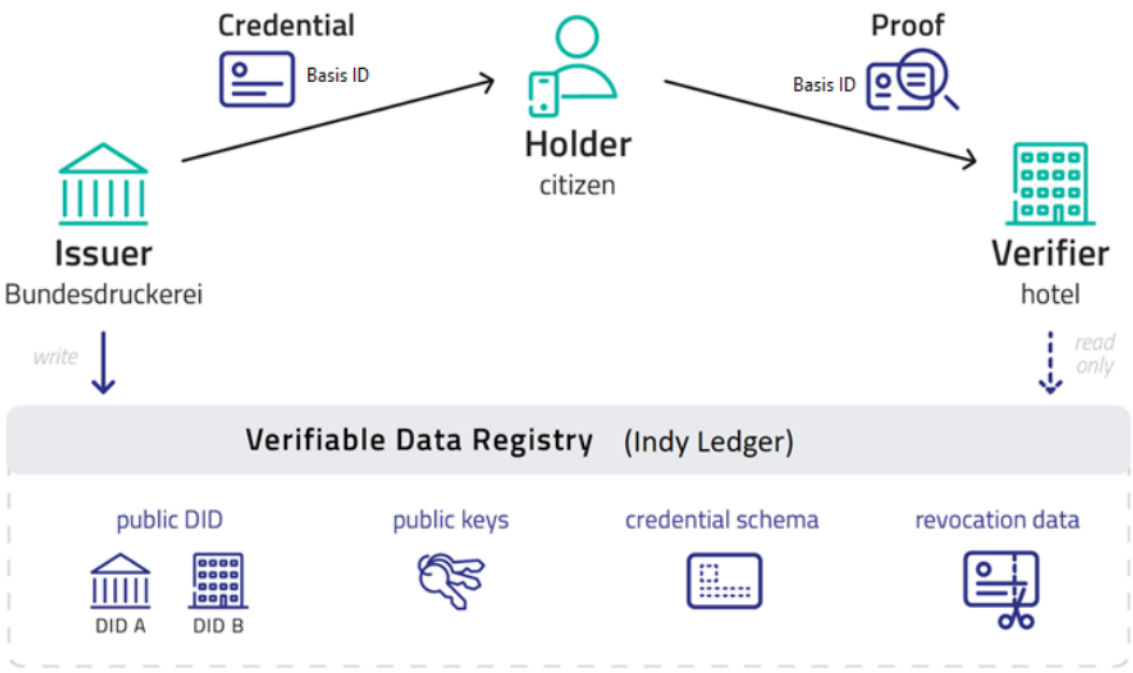
Dieses Dokument wurde in Zusammenarbeit von IBM Deutschland GmbH, esatus AG, Bundesdruckerei GmbH im Rahmen des Projektes "Ökosystem Digitaler Identitäten" erstellt.

Technische Beschreibung der Gerätebindung

1 Zweck dieses Dokuments (Ziele)

Dieses Dokument ist eine Ergänzung zum Dokument *Systemkonzept zu dem Projekt 2080069021* „DIGITALE IDENTITÄTEN:PILOTVORHABEN HOTEL CHECK-IN“. Es beschreibt die geplante Umsetzung der Bindung einer Basis-ID an ein mobiles Endgerät durch die Verwendung einer hardwareunterstützten sicheren Ablaufumgebung.

2 Systemübersicht



Grafik 1: Übersicht SSI-Ökosystem

Der **Issuer** (Bundesdruckerei) stellt ein signiertes, gerätegebundenes Credential (Basis-ID) auf Grundlage der eID Attribute des Personalausweises aus.

Der **Holder** (Nutzer) erhält nach Durchführung der Online-Ausweisfunktion die Basis-ID in seiner ID-Wallet App. Dies ermöglicht eine spätere Nutzung der Basis-ID mit dem beim Ausstellungsprozess verwendeten mobilen Endgerät.

Der **Verifier** (z.B. Hotel) prüft die vom Nutzer vorgezeigten Credentials (Basis-ID und z.B. Arbeitgeberbescheinigung) während eines Authentifizierungsvorgangs. Dabei kann er die Authentizität der Daten mit Hilfe der Prüfinfrastruktur und die Gerätebindung mit Hilfe der ID-Wallet App verifizieren.

Das **Verifiable Data Registry** ist ein öffentliches, genehmigungspflichtiges Blockchain-Netzwerk (Hyperledger Indy) zur Speicherung von öffentlichem Schlüsselmaterial und dezentrale Prüfinfrastruktur, betrieben von authentisierten und autorisierten Knotenbetreibern.

3 Anforderungen

Die Anforderungen aus den Auflagen für den Hotel-CheckIn fordern unter anderem: “Absicherung der für die Authentisierung verwendeten kryptographischen Geheimnisse über hardwareunterstützte Überschlüsselung”.

Mit der hier beschrieben technischen Lösung soll eine Gerätebindung der Basis-ID erreicht werden, die genauer folgende Eigenschaften erfüllt:

- Kopieren auf anderes mobiles Endgerät wird bestmöglich unterbunden
- Mehrfaktor-Authentisierung
    - Nutzung nur mit einem Smartphone möglich (Besitz)
    - und nach Authentisierung mit Inhaber-PIN

3.1 Lösungsansätze

Mit dem Link Secret besteht im derzeit genutzten Anonymous-Credentials-Protokoll bereits ein geheimes Attribut, welches die Zugehörigkeit aller Credentials an eine bestimmte Wallet bindet. Die hierbei verwendeten kryptografischen Protokolle sind allerdings nicht mit den gängigen hardware-gestützten Sicherheitsfunktionen der Mobilgeräte kompatibel.

Durch die Hinzunahme eines zusätzlichen Basis-ID Attributes, welches den öffentlichen Schlüssel einens hardwaregebundenen asymmetrischen Schlüsselpaarsenthält, kann eine Gerätebindung bei der Verwendung eines Anonymous Credentials gewährleistet werden.

4 Lösungsbeschreibung

4.1 Überblick

Nachfolgend die Erläuterungen zur Nutzung eines hardwaregebundenen Schlüsselpaars zur Instanzauthentisierung.

Der öffentliche Schlüssel ist ein signiertes Attribut der Basis-ID (genauso wie die anderen personenbezogenen Attribute). Dies geht einher mit nachfolgendem Prozess:

- Der *Aussteller* prüft die Güte und Herkunft des generierten Schlüsselpaars (Attestierung).
- Der *Verifier* authentifiziert den Nutzer mittels Challenge-Response Verfahren gegen das hardwaregebundene Schlüsselpaar.
- Der *Nutzer* schaltet im Zuge des Challenge-Response Verfahren mit der Inhaber-PIN die einmalige Nutzung des privaten Schlüssels durch die Wallet App zur Erstellung einer Signatur innerhalb des sicheren Schlüsselspeichers frei. Der private Schlüssel verlässt dabei niemals den sicheren Schlüsselspeicher. Alle kryptographischen Operationen finden im sicheren Schlüsselspeicher statt. Das Protokoll der Anonymous Credentials wird um eine Komponente zur Gerätebindung (Challenge-Response Protokoll mit hardware-gebundenem asymmetrischen Schlüsselpaar) erweitert. Durch die technische Umsetzung und Konfiguration der Software von Aussteller und Verifizierer wird sichergestellt, dass dieses Feature für den Hotel-Checkin Use Case zwingend durchgeführt wird.

Die Anwendung dieser Regeln auf Seiten des Ausstellers und Verifizierers wird durch regulatorischen Anforderungen eingefordert. Ein Angriff auf den Verifizierer wird durch organisatorische und technische

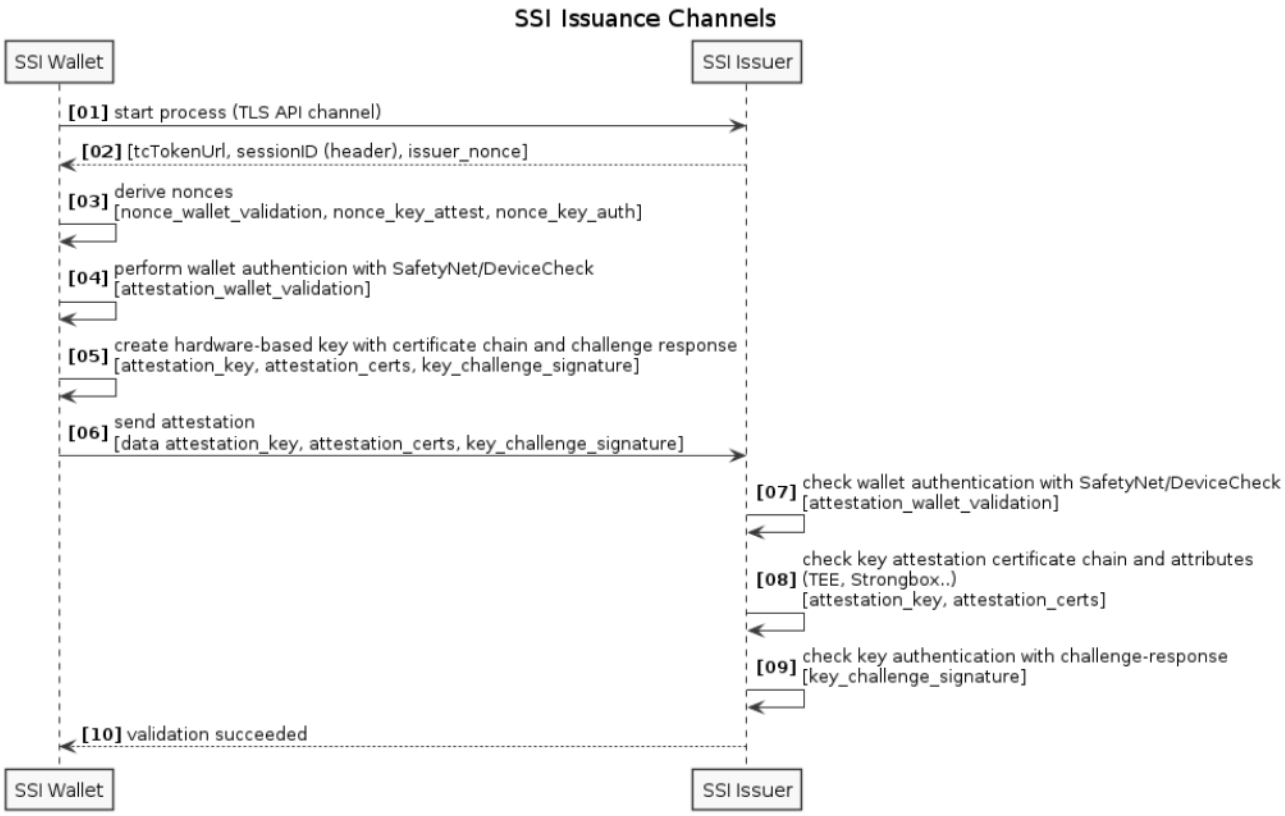
Maßnahmen nach Stand der Technik bestmöglich verhindert,eine Erschwerung des Angriffs kann beispielsweise durch einen geprüfte (beispielsweise signierte) Software ermöglicht werden.

4.2 Ausstellung der Basis-ID

Dieser Abschnitt des Gerätebindung-Konzepts bezieht sich auf Kapitel 4.1 Enrolment des Systemkonzept zu dem Projekt 2080069021.

Die Ausstellung der Basis-ID wird dahingehend erweitert, so dass das hardwaregebundene, asymmetrische Schlüsselpaar vom Issuer geprüft und der öffentliche Schlüssel anschließend als Teil der Basis-ID signiert wird. Genauer, werden folgende Schritte durchgeführt:

1. Wallet startet mit Basis-ID Ausstellungsanfrage (TLS, API key geschützt)
2. Aussteller antwortet zusätzlich mit einer zufällig generierten Nonce
   issuer_nonce = (unix timestamp | UUID_v4 mit 2^122 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 einen neuen hardware-gebundenes asymmetrisches Schlüsselpaar
   a. Die Wallet erstellt (unter Android, siehe Kapitel 4.4) eine Attestierung des Schlüsselpaars **attestation_key** unter Einbeziehung der **nonce_key_attest**
   b. Die Wallet bezieht (unter Android, siehe Kapitel 4.4) die Zertifikatskette der Schlüsselattestierung **attestation_certs**
   c. Die Wallet weist unter Einbeziehung der **nonce_key_auth** den Besitz des privaten Schlüssels nach und generiert die Signatur **key_challenge_signature =** SHA256withECDSA (**nonce_key_auth**)
6. Wallet übermittelt **attestation_key, attestation_certs** und **key_challenge_signature**
7. Aussteller überprüft die Integrität der Wallet mittels SafetyNet/DeviceCheck an Hand der **attestation_wallet_validation**
8. Aussteller 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 (siehe auch Kapitel 4.4)
   b. Attributeigenschaften der Schlüsselattestierung evaluieren
9. Aussteller überprüft die Instanzauthentisierung des Challenge-Response mit Hilfe der Signatur **key_challenge_signature**
10. Aussteller antwortet der Wallet bei Erfolg, Ausstellung kann fortgesetzt werden mit Attribut hardwareDid aus **attestation_key**



Grafik 2: Prozessfluss Ausstellung

4.3 Nutzung der Basis-ID

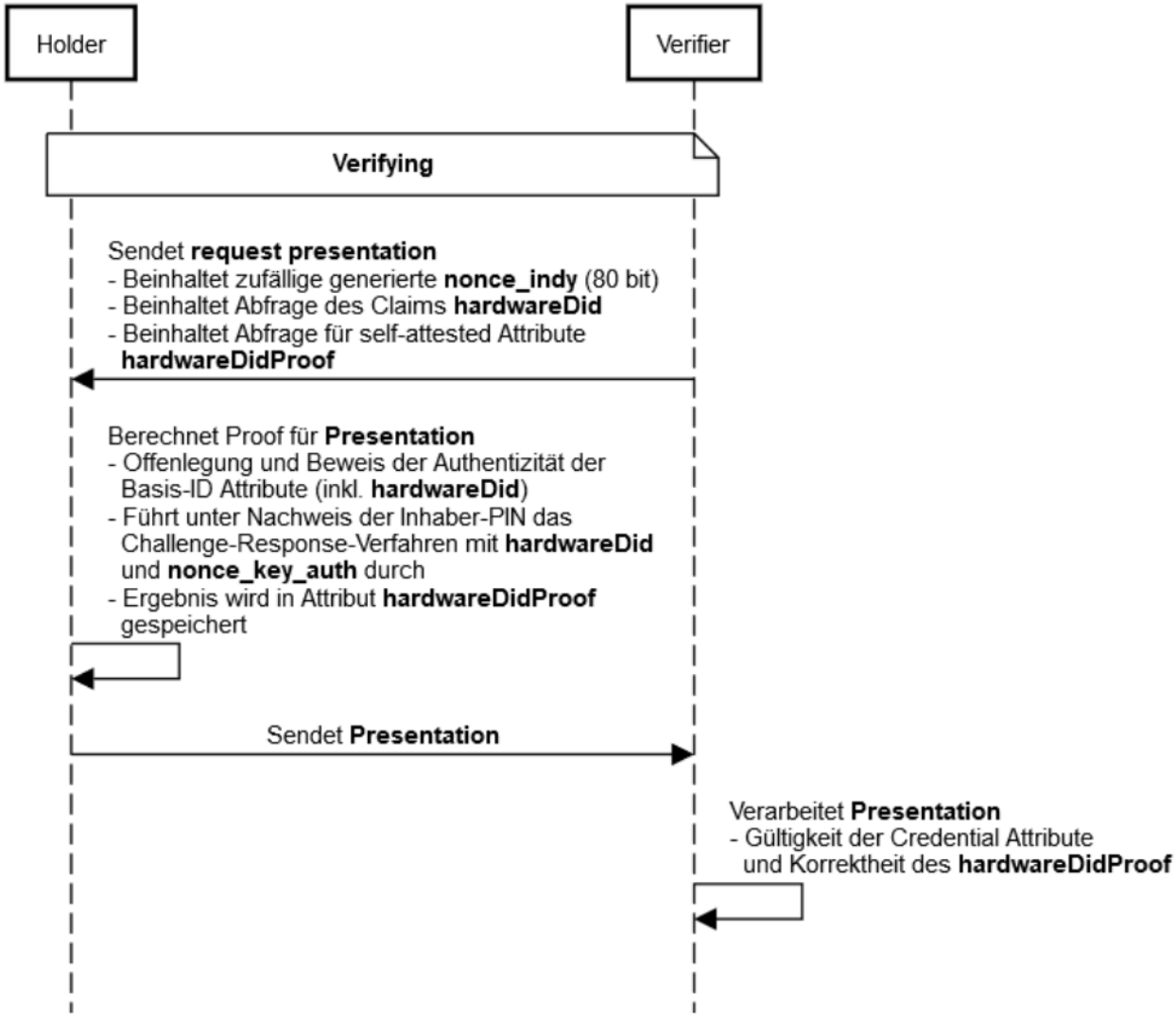
Dieser Abschnitt des Gerätebindung-Konzepts bezieht sich auf Kapitel 4.2 Nutzung des Systemkonzept zu dem Projekt 2080069021.

Bei der Nutzung der Basis-ID wird wie bisher das Aries RFC 0037: Present Proof Protocol zwischen Verifier und Holder angewendet. Zusätzlich werden innerhalb dieses Protokollverlaufes die Informationen und das Challenge-Response Verfahren für den gerätegebundenen Schlüssel eingebunden. Dieser Prozess setzt die Eingabe der Inhaber-PIN voraus und ist folgender Maßen umgesetzt:

1. Verifier sendet **request presentation.** Dieser enthält:
    a. Zufällig generierte Nonce **nonce_indy** (80Bit)
    b. Abfrage zu allen benötigten Attributen der Basis ID inkl. **hardwareDid.**
    c. Zusätzlich Abfrage zu self-attested attribute **hardwareDidProof** (dieses ist nicht Teil des Schemas/der Credential-Definition und kann vom Holder selbst befüllt werden)
2. Wallet berechnet Proof für Presentation
    a. Offenlegung und Beweis der Authentizität der Basis-ID Attribute (inkl **hardwareDid**) nach Anonymous-Credential-Protokoll
    b. führt unter Nachweis der Inhaber-PIN das Challenge-Response-Verfahren mit Verifier mittels **hardwareDid** und **nonce_key_auth** durch, das Ergebnis wird im Attribut **hardwareDidProof** gespeichert,

i. **nonce_key_auth** = SHA256(nonce_indy | 0x02)
ii. **hardwareDidProof** = SHA256withECDSA (**nonce_key_auth**)

3. ID Wallet sendet Presentation
4. Verifier prüft Gültigkeit der Credential Attribute (a) und Korrektheit des **hardwareDidProof**



Grafik 3: Prozessfluss Überprüfung

4.4 Revozierung der Schlüsselattestierungszertifikate (Key Attestation Zertifikate)

Dieser Abschnitt des Gerätebindung-Konzepts bezieht sich auf Kapitel 4.4 Inhaber-Authentisierung des Systemkonzept zu dem Projekt 2080069021.

Eine gegebene Schlüsselbescheinigung 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 [1]).

Google Inc. betreibt eine JSON-formatierte Liste gesperrter Zertifikate , verfügbar unter [2], 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.

In der Vergangenheit wurden standardisierte Zertifikatsperrlisten (CRLs) verwendet. Die Zertifikate selbst enthalten eine Erweiterung, welche einen Link zu entsprechenden CRLs enthält, sodass sie von Entwicklern problemlos abgerufen werden können. Diese Methode ist weiterhin verfügbar, wird aber in Zukunft eingestellt [3].

Das offizielle Beispielprojekt (Java), welches Entwicklern ermöglicht eine Schlüsselbescheinigung zu validieren, enthält bereits geeignete Methoden, um die neuere JSON-formatierte Liste abzurufen [5] und bequem den Sperrstatus der Seriennummer eines bestimmten Zertifikats zu überprüfen [4]. Die Liste enthält auch ältere Zertifikate (falls sie widerrufen wurden), sodass Entwickler problemlos zur neuen Liste wechseln können [6].

[1] https://developer.android.com/training/articles/security-key-attestation#certificate_status

[2] https://android.googleapis.com/attestation/status

[3] https://android-developers.googleblog.com/2019/09/trust-but-verify-attestation-with.html

[4] https://github.com/google/android-key-attestation/blob/0d0a4cd3f759741080d210c3a9f0c9962a130342/server/src/main/java/com/android/example/KeyAttestationExample.java#L223

[5] https://github.com/google/android-key-attestation/blob/0d0a4cd3f759741080d210c3a9f0c9962a130342/server/src/main/java/com/google/android/attestation/CertificateRevocationStatus.java#L35

[6] https://developer.android.com/training/articles/security-key-attestation#crl_legacy

4.5 Nutzerbindung (Inhaber-PIN)

Dieser Abschnitt des Gerätebindung-Konzepts bezieht sich auf Kapitel 7.1.2.2 Inhaber-Authentisierung des Systemkonzept zu dem Projekt 2080069021.Der Wert der Inhaber-PIN(zB 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 4:

Um dies zu realisieren, wird für die Inhaber-PIN bei initialer Einrichtung der ID Wallet auf Basis des Wertes der Wallet-PIN und mit eine zufällig erzeugten Salt per PBKDF2 ein Derivat abgeleitet, beides (Derivat und Salt) wird im Xamarin Secure Storage gespeichert:

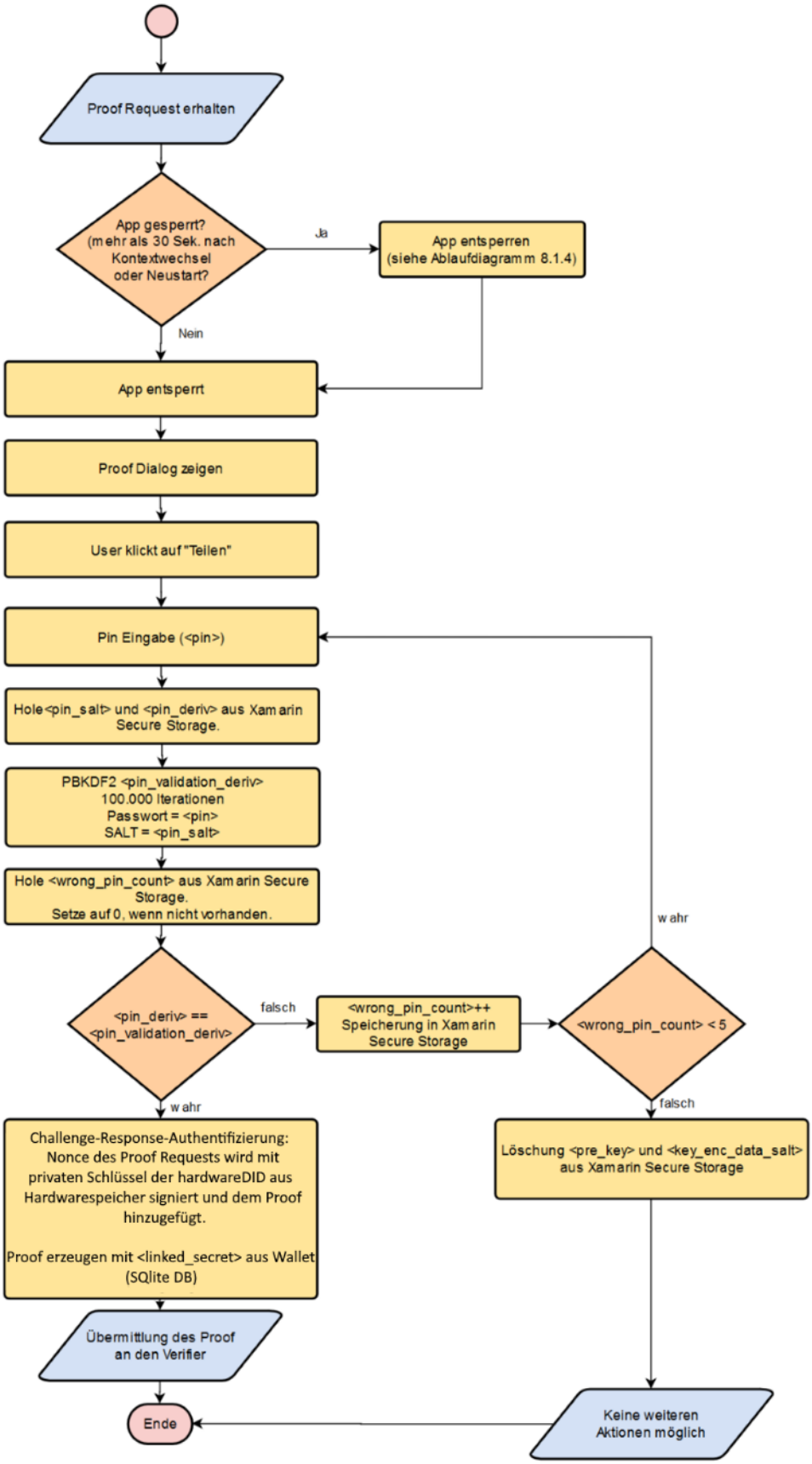
1. Pin_deriv = PBKDF2(PIN | Pin_salt)
2. Speichere Pin_deriv und Pin_salt in Xamarin Secure Storage

Dabei ist das Pin_salt ≠ Key_enc_data_salt

Im Rahmen der Inhaber-Authentisierung wird dann die von der Nutzer*in eingegebene Inhaber-PIN erneut abgeleitet und mit dem in der Xamarin 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 Xamarin Secure Storage.


Proof Request erhalten
App gesperrt?
(mehr als 30 Sek. nach
Kontextwechsel
oder Neustart?
Ja
App entsperren
(siehe Ablaufdiagramm 8.1.4)
Nein
App entsperrt
Proof Dialog zeigen
User klickt auf "Teilen"
Pin Eingabe (<pin>)
Hole<pin_salt> und <pin_deriv> aus Xamarin
Secure Storage.
PBKDF2 <pin_validation_deriv>
100.000 Iterationen
Passwort = <pin>
SALT = <pin_salt>
Hole <wrong_pin_count> aus Xamarin Secure
Storage.
Setze auf 0, wenn nicht vorhanden.
<pin_deriv> ==
<pin_validation_deriv>
falsch
<wrong_pin_count>++
Speicherung in Xamarin
Secure Storage
<wrong_pin_count> < 5
wahr
wahr
falsch
Challenge-Response-Authentifizierung:
Nonce des Proof Requests wird mit
privaten Schlüssel der hardwareDID aus
Hardwarespeicher signiert und dem Proof
hinzugefügt.
Proof erzeugen mit <linked_secret> aus Wallet
(SQlite DB)
Löschung <pre_key> und <key_enc_data_salt>
aus Xamarin Secure Storage
Übermittlung des Proof
an den Verifier
Keine weiteren
Aktionen möglich
Ende

Grafik 4: Ablauf Wallet-Logik Nutzerbindung

4.6 Technische Limitierungen

Unter iOS ist keine direkte key attestation des Geräteschlüssels möglich. Die Glaubwürdigkeit des generierten Schlüssels wird durch den über den selben Kanal durchgeführten iOS DeviceCheck untermauert. Wenn der AppleAttest Service nicht erreichbar ist, bricht der Ausstellungsvorgang ab.

Bei Nutzung der System-PIN kann keine Aussage über die Art und Güte des Authentisierungsobjektes getroffen werden (Pattern, PIN-Länge..), abhängig von DevicePolicy (keinen Einfluss). Deshalb verwenden wir eine App-verwaltete Inhaber-PIN bis die mobilen Plattformen eine geeignete schlüsselspezifische PIN bereitstellen. Dabei wird der Fehlbedienungszähler, PIN-Salt und PIN-Derivat(PBKDF2) im Xamarin Secure Storage gespeichert. Xamarin Secure Storage nutzt den nativen Schlüsselspeicher der jeweiligen mobilen Plattform (Speicherung in iOS KeyChain - AES-256 GCM (mittels ECC Schlüssel aus Secure Enclave verschlüsselt)).

5 Sicherheitsbetrachtung

Die in diesem Konzept beschriebene technische Lösung stärkt die im vorherigen Dokument genannten Sicherheitseigenschaften indem zusätzlich eine starke Gerätebindung erreicht wird.

5.1 Gerätebindung

Die Gerätebindung wird durch die Hinzunahme eines kryptographischen Geräteschlüsselpaars zur Basis-ID erreicht. Der private Schlüssel wird dabei in einer hardwareunterstützten sicheren Ablaufumgebung auf dem mobilen Endgerät des Nutzers generiert und nach Eingabe der Inhaber-PIN genutzt. Eine Extraktion des privaten Schlüssels wird dadurch erschwert (siehe dazu auch 5.2, Punkt 5, sowie [1] und [2]). Der Issuer bestätigt während der Ausstellung mit seiner Signatur über den öffentlichen Schlüssel, dass die Güte des Geräteschlüssel geprüft wurde. Der Verifier kann anhand dieses signierten Attributes und einem aktuellen Nachweis des Nutzers über den Besitz des privaten Schlüssels prüfen, dass der Nutzer das legitime mobile Endgerät verwendet.

5.2 Verknüpfung der Sicherungsfaktoren (Inhaber-PIN & privater Schlüssel der hardwareDid)

Im vorliegenden Ansatz wird eine Multi-Faktorauthentisierung, bestehend aus einem Haben-Faktor (privater Schlüssel der hardwareDid) und Wissen-Faktor (Inhaber-PIN), eingesetzt. Die Nutzung des Haben-Faktor wird von der Wallet verwaltet und nur nach erfolgreicher Validierung der Inhaber-PIN für einen Signaturvorgang freigegeben.

Folgende Maßnahmen werden getroffen, um diese Funktionsweise und damit die Verknüpfung der Sicherungsfaktoren sicherzustellen:

A. Das kryptographische Schlüsselmaterial (privater Schlüssel der hardwareDid) wird im sicheren Speicher (TEE, Strongbox [1] oder Secure Enclave [2]) der mobilen Endgeräte generiert, abgelegt und nur dort für kryptographische Operationen verwendet.

B. Der gehashte Referenzwert der Inhaber-PIN Pin_deriv wird sicher im Xamarin Secure Storage abgelegt (mit einem hardware-und app-gebundenen kryptographischen Schlüssel überschlüsselt).
C. Beim Issuing wird sichergestellt, dass sich das mobile Endgerät in einem vertrauenswürdigen Zustand befindet (Android: SafetyNet Attestation [3], iOS: DeviceCheck Service [4]), sowie die Wallet App integer und authentisch ist (Android: SafetyNet Attestation & Key Attestation [5], iOS: DeviceCheck Service).
D. Beim Issuing wird über die Key Attestation bzw. indirekt über den DeviceCheck Service sichergestellt, dass das kryptographische Schlüsselmaterial im sicheren Speicher des Smartphones generiert und gespeichert wurde.

Folgende Angriffe auf die Multi-Faktorauthentisierung werden damit verhindert bzw. erschwert:

Vor / beim Issuing:

1. Issuing einer Identität in eine manipulierte Wallet App: C stellt sicher das Identitäten nur in eine integere und authentische Wallet App geissued werden.
2. Verknüpfung der Identität mit nicht-hardware-gebundenen kryptographischen Schlüsselmaterial: Um diesen Angriff durchzuführen muss entweder das mobile Endgerät selbst oder die Wallet App manipuliert werden. Diese Angriffe können durch C und D erkannt werden und ein Issuing unterbunden werden.

Während der Nutzung / im Betrieb

1. Entfernen der Inhaber-PIN Logik mittels statischer Manipulation der Wallet-App: Wird die Wallet-App dekompiliert, der Quellcode angepasst und die angepasste Version auf dem selben oder einem anderen mobilen Endgerät installiert, ändert sich die Signatur bzw. der Fingerabdruck der Wallet-App und diesekann aufgrund der Nutztung von Anicht mehr auf das kryptographische Schlüsselmaterial im sicheren Speicher des Endgerätes zugreifen [6] . Dieser Mechanismus funktioniert nur, wenn sich das mobile Endgerät in einem vertrauenswürdigen Zustand befindet. Wird das Endgerät nach dem Aufspielen der Wallet-App sowie dem Aufbringen der Identität kompromittiert, kann dieser Schutz möglicherweise umgangen werden [10].
2. Entfernen der Inhaber-PIN Logik mittels dynamischer Instrumentation der Wallet-App: Um eine dynamische Instrumentierung der Wallet App durchzuführen, muss diese unter Android und iOS entweder statisch angepasst werden [7] (siehe a) oder das Betriebssystem selbst angepasst werden [8]. Unter Android setzt dies ein Unlocken des Bootloaders voraus [8] und führt zu einer systemweiten Löschung [9]. Unter iOS führt der sogenannten Jailbreak dazu, dass das kryptographische Schlüsselmaterial aufgrund von A nicht mehr nutzbar ist [2].
3. Extraktion des gespeicherten Inhaber-PIN: Die Inhaber-PIN liegt nur in gehashter Form (siehe B) und zusätzlich mit einem hardware-gebundenen kryptographischen Schlüssel (siehe A) überschlüsselt (Speicherung der gehashten Inhaber-PIN im Xamarin Secure Storage) vor. Trotz einer potentiell erfolgreichen Extraktion ist der Missbrauch daher deutlich erschwert :
    a. Unter Android: Um die gehashte und verschlüsselte Inhaber-PIN zu extrahieren, muss ein Angreifer unter Android den Sandboxing-Mechanismus kompromittieren, d.h. das Gerät rooten. Auch eine Extraktion der gehasten und verschlüsselten Inhaber-PIN über adb backup ist nicht möglich, da die Backupfunktionalität der Wallet per App Manifest deaktiviert wurde.

   b. Unter iOS: Um die gehashte und verschlüsselte Inhaber-PIN zu extrahieren, muss ein Angreifer unter iOS den Sandboxing-Mechanismus kompromittieren, d.h. das Gerät rooten. Wird in iOS ein Gerätebackup durchgeführt, wird auch die gehashte und verschlüsselte Inhaber-PIN mit eingeschlossen. Wird ein lokales Backup über iTunes durchgeführt und legt der Nutzer bei der Erstellung dieses Backups ein Passwort fest, ist das Backup verschlüsselt. Legt er kein Passwort fest ist es nicht verschlüsselt. Wird ein iCloud Backup des Gerätes durchgeführt, kann Apple als Betreiber der iCloud unter umständen Zugriff zum Backup und der gehashten und verschlüsselten PIN erhalten.
4. PIN-Phising: Ein PIN-Phising kann durch verschiedene Angriffe durchgeführt werden (Installation einer manipulierten Tastatur auf dem mobilen Endgerät, Overlay durch eine andere Applikation). Diese Angriffe setzen immer ein aktives Mitwirken des Nutzers voraus und gelten auch für weitere Lösungen, wie z.B. Smart eID. In der ID Wallet wird ein per Software generiertes und für die App eigenes Pin-Pad verwendet, es wird keine auf dem System installierte Tastatur eingesetzt.
5. Extraktion des kryptographischen Schlüsselmaterials: Das kryptographische Schlüsselmaterial liegt im sicheren Speicher des mobilen Endgerätes (siehe A) und bietet keine Funktion zur Extrahierung bzw. ist durch verschiedene Maßnahmen gegen eine Extraktion gehärtet. Dies gilt sowohl für Android [1] als auch für iOS [2].

[1] https://developer.android.com/training/articles/keystore

[2] https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf

[3] https://developer.android.com/training/safetynet/attestation

[4] https://developer.apple.com/documentation/devicecheck

[5] https://developer.android.com/training/articles/security-key-attestation

[6] https://source.android.com/security/keystore/features#key_access_control

[7] https://frida.re/docs/ios/#without-jailbreak

[8] https://topjohnwu.github.io/Magisk/install.html

[9] https://source.android.com/devices/bootloader/locking_unlocking

[10] https://dl.acm.org/doi/10.1145/3448609

5.3 Biometrische Authentisierungsverfahren

Die Nutzung der biometrischen Authentisierungsverfahren im Hotel Pilot wird bis zu einer Klärung der Anforderungen zurückgestellt. Der Vollständigkeit halber und für zukünftige Diskussionen bleibt das Kapitel vorerst Bestandteil des Konzepts.

5.3.1 Apple iOS

Apple gibt zur Güte der biometrischen Sensoren keine detaillierten Metriken bekannt. Folgende Informationen sind jedoch öffentlich abrufbar:

Touch ID:
Der Touch ID Sensor spricht in iPhones über einen authentischen und vertraulichen Kanal mit der Secure Enclave. Dabei werden vom Sensor die Rohdaten des erkannten Fingerabdrucks an die Secure Enclave geschickt, in der diese Abdrücke abgeleitet und mit einer gespeicherten, mathematischen Repräsentation verglichen werden.

Die Wahrscheinlichkeit das eine beliebige andere Person das iPhone eines Benutzers entsperren kann liegt bei 1:50.000. Beim mehreren registrierten Fingerabdrücken steigt diese Wahrscheinlichkeit auf bis zu 1:10.000 (5 Fingerabdrücken). Zusätzlich lässt Touch ID nur 5 Fehlversuche zu. Danach wird der System-PIN benötigt um das iPhone zu entsperren.

FaceID:

Der biometrische Gesichtsabgleich findet bei iPhones mit Face ID mittels eines neuronalen Netzwerkes statt. Die Face ID Daten und die mathematische Repräsentation des Gesichtes sind verschlüsselt und können nur von der Secure Enclave entschlüsselt werden. Die Konvertierung des aufgenommenen 2D-Bilds des Gesichts, sowie dazugehörige Tiefeninformationen werden von der Secure Neural Engine durchgeführt.

Die Wahrscheinlichkeit das eine beliebige andere Person das iPhone eines Benutzers entsperren kann liegt bei 1:1.000.000. Beim mehreren registrierten Erscheinungsbildern steigt diese Wahrscheinlichkeit auf bis zu 1:500.000 (2 Erscheinungsbilder). Zusätzlich lässt Face ID nur 5 Fehlversuche zu. Danach wird der System-PIN benötigt um das iPhone zu entsperren.

(Quelle: https://manuals.info.apple.com/MANUALS/1000/MA1902/de_DE/apple-platform-security-guide-d.pdf)

5.3.2 Android

Findet eine biometrische Nutzerauthentisierung im Rahmen der Multi-Faktorauthentisierung statt, werden auf der Android Plattform nur biometrische Sensoren der Klasse “Strong” zugelassen. Diese Klasse garantiert die Güte der Sensoren und weist folgende Metriken auf:

Biometric Tier	Metrics	Biometric Pipeline	Constraints
Class 3 - Strong	SAR: 0-7% FAR: 1/50k FRR: 10%	Secure	• 72 hours before fallback to primary authentication (such as PIN, pattern, or password) • Can expose an API to applications (eg: via integration with the BiometricPrompt or FIDO2 APIs • Must Submit BCR

Gerätehersteller werden dazu angehalten Lebenderkennung für alle biometrischen Sensoren zu implementieren. Zusätzlich sollte eine Aufmerksamkeitsdetektion für biometrische Gesichtssensoren umgesetzt werden. Die Aufnahme, Speicherung und Erkennung von biometrischen Sensoren sollte in

einer gesicherten, wenn möglich Hardware-gesicherten Umgebung erfolgen.Diese Anforderungen sind jedoch nicht verbindlich.

Zur Messung der Metriken für spezifische biometrische Sensoren gibt Google einen umfangreichen Testkatalog vor, der vom jeweiligen Smarthonehersteller für jedes Modell und jeden Sensortyp durchgeführt werden muss.(Quellen: https://source.android.com/security/biometric/measure, https://source.android.com/security/biometric#hal-implementation, https://source.android.com/compatibility/11/android-11-cdd.pdf)

6 Kryptographische Verfahren

Für die Umsetzung der Gerätebindung **hardwareDid / hardwareDidProof** werden folgende kryptographische Verfahren genutzt:

- NIST P256 / nist-p256r1
- SHA256withECDSA
- Schlüsselerzeugung unter Android:
    - **Algorithmus**: Elliptic Curve
    - **Signatur**: SHA256withECDSA
    - Inhaber-Pinabfrage bei Nutzung des generierten Schlüssels
    - Parameter für die Schlüsselerzeugung:

Digest	DigestSha256
AlgorithmParameterSpec	secp256r1
SetRandomizedEncryptionRequired	false
SetUserAuthenticationValidityDurationSeconds (da Ziel API Level < 30)	-1
SetIsStrongBoxBacked	True (wenn Strongbox genutzt werden kann)

- Schlüsselüberprüfung für Android:
    - Parameter für die Überprüfung der Zertifikatskette:

attestationVersion	>= 2
attestationSecurityLevel	TrustedEnvironment (1) oder StrongBox (2)
keymasterSecurityLevel	TrustedEnvironment (1) oder StrongBox (2)

- Schlüsselerzeugung unter iOS:
    - **Algorithmus**: Elliptic Curve
    - **Signatur**: EcdsaSignatureMessageX962Sha256
    - Inhaber-Pinabfrage bei Nutzung des generierten Schlüssels
    - Parameter für die Schlüsselerzeugung:

KeyType	ECSecPrimeRandom
SecAccessible	WhenUnlockedThisDeviceOnly
SecAccessControlCreateFlags	PrivateKeyUsage
KeySizeInBits	256

- Schlüsselüberprüfung für 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).