity systems. DIDcomm makes it possible for certain workflows or protocols to be carried out across two or more communicating parties; these could be, for example, establishing a connection between two agents, issuing credentials, or sending proofs. Messages are (generally) represented in JSON and use elements of

JSON-LD. The messaging protocol is completely transport-agnostic; messages can be transmitted via HTTPS, QR codes, NFC, or Bluetooth or on paper.

Connections can use one-time DIDs created ad hoc or public DIDs saved to the network with the corresponding stored keys. Holder wallets use a new randomly generated DID every time they establish a connection. Issuers and verifiers can use public DIDs to establish the DIDComm; in that case the wallet can obtain the trustworthy public key from the ledger. Otherwise the issuer or verifier must use another method to guarantee that the public key provided is trustworthy, for example through the messaging channel used. If HTTPS is used as the messaging protocol, the invitation is sent over an encrypted, authenticated channel to prevent an MiTM attack.

When the basis ID is issued, the public key pinning of the Bundesdruckerei TLS certificate in the wallet app ensures the authenticity of the issuer (Bundesdruckerei) (see chapter 8.1.3.2). The wallet app contains the Bundesdruckerei TLS certificate. The creator of the wallet app received this certificate from the Bundesdruckerei via secure e-mail. The DIDComm invite is routed to the wallet via an API end point. The invite, in turn, contains the Bundesdruckerei's (issuer's) DID.

To provide proof of identity (for hotel check-In), the holder scans a QR code at the hotel reception. This QR code is the trust anchor for the process and points to an API end point, where the DIDComm invite is stored with the verifier's DID it contains. The holder of the credential in the context in question (e.g., location) must decide whether the QR code to be scanned and the proof request that arrives for confirmation comes from a verifier that can be trusted. In the hotel use case, only QR codes displayed in the hotel reception areas are used. This fact is also communicated to pilot participants in advance. Furthermore, when the connection is established the verifier's endpoint HTTPS URL and name are displayed for the holder in their wallet app so that the holder knows who the verifier is (host name displayed). The verifier does not have to be part of the current network; instead, they can take on the role anonymously.

3.6 Issuing a credential

To issue a credential, a DIDComm connection is established between the issuer and the holder (see chapter 4.1). By using the session-based key from a previous holder authentication, the issuer can allocate the DIDComm connection. The credential is issued according to Credential Protocol 1.0 (Aries RFC0036) within this connection. Doing so essentially involves three messages (credential proposal, credential request, and the credential itself) traveling between the issuer and the holder. The result is a credential

signed by the issuer; the legitimate holder can verify its authenticity and integrity to third parties with a zero-knowledge proof. This process uses a Camenisch-Lysyanskaya signature that takes the following data into account so that they also become part of the credential:

(*Holder) attributes* (determined by issuer; visible for holder and issuer) are the actual payload the credential contains; this is the data that can later be selectively shared with a verifier.

The *link secret* is a "hidden attribute"; it is initially generated by the wallet and the issuer adds a blind signature (in combination with a "blinding factor"). Every issuer's signature on a link secret is blind. Every VC for an individual holder has the same link secret. This means that it is an attribute of the credential that is invisible to the issuer. When presenting the proof, the holder demonstrates their knowledge of the link secret and, when using more than one credential, that the link secret matches in each case. This proves that the credential belongs to a legitimate holder; see section 8.4 for the process used for this purpose.

The *revocation information* (optional; determined by the issuer and visible to both the issuer and the holder) enables the issuer to block the credential even after it has been issued. The holder needs the revocation information when presenting the credential to prove that it has not been revoked/blocked. Standard Hyperledger Indy mechanisms[2] are used. As a result, the revocation information is contained both in the credentials (as a second "hidden attribute") on the network/blockchain as a cryptographic accumulator and in the Tails files provided by the issuer.

In conjunction with the credential definition, the issuer adds a "revocation registry" to the ledger that contains the cryptographic accumulator and the URL to the Tails file. The size of the revocation registry is clearly specified; the issuer generates a private accumulator key and uses it to calculate the public key and the accumulator values for the maximum number for the revocation registry. The issuer calculates an initial accumulator status, which includes all (not yet issued) accumulator values (in the form of a product with the accumulator values as factors). The issuer saves the accumulator status on the ledger. This process takes place once and occurs when the credential is defined. The Tails file itself contains no sensitive data; it is hosted by the issuer and called up (once) by the holder only. Each credential is assigned to a specific revocation registry and a specific index in the Tails file; this index, along with a witness and the hidden attribute, is transmitted to the holder when the credential is issued. When presenting the proof, the holder can use the revocation information in the credential to show that they are part of the

[2] https://hyperledger-indy.readthedocs.io/projects/hipe/en/latest/text/0011-cred-revocation/README.html

accumulator and so demonstrate that the credential has not been revoked. If the status of the accumulator has changed since the credential was issued, the wallet holder must query the witness delta indices from the ledger (saved in the revocation entries) and update the corresponding values from the Tails file.

If the issuer wants to block a specific credential, they remove the corresponding Tails file index value from the accumulator and write the updated accumulator status in the ledger. To do so, the issuer must internally save information on how the credentials map to the accumulator values/Tails file indexes. Once this update takes place, the holder can no longer demonstrate to verifiers that their credentials have not been revoked because their witness and their hidden attribute contain no cryptographic proof that they are part of the current accumulator status.

3.7 Credential presentation

The first step in presenting a credential is to create a DIDComm connection. This is initiated with a DIDComm invitation. It contains the verifier's DID, the address of the end point for the SSI agent, and a session-based public key. The wallet SSI agent uses this information to create a connection to the verifier. At this point the verifier can generate a proof request and send it to the wallet through the DIDComm channel.

The wallet trusts that the holder's personal data is being sent to the right verifier (in this case, the hotel reception desk) based on the fact that the holder scans a QR code posted in the reception area. Each reception desk uses a different QR code. Reception employees are required to take steps to ensure that unauthorized third parties cannot switch the QR code; these steps could include physical measures to prevent tampering (such as posting the QR code in a locked display or on a screen) or posting it in an area visible to reception staff and removing it when staff is not present.

When the connection is established with the wallet app, the name of the end point (HTTPS URL host name) that the wallet is connecting to is displayed as well. This enables the holder to see the verifier's identity before answering the proof request.

A proof request contains the requested attributes and predicates along with any conditions, such as rules restricting the response to a specific credential definition. For example, the unambiguous credential definition ID from the basis ID can be required so that only valid credentials (in this case, for example, from the Bundesdruckerei) are allowed. The attributes presented bear the credential issuer's signature and the verifier can use the public key in the credential definition to validate the credential. In addition to

the verified sharing of attributes, it is also possible to use a zero-knowledge proof to share specified predicates. Data is saved for this purposed as comparable integers and can be analyzed using the arithmetical operators =, >, <, >=, and <=. Using this method, it is possible to perform a comparison to determine whether the holder's age is over 18, for example. In addition to the visible attributes, each credential contains two "hidden attributes" representing the link secret; the second hidden attribute is used to check the revocation status. If more than one credential is requested, the verifier can use a zero-knowledge proof of the link secrets for all the credentials to check that the credentials all come from the same wallet. The wallet can also show that the revocation attribute is part of the credential's cryptographic accumulator and, therefore, that the credential has not been blocked by the issuer.

Essentially, every entity with writing rights in the network can use a schema and freely choose names for credential definition. Specific credential definitions are assigned to basis IDs off-ledger; in the project context that means that a second channel (Confluence for project or an encrypted/signed e-mail) is used to provide credential definitions for basis IDs to verifiers. More specifically, in the current project verifiers use a solution prepared as part of the project by IBM that provides a credential definition for the basis IDs.

4 Identity life cycle (basis ID)

This section describes the life cycle of the basis ID.

4.1 Enrollment

Exhibit 10: Basis ID chapter provides an overview of how, through the online identity function, a base ID is created for and provided to a user. The following subsections describe this process in detail.

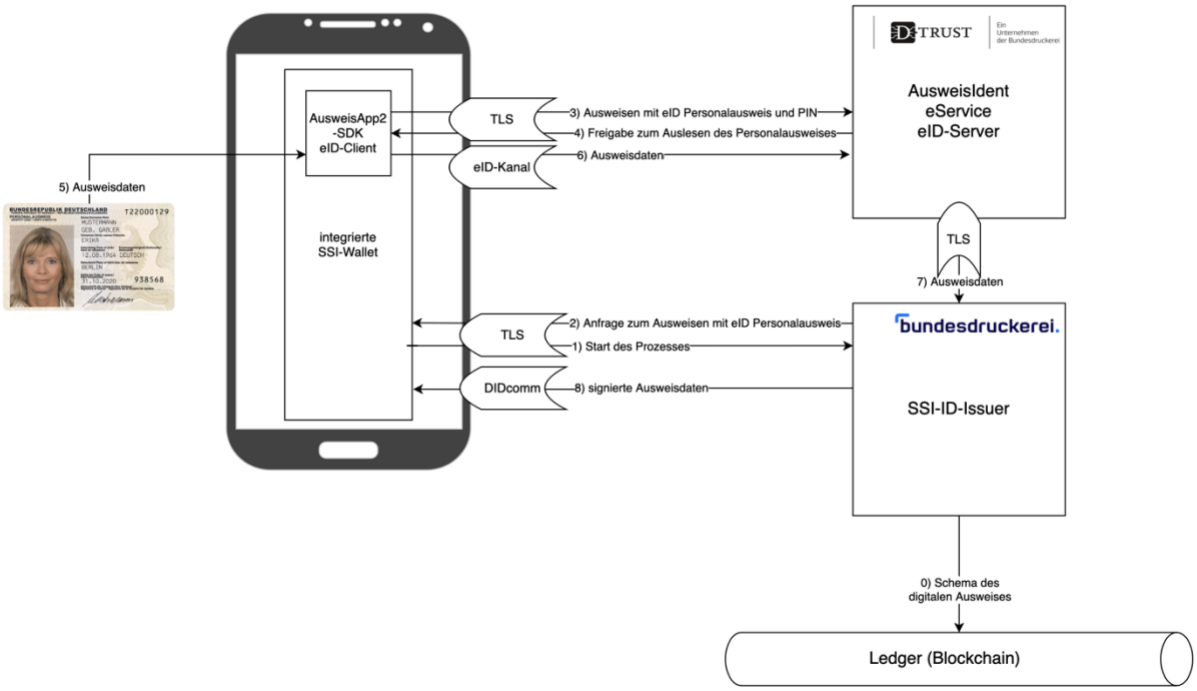


Exhibit 10: Basis ID enrollment

4.1.1 Registration

Holder initiates the process to issue the base ID within the integrated user flow in the wallet app. The wallet launches the process by calling the Bundesdruckerei's API end point in an authorized procedure (via a TLS-encrypted channel). The wallet provides authentication in the form of an API key (specific to the wallet and app version). The user confirms consent to the issuer's terms of use and data privacy statement through click in the wallet.

4.1.2 Identity verification

The user's identity is verified using the eID function of their identity card. The issuer API provides the entry point (tcTokenURL) for the eID process. In this way, the AusweisApp2 SDK integrated in the wallet creates a connection to the eID server, checks the readout authorization, and obtains the user's consent. The user authorizes the readout by entering their eID PIN in the wallet app. In the eID channel, the requested identity data are transferred from the eID card (identity card) to the eID server. Once the readout takes place, the eID server replies to the integrated eID client with the redirectURL. This contains only the

identifier for the eID session. The integrated eID client transfers the redirectURL to the wallet, which then calls up the API end point from the redirect URL. The issuer service requests the identity data from the eID server (via a TLS-encrypted channel) and the eID server releases the readout of the identity data to the issuer service. The issuer service now has the user's verified identity data.

4.1.3 Saving the identity

The issuer service (SSI-ID issuer) does not save any user data; instead, verified user attributes are stored temporarily until the credential is issued and are then deleted. The only data saved are pseudonymous data (to enable credentials to be correctly revoked; see the section on revocation for further details) and anonymous data (for the credential life cycle).

4.1.4 Issuance/delivery

Once the eID process is successfully completed, the issuer service responds to the wallet by sending a DIDComm invitation. This invitation contains the DID, cryptographic keys, and parameters for establishing a connection (DIDComm). The wallet receives these and initiates the DIDComm connection to the issuer. The issuer can generate and digitally sign the credential with the verified user attributes and send it to the wallet over the successfully established connection. The user accepts the request and the wallet stores the credential. The issuer service generates a revocation PIN and saves this data in the internal revocation database for the duration of the life cycle. The revocation PIN is securely transmitted and displayed to the user via the DIDComm channel.

The channel link used to issue the base ID is provided through the following channels (see Exhibit 10: Basis ID): TLS between the issuer service (AusweisIdent/eService) and the wallet, followed by the establishment of the eID channel between the eID document and AusweisIdent/eID server using the AusweisApp2/eID client. The base ID is issued as a verifiable credential in the DIDComm channel. Integrating the AusweisApp2 SDKs in the ID wallet app at the software level ensures a close connection between the online identity function and credential deployment.

4.1.5 Activation

The credential is available for use as soon as the holder accepts it..

4.2 Usage of base ID (example hotel check-in)

The following chapter describes the use of the base ID and company ID when checking in to a hotel.

Remark: This use case is using the base ID and the company ID in combination, but could also be used similarly just with the base ID

Ablauf:

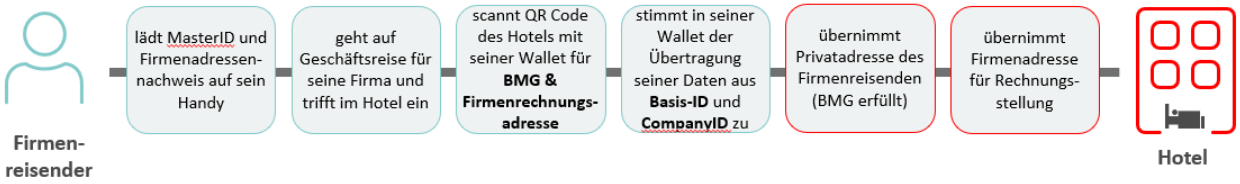


Exhibit 11: Use of the basis ID

The user reads with his wallet app (where the base ID is stored) a QR code at the hotel reception (could in other scenarios also be a deep link) and sends a query to the hotel controller (via a TLS-encrypted channel). The hotel controller enters the procedure in the MongoDB and uses the allocated agent to generate a "proof request". This request contains the end point and the public key for the hotel's agent (for subsequent communication between the wallet app and the agent), a list of the requested attributes from the base ID and company ID to be verified, and additional information (such as the timeframe for verifying non-revocation, which can be open-ended or just a millisecond). The secured, authenticated HTTPS notification channel is used to ensure that the public key is authentic. To do so, the ID wallet app receives the hotel's HTTPS end point (TLS >= 1.2; part of the invitation) and the corresponding certificate. Only certificates from CAs trusted by the mobile operating system (see available list from iOS and Android) are permitted; in other words, marked in the system as trustworthy. Requirements also specify that the base ID must conform to a particular credential definition (i.e., signed on the ledger by the Bundesdruckerei with its public key) and that the company ID must be created by one of the companies on the Indy ledger in line with the company ID scheme.

A redirect then sends the proof request back to the wallet app, which processes it. The wallet app generates a "verifiable presentation" in response to the proof request. This response contains the requested attributes from the base ID and company ID, verification that they come in each case from a VC that the Bundesdruckerei or an employer have signed, and verification that the VC in question was not recalled by the point in time at which it was requested. Unlike certificates (such as JSON-Web or X.509), the VC must not be disclosed for the verification, in combination with the signature, the origin of the values provided (this process to prove that the signature has been provided is known as a zero-knowledge proof and is implemented based on the Camenisch-Lysyanskaya[3] signature scheme). In this way, attributes that are not required, correlating information such as the signature value on the VC, and the revocation ID (in combination with the accumulator information in the blockchain) can remain confidential and do

not leave the user's wallet. In addition, the VC does not leave the wallet either and cannot be used as such by the verifier.

The steps described above in detail:

1) The user opens the app and enters their PIN, and the wallet database is decrypted
2) The user scans the QR code with the opened wallet and receives a redirectURL in response
3) The wallet calls up the redirectURL and receives a DIDcomm message (verifier's proof request and service end point)
    a. The proof request contains the verifier's requirements for which base ID attributes should be presented and a NONCE (80 Bit)
4) The wallet selects all attributes matching the proof request
5) Wallet: requests user consent (with user authentication via PIN)
6) Wallet: requests Tails file from the issuer of the base ID (only at first check-in if not already provided)
7) Wallet: requests validation information from the ledger via a ZMQ connection (scheme information, credential definition information, revocation registry information, current revocation status)
    a. Base ID scheme for information on retrievable attribute names
    b. Base ID credential definition for the issuer's public keys and the credential definition ID for limiting the proof request
    c. The revocation registry contains the Tail file URL and hash plus the initial accumulator status; revocation entries contain post-revocation deltas and updated accumulators
8) The wallet generates the zero-knowledge proof (CL)
    a. Primary proof via attribute, link secret, and NONCE
    b. Proof of non-revocation
9) Wallet: sends proof to verifier's service end point
10) Verifier: requests validation information from the ledger via a ZMQ connection (scheme information, credential definition information, revocation registry information, current revocation status)
    a. Base ID scheme for attribute names
    b. Base ID credential definition for issuer public keys and the credential definition ID for limiting the proof request

c. The revocation registry contains the Tail file URL and hash plus the initial accumulator status; revocation entries contain post-revocation deltas and updated accumulators

11) Verifier checks correctness of
    a. Signatures (incl. NONCE)
    b. Validation information
    c. Revocation status

As soon as the agent receives the response from the wallet app, it uses an NI-ZKP based on the Camenisch-Lysyanskaya signature scheme to check that the verification is correct and also notifies the hotel controller via a webhook. This then queries the data contained in the verification from the hotel agent and triggers the deletion of all information in the hotel agent that were created during the identification process. At this point the hotel controller transmits the data requested from the base ID and the company ID to the hotel backend so that the check-in process can be completed on the existing hotel system. An hotel employee compares the data; a warning is shown in the hotel front-end if any data is invalid or inconsistencies exist among the VC.

4.3 Administration

4.3.1 Updates

The basis ID can be derived in parallel more than once.

4.3.2 Revocation

The user authorizes the revocation either with their identity card (via restricted ID) or with a revocation PIN displayed during the registration process.

The revocation PIN, in the form of a hash, is kept internally for the specific base ID only at the issuer's revocation service. Via the identity card holder's restricted ID (service and card-specific code/pseudonym), the hash is also kept for the specific base ID at the issuer's revocation service. Revocation information for the specifically issued base ID credential are linked to it. When a revocation is requested, the hash is used again to authenticate the user based on the revocation PIN or restricted ID and make a match.

The issuer then checks the authorization and, using the saved revocation list and index, triggers a revocation by changing the cryptographic accumulator on the ledger. More precisely: since the issuer no longer has direct access once the credentials are issued, anonymous revocation information is kept on the decentralized infrastructure (blockchain/ledger). The issuer has authority over this information. The issuer

operates revocation lists (Tail files) for this purpose. A large number of credentials are referenced in each block list, resulting in herds anonymity.

A total of ca. 10 seconds passes between when a revocation is triggered/requested until it is implemented.

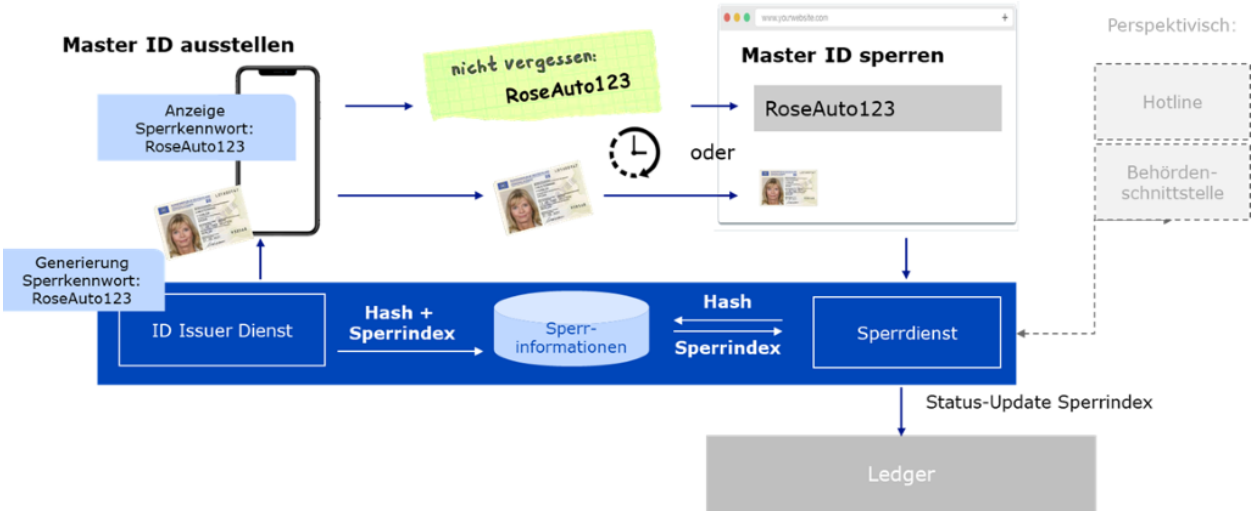


Exhibit 12: Suspending/blocking the basis ID

4.3.3 Reactivation

Reactivation is not possible. A new VC must be generated.

4.3.4 Replacement

Multiple derivation in parallel is possible, but replacement is currently not; if a derived credential is revoked all other derivations are revoked as well.

4.3.5 Deletion

Revoking a credential is equivalent to blocking it; in addition, all holder-related (pseudonymous) data are deleted.

5 Regulatory requirements

§ 29 para. 5 of the Federal Act on Registration (Bundesmeldegesetz or BMG).

6 Responsibilities

IBM Deutschland GmbH is responsible for developing the SSI-based information processing system. In addition to IBM Deutschland GmbH, the companies involved in the project are responsible for providing

and maintaining parts of the information network. The following chapter lays out the trust relationships/responsibilities for each of the planned pilot's main areas.

6.1 Deriving the basis ID

The Bundesdruckerei GmbH (along with its subsidiary D-Trust GmbH) is responsible for ensuring the classic protection objectives are met (integrity, incl. authenticity; confidentiality and availability, incl. resilience) for data, processes, and components related to the derivation of primary identities using the basis ID.

6.2 Providing the blockchain backend

The five following companies are responsible for providing and maintaining the blockchain network (Hyperledger Indy network) required for the pilot project

- BWI GmbH
- Deutsche Bahn AG
- Bundesdruckerei GmbH
- IBM Deutschland GmbH
- esatus AG

On behalf of the Federal Chancellery, IBM assumes overall responsibility for the pilot phase by orchestrating among the network's various node operators. In so doing, IBM supports efforts to define, agree on, and manifest a sustainable governance approach the ecosystem as a whole.

6.3 IT systems required at companies

Plans call for a use case in which the basis ID is supplemented by a company ID containing additional attributes, such as the name and address of the company where the holder is employed. Using the basis ID, hotels can issue registration forms as required under §§ 29, 30 of the BMG.

The planned pilot involves

- specific locations of different employers and
- specific hotels of different hotel chains.

IBM Deutschland GmbH will provide them with additional required IT components. The companies themselves will be responsible for ensuring secure operation.

6.4 Provision of the ID wallet

esatus AG is responsible for developing, maintaining, and providing the ID wallet.

7 Initial security perspective

7.1 ID wallet

The ID wallet is a central component in the identity system's ID life cycle. The wallet identifies holders in order to issue the basis ID using the AusweisApp2 SDK, stores the basis ID and other proofs, and performs verifiable presentations.

The ID wallet used here is currently available for the Android (version >= 7) and iOS (version >= 14) platforms.

7.1.1 Ensuring the integrity and authenticity of the smartphone OS and the application

At the current stage of development, the following actions are in place to ensure and check the integrity and authenticity of the smartphone OS and wallet application. These actions make it more difficult to install the wallet app on a rooted device or a simulator. Further measures for ensuring integrity and authenticity will be evaluated in subsequent development iterations.

7.1.1.1 Check of OS and app integrity

During the issuance process, the issuer of the basis ID checks that the operating system and application on the holder's end device have not been manipulated. Procedures supplied by the operating system are used for this purpose. SafetyNet (https://developer.android.com/training/safetynet) is used to perform the check on Android and DeviceCheck (https://developer.apple.com/documentation/devicecheck) is used for iOS.

To do so, the issuer sends a session-specific challenge (nonce) which, together with information about the mobile end device, is signed by the operating system's attestation server. The holder must transfer this signed information (attestation) to the issuer before the credential can be issued.

In particular, the issuer checks:

- The signature to confirm the attestation's integrity/authenticity
- Whether the attestation is current (timestamp)
- Session reference (nonce/challenge = hash for the HTTPS session ID)

- Package version and application signature key (SafetyNet)
- Result of the integrity review (SafetyNet: ctsProfileMatch, basicIntegrity)
- Evaluation type: (basic, hardware-backed by SafetyNet)

7.1.1.2 App authenticity with API key

For the Bundesdruckerei GmbH to issue a basis ID, an API key must contact the following Bundesdruckerei API end point:

- TLS-API end point for initiating the basis ID integrated flow

The API key (128 Bit UUID, specific for wallet provider and version) is forwarded from the Bundesdruckerei GmbH to esatus AG via an out-of-band channel. The process is defined as follows:

1. The Bundesdruckerei generates the UUID in its deployment environment
2. The UUID is sent to esatus through a secure transport medium (encrypted and signed e-mail).

esatus AG integrates the received API key as a constant (X-API-KEY) in the ID wallet app's source code. To help prevent unauthorized extraction, the API key is obfuscated. Authentication to the end points actually takes place via the X-API-key field in the HTTP header.

7.1.1.3 Limiting the group of participants in the hotel pilots

It is necessary to ensure that basis IDs and company IDs for use in the hotel pilot are only issued to smartphones and wallet apps managed by the participating companies (in the form of mobile device management (MDM)).

To do so, deployment mechanisms for the MDM solution in question are used to install a special version of the wallet app on the pilot participants' company-managed smartphones. This wallet app contains a version-specific API key issued exclusively for the hotel pilots (see chapter 7.1.1.2); furthermore, the Android version is signed with a special developer certificate and the iOS version has a special bundle ID. The Bundesdruckerei maintains a central list of all API keys issued for the hotel pilot and the related developer certificate hash sums and bundle IDs.

When issuing the basis ID, the Bundesdruckerei can use the API key transmitted by the wallet and the hash value for the app developer certificate (via SafetyNet attestation, see chapter 7.1.1.1) or the bundle ID (via iOS DeviceCheck, see chapter 7.1.1.1) to determine unequivocally whether the wallet is running on a managed device that is part of the hotel pilot. The Bundesdruckerei then issues a basis ID that has been cleared for use in the hotel pilot. Specifically, the following attributes are checked:

- X-API-KEY
- Android SafetyNet Attestation API:
    - ApkCertificateDigestSha256
- iOS DCAppAttestService:
    - RP ID (32 bytes)

In the course of the identity proof, the verifier (in this case, the hotel) can use the ID for the credential definition (see appendix) of the credential it receives to check whether the basis ID was issued for a managed smartphone.

If the API key or developer key material (iOS and Android) has been stolen and published by an unauthorized third party, the Bundesdruckerei blocks the issuance of new basis IDs for wallet apps that contain the stolen API key or were signed with the stolen key material. In addition, all basis IDs issued in the past for wallets with the newly blocked API key or developer certificate are revoked.

7.1.2 Secure data storage and cryptographic key management

The ID wallet uses the Hyperledger Indy SDK and its crypto libraries to implement ID system functionalities. Hyperledger Indy is an open-source solution that provides tools, libraries, and reusable components for managing self-determined digital identities. They include the Indy SDK default wallet implementation used by the ID wallet, which contains the following main functions:

- Validator interaction
- Key management (see chapter 8.1)
- Managing verifiable/anonymous credentials

The wallet saves all data for this purpose, including the link secret, in a file-based database (SQLite3) with symmetric encryption that is transparent while running (hardened version of SQLCipher[3]). The symmetric key *key_enc_data* used to secure the database (encryption and data authentication) is derived as follows for each wallet app launch:

- Pin_validation_deriv = PBKDF2(PIN | Key_enc_data_salt)
- key_enc_data = SHA256(Pin_validation_deriv | pre_key)

[3] https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/concepts/default-wallet.html?highlight=wallet

The Key_enc_data_salt and pre_key are are securely stored (encrypted and authorized) in Xamarin's proprietary SecureStorage with the help of the hardware-bound, platform-specific symmetric *key_hardware* key.

The exhibits below show the hierarchical key model, focusing on the "link secret" authentication secret that especially requires protection. For further details on the cryptographic primitives and protocols used, see chapter 8.1.

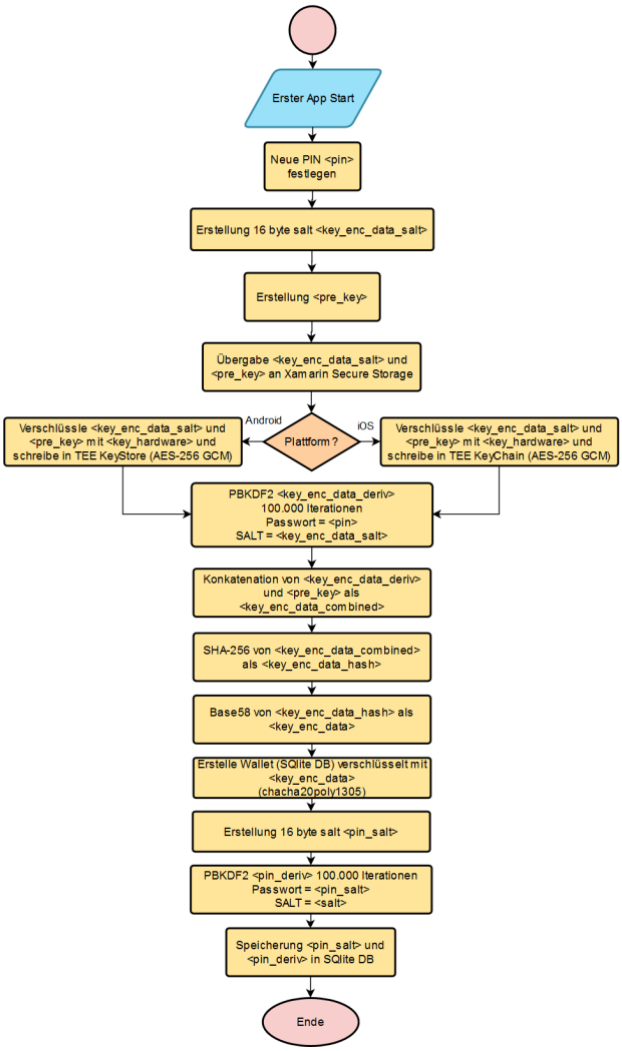


Exhibit 13: Link secret encryption

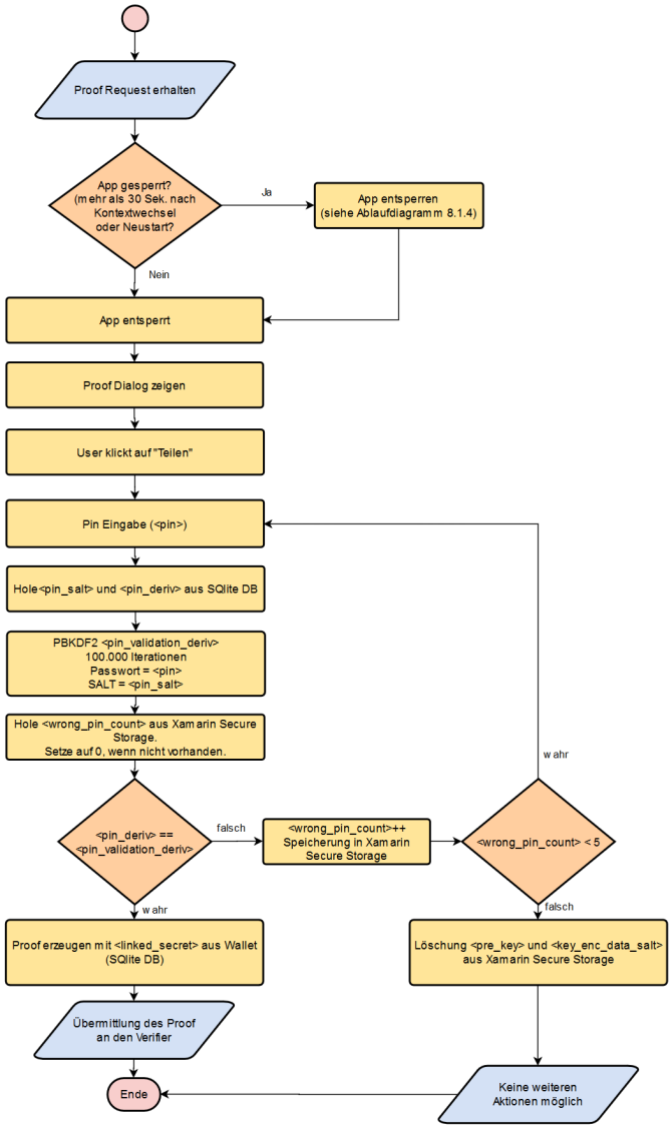


Exhibit 14: Link secret decryption

To be used, the link secret must be decrypted and, as a result, is held in potentially unsecure memory during that time. The following measures help to prevent copying of the link secret:

1. During the pilot only smartphones managed by participating companies in a mobile device management approach are used. In addition, platform-specific integrity checks (see chapter 7.1.1.1) are used to rule out that the devices are rooted or the app has been manipulated.
2. Weak point management is employed during the pilot to detect security gaps on the smartphones used, asses them, and eliminate them based on their severity.

These measures reduce the risk that external attackers could read out the link secret. To prevent smartphone holders from copying the link secret, pilot participants are also informed that they are not permitted to read out or share it.

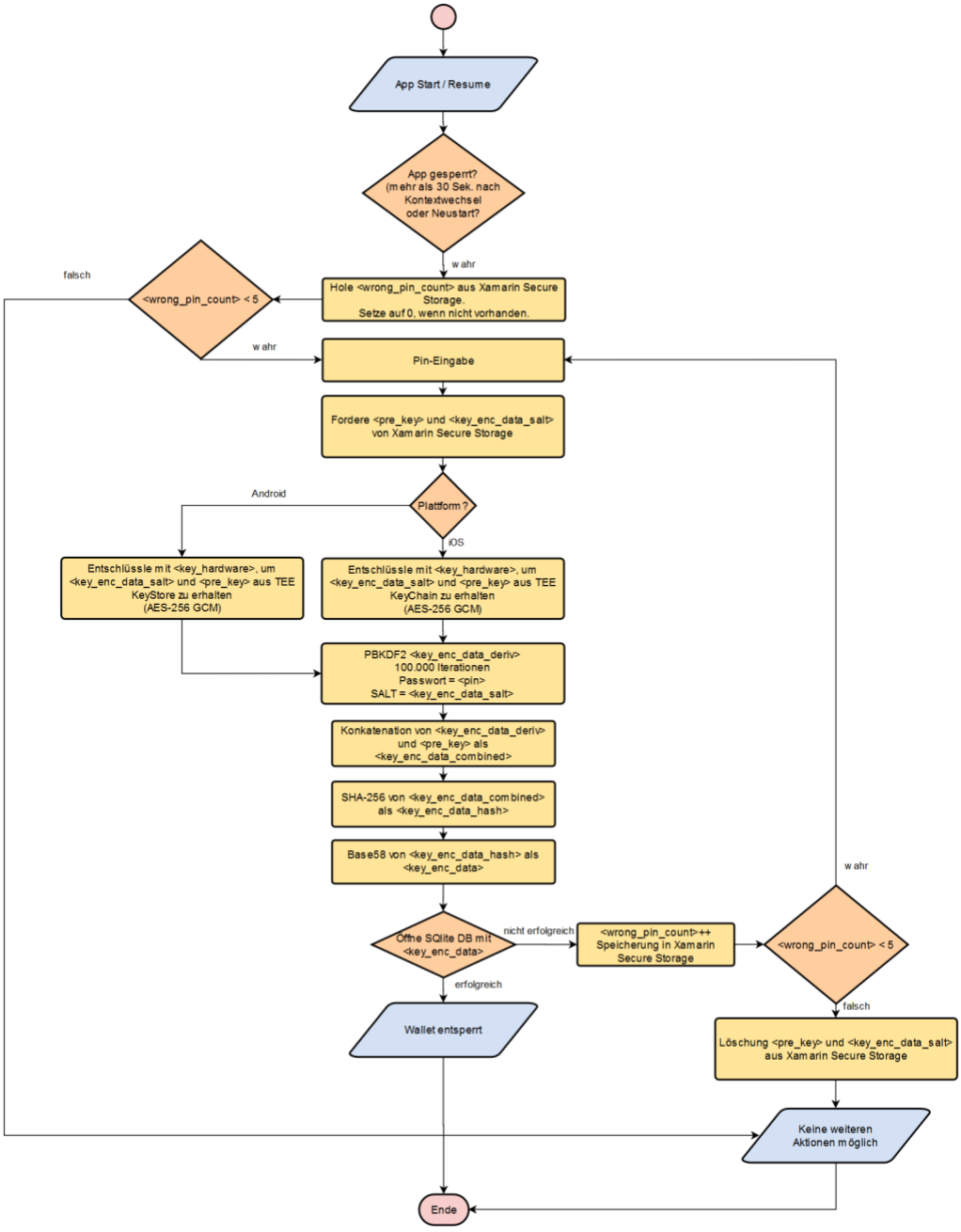
7.1.2.1 Wallet access control

A 6-digit PIN protects the wallet from unauthorized access. The holder sets this PIN when the wallet is launched and can change it later using the same process and the previous PIN. This PIN must be entered when the wallet app is launched and 30 seconds after it is pushed into the background (e.g., when switching the context to another app). The PIN is checked using the key derivation process described above for the SQLite database. If the PIN is not entered correctly, the SQLite database is not decrypted and the wallet app cannot be launched. Holders enter the PIN using a customized version of XamarinFormsPinView[4].

If the PIN is entered incorrectly five times, the pre-key is deleted and the SQLite database can no longer be decrypted. The wrong PIN counter is incorporated into the software.

The following exhibit shows the access control workflow with the characteristics at each point.

[4] https://github.com/lassana/XamarinFormsPinView



7.1.2.2 Holder authorization

To authenticate the holder, the same PIN used when the app is launched must also be entered before each identity proof. The program sequence forces this PIN request, which is not connected to decryption of the SQLite database or the link secret.

To realize this, the initially set holder PIN is derived via PBKDF2 and a randomly selected Salt and saved with the Salt in the encrypted SQLite database:

1. Pin_deriv = PBKDF2(PIN | Pin_salt)
2. Save Pin_deriv and Pin_salt in SQLiteDB

Here Pin_salt ≠ Key_enc_data_salt

During holder authorization, the PIN entered by the holder is derived again and compared against the value saved in the SQLite database:

1. DerivedInput = PBKDF2(PIN_INPUT | Pin_salt)
2. DervidedInput =? Pin_deriv

If the PIN is entered incorrectly five times, the pre-key is deleted and the SQLite database can no longer be decrypted. The wrong PIN counter is incorporated into the software.

7.1.2.3 Limitations during testing/pilot

At the current stage of development, no mechanisms have been implemented to check the security of the biometric verification mechanisms installed in the mobile devices or their metrics (e.g., FAR, antiviolation security) For this reason, such biometric verification mechanisms have been deactivated for the hotel pilot's test phase.

7.1.3 Secure channel bundling when issuing the basis ID

Integrating the AusweisApp2 SDKs in the ID wallet app at the software level ensures channel bundling between the online identity function and credential issuance. The wallet initiates the process of issuing the basis ID. It builds a TLS channel to the SSI issuer's API end point which, in turn is also protected with certificate pinning (public key; see chapter 8.1.3.2). The SSI issuer responds with the session-linked tcTokenURL that points to the AusweisIdent service. Through this query, the online authentication process is triggered by the AusweisIdent service, which acts as both the eID server and the eService in the eID context. The tcTokenURL is transferred from the wallet to the integrated AusweisApp2 SDK and responds with the tcToken, establishing the start parameters for the eID process. Using the tcToken, the eID client/AusweisApp2 SDK provides a secure eID channel between the eID document and the eID server. After the authorization certificate is verified and the holder authorizes the process to read out the eID by entering the PIN, and eID channel is established and the eID attributes in the eID channel are transmitted to the eID server. Then the eID server answers by sending the redirect URL to the eID client/AusweisApp2 SDK, which is transferred to the wallet and called up. The redirectURL first points to the eService (AusweisIdent) and, when called up, is forwarded to a secure API end point (TLS) for the SSI issuer (a new TLS channel is established and the sessions are matched using session cookies saved in the server's volatile

memory). The issuer's eService (AusweisIdent) now retrieves the eID data from the eID server. The SSI issuer retrieves the identity data from the eService using the AusweisIdent interfaces (OpenID Connect). AusweisIdent is the OpenID provider here and the SSI issuer is the relying party. Now the issuer answers the wallet over the secured API end point (TLS) by sending the DIDComm invitation, which is generated by the issuer's SSI agent. A DIDComm invitation can only be used once. With this DIDComm invitation, the wallet can now establish a secure DIDComm channel to the issuer's SSI agent. At this point the credential with the eID attributes is issued through this DIDComm channel.

In summary, channel bundling is achieved as the wallet establishes two TLS channels with the issuer (linked over a session ID) and uses them to initiate the eID channel and the DIDComm channel, which are both managed by the integrated wallet.

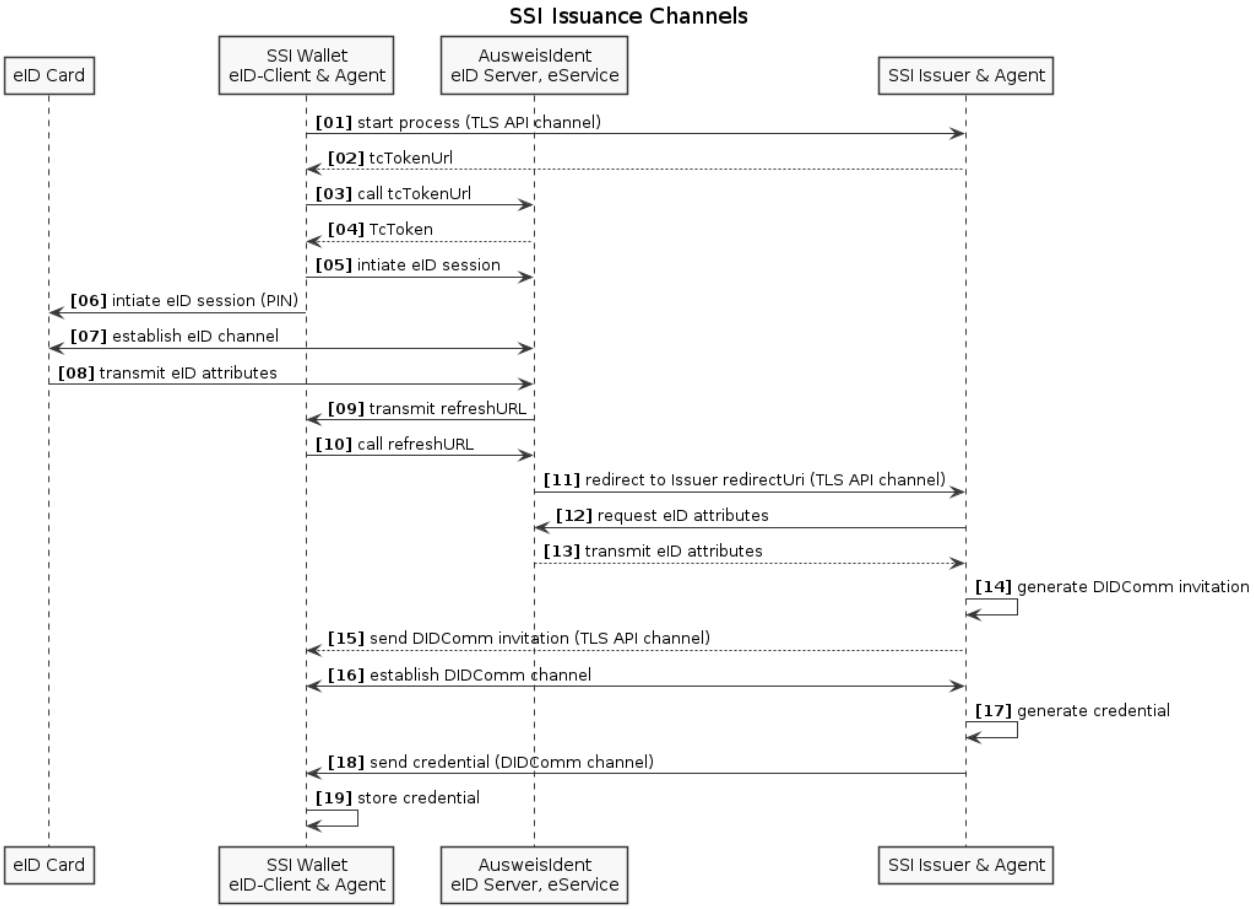


Exhibit 15: Secure channel bundling when issuing the basis ID

7.1.4 ID wallet analytics and private sphere

No data are collected for analysis.

7.1.5 Frameworks and libraries used

The ID wallet app was implemented with the Xamarin cross-platform framework.

The following frameworks are used as well:

- Hyperledger Aries
- Hyperledger Indy
- Hyperledger Ursa
- AusweisApp2 SDK

Standards used

Standard	Reference	Description
DID	https://w3c.github.io/did-core/	Decentralized Identifiers are a standardized identification mechanism that enables decentralized, verifiable digital identities.
DIDComm	https://identity.foundation/didcomm-messaging/spec/	DIDComm is a standardized DID-based communication channel that enables the secure sharing of credentials, proofs, and other messages.
Aries RFC 0160: connection protocol	https://github.com/hyperledger/aries-rfcs/tree/master/features/0160-connection-protocol	This AriesRFC describes a protocol for establishing a DIDComm connection between SSI agents.
Aries RFC 0036: Issue Credential Protocol 1.0	https://github.com/hyperledger/aries-rfcs/tree/master/features/0036-issue-credential	This AriesRFC describes a protocol for issuing a credential (based on Hyperledger Indy) over a DIDComm channel.

Standard	Reference	Description
Aries RFC 0037: Present Proof Protocol 1.0	https://github.com/hyperledger/aries-rfcs/blob/master/features/0037-present-proof/README.md	This AriesRFC describes a protocol for checking/querying a credential (based on Hyperledger Indy) over a DIDComm channel.

Table 1: Standards used

7.1.6 Penetration tests and audits

The Hyperledger ecosystem undergoes regular security audits and penetration tests and makes the results available to the public.[5]

No further results of security audits or penetration tests are available for the planned pilot.

7.2 Weak point and device management

A weak point/vulnerability management system is a central component in managing IT system security. All system components are proactively monitored for known weak points in an ongoing process. In the ID system here, both the DLT network infrastructure and the corresponding software components and mobile end devices must be monitored.

7.2.1 Infrastructure

In the pilot, the infrastructure components for the overall system are operated by a limited, clearly defined group of participants (see chapter 3.1). The participants operate the infrastructure components in the context of their IT network. As a result, existing concepts such as ITSMS, weak point management, etc., also apply to the infrastructure components of the concept described here.

If the group of system participants is expanded in a later iteration, the situation will be evaluated again and separate security management concepts will be developed if necessary.

7.2.2 Smartphone

Assessing the resilience of a mobile end device requires detailed information on its (hardware- and software-based) mechanisms for security and protection of the private sphere. Known weak points of a security component also flow into assessments of resilience. To ensure that the ID solution on the mobile

[5] https://wiki.hyperledger.org/display/SEC/Project+Audits

end device is secure throughout its service life, the information compiled on devices and weak points must be continuously updated and monitored.

In the Android ecosystem, which is highly heterogeneous, the required information cannot be retrieved from a central collection point as it can for, e.g., Apple iOS. As a result, it is necessary to analyze sources such as manufacturer websites, blogs, certification portals, and databases on weak points and provide the findings in a central database.

At the solution's current stage of development, just a small group of selected individuals from the participating companies are taking part with their mobile end devices. Since most of these end devices are supplied by employers and already subject to mobile device management, measures to manage weak points are already taking place.

If additional individuals take part in the system with their mobile end devices at a later stage of development, a corresponding system for managing weak points will be implemented. An implementation concept has been prepared.

8 Cryptographic process

8.1 Wallet

8.1.1 Cryptographic key material

8.1.1.1 key_hardware

- **Description:** Encrypts and authenticates key_enc_data in Xamarin SecureStorage
- **Key schema:**
    - Android: AES-256 GCM
        - Device with TEE/SE: hardware-backed key
        - Device without TEE/SE: software-backed key (a few Android 6 devices; app limited to Android 7+ for go-live)
    - IOS: AES-256 GCM (encrypted as a Secure Enclave with an ECC key)
- **Saved in: native, device-specific key memory**
    - Android: KeyStore
    - iOS: KeyChain
- **Generated by:** Wallet app

- **Generated how:** Via Android KeyStore API / Apple iOS KeyChain
- **Generated when:** At initial launch of wallet app

8.1.1.2 key_enc_data

- **Description:** Encrypts and authenticates the SQLite database
- **Key schema:** CHACHA20-POLY1305
- **Stored in:** memory (while running)
- **Tied to:** ID wallet app + PIN
- **Generated by:** Wallet app
- **Generated how:** Base58(SHA256(Pin_validation_deriv | pre_key))
- **Generated when:** Every time wallet is launched or resumed from the background

8.1.2 Cryptographic safeguards for data requiring protection

8.1.2.1 Connections, credentials, proof history, link secret, basis ID blocking PIN

- **Saved in:** SQLite database in app-internal memory
- **Encrypted and authenticated with:** key_enc_data
- **Algorithm:** CHACHA20-POLY1305
- **Generated by:** Wallet app (Indy SDK)
- **Generated how:** Link secret: Indy SDK ProverCreateMasterSecret function; rest: Indy SDK AddRecord function
- **Generated when:** Link secret: when wallet is generated; rest: when relevant messages/data are received

8.1.2.2 Key_enc_data_salt

- **Data type**: byte[]
- **Length:** 16 Byte
- **Saved in:** Xamarin SecureStorage
- **Encrypted and authenticated with:** key_hardware
- **Algorithm:** AES-256 GCM
- **Generated by:** Wallet app
- **Generated how:** RNGCryptoServiceProvider
- (https://docs.microsoft.com/de-de/dotnet/api/system.security.cryptography.rngcryptoserviceprovider?view=netstandard-2.1)

- **Generated when:** At initial wallet setup or when a new PIN is set

8.1.2.3 Pin_salt

- **Data type**: byte[]
- **Length:** 16 Byte
- **Saved in:** SQLite database in app-internal memory
- **Encrypted and authenticated with:** key_enc_data
- **Algorithm:** CHACHA20-POLY1305
- **Generated by:** Wallet app
- **Generated how:** RNGCryptoServiceProvider
- (https://docs.microsoft.com/de-de/dotnet/api/system.security.cryptography.rngcryptoserviceprovider?view=netstandard-2.1)
- **Generated when:** At initial wallet setup or when a new PIN is set

8.1.2.4 Pin_deriv

- **Data type**: byte[]
- **Length:** 16 Byte
- **Saved in:** SQLite database in app-internal memory
- **Encrypted and authenticated with:** key_enc_data
- **Algorithm:** CHACHA20-POLY1305
- **Generated by:** Wallet app
- **Generated how:** PBKDF2(PIN + Pin_salt) with 100,000 iterations
- **Generated when:** At initial wallet setup or when a new PIN is set

8.1.2.5 Pin_validation_deriv

- **Data type**: byte[]
- **Length:** 16 Byte
- **Saved in:** Memory (while running)
- **Encrypted with:** Not encrypted
- **Generated by:** Wallet app
- **Generated how:** PBKDF2(PIN + Pin_salt) with 100,000 iterations
- **Generated when:** A proof is sent (auth. process)

8.1.2.6 Pre_key

- **Data type**: base58 encoded string (44 digits)
- **Length:** 32 Byte
- **Saved in:** Xamarin SecureStorage
- **Encrypted and authenticated with:** key_hardware
- **Algorithm:** AES-256 GCM
- **Generated by:** Wallet app
- **Generated how:** Indy SDK GenerateWalletKey function
- **Generated when:** At initial wallet setup or when a new PIN is set

8.1.3 Cryptographic procedures in communication

8.1.3.1 HTTP(S)_DEFAULT

- **Description:** For DIDComm communication, DID connection image retrieval, Tails file retrieval, HTTP redirect resolution and HTTP communication for connectionless proofs, connectionless credentials, and connection invitations
- **Connection type:**
    - Android: TLS/SSL (Android HTTP client: min. TLS 1.2) + non-TLS/SSL (HTTP)*
    - iOS: TLS/SSL (default NSUrlSession (iOS 7+), TLS 1.2 support) + non-TLS/SSL (HTTP)*
- * Hyperledger Aries agents can also be used without TLS/SSL. However, every relevant communication will also be encrypted via PAYLOAD_ENCRYPTION (DIDComm) (see chapter 8.2.1).

8.1.3.2 TLS_BDR_API

- **Description:** for API communication (both to initially establish eID and to establish a second channel afterwards to retrieve the DIDComm invitation)
- **Connection type:** TLS/SSL (TLS 1.2 only)
- **Permitted cipher suites:**
    - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
    - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- **Other information:** Public key pinning
    - Exponent: 65537

- Modulus:
  D4:92:74:22:D1:E1:15:72:01:9D:3E:1B:3F:AA:2A:56:D7:D2:A1:1D:2C:91:73:5B:7C:F6:38:7E:F8:F6:98:41:25:1B:AF:03:0C:54:9A:44:3A:FB:07:F0:AC:64:E4:AA:71:DD:CE:0F:D4:E5:19:2C:DC:B9:A3:50:26:26:24:BB:A0:61:10:02:25:E1:01:1F:CF:31:1C:BB:4E:1F:8A:7A:0B:56:6C:46:89:C8:65:33:C5:E0:D3:1C:73:0F:F8:5C:53:2E:49:79:00:9A:44:71:DE:96:EF:0E:D6:BA:C8:08:07:7E:11:64:0A:E4:BF:4B:18:59:A7:6F:01:83:11:B3:6E:DB:B9:6E:07:71:57:C6:A2:BB:F3:60:E7:46:9A:D3:88:6F:A4:2C:3E:D9:8B:63:36:6A:55:73:96:99:45:BD:61:06:32:B3:35:38:D6:D0:3B:27:E3:B5:24:11:56:A3:09:9A:2D:93:75:46:09:E0:F2:7E:F9:C8:36:69:29:57:62:DF:05:B0:CC:C2:45:32:AE:14:5A:36:5A:C2:D5:2D:7D:9E:E1:2A:F3:E7:57:77:85:FA:12:20:14:FF:40:9A:DC:A8:96:CE:5E:F5:62:21:58:06:4A:6F:F2:A7:5C:19:61:BF:74:F4:BF:DE:C2:3D:CB:67:EB:5E:C9:22:D3:FD:36:4F:62:F2:1B:A3:B3:42:EC:9A:25:63:97:63:FA:D5:38:E6:9C:68:7A:E1:39:E0:7B:8E:18:79:67:1B:3D:4C:4E:CF:3F:22:85:87:A3:92:07:86:99:B3:CF:64:C5:59:F0:3B:66:B2:90:30:51:19:E6:96:D0:A4:3E:81:2B:A4:2A:7C:EA:59:10:C0:5B:AC:4B:93:41:2A:1E:53:BF:F3:25:5E:5D:0B:F8:95:53:B9:D9:60:21:DC:3C:FD:3F:35:7B:24:8C:94:7E:A6:5A:8F:FD:3B:34:CA:B0:69:60:BD:5C:73:25:BE:D8:44:B6:A3:49:E0:A5:FC:7B:C9:87:68:C6:86:78:28:E0:FC:CB:A8:E9:1A:AC:55:8A:3A:08:86:83:A9:41:2F:91:1C:F0:92:95:98:43:74:DB:47:2F:83:61:E6:39:AE:8D:01:07:E5:7E:2F:EE:B2:79:65:04:8B:82:58:2D:49:CF:44:06:64:1E:5C:5C:95:33:DF:0D:CF:2A:B3:B9:EC:8F:A4:4B:8F:69:32:29:9A:15:A7:64:6F:AE:06:DD:84:FD:CB:EF:F6:4A:12:8B:E4:5B:C6:F4:11:8E:16:95:FF:C0:D9:4D:C9:9D:2A:3F:04:EF:D5:AD:BC:78:30:BA:16:98:A2:5A:A1:AD:88:E4:54:A8:C2:B2:03

8.2 DLT – communication

8.2.1 Communication agent - agent

- **Description:** Communication between the agents for the holder wallet and for the issuer/verifier (DIDComm communication, connectionless proofs, connectionless credentials, connection invitations)
- **Connection type:** Asynchronous messages via DIDComm
- **Other information:**
  - Key agreement: X25519 (ECDH)
  - Encryption: XChaCha20
  - Data authentication: Poly1305

8.2.2 Communication with blockchain and agents

- **Description:** Communication between network nodes and SSI agents (issuer, holder, or verifier)
- **Connection type:** CurveZMQ
- **Other information:**
    - Key agreement: X25519 (ECDH)
    - Encryption: XSalsa20
    - Data authentication: Poly1305

8.2.3 RBFT consensus algorithm

- **Description:** Sharing transactions to be persisted and Merkle tree root hash multisignature
- **Algorithm used:** BLS signature scheme on the elliptic curve, Boneh-Lynn-Shacham BLS12-381; SHA-256 for Merkle tree
- **Key length:** 381 Bit

8.2.4 Node authenticity

- **Description:** Node authenticity initially via genesis file, then from ledger-internal pool database
- **Algorithm used:** Ed25519
- **Key length:** 256bit

8.3 Verifiable/anonymous credentials

There are currently three main types of credentials in the SSI context:

- JSON-LD verifiable credential with linked data proofs
- JWT verifiable credential with JSON web signature
- ZKP anonymous credential with Camenisch-Lysyanskaya signature

A normal verifiable credential is an immutable signed proof that most closely corresponds to a X509 certificate. Conversely, anonymous credentials offer selective disclosure and verification of predicates (e.g., older than 18) with zero-knowledge proofs to prevent simple correlation with holder data. The Hyperledger Indy/Aries project uses anonymous credentials, which currently offer the best private sphere characteristics.

8.4 Security of the cryptographic protocols used

BSI-TR-02102-2 recommends the TLS cipher suites used in the project. The following summarizes the additional cryptographic primitives and protocols used in the project:

- Payload encryption: AES-GCM, AES-CBC, XSalsa20, XChaCha20, ChaCha20
- Data authentication: HMAC-Sha-256, Poly1305
- Key agreement (ECDH): X25519
- Digital signatures: Ed25519, BLS signatures, CL signatures
- Cryptographic protocols for secure messaging: TLS, CurveZMQ
- Zero-knowledge proof: AnonCred

BSI-TR-02102-1 recommends AES primitives with GCM and CBC operating modes and HMAC-Sha-256. It also recommends the TLS cipher suites used. The following security statements apply to the other algorithms used:

- The XSalsa20, XChaCha20, and ChaCha20 stream ciphers are used in combination with MAC Poly1305 as authenticated encryption with associated data (AEAD) as described in RFC 7539, section 2.8.
- X25519: Diffie-Helmann elliptic curve based on Curve25519 elliptic curve
- Ed25519: EdDSA based on Curve25519 elliptic curve
- Curve25519: standardized by the IETF as RFC 7748 and used frequently
- Boneh–Lynn–Shacham signatures (BLS): The procedure has been proven secure in the random oracle model assuming the difficulty of the computational Diffie-Hellman problem in gap Diffie-Hellman groups, see [7]. The procedure implemented in the project uses BLS12-381 as described in the IETF draft https://tools.ietf.org/html/draft-irtf-cfrg-bls-signature-04, chapter 4.2. This procedure has also been used in Version 2 of the Ethereum blockchain (and elsewhere) since 2020.
- Camenisch-Lysyanskaya signatures: The procedure has been proven secure (under the strong RSA assumption), see [8]. The procedure implemented in the project uses a key length of 2048 Bit.
- AnonCred: A non-interactive zero-knowledge-proof procedure based on Camenisch-Lysyanskaya signatures. Implementation as described in https://github.com/hyperledger-archives/indy-crypto/blob/master/libindy-crypto/docs/AnonCred.pdf is used in the project.
- CurveZMQ: a protocol for secure messaging, see https://rfc.zeromq.org/spec/26/.

9 Cryptoagility

Since the planned activities involve a pilot project for a limited period, implementing cryptoagility is not necessary.

10 Appendix

10.1 Domain transaction genesis file


```
{"reqSignature":{},"txn":{"data":{"alias":"Bundesdruckerei","dest":"PuJU8cnSUrsv2D1PVXJZPV","role":"0","verkey":"~9vNmWu7ftS7ANCvRgGUzN2"},"metadata":{},"type":"1"},"txnMetadata":{"seqNo":1},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"alias":"BWI01","dest":"EXbps5u4eRdkjVcBLeHZgJ","role":"0","verkey":"~15iYdG6ooiXaHfMymztUU9"},"metadata":{},"type":"1"},"txnMetadata":{"seqNo":2},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"alias":"DeutscheBahn","dest":"5uW7EoVwg5PFYL3gbJJbMp","role":"0","verkey":"~BrmD27D8fsjSrJbYSG9b7n"},"metadata":{},"type":"1"},"txnMetadata":{"seqNo":3},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"alias":"esatus_AG","dest":"PRTiNcvTB2wUP2cdtByE2K","role":"0","verkey":"~PrJG1rspfax58T3Fp2A7HM"},"metadata":{},"type":"1"},"txnMetadata":{"seqNo":4},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"alias":"IBM","dest":"QutDhvFxx3DeoZyFzLbEiY","role":"0","verkey":"~JF9J9QyrajooTAaGoJEiLX"},"metadata":{},"type":"1"},"txnMetadata":{"seqNo":5},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"dest":"KYMfYbRgG7Agc8upYvngu9","role":"2","verkey":"~45MfzQeDjRssg6x5LsCJFv"},"metadata":{"from":"QutDhvFxx3DeoZyFzLbEiY"},"type":"1"},"txnMetadata":{"seqNo":6},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"dest":"7vyyugPwC3ArWtRWbz6LCm","role":"2","verkey":"~7bquxp98wiioicxCUiaC5z"},"metadata":{"from":"QutDhvFxx3DeoZyFzLbEiY"},"type":"1"},"txnMetadata":{"seqNo":7},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"dest":"VkVqDPzeDCQe31H3RsMzbf","role":"2","verkey":"~7YpG5tgnCVkBGByQPcj4Gg"},"metadata":{"from":"QutDhvFxx3DeoZyFzLbEiY"},"type":"1"},"txnMetadata":{"seqNo":8},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"dest":"75aDKXUfnvr4x1FdLrnxCX","role":"2","verkey":"~QxvtqDLEaNkNNvk5X4pXgj"},"metadata":{"from":"QutDhvFxx3DeoZyFzLbEiY"},"type":"1"},"txnMetadata":{"seqNo":9},"ver":"1"}

{"reqSignature":{},"txn":{"data":{"dest":"MqD7S5sDAiVnwycP4tdr7","role":"2","verkey":"~Aa9EGid3KQT94N1i1Fe4JL"},"metadata":{"from":"QutDhvFxx3DeoZyFzLbEiY"},"type":"1"},"txnMetadata":{"seqNo":10},"ver":"1"}
```

10.2 Pool transaction genesis

{"reqSignature":{},"txn":{"data":{"data":{"alias":"Bundesdruckerei","blskey":"2Vw6Zes8agmHzVUZhgH
7jhBDkucokvxNQKxbCi6NvJ3TF2hDbqS5satdtDqyqGJcvwhkqB2UFKYvA1EiTXYhdpSf8MEkRWNMwVrFs7UvhjfZwbow2meB
C5nQqAhHDwspyx21J8mU2YF6QtxfbqDDb6wDwVk4atU5Go3Lr1e21eH5is4","blskey_pop":"Qwn4vN7Nx647FeHH3LQUeP
EAEcq3fsWTrcwHqY98wC5jDJxTE83AJ9XFXW5rpddZxjq2fASPHULp11y7biaWobbFVmnjDfB37sxkhJFYrZeu6wQi343Tr7F
GNms6VwiGARRa1xnXwzS93W4rR9JJfj9k8phDkqEQ48PeYUf2p9qz2b","client_ip":"193.28.64.40","client_port"
:"9702","node_ip":"193.28.64.40","node_port":"9701","services":["VALIDATOR"]},"dest":"DF9XCzFED48
jB6Vh9WF1dV6Vxv6MMGFjpBMWGFQkGyi6"},"metadata":{"from":"KYMfYbRgG7Agc8upYvngu9"},"type":"0"},"txn
Metadata":{"seqNo":1,"txnId":"8406be5a8bde1145f642a9955a80475c7256a9957b3a984c74ca6a8484f9c0f7"},
"ver":"1"}

{"reqSignature":{},"txn":{"data":{"data":{"alias":"BWI01","blskey":"3pxvAzRJ8cAhfeEZusPKE8JL7YttD
2kC4yyvK9KME6PUe45Kk6PZ5VMStS1M8Nrvi2jYa2fF14VZq3SVJXMnxEfqRQe3K4FLYynDUqGT8hUEHXACt7DTG9HjZNQ97Y
52LdhT5ip4qa3dfFtvjAZfoiDZ1chBLWBb8XF5P7J7pnoDXqb","blskey_pop":"RDMg1Ttu2Q7bzM26VNggrmi79ugPoprf
xmgpLPVsvLywj1mby3xyC81QokB14SXVmHiRZx9tox6piiRgjPJy3tnV1HbuVWE3qd5zULXyFjVhzZSCjPVa92Sd28Q7biPEt
ocpYfkBag2VFm9m5PoSoKyfzQn9FGPv1dXRJb8kjzGdaF","client_ip":"217.86.140.172","client_port":"9702",
"node_ip":"217.86.140.172","node_port":"9701","services":["VALIDATOR"]},"dest":"6kh5GT8dwreXVFfpr
5YnJzoCcX3ePvfZojkqgM4WBhpi"},"metadata":{"from":"7vyyugPwC3ArWtRWbz6LCm"},"type":"0"},"txnMetada
ta":{"seqNo":2,"txnId":"537e9bbcf1633189f7f5c67dbebf8278c900f89c7b02e33833840c813c004e88"},"ver":
"1"}

{"reqSignature":{},"txn":{"data":{"data":{"alias":"DeutscheBahn","blskey":"2kh3jfpje96bkLnJM6uNmJ
t5p7WaCf1qnA9pyk6KaU4nWbv6N2B8SxdhtqaJDni9z78PGbhJHC7w3UgcT419tbw3vV4MUwWo11ZoKQAeJZ7nuwjoz2qpiJ8
AadRkiSR4WjckxQjSvDegDW7HoiiVif1qd6ZX8KdDofhysGpRwijihZE","blskey_pop":"RbM2bE268JCTHSPQyDaZCNGMf
LVtAwYUjn6creM7cTYUqkSpTiodhoxkBtEjEsQEmVwfLGRXGLqHt5sC4KXMUQpj512tq1T9GRG5gL1K4guwHKULBXLiZy8fu7
KPUuxSUkZ1H1m3uXsAM3rf4x1fXkCNf4NzqktnSjWZvnHb4xZY4f","client_ip":"81.200.193.38","client_port":"
9702","node_ip":"81.200.193.38","node_port":"9701","services":["VALIDATOR"]},"dest":"8RRkFDk3665p
QKtAsLnySA2Kn3ia6R2cvgSq5S2QLzzq"},"metadata":{"from":"VkVqDPzeDCQe31H3RsMzbf"},"type":"0"},"txnM
etadata":{"seqNo":3,"txnId":"58b0a0cb860a20b41082d4d1b2419867156ca7aea3a0bdf2bdb318a270a6a367"},"
ver":"1"}

{"reqSignature":{},"txn":{"data":{"data":{"alias":"esatus_AG","blskey":"32KRjr28fHuBM6ShvW1pWT1c6
Qn9G45g5828D326HdUE3B3Wgne9c2RLwA27NZwUGT8zu4HxpYb2FAXAQiWZQCW8x2qXfSP8w4Kw8QFNdd9L2oewkxecEdVtz6
htFwUydWsjCjfqWmT8ySS7MiDpunFTB72Bgm2Mkw8wuWydYf5tMrD","blskey_pop":"QmVstDJUFYMXFTpFE4ibYtoL9wr2
YrYQKgJKJonVCLtj2vEnv2k8RzWRZiCxcfaRp1nATMmVPwnAwBfV72NwnAk7FBJc8Bfvd3ma68KHM5qe8QJ1UK2qCbbHoPJzx
bSs7Fty5UyGqCveqNv7s7rtfzXUkgL68Fgo78r57KDSumsQBq","client_ip":"194.110.133.220","client_port":"9
702","node_ip":"194.110.133.220","node_port":"9701","services":["VALIDATOR"]},"dest":"4tQECdEPFKD
DpVGtyCK7WdtFMkDeXSk279ppHtm346qU"},"metadata":{"from":"75aDKXUfnvr4x1FdLrnxCX"},"type":"0"},"txn
Metadata":{"seqNo":4,"txnId":"b26a8b1ec76a165330faf13f0b9565f8167211c37d83845226b7986dd0f624e3"},
"ver":"1"}

{"reqSignature":{},"txn":{"data":{"data":{"alias":"IBM","blskey":"3rEFErCAavZBc4ScFh7CLgDKhSPGD1R
PVhMs27jawz6WhLSRYmQkiZTTasHxCK6crzg8VpGGiAUp8Wfi7aJ3LL6qQahn49uwaNsfMiVff424bTisubUvtqpurXf8x63d
4aPRBXofPWz6HyHiKtL2VSnP8zXz8N13Hg84F1MdRYBvWNk","blskey_pop":"RA1HS1pmpCcePde8KsWv8SekXKDyihySig
AzVjUupQvbs9LkQSGquLawwFwd1CJogM3Pwej4RxBtHf1EpS51wJ8Jffc6MSAaBEQUtQGusupSdYFZDrP2i6BuTVAM39LqPKj
twRm7YtcM1uZq8fGt2JUFWJakNVUDmdD9b6RMPtxYAK","client_ip":"158.177.43.116","client_port":"9702","n
ode_ip":"158.177.43.116","node_port":"9701","services":["VALIDATOR"]},"dest":"EbbPNsuV4ghkLb2HYLa
2de8QprnScy9PSJxKr1r7V2C2"},"metadata":{"from":"MqD7S5sDAiVnwycP4tdr7"},"type":"0"},"txnMetadata"

:{"seqNo":5,"txnId":"973a4a818d799636d44525dfac347ef440bc67f5e33c404d7824a784465abead"},"ver":"1"
}

10.3 Credential definition

https://eesdi.esatus.com/tx/EESDI-Pilot-Ledger/domain/204

Credential definition ID: Vq2C7Wfc44Q1cSroPuXaw2:3:CL:126:Basis-ID

{
 "txn": {
  "data": {
   "refSchemaAttributes": [
    "documentType",
    "addressCountry",
    "firstName",
    "addressCity",
    "addressStreet",
    "birthName",
    "dateOfBirth",
    "academicTitle",
    "dateOfExpiry",
    "nationality",
    "familyName",
    "placeOfBirth",
    "addressZipCode"
   ],
   "refSchemaFrom": "5PmwwGsFhq8NDiRCyqjNXy",
   "refSchemaId": "5PmwwGsFhq8NDiRCyqjNXy:2:Basis-ID:1.0",
   "refSchemaName": "Basis-ID",
   "refSchemaTxnSeqno": 126,
   "refSchemaTxnTime": "2021-03-24T13:33:37.000Z",
   "refSchemaVersion": "1.0"

```
  },
  "metadata": {
    "digest": "11f48ad70865cd80e68b6269d80926d1aa51e93a6aa77f12a690784eb5668eb4",
    "from": "Vq2C7Wfc44Q1cSroPuXaw2",
    "payloadDigest": "84262de0361a7b643e1dc15198bf245bff619ea4b433e92c9370002ede84627f",
    "reqId": 1617112251087320000
  },
  "protocolVersion": 2,
  "type": "102",
  "typeName": "CLAIM_DEF"
 },
 "txnMetadata": {
  "seqNo": 204,
  "txnId": "Vq2C7Wfc44Q1cSroPuXaw2:3:CL:126:Basis-ID",
  "txnTime": "2021-03-30T13:50:51.000Z"
 }
}
```

11 Sources

[1] Camenisch, Jan, and Els Van Herreweghen. "Design and implementation of the idemix anonymous credential system." Proceedings of the 9th ACM conference on Computer and communications security. 2002.

[2] The DLPS: A New Framework for Benchmarking Blockchains, https://scholarspace.manoa.hawaii.edu/bitstream/10125/71443/0670.pdf

[3] RBFT: Redundant Byzantine Fault Tolerance , https://pakupaku.me/plaublin/rbft/5000a297.pdf

[4] https://wiki.hyperledger.org/download/attachments/13862116/TECHNICAL_REPORT_Hyperledger_Indy_Linux_Foundation_2018-10-31_v1.0.pdf?version=1&modificationDate=1560353100000&api=v2

[5] Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance and proactive recovery." ACM Transactions on Computer Systems (TOCS) 20.4 (2002): 398-461.

[7] Boneh, Dan, Ben Lynn, and Hovav Shacham. "Short signatures from the Weil pairing." International conference on the theory and application of cryptology and information security. Springer, Berlin, Heidelberg, 2001.

[8] Camenisch, Jan, and Anna Lysyanskaya. "A signature scheme with efficient protocols." International Conference on Security in Communication Networks. Springer, Berlin, Heidelberg, 2002.

[10] Camenisch, Jan, Markulf Kohlweiss, and Claudio Soriente. "An accumulator based on bilinear maps and efficient revocation for anonymous credentials." International workshop on public key cryptography. Springer, Berlin, Heidelberg, 2009.

12 Index of exhibits

13 Index of tables