Kryptokonzept SSI Ökosystem

Februar 2022

Inhaltsverzeichnis

Offene Aufgaben

Einzelne offene Aufgaben sind im Dokument selbst dokumentiert.

- Begriffe und Schreibweisen vereinheitlichen (Holder, Nutzer, Verifier, etc.)
- Tabellen einheitlich formatieren
- Genderneutrale Sprache
- Abbildung 1 usecase-unabhängig gestalten
- Kapitel 5.3 (Faktor Wissen) vervollständigen
- Kapitel 7.4, 7.5, 10, 11.3.2, 11.4.2 ergänzen
- Begriffsdefinitionen vervollständigen
- Die genannte Orientierung am Leitfaden des BSI sollte vielleicht in der Kapitelstruktur kenntlicher gemacht werden.
- Übersicht der geltenden Vorgaben für die eingesetzte Kryptographie, wie beispielsweise TRs des BSI oder andere

- Eine stärkere Abgrenzung des Betrachtungsgegenstandes
- Anforderungen, um den geforderten Schutzbedarf decken zu können
- Abwägungsgründe für die genutzten Algorithmen, Kryptokomponenten
- Die Gültigkeitsdauer der Nachweise sowie deren Erneuerungsprozess
- Die von der Gültigkeitsdauer der Nachweise verbunden eingesetzten Signaturalgorithmen
- Sichere Schlüsselerzeugung und die genutzte Zufallszahlengeneratoren
- Darstellung der eingesetzten Schlüsselspeicher
- Auswahl sowie Anforderungen an den Einsatz geprüfter Softwarebibliotheken oder Hardware (z.B. CC)
- Festlegungen, welche die organisatorische Sicherheit (z.B. Betrieb) betreffen, wie bspw. Organisationen, Rollen, Verantwortlichkeiten

1 Einführung

Die Deutsche Bundesregierung plant den Aufbau eines dezentralen Identity-Management-Systems, das es Bürger*innen ermöglicht, ihre Identitätsdaten selbstbestimmt zu verwalten und zu verwenden (Self-Sovereign Identity, SSI), kurz SSI-Ökosystem. An dem SSI-Ökosystem sind eine Reihe von unterschiedlichen Organisationen beteiligt, die verschiedene Aufgaben übernehmen.

Ziel dieses Dokuments ist, die im System genutzten kryptographischen Verfahren im Detail zu beschreiben. Es orientiert sich in der Struktur am Leitfaden des BSI zur Erstellung von Kryptokonzepten.

Dieses Dokument definiert Vorgaben an eine Referenzarchitektur, welche in der Implementierung in adaptierter Form umgesetzt werden kann. Die Vorgaben sind in diesem Fall auch für ergänzte Komponenten und Kommunikationsverbindungen zu berücksichtigen und im Kontext der verwendeten Standards zu prüfen und anzuwenden.

2 Infrastruktur

Innerhalb des SSI-Ökosystems sollen verschiedene Anwendungsfälle für die sichere und datenschutzfreundliche Identifizierung von Nutzern gegenüber Dienstanbietern (im Folgenden Verifier) realisiert werden. Hierfür lassen sich Nutzer die für den Dienst notwendigen Identitätsattribute (im Folgenden Verifiable Credential) von Ausstellern (im Folgenden Issuer) ausstellen und speichern diese in einer Wallet auf ihrem Smartphone. Ein Verifier kann diese Verifiable Credentials prüfen und feststellen, dass diese echt sind und zu der sich authentisierenden Person gehören.

Beispielsweise beinhaltet die sogenannte Basis-ID, die von der Bundesdruckerei als Issuer ausgestellt wird, die Verifiable Credentials Name, Vorname, Geburtsdatum und Adresse. Es können aber auch Informationen über akademische Abschlüsse, die von den jeweiligen Hochschulen als Issuer ausgegeben werden, verwendet werden.

Bei der Umsetzung der Funktionalität wird ein Distributed Ledger Netzwerk verwendet, um öffentliche Schlüssel der beteiligten Parteien, DIDs, Credential Schemata und Revozierungsinformationen sicher zu speichern, siehe auch Abschnitt 7. Das Distributed Ledger Netzwerk wird von verschiedenen Knotenbetreibern betrieben.

Abb. 1 gibt einen Überblick über das System am Beispiel der Basis ID.

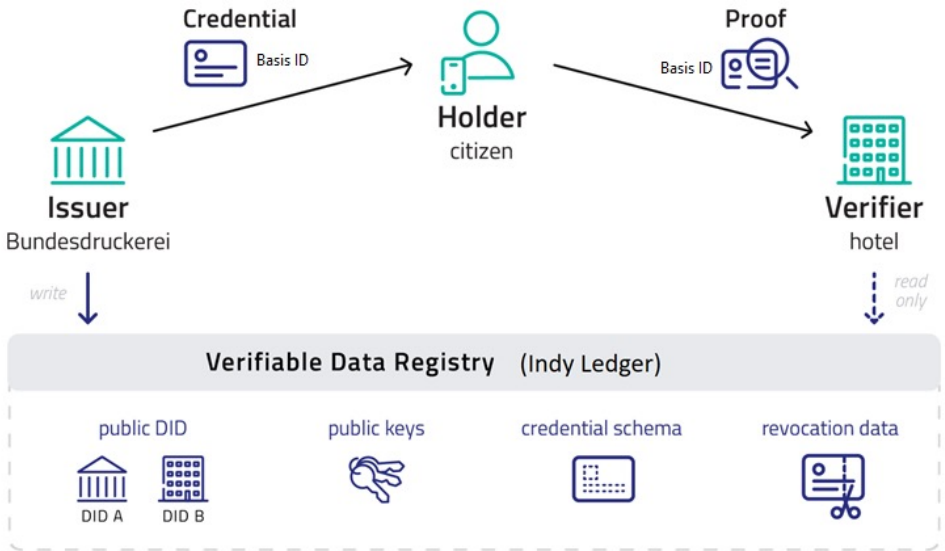

Abbildung 1: Systemübersicht

2.1 Komponenten des Systems

Holder erhalten die Verifiable Credentials von Issuern, speichern sie in ihrer Wallet-App und können sie gegenüber Verifiern präsentieren.

Das System besteht demnach aus den folgenden Entitäten:

- **Verifiable Data Registry**: Ein öffentliches, genehmigungspflichtiges Netzwerk (Hyperledger Indy bestehend aus den Indy Nodes) zur Speicherung von öffentlichem Schlüsselmaterial und dezentrale Prüfinfrastruktur (für das Zurückziehen von Verifiable Credentials), betrieben von authentisierten und autorisierten Knotenbetreibern.
- **Issuer** stellen Identitätsattribute in Form von Verifiable Credentials an den Holder aus. Sie betreiben **Tails-Server**, welche die für Revocation benötigten Informationen zur Verfügung stellen.
- **Verifier** sind Empfänger von Verifiable Credentials und stoßen den Verifizierungsprozess über das Netzwerk an, um die Authentizität der Daten zu prüfen und den Status (Revozierung) zu erfragen.
- **Wallet-App** und **Mediator-Agent** stellen die Schnittstelle dar, welche ein Holder nutzt um Basis-ID und Nachweise sowie zugehörige Notifications zu speichern, freizugeben und zu übertragen.

2.2 Datenerhebung

Die Stärke der eingesetzten kryptographischen Mechanismen und der organisatorischen Sicherheit hängt von der Höhe des Schutzbedarfs der Daten, IT-Systeme und Räumlichkeiten im Hinblick auf die aufgeführten kryptographischen Ziele ab, siehe Tabelle 1.

Im Ökosystem werden personenbezogene Daten verarbeitet und gespeichert. Der Inhalt der Daten hängt vom Anwendungsfall ab. Geplant ist, eine große Anzahl verschiedener Anwendungsfälle über das SSI-Ökosystem zu ermöglichen.

Zum Stand der Erstellung des Kryptokonzepts werden medizinische Daten als Nachweise ausgeschlossen. Eine umfassende Schutzbedarfsanalyse ist im IT-Sicherheitskonzept zu finden. Für die personenbezogenen Daten wird von folgenden Schutzbedarfen ausgegangen:

Kryptographisches Ziel	Schutzbedarf
Vertraulichkeit	hoch
Verfügbarkeit	normal
Integrität	hoch

Tabelle 1: Schutzbedarf personenbezogener Daten

3 AnonCreds 1.0

Ziel dieses Abschnitts ist eine kurze Beschreibung des verwendeten Protokolls AnonCreds 1.0 inkl. einer Auflistung der Schlüssellängen und der Längen sonstiger kryptographischer Parameter.

AnonCreds wird genutzt, um die Authentizität von Verifiable Credentials sicherzustellen und Verifiable Credentials zurückziehen zu können.

Basis für die Authentizität der Verifiable Credentials ist das Signaturverfahren von Camenisch-Lysyanskaya (CL-Signaturen), siehe [].

Basis für das Zurückziehen von Verifiable Credentials ist das Verfahren Type-3 Pairing wie in [] beschrieben.

3.1 Camenisch-Lysyanskaya Signaturen

Wir beschreiben zunächst kurz CL-Signaturen. Um ein Verifiable Credential mit L Attributen zu signieren, wird ein Signaturschlüssel wie folgt erzeugt:

- Wähle zwei große Primzahlen p, q und setzt $n = pq$.
- Wähle $L + 2$ quadratische Reste module n $R_1, \ldots, R_L, S, Z \in QR_n$
- Der öffentliche Schlüssel ist $pk = (n, R_1, \ldots, R_L, S, Z)$, der geheime Schlüssel $sk = (p, q)$

Alle Zufallswerte werden bzgl. der Gleichverteilung gewählt.

Um $m_1, \ldots, m_L$ Attribute zu signieren, werden die folgenden Schritte durchgeführt:

- Wähle $e \in Z_n^*$, $e \neq p, q$
- Wähle $v \in Z_n$

- Berechne $d \in Z_n^*$ mit $e \cdot d = 1 \bmod \phi(n)$ (dabei ist $\phi(n) = (p-1)(q-1)$)
- Berechne $A = (\frac{Z}{R_1^{m_1} \cdot \ldots \cdot R_L^{m_L} \cdot S^v})^d \bmod n$

v wird bzgl. der Gleichverteilung gewählt.

Die Signatur ist dann (A, e, v).

Um die Signatur zur prüfen, wird zunächst $Z' = A^e \cdot R_1^{m_1} \cdot \ldots \cdot R_L^{m_L} \cdot S^v \bmod n$ berechnet und $Z = Z'$ geprüft.

3.2 Revozierungsinformationen

Verifiable Credentials werden zusätzlich mit dem Revozierungsstatus versehen. Hierzu erzeugt der Issuer zunächst ein Schlüsselpaar wie folgt:

- Wähle eine große Primzahl q
- Wähle drei Gruppen G_1, G_2, G_T der Ordnung q und Erzeuger g_1 von G_1 und g_2 von G_2
- Wähle eine bilineare Abbildung $e : G_1 \times G_2 \longrightarrow G_T$
- Wähle $h, h_0, h_1, h_2, \widetilde{h} \in G_1$, $u, \widehat{h} \in G_2$ und $sk, x \in F_q$
- Berechne $pk = g^{sk}$ und $y = \widehat{h}^x$
- Wähle $\gamma \in F_q$
- Berechne
  - $g_1, g_2 \ldots, g_N, g_{N+2}, \ldots, g_{2N}$ mit $g_i = g^{\gamma^i}$
  - $g'_1, g'_2 \ldots, g'_N, g'_{N+2}, \ldots, g'_{2N}$ mit $g'_i = g'^{\gamma^i}$
  - $z = e(g, g')^{\gamma^{N+1}}$
  - Der öffentliche Schlüssel ist $(h, h_0, h_1, h_2, \widetilde{h}, \widehat{h}, u, pk, y, z)$, der geheime (sk, x, γ)

Alle Zufallswerte werden bzgl. der Gleichverteilung gewählt.

Damit kann der Issuer für N von ihm ausgegebene Verifiable Credentials bearbeiten. Gibt er mehr als N Verifiable Credentials aus, erzeugt er ein neues Schlüsselpaar.

Für jedes vom Issuer ausgegebene Verifiable Credentials berechnet der Issuer

3.3 Initialisierung

Ein Issuer erzeugt zwei Schlüsselpaare (für die CL-Signatur und Type-3 Pairing) und legt diese im Netzwerk ab.

Zu ergänzen, wie Speicherung abgesichert ist.

Für das CL-Signatur-Schlüsselpaar haben die beiden Primzahlen p, q eine Biltänge von XX.

Für das Type-3-Pairing-Schlüsselpaar werden folgende Objekte genutzt:

- q hat eine Bitlänge von XX
- $G_1 = XX, G_2 = XX, G_T = XX$
- $e : G_1 \times G_2 \longrightarrow G_T, (x, y) \mapsto z$

3.4 Ausstellung von Verifiable Credentials inkl. Revozierungsstatus

3.5 Revozierung von Verifiable Credentials

3.6 Verifizierung von Verifiable Credentials inkl. Ermittlung des Revozierungsstatus

4 Nachweise

Nachweise belegen Eigenschaften der Holder, wie z.B. einen Führerschein oder einen akademischen Abschluss. Nachweise dienen **nicht** dazu, sich als Holder gegenüber Verifiern zu authentisieren. Es muss nur die Echtheit der Nachweise (über entsprechende Signaturen) verifizierbar sein. Nachweise enthalten die in Tabelle 2 aufgeführten Daten.

Dabei ist m_1 das sogenannte Link Secret, m_2 die Credential ID, die für den Rückruf des Nachweises genutzt wird und m_3 die Policy ID (derzeit keine Verwendung). In den Attributen $m_5, \cdots, m_L$ sind die Attribute des Holders enthalten (z.B. Name, Vorname, akademischer Abschluss).

Die Signatur (A, e, v) dient der Verifizierung der elektronischen Identität durch den Verifier.

m_1:	Private ID (Link Secret)
m_2:	Credential ID (Revocation)
m_3:	Policy ID
m_4:	Attribute 4
m_5:	Attribute 5
$\vdots$	$\vdots$
m_L:	Attribute L
Signatur:	(A, e, v)

Tabelle 2: Credential schema used in AnonCreds 1.0

Im Folgenden wird generisch beschrieben, wie Nachweise ausgestellt, genutzt, zurückgerufen und gespeichert werden und welche kryptographischen Protokolle zur Sicherstellung des Sicherheitsziels Authentizität genutzt werden.

4.1 Ausstellung von Nachweisen

Nachweise werden von Issuern ausgestellt, die belegen können, dass die in den Nachweisen behaupteten Attribute echt sind. So kann z.B. ein Nachweis über einen akademischen Abschluss von einer Hochschule, bei welcher der akademische Grad erworben wurde, ausgestellt werden.

Der Issuer startet die Kommunikation mit der Wallet des Holders über den Mediation Service. Hierfür wird zunächst eine TLS-Verbindung zwischen Issuer und Mediation Service (mit gegenseitiger Authentisierung) und eine TLS-Verbindung zwischen Mediation Service und Wallet (nur Server-Authentisierung) aufgebaut. Über diese Kanäle wird sendet dann der Issuer eine DIDComm-Invitation. Diese beinhaltet die URL des Issuers. Issuer und Wallet bauen dann eine TLS-Verbindung (nur Server-Authentisierung) auf. Im Anschluss wird eine DIDComm-Verbindung zwischen Issuer und Wallet aufgebaut, siehe Kapitel 6 für die zur Absicherung der Kommunikationskanäle verwendeten kryptographischen Verfahren.

Der Holder authentisiert sich zunächst gegenüber dem Issuer (z.B. mittels der Basis ID, siehe Abschnitt 5), welcher die Identität prüft. Eine detaillierte Prozessdarstellung kann in [**anoncred**], Kapitel 4 gefunden werden.

Zusätzlich werden vom Issuer Informationen zum Zurückrufen von Nachweisen erzeugt und im Netzwerk gespeichert. Hierzu baut der Issuer zunächst

eine CurveZeroMQ-Verbindung mit dem Netzwerk auf, siehe Kapitel 6 für die dabei verwendeten kryptographischen Verfahren.

Der detaillierte Prozess zum Erzeugen von Rückrufinformationen für Nachweise ist in [**anoncred**], Kapitel 7 beschrieben.

4.2 Rückruf von Nachweisen

Ein Rückruf von Nachweisen kann notwendig werden, wenn Eigenschaften des Holders, die in Nachweisen enthalten sind, ungültig werden. Rückrufe können von Issuern als auch von Holdern veranlasst werden. Veranlasst der Holder den Rückruf, muss er/sie sich zunächst gegenüber dem Issuer authentisieren (z.B. über die Basis ID oder ein Sperrkennwort, welches er/sie im Ausstellungsprozess erhalten hat). Der Rückruf wird vom Issuer durchgeführt, indem er zunächst eine CurveZeroMQ-Verbindung mit dem Netzwerk aufbaut, siehe Kapitel 6 für die hier verwendeten kryptographischen Verfahren.

Der detaillierte Prozess zum Rückruf von Nachweisen ist in [**anoncred**], Kapitel 7.3 beschrieben.

4.3 Nutzung

Der Verifier startet die Kommunikation mit der Wallet des Holders über den Mediation Service. Hierfür wird zunächst eine TLS-Verbindung zwischen Verifier und Mediation Service (mit gegenseitiger Authentisierung) und eine TLS-Verbindung zwischen Mediation Service und Wallet (nur Server-Authentisierung) aufgebaut. Über diese Kanäle wird sendet dann der Verifier eine DIDComm-Invitation. Diese beinhaltet die URL des Verifiers. Verifier und Wallet bauen dann eine TLS-Verbindung (nur Server-Authentisierung) auf. Im Anschluss wird eine DIDComm-Verbindung zwischen Verifier und Wallet aufgebaut, siehe Kapitel 6 für die zur Absicherung der Kommunikationskanäle verwendeten kryptographischen Verfahren.

Der Verifizierungsprozess ist in [**anoncred**], Kapitel 5 beschrieben.

Weiter prüft der Verifier, ob der Nachweis zurückgezogen wurde. Hierfür baut er zusätzlich eine CurveZeroMQ-Verbindung mit dem Netzwerk auf, siehe Kapitel 6 für die hier verwendeten kryptographischen Verfahren.

Der Prozess zum Prüfen des Revozierungsstatus ist in [**anoncred**], Kapitel 7.2 beschrieben.

5 Basis ID

Die Basis ID ist eine elektronische Identität, mit der sich Holder gegenüber Verifiern authentisieren können. Die Basis ID beinhaltet die in Tabelle 3 aufgeführten Daten.

m_1:	Private ID (Link Secret)
m_2:	Credential ID (Revocation)
m_3:	Policy ID
m_4:	Public Key
m_5:	Attribute 1
m_6:	Attribute 2
$\vdots$	$\vdots$
m_L:	Attribute n
Signatur:	(A, e, v)

Tabelle 3: Credential schema used in AnonCreds 1.0

Zu ergänzen, wo pk gespeichert ist

Dabei ist m_1 das sogenannte Link Secret, m_2 die Credential ID, die für den Rückruf der Basis ID genutzt wird, m_3 die Policy ID (derzeit keine Verwendung) und m_4 ein öffentlicher Schlüssel des Holders, der für ein Challenge-Response-Protokoll genutzt wird, um die Gerätebindung umzusetzen, siehe Abschnitt 5.1. In den Attributen $m_5, \cdots, m_L$ sind die Identitätsattribute der Person enthalten (Name, Vorname, Adresse usw.).

Die Signatur (A, e, v) dient der Verifizierung der elektronischen Identität durch den Verifier.

Die Mechanismen zur Ausstellung, Rückruf und Nutzung der Basis ID sind analog zu den bereits in Kapitel 4 beschriebenen Prozessen für Nachweise. Im Gegensatz zu Nachweisen wird die Basis ID zur Authentisierung genutzt. Daher ist eine Zwei-Faktor-Authentisierung umzusetzen. Die Lösung nutzt die beiden Faktoren Wissen und Besitz. Wissen ist über eine PIN umgesetzt, siehe dazu Abschnitt 5.3. Der Faktor Besitz wird über einen asymmetrischen geheimen Schlüssel abgebildet, der sich im sicheren Speicher des Smartphones befindet, siehe Abschnitt 5.1.

5.1 Faktor Besitz: Ausstellung

Das Protokoll AnonCreds wird für die Basis ID um ein Challenge-Response-Verfahren erweitert, um den Faktor Besitz umzusetzen. Im Folgenden werden lediglich die zusätzlichen Mechanismen zur Umsetzung der Zwei-Faktor-Authentisierung beschrieben.

Grob skizziert ist der Prozess wie folgt umgesetzt. Die einzelnen Schritte werden, wenn sinnvoll, in den folgenden Abschnitten erläutert.

1. Die Wallet startet den Prozess mit dem Aufruf des Bundesdruckerei-API-Endpunkts (TLS-gesichert, siehe Kapitel 6 für die verwendeten kryptographischen Verfahren)
2. Die Wallet authentisiert sich mittels API-Key (siehe Unterabschnitt 5.1.1) und mittels smartphonespezifischer Sicherheitsfunktionen (siehe Unterabschnitt 5.1.2)
3. Der Holder authentisiert sich gegenüber dem Issuer mittels der Online-Ausweisfunktion, siehe Unterabschnitt 5.1.3
4. Nach erfolgreicher Authentisierung des Holders mittels Online-Ausweisfunktion antwortet der Issuer mit einer DIDComm-Invitation
5. Der Issuer erzeugt die für die Berechnung der Basis ID notwendigen Werte und sendet diese an die Wallet (über den DIDComm-Kanal), siehe Unterabschnitt 5.1.4
6. Weiter erzeugt der Issuer ein Sperrkennwort, über das der Holder seine Basis ID sperren lassen kann, siehe Unterabschnitt 5.1.5

Für die Authentisierung des Holder über die Online-Ausweisfunktion und das Ausstellen der Werte für die Berechnung der Basis ID werden verschiedene Kommunikationsverbidnungen genutzt. Diese müssen kryptographisch verbunden sein, siehe hierfür Unterabschnitt 5.1.6.

5.1.1 API Key

Das Ausstellen einer Basis-ID durch die Bundesdruckerei GmbH erfordert die Verwendung eines API Keys für das Kontaktieren des folgenden Bundesdruckerei API Endpunkt:

- TLS-API Endpunkt zum Initiieren des integrierten Flow der Basis-ID

Der API Key (128 Bit UUID, Wallethersteller- und versionsspezifisch) wird über einen out-of-band Kanal von der Bundesdruckerei GmbH an die esatus AG weitergereicht. Der Prozess ist folgendermaßen definiert:

- UUID wird von der Bundesdruckerei in ihrer Deployment Umgebung generiert
- UUID wird mittels sicherem Transportmedium (verschlüsselte und signierte eMail) an esatus geschickt.

Die esatus AG bindet den erhaltenen API Key als Konstante (X-API-KEY) in den Quellcode der ID Wallet App ein.

Die eigentliche Authentisierung gegenüber der Endpunkte erfolgt via X-API-Key Feld im HTTP-Header.

5.1.2 SafetyNet und DeviceCheck

Kurze Erläuterung

5.1.3 Online-Ausweisfunktion

Die Identifizierung der Nutzer*in erfolgt mittels der eID-Funktion des Personalausweises. Die Issuer-API liefert dafür den Einstiegspunkt (tcTokenUrl) für den eID-Prozess. Damit baut das in der Wallet integrierte AusweisApp2-SDK eine Verbindung zum eID-Server auf, prüft die Ausleseberechtigung und holt die Zustimmung der Nutzer*in ein. Dieser autorisiert den Auslesevorgang mit Eingabe seiner eID-PIN in der Wallet-App. Im eID-Kanal werden die angeforderten Identitätsdaten von der eID-Karte (Personalausweis) an den eID-Server übertragen. Der eID-Server antwortet dem integrierten eID-Client nach erfolgreichem Auslesen mit der redirectUrl. Diese beinhaltet lediglich den Identifier für die eID-Session, der integrierte eID-Client übergibt die redirectUrl an die Wallet und diese ruft den API-Endpunkt aus redirectUrl auf. Der Ausstellungsdienst fragt die Identitätsdaten beim eID-Server an (über einen TLS-gesicherten Kanal) und der eID-Server gibt die ausgelesenen Identitätsdaten an den Issuer-Service frei. Der Ausstellungsdienst hat jetzt die verifizierten Identitätsdaten der Nutzer*in.

5.1.4 Ausstellung der Basis ID

Für die Ausstellung der Basis ID werden folgende Schritte durchgeführt:

1. Wallet startet mit Basis ID Ausstellungsanfrage
2. Issuer antwortet zusätzlich mit einer zufällig generierten Nonce

   $$\text{issuer\_nonce} = (\text{unixtimestamp}||\text{UUID\_}v4)$$

   mit 122 zufälligen Bits
3. Wallet leitet aus der $issuer\_nonce$ ab:
   (a) Input für SafetyNet/DeviceCheck Wallet Attestation API

   $$nonce\_wallet\_validation = SHA256(issuer\_nonce|0x00)$$

   (b) Input für KeyAttestation API

   $$nonce\_key\_attest = SHA256(issuer\_nonce|0x01)$$

   (c) Input für KeyStore Crypto Operation

   $$nonce\_key\_auth = SHA256(issuer\_nonce|0x02)$$

4. Wallet erstellt die Wallet-Authentisierung $attestation\_wallet\_validation$ mittels SafetyNet/DeviceCheck unter Einbeziehung der $nonce\_wallet\_validation$
5. Wallet erstellt ein neues hardware-gebundenes asymmetrisches Schlüsselpaar
   (a) Unter iOS: Die Wallet weist unter Einbeziehung der $nonce\_key\_auth$ den Besitz des privaten Schlüssels durch die Berechnung der Signatur nach

   $$key\_challenge\_signature = SHA256withECDSA(nonce\_key\_auth)$$

   (a) Unter Android
      i. Die Wallet erstellt eine Attestierung des Schlüsselpaars $attestation\_key$ unter Einbeziehung der $nonce\_key\_attest$
      ii. Die Wallet bezieht die Zertifikatskette der Schlüsselattestierung $attestation\_certs$

iii. Die Wallet weist unter Einbeziehung der *nonce_key_auth* den Besitz des privaten Schlüssels durch die Berechnung der Signatur nach

$$key\_challenge\_signature = SHA256withECDSA(nonce\_key\_auth)$$

6. Wallet übermittelt *attestation_key*, *attestation_certs* (nur unter Android) und *key_challenge_signature*
7. Issuer überprüft die Integrität der Wallet mittels SafetyNet (Android) bzw. DeviceCheck (iOS) an Hand der *attestation_wallet_validation*
8. Issuer prüft den attestierten Schlüssel *attestation_key* mit Hilfe der Zertifikatskette *attestation_certs*
   (a) Zertifikatskette gegen die Google Root Zertifikate prüfen sowie Revozierungsliste der einzelnen KeyAttestation Zertifkate abgleichen
   (b) Attributeigenschaften der Schlüsselattestierung evaluieren
9. Issuer überprüft die Instanzauthentisierung des Challenge-Response mit Hilfe der Signatur *key_challenge_signature*
10. Issuer antwortet der Wallet bei Erfolg, Ausstellung kann fortgesetzt werden mit Attribut hardwareDid aus *attestation_key*

5.1.5 Sperrinformationen

Der Issuer generiert ein Sperrkennwort und speichert die Daten für den Lebenszyklus in der internen Sperrdatenbank. Das Sperrkennwort wird über den DIDComm-Kanal übertragen und dem Holder angezeigt.

5.1.6 Kanalbindung

Die enge Bindung zwischen Onlineausweisfunktion und Deployment der Credentials wird durch die Integration des AusweisApp2-SDKs in die ID Wallet App auf Softwareebene wie folgt gewährleistet:

Der Ausstellungsprozess der Basis-ID wird durch die Wallet initiiert. Sie baut einen TLS-Kanal zum API-Endpunkt der Aussteller*in (SSI Issuer) auf, dieser Kanal wird zusätzlich mit einem API-Schlüssel und Zertifikatspinning (Public Key) geschützt. Der SSI Issuer antwortet mit der sitzungsgebundenen *tcTokenUrl*, die auf den AusweisIdent eService zeigt. Durch diese Anfrage

wird der Online-Authentisierungsvorgang des Personalausweises mittels AusweisIdent Service angestoßen, dieser nimmt im eID-Kontext sowohl die Rolle des eIDServers als auch des eService ein. Die *tcTokenUrl* wird von der Wallet an das integrierte AusweisApp2-SDK übergeben und antwortet mit dem *tcToken*, dieses bildet die Startparameter für den eID-Prozess. Anhand des *tcToken* vermittelt der eID-Client/AusweisApp2-SDK einen sicheren eID-Kanal zwischen dem eID-Dokument und dem eID-Server. Nach Prüfung des Berechtigungszertifikats und Autorisierung des eID-Ausleseprozesses durch die Nutzer*in mittels PIN-Eingabe wird der eID-Kanal aufgebaut und die eID-Attribute im eID-Kanal an den eID-Server übertragen. Anschließend antwortet der eID-Server mit der *redirectUrl* an den eID-Client/AusweisApp2-SDK, diese wird an die Wallet übergeben und aufgerufen. Die *redirectURL* zeigt zunächst auf den eService (AusweisIdent) und wird beim Aufruf abschließend an einen sicheren API-Endpunkt (TLS) der Aussteller*in (SSI Issuer) weitergeleitet (es wird ein neuer TLS-Kanal aufegbaut und die Sessions mittels, im volatilen Speicher des Servers abgelegten Session-Cookies gematcht). Der eService der Aussteller (AusweisIdent) holt nun die eID-Daten vom eID-Server ab. Der SSI Issuer holt die Ausweisdaten vom eService mittels AusweisIdent-Schnittstellen (OpenID Connect) ab. Dabei stellt AusweisIdent die Rolle des OpenID Anbieters dar und der SSI Issuer ist in der Rolle der Relying Party. Der Aussteller antwortet der Wallet nun über den gesicherten API-Endpunkt (TLS) mit der DIDComm-Invitation, diese wird vom SSI-Agenten des Ausstellers generiert. Eine DIDComm-Invitation kann nicht mehrfach benutzt werden. Die Wallet kann mittels dieser DIDComm-Invitation nun einen sicheren DIDComm-Kanal zum SSI-Agent des Ausstellers aufbauen. Über diesen DIDComm-Kanal wird nun das Credential mit den eID-Atttributen ausgestellt. Zusammenfassend wird die Kanalbündelung erreicht, indem die Wallet zwei TLS-Kanäle mit der Aussteller*in aufbaut (verbunden über eine Session ID) und über diese TLS-Kanäle sowohl der eID-Kanal als auch der DIDComm-Kanal initiiert und von der integrierten Wallet verwaltet werden.

5.2 Faktor Besitz: Nutzung

Trusted-Verifier-Konzept ist zu integrieren.

Für die Nutzung der Basis ID wird neben den Standardmechanismen des AnonCreds-Protokolls zwischen Verifier und Holder zusätzlich innerhalb dieses Protokollverlaufs die Informationen und das Challenge-Response Verfahren für den gerätegebundenen Schlüssel eingebunden. Dieser Prozess setzt

die Eingabe der Inhaber-PIN (siehe Abschnitt 5.3) voraus und ist wie folgt umgesetzt:

1. Verifier sendet den presentation request. Dieser enthält:
   (a) Zufällig generierte Nonce nonce_indy (80 Bit)
   (b) Abfrage zu allen benötigten Attributen der Basis ID inkl. öffentlichem Schlüssel des Holders
   (c) Zusätzlich Anfrage zur Durchführung des Challenge-Response-Protokolls
2. Wallet berechnet presentation:
   (a) Offenlegung und Beweis der Authentizität der Basis-ID Attribute (enthält öffentlichen Schlüssel des Holders) nach AnonCreds 1.0
   (b) Die Wallet leitet aus der *indy_nonce* ab: $nonce\_key\_auth = SHA256(nonce\_indy|0x02)$
   (c) führt unter Nachweis der Inhaber-PIN das Challenge-Response-Verfahren mit Verifier mittels *attestation_key* aus dem Attribut öffentlicher Schlüssel und *nonce_key_auth* durch: $hardwareDidProof = SHA256withECDSA(nonce\_key\_auth)$
   (d) das Ergebnis wird im Attribut hardwareDidProof gespeichert
3. Wallet sendet Presentation
4. Verifier prüft Gültigkeit der Credential Attribute und Korrektheit des hardwareDidProof

5.3 Faktor Wissen

Dieser Abschnitt bezieht sich auf die Inhaber-Authentisierung. Der Wert der Inhaber-PIN (z.B. 123456) ist der gleiche wie der Wert der Wallet-PIN, die Verifizierung der Inhaber-PIN unterscheidet sich jedoch von der Verifikation der Wallet-PIN. Diese Inhaber-PIN-Abfrage wird über den Programmablauf erzwungen, siehe Grafik 3.

Um dies zu realisieren, wird für die Inhaber-PIN bei initialer Einrichtung der ID Wallet auf Basis des Wertes der Wallet-PIN und mit einer zufällig erzeugten Salt (16 Byte) per PBKDF2 (100.000 Iterationen) ein Derivat abgeleitet. Beides (Derivat und Salt) wird im Secure Storage der jeweiligen Plattform (iOS/Android) gespeichert.

1. Pin_deriv = PBKDF2(PIN | Pin_salt)
2. Speichere Pin_deriv und Pin_salt in Secure Storage

Dabei ist das Pin_salt != Key_enc_data_salt

Im Rahmen der Inhaber-Authentisierung wird dann die vom Nutzer eingegebene Inhaber-PIN erneut abgeleitet und mit dem im Secure Storage gespeicherten Wert verglichen:

1. DerivedInput = PBKDF2(PIN_INPUT | Pin_salt)
2. DerivdedInput =? Pin_deriv
3. Wenn DerividedInput == Pin_deriv Freigabe des Challenge-Response-Verfahrens mit Nonce des Proof-Requests unter Verwendung des privaten Schlüssels von hardwareDID (Der Aufruf zur Durchführung der Signatur der Challenge und das Senden des eigentlichen Proofs – unter Verwendung des sicheren Gerätespeichers – wird erst nach einer erfolgreichen Überprüfung der Inhaber-Pin per Softwarelogik ausgelöst).
4. Hinzufügen des Challenge-Response-Ergebnis zum Proof (hardwareDidProof)
5. Übermittlung des Proof-Response inkl. Challenge-Response

Nach 5-maliger Falscheingabe des Inhaber-PINs wird der pre_key gelöscht und die Entschlüsselung der SQLite Datenbank damit unmöglich. Der Fehlbedienungszähler ist in Software umgesetzt, die Speicherung des Counters erfolgt im jeweiligen Secure Storage.

Dabei wird der Secure Storage wie folgt umgesetzt:

1. Android
   (a) TEE
      i.
   (b) Strongbox
      i.
2. iOS
   (a) Secure Enclave
      i.

5.4 Schlüssellängen und Schlüsselgenerierung

Für die Umsetzung der Gerätebindung werden folgende kryptographische Verfahren genutzt:

- Elliptische Kurve NIST P256 (nist-p256r1)
- Signaturalgorithmus ECDSA mit SHA256

In der konkreten Umsetzung werden unter Android folgende Parameter verwendet:

- Schlüsselerzeugung unter Android:
  - Digest: DigestSha256
  - Signatur: SHA256withECDSA
  - AlgorithmParameterSpec: secp256r1
  - Inhaber-Pinabfrage bei Nutzung des generierten Schlüssels
  - SetRandomizedEncryptionRequired: false
  - SetUserAuthenticationValidityDurationSeconds (da Ziel API Level < 30) : -1
  - SetIsStrongBoxBacked: True (wenn Strongbox genutzt werden kann)
- Schlüsselüberprüfung unter Android für die Überprüfung der Zertifikatskette:
  - attestationVersion: $>= 2$
  - attestationSecurityLevel: TrustedEnvironment (1) oder StrongBox (2)
  - keymasterSecurityLevel: TrustedEnvironment (1) oder StrongBox (2)

In der konkreten Umsetzung werden unter iOS folgende Parameter verwendet:

- Schlüsselerzeugung unter iOS:
  - Signatur: EcdsaSignatureMessageX962Sha256
  - KeyType: ECSecPrimeRandom

  - SecAccessible: WhenUnlockedThisDeviceOnly
  - SecAccessControlCreateFlags: PrivateKeyUsage
  - KeySizeInBits: 256

- Schlüsselüberprüfung unter iOS:
  - Unter iOS ist keine gesonderte Überprüfung des Schlüsselmaterials möglich, hier wird die Authentizität der ID Wallet über die Durchführung und Überprüfung eines DeviceCheck gewährleistet (siehe Kapitel 4.6).

In jedem Fall erfolgt eine Inhaber-Pinabfrage bei Nutzung des generierten Schlüssels, wie detailliert beschrieben in **??**

6 Absicherung der Kommunikationsverbindungen zwischen Komponenten des Ökosystems

Tabelle 4 gibt einen Überblick über alle im Netzwerk genutzten Kommunikationsverbindungen.

Welche kryptographischen Verfahren genutzt werden, wird in den folgenden Unterabschnitten erläutert.

Kommunikationsverbindung	Protokoll
Issuer ↔ Wallet	DIDComm
Tails Server ↔ Wallet	tls
Issuer ↔ Mediation Service	tls
Mediation Service ↔ Wallet	tls
Issuer ↔ Netzwerk	CurveZeroMQ
Verifier ↔ Wallet	DIDComm
Verifier ↔ Mediation Service	tls
Verifier ↔ Netzwerk	CurveZeroMQ
Wallet ↔ Netzwerk	CurveZeroMQ

Tabelle 4: Kommunikationsbeziehungen zwischen Komponenten des Ökosystems

6.1 TLS-Verbindungen

Es wird TLS 1.2 oder höher verwendet. Folgende Cipher Suites werden genutzt:

- 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

6.2 DIDComm

Eine Beschreibung des DIDComm-Protokolls findet sich in []. Folgende Algorithmen werden verwendet:

- Schlüsseleinigung: X25519, siehe []
- Verschlüsselung: XChaCha20, siehe []
- Datenauthentisierung: Poly1305, siehe []

DIDComm wird für die Kommunikation zwischen Wallet und Issuer oder Verfifier genutzt. Die Wallet generiert für jeden Verbindungsaufbau ein neues Schlüsselpaar für die Schlüsseleinigung, Verifier bzw. Issuer nutzen das Schlüsselpaar, welches sie beim Onboarding im Netzwerk für den Signaturalgorithmus Ed25519 generieren und aus welchem ihre DID abgeleitet wird, siehe Abschnitt 7.2.

6.3 CurveZeroMQ-Verbindungen

Eine Beschreibung des CurveZeroMQ-Protokolls findet sich in []. Folgende Algorithmen werden genutzt:

- Schlüsseleinigung: X25519, siehe []
- Verschlüsselung: XSalsa20, siehe []
- Datenauthentisierung: Poly1305, siehe []

CurveZeroMQ wird für die Kommunikation zwischen dem Netzwerk und Issuer oder Verifier genutzt. Alle Parteien nutzen für die Schlüsseleinigung diejenigen Schlüssel, welche sie beim Onboarding im Netzwerk für den Signaturalgorithmus Ed25519 generieren und aus welchem ihre DID abgeleitet wird, siehe Abschnitt 7.2.

7 Distributed Ledger Netzwerk

Bei der Umsetzung der Funktionalität wird ein Distributed Ledger Netzwerk verwendet, um

- die öffentlichen Schlüssel und DIDs aller beteiligten Parteien
  - Issuer: öffentliche Schlüssel zum Verifizieren von Verifiable Credentials, öffentliche Schlüssel zur Verifizierung des Revozierungsstatuses, öffentliche Schlüssel für die Schlüsseleinigung in den Protokollen DIDComm und CurveZeroMQ,
  - Verifier: öffentliche Schlüssel für die Schlüsseleinigung in den Protokollen DIDComm und CurveZeroMQ,
  - Knotenbetreiber: öffentliche Schlüssel für die Verwendung des Konsensalgorithmus, öffentliche Schlüssel für die Signatur von Daten)
- Credential Schemes
- Genesis Files
- Informationen zum Revozierungsstatus von Verifiable Credentials

sicher zu speichern.

Das Netzwerk besteht aus mehreren Netzwerkknoten (validator nodes), die von ernannten Akteuren (stewards) betrieben werden. Die validator nodes bilden untereinander Vertrauen durch den Konsensalgorithmus, siehe Abschnitt 7.2. Die Lesezugriffe auf das Netzwerk sind frei zugänglich und erfordern keine Authentisierung. Schreibzugriffe sind genehmigungspflichtig und erfordern neben der Signatur des Autors auch noch der Signatur eines Endorser, die von den validator nodes geprüft werden, siehe Abschnitt 7.2.

Eine sinnvolle und angestrebte Größe des Netzwerks beinhaltet circa 15-25 validator nodes, sodass eine Ausfallsicherheit gewährleistet ist und der Konsensalgorithmus effizient arbeitet. Fallen mehr als ein Drittel der Knoten aus,

so wechselt das Netzwerk in einen „read-only“ modus, dabei können keine neuen Transaktionen wegen des fehlenden Konsens erzeugt werden, Lesezugriffe sind aber weiterhin möglich.

7.1 Rollen

Um die vertrauensgebenden Rollen der technischen Basis zu komplementieren, gibt es eine Governance-Organisation. Diese repräsentiert das Netzwerk und bildet den rechtlichen Rahmen mit der realen Geschäftswelt ab. Sie schließt Verträge mit Personen und Organisationen und bindet sie an ihre technischen Rollen und die damit gegebenen Rechte und Verpflichtungen. Diese Pflichten sichern in erster Linie das Vertrauen, dass die Akteure nach bestem Wissen und Gewissen handeln und alle notwendigen Sicherheitsvorkehrungen treffen, um das Netzwerk zu schützen.

Um die Regeln des genehmigungspflichtigen Netzwerks durchzusetzen, hat der Ledger seine eigene Konfiguration und ein integriertes Rechte- und Regelsystem gespeichert. Die Rollen umfassen:

- Trustees verwalten die Rechte anderer Mitglieder und können ihnen Rollen und damit Rechte innerhalb des Netzwerks zuweisen. Zudem legen sie auch das Regelwerk und die detaillierten Berechtigungen und Anforderungen der einzelnen Regeln fest. Im Projekt sind dies: BWI GmbH, Deutsche Bahn AG, Bundesdruckerei GmbH, IBM Deutschland GmbH, esatus AG.

- Stewards sind die Betreiber der Netzwerkknoten (validator nodes). Jeder Steward betreibt einen Knoten und hat das Recht, die dazugehörigen Informationen wie IP, Port und Schlüssel, zu ändern. Im Projekt sind dies: BWI GmbH, Deutsche Bahn AG, Bundesdruckerei GmbH, IBM Deutschland GmbH, esatus AG.

- Endorser bestätigen den Schreibzugriff regulärer Datentransaktionen. Hauptaufgabe liegt darin, die Identitäten ohne weitere Rollen, wie z.B. Issuer, auf das Netzwerk zu schreiben, indem er die Transaktionen mittels seiner Signatur autorisiert. Im Projekt sind dies: BWI GmbH, Deutsche Bahn AG, Bundesdruckerei GmbH, IBM Deutschland GmbH, esatus AG.

- Der Netzwerkmonitor darf die Metainformationen der Stewards auslesen, um den Gesamtstatus und das Wohlbefinden des Netzwerks zu

überprüfen und zu überwachen. Der Netzwerkmonitor kann keine sicherheitsrelevanten Informationen lesen oder schreiben. Der Netzwerkmonitor ist Teil der Governance-Organisation.

Jede dieser Rollen ist ein sogenannter Decentralized Identifier (DID) auf dem Netzwerk. Die Autorisierung kann durch die Knoten geprüft werden, in dem die Rolle der DIDs und die Signatur mittels des zugehörigen öffentlichen Schlüssels validiert wird. Die Projektteilnehmer werden über bestehende Projekt-Kommunikationsmittel (Projekt-Confluence mit abgesicherter HTTPS-Verbindung und Nutzerauthentifizierung über Benutzername/Passwort oder eMail-Verschlüsselung/Signatur) identifiziert und ihrer Rolle zugewiesen. Hierüber werden auch insbesondere die öffentlichen Schlüssel ausgetauscht. Zu Beginn des Projekts wurden alle Teilnehmer identifiziert. Die Einhaltung der Regeln wird von den Knotenbetreibern durchgesetzt, genauer die eingesetzte Software Hyperledger Indy. Vorerst sind Zu- und Abgänge von Knotenbetreibern nicht vorgesehen. Das spezifische Regelwerk, welche und wie viele Zustimmungen (und damit Signaturen) für das Anlegen von Identitäten und Rollen notwendig ist, steht im ledger-internen Regelwerk. Dieses beinhaltet neben den Regeln zum Erstellen und Ändern von Identitäten, Schemata, Credential Definitionen und Revozierungsinformationen auch die Regeln zu den Validator Nodes und die Regeln zum Ändern des Regelwerks selbst. Die Regelbedingungen können dabei auf ein oder mehrere Rollen verweisen und mit logischen Operationen kombiniert werden. Das Regelwerk wird normalerweise von einer Governance-Organisation gestellt und technisch durch die Trustees umgesetzt. Derzeit wird das Standard-Regelwerk benutzt[1].

7.2 Erstellen des Netzwerkes

Für das Erstellen eines Netzwerks erzeugt jeder Knotenbetreiber zunächst ein Schlüsselpaar (Public und Private Key) für den Konsensalgorithmus.

- Signaturalgorithmus: Boneh-Lynn-Shacham-Multisignaturen, siehe []
- Schlüssellänge:

Schlüssellänge ist zu ergänzen.

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

Konsens wird mit dem Plenum-Protokoll gebildet, ein Redundant Byzantine Fault Tolerance Algorithmus (RBFT), welcher eine redundante Version des als sicher evaluierten PBFT darstellt, siehe []. Dieser bedingt für eine maximale Anzahl F von bösartig oder fehlerhaft agierenden Knoten eine Anzahl $N = 3F + 1$ von Gesamtknoten im Netzwerk. Bis zu 1/3 der Knoten können auch kurzzeitig ausfallen oder nicht erreichbar sein, ohne dass die Funktionalität des Netzwerks beeinträchtigt ist. Sie können danach von den laufenden Knoten auf den aktuellen Stand gebracht werden und wieder am Konsens teilnehmen.

Jeder Knotenbetreiber erzeugt zusätzlich ein Schlüsselpaar (Public und Private Key) für einen Signaturalgorithmus.

- Signaturalgorithmus: Ed25519
- Schlüssellänge:

Schlüssellänge ist zu ergänzen.

Weiter wird aus dem Schlüsselpaar eine DID abgeleitet. Dieser DID wird dann die Rolle eines Stewards zugewiesen (von allen Knotenbetreibern über das Akzeptieren der Genesis-Files, siehe Abschnitt 7.5), was dem Knotenbetreiber später die Administration seines Knotens im Netzwerk ermöglicht. Beide Public Keys (BLS und Ed25519) eines Knotenbetreibers werden gemeinsam mit der IP-Adresse des Servers, auf welchem der Knoten läuft, und den beiden für das Netzwerken verwendeten Ports (für die Knoten-zu-Knoten-Kommunikation und Knoten-zu-Client-Kommunikation) an alle weiteren Knotenbetreiber weitergegeben, sodass sich die Knoten miteinander verbinden können und das Netzwerk starten kann. Dafür werden gemeinsam (innerhalb des abgesicherten Projekt-Confluence) zwei Genesisfiles erzeugt:

- In pool_transactions_genesis werden die initialen Knotenbetreiber mit IP, Port und öffentlichem Schlüssel beschrieben
- In do-main_transaction_genesis sind die initialen Trustees und Stewards beschrieben.

Die Genesisfiles und deren Hashes werden über ein sicheres Transportmedium (über Projekt-Confluence mit abgesicherter HTTPS-Verbindung und Nutzerauthentifizierung über Benutzername/Passwort) an alle Knotenbetreiber verteilt und auf Richtigkeit geprüft (eigene Angaben und Schlüssel werden überprüft, zusätzliche Prüfung mittels SHA256 Checksumme).

Anschließend dienen die Genesisfiles als Konfiguration zum Starten des Netzwerks. Die Knoten verbinden sich und stellen einen Konsens her, somit ist das Netzwerk bereit für die erste Transaktion. Die Genesis-Files sind nicht signiert, die Integrität ist über die Veröffentlichung der Hashes im Projekt-Confluence sichergestellt. Perspektivisch können Genesis-Files bzw. deren Hashes in öffentlichen Zeitungen, Bundesanzeigern oder in Online-Verzeichnissen von Gremien hinterlegt werden.

7.3 Onboarding einer öffentlichen DID

Für eine initiale Transaktion erstellt ein Autor (Issuer, Verifier oder später hinzugefügte Knotenbetreiber) ein Schlüsselpaar (Ed25519) und die zum öffentlichen Schlüssel zugehörige DID.

Die DID und der öffentliche Schlüssel werden in einer initialen Transaktion verpackt und im Netzwerk gespeichert.

Zu ergänzen wie Speicherung abgesichert wird.

Alle weiteren Transaktionen muss der Autor mit seinem privaten Schlüssel digital signieren, sodass die Authentizität der Daten gewährleistet und einfach überprüfbar ist. Die so entstehenden Transaktionen schickt der Autor an mehrere Knoten. Sobald er mehr als $F+1$ (F = maximale Anzahl fehlerhafter Knoten) erfolgreiche Antworten erhält, ist die Finalität bzw. Konsistenz gegeben.

Im Netzwerk werden dabei der Konsens über die Daten mittels einer Boneh–Lynn–Shacham (BLS)-Multisignatur abgespeichert, welche die zweifelsfreie Konsistenz der validator nodes beweist, sodass bei einer Leseanfrage nur ein einziger Knoten angefragt werden muss. Die gespeicherten Transaktionen werden als geordnete Liste mit einem Merkle-Tree gespeichert. Der korrekte Zustand des Netzwerks kann anhand des aktuellen state proof und des Auditierpfads des Merkle-Trees überprüft werden. Die Authentizität der Validator Nodes kann wiederum anhand ihrer öffentlichen Schlüssel aus dem genesis file sichergestellt werden. Die genesis files sind der oberste Vertrauensanker und werden, wie in Abschnitt 7.2 beschrieben, sicher zwischen allen Beteiligten ausgetauscht.

7.4 Offboarding einer öffentlichen DID

7.5 Genesis-Files

7.6 Erstellen von Schemata und Credential Definitions

Issuer müssen klar definieren, welche Attribute die von ihnen ausgestellten Verifiable Credentials enthalten. Hierzu wird zunächst ein grundlegendes Schema („Template“) im Netzwerk von einem Steward erstellt. Dieses enthält die Definition, welche Attribute die jeweiligen Verifiable Credentials beinhalten müssen, um dem Schema zu entsprechen. Das Schema für eine Basis-ID beinhaltet beispielsweise eine Teilmenge der Datenfelder eines deutschen Personalausweises (Stadt, Familienname, Geburtsort, Geburtsname, Vorname, Geburtsdatum, Straße, Land, Ablaufdatum, akademischer Titel, PLZ). Der Name des Schemas, die beinhalteten Attribute sowie die DID des Issuers werden im Netzwerk gespeichert.

Auf Basis der Schema-Definition wird von allen beteiligten Issuern, die basierend auf einem Schema Verifiable Credentials erzeugen möchten, eine eigene sogenannte “Credential Definition” erstellt. In dieser ist die DID (und damit auch der Public Key) des Credential-Definition-Erstellers aufgeführt, zusätzlich zu dem Schema, auf das sich die Credential Definition bezieht (referenziert durch die Blocknummer, in der das Schema definiert wurde). Zudem wird für jedes Attribut eine Zufallszahl (2048 Bit) erzeugt, die beim späteren Nachweisen der Attribute durch die Nutzerinnen und den Verifizierenden als Referenz benötigt wird (ein sogenannter Common Reference String, welcher für die eingesetzten Zero-Knowledge Proofs benötigt wird).

8 Anforderungen an Issuer

Wie bereits beschrieben, ist die Governance-Organisation verantwortlich für das SSI-Ökosystem. Dies schließt auch Anforderungen hinsichtlich Informationssicherheit und Datenschutz mit ein.

Die Definition der Anforderungen an Issuer ist abhängig vom umzusetzenden Sicherheitsniveau das für einen Nachweis bzw. einen Anwendungsfall definiert wurde. Zwingend erforderlich für eine Teilnahme am Ökosystem ist die Umsetzung eines Informationssicherheitsmanagementsystems. Weitere erforderliche Maßnahmen ergeben sich aus einer durchzuführenden Risikoanalyse.

Bei Nichterfüllen der Anforderungen können Issuer aus dem SSI-Ökosystem ausgeschlossen werden (durch Löschen der öffentlichen Schlüssel für die Verifizierung von Credentials aus dem Netzwerk).

9 Anforderungen an Verifier

Auch hier legt die Governance-Organisation Anforderungen hinsichtlich Informationssicherheit und Datenschutz fest. Es sind nur Verifier als Beteiligte des SSI-Ökosystems zuzulassen, die diese Anforderungen umsetzen.

Bei Nichterfüllen der Anforderungen können Verifier aus dem SSI-Ökosystem ausgeschlossen werden (durch Löschen der öffentlichen Schlüssel aus dem Netzwerk).

10 Anforderungen an Knotenbetreiber

11 Kryptoagilität

11.1 Einleitung

Das BSI schreibt vor, dass bei der Entwicklung von IT-Systemen das Thema Kryptoagilität beachtet werden muss, siehe []. Dies bedeutet, dass die kryptographische Funktionalität so umzusetzen ist, dass bei Bekanntwerden von Schwächen der eingesetzten kryptographischen Verfahren ohne größeren Aufwand auf Alternativen gewechselt werden kann. Aktuell ist dies im bestehenden System nicht vollständig umgesetzt, insbesondere auch, was zukünftige Bedrohungen kryptographischer Verfahren durch Quantencomputer betrifft.

Ziel dieses Kapitels ist, zu beschreiben, welche kryptographischen Verfahren betroffen sind und die Maßnahmen darzustellen, die zu einem agilen Einsatz kryptographischer Verfahren im System führen.

11.2 Klassifizierung der eingesetzten kryptographischen Verfahren

Das SSI-Ökosystem verarbeitet personenbezogene Daten der Identitätsinhaberinnen. Kryptographisch werden zwei Schutzziele (Authentizität und Vertraulichkeit) umgesetzt:

- Authentizität der Identitätsattribute der Identitätsinhaber*innen
- Authentizität der Knoten des Netzwerks für den Konsenz
- Vertraulichkeit der Identitätsattribute der Identitätsinhaberinnen (Absicherung der Kommunikationsverbindungen)

11.3 Kurzfristiges Vorgehen Kryptoagilität

Kurzfristig (innerhalb eines Jahres) sollen für alle im System verwendeten kryptographischen Verfahren alternative Verfahren implementiert werden, sodass bei Bekanntwerden von Schwächen ohne großen Aufwand auf Alternativen gewechselt werden kann. Dies deckt noch nicht Bedrohungen durch Quantencomputer ab, siehe hierzu Abschnitt 11.4.

Das Vorgehen wird in den folgenden drei Unterabschnitten beschrieben.

11.3.1 Authentizität der Identitätsattribute

Derzeit wird das Protokoll AnonCred in Version 1 eingesetzt. Die Signatur der Identitätsattribute basiert hier auf sogenannten CL-Signaturen, das Zurückrufen auf sogenannten Type-3-pairing. Kurzfristig soll auch die Version 2 des AnonCred-Protokolls implementiert werden. Hier basiert die Signatur der Identitätsattribute auf BBS-Signaturen, das Zurückrufen ist allerdings wie in Version 1 umgesetzt. Hier wird eine Alternative erarbeitet und implementiert.

Wie in Abschnitt **??** beschrieben, werden zur Umsetzung der Gerätebindung für die BasisID die hardware-basierten Sicherheitsfunktionen der Smartphones genutzt, die von den Herstellern zur Verfügung gestellt werden. Insbesondere können für das hier verwendete Challenge-Response-Protokoll also nur Signaturalgorithmen verwendet werden, die in den TEEs enthalten sind.

Beide im System genutzten Smartphone-Plattformen bieten Signaturalgorithmen an, die sowohl auf ECC als auch auf RSA basieren, so dass hier das Thema Kryptoagilität als umgesetzt betrachtet werden kann.

11.3.2 Authentizität der Knoten des Netzwerks

11.3.3 Vertraulichkeit der Identitätsattribute

Im System werden für die Absicherung der Kommunikationswege zur Umsetzung des Schutzziels Vertraulichkeit etablierte Protokolle (TLS, ZeroMQ) genutzt. Beide Protokolle ermöglichen die Nutzung verschiedener kryptographischer Primitiven für die Schlüsseleinigung, die Datenverschlüsselung und die Datenauthentisierung. Bei Bekanntwerden von Schwächen der hierfür im System verwendeten kryptographischen Verfahren kann also schnell auf alternative Verfahren gewechselt werden, so dass hier Kryptoagilität als erfüllt gesehen wird.

11.4 Mittelfristiges Vorgehen Kryptoagilität

Im System werden keine Langzeitgeheimnisse verarbeitet. Eine Implementierung von quantencomputerresistenten Verfahren muss daher nicht kurzfristig umgesetzt werden. Geplant ist, innerhalb eines Zeitraums von drei Jahren für alle eingesetzten kryptographischen Verfahren auch Alternativen zu implementieren, die resistent gegenüber Quantencomputern sind.

11.4.1 Authentizität der Identitätsattribute

Die derzeit eingesetzten kryptographischen Primitiven im AnonCred-Protokoll (Signaturen und Type-3-Pairing) sind nicht resistent gegen Quantencomputer. Geplant ist die Entwicklung von Alternativen, die quantencomputerresistente kryptographische Primitiven verwenden und der mathematische Nachweis derer Sicherheitseigenschaften. Hierfür ist ein Zeitraum von drei Jahren vorgesehen.

Für die Umsetzung der Gerätebindung müssen die hardware-basierten Sicherheitsfunktionen genutzt werden, die von den Smartphone-Herstellern zur Verfügung gestellt werden.

11.4.2 Authentizität der Knoten des Netzwerks

11.4.3 Vertraulichkeit der Identitätsattribute

Im System werden für die Absicherung der Kommunikationswege zur Umsetzung des Schutzziels Vertraulichkeit etablierte Protokolle (TLS, ZeroMQ) genutzt. Es kann daher davon ausgegangen werden, dass mittelfristig quantencomputerresistente Kryptoverfahren in diesen Protokollen aufgenommen werden und auch für das System zur Verfügung stehen.

Im Projekt wird die Weiterentwicklung der oben genannten Protokolle beobachtet, um frühzeitig hier aufgenommene PQC-Verfahren im System zu integrieren.

Begriffserklärung

Key Attestation: Eine gegebene Schlüsselbescheinigung (key attestation) erzeugt eine Kette von Zertifikaten bis hin zu einem von Google ausgestellten Stammzertifikat. Jedes Zertifikat in der Kette könnte aufgrund mehrerer Gründe, z. B. einem kompromittierten Schlüssel oder einem Softwarefehler, widerrufen sein. Daher muss der Sperrstatus für jedes Zertifikat in der Kette ermittelt werden, wenn ein bestimmter Schlüsselnachweis validiert wird (siehe [**keyattestation-doc**]). Google Inc. betreibt eine JSON-formatierte Liste gesperrter Zertifikate , verfügbar unter [**keyattestation-status**], einschließlich der Seriennummern und der Gründe für den Widerruf. Die Liste enthält nur gesperrte Zertifikate. Falls die Liste demnach eine bestimmte Seriennummer nicht enthält, kann das entsprechende Zertifikat als gültig angesehen

werden.

Indy Netzwerk: Das Indy Netzwerk besteht aus den Indy Knoten welche durch die beteiligten Knotenbetreiber implementiert werden und setzt damit unter Nutzung eines Distributed Ledgers ein Verifiable Data Registry um. Ein Verifiable Data Registry ist ein System, welches die Erstellung, Verifizierung, Aktualisierung oder Revozierung von digitalen Identitäten und DID-Dokumenten ermöglicht. Der Ledger speichert die Credential Schema Definitions, Public Keys, Public DIDs und evtl. Revozierungsdaten.

Indy Knoten

Distributed Ledger

Verifiable Data Regestry

Verifiable Credential

DID

DID-Dokument

HardwareDID

Credential Schema Definition

Public DID

Revozierungsdaten

Presentation request

Proof request

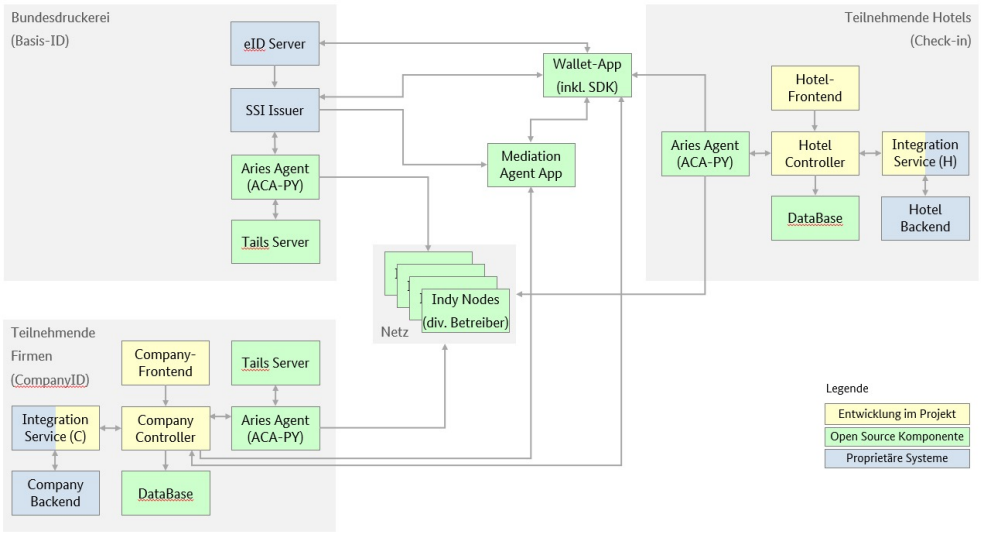

Abbildung 2: Systemübersicht

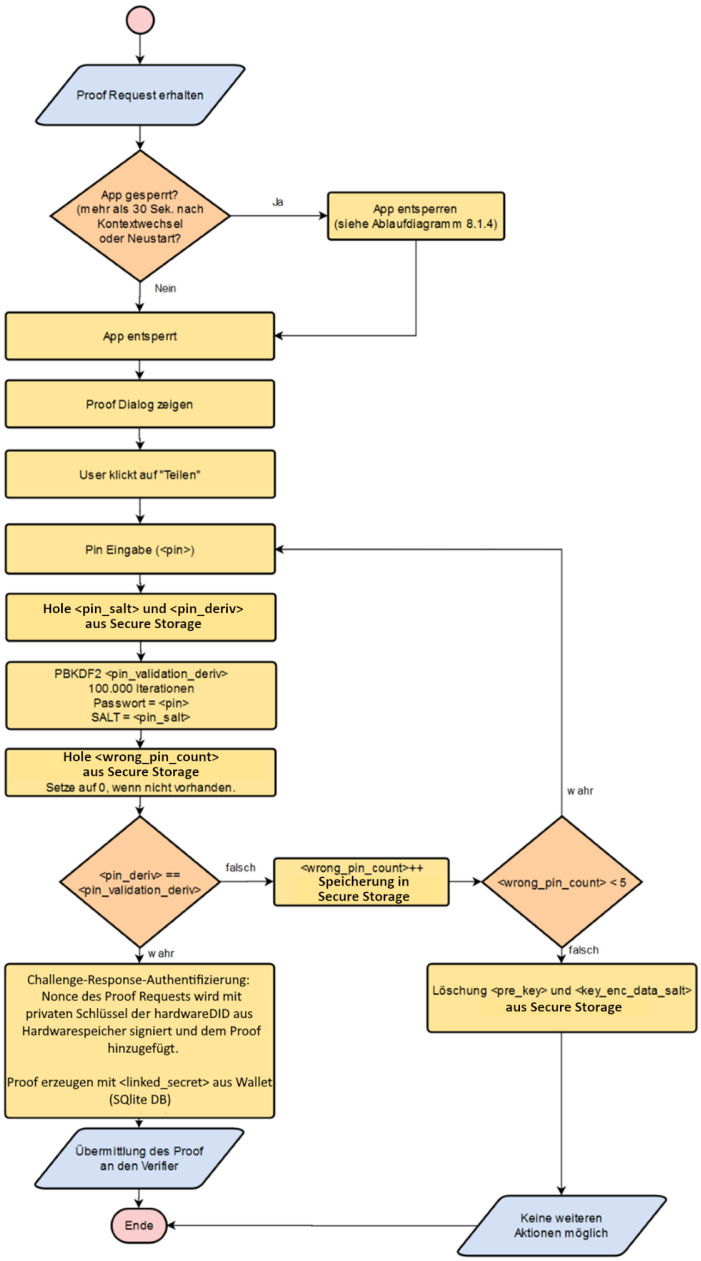

Abbildung 3: Nutzer-Authentisierung